UNIVERSIDAD POLITÉCNICA DE MADRID

Escuela Técnica Superior de Ingenieros Informáticos

Cryptography for a Verifiable World: Foundations and Applications of Succinct Proof Systems

DOCTORAL THESIS

Submitted for the degree of Doctor by:

David Balbás Gutiérrez

M.Sc. in Computer Science

Madrid, 2025

UNIVERSIDAD POLITÉCNICA DE MADRID

Escuela Técnica Superior de Ingenieros Informáticos

Doctoral Degree in Software, Systems and Computing

Cryptography for a Verifiable World: Foundations and Applications of Succinct Proof Systems

DOCTORAL THESIS

Submitted for the degree of Doctor by:

David Balbás Gutiérrez

M.Sc. in Computer Science

Under the supervision of:

Prof. Dario Fiore

Madrid, 2025

Title: Cryptography for a Verifiable World: Foundations and Applications of Succinct Proof Systems

Author: David Balbás Gutiérrez

Doctoral Programme: Software, Systems and Computing

Thesis Supervision:

Dr. Dario Fiore, Associate Research Professor, IMDEA Software Institute

External Reviewers:

Dr. Antonio Faonio, Associate Professor, EURECOM
Dr. Giulio Malavolta, Assistant Professor, Bocconi University

Thesis Defence Committee:

Dr. Antonio Faonio, Associate Professor, EURECOM
Dr. Ignacio Cascudo Pueyo, Associate Professor, IMDEA Software Institute
Dr. Giulio Malavolta, Assistant Professor, Bocconi University
Dr. Ida Tucker, Senior Cryptography Researcher, PQ Shield
Dr. Eduardo Soria Vázquez, Lead Cryptographer, Technology Innovation Institute

Thesis Defence Date: 17 December 2025

This work is supported by the PICOCRYPT project that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (Grant agreement No. 101001283), by projects PRODIGY (TED2021-132464B-I00) and ESPADA (PID2022-142290OB-I00) funded by MCIN/AEI/10.13039/501100011033/ and the European Union NextGenerationEU / PRTR, by the Excellence Seal María de Maeztu with grant number CEX2024-001471-M/funded by MICIU/AEI/10.13039/501100011033, and by an FPU grant from Ministerio de Universidades (FPU21/00600).

Acknowledgements

There are countless things that will sometimes go wrong during a PhD: failing to find solutions to problems, lacking active collaborations, running short on ideas, unfair reviewers, facing overly competitive peers, feeling uncertain about the future, and many others. In retrospect, my time as a PhD student has been a fulfilling and rewarding experience on many levels, but it was definitely not a walk in the park. Overcoming the hard and frustrating moments was only possible thanks to the incredibly supportive environment around me.

Huge thanks to my supervisor Dario. I feel incredibly lucky to have worked with you during the past four years. I am grateful for your continuous guidance and enthusiasm, for suggesting great problems (and solutions), for the freedom to establish my own collaborations on diverse topics, for the opportunities and funding to travel, and for always finding the time and energy to discuss when I was stuck on problems.

Thanks to Maribel for your priceless scientific and non-scientific advice, for being always there when I needed a hand, and for introducing me to Dario and to so many people from the community. Thanks also to Claudio for our collaboration, support, and for showing me the value (and the challenge!) of motivating research projects by real-world problems.

Thanks to Ignacio, Ida, Giulio, Antonio, Eduardo, Miguel and Ángel for being part of my thesis committee. Special thanks to Giulio and Antonio for reviewing the thesis and for your positive comments.

During these years, I was super lucky to do long-term research visits in Finland and in Japan. Huge thanks to Russell for our fruitful collaboration, for dedicating much of your time to me, and for inviting me and making me feel welcome at Aalto. Your work capacity and the sharpness of your ideas is inspiring. Huge thanks to Abe-san for hosting me at NTT and for patiently sharing and explaining many ideas and problems. Thanks also to Ohkubo-san and to Octavio, Calvin, Dipayan, Dung and Sohyun for our discussions and collaborations, for raising my sushi and ramen standards, and for many nice memories that I brought back from Japan.

Thanks also to everyone who hosted me for short-term visits: Giulio and Phillip in Bochum, Serge and Daniel in Lausanne, Krzysztof and Guille in Vienna, Sherman in Hong Kong, and Kenny and Matilda in Zurich. I have been lucky to participate in discussions with you and your groups. Needless to say, I am extremely grateful for having collaborated with so many excellent researchers throughout my PhD. Special mention to my dear collaborators and friends Daniel and Phillip for turning so many work trips into fun adventures.

I always felt at home at IMDEA Software, mainly due to its friendly and supportive atmosphere. I am deeply grateful to everyone who contributed to it. From the cryptographers, thanks to Ignacio, Pedro, Ida, Peter, Lydia, Dimitris, Daniele, Mahak, Antonio, Douglas, Dimitrios, Emanuele, Javier, Pablo... Special mention to Claudia and Miguel for all our chats, gossips and laughs in our office, and to Diego, Gaspard, Damien, Gennaro and Hamza for our many trips and friendship beyond work. From the non-cryptographers, thanks to Silvia, Paloma, Juanma, Louis, Konstantinos, Nacho, Ale, Gibrán, Marga, Andoni, Georgia, Martín, Chana, Loïc, and so many others for flooding the institute with your positive energy. Thanks also to everyone at admin for always helping out and taking care of loads of paperwork for us.

Por último, gracias a todos mis geniales compañeros de piso por haberme acompañado (y aguantado) todos estos años: Dani, Ricardo, Antonio, Elena, nuevamente Gaspard, y especialmente María Luisa. Mis amigos madrileños, gatos y no gatos: Flavia, Matías, Carlos, el grupo de escaladores... Gracias por tantas buenas quedadas, excursiones, climbing weekends, y un millón de cosas más. Mi gente de Santander, los de siempre, gracias por traerme el viento del norte en decenas de visitas y por hacerme sentir en casa en cualquier lugar donde estéis. Laurane, gracias por tu apoyo incondicional en esta última etapa, por tu cariño, y por comprenderme a menudo mejor que yo mismo. Y mi familia Ana, Juan, y Victoria: infinitas gracias por estar siempre a mi lado y por apoyarme en todos mis pasos.

Santander, agosto de 2025

Abstract

Cryptographic proof systems enable an entity, the prover, to convince another entity, the verifier, about the validity of a statement. To be useful in practice, cryptographic proof systems must be *expressive*, enabling proofs for a large class of statements (such as those in the complexity class NP). A highly desirable property is *succinctness*, meaning that the size of the proof (and the time required to verify it) is much smaller than the size of the witness for the given statement. Nowadays, succinct cryptographic proof systems find a broad range of applications both within and beyond cryptography, such as outsourcing computation, anonymous credentials, signature aggregation, blockchain protocols, and many others. Generally, they provide a trust generation mechanism based on enabling the efficient verification of third-party claims.

In this thesis, we study both foundational and practical aspects of succinct proof systems. In the first part, we focus on building *succinct proof systems from standard assumptions*, a problem for which there exist impossibility results for so-called succinct non-interactive arguments (SNARGs). To circumvent these impossibilities, we seek to relax the power of SNARGs in different ways, focusing on three families of proof systems: functional commitments, batch arguments for NP, and homomorphic signatures. For *functional commitments*, we present the first construction of this primitive from falsifiable assumptions that supports the evaluation of arbitrary arithmetic circuits of unbounded depth. To this end, we introduce a novel concept called chainable functional commitments, which we instantiate both from lattice-based and from pairing-based assumptions. Later, we present a pairing-based construction that improves on our original result by achieving more compact public parameters. For *batch arguments*, we present the first algebraic construction that achieves circuit-succinctness, i.e., where the proof size scales linearly in the size of a single witness and is independent of the size of the circuit that decides the NP relation. For *homomorphic signatures*, we achieve the first homomorphic signature for unbounded-depth circuits, as well as the first multi-key homomorphic signature scheme that is fully-succinct, i.e., where the proof size scales sublinearly on the number of messages, the circuit size, and the number of signing parties.

In the second part of the thesis, we focus on building and implementing *efficient proof systems with practical applications*. The main goal is to bridge the gap between general-purpose and special-purpose proof systems: while the former are universally applicable, the latter provide better concrete efficiency. To this end, we present a family of sumcheck-based proof systems for verifiable computation of sequential computations which are modularly composable at the information-theoretic level, including a novel proof system for multi-channel convolutions. Our schemes yield both asymptotic and concrete performance improvements over the state-of-the-art techniques for verifiable machine learning and image processing.

Resumen

Los sistemas de pruebas criptográficas permiten que una entidad, llamada probador, convenza a otra entidad, llamada verificador, de la validez de una afirmación. Para ser útiles en la práctica, los sistemas de pruebas criptográficas deben ser expresivos, permitiendo pruebas para una amplia clase de instancias (como aquellas en la clase de complejidad NP). Una propiedad altamente deseable es la concisión, lo que significa que el tamaño de la prueba (y el tiempo requerido para verificarla) es mucho menor que el tamaño del testigo correspondiente a la instancia. Hoy en día, los sistemas de pruebas criptográficas concisas encuentran innumerables aplicaciones tanto dentro como fuera de la criptografía, incluyendo la computación delegada, las credenciales anónimas, la agregación de firmas, los protocolos de blockchain, y muchas otras. En términos generales, ofrecen una forma de generar *confianza*, ya que permiten verificar afirmaciones realizadas por terceros de forma eficiente.

En esta tesis, abordamos cuestiones fundamentales y prácticas en torno a los sistemas de pruebas concisas. En la primera parte, nos centramos en construir *sistemas de pruebas concisas a partir de supuestos criptográficos estándar*, un problema para el cual existen resultados de imposibilidad en el caso de los denominados "succinct non-interactive arguments" (SNARGs). Para sortear estas imposibilidades, buscamos relajar la definición de los SNARGs desde distintos ángulos, centrándonos en tres familias de sistemas de prueba: los esquemas de compromiso funcionales ("functional commitments"), los argumentos en tandas para NP ("batch arguments for NP") y las firmas homomórficas ("homomorphic signatures"). En cuanto a *functional commitments*, presentamos la primera construcción de esta primitiva a partir de supuestos criptográficos refutables que soporta la evaluación de circuitos arbitrarios de profundidad no acotada. Para ello, introducimos el concepto de esquema funcional de compromiso encadenable, del cual presentamos construcciones tanto basadas en retículos ("lattices") como en emparejamientos ("pairings"). Posteriormente, presentamos una construcción que mejora nuestro resultado original consiguiendo parámetros públicos más concisos. En cuanto a *batch arguments*, presentamos la primera construcción algebraica que logra ser concisa en el circuito, es decir, donde el tamaño de la prueba escala linealmente en el tamaño del testigo de la instancia correspondiente y no depende del tamaño del circuito que decide la relación NP. En cuanto a *homomorphic signatures*, en primer lugar logramos la primera construcción que soporta circuitos de profundidad no acotada. Además, presentamos el primer esquema multi-clave que es completamente conciso, es decir, en el que el tamaño de la prueba es sublineal en el número

de firmas, en el tamaño del circuito, y en el número de partes firmantes.

En la segunda parte de la tesis, nos enfocamos en construir e implementar *sistemas de pruebas eficientes con aplicaciones prácticas*. El principal objetivo es cerrar la brecha entre los sistemas de prueba de propósito general y los específicos: mientras que los primeros son aplicables universalmente, los segundos ofrecen mejor rendimiento para problemas determinados. Con este fin, presentamos una familia de sistemas de prueba "sumcheck" (instanciados a partir de sumas sobre polinomios multilineales) para la computación verificable de operaciones secuenciales. Estos sistemas son modulares y componibles a bajo nivel, e incluyen un nuevo sistema de prueba específico para convoluciones multicanal. Nuestras propuestas mejoran tanto la complejidad asintótica como el rendimiento concreto de las técnicas actuales para verificar algoritmos de inteligencia artificial y de procesado de imágenes.

Contents

List of Figures **xi**

List of Tables **xiii**

List of Publications **xv**

1 Introduction **1**

1.1 Cryptography as a Formal Science 1

1.2 Cryptographic Proof Systems from Theory to Practice 5

1.3 Thesis Contributions 14

2 Related Work **17**

2.1 Functional Commitments 17

2.2 Batch Arguments for NP 19

2.3 Homomorphic Signatures 19

2.4 Efficient Proof Systems 21

3 Background **25**

3.1 Notation 25

3.2 Bilinear Groups 25

3.3 Lattices 26

3.4 Commitment Schemes and Advanced Properties 27

3.5 Proof Systems 30

3.6 Digital Signatures 32

3.7 Somewhere Extractable Commitments 32

3.8 Multilinear Extensions 34

I Succinct Proof Systems from Standard Assumptions **37**

4 Chainable Functional Commitments for Circuits **39**

4.1 Contributions 39

4.2 Technical Overview 43

4.3 Functional Commitments 49

4.4 Chainable Functional Commitments 53
4.5 FC for Circuits from CFC for Quadratic Polynomials 54
4.6 Paring-based CFC for Quadratic Functions 61
4.7 Lattice-based CFC for Quadratic Functions 73
4.8 Generic Transformations for Functional Commitments 81

5 Pairing-based Functional Commitments with Shorter Parameters 85
5.1 Contributions 85
5.2 Technical Overview 86
5.3 Preliminaries 90
5.4 Pairing-based CFC for Quadratic Functions 93

6 Circuit-Succinct Algebraic Batch Arguments for NP 107
6.1 Contributions 107
6.2 Technical Overview 108
6.3 Preliminaries and Circuit Model 116
6.4 Projective Chainable Functional Commitments 119
6.5 Wire-by-Wire Compiler: From Functional Extractability to BARGs 125
6.6 Circuit-Succinct Compiler from PCFC to BARG 129
6.7 Algebraic PCFC from Bilinear Groups 135

7 Fully-Succinct Multi-Key Homomorphic Signatures 155
7.1 Contributions 155
7.2 Technical Overview 156
7.3 Multi-Key Homomorphic Signatures 160
7.4 Our MKHS Construction 165
7.5 Extensions and Instantiations 181

II Efficient Proof Systems and Applications 185

8 Modular Sumcheck Proofs and Applications to Machine Learning and Image Processing 187
8.1 Contributions 187
8.2 Composition Framework for Interactive Proofs 189
8.3 Verifiable Evaluation from Multilinear Polynomials 198
8.4 Verifiable Evaluation for Machine Learning 202
8.5 Evaluation 211

III Conclusion 215

9 Conclusions and Future Work 217

Bibliography 223

List of Figures

1.1 Positioning of this thesis in the landscape of research directions in cryptographic proof systems. 4

3.1 EUF-CMA security game for a signature scheme Σ. 33

4.1 Construction of our FC for circuits from a CFC for quadratic functions. . . 57

5.1 Representation of the different proof systems that conform our CFC. . . . 88
5.2 Power type chaining proof Π_{pow}. 95
5.3 Quadratic type chaining proof Π_{quad} (single-input). 98
5.4 Quadratic type chaining proof Π_{quad} (multi-input). 99
5.5 Equality type chaining proof Π_{eq}. 102
5.6 CFC construction from the type chaining proof systems Π_{pow}, Π_{quad} and Π_{eq}. 104
5.7 Games Hyb^0, $\mathsf{Hyb}^{1,i}$, Hyb^2 for the proof of evaluation binding of CFC. . . . 105

6.1 Overview of the sub-proofs and commitments involved in the PCFC functional proofs. 115
6.2 Basic Compiler from PCFC to BARG. 125
6.3 Games Hyb^0, $\mathsf{Hyb}^{1,j}$, $\mathsf{Hyb}^{2,j}$ for the proof of BARG somewhere extractability. 127
6.4 Compiler from PCFC to fully succinct BARG. 131
6.5 Games Hyb^0, $\mathsf{Hyb}^{1,j}$, Hyb^2 for the proof of BARG somewhere extractability. 133
6.6 First set of games for the proof of Theorem 6.52. 151
6.7 Second set of games for the proof of Theorem 6.52. 152

7.1 Simplified description of the BARG circuit C in our MKHS construction. . 159
7.2 Security experiment $\mathsf{HomUF\text{-}CMA}_{\mathcal{A},\mathsf{MKHS}}(1^\lambda)$. 163
7.3 Security experiments $\{\mathsf{CtxtHiding}^b\}_{b=0,1}$ for context-hiding. 165
7.4 Construction of a multi-key homomorphic signature scheme MKHS. . . . 167
7.5 Description of the BARG circuit C. 168
7.6 Efficient verification algorithms for our construction of a multi-key homomorphic signature scheme MKHS. 169
7.7 Games Hyb^0, Hyb^1, Hyb^2, Hyb^3 for the proof of Lemma 7.14. 171
7.8 Games $\mathsf{Hyb}^{1,0}$, $\mathsf{Hyb}^{1,1}$ for the proof of Lemma 7.16. 174
7.9 Games $\mathsf{Hyb}^{1,2}$, $\mathsf{Hyb}^{1,3}$ for the proof of Lemma 7.16. 174

7.10 Games $\mathsf{Hyb}^{1,4}$, $\mathsf{Hyb}^{1,5}$ for the proof of Lemma 7.16. 175
7.11 Games $\mathsf{Hyb}^{1,6}$, $\mathsf{Hyb}^{1,7}$ for the proof of Lemma 7.16. 176
7.12 Games $\mathsf{Hyb}^{i,1}$, $\mathsf{Hyb}^{i,2}$, $\mathsf{Hyb}^{i,3}$ for the proof of Lemma 7.16. 177
7.13 Games $\mathsf{Hyb}^{i,4}$, $\mathsf{Hyb}^{i,5}$, $\mathsf{Hyb}^{i,6}$ for the proof of Lemma 7.16. 178
7.14 Games $\mathsf{Hyb}^{i,7}$, $\mathsf{Hyb}^{i,8}$, $\mathsf{Hyb}^{i,9}$ for the proof of Lemma 7.16. 179
7.15 Games $\mathsf{Hyb}^{\ell,8}$, $\mathsf{Hyb}^{\ell,8*}$ for the proof of Lemma 7.16. 180
7.16 MKHS.Eval and MKHS.Ver algorithms of our multi-hop succinct multi-key homomorphic signature scheme MKHS. 182

8.1 Interactive proof constructed from a verifiable evaluation scheme on fingerprinted data VE and a fingerprinting scheme H. 191
8.2 Composition of VEs for functions f_1, f_2 following Proposition 8.5. 193
8.3 Construction of an interactive argument of knowledge Π for the relation $\mathcal{R}_\Pi$. 196
8.4 Multilinear sumcheck protocol $\mathsf{VE}_{\mathsf{SC}}$. 200
8.5 Modular construction of a VE for neural networks and compilation to an interactive argument of knowledge. 208
8.6 Illustration of a RNN with a loop at layer X_1. 210
8.7 Unrolled computation trace of the RNN in Figure 8.6. 210
8.8 Prover time for varying number of channels c, d and fixed $n = 64$ and $m = 4$. 212
8.9 Prover time for the convolution layers of a VGG11 network. 214

List of Tables

4.1 Comparison of algebraic functional commitments for functions $\mathbb{F}^{\ell} \rightarrow \mathbb{F}^{\ell_y}$. . . 41

6.1 Structure of the committed values in Figure 6.4. 132

8.1 Comparison between $\mathsf{VE}_{\mathsf{conv}}$ and the convolution proofs in zkCNN. 211

List of Publications

This thesis comprises five articles, listed in chronological order below. The first three are published in peer-reviewed academic conferences whereas the last two are under review at the time of writing.

1. *Chainable Functional Commitments for Unbounded-Depth Circuits.*

   David Balbás, Dario Catalano, Dario Fiore, Russell W. F. Lai.

   In proceedings of Theory of Cryptography – TCC 2023. [BCFL23]

2. *Modular Sumcheck Proofs with Applications to Machine Learning and Image Processing.*

   David Balbás, Dario Fiore, Maribel González-Vasco, Damien Robissout, Claudio Soriente.

   In proceedings of ACM Computer and Communications Security – CCS 2023. [BFG$^+$23]

3. *Fully-Succinct Multi-Key Homomorphic Signatures from Standard Assumptions*.

   Gaspard Anthoine, David Balbás, Dario Fiore.

   In proceedings of Advances in Cryptology – CRYPTO 2024. [ABF24]

4. *Circuit-Succinct Algebraic Batch Arguments from Projective Functional Commitments*.

   David Balbás, Dario Fiore, Russell W. F. Lai.

   Unpublished manuscript. [BFL25a]

5. *Pairing-based Functional Commitments for Circuits, Revisited*.

   David Balbás, Dario Fiore, Russell W. F. Lai.

   Unpublished manuscript. [BFL25b]

Other publications and manuscripts in submission that have been carried out throughout the PhD program, but whose results are not included in the thesis, are listed below.

6. *Analysis and Improvements of the Sender Keys Protocol for Group Messaging.*

   David Balbás, Daniel Collins, Phillip Gajland.

   In proceedings of RECSI 2022. [BCG22]

7. *Cryptographic Administrators for Secure Group Messaging.*

   David Balbás, Daniel Collins, Serge Vaudenay.

   In proceedings of USENIX Security 2023. [BCV23]

8. *WhatsUpp with Sender Keys? Analysis, Improvements and Security Proofs.*

   David Balbás, Daniel Collins, Phillip Gajland.

   In proceedings of ASIACRYPT 2023. [BCG23]

9. *Critical Rounds in Multi-Round Proofs: Proof of Partial Knowledge, Trapdoor Commitments, and Advanced Signatures*.

   Masayuki Abe, David Balbás, Dung Bui, Zehua Shang, Miyako Ohkubo, Akira Takahashi, Mehdi Tibouchi.

   Unpublished manuscript. [ABB$^+$24]

10. *Group Key Progression: Strong Security for Persistent Data*.

    Matilda Backendal, David Balbás, Miro Haller.

    Unpublished manuscript. [BBH25]

11. *FHorgEt: A Cryptographic Solution for Secure Machine Unlearning*

    David Balbás, Dario Fiore, Georgios Raikos, Damien Robissout, Claudio Soriente.

    Unpublished manuscript. [BFR$^+$25]

1 Introduction

1.1 Cryptography as a Formal Science

Throughout the last decades, cryptography has evolved from being the "art" of hiding information to becoming the formal science that protects the confidentiality, integrity, and authenticity of data in the digital world. Traditionally, cryptography was based on the principle of "security by obscurity", where the security of a system relied on keeping the algorithms that protect it secret. Fortunately, this is no longer true and the design of cryptographic algorithms is (generally) subject to public scrutiny. More importantly, the security that these designs achieve is systematically analysed using a formal methodology called the *provable security* paradigm.

In a nutshell, provable security consists of providing formal evidence that the security of a system relies on the difficulty of solving a well-established mathematical problem. The main advantage of this paradigm is that cryptographers and cryptanalysts can focus on trying to break the underlying problem, rather than the cryptographic system itself. As such, much of the cryptography that we use today is based on the hardness of a handful of mathematical problems, such as factoring large integers, computing discrete logarithms, or finding short vectors over lattices. The components of the provable security methodology are the following.

- **Functionality.** Towards building a sound cryptographic primitive or protocol, the first design consideration is correctness: *what are the algorithms of the scheme, and what is their intended behaviour when executed honestly?* As an example, consider digital signature schemes, which enable a signer to attest to the authenticity of a message. Any digital signature scheme is defined by three algorithms: a key generation algorithm that produces a pair of signing and verification keys, a signing algorithm for certifying a message, and a verification algorithm for checking the validity of a signature on a message. Besides, the definition should capture that if a signature is generated honestly on a message using a valid signing key, the signature will verify correctly for that message and the associated verification key.
- **Adversarial model.** The second consideration is the adversarial model: *what are the capabilities of the adversary we want to protect against?* In this thesis, we generally consider

either adversaries with unlimited computational resources (unbounded) or probabilistic polynomial-time adversaries (PPT), which are the most common adversarial models in cryptography.

- **Security definition.** The third consideration is the security definition: *what does it mean for a scheme to be secure?* In the case of signature schemes, the most natural idea is that an (admissible, according to the adversarial model) adversary should not be able to *forge* a signature, this is, to craft a signature that validates for a given verification key without knowing its corresponding signing key. Often, multiple security definitions coexist for the same primitive, leading to different constructions, trade-offs, and even impossibility results.

These aspects define the framework in which a cryptographic primitive or protocol is designed and analysed. As introduced before, the end goal of provable security is to show that a given construction satisfies the above properties (i.e. it is correct and secure in the stated adversarial model) by means of a formal proof, for which we additionally require the following components.

- **Construction.** This is the description of the actual cryptographic scheme that we study, whose behaviour needs to be formally specified by means of algorithms and a pseudocode-like description. For a given signature scheme, we need to describe how the algorithms for key generation, signing and verification actually work.
- **Assumptions.** Unless the adversaries we consider are unbounded, security theorems are generally not unconditional – they rely on the (presumed) hardness of some mathematical problem, which we call a *cryptographic assumption*. There exist a vast number of cryptographic assumptions, but they differ on their quality and on the amount of cryptanalysis that they receive. There are a handful of core, *standard assumptions* that the community accepts as being plausibly hard, as they have resisted cryptanalysis for decades. Examples include the discrete logarithm assumption and the learning with errors assumption. In this thesis, we will primarily consider public-key (i.e. number-theoretic) cryptographic assumptions.[1]
- **Security theorems and proofs.** Eventually, the outcome of this process is a theorem that shows a reduction (in a complexity-theoretic sense) from the security of the scheme to a cryptographic assumption. Such results are usually stated as follows: If assumption A holds, then construction C is secure according to security definition S in the given adversarial model. Or, in other words, *"If an admissible adversary can break construction C according to security definition S, then one can also break assumption A"*.

It may take years for the community to converge on a security definition that is considered satisfactory, but for the most common cryptographic primitives, security definitions are now stable and well understood. On the other hand, assumptions and constructions often play a "cat-and-mouse" game where novel assumptions are proposed to improve

[1] According to Impagliazzo's five worlds [Imp95], this thesis lives in Cryptomania.

existing constructions, and the former are cryptanalysed leading to breaks (or redesigns) of the latter. As novel assumptions are more likely to be broken, constructions whose security relies on standard assumptions are considered more robust and reliable.

The quest for advanced cryptographic primitives. While the traditional uses of cryptography over the internet have been essentially encryption, signatures and key exchange, a large fraction of the cryptographic community now focuses on the design and analysis of cryptographic objects with more advanced functionalities. Some prominent examples include secure multiparty computation [Yao82, GMW87], functional encryption [SW05, BSW11], fully homomorphic encryption [Gen09], and proof systems [GMR85]. These primitives go beyond the usual confidentiality, authenticity and integrity goals for data at rest and in transit, achieving almost-magical security guarantees for data under computation. Let us illustrate this with an example. Consider a scenario where a user Alice authenticates a large data set $m_1, \dots, m_n$ with a signature scheme, producing signatures $\sigma_1, \dots, \sigma_n$. To verify that $m_1, \dots, m_n$ have not been tampered with (integrity) and indeed come from Alice (authenticity), any user Bob can simply check the signatures using Alice's public key. Now, consider that a third entity, called the *evaluator*, performs a computation on Alice's data, denoted as $y = f(m_1, \dots, m_n)$, and sends y to Bob. *How can Bob be convinced that y is the correct result obtained by running f on data signed by Alice?* A trivial solution is to send all data to Bob and let him verify the signatures and recompute f, but this is slow and renders the evaluator's work useless. *Cryptographic proof systems* provide a better solution by enabling the evaluator to *generate a short, publicly-verifiable proof* π that (a) y is obtained by actually computing f on some data $m_1, \dots, m_n$, and that (b) $m_1, \dots, m_n$ were messages signed by Alice.[2] Then, Bob can simply verify the proof π to be convinced of the validity of these statements efficiently[3], even without the knowledge of the signatures and the data itself.

The complexity of designing proof systems, and in general any advanced cryptographic primitive, raises research questions at multiple levels. At a foundational level, researchers aim to show feasibility, impossibility, and separability results under different families of cryptographic assumptions and adversarial models. For example, it is known that some families of proof systems cannot be proven secure from standard assumptions [GW11]. These results are of paramount importance as they identify the limitations and requirements intrinsic to certain primitives, improving our understanding on how (not) to build them. For more established primitives and solutions, there is an important effort on breaking asymptotic barriers towards better concrete efficiency, as well as to improve the quality and the analysis of the assumptions that are used. Eventually, a more practical line of work aims to develop efficient primitives with optimizations, carefully selected parameters, and

[2] Looking ahead, this is exactly the setting that motivates *homomorphic signature schemes*, which we introduce in the next section.

[3] In this introduction, we use the term efficiency to refer to concretely small running times, i.e., efficient algorithms are those that can actually be run on modern hardware in a reasonable amount of time. This is in contrast to the use of the term efficient as a synonym of polynomial-time, which is usual in complexity theory.

reference implementations. It is worth noting that constructions of advanced primitives that support general computation are often not efficient enough for practical use. Therefore, the community also seeks better performing solutions tailored for specific applications. The entire process is not a straight line, but rather a feedback loop where foundational results introduce novel techniques for building better primitives, which then need to be improved and made practical.

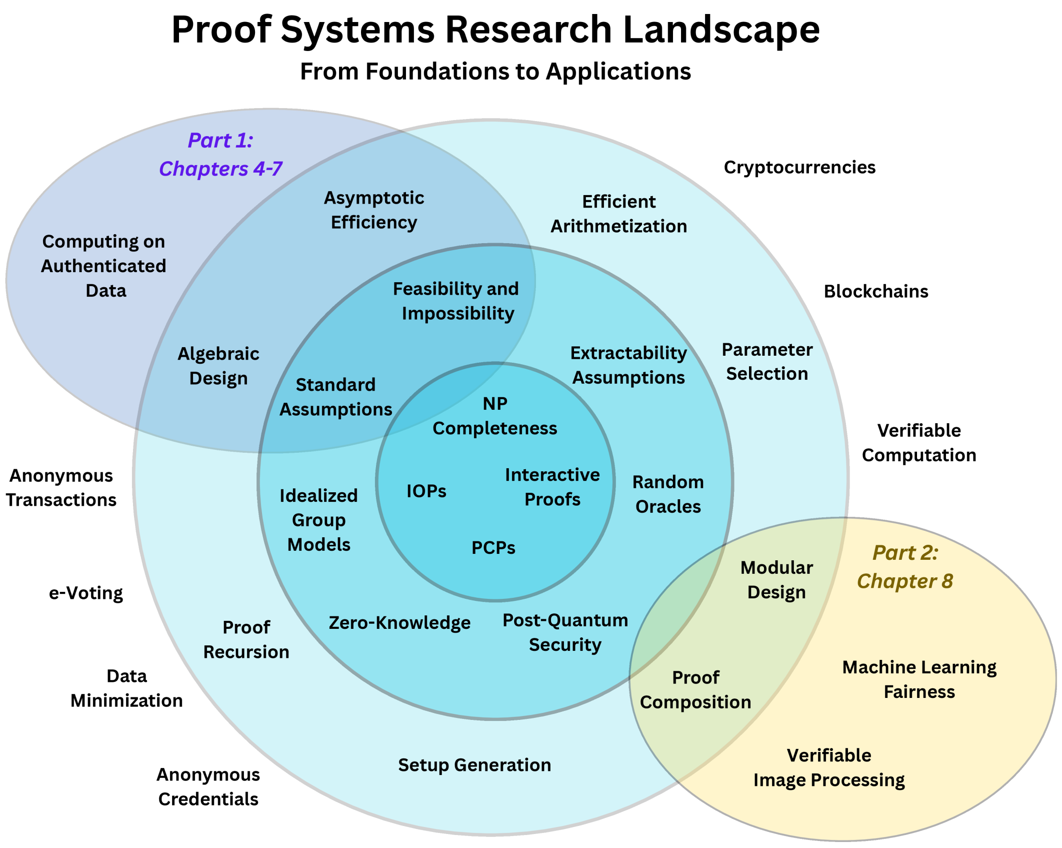

Figure 1.1: *Positioning of this thesis in the landscape of research directions in cryptographic proof systems. The inner circle corresponds to* computational complexity, *which is at the core of any proof system. The inner ring corresponds to* cryptography, *including assumptions and models. The outer ring corresponds to* design *challenges and efficiency considerations. The outer area corresponds to* applications.

Where is this thesis positioned? Throughout the thesis, we address several research questions on cryptographic proof systems, which we introduce in further technical detail in the coming section. In Figure 1.1, we position our results in the landscape of research directions in cryptographic proof systems. Many terms that appear in Figure 1.1 may not be familiar to the reader at this point, but they will be introduced in the following sections. In the first part of the thesis, we focus primarily on foundational problems. We provide positive results on constructing some families of proof systems from standard cryptographic assumptions for the first time. We also introduce novel techniques that enable us to break asymptotic barriers towards improving their efficiency. In the second part, we design and implement proof systems tailored for applications of practical interest.

Our solutions significantly improve existing solutions in terms of modularity and efficiency, both concretely and asymptotically.

1.2 Cryptographic Proof Systems from Theory to Practice

A proof system enables an entity, the *prover*, to convince another entity, the *verifier*, that a given statement is true. The first proof systems were introduced in the 1980s, but they were comically inefficient (even if polynomial-time) and could not really be used in practice. In the last two decades, the field has seen a series of breakthroughs that have led to a rich landscape of efficient proof systems. Today, they have countless applications both within and beyond cryptography, including verifiable computation [GGP10], anonymous credentials [CL01], electronic voting [Cha81], privacy-preserving transactions [BCG$^+$14], blockchains, and many others.

In this section, we introduce proof systems, their properties, and state several research questions that motivate the results in this thesis.

1.2.1 Proof Systems, Zero-Knowledge and Succinctness

Historically, the first proof systems were interactive proofs (IPs) [GMR85]. As the name reflects, in an IP the prover and verifier exchange a series of messages where the verifier can ask questions (challenges) and the prover must respond accordingly. The goal of the protocol is to show that a given statement $\mathtt{x}$ is true, i.e., that $\mathtt{x}$ belongs to a given language $\mathcal{L}$ for which the protocol is designed.[4] At the end of the interaction, the verifier accepts or rejects based on the transcript of the messages exchanged. Such a protocol must satisfy *completeness* (a correctness notion) and *soundness* (a security notion). Completeness means that an honest prover will succeed in convincing a verifier of the validity of a true statement $\mathtt{x} \in \mathcal{L}$. Soundness implies that no malicious prover can convince a verifier that a false statement $\mathtt{x} \notin \mathcal{L}$ is true, except with small probability.

Interaction, however, presents some drawbacks. The interacting parties need to be online at the same time, and communication can be expensive or slow. Also, the prover cannot proceed until it receives the verifier's challenges. Moreover, interactive proofs cannot be used to convince a third party that was not involved in the interaction, and the prover needs to repeat the interaction for every verifier. This is in strong contrast to the traditional notion of a mathematical proof, which can be checked independently once it is generated. To address these issues, there is a large interest in studying *non-interactive* proofs, where the prover generates a single message (the proof) that can be checked by a verifier offline.

Building a non-interactive proof for a language $\mathcal{L}$ in the class **NP**[5] is essentially trivial.

[4] The power of IPs is surprising: they enable devices with limited computational power, such as smartphones, to verify any computation that can be carried out using a polynomial amount of space [Sha90].

[5] This complexity class is rich enough to capture many interesting computational problems, including virtually all mathematical assumptions that are useful in cryptography. For example, a proof system for NP allows one to prove possession of the secret key associated to a given public key, that two ciphertexts encrypt the

By definition of the class NP, for any $\mathtt{x} \in \mathcal{L}$ there exists a witness $\mathtt{w}$ that allows a verifier to decide language membership in polynomial time. This is, there exists a polynomial-time algorithm C such that $C(\mathtt{x}, \mathtt{w}) = 1$ if and only if $\mathtt{x} \in \mathcal{L}$. Hence, the prover can simply send $\mathtt{w}$ to the verifier, which will run $C(\mathtt{x}, \mathtt{w})$ and accept or reject accordingly. Naturally, this approach presents serious limitations. First, the prover may not want to expose the witness to the verifier, such as if we are proving possession of a secret key. Second, the size of $\mathtt{w}$ or the time required to run C can be very large and sending it to the verifier may not be practical. For instance, if we are aggregating many signatures (e.g. for compressing transactions on a blockchain), sending all signatures defeats the purpose of the aggregation. Therefore, non-interactive proofs must have some additional properties to be advantageous in practice, the most important ones being *zero-knowledge* and *succinctness*.

- **Zero-knowledge.** A proof system is zero-knowledge [GMR85] if the verifier does not learn anything about the statement being proven other than its validity. This property is formalized by means of a polynomial-time algorithm called the *simulator*, which thanks to some additional power (such as knowledge of a trapdoor embedded in the proof system), can generate a valid-looking proof for any (possibly false) statement $\mathtt{x}$ without knowing a witness for it. As simulated proofs are indistinguishable from honest proofs and can be generated without knowing a witness, it follows that no information from the witness can be extracted from a valid proof either.
- **Succinctness.** A proof system is succinct if the size of the proof is small compared to the size of the witness of the NP relation being proven (or the size of the circuit C). Succinctness enables encoding proofs for very large statements, such as for valid inference of a convolutional neural network or for aggregation of many financial transactions, into a string of a few hundred or thousand bytes.

Unfortunately, there is strong evidence that building succinct non-interactive proofs that achieve unconditional soundness is impossible [GH98, GVW02, Wee05], and similarly for zero-knowledge proofs where both soundness and zero-knowledge hold unconditionally [AH87, GK96, Rot25]. The good news is that we can relax soundness to be computational, that is, to hold only against polynomial-time adversaries while relying on the hardness of some computational assumption. We call such proof systems *cryptographic proofs* or simply *arguments*. For zero-knowledge, it was shown early on that any NP language admits a computational zero-knowledge argument, both in the interactive case [BGG$^+$90] and in the non-interactive case (NIZK) [BFM88]. While it is relatively straightforward to add the zero-knowledge property to most NIZKs (at least in theory, albeit at a cost in concrete efficiency), achieving succinctness introduces several surprising additional challenges.

same value, or that an image has been modified in a legitimate way with respect to the original. Furthermore, provers for languages in NP can run in polynomial time, but this is not always possible outside of this class.

1.2.2 Succinct Non-Interactive Arguments

Proof systems that are succinct and non-interactive are called succinct non-interactive arguments (SNARGs) [Kil92, Mic94]. Essentially, SNARGs enable a prover to convince a verifier of a statement's validity with a short proof. For example, a SNARG for NP can prove the validity of a statement $\mathtt{x}$ checkable by a circuit C, i.e., that $\exists \mathtt{w} : C(\mathtt{x}, \mathtt{w}) = 1$, with a proof of size $\mathsf{poly}(\lambda, \log|\mathtt{w}|)$, that is, only logarithmic in the size of the NP witness (or even constant). Most SNARGs work in a preprocessing model, where the verifier (or any third party) can preprocess the the circuit C before the proof is generated. Then, the "online" verification time can be as short as the size of the proof, i.e., also $\mathsf{poly}(\lambda, |\mathtt{x}|, \log|\mathtt{w}|)$, or simply $\mathsf{poly}(\lambda, \log|\mathtt{w}|)$ if the statement $\mathtt{x}$ is also pre-processed by hard-coding it into the circuit. Nowadays, there is a vast number of approaches to building SNARGs, which we survey (non-exhaustively) in Section 2.4.

Security of SNARGs. Most families of SNARGs work in the so-called common reference string (CRS) model, in which the prover and the verifier have access to a public string which includes some randomness or correlated information [Dam00].[6] The standard security notion of SNARGs is *adaptive soundness*, which states that, given an NP language $\mathcal{L}$, no polynomial-time adversary can produce an accepting proof π for a false statement $\mathtt{x} \notin \mathcal{L}$. The term *adaptive* means that the adversary can choose the statement $\mathtt{x}$ to be proven after seeing the CRS, which is the setting that is the closest to real-world settings.

The notion of adaptive soundness is however insufficient in many cases. Consider for example the NP language of the discrete logarithm relation for a group $\mathbb{G}$ of prime order p, where a statement $\mathtt{x} = (g, h)$ are two nontrivial group elements $g, h \in \mathbb{G}$ and the goal is to prove that $g^a = h$ for some exponent $\mathtt{w} = a \in \mathbb{F}_p$. This statement is always true regardless of what g, h are, as there is always some witness $a \in \mathbb{F}_p$ that satisfies the relation. Hence, any SNARG for this language would be trivially sound — the challenge is to prove the *knowledge* of the witness rather than its mere existence. This intuition is formalized by the notion of *knowledge soundness*, which states that for every polynomial-time adversary that generates a valid proof π for some statement $\mathtt{x}$, there exists a polynomial time algorithm called the *extractor* which, given access to the code of the adversary, can output a witness $\mathtt{w}$ such that $C(\mathtt{x}, \mathtt{w}) = 1$ with very high probability. Proving knowledge soundness is challenging, as one must explicitly build such an extractor and argue about its success probability.[7]

We remark that both soundness and knowledge soundness are compatible with the

[6] A common template for a CRS is to include correlated group elements. For instance, in a prime order group setting $\mathbb{G}$, a usual CRS may include elements $g, h_1, h_2, \ldots, h_n \in \mathbb{G}$ such that $h_i = g^{a^i}$ for some $a \in \mathbb{Z}_p$. Here, a is a CRS trapdoor, and its knowledge would enable the prover to cheat. The CRS has to be generated either by a trusted party or by a distributed protocol ran among multiple parties [BCG+15, NRBB22].

[7] Although extractors are generally only used in proofs by reduction, there are examples where the extractor of a proof system is actually executed as part of a bigger protocol. In our recent work [ABB+24], not included in this thesis, we explore the usefulness of extractors (and zero-knowledge simulators) of a class of multi-round proof systems. Their use in constructions enables surprising applications such as trapdoor commitments [Dam90], proofs of partial knowledge [CDS94, ACF21, ABO+24], or even advanced signature schemes such as adaptor signatures [Poe17, AEE+21].

zero-knowledge property as discussed earlier.

Assumptions in SNARGs. According to the definition of extractability that we just introduced, it may seem surprising that a SNARG can actually achieve such a notion. Indeed, a SNARG proof is a string of size $\mathsf{poly}(\lambda, \log|\mathtt{w}|)$, and the extractor must be able to output a witness $\mathtt{w}$ of size $|\mathtt{w}|$. So, information-theoretically, it seems impossible to extract a witness from a proof that is much smaller than the witness itself, as some information is lost due to the compression. One could try to build an extractor which actually *solves* the problem of finding a witness for $\mathtt{x}$, but this is hard in general for NP relations as the extractor must run in polynomial time.

This intuition was formalized by Gentry and Wichs [GW11], who proved that it is impossible to construct adaptively-sound SNARGs[8] for NP whose security relies on a black-box reduction to a large class of cryptographic assumptions that we call *falsifiable*. Falsifiable assumptions [Nao03] are assumptions where the winning condition can be checked by the challenger in polynomial time. Essentially all standard cryptographic assumptions, including but not limited to DDH, CDH, RSA, SIS, and LWE, are falsifiable.[9] On the other hand, non-falsifiable assumptions are those whose winning condition cannot be checked in polynomial time. The most prominent examples include the random oracle model [BR93], idealized group-based models (such as the generic group model [Sho97, Mau05] and the algebraic group model [FKL18]) and knowledge assumptions.

Let us illustrate these concepts with examples. Consider the computational Diffie-Hellman (CDH) assumption, which can be stated as follows. Given a group $\mathbb{G}$ of prime order p and a generator $g \in \mathbb{G}$, the challenger samples $a, b \in \mathbb{F}_p$ at random and sends two group elements $h_1 = g^a$ and $h_2 = g^b$ to the adversary, whose goal is to return the element g^{ab}. Observe that the challenger can easily check whether the adversary's output is correct as it knows a, b in the clear and can compute g^{ab}.

On the other hand, consider the knowledge of exponent assumption [Dam92, BP04] (KEA). Given a group $\mathbb{G}$ of prime order p and a generator $g \in \mathbb{G}$, the challenger samples $a \in \mathbb{F}$ at random and gives $h = g^a$ to the adversary. The adversary's goal is to return two group elements $x, y \in \mathbb{G}$ such that $y = x^a$. The assumption says that for any successful adversary, there exists an extractor that can output a coefficient $b \in \mathbb{F}$ such that $x = g^b$ and $y = h^b$ in polynomial time. Namely, we are assuming that the only way in which *any* adversary can obtain such x, y is by computing g^b and h^b for some $b \in \mathbb{F}_p$. Naturally, a challenger does not have a way to decide whether the adversary crafted x, y in this way or not, and so the assumption is not falsifiable. To see why such assumptions may help proving knowledge soundness of a SNARG, observe that the KEA itself assumes the existence of an extractor which can look at the adversary's memory to retrieve the coefficient $b \in \mathbb{F}$. Therefore, this "assumption-extractor" can give us the additional information we need to build a successful SNARG extractor.

[8] We remark that their celebrated result applies also to sound SNARGs and not only to knowledge-sound SNARGs.

[9] Falsifiable assumptions are considered part of the "standard model" of cryptography.

To summarize, Gentry and Wichs' result shows that, to build fully-fledged SNARGs for NP, we need to give up on falsifiable assumptions.[10] But falsifiable assumptions are objectively of better quality than non-falsifiable ones and strongly preferred by the cryptographic community. Therefore, given the importance of SNARGs and their many applications in practice, finding ways to circumvent the impossibility is a strongly motivated and popular line of research.

1.2.3 Succinct Non-Interactive Proof Systems from Standard Assumptions

In search of stable foundations for SNARGs, various lines of research have explored alternative methods to construct succinct arguments that rely on falsifiable assumptions. One notable approach is to *relax the full power of SNARGs*, while ensuring that they remain useful in applications. Depending on which property is loosened, we obtain a family of flavours of proof systems:

- Relaxing the *expressiveness* of SNARGs and focusing only on deterministic polynomial-time computations leads to the notion of *delegation schemes* (also called SNARGs for **P**) [KPY19, GZ21, JKKZ21, CJJ22]. These proof systems have a direct application to verifiable computation where the verifier knows all the inputs of a computation.
- Relaxing the *soundness* property of SNARGs and focusing on a weaker property called *evaluation binding* leads to the notion of *functional commitments* (FCs) [LRY16].
- Relaxing the *succinctness* of SNARGs leads to the notion of *batch arguments for NP* (BARGs) [KPY19, CJJ21].
- Finally, it is also possible to aim for SNARGs that are *tailored to specific applications* instead of general-purpose computations, leading to primitives such as *homomorphic signatures* (HS) [JMSW02].

The above relaxations are not necessarily weaknesses but can actually be interesting features — these proof systems can be an attractive alternative to SNARGs wherever their respective notions are sufficient, while relying only on standard, well-founded assumptions. Besides the notable technical challenge of building efficient instantiations of these primitives, there are still lots of open questions surrounding them, including finding more applications where they can effectively replace the use of SNARGs. The first part of this thesis addresses several open questions on functional commitments, batch arguments for NP, homomorphic signatures, and the connections between them. As many families of SNARGs do, these proof systems work in the CRS model. We introduce them in detail below.

[10]This is not entirely precise as there are two known approaches to circumventing Gentry-Wichs. One is to use non-black-box reductions to falsifiable assumptions, which is a very active line of research [WW24a, MPV24, DWW24, JKLM25]. At a very high level, these works rely on complexity leveraging and subexponential hardness assumptions to actually build an extractor that computes an NP witness. The other is to build non-adaptively sound SNARGs, which are SNARGs that sound only for statements that are not adaptively chosen by the adversary after seeing the CRS [SW14]. Both approaches seem to be highly non-practical at the time of writing this thesis, as they extensively rely on indistinguishability obfuscation [BGI$^+$01].

Functional Commitments

Commitment schemes are ubiquitous primitives in cryptography that allow one to compress a (vector of) message $\boldsymbol{x}$ into a succinct commitment com. Later, one can reveal $\boldsymbol{x}$ by sending it along with a short opening proof. Commitments must be *binding*, meaning that it should be infeasible to open a commitment com to two different $\boldsymbol{x}, \boldsymbol{x}'$ successfully. Commitments may also be *hiding*, meaning that com does not reveal anything about $\boldsymbol{x}$.

A functional commitment scheme [LRY16] enables opening the commitment com to *a function of the committed message*, that is, revealing $f(\boldsymbol{x})$ for some function f. Naturally, the property that makes functional commitments interesting is that not only the commitments but also the functional opening proofs are succinct, i.e., sublinear in the size of the function f.

There are essentially two ways to contextualize this primitive. First, FCs can be seen as a generalization of commitment schemes that support more expressive openings, in the line of vector commitments [CFM08, LY10, CF13] and polynomial commitments [KZG10]. Second, FCs can be seen as a class of (commit-and-prove) SNARGs with a weaker security property than knowledge soundness called *evaluation binding*. Essentially, evaluation binding says that no polynomial-time adversary should be able to validly open any commitment com to two different values $y \neq y'$ for the same function f. This is a natural generalization of the standard notion of commitment binding. As mentioned above, evaluation binding is actually a feature, as it enables instantiations of functional commitments from standard assumptions. For some applications, such as homomorphic signatures and verifiable databases, evaluation binding FCs can totally replace SNARGs as shown in [CFT22]. Furthermore, any functional commitment scheme implies a universal delegation scheme or SNARG for P. An additional motivation to study functional commitments is that they offer an alternative blueprint to constructing SNARKs, yielding elegant constructions with modular security proofs. Indeed, any evaluation binding FC can be compiled into a knowledge-sound SNARK by adding a simple proof of knowledge for the commitment, i.e., for a statement such as "I know a vector $\boldsymbol{x}$ that opens the commitment".

Prior to this thesis, there were different constructions of functional commitments that supported limited classes of functions. Besides the special cases of vector and polynomial commitments, these included linear maps [LRY16, LM19], semi-sparse polynomials [LP20] and constant-degree polynomials [ACL$^+$22, CFT22]. However, no construction that could support arbitrary functions was known, nor did it seem obvious that existing techniques could lead to such a result. Therefore, the main open question in the field was the following:

Can we build a succinct functional commitment scheme for arbitrary functions?

Batch Arguments for NP

A batch argument for NP allows one to prove membership of a collection of NP statements $\mathtt{x}_1, \ldots, \mathtt{x}_k$, or in other words, that $\forall i = 1, \ldots, k, \exists \mathtt{w}_i : C(\mathtt{x}_i, \mathtt{w}_i) = 1$ for the circuit C that decides the NP language. As opposed to the usual notion of SNARKs where the proof is polylogarithmic on the size of the witness (which in this case would be the entire vector of witnesses $\mathtt{w} = (\mathtt{w}_1, \cdots, \mathtt{w}_k)$), the succinctness notion of BARGs is more relaxed. Namely, the proof is only required to scale *sublinearly in the number of statements*, e.g., of size $\mathsf{poly}(\lambda, |C|, \log k)$, but not sublinearly in the circuit size, or in the witness size. Due to its distinctive properties — succinct proofs and realisations from standard falsifiable assumptions — BARGs are emerging as a useful tool to realise other cryptographic primitives from standard assumptions, similarly to functional commitments. Notably, BARGs can also be used to build delegation schemes [BHK17, KPY19, KVZ21, CJJ22] and to aggregate signatures [WW22, BCJP24].

In the state of the art, the main approach to construct BARGs involves probabilistically checkable proofs (PCPs) and correlation intractable (CI) hash functions [CGH98, CCH+19, PS19, JJ21]. This direction has led to non-black-box constructions from LWE [CJJ22, DGKV22, PP22] or subexponential DDH [CGJ+23]. Notably, the constructions of [DGKV22, PP22] achieve a proof size (and verification time for [PP22]) of $|\pi| = |\mathtt{w}| + O(|\mathtt{w}|/\lambda) + \mathsf{poly}(\lambda, \log k)$, which is sublinear in $|C|$ and, more importantly, it is *rate-1*, meaning that the $|\mathtt{w}|$ term is not multiplied by any other factor. However, despite their near-optimal asymptotic efficiency, these constructions have large concrete overheads, i.e. the poly functions in proof size and verification time expressions are large and not explicitly specified. Hence, these techniques lead to "galactic" proof systems and are unlikely to yield practical schemes.

An exception to the approach above is the work of Waters and Wu [WW22] who gave direct algebraic constructions over bilinear pairings based on the decision linear (k-Lin) assumption. Leveraging the algebraic structure of the assumptions relied on by the schemes often translates into more concretely efficient, implementable, and relatively simple constructions. Indeed, the scheme of [WW22] achieves a proof size of $|\pi| = O(\lambda \cdot |C|)$ with small concrete constants. The result is later improved by [GLWW24] who reduces the public parameter size while keeping essentially the same design. However, in both [WW22, GLWW24] the verifier runs in time $\Omega(\lambda \cdot |C|)$, even if offline preprocessing is allowed. Hence, a natural open question is whether we can build a BARG which achieves optimal succinctness, i.e., where proofs grow linearly with the size of one witness (but not necessarily rate-1, i.e., modulo λ factors). In summary, we ask the following question:

Can we build an algebraic BARG for NP where the proof size only depends on the size of a single witness?

Homomorphic Signatures

In parallel to the effort of ensuring privacy for data used during computation, which is the object of study of, e.g., works on fully homomorphic encryption [Gen09], another important goal is to provide data *authenticity*. Recall the example from earlier on, where a user Alice authenticates a large data set $m_1, \ldots, m_n$ with a signature scheme, producing signatures $\sigma_1, \ldots, \sigma_n$. Then, an evaluator performs a computation $y = f(m_1, \ldots, m_n)$ on Alice's data and sends y to Bob, which has to either trust the evaluator or verify the signatures and recompute y on his own. *Homomorphic signatures* [JMSW02] stand out as a solution to provide *succinct authenticity-preserving proofs of computation*. They allow the evaluator to compute on signed data, deriving not only the output y but also a succinct signature $\sigma_{f,y}$ that acts both as an authenticator of y and as a proof of computation for f. Anyone can publicly verify the tuple $(f, y, \sigma_{f,y})$ and get convinced of the correctness of y as the result of computing f on Alice's data, without having to download the large data input. Homomorphic signatures may incorporate useful additional properties, such as amortized efficiency (enabling verification in time independent of the complexity of f, after preprocessing), multi-hop evaluation (supporting evaluation over already evaluated signatures), and context-hiding (preventing the verifier from learning information on the inputs beyond the computation's output).

Prior to this thesis, the state of the art in homomorphic signatures from standard assumptions included the lattice-based construction of Gorbunov, Vaikuntanathan and Wichs, which supports the evaluation of arbitrary boolean circuits whose depth has to be (polynomially) bounded at setup time [GVW15]. Then, Catalano, Fiore and Tucker [CFT22] showed how to build homomorphic signatures generically from additive-homomorphic functional commitments, where the HS supports the same class of functions as the functional commitment. An important open question is thus the following:

Can we build homomorphic signature schemes for unbounded-depth circuits, optimally supporting additional properties such as efficient verification, multi-hop evaluation and context-hiding?

Multi-key Homomorphic Signatures. In many scenarios, however, computations are performed on data that belongs to (and is authenticated by) multiple entities. Typical examples include aggregating data collected by several hospitals for clinical studies, smart monitoring of signals produced by IoT devices (e.g., medical/environmental/traffic sensors, wearable devices, etc.), or transactions made by different users in a blockchain. In this context, the standard notion of homomorphic signatures falls short, since it requires that all messages are signed under the same key. To address this issue, Fiore, Mitrokotsa, Nizzardo, and Pagnin introduced *multi-key homomorphic signatures* (MKHS) [FMNP16]. In a MHKS, the *evaluator* computes a function f over n messages $m_1, \dots, m_n$, where each m_i is authenticated by someone in a set of t parties that we denote by $\mathsf{id}_1, \dots, \mathsf{id}_t$. In this case, the resulting signature $\sigma_{f,y}$ must vouch for the correctness of y as the output of f on inputs that were signed under public signature keys $\mathsf{vk}_1, \dots, \mathsf{vk}_t$, where each vk_i corresponds to id_is.

The construction of succinct MKHS involves greater challenges than for their single-user counterparts. Prior to this thesis, all the MKHS schemes from falsifiable assumptions [FMNP16, FP18, SBB19, SFVA21] achieved only a weak notion of succinctness in which, for a function f with n inputs signed by t distinct users, the size of $\sigma_{f,y}$ grows as $\mathsf{poly}(\lambda, t, \log n)$, i.e., at least linearly in the number of users involved in the computation. Even if this level of succinctness may be acceptable in applications where a few users provide each a large amount of data, it is clearly undesirable in scenarios that involve computing on data from many parties, such as the case of IoT sensors or users in a blockchain. The state of the art in MKHS therefore raises the following open question:

Can we build a fully-succinct multi-key homomorphic signature scheme supporting arbitrary functions?

1.2.4 Practically Efficient Proof Systems for Real-World Applications

Beyond inspiring many foundational questions, SNARGs have a huge potential for solving real-world problems. While primitives such as BARGs and functional commitments still have limitations in terms of practical efficiency, some SNARGs based on stronger assumptions do achieve practical performance. In the second part of this thesis, we address the challenge of building SNARGs for real-world applications that involve large inputs, with an emphasis on *proof composability* and *prover efficiency*.

One of the most prominent applications of SNARGs where efficient provers are crucial is the outsourcing of computation. In this setting, clients outsource the data processing task to a potentially untrusted server that (i) has enough resources to carry out the computation and optionally (ii) may hold additional data that is required to complete the task but that cannot be shared with clients. As an example, consider the scenario where a bank owns a machine learning model F that decides credit worthiness $Y = F(X, W)$, given some customer data X and model parameters W. A proof system for this scenario should provide *publicly verifiable* (hence auditable) proofs with strong guarantees for:

- **Integrity.** The prediction is indeed generated by the model, given solely the data provided by the customer and the model parameters. Integrity also guarantees that no bias or unauthorized data—such as gender or race—were used in the computation. This is relevant as the bank (or similar stakeholders) must abide to legal directives that forbid discrimination when providing goods or services [Cou00, Cou04].
- **Fairness.** If the model is certified by a third-party auditor, customer may obtain guarantees of fair treatment, i.e., the decision process has been the same across all customers. We note that Supreme Audit Institutions have recently defined best-practices to audit ML models and certified ML may be soon available in real-world applications [the20, Fed22].
- **Privacy.** If the model parameters W are proprietary, the bank may publish a (certified) commitment to W while proving that $Y = F(X, W)$ in zero knowledge. This allows the customer to verify that computation was carried out correctly, while W is kept private and nothing is leaked other than what can be inferred by the prediction itself.

Despite the rapid progress in the last decade, general-purpose cryptographic proof systems fail to scale well to very large inputs. The main bottleneck appears at the prover side, both on running time and memory usage. For example, if one wants to prove the evaluation of a ML model using an IOP-based proof [BCS16] such as Plonk [GWC19], one needs to start by committing to the entire computation trace which, for commodity hardware and medium-sized models, rapidly exhausts the computer's memory. Among the many families of cryptographic proof systems in the literature, sumcheck-based proof systems [XZZ+19, Set20, GLS+23, XZS22, CBBZ23] achieve the best prover performance, which is asymptotically optimal: linear in the size of a circuit that performs the computation. Nevertheless, modelling computation as a circuit introduces high overheads that make even these systems impractical when executed on computations that process large amounts of data.

Dedicated proof systems trade-off generality for performance. In particular, they *avoid the general circuit encodings* of their general-purpose counterparts, and achieve better performance, albeit for *restricted classes of functions*. For example, previous work has shown how to exploit the sequentiality and low multiplicative depth of some classes of functions to achieve low overhead both for provers and verifiers. Two examples of concrete practical interest are proofs of machine learning inference and proofs of image processing. Works such as vCNN [LKKO20] and zkCNN [LXZ21] enable verifiable ML applications by exploiting the sequential composition of neural network-like architectures, where data is processed one layer (i.e, function) at a time and the output of the current layer is fed as input to the next one. The same principles are used by PhotoProof [NT16] and ZK-IMG [KHSS22] that exploit the sequential composition of image processing tasks and provide proof systems tailored to verifiable image processing.

Prior to this thesis, protocols such as the ones above unfortunately present poor composability and leave little room for modification and improvement. This severely limits their applicability in e.g. data processing pipelines, where different operations are applied to the inputs sequentially. Moreover, there seems to be room for improving the efficiency of existing proofs for these applications, both concretely and asymptotically. Therefore, an important open question in practice is the following:

Can we build proof systems that combine the versatility of general-purpose proofs and the efficiency of special-purpose proofs?

And, by focusing on the specific applications of machine learning inference and image processing, we also note the following question:

Can we design and implement efficient and modular proofs for machine learning inference and image processing?

1.3 Thesis Contributions

1.3.1 Functional Commitments for All Functions

In Chapter 4, we present the first constructions of functional commitments for all functions based on falsifiable assumptions. Our main contribution is the introduction of a novel primitive that we call chainable functional commitment (CFC), which extends the functionality of FCs by allowing one to 1) open to functions of multiple inputs $f(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$ that are committed independently, 2) while preserving the output also in committed form. We show that CFCs for quadratic polynomial maps generically imply FCs for arithmetic circuits. Then, we efficiently realise CFCs for quadratic polynomials over pairing groups and lattices, resulting in functional commitment schemes for arithmetic circuits of unbounded depth (but bounded width) based on either pairing-based or lattice-based falsifiable assumptions. Additionally, our functional commitments feature useful properties such as being additively homomorphic and supporting efficient verification with preprocessing. Using the transformation from [CFT22] that constructs homomorphic signatures from FCs, we also obtain the first pairing- and lattice-based homomorphic signatures that support the evaluation of unbounded-depth circuits.

The main drawback of our constructions in Chapter 4 is the size of the commitment key, which scales as $O(n^5)$ for a circuit of width n. In Chapter 5, we improve on this aspect by introducing a new construction of CFCs for quadratic functions which achieves a commitment key of size $O(n^3)$ while preserving all the interesting properties of the previous construction, such as additive homomorphism and efficient verification. For this scheme, security relies on a series of falsifiable q-type assumptions over pairing groups. Our construction is built following a modular framework that can be used to simplify the design principles of other functional commitment schemes.

1.3.2 Circuit-Succinct Algebraic Batch Arguments

In Chapter 6, we give the first algebraic (pairing-based) construction of BARG that is circuit-succinct, achieving proof size and online verifier runtime $O(\lambda \cdot |\mathtt{w}|)$. We achieve our result by means of a compiler which builds a BARG generically from a new primitive called *projective chainable functional commitment* (PCFC). PCFCs are essentially a generalization of CFCs which support somewhere extraction, subvector projection, and functional openings. We then construct a PCFC from the standard MDDH assumption in bilinear groups, extending and generalising the proof systems that conform the functional commitment for circuits by Wee and Wu [WW24b]. Our black-box transformation may be of independent interest for understanding the connection between functional commitments and BARGs and towards obtaining other algebraic constructions of the latter.

1.3.3 Fully-Succinct Multi-Key Homomorphic Signatures

In Chapter 7, we present the first construction of multi-key homomorphic signatures that are fully succinct while achieving adaptive security under standard falsifiable assumptions. Our MKHS achieve succinctness $\mathsf{poly}(\lambda, \log n, \log t)$ where n is the number of evaluated messages and t is the number of users that sign these inputs. Our result is achieved through a novel combination of batch arguments for NP and functional commitments, and yields diverse MKHS instantiations for circuits of unbounded depth based on either pairing or lattice assumptions. For instance, by choosing an adequate BARG and FC, we can even obtain signatures whose size is constant in n and t, while still supporting arbitrary functions. Additionally, our scheme supports efficient verification with pre-processing, and they can easily be extended to achieve multi-hop evaluation (for a constant number of steps) and context-hiding.

1.3.4 Modular and Efficient Proofs for Machine Learning and Image Processing

In Chapter 8, we present a family of efficient proof systems for verifiable computation of sequential operations. Our solutions aim to combine the performance of tailored solutions for specific applications with the versatility of general-purpose proof systems. The main tool of our framework is a new information-theoretic primitive called Verifiable Evaluation Scheme on Fingerprinted Data (VE) that captures the properties of diverse sumcheck-based interactive proofs, including the well-established GKR protocol [GKR08]. Then, we show that VEs for specific functionalities can be composed sequentially at an information-theoretic level, avoiding the notable overhead of committing and opening after every step and yielding memory-efficient provers. Thanks to this framework, we build proof systems that enable the verifiability of data-processing pipelines.

We propose a novel VE for convolution operations that can handle multiple input-output channels and batching, and we use it in our framework to build proof systems for convolutional neural networks and for image processing operations. We produce a proof of concept implementation of our schemes and show that we achieve up to $5\times$ faster proving time and $10\times$ shorter proofs compared to the state-of-the-art, in addition to asymptotic improvements.

2 Related Work

In this chapter, we include a discussion on the state of the art which is centred around the existing literature at the time of publishing each of the contributions of this thesis. This overview is meant to complement the introduction to proof systems and succinct arguments in Section 1.2, and so we do not discuss several of the works that appear there. Whenever it is relevant, we also include a discussion on concurrent and follow-up work on the results of this thesis.

2.1 Functional Commitments

The idea of a commitment scheme where one can open to functions of the committed data was implicitly suggested by Gorbunov, Vaikuntanathan and Wichs [GVW15], although their construction is not succinct as the commitment size is linear in the length of the vector. Libert, Ramanna and Yung [LRY16] were the first to formalize *succinct* functional commitments as a generalization of vector commitments [CFM08, LY10, CF13]. They proposed a succinct FC for linear forms and showed applications of this primitive to polynomial commitments [KZG10] and accumulators. Recent works have extended FCs to support more expressive functions, including linear maps [LM19], semi-sparse polynomials [LP20], and constant-degree polynomials [ACL$^+$22, CFT22]. Catalano, Fiore and Tucker [CFT22] also proposed an FC for monotone span programs, which only achieves a weaker notion of evaluation binding where the adversary must reveal the committed vector, as for delegation schemes. A different, but also weaker security model is also considered in [PPS21], who introduced a lattice-based FC scheme where a trusted authority is assumed to generate, using a secret key, an opening key for each function for which the prover wants to release an opening.

Algebraic Delegation Schemes. As pointed out in the introduction, functional commitments for circuits imply delegation schemes (also known as SNARGs for P), and indeed there are similarities on the design of algebraic constructions of both primitives. The main difference between delegation schemes and functional commitments is that in the latter, the input is committed as opposed to publicly known. Even if the input is committed as part of the construction, in delegation schemes soundness only holds with respect to adversaries that reveal such commitment, as in the weak evaluation binding notion of [CFT22].

The pairing-based delegation schemes by González and Ràfols [GR19] and González and Zacharakis [GZ21] share some similarities with our constructions in Chapter 4 and Chapter 5. In particular, both constructions also proceed level-by-level in proving the evaluation of arithmetic circuit (an idea that dates back to the GKR protocol [GKR08]). Then, the prover (a) computes a set of commitments to the wires at each level, and (b) proves that the committed vectors are consistent with respect to the circuit evaluation. We remark that both [GR19, GZ21] require a function-specific setup and, as opposed to functional commitments, functions cannot be chosen at opening time.

Concurrent Work to Chapter 4. Concurrently to our results in Chapter 4, de Castro and Peikert [dCP23], and Wee and Wu [WW23b], also propose lattice-based constructions of functional commitments for circuits (as well as polynomial and vector commitments). Their approaches differ significantly from ours, as they both rely on homomorphic evaluation techniques [GSW13]. As a drawback of this approach, none of them support efficient verification with preprocessing.

The work of [dCP23] constructs a "dual" FC (where one commits to the function f and proves that $f(\boldsymbol{x}) = \boldsymbol{y}$ for a given $\boldsymbol{x}$)[1] for bounded-depth boolean circuits. Their construction is selectively secure under the standard SIS assumption and admits a transparent setup (i.e., the public parameters are a uniformly random string). Nevertheless, their FC does not have succinct openings, as the opening size is linear in either the input size or the size of f (in our setting where one commits to f and opens to $\boldsymbol{x}$). The FC in [WW23b] supports circuits of bounded depth, needs a structured setup, and is secure under a new structured-BASIS assumption introduced in the same work. Their FC has succinct openings that are polylogarithmic in the input size and polynomial in the circuit depth, but are not succinct in the size of the output vector.

Follow-Up Work. After the publication of our results from Chapter 4, there were significant advances around functional commitments. First, Wee and Wu introduced a lattice-based functional commitment scheme in [WW23a] which achieved efficient verification, together with other improvements such as simpler assumptions. Besides, they provide an attack against the so-called knowledge k-R-ISIS assumption [ACL+22]. This assumption was applicable to our lattice-based FC in Chapter 4, as we could use it to prove that the scheme is both an FC and a SNARK, i.e., that it also satisfies knowledge soundness.

Second, in a different work Wee and Wu introduced a fully-succinct algebraic functional commitment scheme from standard assumptions over bilinear groups [WW24b]. This scheme actually builds on the ideas of our pairing-based FC from Chapter 4, but takes them a step further by introducing a compression mechanism that achieves constant-size proofs for all circuits. This work addresses the main open question left by our results, which is whether one can build an FC with constant-size openings. The main drawback of

[1] One can recover the standard notion of committing to $\boldsymbol{x}$ and opening to f via universal evaluators.

their scheme is the size of the public parameters, which grow as $O\left(\ell_C^5\right)$ where ℓ_C is the largest supported size for a circuit.

Third, in a very recent work, Wee [Wee25] presents a functional commitment scheme from the SIS assumption over lattices which, remarkably, presents a CRS which is sublinear in the size of the inputs, and where the opening proofs grow with the circuit depth.

2.2 Batch Arguments for NP

As we discuss in the introduction, the usual BARG succinctness notion requires that the proof size is bounded by $\mathsf{poly}(\lambda, |C|, \log k)$, where k is the size of the batch. An interesting exception in the state of the art is the BARG for NP of Garg et al. [GSWW22] in which proofs have size $\mathsf{poly}(\lambda, \log|C|, \log k)$, that is sublinear also in the size of the circuit. With such a level of succinctness, the BARG of [GSWW22] clearly implies a SNARG for NP that bypasses the impossibility result of [GW11]. This is possible because the scheme, built from indistinguishability obfuscation and one-way functions, is non-adaptively sound. In contrast, in this thesis we focus on building BARGs achieving adaptive soundness, in line with most of the BARG literature.

Another approach to build BARGs from other cryptographic primitives is that of Kalai et al. who show how to build BARGs for NP from *flexible SNARGs for RAM with partial input soundness*, additionally assuming the existence of rate-1 string OT [KLVW23]. This construction provides an interesting connection between functional commitments and BARGs, in a similar spirit to our construction from Chapter 6. This is especially apparent when observing that their notion of SNARG for RAM is similar to that of functional commitments. However, on a closer look, our approaches are very different. First, as opposed to our algebraic schemes, their BARG construction is highly non-black-box due to the recursive usage of the SNARG for RAM. Thus, it cannot yield algebraic BARGs, departing from one of our main motivations. Second, they do not provide a direct construction of flexible SNARGs for RAM but show that, in turn, these can be built from BARGs (again, additionally assuming rate-1 string OT). While this result establishes an interesting connection between the two primitives, it ultimately does not yield new BARGs since the flexible SNARG for RAM should be instantiated from existing BARGs and thus from either PCPs and CI-hash, or from [WW22].

A parallel line of work aims to build increasingly expressive batch arguments to support computations closer to general NP computations, such as BARGs for monotone policies [BBK$^+$23, NWW24, NWW25].

2.3 Homomorphic Signatures

The concept of homomorphic signatures was introduced by Desmedt [Des93] and Johnson et al. [JMSW02] and properly formalized by Boneh and Freeman [BF11]. Starting from seminal works on *linearly-homomorphic* signatures, e.g., [BFKW09, GKKR10, AL11, CFW12, Fre12, LPJY13, CFGV13, CFN15], the expressivity of HS has significantly improved, capturing bounded-degree polynomials [BF11, CFW14, CFT22], and circuits of logarithmic depth [KNYY19, CFT22], and bounded polynomial depth [GVW15]. Among these works, the closest in terms of techniques to the results in this thesis is the one of Catalano, Fiore and Tucker [CFT22] who first proposed to use functional commitments to build HS. In their solution, each signer signs a commitment to the vector with m_i in position i and 0 elsewhere; the evaluator builds a commitment to the inputs using the additive homomorphic property and uses a (single-key) linearly-homomorphic signature to prove that the commitment is correctly aggregated. Indeed, are able to use this approach directly to build an homomorphic signature for arbitrary functions from functional commitments in Chapter 4. Unfortunately, generalizing this approach to the multi-key setting fails, as there exists no fully-succinct MKHS scheme, not even for linear functions.

Aggregate Signatures. The concept of aggregate signatures was introduced by Boneh et. al. [BGLS03]. Their initial construction was pairing-based and relied on random oracles. Since then, constructions have also been proposed from multilinear maps [RS09] and indistinguishability obfuscation [HKW15]. In recent years, progress on building BARGs for NP sparked multiple constructions of aggregate signatures from standard assumptions. Examples include [CJJ21, DGKV22, WW22, Goy24] for n-out-of-n policies, and [NWW23, BCJP24] for monotone policies.

Multi-Key Homomorphic Signatures. Fiore et al. [FMNP16] introduced the definition of multi-key homomorphic signatures and proposed a construction that supports circuits of bounded depth and is weakly succinct (see earlier for a detailed comparison). Lai et al. [LTWC18] proposed the first fully succinct MKHS by using SNARKs. Their construction achieves the strong notion of *unforgeability under insider corruption*, which tolerates adversaries that can even corrupt users involved in the input of a computation. Unfortunately, [LTWC18] also shows that MKHS secure in this model imply SNARGs and thus need non-falsifiable assumptions. Moreover, in this context one needs the stronger notion of *knowledge-soundness in the presence of signing oracles*, for which there are some impossibility results [FN16].

The state of the art in MKHS includes works that have investigated how to construct MKHS schemes starting from single-key HS [FP18, SFVA21], as well as constructions that aim for concrete efficiency for linear functions [AP19, SBB19]. However, with the only exception of the SNARK-based solution of [LTWC18], all these works feature signatures whose size grows linearly in the number of users.

Concurrent and Follow-up Work. The results in Chapter 4 imply an homomorphic signature for arbitrary circuits of unbounded depth. Shortly after, Gay and Ursu presented a different construction based on indistinguishability obfuscation [GU24]. Separately, Goyal [Goy24] introduced the notion of mutable BARGs, which can be used to realize homomorphic signatures. These signatures can be composed a constant number of times.

Concurrently to our results in Chapter 7 for multi-key homomorphic signatures, Afshar, Cheng and Goyal [ACG24] improved the result from [Goy24] to obtain multi-hop HS that can be composed a polynomial number of times. The construction is highly non-black-box and relies on composition of rate-1 BARGs. Afshar, Cheng and Goyal later extended their results to realize composable multi-key HS.

2.4 Efficient Proof Systems

The first construction of a SNARG for NP by Kilian [Kil92] is essentially a cryptographic compilation of probabilistic checkable proofs and an interactive protocol. This construction was originally far from practical, but its spirit is still present in many SNARGs which do aim for concrete efficiency. We recall the general blueprint of Kilian's SNARG and many other constructions, which consists of three steps.

- **Parameter setup.** The first step is to establish a set of public parameters that are available to both prover and verifier. Depending on the proof system, this may include the specification of a hash function, common randomness, or a so-called common reference string (CRS), which is a string that includes some correlated information [Dam00]. For the latter, the CRS may include a trapdoor that would allow anyone to generate simulated proofs for any statement.[2] Hence, the CRS has to be generated either by a trusted party or by a distributed protocol ran among multiple parties [BCG$^+$15, NRBB22].
- **Commitment.** Due to the succinctness requirement, it is not possible to send the entire NP witness to the verifier, but the prover still needs to encode it in some succinct representation to be able to prove something about it. The encoding is realised via a succinct *commitment scheme* with *fine-grained openings*. The most popular are *vector commitments* [Mer79, CF13], for which one can open a specific position x_i of a vector $\boldsymbol{x}$, and polynomial commitments [KZG10], for which one can commit to a polynomial p and open its evaluation $p(z)$ at a specific point z.
- **Proof.** The final step is to succinctly prove the satisfiability of C for the statement $\mathtt{x}$ and committed witness $\mathtt{w}$. There are a variety of recipes for this step, but generally they involve: (a) an information-theoretic component such as a probabilistic checkable proof (PCP) [BFL90, AS92], which is compressed using a commitment scheme, (b) a public-coin interactive protocol to obtain random verifier's challenges which is made non-interactive via the Fiat-Shamir transform [FS87], or (c) a combination of both, such as in Kilian's original scheme [Kil92]. We remark that for some constructions such as those based on linear PCPs [BCI$^+$13], the commitment step is actually implicit in the proof and not a well-differentiated component.

For example, in Kilian's SNARG a PCP proof is committed with a Merkle tree [Mer79] (using it as a vector commitment scheme, such that one can open the commitment to

[2] One common example of this is to include correlated group elements. For instance, in a prime order group setting $\mathbb{G}$, a usual CRS may include elements $g, h_1, h_2, \ldots, h_n \in \mathbb{G}$ such that $h_i = g^{a^i}$ for some $a \in \mathbb{Z}_p$. Here, a is the CRS trapdoor.

specific positions), and then obtains the verifier challenges via the Fiat-Shamir transform. Naturally, the security of the scheme relies on the random oracle model [BR93]. Nowadays, there exist many valid recipes that offer different efficiency/security/succinctness trade-offs. Some notable approaches are SNARGs based on linear PCPs and quadratic span programs [GGPR13, PHGR13, Lip13, DFGK14, Gro16, GNS23], SNARGs based on (polynomial) interactive oracle proofs (IOPs) [BCR+19, CHM+20, MBKM19, GWC19, CFF+21], and SNARGs based on folding arguments [BGH19, KST22, KP23], which have recently led to efficient constructions of lattice-based SNARGs [BS23, BC24, FKNP24, KLNO24, CB25]. Of particular interest to us is the approach of building SNARGs from sumcheck-based interactive proofs, which we introduce below.

Sumcheck-based proofs. Sumcheck-based proofs are a family of cryptographic proof systems that rely on the sumcheck protocol [LFKN92], an interactive protocol for proving that the sum of all values of a polynomial over a certain space (such as the boolean hypercube) equals a given constant. The seminal paper of Goldwasser, Kalai and Rothblum [GKR08] showed how to use the sumcheck protocol to construct a doubly-efficient interactive proof (known as GKR in the literature) for layered arithmetic circuits. Several papers improved the proving time of GKR either in general [CMT12a], for circuits with specific structure [Tha13, WJB+17, ZGK+18] or through variants of the original protocol [XZZ+19, ZLW+21]. Thaler was the first to show sumcheck-based protocols for specialized computations, such as matrix multiplications, with optimal prover time [Tha13]. Another line of work, started by Zhang et al. [ZGK+17], showed how to use GKR in combination with polynomial commitments to build (zero-knowledge) SNARGs [WTs+18, XZZ+19, Set20, ZXZS20]. Arguments based on this approach, and especially those where sumcheck is applied on multilinear polynomials, are among the most efficient ones for proving time. Indeed, most of their computational effort relates to an information-theoretic-secure protocol involving only a linear number (in the size of the computation) finite field operations. Recent works show how to combine the sumcheck protocol with multilinear polynomial commitments to build succinct non-interactive arguments [GLS+23, XZS22, CBBZ23].

In Chapter 8, we present a modular composition framework that is close in the spirit to that of Campanelli, Fiore and Querol [CFQ19] who build zk-SNARKs modularly via the efficient composition of specialized commit-and-prove SNARKs. Our techniques work at the information-theoretic level and are based on polynomial evaluations as opposed to commitments and SNARKs, allowing for a less demanding security notion than computational binding and thus for improved efficiency.

2.4.1 Proofs for Machine Learning and Image Processing

Machine Learning. For applications to verifiable Machine Learning, the closest work to our contributions in Chapter 8 is zkCNN [LXZ21] which shows how to exploit the sequential nature of neural networks to build a tailored argument system. vCNN [LKKO20] and ZEN [FQZ+21] also tackle the problem of zero-knowledge neural network predictions.

vCNN combines different commit-and-prove SNARKs to efficiently prove the CNN layers, notably they use quadratic polynomial programs for convolution layers and quadratic arithmetic programs for ReLU and Pooling layers. ZEN presents a quantisation mechanism (based on [JKC+18]) for R1CS-based proof systems that achieves significantly less constraints and hence a faster proving time and smaller public parameters. Although we do not directly compare our contributions to vCNN and ZEN in Chapter 8, we observe that [LXZ21] shows that zkCNN is orders of magnitude faster than vCNN and ZEN, and thus by improving over zkCNN our work also improves on the former.

After our contributions in Chapter 8 were published, a series of works provided further improvements on verifiable machine learning. These include works for verifiable decision trees [CFF+24, PP24], for proofs of neural network training [GGJ+23, APKP24], and for inference of large language models [SLZ24].

Image Processing. Besides solutions based on general-purpose zkSNARKs, there are a few works that build specialized proof systems for image processing transformations, notably PhotoProof [NT16], ZK-IMG [KHSS22], and VILS [CHN+22]. PhotoProof [NT16] presents an image authentication framework where images are output by "secure" cameras (i.e., cameras capable of signing images) and Proof-Carrying Data [CT10] is used to define a set of admissible transformations. The PhotoProof prototype is based on `libsnark` [sci17] and experiments show that proving one transformation of a 128×128 image takes more than 300 seconds and a public key of a few GBs. ZK-IMG [KHSS22] improves over PhotoProof by using `halo2` [ZCa22] as the underlying ZK-SNARK system and by showing how to chain proofs of sequential transformations without revealing the intermediate outputs—a feature that may be desirable in scenarios where the input image is private. Performance reported in [KHSS22] show that convolution operations can take more than 80 seconds to generate a proof for images of 1280×720 pixels. Finally, VILS [CHN+22] takes an alternative approach to authenticated image editing by computing all possible image transformation at the source (i.e., by the secure camera) and accumulating them in a cryptographic accumulator.

Following up the publication of our results, several works introduced improvements and novel approaches to verifiable image processing [DEH25, MVVZ24, DCB25].

3 Background

3.1 Notation

The definitions, games, and constructions in this thesis use standard notation. Algorithms, oracle names, and cryptographic parameters are denoted in $\mathsf{sans\text{-}serif}$ font.

Functions and algorithms. We denote the security parameter by $\lambda \in \mathbb{N}$, and its unary representation by 1^λ. The security parameter measures the level of security of cryptographic schemes and indicates the amount of computational resources required to break the scheme: for a given security parameter λ, one needs $\Theta\left(2^\lambda\right)$ operations. We call a function ϵ *negligible*, denoted $\epsilon(\lambda) = \mathsf{negl}(\lambda)$, if $\epsilon(\lambda) = O(\lambda^{-c})$ for every constant $c > 0$, and call a function $p(\lambda)$ *polynomial*, denoted $p(\lambda) = \mathsf{poly}(\lambda)$, if $p(\lambda) = O(\lambda^c)$ for some constant $c > 0$. We say that an algorithm is *probabilistic polynomial time* (PPT) if it consumes randomness and its running time is bounded by some $p(\lambda) = \mathsf{poly}(\lambda)$. For a finite set S, $x \leftarrow\!\!\$\ S$ denotes sampling x uniformly at random in S. For an algorithm A, we write $y \leftarrow A(x)$ for the output of A on input x. Sometimes, to remark that an algorithm is randomized, we may write $y \leftarrow\!\!\$\ A(x)$. An algorithm can input or return blank values, represented by $\perp$. In interactive algorithms, we $\underline{\text{underline}}$ steps that involve interaction, such as $\underline{\text{Send}}$ or $\underline{\text{Get}}$.

Sets, vectors, strings. For a positive $n \in \mathbb{N}$, $[n]$ is the set $\{1, \ldots, n\}$. We denote vectors $\boldsymbol{x}$ and matrices $\mathbf{M}$ using bold fonts. For a ring $\mathcal{R}$, given two vectors $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{R}^n$, $\boldsymbol{z} := (\boldsymbol{x} \otimes \boldsymbol{y}) \in \mathcal{R}^{n^2}$ denotes their Kronecker product (that is a vectorization of the outer product), i.e., $\forall i, j \in [n] : z_{i+(j-1)n} = x_i y_j$. Given two strings x, y, we denote their concatenation by $x|y$. We write $\mathbf{0}_k$ and $\mathbf{1}_k$ for the vectors of k zeros and ones respectively. We use $\mathcal{M}$ to denote the message space, and sometimes $\bar{\mathcal{M}} \subseteq \mathcal{M}$ for some message subspace, for all cryptographic primitives considered. We denote the message space of dimension ℓ by $\mathcal{M}^\ell$. To denote a (sub)vector with coordinates in a position subset $J \subseteq [\ell]$, we write $\boldsymbol{x}_J \in \mathcal{M}^J$.

3.2 Bilinear Groups

A *bilinear group generator* $\mathcal{BG}(1^\lambda)$ is an algorithm that returns $\mathsf{bgp} := (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2)$, where $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$ are groups of prime order q, $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$ are fixed generators, and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is an efficiently computable, non-degenerate, bilinear map. In our work we use Type-3 groups in which it is assumed that there is no efficiently computable isomorphism between $\mathbb{G}_1$ and $\mathbb{G}_2$. We use the bracket notation of [EHK$^+$13] for group elements: for $s \in \{1, 2, T\}$ and $x \in \mathbb{Z}_q$, $[x]_s$ denotes $g_s^x \in \mathbb{G}_s$. We use additive notation for $\mathbb{G}_1$ and $\mathbb{G}_2$ and multiplicative notation for $\mathbb{G}_T$. We note that given an element $[x]_s \in \mathbb{G}_s$, for $s = 1, 2$, and a scalar a, one can efficiently compute $a \cdot [x] = [ax] = g_s^{ax} \in \mathbb{G}_s$; given group elements $[a]_1 \in \mathbb{G}_1$ and $[b]_2 \in \mathbb{G}_2$, one can efficiently compute $[ab]_T = [a]_1 \cdot [b]_2$. For a matrix $\mathbf{A} \in \mathbb{Z}_q^{m\times n}$, we represent a matrix of group elements $g_s^{\mathbf{A}}$ as $[\mathbf{A}]_s \in \mathbb{G}_s^{m\times n}$.

3.3 Lattices

3.3.1 Lattice Preliminaries

Let $\mathcal{R} = \mathbb{Z}[\zeta]$, where ζ is a fixed primitive m-th root of unity, be the ring of integers of the m-th cyclotomic field of degree $d = \varphi(m)$, where elements are represented by their coefficient embedding $x = \sum_{i=0}^{d-1} x_i \cdot \zeta^i$. If m is a prime-power (resp. power of 2), we call $\mathcal{R}$ a prime-power (resp. power-of-two) cyclotomic ring. For the rest of this section we will assume that $m = \mathsf{poly}(\lambda)$.

For $x \in \mathcal{R}$, write $\|x\| := \max_{i=0}^{d-1} |x_i|$ for the infinity norm induced on $\mathcal{R}$ by $\mathbb{Z}$. The norm generalises naturally to vectors $\boldsymbol{u} = (u_1, \ldots, u_n) \in \mathcal{R}^n$, with $\|\boldsymbol{u}\| := \max_{i=1}^{n} \|u_i\|$. For $q \in \mathbb{N}$, write $\mathcal{R}_q := \mathcal{R}/q\mathcal{R}$. We always assume that q is a (rational) prime. By a slight abuse of notation, we identity $\mathcal{R}_q$ with its balanced representation, i.e. if $x = \sum_{i=0}^{d-1} x_i \cdot \zeta^i \in \mathcal{R}_q$ then $|x_i| \le q/2$ for all i. The set of units, i.e., invertible elements, in $\mathcal{R}_q$ is denoted by $\mathcal{R}_q^\times$.

The ring expansion factor $\gamma_\mathcal{R}$ of $\mathcal{R}$ is defined as $\gamma_\mathcal{R} := \max_{a,b\in\mathcal{R}} \frac{\|a\cdot b\|}{\|a\|\cdot\|b\|}$. It is known [AL21] that if $\mathcal{R}$ is a prime-power cyclotomic ring then $\gamma_\mathcal{R} \le 2 \cdot d$, and if $\mathcal{R}$ is a power-of-two cyclotomic ring then $\gamma_\mathcal{R} \le d$.

3.3.2 Lattice Trapdoors

We recall the following standard algorithms (e.g., [GPV08, MP12, GM18]) associated to lattice trapdoors and their properties for sufficiently large "leftover hash lemma parameter" $\mathsf{lhl}(\mathcal{R}, \eta, q, \beta) = O(\eta \log_\beta q)$:

- $(\mathbf{A}, \mathsf{td}_\mathbf{A}) \leftarrow \mathsf{TrapGen}(\mathcal{R}, 1^\eta, 1^\zeta, q, \beta)$: The trapdoor generation algorithm generates a matrix $\mathbf{A} \in \mathcal{R}_q^{\eta\times\zeta}$ along with a trapdoor $\mathsf{td}_\mathbf{A}$. It is assumed that (η, ζ, q, β) are implicitly specified by $\mathsf{td}_\mathbf{A}$. When $\zeta \ge \mathsf{lhl}(\mathcal{R}, \eta, q, \beta)$, the distribution of $\mathbf{A}$ is within $\mathsf{negl}(\lambda)$ statistical distance of $U(\mathcal{R}_q^{\eta\times\zeta})$.
- $\boldsymbol{u} \leftarrow \mathsf{SampD}(\mathcal{R}, 1^\eta, 1^\zeta, q, \beta')$: The domain sampling algorithm samples a vector $\boldsymbol{u} \in \mathcal{R}^\zeta$ with norm $\|\boldsymbol{u}\| \le \beta'$. When $\beta' \ge \beta$ and $\zeta \ge \mathsf{lhl}(\mathcal{R}, \eta, q, \beta)$, then the distribution of $(\mathbf{A}, \mathbf{A} \cdot \boldsymbol{u} \bmod q)$ for a uniformly random $\mathbf{A} \leftarrow\!\!\$\, \mathcal{R}_q^{\eta\times\zeta}$ is within $\mathsf{negl}(\lambda)$ statistical distance of $U(\mathcal{R}_q^{\eta\times\zeta} \times \mathcal{R}_q^\eta)$.
- $\boldsymbol{u} \leftarrow \mathsf{SampPre}(\mathsf{td}_\mathbf{A}, \boldsymbol{v}, \beta')$: The preimage sampling algorithm inputs a vector $\boldsymbol{v} \in \mathcal{R}_q^n$ and outputs a vector $\boldsymbol{u} \in \mathcal{R}^\zeta$. If the parameters (η, ζ, q, β) of $\mathsf{td}_\mathbf{A}$ satisfy $\beta' \ge \beta$ and

$\zeta \geq \mathsf{lhl}(\mathcal{R}, \eta, q, \beta)$, then $\boldsymbol{u}$ and $\boldsymbol{v}$ satisfy $\mathbf{A} \cdot \boldsymbol{u} = \boldsymbol{v} \bmod q$ and $\|\boldsymbol{u}\| \leq \beta'$. Furthermore, $\boldsymbol{u}$ is within $\mathsf{negl}(\lambda)$ statistical distance to $\boldsymbol{u} \leftarrow \mathsf{SampD}(\mathcal{R}, 1^n, 1^\zeta, q, \beta')$ conditioned on $\mathbf{A} \cdot \boldsymbol{u} = \boldsymbol{v} \bmod q$.

3.4 Commitment Schemes and Advanced Properties

Commitment schemes allow a party to succinctly commit to a value or vector of values while keeping them hidden, and later reveal them along with an opening proof. Commitments are ubiquitous in the design of advanced cryptographic primitives and protocols, and so are they in this thesis.

Next, we introduce a general definition of commitment schemes for vectors that considers both deterministic and randomized commitments, as well as the possibility to commit to subvectors explicitly. For the latter, we allow the commitment algorithm to additionally input an index set $J \in \mathcal{J} \subseteq 2^{[\ell]}$ where ℓ is the maximum vector length, and commit to a (sub)vector $\boldsymbol{x} \in \mathcal{M}^J$, and extend the other algorithms accordingly. Although not often considered explicitly in definitions, such a functionality is naturally supported by common commitment schemes.

Definition 3.1 (Commitment). *A commitment scheme for an index set family $\mathcal{J} \subseteq 2^{[\ell]}$ is a tuple of PPT algorithms* $(\mathsf{Setup}, \mathsf{Com}, \mathsf{Open}, \mathsf{Ver})$ *with the following syntax:*

$\mathsf{Setup}(1^\lambda, 1^\ell) \longrightarrow \mathsf{ck}$**:** *On input the security parameter λ and the vector length ℓ, output a commitment key* ck.

$\mathsf{Com}(\mathsf{ck}, J, \boldsymbol{x}; r) \longrightarrow (\mathsf{com}, \mathsf{aux})$**:** *On input the commitment key* ck*, an index set $J \in \mathcal{J}$, a vector $\boldsymbol{x} \in \mathcal{M}^J$, and input randomness r, output a commitment* com *and auxiliary information* aux.[1]

$\mathsf{Open}(\mathsf{ck}, \mathsf{aux}) \longrightarrow \mathsf{open}$**:** *On input the commitment key* ck *and auxiliary information* aux*, outputs an opening* open.[2]

$\mathsf{Ver}(\mathsf{ck}, \mathsf{com}, J, \boldsymbol{x}, \mathsf{open}) \longrightarrow b$**:** *On input the commitment key* ck*, a commitment* com*, an index set $J \in \mathcal{J}$, a vector $\boldsymbol{x} \in \mathcal{M}^J$ and an opening* open*, output accept ($b = 1$) or reject ($b = 0$).*

The index set J is taken as $J = [\ell]$ when omitted. A commitment scheme must satisfy the following properties:

Correctness. *For every $\lambda, \ell \in \mathbb{N}$, any $J \in \mathcal{J}$ and any input $\boldsymbol{x} \in \mathcal{M}^J$,*

$$\Pr\left[\mathsf{Ver}(\mathsf{ck}, \mathsf{com}, J, \boldsymbol{x}, \mathsf{open}) = 1 \;\middle|\; \begin{array}{l} \mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell) \\ (\mathsf{com}, \mathsf{aux}) \leftarrow \mathsf{Com}(\mathsf{ck}, J, \boldsymbol{x}) \quad \mathsf{open} \leftarrow \mathsf{Open}(\mathsf{ck}, \mathsf{aux}) \end{array}\right] = 1.$$

[1] In our constructions, we often omit r from the inputs; in such a case we assume either that r is randomly sampled or that the commitment algorithm is deterministic.

[2] In the literature, it is usual to consider definitions of commitments where the opening is output by the commitment algorithm (instead of the auxiliary information).

Computational binding. *For any PPT adversary $\mathcal{A}$ and vector length $\ell = \mathsf{poly}(\lambda)$,*

$$\Pr\left[\begin{array}{c|l} \mathsf{Ver}(\mathsf{ck}, \mathsf{com}, J, \boldsymbol{x}, \mathsf{open}) = 1 & \mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell) \\ \wedge\ \mathsf{Ver}(\mathsf{ck}, \mathsf{com}, J', \boldsymbol{x}', \mathsf{open}') = 1 & (\mathsf{com}, J, \boldsymbol{x}, J', \boldsymbol{x}', \mathsf{open}, \mathsf{open}') \leftarrow \mathcal{A}(\mathsf{ck}) \\ & J, J' \in \mathcal{J} \wedge \boldsymbol{x}, \boldsymbol{x}' \in \mathcal{M}^J \end{array}\right] \leq \mathsf{negl}(\lambda).$$

Succinctness. *For any $\lambda, \ell \in \mathbb{N}$, $\boldsymbol{x} \in \mathcal{M}^\ell$, it holds that $|\mathsf{com}| \leq \mathsf{poly}(\lambda, \log \ell)$.*

Randomized commitments may also satisfy the following property:

Statistical Hiding. *For any $1^\lambda, \ell \in \mathbb{N}$, $\mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell)$, any $J, J' \in \mathcal{J}$ and any pair of inputs $\boldsymbol{x} \in \mathcal{M}^J$ and $\boldsymbol{x}' \in \mathcal{M}^{J'}$, the following distributions are $\mathsf{negl}(\lambda)$ statistically close:*

$$\{\mathsf{com} : (\mathsf{com}, \mathsf{aux}) \leftarrow \mathsf{Com}(\mathsf{ck}, J, \boldsymbol{x})\} \approx$$
$$\{\mathsf{com}' : (\mathsf{com}', \mathsf{aux}') \leftarrow \mathsf{Com}(\mathsf{ck}, J', \boldsymbol{x}')\}.$$

The scheme is perfectly *hiding if both distributions are identical.*

Remark 3.2. *Throughout this thesis, we often consider* deterministic *commitments. In this case, we assume that the opening information is the committed vector itself. We also do not consider a verification algorithm explicitly as the verifier can simply recompute the deterministic commitment given the input vector. Therefore, we simply introduce them as* $(\mathsf{Setup}, \mathsf{Com})$.

Remark 3.3. *We assume that all algorithms taking a commitment key* ck *as input have random access to* ck. *This is to capture schemes where the commitment key consists of many components and where not all algorithms need access to all components.*

3.4.1 Additive Homomorphism

A commitment scheme is additively homomorphic if given two commitments com_1 and com_2 to vectors $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ respectively, there exists an efficient algorithm to compute a commitment to $\boldsymbol{x}_1 + \boldsymbol{x}_2$. We follow the description of [CFT22].

Definition 3.4 (Additive-homomorphic commitments)**.** *Let* Com *be a commitment scheme where the message space $\mathcal{M}$ is a ring.* Com *is additive homomorphic if there exist deterministic algorithms:*

- $\mathsf{Com.Add}(\mathsf{ck}, \mathsf{com}_1, \ldots, \mathsf{com}_n) \rightarrow \mathsf{com}$,
- $\mathsf{Com.Add}_{\mathsf{aux}}(\mathsf{ck}, \mathsf{aux}_1, \ldots, \mathsf{aux}_n) \rightarrow \mathsf{aux}$, *and*
- $\mathsf{Com.Add}_r(\mathsf{ck}, r_1, \ldots, r_n) \rightarrow r$

such that for any $\boldsymbol{x}_i \in \mathcal{M}$ and $(\mathsf{com}_i, \mathsf{aux}_i) \leftarrow \mathsf{Com}(\mathsf{ck}, \boldsymbol{x}_i; r_i)$, if $\mathsf{com} \leftarrow \mathsf{Com.Add}(\mathsf{ck}, \mathsf{com}_1, \ldots, \mathsf{com}_n)$, $\mathsf{aux} \leftarrow \mathsf{Com.Add}_{\mathsf{aux}}(\mathsf{ck}, \mathsf{aux}_1, \ldots, \mathsf{aux}_n)$, and $r \leftarrow \mathsf{Com.Add}_r(\mathsf{ck}, r_1, \ldots, r_n)$, then $(\mathsf{com}, \mathsf{aux}) = \mathsf{Com}(\mathsf{ck}, \sum_{i=1}^n \boldsymbol{x}_i; r)$.

3.4.2 Aggregatability

For deterministic commitments, we define a notion of aggregatability which allows to aggregate commitments of subvectors with disjoint index sets into a commitment of the union. Naturally, any additive-homomorphic commitment scheme is aggregatable, as one can aggregate commitments to subvectors by summing them up.

Definition 3.5 (Aggregatable commitments). *Let $\mathcal{J} \subseteq 2^{[\ell]}$ be an index set family and $\hat{\mathcal{J}} \subset \mathcal{J}$. A commitment scheme is aggregatable if*

- *there exist a deterministic aggregation algorithm* $\mathsf{Agg}(\mathsf{ck}, (\mathsf{com}_J)_{J\in\hat{\mathcal{J}}}) \to \mathsf{com}$, *which takes as input the commitment key* ck *and tuples of commitments* $(\mathsf{com}_J)_{J\in\hat{\mathcal{J}}}$ *and outputs a commitment* com *in time at most* $\mathsf{poly}(\lambda, \log \ell) \cdot |\hat{\mathcal{J}}|$, *and*
- *if $J \cap J' = \emptyset$ for all distinct $J, J' \in \hat{\mathcal{J}}$, then for any $\boldsymbol{x}_J \in \mathcal{M}^J$ for $J \in \hat{\mathcal{J}}$, the following property is satisfied:*

$$\Pr\left[\mathsf{Agg}(\mathsf{ck}, (\mathsf{com}_J)_{J\in\hat{\mathcal{J}}}) = \mathsf{com} \;\middle|\; \begin{array}{l} \mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell) \\ \mathsf{com}_J \leftarrow \mathsf{Com}(\mathsf{ck}, J, \boldsymbol{x}_J) \;\; \forall J \in \hat{\mathcal{J}} \\ \mathsf{com} \leftarrow \mathsf{Com}\left(\mathsf{ck}, \bigcup_{J\in\hat{\mathcal{J}}} J, \bigcup_{J\in\hat{\mathcal{J}}} \boldsymbol{x}_J\right) \end{array}\right] = 1.$$

The definition can be relaxed, but made more complicated, to allow for probabilistic commitments. We omit it as we do not use this property in this thesis.

3.4.3 Local Updatability

We introduce a notion of *local updatability* for deterministic commitments that is central to the results of Chapter 7. A commitment scheme supports local updatability if one can update a commitment com at a position $i \in [\ell]$ (or more generally, at a set of positions $S \subseteq [\ell]$) in a succinct way. Namely, the update must be verifiable in time $O(\lambda, \log \ell, |S|)$.

Local update soundness is defined such that given an honestly generated commitment com to $\boldsymbol{x}$, and a set of updates $\{x'_i\}_{i\in S}$ that update $\boldsymbol{x}$ to $\boldsymbol{x}'$, it must be hard to forge a valid update from com to any com′ such that com′ does not commit to $\boldsymbol{x}'$. For this property, we enforce that commitments com are deterministic.

Definition 3.6 (Local updatability). *A deterministic commitment scheme* Com *is locally updatable if there exists a pair of algorithms* $(\mathsf{Upd}, \mathsf{VerUpd})$ *as follows:*

$\mathsf{Com.Upd}(\mathsf{ck}, \mathsf{aux}, S, \{x'_i\}_{i\in S}) \to (\mathsf{com}', \mathsf{aux}', \pi)$ *on input the commitment key* ck, *auxiliary information*[3] aux, *a set of positions $S \subseteq [\ell]$, and updates $\{x'_i\}_{i\in S}$, outputs an updated deterministic commitment* com', *updated auxiliary input* aux', *and an update proof π.*

$\mathsf{Com.VerUpd}(\mathsf{ck}_S, S, \mathsf{com}, \{x_i\}_{i\in S}, \mathsf{com}', \{x'_i\}_{i\in S}, \pi) \to 0/1$ *on input a section of the commitment key* ck_S, *a commitment* com, *a set of positions $S \subseteq [\ell]$, inputs $\{x_i\}_{i\in S}$, updates $\{x'_i\}_{i\in S}$, an updated deterministic commitment* com' *and an update proof π, accepts (outputs 1) or rejects (outputs 0).*

[3] We note that in some algebraic schemes, only the section of aux corresponding to the set S may be needed.

Let $\boldsymbol{x}' \leftarrow \mathtt{Up}(\boldsymbol{x}, \{x'_i\}_{i\in S})$ be a function that updates $\boldsymbol{x}$ to $\boldsymbol{x}'$, i.e., outputs a vector $\boldsymbol{x}'$ that contains x_i at every coordinate $i \notin S$, and x'_i at every $i \in S$. Then, these algorithms must satisfy the following properties.

Correctness. *For any $\ell \in \mathbb{N}$, any $f : \mathcal{M}^\ell \to \mathcal{M}^m$ in the class $\mathcal{F}$, any $\boldsymbol{x} \in \mathcal{M}^\ell$, any set $S \subseteq [\ell]$, and any set $\{x'_i\}_{i\in S}$ such that $x'_i \in \mathcal{M}\ \forall i \in S$, we have:*

$$\Pr\left[\begin{array}{l} \mathsf{VerUpd}(\mathsf{ck}_S, S, \mathsf{com}, \{x_i\}, \mathsf{com}', \{x'_i\}, \pi) = 1 \\ \wedge\ (\mathsf{com}', \mathsf{aux}') = \mathsf{Com}(\mathsf{ck}, \boldsymbol{x}') \end{array} : \begin{array}{r} \mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell) \\ (\mathsf{com}, \mathsf{aux}) \leftarrow \mathsf{Com}(\mathsf{ck}, \boldsymbol{x}) \\ \boldsymbol{x}' \leftarrow \mathtt{Up}(\boldsymbol{x}, \{x'_i\}_{i\in S}) \\ (\mathsf{com}', \mathsf{aux}', \pi) \leftarrow \\ \mathsf{Com.Upd}(\mathsf{ck}, \mathsf{aux}, S, \{x'_i\}) \end{array}\right] = 1.$$

Soundness. *For any PPT adversary $\mathcal{A}$, the following probability is $\mathsf{negl}(\lambda)$:*

$$\Pr\left[\begin{array}{l} \mathsf{VerUpd}(\mathsf{ck}_S, S, \mathsf{com}, \{x_i\}, \mathsf{com}', \{x'_i\}, \pi) = 1 \\ \wedge\ \mathsf{com}' \neq \mathsf{Com}(\mathsf{ck}, \boldsymbol{x}') \end{array} : \begin{array}{r} \mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell) \\ (\mathsf{com}, \mathsf{aux}) \leftarrow \mathsf{Com}(\mathsf{ck}, \boldsymbol{x}) \\ (S, \{x'_i\}, \mathsf{com}', \pi) \leftarrow \mathcal{A}(\mathsf{ck}, \mathsf{com}, \mathsf{aux}) \\ \boldsymbol{x}' \leftarrow \mathtt{Up}(\boldsymbol{x}, \{x'_i\}_{i\in S}) \end{array}\right].$$

Succinctness. *For any admissible parameters, the update proof $|\pi| \leq \mathsf{poly}(\lambda, \log \ell) \cdot O(|S|)$. Besides, $\mathsf{Com.VerUpd}$ runs in time bounded by $\mathsf{poly}(\lambda, \log \ell) \cdot O(|S|)$.*

Most commitment constructions in the literature do not explicitly state a local updatability property, even though many present it naturally. One such way to achieve local updatability, such as in the schemes in [CFT22, WW24b] and in Chapter 4 and Chapter 5, is via additive homomorphism.

3.5 Proof Systems

Next, we introduce a standard definition of arguments of knowledge, both for the interactive and non-interactive variants. We denote $R : \mathcal{X} \times \mathcal{W} \to \{0, 1\}$ to be a binary relation defined over instances $\mathcal{X}$ and witnesses $\mathcal{W}$. We say that $\mathtt{x} \in \mathcal{X}, \mathtt{w} \in \mathcal{Y}$ are a valid statement-witness pair whenever $R(\mathtt{x}, \mathtt{w}) = 1$.

Definition 3.7 (Interactive Argument of Knowledge)**.** *An interactive argument of knowledge* AoK *for an NP relation $\mathcal{R}$ is a tuple of algorithms* $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Ver})$ *such that:*

$\mathsf{Setup}(1^\lambda, \mathcal{R}) \to \mathsf{crs}$ *outputs a common reference string* crs.

$\langle \mathsf{Prove}(\mathtt{w}), \mathsf{Ver}\rangle(\mathsf{crs}, \mathtt{x}) \to (\pi, b)$ *are a pair of interactive algorithms that on input the common reference string* crs*, the statement* $\mathtt{x}$ *and (only for* Prove*) a witness* $\mathtt{w}$*, output a proof transcript π and an acceptance bit b.*

Besides, an interactive argument of knowledge satisfies the following two properties:

Completeness. AoK *is complete if for any* $\lambda \in \mathbb{N}$ *and* $(\mathtt{x}, \mathtt{w}) \in \mathcal{R}$ *it holds* $\Pr[\langle \mathsf{Prove}(\mathtt{w}), \mathsf{Ver}\rangle(\mathsf{crs}, \mathtt{x}) \longrightarrow 1] = 1$ *where* $\mathsf{crs} \longleftarrow \mathsf{Setup}(1^\lambda, \mathcal{R})$.

Knowledge-soundness. *For any PPT adversary $\mathcal{A}$ there exists a polynomial-time extractor $\mathcal{E}$ such that*

$$\Pr\left[\begin{array}{c|l} \langle \mathcal{A}, \mathsf{Ver}\rangle(\mathsf{crs}, \mathtt{x}) \longrightarrow 1 & \mathsf{crs} \longleftarrow \mathsf{Setup}(1^\lambda, \mathcal{R}) \\ \wedge (\mathtt{x}, \mathtt{w}) \notin \mathcal{R} & \mathtt{x} \longleftarrow \mathcal{A}(\mathsf{crs}) \\ & \mathtt{w} \longleftarrow \mathcal{E}^{\mathcal{A}}() \end{array}\right] = \mathsf{negl}(\lambda).$$

Definition 3.8 (Non-Interactive Argument of Knowledge). *A non-interactive argument of knowledge* NARK *for an NP relation $\mathcal{R}$ is a tuple of algorithms* $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Ver})$ *such that:*

$\mathsf{Setup}(1^\lambda, \mathcal{R}) \longrightarrow \mathsf{crs}$ *outputs a common reference string* crs.

$\mathsf{Prove}(\mathsf{crs}, \mathtt{x}, \mathtt{w}) \longrightarrow \pi$ *on input* crs, *a statement* $\mathtt{x}$ *and a witness* $\mathtt{w}$ *such that* $(\mathtt{x}, \mathtt{w}) \in \mathcal{R}$, *it returns a proof* π.

$\mathsf{Ver}(\mathsf{crs}, \mathtt{x}, \pi) \longrightarrow b$ *given* crs, *a statement* $\mathtt{x}$ *and a proof* π, *it outputs 1 (accept) or 0 (reject).*

Besides, a non-interactive argument of knowledge satisfies the following two properties:

Completeness. NARK *is complete if for any* $\lambda \in \mathbb{N}$ *and* $(\mathtt{x}, \mathtt{w}) \in \mathcal{R}$ *it holds* $\Pr[\mathsf{Ver}(\mathsf{crs}, \mathtt{x}, \mathsf{Prove}(\mathsf{crs}, \mathtt{x}, \mathtt{w})) = 1] = 1$ *where* $\mathsf{crs} \longleftarrow \mathsf{Setup}(1^\lambda, \mathcal{R})$.

Knowledge-soundness. *For any PT adversary $\mathcal{A}$ there exists an extractor $\mathcal{E}$ (taking the same input of $\mathcal{A}$ including the random tape ρ) such that*

$$\Pr\left[\begin{array}{c|l} \mathsf{Ver}(\mathsf{crs}, \mathtt{x}, \pi) = 1 & \mathsf{crs} \longleftarrow \mathsf{Setup}(1^\lambda, \mathcal{R}) \\ \wedge (\mathtt{x}, \mathtt{w}) \notin \mathcal{R} & (\mathtt{x}, \pi) \longleftarrow \mathcal{A}(\mathsf{crs}; \rho) \\ & \mathtt{w} \longleftarrow \mathcal{E}(\mathsf{crs}; \rho) \end{array}\right] = \mathsf{negl}(\lambda).$$

We briefly introduce two notions that are relevant for arguments of knowledge: zero-knowledge and commit-and-prove arguments. For simplicity, we introduce them for non-interactive arguments of knowledge. For an extensive treatment, we refer to [CFQ19].

Definition 3.9 (Zero-knowledge NARK). *A non-interactive argument of knowledge* NARK *is computationally (resp. statistically, perfectly) zero-knowledge if there exists a simulator* $\mathsf{Sim} = (\mathsf{Sim}_0, \mathsf{Sim}_1)$ *such that:*

(a) $\mathsf{Sim}_0(1^\lambda, \mathcal{R}) \longrightarrow (\mathsf{crs}, \mathsf{td})$ *generates a* crs *that is computationally (resp. statistically, perfectly) indistinguishable from that generated by* Setup, *and*

(b) for any $(\mathtt{x}, \mathtt{w}) \in \mathcal{R}$, *and* $(\mathsf{crs}, \mathsf{td}) \longleftarrow \mathsf{Sim}_0(1^\lambda, \mathcal{R})$, $\mathsf{Sim}_1(\mathsf{td}, \mathtt{x})$ *generates proofs that are computationally (resp. statistically, perfectly) indistinguishable from those generated by* $\mathsf{Prove}(\mathsf{crs}, \mathtt{x}, \mathtt{w})$.

Definition 3.10 (Commit-and-Prove NARK.). *A commit-and-prove non-interactive argument of knowledge for a relation $\mathcal{R}$ and a commitment scheme* Com *is an argument of knowledge for the NP relation* $\mathcal{R}_{\mathsf{Com}}$ *such that* $((\mathtt{x}, \mathsf{com}); (u, \mathsf{open}, \mathtt{w})) \in \mathcal{R}_{\mathsf{Com}}$ *iff* $(\mathtt{x}, (u, \mathtt{w})) \in \mathcal{R}$ *and* $\mathsf{Com}.\mathsf{Ver}(\mathsf{ck}, \mathsf{com}, u, \mathsf{open}) = 1$.

We also introduce the general notion of interactive proofs for the language of verifiable computation, given by tuples (f, x, y) where f is a function, x is an input to f, and y is an output of f.

Definition 3.11 (Interactive Proof for Verifiable Computation). *Let $\mathcal{F}$ be a family of functions, and let $\mathcal{L}_{\mathcal{F}} = \{(f, x, y) : f \in \mathcal{F} \wedge f(x) = y\}$ the language that corresponds to the correct computation of a function $f(x)$. An interactive proof for $\mathcal{L}_{\mathcal{F}}$ is a pair of algorithms $b \leftarrow \langle \mathsf{P}, \mathsf{V} \rangle (f, x, y)$ such that the following properties hold:*

Completeness. *For any $(f, x, y) \in \mathcal{L}_{\mathcal{F}}$, $\Pr[\langle \mathsf{P}, \mathsf{V} \rangle (f, x, y) \rightarrow 1] = 1$.*

ϵ-Soundness. *For any algorithm P^* and $(f, x, y) \notin \mathcal{L}_{\mathcal{F}}$,*

$$\Pr[\langle \mathsf{P}^*, \mathsf{V} \rangle (f, x, y) \rightarrow 1] \leq \epsilon.$$

The probabilities are defined over the random coins of the verifier.

3.6 Digital Signatures

Definition 3.12 (Digital signature). *A digital signature scheme $\Sigma = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Ver})$ is defined as the following tuple of efficient algorithms.*

$\mathsf{KeyGen}(1^\lambda) \rightarrow (\mathsf{sk}, \mathsf{vk})$: *On input the security parameter, creates a public-private key pair* $(\mathsf{sk}, \mathsf{vk})$

$\mathsf{Sign}(\mathsf{sk}, m) \rightarrow \sigma$: *On input a message $m \in \mathcal{M}$ and the secret key sk, generates a signature σ.*

$\mathsf{Ver}(\mathsf{vk}, \sigma, m) \rightarrow b$: *Given a signature σ, a message $m \in \mathcal{M}$ and a public key pk, outputs $b \in \{0, 1\}$, indicating acceptance or rejection.*

We say that the signature scheme is correct if for any admissible $m \in \mathcal{M}$ and all choices of randomness, if $(\mathsf{sk}, \mathsf{vk}) \leftarrow\$ \mathsf{KeyGen}(1^\lambda)$ and $\sigma \leftarrow\$ \mathsf{Sign}(\mathsf{sk}, m)$, then $\mathsf{Ver}(\mathsf{vk}, \sigma, m) = 1$.

Definition 3.13 (EUF-CMA security for signatures). Let Σ be a signature scheme. Existential unforgeability, or EUF-CMA security, for Σ is defined via the game $\mathsf{EUF\text{-}CMA}_{\mathcal{A},\Sigma}(\lambda)$ depicted in Figure 3.1. We define the advantage of adversary $\mathcal{A}$ in the game

$$\mathsf{Adv}^{\mathrm{eufcma}}_{\mathcal{A},\Sigma}(\lambda) := \Pr[\mathsf{EUF\text{-}CMA}_{\mathcal{A},\Sigma}(\lambda) = 1].$$

We say that Σ is EUF-CMA if for all PPT adversaries $\mathcal{A}$ we have $\mathsf{Adv}^{\mathrm{eufcma}}_{\mathcal{A},\Sigma}(\lambda) = \mathsf{negl}(\lambda)$.

3.7 Somewhere Extractable Commitments

We recall the notion of somewhere extractable commitment scheme from [CJJ22, WW22], which is closely related to the notion of somewhere statistically binding hash functions introduced in [HW15, OPWW15]. In a nutshell, a somewhere extractable commitment is a vector commitment [CF13] with a dual-mode, programmable commitment key dk. In extractable mode, a trapdoor td allows one to extract at the programmed location i^*.

```
$\mathsf{EUF\text{-}CMA}_{\mathcal{A},\Sigma}(\lambda)$:
    $(\mathsf{sk}, \mathsf{vk}) \leftarrow\$ \Sigma.\mathsf{KeyGen}(1^\lambda)$
    $\mathsf{L_{Sig}} \leftarrow \emptyset$
    $(m, \sigma) \leftarrow \mathcal{A}^{O^{\mathsf{Sign}}}(\mathsf{pk})$
    **Output 1 iff** $\Sigma.\mathsf{Ver}(\mathsf{pk}, \sigma, m) = 1 \wedge m \notin \mathsf{L_{Sig}}$

**Oracle** $O^{\mathsf{Sign}}(m)$
    $\sigma \leftarrow\$ \Sigma.\mathsf{Sign}(\mathsf{sk}, m)$
    $\mathsf{L_{Sig}} \leftarrow \mathsf{L_{Sig}} \cup \{m\}$
    **return** $\sigma$
```

Figure 3.1: *EUF-CMA security game for a signature scheme* Σ.

Moreover, a commitment key in extractable mode is indistiguishable from a key in normal mode.

Without loss of generality, we adopt the convention that SEC admits local verification, where the commitment key dk naturally splits into sub-keys $\{\mathsf{dk}_i\}_{i\in[\ell]}$, not necessarily disjoint. Then, the verification algorithm at location i requires only to read dk_i. If this property does not apply to a given SEC scheme, one can simply let $\mathsf{dk}_i := \mathsf{dk}\ \forall i \in [\ell]$.

Definition 3.14 (Somewhere Extractable Commitment, adapted from [CJJ22, WW22]). *A somewhere extractable commitment scheme* SEC *with local verification is a tuple of algorithms* SEC = (Setup, Com, Open, Ver) *defined as follows.*

$\mathsf{Setup}(1^\lambda, 1^\ell, B) \to \mathsf{dk}$: *On input the security parameter λ, the input length ℓ, and the block size B, outputs a commitment key* dk.

$\mathsf{Com}(\mathsf{dk}, \boldsymbol{x}) \to (\mathsf{com}, \mathsf{aux})$: *On input the commitment key* dk, *and a vector $\boldsymbol{x} \in \mathcal{M}^{\ell B}$, outputs a commitment* com *and auxiliary input* aux.

$\mathsf{Open}(\mathsf{dk}, \mathsf{aux}, i) \to \pi_i$: *On input the commitment key* dk, *the auxiliary input* aux, *and an index i, outputs a local opening π_i.*

$\mathsf{Ver}(\mathsf{dk}_i, \mathsf{com}, i, x_i, \pi_i) \to b$: *On input the (local) verification key* dk_i, *the commitment* com, *an index $i \in [\ell]$, an input $x_i \in \mathcal{M}^B$, and a proof π_i, outputs a bit $b \in \{0, 1\}$.*

In addition, SEC *must include the following trapdoor-extraction algorithms:*

$\mathsf{TdSetup}(1^\lambda, 1^\ell, B, i^*) \to (\mathsf{dk}, \mathsf{td})$ *works as the setup algorithm, and additionally outputs a trapdoor* td *associated to index i^*.*

$\mathsf{Ext}(\mathsf{td}, \mathsf{com}, i^*) \to x_{i^*}$ *On input a trapdoor* td, *a commitment* com *and an index i^*, extracts an input x_{i^*}.*

Moreover, the algorithms must satisfy the following properties:

Correctness. *For any $\lambda \in \mathbb{N}$, any integers ℓ, B, index $i \in [\ell]$, and admissible inputs $\boldsymbol{x} \in \mathcal{M}^{nB}$,*

$$\Pr\left[\mathsf{Ver}(\mathsf{dk}_i, \mathsf{com}, i, x_i, \pi_i) = 1 \;\middle|\; \begin{array}{l} \mathsf{dk} \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell, B, i) \\ (\mathsf{com}, \mathsf{aux}) \leftarrow \mathsf{Com}(\mathsf{dk}, \boldsymbol{x}) \\ \pi_i \leftarrow \mathsf{Open}(\mathsf{dk}, \mathsf{aux}, i) \end{array} \right] = 1$$

Succinct local verification. *For any admissible set of parameters, there exists a function $s_{\mathsf{SEC}}(\lambda, \ell, B) = \mathsf{poly}(\lambda, B) \cdot o(\ell)$ such that the following properties hold:*

- *Succinct local verification keys: $|\mathsf{dk}_i| \leq s_{\mathsf{SEC}}(\lambda, \ell, B)$.*
- *Succinct commitments: $|\mathsf{com}| \leq s_{\mathsf{SEC}}(\lambda, \ell, B)$.*
- *Succinct local openings: $|\pi_i| \leq s_{\mathsf{SEC}}(\lambda, \ell, B)$.*
- *Fast local verification: $\mathsf{Ver}(\mathsf{dk}_i, \mathsf{com}, i, x_i, \pi_i)$ runs in time $\leq s_{\mathsf{SEC}}(\lambda, \ell, B)$.*

Setup indistinguishability. *For any PPT adversary $\mathcal{A}$, and any integers ℓ, B,*

$$\Pr\left[\mathcal{A}(\mathsf{dk}) = 1 \;\middle|\; \begin{array}{l} i^* \leftarrow \mathcal{A}(1^\lambda, 1^\ell, B) \\ (\mathsf{dk}, \mathsf{td}) \leftarrow \mathsf{TdSetup}(1^\lambda, 1^\ell, B, i^*) \end{array}\right] - \Pr\left[\mathcal{A}(\mathsf{dk}) = 1 \;\middle|\; \begin{array}{l} i^* \leftarrow \mathcal{A}(1^\lambda, 1^\ell, B) \\ \mathsf{dk} \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell, B) \end{array}\right] \leq \mathsf{negl}(\lambda)$$

Somewhere extractability. *For any PPT adversary $\mathcal{A}$, and any integers ℓ, B,*

$$\Pr\left[\begin{array}{l} \mathsf{Ver}(\mathsf{dk}_{i^*}, \mathsf{com}, i^*, x_{i^*}, \pi_{i^*}) = 1 \\ \wedge\ \mathsf{Ext}(\mathsf{td}, \mathsf{com}, i^*) \neq x_{i^*} \end{array} \;\middle|\; \begin{array}{l} i^* \leftarrow \mathcal{A}(1^\lambda, 1^\ell, B) \\ (\mathsf{dk}, \mathsf{td}) \leftarrow \mathsf{TdSetup}(1^\lambda, 1^\ell, B, i^*) \\ (\mathsf{com}, x_{i^*}, \pi_{i^*}) \leftarrow \mathcal{A}(\mathsf{dk}) \end{array}\right] \leq \mathsf{negl}(\lambda)$$

3.8 Multilinear Extensions

We recall the notion of a multilinear extension (MLE), which is a polynomial extension of a function f defined over binary vectors $\{0,1\}^\ell$ to a polynomial $\tilde{f}$ defined over the whole space $\mathbb{F}^\ell$.

Definition 3.15 (Multilinear extension). *Let $f : \{0,1\}^\ell \to \mathbb{F}$ be a function. The multilinear extension $\tilde{f}$ of f is the unique multilinear polynomial $\tilde{f} : \mathbb{F}^\ell \to \mathbb{F}$ such that $f(\boldsymbol{x}) = \tilde{f}(\boldsymbol{x})$ for all $x \in \{0,1\}^\ell$. It has the following closed form:*

$$\tilde{f}(\boldsymbol{x}) = \sum_{\boldsymbol{b} \in \{0,1\}^\ell} \tilde{I}(\boldsymbol{x}, \boldsymbol{b}) \cdot f(\boldsymbol{b})$$

Where $\tilde{I}(\boldsymbol{x}, \boldsymbol{b}) = \prod_{i=1}^{\ell} ((1 - x_i)(1 - b_i) + x_i b_i)$ is the MLE of the indicator function $I : \{0,1\}^\ell \times \{0,1\}^\ell \to \{0,1\}$ such that $I(\boldsymbol{x}, \boldsymbol{b}) = 1$ if $\boldsymbol{x} = \boldsymbol{b}$ and $I(\boldsymbol{x}, \boldsymbol{b}) = 0$ elsewhere.

For $n \in \mathbb{N}$ and a vector $\boldsymbol{x} \in \mathbb{F}^n$ and $\ell = \lceil \log n \rceil$, there exists a (unique) indexing function $f_{\boldsymbol{x}} : \{0,1\}^\ell \to \mathbb{F}$ given by $f_{\boldsymbol{x}}(\boldsymbol{b}) = x_i$ where $\boldsymbol{b} = (b_1, \ldots, b_\ell)$ is the binary representation of i. Then, we define the MLE of $\boldsymbol{x}$, that we denote by $\tilde{\boldsymbol{x}} : \mathbb{F}^\ell \to \mathbb{F}$, as the MLE of the indexing function $f_{\boldsymbol{x}}$.

Intuitively, a multilinear extension acts as an "error-correcting code" of the underlying function, as if two functions f, f' differ (even if it is on a single value), their multilinear extensions will differ almost everywhere, due to the Schwartz-Zippel lemma that we introduce below.

Lemma 3.16 (Schwartz-Zippel Lemma [Sch80, Zip79]). *Let $\mathbb{F}$ be a field and let $f : \mathbb{F}^\ell \to \mathbb{F}$ be a polynomial of total degree at most d. Then,*

$$\Pr_{\boldsymbol{x} \leftarrow\!\$\, \mathbb{F}^\ell} \left[f(\boldsymbol{x}) = 0 \right] \leq \frac{d}{|\mathbb{F}|}.$$

Finally, we introduce a lemma on the computational complexity of evaluating multilinear extensions.

Lemma 3.17 ([VSBW13]). *Given $f(\boldsymbol{x})$ for all $\boldsymbol{x} \in \{0,1\}^\ell$ and a vector $\boldsymbol{r} \in \mathbb{F}^\ell$, the value $\tilde{f}(\boldsymbol{r})$ can be computed in $O\left(2^\ell\right)$ time and $O\left(2^\ell\right)$ space.*

Part I

Succinct Proof Systems from Standard Assumptions

4

Chainable Functional Commitments for Circuits

In this chapter, we present the first constructions of functional commitments for unbounded-depth circuits from pairing-based and lattice-based falsifiable assumptions. The chapter is primarily based on the results from the article "*Chainable Functional Commitments for Unbounded Depth Circuits*" [BCFL23].[1] We additionally include some results on functional commitments that appear in [ABF24].

The chapter is structured as follows. In Section 4.1 we summarize our contributions, including the main results and their implications, followed by a technical overview in Section 4.2. In Section 4.3, we introduce functional commitments. In Section 4.4 we define our new primitive of chainable functional commitments (CFC). In Section 4.5, we present a generic compiler that translates a CFC for quadratic functions into a FC for arbitrary arithmetic circuits. Finally, in Section 4.6 and Section 4.7 we present algebraic constructions of CFCs for quadratic functions from bilinear groups and lattices, respectively.

4.1 Contributions

We present the first constructions of Functional Commitments that support the evaluation of arbitrary arithmetic circuits of unbounded depth[2] and are based on falsifiable assumptions. Our FC schemes are also *chainable*, meaning that it is possible to open to functions of multiple committed inputs while preserving the output to be in committed form. To capture such functionality, we introduce a novel primitive called *Chainable Functional Commitment* (CFC).

In our FC schemes only the maximal *width* of the circuits has to be fixed at setup time. The size of the commitments is succinct in the input size; the size of functional opening

[1] Some results from [BCFL23] are omitted for conciseness, notably those related to strong evaluation binding and extractability of our constructions, as well as an analysis of our security assumptions. We refer the reader to the full version of the paper for a complete treatment of these results.

[2] Looking ahead, our pairing-based instantiation supports arithmetic circuits over $\mathbb{Z}_q$, while our lattice-based instantiation supports arithmetic circuits over cyclotomic rings $\mathbb{Z}[\zeta]$ where wires carry values of bounded norm.

proofs grows with the multiplicative depth d_C of the evaluated circuit C, but is otherwise independent of the circuit's size or the input length. Notably, our FCs provide an exponential improvement compared to previous ones that could only support polynomials of degree $\delta = O(1)$ with an efficiency (prover time and parameter size) degrading exponentially in δ (as $O(\ell^{\delta})$)[3] [ACL+22, CFT22].

We design our FCs for circuits in two steps: (1) a generic construction of an FC for unbounded-depth circuits based on CFCs for quadratic functions, and (2) two realizations of CFCs, one based on bilinear pairings and one based on lattices. The pairing-based CFC relies on a new falsifiable assumption that we justify in the bilinear generic group model, while the lattice-based CFC relies on a slight extension of the k-R-ISIS assumption recently introduced in [ACL+22]. Using either one of these two CFC constructions (and considering a few tradeoffs of our generic construction), we obtain a variety of FC schemes.

Our FC schemes enjoy useful additional properties.

1. They are *additively homomorphic*, which as shown in [CFT22] makes the FC updatable and allows for building homomorphic signatures (HS). Notably, our new FC for circuits yields new HS realizations that advance the state of the art (see slightly below for details).
2. They enjoy *amortized efficient verification*, which means that the verifier can precompute a verification key vk_C associated to a circuit C and use this key (an unbounded number of times) to verify functional openings for C in time (asymptotically) faster than evaluating C.
3. Our FC schemes can be easily modified to have *perfectly hiding commitments* and efficiently compiled into FCs with *zero-knowledge functional openings*.

Both efficient verification and zero-knowledge functional openings are relevant in the construction of HS from FCs since, as showed in [CFT22], they imply the analogous properties of efficient verification [CFW14, GVW15] and *context hiding* [BF11] in the resulting HS schemes.

Comparison to the state of the art. In Table 4.1 we compare our two main instantiations of FCs for circuits from pairings (Corollary 4.18.1) and from lattices (Corollary 4.24.2) to other algebraic functional commitment schemes in the literature. In the first half of Table 4.1 we describe schemes that appeared prior to this work, none of which supports arithmetic circuits but only restricted functionalities such as linear maps, semi-sparse polynomials and constant-degree polynomials. In the second half, we include schemes that are concurrent [dCP23, WW23b] and subsequent [WW23a, WW24b] to our results from this chapter [BCFL23]. For completeness, we also include our scheme in Chapter 5, which improves on the public parameter succinctness of the pairing-based scheme from (Corollary 4.18.1).

[3] Note, when used for a circuit of depth d these solutions may have efficiency doubly exponential in d since in general $\delta \approx 2^d$.

FC scheme	Functions	$\lvert\mathsf{pp}\rvert$	$\lvert\mathsf{com}\rvert$	$\lvert\pi\rvert$	AH	CH	EV
[LRY16]	linear maps	$\lambda\ell$	λ	$\lambda\ell_y$	✓	–	✓
[LM19]	linear maps	$\lambda\ell\ell_y$	λ	λ	✓	–	✓
[LP20]	semi-sp. poly	$\lambda\mu$	$\lambda\ell_y$	λ	–	–	✓
[ACL+22]†	deg-δ poly	$p(\lambda)\,\ell^{2\delta}$	$p(\lambda)\,\log\ell$	$p(\lambda)\,\log^2(\ell\ell_y)$	✓	–	✓
[CFT22]	deg-δ poly	$\lambda\ell^{2\delta+1}$	$\lambda\delta_f$	$\lambda\delta_f$	✓	–	✓
Cor. 4.18.1	arithm. circ.	λw^5	λ	λd^2	✓	✓	✓
Cor. 4.24.2 †	arithm. circ.	$p(\lambda)\,w^5$	$p(\lambda)\,\log w$	$p(\lambda)\,d\log^2 w$	✓	✓	✓
[dCP23]†	boolean circ.	$p(\lambda,d)\,\ell^2$	$p(\lambda,\log\ell)$	$p(\lambda,\ell,d)$	–	–	–
[WW23b]†	boolean circ.	$p(\lambda,d)\,\ell^2$	$p(\lambda,\log\ell)$	$p(\lambda,\log\ell,d)\,\ell_y$	–	–	–
[WW23a]†	boolean circ.	$p(\lambda,d)\,\ell^2$	$p(\lambda,\log\ell)$	$p(\lambda,\log\ell,d)\,\ell_y$	–	–	✓
[WW24b]	arithm. circ.	λs^5	λ	λ	✓	✓	✓
Theorem 5.8	arithm. circ.	λw^3	λ	λd^2	✓	✓	✓

Table 4.1: *Comparison of algebraic functional commitments for functions $\mathbb{F}^\ell \to \mathbb{F}^{\ell_y}$. Schemes with † are lattice-based, all other schemes are pairing-based. Constants are omitted, e.g., $\lambda\ell$ means $O(\lambda\ell)$ and $p(\cdot) = \mathsf{poly}(\cdot)$ represents some arbitrary polynomial function. For semi-sparse polynomials $\mu \geq \ell$ is a sparsity-dependent parameter (cf. [LP20]). For constant-degree polynomials δ_f is the degree of the polynomial f used in the opening while δ is the maximum degree fixed at setup. For circuits, d is the depth of the circuit $\mathbf{C}$ used in the opening, w is its width (note that $w \geq \ell, \ell_y$), and s its size (note $s \leq w \cdot d$). **AH** means 'additively homomorphic'; such schemes can be turned into homomorphic signatures following the compiler in [CFT22]. **CH** means 'chainable'; such schemes can be used to build a functional commitment for unbounded-depth circuits, following our compiler in Section 4.5. **EV** means 'efficient verification'; such schemes admit sublinear time verification (in the circuit or function size) after preprocessing.*

Application to Homomorphic Signatures. By applying a recently proposed transformation [CFT22], our new FCs for circuits yield new HS that support the same class of functions and succinctness as supported by the FC, advancing the state of the art. Notably, we obtain:

- *The first HS for circuits based on pairings.* Previously existing HS based on pairings can capture at most circuits in NC^1 [KNYY19, CFT22] and need a bound on the circuit size. In contrast, our HS can evaluate circuits of any polynomial depth, achieving virtually the same capability of the lattice-based HS of [GVW15] that is based on lattices and supports circuits with bounded number of inputs ℓ and bounded (polynomial) depth d. [4] Our result shows for the first time that we can build HS for circuits without the need of algebraic structures, such as lattices, that are notoriously powerful.
- *The first HS that do not require an a-priori bound on the depth.* The work of Gorbunov, Vaikuntanathan and Wichs [GVW15] left open the problem of constructing *fully-homomorphic* signatures, i.e., HS that can evaluate any computation in the class P without having to fix any bound at key generation time. In our new HS we do not need to fix a bound on the depth. Instead, we need a bound on the width of the circuits at key generation time. Although this result does not fully solve the open problem of realizing fully-homomorphic signatures, we believe that our schemes make one step ahead in this direction, as dealing with a bound on the circuit's depth is more difficult than dealing with a bound on the width. As evidence for this, we show a variant of our FC scheme (see Section 4.5.4) for which one can fix a bound $\ell_{\max}$ and support circuits of larger width $O(\ell_{\max})$ with an $O(1)$ increase in proof size.

Like the scheme of [GVW15], our HS constructions have efficient (offline/online) verification and are context-hiding. As a drawback, our HS allow only a limited form of multi-hop evaluation, that is the ability of computing on already evaluated signatures. In our case, we can compose computations sequentially (i.e., given a signature $\sigma_{f,y}$ for $\boldsymbol{y} = f(\boldsymbol{x})$ we can generate one for $\boldsymbol{z} = g(\boldsymbol{y}) = g(f(\boldsymbol{x}))$), while [GVW15] supports arbitrary compositions (e.g., given signatures for $\{\boldsymbol{y}_i = f_i(\boldsymbol{x})\}_i$, one can generate one for $z = g(f_1(\boldsymbol{x}), \ldots, f_n(\boldsymbol{x}))$). On the other hand, for circuits with multiple outputs, the size of our signatures is independent of the output size, whereas in [GVW15] signatures grow linearly with the number of outputs.

Our Novel Tool: Chainable Functional Commitments. The key novelty that allows us to overcome the barrier in the state of the art and build the first FCs for circuits is the introduction and realization of chainable functional commitments (CFC) – a new primitive of potentially independent interest.

In brief, a CFC is a functional commitment where one can "open" to *committed outputs*. More concretely, while a (basic) FC allows proving statements of the form "$f(\boldsymbol{x}) = \boldsymbol{y}$" for committed $\boldsymbol{x}$ and publicly known $\boldsymbol{y}$, a CFC allows generating a proof π_f that com_y is a

[4] In their scheme, the signature size grows polynomially with the depth of the evaluated circuit (precisely, as $d^3 \cdot \mathsf{poly}(\lambda)$)

commitment to $\boldsymbol{y} = f(\boldsymbol{x}_1, \ldots \boldsymbol{x}_m)$ for vectors $\boldsymbol{x}_1, \ldots \boldsymbol{x}_m$, each independently committed in $\mathsf{com}_1, \ldots, \mathsf{com}_m$. In terms of security, CFCs must satisfy the analogue of evaluation binding, that is one cannot open the same input commitments $(\mathsf{com}_1, \ldots, \mathsf{com}_m)$ to two distinct output commitments $\mathsf{com}_y \neq \mathsf{com}'_y$ for the same f.

Keeping outputs committed is what makes CFCs "chainable", in the sense that committed outputs can serve as (committed) inputs for further functional openings. For instance, using the syntax above, one can compute an opening π_g proving that com_z is a commitment to $\boldsymbol{z} = g(\boldsymbol{y})$. This way, the concatenation of $\mathsf{com}_y, \pi_f, \pi_g$ yields a proof that $\boldsymbol{z} = g(f(\boldsymbol{x}_1, \ldots \boldsymbol{x}_m))$.

The introduction and realization of CFCs are in our opinion the main conceptual and technical contributions of this paper. From a conceptual point of view, the chaining functionality turns out to be a fundamental feature to tackle the challenge of supporting a computation as expressive as an arithmetic circuit. Indeed, we show that from a CFC for quadratic polynomial maps it is possible to construct a (C)FC for arithmetic circuits. From the technical point of view, we propose new techniques that depart from the ones of existing FCs for polynomials [ACL+22, CFT22] in that the latter only work when the output vector is known to the verifier and there is a single input commitment.

Other Contributions. To broaden the instantiations of our constructions and to enable the evaluation of unbounded-depth circuits, we give two additional results on output-succinctness and unbounded FCs, which are generic and may be of independent interest.

- In **Theorem 4.25**, we show that any succinct FC for ℓ-to-1 functions can be transformed into a ℓ-to-m FC that is fully succinct in the output, i.e., the proof size does not grow with m.
- In **Theorem 4.26**, we show that any suitably expressive FC can be boosted into a single-input *chainable* FC [BCFL23], which is sufficient to construct FC schemes for unbounded-depth circuits.

4.2 Technical Overview

We construct our FCs for circuits in two main steps: (1) a generic construction of (C)FC for circuits from CFCs for quadratic polynomial maps (Section 4.5), and (2) the realization of these CFCs based on either pairings (Section 4.6) or lattices (Section 4.7). Below we give an informal overview of these constructions.

4.2.1 (C)FC for Circuits from CFCs for Quadratic Functions

Our first result is a transformation from CFCs for quadratic polynomials to FCs for circuits that is summarized in the following theorem.

Theorem 4.26 (informal) *Let* CFC *be a* chainable *functional commitment for quadratic polynomial maps $f(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m) = \boldsymbol{y}$ for any number of inputs m, such that each committed input* *vector $\boldsymbol{x}_i$ and the committed output $\boldsymbol{y}$ have length bounded by ℓ. Then, there exist a functional commitment* FC *for arithmetic circuits of bounded width ℓ and unbounded depth d, such that:*

- the commitment size of FC *is the same as that of* CFC*;*

- if CFC *has opening proofs of size $s(\ell, m)$, then* FC *has openings of size at most $d \cdot s(\ell, d)$. Moreover, if* CFC *is additively homomorphic and/or efficiently verifiable, so is* FC.

Our transform starts from the observation that the gates of an arithmetic circuit[5] can be partitioned into "levels" according to their multiplicative depth, i.e., level h contains all the gates of multiplicative depth h and level 0 contains the inputs. All the outputs of level h, denoted by $\boldsymbol{x}^{(h)}$, are computed by a quadratic polynomial map taking inputs from previous levels $< h$, and thus the evaluation of a circuit C of width $\leq \ell$ and depth d can be described as the sequential evaluation of quadratic polynomial maps $f^{(h)} : \mathcal{M}^{\ell h} \to \mathcal{M}^{\ell}$ for $h = 1$ to d.

The basic idea of our generic FC is that, starting with a commitment com_0 to the inputs $\boldsymbol{x}^{(0)}$, we can open it to $\boldsymbol{y} = C(\boldsymbol{x}^{(0)})$ in two steps. First, we commit to the outputs of every level. Second, we use the CFC functional opening functionality to prove that these values are computed correctly from values committed in previous levels. Slightly more in detail, at level h we create a commitment com_h to the outputs $\boldsymbol{x}^{(h)} = f^{(h)}(\boldsymbol{x}^{(0)}, \ldots, \boldsymbol{x}^{(h-1)})$ and generate a CFC functional opening proof π_h to show consistency w.r.t. commitments $\mathsf{com}_0, \ldots, \mathsf{com}_h$. Eventually, this strategy reaches the commitment com_d of the last level that includes the outputs, which can be opened to $\boldsymbol{y}$ (or kept committed if one wants to build a CFC for circuits). The final proof π consists of all intermediate proofs and commitments, $\pi := (\pi_1, \ldots, \pi_d, \mathsf{com}_1, \ldots, \mathsf{com}_{d-1})$.

Security reduces to the security of the CFC for quadratic functions. To see this, consider an adversary that breaks FC evaluation binding by coming up with proofs π, π' that verify for $\boldsymbol{y} \neq \boldsymbol{y}'$ and for the same com_0. Then, there *must* exist some level h such that the intermediate commitments $\mathsf{com}_h \neq \mathsf{com}'_h$ differ (where possibly $h = d$). If we take h^* to be the smallest amongst such h, then we can break evaluation binding of the quadratic CFC at level h^*.

As one can see, this construction makes our functional opening proofs grow with the depth of the circuit. However, if the CFC commitments and proofs are short (e.g., $s(\ell, m)$ is constant or logarithmic in the circuit width ℓ), then the FC proofs keep only such dependence on the depth.[6] In addition, in Section 4.5.4 we describe different strategies to (a) reduce the proof size for not-so-densely connected circuits (for instance layered circuits), and (b) overcome the width bound without changing the parameters at the expense of increased proof size.

[5] In our model we assume wlog arithmetic circuits where every gate is a quadratic polynomial of unbounded fan-in.

[6] The CFC functional openings size $s(\ell, d)$ may be linear in d for "dense" quadratic functions, but this would contribute at most an additional factor of d to the succinctness of FC in the worst case.

4.2.2 A Framework for CFCs for Quadratic Functions

We next overview our general strategy of construction CFCs for quadratic functions, which admits pairing- and lattice-based instantiations.

Theorems 4.17 and 4.20 (informal). *Assuming the ℓ-HiKer assumption (resp. the Twin-k-$\mathcal{R}$-*ISIS *assumption), our pairing-based (resp. lattice-based) CFC construction is a succinct CFC scheme for quadratic functions over any m vectors of length $\leq \ell$ that admits efficient verification, is additively homomorphic, and whose functional openings can be made zero-knowledge. For arbitrary quadratic functions, the opening proofs have size $O(\lambda m^2)$ (resp. $O(\mathsf{poly}(\lambda) \cdot m \cdot \mathsf{polylog}(m \cdot \ell))$)*[7]*.*

To build our CFCs we devise new commitment and opening techniques that capture a quadratic polynomial map $\boldsymbol{y} = f(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m)$ where each input is committed in com_i, and the output is committed too in $\mathsf{com}_{\boldsymbol{y}}$. Our two constructions (pairing-based and lattice-based) of CFCs for quadratic functions have a similar high-level design that we introduce below.

For the pairing setting we adopt the implicit notation for bilinear groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ of prime order q by which $[\boldsymbol{x}]_s$ denotes the vector of group elements $(g_s^{x_1}, \ldots, g_s^{x_n}) \in \mathbb{G}_s^\ell$ for a fixed generator g_s. For the lattice setting, we let $\mathcal{R}$ be a cyclotomic ring and q be a large enough rational prime. In this overview, we adopt the bracket notation $[x]$ to express the representation of a given group or ring element without further distinction.

Abstract functionality. To start, we define three (vectors of) commitment keys $[\boldsymbol{\alpha}]$, $[\boldsymbol{\beta}]$, and $[\boldsymbol{\gamma}]$, that live either in $\mathbb{G}_1^\ell$ in the pairing setting, or in $\mathcal{R}_q^\ell$ in the lattice setting. A commitment of type α to a vector $\boldsymbol{x} \in \mathbb{Z}_q^\ell$ is computed *à la* Pedersen, i.e., via an inner product, as $X^{(\alpha)} = [\langle \boldsymbol{x}, \boldsymbol{\alpha} \rangle]$. Commitments of type β and γ are defined analogously.

In our CFCs the commitments generated by the commit algorithm Com and used by the functional opening algorithm $\mathsf{FuncProve}$ are only those of type α, whereas commitments of type β and γ are used as auxiliary values in the proofs. In order to create a CFC opening to a quadratic polynomial, our main tool is a technique realizing the following functionality:

– $\underline{[(\alpha, \beta) \rightarrow \gamma]\text{-Quadratic opening:}}$ given m pairs of commitments to m inputs in both keys $\left\{X_i^{(\alpha)} = [\langle \boldsymbol{x}_i, \boldsymbol{\alpha} \rangle], X_i^{(\beta)} = [\langle \boldsymbol{x}_i, \boldsymbol{\beta} \rangle]\right\}_{i=1\ldots m}$ and a commitment $Y^{(\gamma)} = [\langle \boldsymbol{y}, \boldsymbol{\gamma} \rangle]$, generate a succinct functional proof $\pi_f^{(\gamma)}$ that $\boldsymbol{y} = f(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m)$.

Before seeing how we generate this opening, we observe that $\pi_f^{(\gamma)}$ does not yet achieve our goal since it assumes the availability of both type-α and type-β commitments on the inputs, and it only allows us to "move" to a type-γ commitment of the output, preventing us from achieving chainability.

We solve both issues by designing two special cases of the functionality above:

[7] Following Theorem 2, this gives a proof size of $O(\lambda d^3)$ for our pairing-based FC and $O(\mathsf{poly}(\lambda) \cdot d^2 \cdot \mathsf{polylog}(d \cdot w))$ for our lattice-based FC for circuits of depth d and width w. Nevertheless, the proof size can be reduced by a factor of d in both cases, as we show in Table 4.1. We refer to Sections 4.6 and 4.7 for details.

– $[\alpha \to \beta]$-Identity opening: given a type-α commitment $X^{(\alpha)} = [\langle \boldsymbol{x}, \boldsymbol{\alpha} \rangle]$ show that a type-β commitment $X^{(\beta)}$ commits to the same $\boldsymbol{x}$, i.e., $X^{(\beta)} = [\langle \boldsymbol{x}, \boldsymbol{\beta} \rangle]$;

– $[\gamma \to \alpha]$-Identity opening: given a type-γ commitment $Y^{(\gamma)} = [\langle \boldsymbol{y}, \boldsymbol{\gamma} \rangle]$ show that a type-α commitment $Y^{(\alpha)}$ commits to the same $\boldsymbol{y}$, i.e., $Y^{(\alpha)} = [\langle \boldsymbol{y}, \boldsymbol{\alpha} \rangle]$.

We use the identity opening mechanisms to "close the circle" in such a way to obtain a quadratic opening mechanism where all inputs and outputs are only type-α commitments. To summarize, our CFC FuncProve algorithm consists of the following steps:

(i) compute a type-β commitment $X_i^{(\beta)}$ to each input along with an $[\alpha \to \beta]$-Identity opening proof that $X_i^{(\beta)}$ commits to the same $\boldsymbol{x}_i$ in $X_i^{(\alpha)}$;

(ii) compute a type-γ commitment $Y^{(\gamma)}$ to the result $\boldsymbol{y} = f(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m)$ and a $[(\alpha, \beta) \to \gamma]$-Quadratic opening proof attesting the validity of $\boldsymbol{y}$ w.r.t. the input commitment pairs $(X_i^{(\alpha)}, X_i^{(\beta)})$;

(iii) finally, use the $[\gamma \to \alpha]$-identity opening to ensure that $Y^{(\alpha)}$ is a commitment to the same $\boldsymbol{y}$ in the $Y^{(\gamma)}$ computed in (ii).

Our $[(\alpha, \beta) \to \gamma]$-quadratic opening method. We use the fact that a quadratic polynomial map $f : \mathcal{M}^{nm} \to \mathcal{M}^{\ell}$ can be linearized via appropriately defined vector $\boldsymbol{e}$ and matrices $\mathbf{F}_i$ and $\mathbf{G}_{i,j}$ such that

$$\boldsymbol{y} = f(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m) = \boldsymbol{e} + \sum_i \mathbf{F}_i \cdot \boldsymbol{x}_i + \sum_{i,j \geq i} \mathbf{G}_{i,j} \cdot (\boldsymbol{x}_i \otimes \boldsymbol{x}_j)$$

where $\otimes$ denotes the tensor product.

In this overview, we only show how to produce a functional opening for a single quadratic term, i.e., to show that $\boldsymbol{y}_{i,j} = \mathbf{G}_{i,j} \cdot (\boldsymbol{x}_i \otimes \boldsymbol{x}_j)$ given input commitments $X_i^{(\alpha)}, X_i^{(\beta)}, X_j^{(\alpha)}, X_j^{(\beta)}$ and output $Y_{i,j}^{(\gamma)}$. This is the core of our technique since the full opening for f is obtained by doing an additive aggregation of openings for all the terms in the sum.

To open to $\mathbf{G}_{i,j}$, we first ensure that the verifier knows a commitment $Z_{i,j}$ to the tensor product $\boldsymbol{x}_i \otimes \boldsymbol{x}_j$, calculated as

$$Z_{i,j} := [\langle \boldsymbol{x}_i \otimes \boldsymbol{x}_j, \boldsymbol{\alpha} \otimes \boldsymbol{\beta} \rangle].$$

The way in which the verifier obtains $Z_{i,j}$ varies in the pairing and lattice constructions. Then, the prover generates a linear map opening that the vector $\boldsymbol{y}_{i,j}$ in the type-γ commitment $Y_{i,j}^{(\gamma)}$ is the result of applying $\mathbf{G}_{i,j}$ to the vector committed in $Z_{i,j}$. We compute this proof as follows. Denote with $\mathbf{G}_{i,j,k}$ the k-th row of $\mathbf{G}_{i,j}$ and with $\frac{1}{\boldsymbol{\alpha} \otimes \boldsymbol{\beta}}$ the component-wise inverse of $\boldsymbol{\alpha} \otimes \boldsymbol{\beta}$. Let

$$\Gamma_{i,j} = \sum_{k=1}^{\ell} \mathbf{G}_{i,j,k} \cdot \left[\frac{\gamma_k}{\boldsymbol{\alpha} \otimes \boldsymbol{\beta}} \right]$$

be an encoding of the matrix $\mathbf{G}_{i,j}$ that should be computable by the verifier (who can also pre-compute $\Gamma_{i,j}$). Then we rely on the fact that

$$Z_{i,j} \cdot \Gamma_{i,j} = [\langle \underbrace{\mathbf{G}_{i,j} \cdot \boldsymbol{x}_i \otimes \boldsymbol{x}_j}_{\boldsymbol{y}_{i,j}}, \boldsymbol{\gamma} \rangle] + \sum_{(h,l) \neq (h',l'),k} c_{h,l,h',l',k} \cdot \left[\frac{\alpha_{h'} \beta_{l'}}{\alpha_h \beta_l} \gamma_k \right]. \tag{4.1}$$

Namely, $Z_{i,j} \cdot \Gamma_{i,j}$ can be split into the sum between a non-rational term that actually encodes the (commitment to the) result $[\langle \boldsymbol{y}_{i,j}, \boldsymbol{\gamma} \rangle]$, and a linear combination of rational monomials, which is eventually encoded as part of the opening proof, and whose coefficients can be efficiently computed given $\mathbf{G}_{i,j}, \boldsymbol{x}_i$ and $\boldsymbol{x}_j$.

For the prover to prove such splitting, and for the verifier to compute the encoding, we need to include additional elements in the public parameters. In particular, we add: $[\boldsymbol{\alpha} \otimes \boldsymbol{\beta}]$ for computing $Z_{i,j}$, $\left[\frac{\gamma_k}{\boldsymbol{\alpha} \otimes \boldsymbol{\beta}}\right]$ for computing $\Gamma_{i,j}$, and $\left[\frac{\alpha_{h'}\beta_{l'}}{\alpha_h \beta_l}\gamma_k\right]$ for computing the sum in (4.1). To obtain security, we instantiate the different commitments and verification checks over pairing groups and lattice rings, whose particularities we describe next.

4.2.3 Pairing-Based CFC

In our pairing-based CFC in Section 4.6, the elements in the public parameters belong to the groups $\mathbb{G}_1$ and $\mathbb{G}_2$, and the input commitment is computed in $\mathbb{G}_1$ as $\left[X^{(\alpha)}\right]_1 = [\langle \boldsymbol{x}, \boldsymbol{\alpha} \rangle]_1$. For the verifier to obtain the commitment to the tensor product $Z_{i,j}$, the prover calculates and sends $\left[Z_{i,j}\right]_1 := \left[\langle \boldsymbol{x}_i \otimes \boldsymbol{x}_j, \boldsymbol{\alpha} \otimes \boldsymbol{\beta} \rangle\right]_1$ and $\left[X_i^{(2)}\right]_2 := [\langle \boldsymbol{x}_i, \boldsymbol{\alpha} \rangle]_2$, and the verifier checks

$$\left[X_i^{(\alpha)}\right]_1 \cdot [1]_2 \stackrel{?}{=} [1]_1 \cdot \left[X_i^{(2)}\right]_2$$

to test that $\left[X_i^{(2)}\right]_2 \in \mathbb{G}_2$ encodes the same vector of $\left[X_i^{(\alpha)}\right]_1 \in \mathbb{G}_1$, and

$$\left[Z_{i,j}\right]_1 \cdot [1]_2 \stackrel{?}{=} \left[X_j^{(\beta)}\right]_1 \cdot \left[X_i^{(2)}\right]_2$$

to test the well-formedness of $\left[Z_{i,j}\right]_1$. To let the prover compute this, we add elements $[\boldsymbol{\alpha}]_2$ in $\mathbb{G}_2$ to the public parameters.

Finally, the prover computes and sends $\left[\pi_{i,j}^{(\gamma)}\right]_1 = \sum_{(h,l)\neq(h',l'),k} c_{h,l,h',l',k} \cdot \left[\frac{\alpha_{h'}\beta_{l'}}{\alpha_h\beta_l}\gamma_k\eta_\gamma\right]_1$. This way, the verifier can test equation (4.1) using pairings as

$$\left[Z_{i,j}\right]_1 \cdot \left[\Gamma_{i,j}\right]_2 \stackrel{?}{=} \left[Y_{i,j}^{(\gamma)}\right]_1 \cdot \left[\eta_\gamma\right]_2 + \left[\pi_{i,j}^{(\gamma)}\right]_1 \cdot [1]_2 . \tag{4.2}$$

Note that we are introducing an additional variable η_γ in the verification, which is central to the security of the scheme. More precisely, in our pairing-based CFC we provide in the public parameters the elements: $\left[\eta_\gamma\right]_2$ (to be used in the verification above), $\left\{\left[\frac{\gamma_k\eta_\gamma}{\boldsymbol{\alpha}\otimes\boldsymbol{\beta}}\right]_2\right\}_k$ (used to compute $\left[\Gamma_{i,j}\right]_2$), and $\left\{\left[\frac{\alpha_{h'}\beta_{l'}}{\alpha_h\beta_l}\gamma_k\eta_\gamma\right]_1\right\}_{(h,l)\neq(h',l'),k}$ (to compute the proof $\left[\pi_{i,j}^{(\gamma)}\right]_1$). The security of the scheme relies precisely on the fact that the public parameters do not include any term of the form $\left[\gamma_k\eta_\gamma\right]_1$ in the group $\mathbb{G}_1$.

To see how this relates to the scheme, suppose that one breaks evaluation binding by finding two proofs $\left[\pi_{i,j}^{(\gamma)}\right]_1, \left[\tilde{\pi}_{i,j}^{(\gamma)}\right]_1$ that open to different commitments $\left[Y_{i,j}^{(\gamma)}\right]_1$ and $\left[\tilde{Y}_{i,j}^{(\gamma)}\right]_1$ for the same function $\mathbf{G}_{i,j}$. Then, by (4.2), we can compute $[U]_1 = \left[\tilde{Y}_{i,j}^{(\gamma)}\right]_1 / \left[Y_{i,j}^{(\gamma)}\right]_1$ and $[V]_1 = \left[\pi_{i,j}^{(\gamma)}\right]_1 / \left[\tilde{\pi}_{i,j}^{(\gamma)}\right]_1$ such that $([U]_1, [V]_1)$ is in the linear span of $\left([1]_1, \left[\eta_\gamma\right]_1\right)$. However, elements of this form cannot be derived from linear combinations of group elements in the public parameters. This is captured formally by our HintedKernel (HiKer) assumption, which we justify in the generic (bilinear) group model. Our HiKer assumption can be seen as a "hinted" version of the KerMDH assumptions of [MRV16] (see Section 6.3).[8]

In terms of succinctness, the functional proof size of our pairing-based CFC is linear in the density of the quadratic polynomial, that is the number of nonzero quadratic terms $\boldsymbol{x}_i \boldsymbol{x}_j$, which is in the worst case quadratic on the number m of input commitments. This is due to the fact that, even if we can compress all proofs $\left[\pi_{i,j}^{(\gamma)}\right]_1$ in one, the prover still needs to provide every $\left[Z_{i,j}\right]_1$ for $1 \leq i \leq j \leq m$. Fortunately, in our construction of FC for circuits, we can reduce the proof size of the CFC at each layer from quadratic to linear. We refer to Corollary 4.18 for further details.

4.2.4 Lattice-Based CFC

In our lattice-based CFC in Section 4.7, we sample commitment keys $\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}$ uniformly from $\mathcal{R}_q^\ell$. The public parameters also contain two trapdoored matrices $\mathbf{A}, \mathbf{B} \in \mathcal{R}_q^{\eta \times \zeta}$ and a vector $\boldsymbol{t} \in \mathcal{R}_q^{\eta}$, where $\mathcal{R}$ is the ring of integers of a cyclotomic field, and η, ζ are determined by the trapdoor sampling algorithm. Instead of providing the ring elements $\boldsymbol{\alpha} \otimes \boldsymbol{\beta}$ (and all elements that result from evaluating diverse monomials g on $\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}$) as in the pairing-based construction, we include a short preimage $\boldsymbol{u}_g$ of each ring element such that $\mathbf{A} \cdot \boldsymbol{u}_g \equiv \boldsymbol{t} \cdot g(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}) \bmod q$, obtained with the help of the trapdoors[9].

Given commitments $X_i^{(\alpha)} = \langle \boldsymbol{\alpha}, \boldsymbol{x}_i \rangle$, the verifier can easily compute the commitment to the tensor product $Z_{i,j} = X_i^{(\alpha)} \cdot X_j^{(\beta)} = \langle \boldsymbol{\alpha} \otimes \boldsymbol{\beta}, \boldsymbol{x}_i \otimes \boldsymbol{x}_j \rangle$ thanks to the ring structure of $\mathcal{R}_q$. We note that in the scheme, we need to make an additional restriction that both the vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m$ and the coefficients of the polynomial map f are short. This implies that the coefficients $c_{h,l,h',l',k}$ in equation (4.1) are also short.

With this restriction, we enable the proof of the split using the short preimages of each ring element $\frac{\alpha_{h'}\beta_{l'}}{\alpha_h \beta_l}\gamma_k$ available in the public parameters. This allows the prover to compute a short preimage $\boldsymbol{u}_{i,j}^{(\gamma)}$ for the element $Z_{i,j} \cdot \Gamma_{i,j} - Y_{i,j}^{(\gamma)}$, which the verifier can efficiently check by $\mathbf{A} \cdot \boldsymbol{u}_{i,j}^{(\gamma)} \equiv \boldsymbol{t} \cdot \left(Z_{i,j} \cdot \Gamma_{i,j} - Y_{i,j}^{(\gamma)}\right) \bmod q$ (note again that $\Gamma_{i,j}$ depends on the function and can be pre-computed), in addition to a norm check on $\boldsymbol{u}_{i,j}^{(\gamma)}$.

In comparison to our pairing-based CFC, here the prover no longer needs to provide the $Z_{i,j}$ elements, but only the $X_i^{(\beta)}$ for every $1 \leq i \leq m$ such that $\boldsymbol{x}_i \boldsymbol{x}_j$ is non-zero for some j. Thus, the functional opening proof size of our lattice-based CFC is (at most) linear in the number m of committed vectors. Naively, this results in proofs for our lattice-based FC for circuits that grow quadratically on the circuit depth. However, in Corollary 4.24 we show how to reduce this dependency from quadratic to linear for any circuit (even non-layered ones). We also note that the lattice parameters need to be set as a function of the size ℓ

[8] For matrices $[\mathbf{A}]_2$ from certain (random) distributions, KerMDH asks the adversary to find a nonzero vector $[\boldsymbol{z}]_1$ such that $\mathbf{A}\boldsymbol{z} = \mathbf{0}$. In HiKer, the adversary is challenged to find nonzero $[z]_1 = [u, v]_1$ such that $u\eta + v = 0$, when given $[\mathbf{A}]_2 = \left[1, \eta\right]_2$, but also other group elements, the "hints", that depend on η and other random variables.

[9] Preimages of some monomials with respect to $\mathbf{B}$ are also included in the public parameters, but we omit them in this overview.

of the committed vectors, therefore the proof also grows (logarithmically) in the circuit width.

The security of the scheme essentially relies on the fact that no short preimage for ring elements γ_k is available to the prover. We capture this fact via the Twin-k-R-ISIS assumption (Section 4.7), which extends the k-R-ISIS assumption from [ACL⁺22]. Essentially, k-R-ISIS states that even when given short preimages $\boldsymbol{u}_g$ satisfying $\mathbf{A} \cdot \boldsymbol{u}_g \equiv \boldsymbol{t} \cdot g(\boldsymbol{v}) \bmod q$ for all g in a given set of monomials, it is hard to find a SIS solution (i.e. a short non-zero preimage $\boldsymbol{u}^*$) such that $\mathbf{A} \cdot \boldsymbol{u}^* \equiv \mathbf{0} \bmod q$. Our Twin-$k$-$R$-ISIS states that finding a solution $(\boldsymbol{u}^*, \boldsymbol{v}^*)$ for $\mathbf{A} \cdot \boldsymbol{u}^* + \mathbf{B} \cdot \boldsymbol{v}^* \equiv \mathbf{0} \bmod q$ is still hard even if we also provide preimages of a *strictly different* set of monomials with respect to a second (independent) matrix $\mathbf{B}$.

In Section 3.3.1, we analyse Twin-k-R-ISIS and compare it to k-R-ISIS and the BASIS assumption from [WW23b]. Although both our Twin-k-R-ISIS and HiKer assumptions are new and non-standard, we remark that they are well-parametrized assumptions with a simple winning condition, which differs from that of the FC scheme. As typical in the lifetime of new cryptographic primitives, we expect that future work can fill this gap.

4.3 Functional Commitments

In this section we give the definition of functional commitments (FC) for generic classes of functions, by generalizing the one given in [LRY16] for linear functions. For notational simplicity we omit explicit references to subvectors since they are not needed in this thesis, although they can be easily added to the definition.

Definition 4.1 (Functional Commitments). *Let $\mathcal{M}$ be some domain and let $\mathcal{F} \subseteq \{f : \mathcal{M}^\ell \to \mathcal{M}^{\ell_y}\}$ be a family of functions over $\mathcal{M}$, with ℓ inputs and ℓ_y outputs. A functional commitment scheme (FC) for $\mathcal{F}$ consists of a commitment scheme* (Setup, Com)[10] *for $\mathcal{M}^\ell$ augmented with two additional algorithms* (FuncProve, FuncVer) *with the following syntax:*

FuncProve(ck, aux, f) $\to \pi$ *on input an auxiliary information* aux *and a function $f \in \mathcal{F}$, outputs a functional opening proof π.*

FuncVer(ck, com, $f, \boldsymbol{y}, \pi$) $\to b \in \{0, 1\}$ *on input a commitment* com, *a functional opening proof π, a function $f \in \mathcal{F}$ and a value $\boldsymbol{y} \in \mathcal{M}^{\ell_y}$, accepts ($b = 1$) or rejects ($b = 0$).*

Moreover, the algorithms must satisfy the following properties:

Correctness. FC *is correct if for any $\ell \in \mathbb{N}$, all* ck $\leftarrow\$$ Setup($1^\lambda, 1^\ell$)*, any $f : \mathcal{M}^\ell \to \mathcal{M}^{\ell_y}$ in the class $\mathcal{F}$, and any $\boldsymbol{x} \in \mathcal{M}^\ell$, if* (com, aux) $\leftarrow$ Com(ck, $\boldsymbol{x}$)*, then*

$$\Pr[\mathsf{FuncVer}(\mathsf{ck}, \mathsf{com}, f, f(\boldsymbol{x}), \mathsf{FuncProve}(\mathsf{ck}, \mathsf{aux}, f)) = 1] = 1.$$

Succinctness. *Let us assume that the admissible functions can be partitioned as $\mathcal{F} = \{\mathcal{F}_\kappa\}_{\kappa \in \mathcal{K}}$ for some set $\mathcal{K}$, and let $s_{\mathsf{FC}} : \mathbb{N} \times \mathcal{K} \to \mathbb{N}$ be a function. A functional commitment* FC *for $\mathcal{F}$ is*

[10] In Theorem 3.1, we provide a generic definition of commitment schemes that also includes (standard) opening and verification algorithms Open, Ver. Here, we omit them as they are subsumed by the FC algorithms FuncProve, FuncVer.

said to be $s_{\mathsf{FC}}(\ell, \kappa)$-succinct *if (a) the underlying commitment scheme* $(\mathsf{Setup}, \mathsf{Com})$ *is succinct (Theorem 3.1), and (b) there exists a polynomial* $p(\lambda) = \mathsf{poly}(\lambda)$ *such that for any* $\kappa \in \mathcal{K}$*, function* $f : \mathcal{M}^{\ell} \to \mathcal{M}^{\ell_y}$ *such that* $f \in \mathcal{F}_{\kappa}$*, honestly generated commitment key* $\mathsf{ck} \leftarrow \mathsf{Setup}(1^{\lambda}, 1^{\ell})$*, vector* $\boldsymbol{x} \in \mathcal{M}^{\ell}$ *and opening* $\pi \leftarrow \mathsf{FuncProve}(\mathsf{ck}, \mathsf{aux}, f)$*, it holds that* $|\pi| \leq p(\lambda) \cdot s_{\mathsf{FC}}(\ell, \kappa)$*.*

In order to model and compare different constructions, the notion of succinctness that we introduce is parametric with respect to a function $s_{\mathsf{FC}}(\ell, \kappa)$ that depends on the input-output length ℓ and some parameter κ of the evaluated function. To give some examples, κ could be an integer expressing the depth/size of a circuit (and thus $\mathcal{F}_{\kappa}$ are all circuits of depth/size κ), the degree of a polynomial, or the running time of a Turing machine. Accordingly, $\mathcal{K}$ is a set that partitions the class of admissible functions, e.g., $\mathcal{K} = [D]$ if the admissible functions are all circuits of depth $\leq D$, or $\mathcal{K} = \mathbb{N}$ if one wants to capture circuits of any depth.

Remark 4.2. *In Chapter 7, we express FC succinctness by a function* s_{FC} *such that* $|\pi| \leq s_{\mathsf{FC}}(\lambda, \ell, \ell_y, |f|)$ *and where* $|f|$ *denotes the size of the circuit description of* f*. This is done for clarity, as we will not need to partition the class of functions* $\mathcal{F}$ *into subclasses* $\mathcal{F}_{\kappa}$*.*

One can also express succinctness asymptotically, such that for any admissible set of parameters, $|\pi| \leq \mathsf{poly}(\lambda, \log \ell, \log \ell_y, o(|f|))$*.*

The natural security definition of FCs, proposed in [LRY16], is called evaluation binding and says that a PPT adversary cannot open a commitment to two distinct outputs for the same function.

Definition 4.3 (Evaluation Binding). *For any PPT adversary* $\mathcal{A}$*, the following probability is* $\mathsf{negl}(\lambda)$*:*

$$\Pr\left[\begin{array}{l} \quad\mathsf{FuncVer}(\mathsf{ck}, \mathsf{com}, \boldsymbol{y}, f, \pi) = 1 \\ \wedge\ \mathsf{FuncVer}(\mathsf{ck}, \mathsf{com}, \boldsymbol{y}', f, \pi') = 1 \\ \wedge\ \boldsymbol{y} \neq \boldsymbol{y}' \end{array} \middle| \begin{array}{l} \mathsf{ck} \leftarrow \mathsf{Setup}(1^{\lambda}, 1^{\ell}) \\ \begin{pmatrix} \mathsf{com}, & f, \\ \boldsymbol{y}, & \pi, \\ \boldsymbol{y}', & \pi' \end{pmatrix} \leftarrow \mathcal{A}(\mathsf{ck}) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

For simplicity of presentation, in all our security definitions, we omit checking the domains of the elements returned by the adversary, e.g., that $f \in \mathcal{F}$ and $\boldsymbol{y} \in \mathcal{M}^{\ell}$ etc.

We show that evaluation binding implies the classical binding notion.

Proposition 4.4. *Let* FC *be an FC scheme satisfying evaluation binding. Then* $\mathsf{FC.Com}$ *is a computationally binding commitment scheme, namely any PPT adversary has probability* $\mathsf{negl}(\lambda)$ *of finding a tuple* $(\boldsymbol{x}, r, \boldsymbol{x}', r')$ *such that* $\boldsymbol{x} \neq \boldsymbol{x}'$ *and* $\mathsf{Com}(\mathsf{ck}, \boldsymbol{x}; r) = \mathsf{Com}(\mathsf{ck}, \boldsymbol{x}'; r')$*.*

Proof. The proof is rather simple and works as follows. Consider an adversary $\mathcal{A}$ that returns $(\boldsymbol{x}, r, \boldsymbol{x}', r')$ such that $\boldsymbol{x} \neq \boldsymbol{x}'$ and $\mathsf{Com}(\mathsf{ck}, \boldsymbol{x}; r) = \mathsf{Com}(\mathsf{ck}, \boldsymbol{x}'; r')$ with non-negligible probability. Then we can use it to build an adversary $\mathcal{B}$ that returns $(\mathsf{com}, f, \boldsymbol{y}, \pi, \boldsymbol{y}', \pi')$ such that $\mathsf{FuncVer}(\mathsf{ck}, \mathsf{com}, f, \boldsymbol{y}, \pi) = \mathsf{FuncVer}(\mathsf{ck}, \mathsf{com}, f, \boldsymbol{y}', \pi') = 1$ and $\boldsymbol{y} \neq \boldsymbol{y}'$. To do so, $\mathcal{B}$ runs $\mathcal{A}$ and then looks for a function f such that $\boldsymbol{y} = f(\boldsymbol{x}) \neq f(\boldsymbol{x}') = \boldsymbol{y}'$, and computes

$(\mathsf{com}, \mathsf{aux}) \leftarrow \mathsf{Com}(\mathsf{ck}, \boldsymbol{x}; r)$, $(\mathsf{com}', \mathsf{aux}') \leftarrow \mathsf{Com}(\mathsf{ck}, \boldsymbol{x}'; r')$, $\pi \leftarrow \mathsf{FuncProve}(\mathsf{ck}, \mathsf{aux}, f)$, $\pi' \leftarrow \mathsf{FuncProve}(\mathsf{ck}, \mathsf{aux}', f)$. By the correctness of FC, π and π' must verify for $\boldsymbol{y}$ and $\boldsymbol{y}'$ respectively, and for the same commitment $\mathsf{com} = \mathsf{com}'$ (due to the break of binding by $\mathcal{A}$). Therefore, $\mathcal{B}$'s output is a valid attack against evaluation binding. □

4.3.1 Additional Properties of FCs

Here we define several extra properties of functional commitments that can be useful in applications.

Efficient Amortized Verification. An FC with this property enables the verifier to precompute a verification key ck_f associated to the function f, with which they can check any opening for f in time asymptotically faster than running f.

Definition 4.5 (FC Efficient Verification). *A functional commitment admits efficient verification if there exists a pair of algorithms:*

$\mathsf{PreFuncVer}(\mathsf{ck}, f) \to \mathsf{ck}_f$ *on input the commitment key* ck *and a function* $f \in \mathcal{F}$, *outputs a commitment key for the function* ck_f.

$\mathsf{EffFuncVer}(\mathsf{ck}, \mathsf{com}, \boldsymbol{y}, \mathsf{ck}_f, \pi) \to b \in \{0, 1\}$ *on input the commitment key* ck, *a commitment* com, *an output* $\boldsymbol{y}$, *a functional opening proof* π, *and a commitment key* ck_f *for a function* $f \in \mathcal{F}$, *accepts* ($b = 1$) *or rejects* ($b = 0$).

Furthermore, for any $\ell = \mathsf{poly}(\lambda)$, *function* $f : \mathcal{M}^\ell \to \mathcal{M}^{\ell_y}$ *s.t.* $f \in \mathcal{F}$, *any honestly generated commitment key* $\mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell)$, *vector* $\boldsymbol{x} \in \mathcal{M}^\ell$, *commitment* $(\mathsf{com}, \mathsf{aux}) \leftarrow \mathsf{Com}(\mathsf{ck}, \boldsymbol{x})$ *and opening* $\pi \leftarrow \mathsf{FuncProve}(\mathsf{ck}, \mathsf{aux}, f)$, *it holds:*

- $\mathsf{EffFuncVer}(\mathsf{PreFuncVer}(\mathsf{ck}, f), \mathsf{com}, \boldsymbol{y}, \pi) = \mathsf{FuncVer}(\mathsf{ck}, \mathsf{com}, f, \boldsymbol{y}, \pi)$,
- ck_f *is succinct, i.e.* $|\mathsf{ck}_f| \leq p(\lambda) \cdot s_{\mathsf{FC}}(\ell, \kappa)$, *and* $\mathsf{FC.EffVer}(\mathsf{ck}, \mathsf{com}, \boldsymbol{y}, \mathsf{ck}_f, \pi)$ *runs in time* $\leq p(\lambda) \cdot s_{\mathsf{FC}}(\ell, \kappa) + \mathsf{poly}(\lambda, \ell_y)$.[11]

Hiding and Zero Knowledge. An FC is hiding if the commitments produced through Com are hiding (see Theorem 3.1). For zero-knowledge, the goal is that the openings produced by $\mathsf{FuncProve}$ should not reveal more information about the committed vector beyond what can be deduced from the output, i.e., that $\boldsymbol{x}$ is such that $\boldsymbol{y} = f(\boldsymbol{x})$. We include the formal definition, which we take from [CFT22].

Definition 4.6 (Zero-knowledge openings). *An FC has perfect (resp. statistical, computational) zero-knowledge openings if there is a simulator* $\mathsf{Sim} = (\mathsf{Sim}_{\mathsf{Setup}}, \mathsf{Sim}_{\mathsf{Com}}, \mathsf{Sim}_{\mathsf{Equiv}}, \mathsf{Sim}_{\mathsf{Open}})$ *such that*

- *(i)* $\mathsf{Sim}_{\mathsf{Setup}}$ *generates indistinguishable keys, along with a trapdoor, i.e., the distributions* $\{\mathsf{ck} : \mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell)\}$ *and* $\{\mathsf{ck} : (\mathsf{ck}, \mathsf{td}) \leftarrow \mathsf{Sim}_{\mathsf{Setup}}(1^\lambda, \ell)\}$ *are identical (resp. statistically, computationally indistinguishable).*

[11]The term $\mathsf{poly}(\lambda, m)$ appears since the EffVer algorithm needs to at least read the output $\boldsymbol{y}$, that has length ℓ_y.

- *(ii) for any vector* $\boldsymbol{x} \in \mathcal{M}^\ell$, *keys* $(\mathsf{ck}, \mathsf{td}) \leftarrow \mathsf{Sim}_{\mathsf{Setup}}(1^\lambda, \ell)$, *functions* $f_1, \dots, f_Q \in \mathcal{F}$, *and commitments* $(\mathsf{com}, \mathsf{aux}) \leftarrow \mathsf{Com}(\mathsf{ck}, \boldsymbol{x})$ *and* $(\widetilde{\mathsf{com}}, \widetilde{\mathsf{aux}}) \leftarrow \mathsf{Sim}_{\mathsf{Com}}(\mathsf{ck})$, *the following two distributions are identical (resp. statistically, computationally indistinguishable):*

$$(\widetilde{\mathsf{com}}, \{\mathsf{Sim}_{\mathsf{Open}}(\mathsf{td}, \widetilde{\mathsf{aux}}, \widetilde{\mathsf{com}}, f_j, f_j(\boldsymbol{x}))\}_{j=1}^{Q}) \approx (\mathsf{com}, \{\mathsf{FuncProve}(\mathsf{ck}, \mathsf{aux}, f_j)\}_{j=1}^{Q})$$

We state a simple result showing that an FC with hiding commitments (but not necessarily zero-knowledge functional openings) can be converted, via the use of a NIZK scheme, into one that also achieves zero-knowledge functional openings.

Theorem 4.7. *Let* FC *be an FC scheme that satisfies com-hiding (Definition 3.1), and let* Π *be a knowledge-sound NIZK for the NP relation* $R_{\mathsf{FC}} = \{((\mathsf{ck}, \mathsf{com}, f, \boldsymbol{y}); \pi) : \mathsf{FuncVer}(\mathsf{ck}, \mathsf{com}, f, \boldsymbol{y}, \pi) = 1\}$. *Then there exists an FC scheme* FC^* *for the same class of functions supported by* FC *that has com-hiding and zero-knowledge functional openings. Furthermore, if* FC *is additive-homomorphic, so is* FC^**; if* FC *has efficient verification and* Π *supports* $R'_{\mathsf{FC}} = \{(\mathsf{ck}_f, \mathsf{com}, \boldsymbol{y}; \pi) : \mathsf{EffVer}(\mathsf{ck}_f, \mathsf{com}, \boldsymbol{y}, \pi) = 1\}$, *then* FC^* *has also efficient verification.*

Proof. Let $\Pi = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{FuncVer}, \mathsf{Sim})$ be a NIZK for the relation $R_{\mathsf{FC}} = \{((\mathsf{ck}, C, f, \boldsymbol{y}); \pi) : \mathsf{FuncVer}(\mathsf{ck}, \mathsf{com}, f, \boldsymbol{y}, \pi) = 1\}$. Then, we can construct a FC scheme FC^* that satisfies com-hiding and zero knowledge openings as follows:

- $\mathsf{FC}^*.\mathsf{Setup}(1^\lambda)$ runs $\mathsf{ck} \leftarrow \mathsf{FC}.\mathsf{Setup}(1^\lambda)$ and $\mathsf{crs} \leftarrow \Pi.\mathsf{Setup}(1^\lambda)$, and outputs $(\mathsf{ck}, \mathsf{crs})$.
- $\mathsf{FC}^*.\mathsf{Com}((\mathsf{ck}, \mathsf{crs}), \boldsymbol{x}; r)$ directly outputs $(\mathsf{com}, \mathsf{aux}) \leftarrow \mathsf{FC}.\mathsf{Com}(\mathsf{ck}, \boldsymbol{x}; r)$ (note that $\mathsf{FC}.\mathsf{Com}$ is com-hiding).
- $\mathsf{FC}^*.\mathsf{FuncProve}((\mathsf{ck}, \mathsf{crs}), \mathsf{aux}, f)$ runs $\pi \leftarrow \mathsf{FC}.\mathsf{FuncProve}(\mathsf{ck}, \mathsf{aux}, f)$ and then outputs $\pi^* \leftarrow \Pi.\mathsf{Prove}(\mathsf{crs}, (\mathsf{ck}, \mathsf{com}, f, \boldsymbol{y}), \pi)$.
- $\mathsf{FC}^*.\mathsf{FuncVer}((\mathsf{ck}, \mathsf{crs}), \mathsf{com}, f, \boldsymbol{y}, \pi^*)$ outputs $b \leftarrow \Pi.\mathsf{FuncVer}(\mathsf{crs}, (\mathsf{ck}, \mathsf{com}, f, \boldsymbol{y}), \pi^*)$.

Additive homomorphism and efficient verification of FC^* follow from the respective properties of FC.

Zero knowledge follows from the zero knowledge property of the NIZK, since we can construct a simulator $\mathsf{FC}^*.\mathsf{Sim}$ given the NIZK simulator $\Pi.\mathsf{Sim}$ as follows. $\mathsf{FC}^*.\mathsf{Sim}_{\mathsf{Com}}$, $\mathsf{FC}^*.\mathsf{Sim}_{\mathsf{Equiv}}$ are the same as their respective FC com-hiding simulators. $\mathsf{FC}^*.\mathsf{Sim}_{\mathsf{Setup}}$ runs both $\mathsf{FC}.\mathsf{Sim}_{\mathsf{Setup}}$ and the NIZK simulator $\Pi.\mathsf{Sim}_{\mathsf{Prove}}$ and outputs simulated $\widetilde{\mathsf{crs}}, \widetilde{\mathsf{ck}}$ and respective trapdoors. Finally, $\mathsf{FC}^*.\mathsf{Sim}_{\mathsf{Open}}$ runs the NIZK simulator $\Pi.\mathsf{Sim}_{\mathsf{Prove}}$ on the simulated $\widetilde{\mathsf{crs}}$ and its trapdoor.

Evaluation binding for FC^* requires the knowledge soundness of the NIZK. Namely, given two proofs $\pi^*, \pi^{*\prime}$ for different outputs $\boldsymbol{y}, \boldsymbol{y}'$ and the same commitment com, one can run the NIZK extractor to obtain π, π' from $\pi^*, \pi^{*\prime}$. Then, it is possible to make a reduction to the security (evaluation binding) of FC. □

4.4 Chainable Functional Commitments

As described in the technical overview, we introduce the notion of Chainable Functional Commitments (CFC), which is an extension of the FC primitive that allows one to "chain" multiple openings to different functions. Here, we follow and extend the original syntax from [BCFL23] to allow each of $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m, \boldsymbol{y}$ to be indexed by a different index set, this is, to support subvectors as in the definition of commitments (Theorem 3.1).

Definition 4.8 (Chainable Functional Commitments (CFC)). *A chainable functional commitment (CFC) for a class of functions $\mathcal{F} \subseteq \{f : \mathcal{M}^{m\cdot\ell} \to \mathcal{M}^{\ell}\}$ and index set family $\mathcal{J}$ consists of a commitment scheme* ($\mathsf{Setup}, \mathsf{Com}$) *for $\mathcal{J}$ and additionally PPT algorithms* ($\mathsf{FuncProve}, \mathsf{FuncVer}$) *with the following syntax:*

$\mathsf{FuncProve}(\mathsf{ck}, (\boldsymbol{x}_i)_{i\in[m]}, f) \to \pi$: *given vectors $\boldsymbol{x}_i \in \mathcal{M}^{J_i}$ for $i \in [m]$ and a function $f \in \mathcal{F}$ where $f : \prod_{i\in[m]} \mathcal{M}^{J_i} \to \mathcal{M}^{J_y}$, returns a functional opening proof π.*

$\mathsf{FuncVer}(\mathsf{ck}, (\mathsf{com}_i)_{i\in[m]}, \mathsf{com}_y, f, \pi) \to b \in \{0,1\}$: *on input commitments $(\mathsf{com}_i)_{i\in[m]}$ for $i \in [m]$ to the m inputs and com_y to the output, opening proof π, and function $f \in \mathcal{F}$ where $f : \prod_{i\in[m]} \mathcal{M}^{J_i} \to \mathcal{M}^{J_y}$, accepts ($b = 1$) or rejects ($b = 0$).*

A CFC must satisfy the following properties.

Correctness. *For $\lambda, \ell, m, \in \mathbb{N}$, $J_1, \ldots, J_m, J_y \in \mathcal{J}$, $f \in \mathcal{F}$ where $f : \prod_{i\in[m]} \mathcal{M}^{J_i} \to \mathcal{M}^{J_y}$, and $\boldsymbol{x}_i \in \mathcal{M}^{J_i}$ for $i \in [m]$, it holds that*

$$\Pr\left[\mathsf{FuncVer}(\mathsf{ck}, (\mathsf{com}_i)_{i\in[m]}, \mathsf{com}_y, f, \pi) = 1 \;\middle|\; \begin{array}{l} \mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell) \\ \mathsf{com}_i \leftarrow \mathsf{Com}(\mathsf{ck}, J_i, \boldsymbol{x}_i) \;\; \forall i \in [m] \\ \mathsf{com}_y \leftarrow \mathsf{Com}(\mathsf{ck}, J_y, f(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m)) \\ \pi \leftarrow \mathsf{FuncProve}(\mathsf{ck}, (\boldsymbol{x}_i)_{i\in[m]}, f) \end{array} \right] = 1.$$

Succinctness. *We define succinctness for CFCs analogously to succinctness for FCs. For any admissible set of parameters as before, $|\pi| \le p(\lambda) \cdot s_{\mathsf{CFC}}(\ell, m, \kappa)$.*

As in the case of FCs (Definition 4.1) we define succinctness in a parametric way, and we are interested in CFC constructions supporting non-trivial functions $s_{\mathsf{CFC}}(\ell, m, \kappa)$ that are sublinear or constant in ℓ, m. We introduce the CFC security notion below, which is analogous to Theorem 4.3.

Definition 4.9 (CFC Evaluation Binding). *A CFC satisfies evaluation binding if for any PPT adversary $\mathcal{A}$,*

$$\Pr\left[\begin{array}{l} \quad \mathsf{FuncVer}(\mathsf{ck}, (\mathsf{com}_i)_{i\in[m]}, \mathsf{com}_y, f, \pi) = 1 \\ \wedge\ \mathsf{FuncVer}(\mathsf{ck}, (\mathsf{com}_i)_{i\in[m]}, \mathsf{com}'_y, f, \pi') = 1 \\ \wedge\ \mathsf{com}_y \ne \mathsf{com}'_y \end{array} \;\middle|\; \begin{array}{l} \mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell) \\ \begin{pmatrix} (\mathsf{com}_i)_{i\in[m]}, & f, \\ \mathsf{com}_y, & \pi, \\ \mathsf{com}'_y, & \pi' \end{pmatrix} \leftarrow \mathcal{A}(\mathsf{ck}) \end{array} \right] \le \mathsf{negl}(\lambda).$$

As one can notice, the above notion of evaluation binding can only hold in the case when the output commitments com_y are generated deterministically. This is still enough for using CFCs to construct FCs with hiding commitments to inputs and zero-knowledge openings (thanks to Theorem 4.7). Looking ahead, evaluation binding for randomized (e.g. hiding) CFCs may be defined if a "projective space" is embedded in the commitment key, such as in the projective commitments defined in Chapter 6.

Definition 4.10 (CFC Efficient Verification)**.** *A CFC admits efficient verification with preprocessing if there exists a pair of algorithms:*

$\mathsf{PreFuncVer}(\mathsf{ck}, f) \to \mathsf{ck}_f$ *on input the commitment key* ck *and a function* $f \in \mathcal{F}$ *where* $f : \prod_{i\in[m]} \mathcal{M}^{J_i} \to \mathcal{M}^{J_y}$*, outputs a function key* ck_f *of size* $|\mathsf{ck}_f| \leq p(\lambda) \cdot s_{\mathsf{CFC}}(\ell, m, \kappa)$*.*

$\mathsf{EffFuncVer}(\mathsf{ck}_f, (\mathsf{com}_i)_{i\in[m]}, \mathsf{com}_y, \pi) \to b \in \{0, 1\}$ *on input a function key* ck_f*, commitments* $(\mathsf{com}_i)_{i\in[m]}$ *to the* m *inputs and* com_y *to the output, and an opening proof* π*, accepts* $(b = 1)$ *or rejects* $(b = 0)$ *in time bounded by* $p(\lambda) \cdot s_{\mathsf{CFC}}(\ell, m, \kappa)$

Furthermore, the following function equivalence holds:

$$\mathsf{FuncVer}(\mathsf{ck}, (\mathsf{com}_i)_{i\in[m]}, \mathsf{com}_y, f, \pi) \equiv \mathsf{EffFuncVer}(\mathsf{PreFuncVer}(\mathsf{ck}, f), (\mathsf{com}_i)_{i\in[m]}, \mathsf{com}_y, \pi).$$

4.5 FC for Circuits from CFC for Quadratic Polynomials

In this section we introduce a generic construction of a Functional Commitment scheme for arithmetic circuits of bounded width ℓ, from any Chainable Functional Commitment for quadratic functions over inputs of length ℓ.

4.5.1 Circuit Model and Notation

Let $\mathcal{R}$ be a commutative ring. We consider arithmetic circuits $C : \mathcal{R}^\ell \to \mathcal{R}^\ell$ where every gate is a quadratic polynomial with bounded coefficients. It is not hard to see that such a model captures the more common model of arithmetic circuits consisting of fan-in-2 gates that compute either addition or multiplication.

More in detail, we model C as a directed acyclic graph (DAG) where every node is either an *input*, an *output* or a *gate*, and input (resp. output) nodes have in-degree (resp. out-degree) 0. We partition the nodes in the DAG defined by C in *levels* as follows. Level 0 contains all the input nodes. Let the *depth* of a gate g be the length of the longest path from any input to g, in the DAG defined by the circuit. Then, for $h \geq 1$, we define level h as the subset of gates of depth h. Note that any gate in level h has *at least* one input coming from a gate at level $h - 1$ (while other inputs may come from gates at any other previous level $0, \ldots, h - 2$). The *depth* of the circuit C, denoted d_C (or simply d when clear from the context), is the number of levels of C. Finally, we assume that the last level d_C also contains output nodes.[12]

[12]This can be assumed without loss of generality. If we have an output $x_i^{(h)}$ at level $h < d$, we can introduce a linear gate at level d that takes $x_i^{(h)}$ and some arbitrary $x_j^{(d-1)}$ as input, and outputs $x_k^{(d)} = x_i^{(h)} + 0 \cdot x_j^{(d-1)}$.

In this model, we define the *width* of C, denoted by ℓ, as the maximum number of nodes in any level $h = 0$ to d_C. Note that the width upper bounds the input length. For simplicity, we assume without loss of generality circuits with maximal ℓ inputs and ℓ gates in every level.

When we evaluate C on an input $\boldsymbol{x}$, we denote the input values by $\boldsymbol{x}^{(0)}$, and the outputs of the gates in level h by the vector $\boldsymbol{x}^{(h)}$. We note that, for every $k \in [\ell]$, the output of the k-th gate in level h can be defined as $x_k^{(h)} = f_k^{(h)}(\boldsymbol{x}^{(0)}, \ldots, \boldsymbol{x}^{(h-1)})$ where $f_k^{(h)} : \mathcal{R}^{\ell h} \to \mathcal{R}$ is a quadratic polynomial. We group all these ℓ polynomials $f_1^{(h)}, \ldots, f_\ell^{(h)}$ into the quadratic polynomial map $f^{(h)} : \mathcal{R}^{\ell h} \to \mathcal{R}^\ell$ such that $\boldsymbol{x}^{(h)} = f^{(h)}(\boldsymbol{x}^{(0)}, \ldots, \boldsymbol{x}^{(h-1)})$. We denote the operation that extracts these functions $\{f^{(h)}\}$ from C by $(f^{(1)}, \ldots, f^{(d)}) \leftarrow \mathsf{Parse}(C)$.

4.5.2 Quadratic Functions

As we mentioned above, a gate in our circuit model computes a quadratic polynomial. Thus all the gates at a given level form a vector of ℓ quadratic polynomials that take up to $m = \mathsf{poly}(\lambda)$ vectors and output a single vector. We define this class of functions as

$$\mathcal{F}_{\mathsf{quad}} = \{f : \mathcal{R}^{\ell m} \to \mathcal{R}^\ell \ : f = (f_k)_{k \in [\ell]} \wedge \forall k \in [\ell] \, f_k \in \mathcal{R}[X_1^{(1)}, \ldots, X_\ell^{(m)}]^{\leq 2}\}.$$

A quadratic polynomial map $f \in \mathcal{F}_{\mathsf{quad}}$, $f : \mathcal{R}^{mn} \to \mathcal{R}^\ell$, such as those representing the computation done at a given level of a circuit, can be expressed in a compact form. For $f(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}) = \boldsymbol{y}$, we can define d matrices $\mathbf{F}^{(h)} \in \mathcal{R}^{\ell \times \ell}$, $d(d+1)/2$ matrices $\mathbf{G}^{(h,h')} \in \mathcal{R}^{\ell \times \ell^2}$, and a vector $\boldsymbol{e} \in \mathbb{F}^\ell$ such that

$$f(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}) = \boldsymbol{e} + \sum_{h \in \mathcal{S}_1(f)} \mathbf{F}^{(h)} \cdot \boldsymbol{x}^{(h)} + \sum_{(h,h') \in \mathcal{S}_2^{\otimes}(f)} \mathbf{G}^{(h,h')} \cdot (\boldsymbol{x}^{(h)} \otimes \boldsymbol{x}^{(h')}). \tag{4.3}$$

The sets $\mathcal{S}_1(f)$ and $\mathcal{S}_2^{\otimes}(f)$ are the *linear support* and the *quadratic support* of f that we define below; for now $\mathcal{S}_1 = [m]$, $\mathcal{S}_2^{\otimes} = \{(h, h') \in [m] \times [m] : h \leq h'\}$.[13]

We note that, in an arbitrary circuit, the function $f^{(h)}$ at each level may depend on values from *any* previous level, but not necessarily from all of them. To capture such connectivity precisely, we define the *linear support* of $f \in \mathcal{F}_{\mathsf{quad}}$, denoted $\mathcal{S}_1(f) \subseteq [m]$, as the set of indices h where the linear part of f is nonzero with respect to any term $X_i^{(h)}$. Formally,

$$\mathcal{S}_1(f) := \{h \in [m] : \mathbf{F}^{(h)} \neq \mathbf{0}\}.$$

Analogously, we define the *quadratic support* of f, denoted $\mathcal{S}_2(f) \subseteq [m]$, as the indices h where f is nonzero with respect to any term $X_i^{(h)} \cdot X_j^{(h')}$ for one or more $h' \in [m]$. Formally,

$$\mathcal{S}_2(f) := \{h \in [m] : \exists h' \, \mathbf{G}^{(h,h')} \neq \mathbf{0}\}.$$

We will also express the quadratic support using pairs of indices,

$$\mathcal{S}_2^{\otimes}(f) := \{(h, h') \in [m] \times [m] : h \leq h' \wedge \mathbf{G}^{(h,h)} \neq \mathbf{0}\}.$$

[13]This representation is not unique as $\boldsymbol{x}^{(h)} \otimes \boldsymbol{x}^{(h')}$ contains repeated entries, but this can be solved by agreeing on appropriately placing zero coefficients.

We also say $h \in \mathcal{S}_2(f)$ whenever $(t, h) \in \mathcal{S}_2^{\otimes}(f)$ or $(h, t) \in \mathcal{S}_2^{\otimes}(f)$ for some $t \in [m]$. Finally, we define the *support* of f as the union of its linear and quadratic supports, namely $\mathcal{S}(f) = \mathcal{S}_1(f) \cup \mathcal{S}_2(f)$. By using the linear and quadratic supports, we can express polynomial functions in $\mathcal{F}_{\mathsf{quad}}$ as follows:

$$f(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}) = \boldsymbol{e} + \sum_{h \in \mathcal{S}_1(f)} \mathbf{F}^{(h)} \cdot \boldsymbol{x}^{(h)} + \sum_{(h,h') \in \mathcal{S}_2^{\otimes}(f)} \mathbf{G}^{(h,h')} \cdot (\boldsymbol{x}^{(h)} \otimes \boldsymbol{x}^{(h')}). \qquad (4.4)$$

Consider a circuit C and let $(f^{(1)}, \ldots, f^{(d)}) \leftarrow \mathsf{Parse}(C)$. Then every function $f^{(h)}$ can be expressed and computed using only the inputs in $\mathcal{S}(f^{(h)})$, namely $f^{(h)}((\boldsymbol{x}^{(h')})_{h' \in \mathcal{S}(f^{(h)})}) = f^{(h)}(\boldsymbol{x}^{(0)}, \ldots, \boldsymbol{x}^{(h-1)})$.

We call the number of inputs in the support of $f^{(h)}$, namely $|\mathcal{S}(f^{(h)})|$, the *in-degree of level h*. We say that *a circuit C has in-degree t_C* if $t_C = \max_{h \in [d_C]} |\mathcal{S}(f^{(h)})|$. We call C a *layered circuit* if it has in-degree 1. Notice that for a layered circuit it holds that $\boldsymbol{x}^{(d)} = C(\boldsymbol{x}^{(0)})$ where $\boldsymbol{x}^{(h)} = f^{(h)}(\boldsymbol{x}^{(h-1)})$ for all $h = 1$ to d.

4.5.3 Our Compiler: from CFC to FC

Classes of circuits. To properly define the succinctness and the functions supported by our FC construction, we parametrize the circuits according to three parameters, the depth, the in-degree, and the width. Let $\mathcal{F}_{(d,t,w)} = \{C : \mathcal{R}^\ell \to \mathcal{R}^\ell : d_C = d, t_C = t, w_C = w\}$, where $d_C \in \mathbb{N}$, $t_C \le d$, $w_C \le w$ are the depth, in-degree, and width of C, respectively. Then our FC scheme supports any arithmetic circuit of width at most ℓ, in the model described above. We denote this class by $\mathcal{F}_n := \{\mathcal{F}_{(d,t,w)}\}_{d \in \mathbb{N}, t \le d, w \le \ell}$.

Construction. In Figure 4.1 we present our FC construction for $\mathcal{F}_n$. We assume, without loss of generality, that the auxiliary input aux generated by $\mathsf{CFC.Com}$ contains the committed input $\boldsymbol{x}$. In the protocol, we retrieve $\boldsymbol{x}$ from aux via a Parse function. Note that the same construction becomes a CFC for $\mathcal{F}_n$ if the verifier takes com_d as input and skips the recomputation of com_d in Figure 4.1.

Our goal in this section is to prove the following theorem.

Theorem 4.11. *Let* $\mathsf{CFC} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{FuncProve}, \mathsf{FuncVer})$ *be a chainable functional commitment scheme for the class of functions* $\mathcal{F}_{\mathsf{quad}}$. *Then, the scheme* FC *in Figure 4.1 is an FC for the class $\mathcal{F}_n$ of arithmetic circuits $C : \mathcal{R}^\ell \to \mathcal{R}^\ell$ of width $\le \ell$.*

Let $\mathcal{K}$ be a partitioning of $\mathcal{F}_{\mathsf{quad}}$ such that CFC *is $s(\ell, m, \kappa)$-succinct for $\mathcal{F}_{\mathsf{quad}} = \{\mathcal{F}_{\mathsf{quad},\kappa}\}$. Then* FC *is $d \cdot (s_{\max}(\ell, t) + 1)$-succinct for the class $\mathcal{F}_n = \{\mathcal{F}_{(d,t,w)}\}_{d \in \mathbb{N}, t \le d, w \le \ell}$, where $s_{\max}(\ell, t) := \max_{\kappa \in \mathcal{K}} s(\ell, t, \kappa)$. Moreover, given an additively homomorphic and/or efficiently verifiable* CFC, *so is* FC.

Proof. Correctness and additive homomorphism of FC follow immediately from the respective properties of CFC.

Succinctness. If CFC is $s(\ell, m, \kappa)$-succinct for the class of quadratic polynomials in $\mathcal{F}_{\mathsf{quad}} = \{\mathcal{F}_{\mathsf{quad},\kappa}\}$, then FC is $s'(\ell, (d, t))$-succinct for $\mathcal{F}_\ell = \{\mathcal{F}_{(d,t,\ell)}\}$ where $s'(\ell, (d, t)) =$

```
$\mathsf{FC.Setup}(1^\lambda, 1^\ell)$
return $\mathsf{CFC.Setup}(1^\lambda, 1^\ell)$
```

```
$\mathsf{FC.Com}(\mathsf{ck}, \boldsymbol{x})$
return $\mathsf{CFC.Com}(\mathsf{ck}, \boldsymbol{x})$
```

```
$\mathsf{FC.FuncProve}(\mathsf{ck}, \mathsf{aux}, C)$
$(f^{(1)}, \dots, f^{(d)}) \leftarrow \mathsf{Parse}(C)$
$\boldsymbol{x}^{(0)} \leftarrow \mathsf{Parse}(\mathsf{aux})$
for $h \in [d]$ :
    // Evaluate and commit to each level
    $\boldsymbol{x}^{(h)} \leftarrow f^{(h)}(\boldsymbol{x}^{(0)}, \boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(h-1)})$
    $(\mathsf{com}_h, \mathsf{aux}_h) \leftarrow \mathsf{CFC.Com}(\mathsf{ck}, \boldsymbol{x}^{(h)})$
    // Compute the opening for the level
    $\pi_h \leftarrow \mathsf{CFC.FuncProve}(\mathsf{ck},$
        $(\mathsf{aux}_{h'})_{h' \in \mathcal{S}(f^{(h)})}, f^{(h)})$
return $(\pi_1, \dots, \pi_d, \mathsf{com}_1, \dots, \mathsf{com}_{d-1})$
```

```
$\mathsf{FC.FuncVer}(\mathsf{ck}, \mathsf{com}, C, \boldsymbol{y}, \pi)$
$(f^{(1)}, \dots, f^{(d)}) \leftarrow \mathsf{Parse}(C)$
$\mathsf{com}_0 \leftarrow \mathsf{com}$
$(\pi_1, \dots, \pi_d, \mathsf{com}_1, \dots, \mathsf{com}_{d-1}) \leftarrow \pi$
// Recompute commitment to output
$\mathsf{com}_d \leftarrow \mathsf{CFC.Com}(\mathsf{ck}, \boldsymbol{y})$
for $h \in [d]$ :
// Verify all proofs
    $b_h \leftarrow \mathsf{CFC.FuncVer}(\mathsf{ck},$
        $(\mathsf{com}_{h'})_{h' \in \mathcal{S}(f^{(h)})}, \mathsf{com}_h, f^{(h)}, \pi_h)$
return $b_1 \wedge \dots \wedge b_d$
```

Figure 4.1: *Construction of our FC for circuits from a CFC for the class $\mathcal{F}_{\mathsf{quad}}$. For notational succinctness, we let $\mathcal{S}_h := \mathcal{S}(f^{(h)})$.*

$d \cdot (s_{\max}(\ell, t) + 1)$. Indeed, $\mathsf{FC.FuncProve}$ produces $d - 1$ commitments com_h for $h \in [d-1]$, each of them having size bounded by a fixed polynomial $p(\lambda) = \mathsf{poly}(\lambda)$. Besides, it generates d CFC evaluation proofs π_h, each of them involving $|\mathcal{S}(f^{(h)})| \le t$ input commitments, and thus having size $\le p(\lambda) \cdot s(\ell, |\mathcal{S}(f^{(h)})|, \kappa) \le p(\lambda) \cdot s_{\max}(\ell, t)$. Hence, we can bound the size of an $\mathsf{FC.FuncProve}$ proof by $|\pi| \le p(\lambda) \cdot d \cdot (s_{\max}(\ell, t) + 1)$. A particularly relevant case is that for layered circuits we obtain $|\pi| \le p(\lambda) \cdot d \cdot (s_{\max}(\ell, 1) + 1)$.

We obtain better succinctness by using a slightly different, yet general, circuit model. To keep the presentation of the main scheme simpler, we present this optimization in Section 4.5.4. The proof size reduction is specific to our CFC construction from pairings (see Section 4.6.5 for the resulting efficiency).

Efficient verification. If CFC has amortized efficient verification (Definition 4.5), we can set $\mathsf{FC.VerPrep}(\mathsf{ck}, f)$ to obtain $\mathsf{vk}_h \leftarrow \mathsf{CFC.VerPrep}(\mathsf{ck}, f^{(h)})$ for $h \in [d]$ and output $\mathsf{vk}_f := (\mathsf{vk}_1, \dots, \mathsf{vk}_d)$. Then, $\mathsf{FC.EffVer}$ simply recomputes the commitment to the output com_d and runs $\mathsf{CFC.EffVer}$ for each circuit level. Let T_{CFC} be largest of the running times of $\mathsf{CFC.FuncVer}$ for all CFC instances in the FC construction, and let T_{Com} be the running time of $\mathsf{CFC.Com}$. Then, the running time of $\mathsf{FC.FuncVer}$ is $T_{\mathsf{FC}} \le d \cdot T_{\mathsf{CFC}} + T_{\mathsf{Com}}$. As the running time of $\mathsf{CFC.EffVer}$ is $o(T_{\mathsf{CFC}})$, the running time of $\mathsf{FC.EffVer}$ is $d \cdot o(T_{\mathsf{CFC}}) + T_{\mathsf{com}}$, which is $o(T_{\mathsf{FC}})$ whenever $T_{\mathsf{Com}} = o(d \cdot T_{\mathsf{CFC}})$. Usually, $T_{\mathsf{com}} = O(|\boldsymbol{y}|)$ (and in fact $T_{\mathsf{com}} = \Omega(|\boldsymbol{y}|)$) where $|\boldsymbol{y}| \le \ell$ is the length of the committed vector. Hence, in practice FC has amortized efficient verification unless $d = O(|C|)$, a case in which the proof size also becomes very large.

We remark that for both our pairing-based and lattice-based CFC instances, the running time of $\mathsf{FC.EffVer}$ is actually bounded by $p(\lambda) \cdot (|\boldsymbol{y}| + |\pi|)$ where $p(\lambda) = \mathsf{poly}(\lambda)$, which is optimal since the verifier at least needs to parse the proof and the output.

Security. In Lemma 4.12, we prove that if CFC is evaluation binding, then FC is evaluation binding. We remark that in the full version of [BCFL23] we show an analogous result for knowledge extractability.

Lemma 4.12. *If* CFC *is evaluation binding (Theorem 4.9), then our FC construction for arbitrary circuits is also evaluation binding.*

Proof. Consider an adversary $\mathcal{A}$ who returns a tuple $(\mathsf{com}, C, \boldsymbol{y}, \pi, \boldsymbol{y}', \pi')$ that breaks evaluation binding, and parse the proofs as follows

$$\pi := (\pi_1, \dots, \pi_d, \mathsf{com}_1, \dots, \mathsf{com}_{d-1}), \quad \pi' := (\pi'_1, \dots, \pi'_d, \mathsf{com}'_1, \dots, \mathsf{com}'_{d-1})$$

We will show that, if both proofs π and π' verify for $\boldsymbol{y}$ and $\boldsymbol{y}'$ respectively, with $\boldsymbol{y} \neq \boldsymbol{y}'$, then we can construct an adversary $\mathcal{B}$ against the evaluation binding of the CFC. We construct $\mathcal{B}$ as follows.

First, $\mathcal{B}$ is given a commitment key ck and calls $\mathcal{A}(\mathsf{ck})$ to obtain the output $(\mathsf{com}, C, \boldsymbol{y}, \pi, \boldsymbol{y}', \pi')$. Then, $\mathcal{B}$ obtains the commitments to the outputs $\mathsf{com}_y \leftarrow \mathsf{Com}(\mathsf{ck}, \boldsymbol{y})$ and $\mathsf{com}_{y'} \leftarrow \mathsf{Com}(\mathsf{ck}, \boldsymbol{y}')$.

If $\mathsf{com}_y = \mathsf{com}_{y'}$, then $\mathcal{B}$ can break the binding property of the commitment (and hence evaluation binding due to Proposition 4.4), since com_y opens to different $\boldsymbol{y} \neq \boldsymbol{y}'$.

Hence, let us assume $\mathsf{com}_y \neq \mathsf{com}'_y$, and denote $\mathsf{com}_0 = \mathsf{com}'_0 = \mathsf{com}$. Then, look at both proofs produced by $\mathcal{A}$ and set $1 \leq h^* \leq d$ to be the smallest index such that $\mathsf{com}_{h^*} \neq \mathsf{com}'_{h^*}$ and $\mathsf{com}_h = \mathsf{com}'_h$ for all $h = 0$ to $h^* - 1$. Notice that such index *must exist* since, at least, we have $\mathsf{com}_0 = \mathsf{com}'_0$ and $\mathsf{com}_d = \mathsf{com}_y \neq \mathsf{com}_{y'} = \mathsf{com}'_d$.

Then, $\mathcal{B}$ breaks evaluation binding of CFC by outputting $((\mathsf{com}_h)_{h \in \mathcal{S}(f^{(h^*)})}, f^{(h^*)}, \mathsf{com}_{h^*}, \pi_{h^*}, \mathsf{com}'_{h^*}, \pi'_{h^*})$. □

□

4.5.4 Efficiency Tradeoffs

We describe optimization strategies for our FC construction. Our main goals are to reduce the proof size in many cases, and to support circuits of larger width than initially specified at setup time.

A refined circuit model. We introduce a variant of our circuit model that results in a notable reduction of the proof size of our pairing-based CFC in Section 4.6. The new circuit model differs from the previous model in that here every quadratic monomial of each polynomial gate $f_k^{(h)}$ at level h is assumed to take at least one of its inputs from level $h-1$. In particular, the quadratic term of functions $f_k^{(h)}(\boldsymbol{x}^{(0)}, \dots, \boldsymbol{x}^{(h-1)})$ is a linear combination of all products of variables $x_i^{(h-1)} \cdot x_j^{(h')}$, $\forall i, j \in [\ell]$, at levels $h-1$ and h' such that $0 \leq h' \leq h-1$.

This circuit model also generalizes the standard arithmetic circuit model with fan-in 2 additive or multiplicative gates. We denote the class of functions in the levels of the new model by $\mathcal{F}_{\mathsf{level}} \subset \mathcal{F}_{\mathsf{quad}}$, which we define as

$$\mathcal{F}_{\mathsf{level}} = \{f \in \mathcal{F}_{\mathsf{quad}} : \mathcal{S}_2^{\otimes}(f) \subseteq \{(h', m) \in [m] \times \{m\}\}\}.$$

Note that we can extend any parametrization $\mathcal{F}_{\mathsf{quad}} = \{\mathcal{F}_{\mathsf{quad},\kappa}\}$ to $\mathcal{F}_{\mathsf{level}} = \{\mathcal{F}_{\mathsf{level},\kappa}\}$ by setting $\mathcal{F}_{\mathsf{level},\kappa} := \mathcal{F}_{\mathsf{level}} \cap \mathcal{F}_{\mathsf{quad},\kappa}$. The main advantage of this model is that for any $f \in \mathcal{F}_{\mathsf{level}}$, $\left|\mathcal{S}_2^{\otimes}(f)\right| \leq m$, instead of being $\leq m^2$ in the more general case. When switching to this model, it is sufficient to instantiate our FC construction with a CFC scheme that only supports quadratic functions in $\mathcal{F}_{\mathsf{level}}$ and not all $\mathcal{F}_{\mathsf{quad}}$.

Reducing proof size. Assume that we want to evaluate a circuit C of width w and depth d that is densely interconnected (i.e. the in-degree $t = O(d)$) when our commitment key ck supports circuits of width up to $\ell > w$. We present an optimization that reduces the proof size of our FC scheme.

Proposition 4.13. *Let* CFC *be a* $s(\ell, m, \kappa)$*-succinct CFC for* $\mathcal{F}_{\mathsf{level}} = \{\mathcal{F}_{\mathsf{level},\kappa}\}$ *(resp. for* $\mathcal{F}_{\mathsf{quad}} = \{\mathcal{F}_{\mathsf{quad},\kappa}\}$*), and let* $\mathcal{F}_n = \{\mathcal{F}_{(d,t,w)}\}$ *be the class of circuits parametrized by depth* d*, in-degree* t*, and width* $w \leq \ell$*. Then, we can construct a* $s'(\ell, (d, t, w))$*-succinct FC scheme* FC *where* $s'(\ell, (d, t, w)) = d \cdot (s_{\max}(\ell, \lceil dw/\ell \rceil) + 1)$.

In particular, for circuits of bounded size $|C| = d \cdot w \leq \ell$*, the proof size is the same as for layered circuits, namely* $s'(\ell, (d, t, w)) = d \cdot (s_{\max}(\ell, 1) + 1)$.

Proof. The construction of the optimized FC scheme consists in reshaping the original input circuit C into an equivalent semi-layered (i.e., $t \ll d$) circuit C' of depth d and width bounded by ℓ. The FC scheme is then identical to the scheme in Figure 4.1. In fact, as FC needs to support circuits of any width $w \leq \ell$, $\mathsf{FC.Setup}(1^\lambda, \ell)$ outputs $\mathsf{ck} \leftarrow \mathsf{CFC.Setup}(1^\lambda, 1^\ell)$.

Let $r = \lfloor \ell/w \rfloor$. For each level h of C with values $\boldsymbol{x}^{(h)}$, we construct level h in circuit C' with values $\boldsymbol{z}^{(h)}$ as described below.

- Let $\boldsymbol{z}^{(0)} := \boldsymbol{x}$. For $h = 1, \ldots, r-1$, set $\boldsymbol{z}^{(h)} := \boldsymbol{x}^{(0)} || \boldsymbol{x}^{(1)} || \cdots || \boldsymbol{x}^{(h)}$ as the concatenation of variables from previous levels. Then, define the wiring in C' by introducing relay gates between levels, such that $\boldsymbol{x}^{(0)}$ is copied to levels $h = 1, \ldots, r-1$, $\boldsymbol{x}^{(1)}$ is copied to levels $h = 2, \ldots, r-1$, etc. Note that, up to level r, C' is the equivalent of C as a layered circuit.
- At level r, set $\boldsymbol{z}^{(r)} := \boldsymbol{x}^{(r)}$. Note that $\boldsymbol{z}^{(r)}$ only depends on inputs at level $r-1$ in C', since all $\boldsymbol{x}^{(0)}, \ldots, \boldsymbol{x}^{(r-1)}$ are duplicated at level $\boldsymbol{z}^{(r-1)}$.
- For levels $h = r+1, \ldots, 2r-1$, expand again as $\boldsymbol{z}^{(h)} := \boldsymbol{x}^{(r)} || \boldsymbol{x}^{(r+1)} || \cdots || \boldsymbol{x}^{(h)}$. Note that values at level h depend only on levels $r-1$ and $h-1$, as $\boldsymbol{z}^{(r-1)}$ contains all values from levels 0 to $r-1$ in C.
- Repeat the steps above, bootstrapping the circuit at levels $2r, 3r, \ldots, d$.

The functions $f^{(1)}, \dots, f^{(d)}$ that describe the levels of C' are such that level h has in-degree $|\mathcal{S}(f^{(h)})| = \lceil h/r \rceil$. Hence, if the CFC is $s(\ell, m, \kappa)$-succinct for $\mathcal{F}_{\mathsf{level}} = \{\mathcal{F}_{\mathsf{level},\kappa}\}$ (resp. for $\mathcal{F}_{\mathsf{quad}} = \{\mathcal{F}_{\mathsf{quad},\kappa}\}$) then the proof size of the FC scheme for C' becomes

$$|\pi| = \sum_{h=0}^{d-1} s(\ell, \lceil h/r \rceil, \kappa) + 1 \leq d \cdot (s_{\max}(\ell, \lceil d/r \rceil) + 1).$$

□

Note that the parameters can be tuned in a per-level basis, allowing for more succinct proofs in practice or when the initial in-degree is low.

Supporting circuits of arbitrary width. Suppose that the parameters of the FC scheme are set up for circuits of bounded width ℓ, and that we want to evaluate a circuit C of width $w > \ell$. The following result shows that this is possible at the cost of increasing the proof size.

Proposition 4.14. *Let* CFC *be a $s(\ell, m, \kappa)$-succinct for $\mathcal{F}_{\mathsf{level}} = \{\mathcal{F}_{\mathsf{level},\kappa}\}$ (resp. for $\mathcal{F}_{\mathsf{quad}} = \{\mathcal{F}_{\mathsf{quad},\kappa}\}$). Let* FC *be our construction in Figure 4.1 for the class of circuits $\mathcal{F}_\ell = \{\mathcal{F}_{(d,t,w)}\}$ of bounded width $w \leq \ell$. Then, we can construct an FC scheme $\tilde{\mathsf{FC}}$ for $\mathcal{F} = \{\mathcal{F}_{(d,t,w)}\}$ for any $w \in \mathbb{N}$ such that $\tilde{\mathsf{FC}}.\mathsf{Setup}(1^\lambda) = \mathsf{FC}.\mathsf{Setup}(1^\lambda, \ell)$ where the proof size is $|\pi| \leq d \cdot \lceil w/\ell \rceil \cdot (s_{\max}(\ell, t \cdot \lceil w/\ell \rceil) + 1)$.*

Proof. We describe $\tilde{\mathsf{FC}}$ in two steps. First, we introduce a circuit transformation from the original C to an equivalent C' of width ℓ and larger depth. Then, we describe the $\tilde{\mathsf{FC}}.\mathsf{Com}$, $\tilde{\mathsf{FC}}.\mathsf{FuncProve}$ and $\tilde{\mathsf{FC}}.\mathsf{FuncVer}$ algorithms. We can construct C' as follows:

- Let $r = \lceil w/\ell \rceil$. For each level $\boldsymbol{x}^{(h)}$, $h = 0, \dots, d$ of C, define sub-levels $\boldsymbol{z}^{(h,s)}$ with indices $(h, 1), \dots, (h, r)$ in C' as the natural split of $\boldsymbol{x}^{(h)}$ in r blocks. This is, for $s \in [r]$, define $\boldsymbol{z}^{(h,s)} = (x^{(h)}_{(s-1)\ell}, x^{(h)}_{(s-1)\ell+1}, \dots, x^{(h)}_{sn-1})$.
- For each level function $f^{(h)} : \mathcal{R}^{mw} \to \mathcal{R}^w$ corresponding to C, let $m' = m \cdot r$ and define r functions $g^{(h,s)} : \mathcal{R}^{m'\ell} \to \mathcal{R}^\ell$ for $s \in [r]$ such that $g^{(h,s)}(\boldsymbol{z}^{(0,1)}, \dots, \boldsymbol{z}^{(h-1,r)}) = \boldsymbol{z}^{(h,s)}$. Note that these functions can be built from a restriction of $f^{(h)}$ to a subset of its outputs.

The commit algorithm $\tilde{\mathsf{FC}}.\mathsf{Com}(\mathsf{ck}, \boldsymbol{x})$ partitions the input $\boldsymbol{x} \in \mathcal{R}^w$ in r blocks $\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(r)}$ of size ℓ as described above, obtains $(\mathsf{com}_{(s)}, \mathsf{aux}_{(s)}) \leftarrow \mathsf{Com}(\mathsf{ck}, \boldsymbol{x}^{(s)})$. It outputs $\tilde{\mathsf{com}} = (\mathsf{com}_{(1)}, \dots, \mathsf{com}_{(r)})$ and $\tilde{\mathsf{aux}} = (\mathsf{aux}_{(1)}, \dots, \mathsf{aux}_{(r)})$.

The functional opening algorithm $\tilde{\mathsf{FC}}.\mathsf{FuncProve}(\mathsf{ck}, \tilde{\mathsf{aux}}, C)$ works as follows:

- Obtain C' from C as presented above, parse $(\boldsymbol{z}^{(0,1)}, \dots, \boldsymbol{z}^{(0,r)}) \leftarrow \mathsf{Parse}(\tilde{\mathsf{aux}})$, and compute $C'(\boldsymbol{z}^{(0,1)}, \dots, \boldsymbol{z}^{(0,r)})$ and all the intermediate values $\boldsymbol{z}^{(h,s)}$ for $h \in [d]$ and $s \in [r]$.
- Commit to each $\boldsymbol{z}^{(h,s)}$ as $(\mathsf{com}_{(h,s)}, \mathsf{aux}_{(h,s)}) \leftarrow \mathsf{CFC}.\mathsf{Com}(\mathsf{ck}, \boldsymbol{z}^{(h,s)})$ for $h \in [d-1]$ and $s \in [r]$.
- Compute the opening proofs for all functions,

$$\forall\, h \in [d], s \in [r] : \ \pi_{(h,s)} \leftarrow \mathsf{CFC}.\mathsf{FuncProve}(\mathsf{ck}, (\mathsf{aux}_{(h',s')})_{h' \in \mathcal{S}(f^{(h)}), s' \in [r]}, g^{(h,s)}).$$

- Return $\tilde{\pi} = (\pi_{(h,s)}, \mathsf{com}_{(h,s)})_{h\in[d],s\in[r]}$.

The verification algorithm $\tilde{\mathsf{FC}}.\mathsf{FuncVer}(\mathsf{ck}, \tilde{\mathsf{com}}, f, \boldsymbol{y}, \tilde{\pi})$ first computes r commitments to the output $\boldsymbol{z}^{(d,s)} \leftarrow \mathsf{Com}(\boldsymbol{y}^{(s)})$ for $s \in [r]$ and then verifies all proofs.

Overall, if the CFC is $s(\ell, m, \kappa)$-succinct for $\mathcal{F}_{\mathsf{level}} = \{\mathcal{F}_{\mathsf{level},\kappa}\}$ (resp. $\mathcal{F}_{\mathsf{quad}} = \{\mathcal{F}_{\mathsf{quad},\kappa}\}$), and the original circuit $C \in \mathcal{F}_{(d,t,w)}$ (i.e., the in-degree of C is bounded by t), then the proof size of the FC scheme for C' becomes

$$|\pi| = (d-1)r + r \cdot \sum_{h=0}^{d-1} s(\ell, hr, \kappa) \leq dr \cdot (s_{\max}(\ell, tr) + 1).$$

□

4.6 Paring-based CFC for Quadratic Functions

We present our construction of a chainable functional commitment for quadratic functions based on pairings. With our CFC, one can commit to a set of vectors $\boldsymbol{x}_1, \ldots \boldsymbol{x}_m$ of length ℓ and then open the commitment to a quadratic function $f : \mathbb{F}^{mn} \rightarrow \mathbb{F}^{\ell}$, for any $m = \mathsf{poly}(\lambda)$. The functional opening proofs of our scheme are quadratic in the number m of input vectors, but constant in the (possibly padded) length ℓ of each input vector and of the output. Security is proven in the standard model based on a new falsifiable assumption that we introduce in Section 4.6.1 and justify in the generic bilinear group model. In Section 4.6.5 we discuss the FCs for circuits that we obtain by applying the generic transform of Section 4.5 to this pairing-based CFC.

We present our CFC with deterministic commitments and functional openings. We detail how to make our commitments perfectly com-hiding in Section 4.6.8. We note that the FCs for circuits obtained from the com-hiding CFC are also com-hiding, and their functional openings can be made zero-knowledge by applying Theorem 4.7, which we can efficiently instantiate using, e.g., the Groth-Sahai [GS08] NIZK.

4.6.1 The HiKer assumption

We prove that our construction satisfies evaluation binding under a new falsifiable assumption, called HintedKernel (HiKer), that we justify in the generic group model in Theorem 4.16.

The name of the assumption comes from its similarity with the KerDH assumption of [MRV16] (introduced in Section 6.3)which for matrices $[\mathbf{A}]_2$ from certain (random) distributions asks the adversary to find a nonzero vector $[\boldsymbol{z}]_1$ such that $\mathbf{A}\boldsymbol{z} = \mathbf{0}$. In our case the adversary is challenged to find a nonzero $[u, v]_1$ such that $u\eta + v = 0$, when given $[1, \eta]_2$ but also other group elements, the "hints", that depend on η and other random variables.

Definition 4.15 (ℓ-HiKer Assumption)**.** *Let* $\mathsf{bgp} = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2)$ *be a bilinear group setting, let* $\ell \in \mathbb{N}$ *and let* $\mathcal{G}_1, \mathcal{G}_2$ *be the following sets of Laurent monomials in* $\mathbb{Z}_q[S_1, T_1, \ldots, S_n, T_n, H]$*:*

$$\mathcal{G}_1(\boldsymbol{S}, \boldsymbol{T}, H) := \{S_i,\, T_i\}_{i\in[\ell]} \cup \{S_i \cdot T_j\}_{i,j\in[\ell]} \cup \left\{\frac{S_{i'}}{S_i} \cdot T_i \cdot H\right\}_{\substack{i,i'\in[\ell]\\ i\neq i'}} \cup \left\{\frac{S_{i'} \cdot T_{j'}}{S_i \cdot T_j} \cdot H\right\}_{\substack{i,j,i',j'\in[\ell]\\ (i,j)\neq(i',j')}}$$
$$\mathcal{G}_2(\boldsymbol{S}, \boldsymbol{T}, H) := \{H\} \cup \{S_i\}_{i\in[\ell]} \cup \left\{\frac{1}{S_i} \cdot T_i \cdot H,\ \frac{1}{S_i} \cdot H\right\}_{i\in[\ell]} \cup \left\{\frac{1}{S_i} \cdot \frac{1}{T_j} \cdot H\right\}_{i,j\in[\ell]}$$

The ℓ-HintedKernel (ℓ-HiKer) assumption holds if for every $\ell = \mathsf{poly}(\lambda)$ *and any PPT* $\mathcal{A}$*, the following advantage is negligible*

$$\mathbf{Adv}^{\ell\text{-}HiKer}_{\mathcal{A}}(\lambda) = \Pr\left[\begin{array}{c} ([U]_1, [V]_1) \neq ([1]_1, [1]_1) \\ \wedge\, [U]_1 \cdot [\eta]_2 = [V]_1 \cdot [1]_2 \end{array} \middle|\, ([U]_1, [V]_1) \leftarrow \mathcal{A}\left(\mathsf{bgp}, \begin{array}{c} [\mathcal{G}_1(\boldsymbol{\sigma}, \boldsymbol{\tau}, \eta)]_1, \\ [\mathcal{G}_2(\boldsymbol{\sigma}, \boldsymbol{\tau}, \eta)]_2 \end{array}\right)\right]$$

where the probability is over the random choices of $\boldsymbol{\sigma}, \boldsymbol{\tau}, \eta$ *and* $\mathcal{A}$*'s random coins.*

Lemma 4.16. *The ℓ-HiKer assumption holds in the generic bilinear group model [BBG05].*

Proof. First, note that the assumption is equivalent to an assumption without rational terms. Indeed, for a uniformly sampled η', consider the assumption above where $\eta = \eta' \prod_{i,j\in[\ell]} \sigma_i \tau_j$.

The intuition is that since the solution (U, V) satisfies the equation $[U]_1 \cdot [\eta]_2 = [V]_2 \cdot [1]_2$ then it must be of the form $(U, V) = [u, \eta u]_1$ for some u. However, if we look at the input of the adversary in $\mathbb{G}_1$, there is no pair of elements in the linear span of $[1, \eta]_1$. Note also that elements in $\mathbb{G}_2$ cannot be used by a GGM extractor as bgp is a Type-III bilinear group setting. A detailed proof follows.

Formally, let $\mathcal{A}$ be an adversary that on input (bgp, Ω) outputs two elements $[U]_1, [V]_1 \in \mathbb{G}_1$ such that $[U]_1 \cdot [\eta]_2 = [V]_1 \cdot [1]_2$. Then, the GGM extractor must output two polynomials $p_u(\boldsymbol{S}, \boldsymbol{T}, H), p_v(\boldsymbol{S}, \boldsymbol{T}, H)$ with coefficients $u_0, u_{\sigma,i}, u_{\tau,i}, u_{i,j}, u_{i,i'}, u_{i,j,i',j'}$ and $v_0, v_{\sigma,i}, v_{\tau,i}, v_{i,j}, v_{i,i'}, v_{i,j,i',j'}$ such that:

$$\begin{aligned} 0 = {} & p_u(\boldsymbol{S}, \boldsymbol{T}, H)H + p_v(\boldsymbol{S}, \boldsymbol{T}, H) = \\ & u_0 H + v_0 + \sum_i [(u_{\sigma,i} S_i + u_{\tau,i} T_i)H + v_{\sigma,i} S_i + v_{\tau,i} T_i] + \sum_{i,j} \left[u_{i,j} S_i T_j H + v_{i,j} S_i T_j\right] \\ & + \sum_{\substack{i,i'\in[\ell]\\ i\neq i'}} \left[u_{i,i'} \frac{S_{i'}}{S_i} T_i H^2 + v_{i,i'} \frac{S_{i'}}{S_i} T_i H\right] + \sum_{\substack{i,j,i',j'\in[\ell]\\ (i,j)\neq(i',j')}} \left[u_{i,j,i',j'} \frac{S_{i'} T_{j'}}{S_i T_j} H^2 + v_{i,j,i',j'} \frac{S_{i'} T_{j'}}{S_i T_j} H\right]. \end{aligned}$$

Due to the equivalence mentioned above, we can effectively do a change of variable $H \mapsto HA$ where $A = \prod_{i,j\in[\ell]} S_i T_j$ and reorganize the expression as a polynomial $c_0 + c_1 H + c_2 H^2$ in H, where

$$c_0 = \left[v_0 + \sum_{i\in[\ell]} (v_{\sigma,i} S_i + v_{\tau,i} T_i) + \sum_{i,j\in[\ell]} v_{i,j} S_i S_j \right],$$

$$c_1 = \left[u_0 + \sum_{i\in[\ell]} (u_{\sigma,i} S_i + u_{\tau,i} T_i) + \sum_{i,j\in[\ell]} u_{i,j} S_i T_j + \sum_{\substack{i,i'\in[\ell]\\ i\neq i'}} v_{i,i'} \frac{S_{i'}}{S_i} T_i + \sum_{\substack{i,j,i',j'\in[\ell]\\ (i,j)\neq(i',j')}} v_{i,j,i',j'} \frac{S_{i'}T_{j'}}{S_i T_j} \right] A,$$

$$c_2 = \left[\sum_{\substack{i,j,i',j'\in[\ell]\\ (i,j)\neq(i',j')}} u_{i,j,i',j'} \frac{S_{i'}T_{j'}}{S_i T_j} + \sum_{\substack{i,i'\in[\ell]\\ i\neq i'}} u_{i,i'} \frac{S_{i'}}{S_i} T_i \right] A^2.$$

For the above to equal the zero polynomial in H, all terms must cancel. We analyze the constant, linear, and quadratic terms separately. Note that as all fractions are multiplied by $A = \prod_{i,j\in[\ell]} S_i T_j$, all denominators vanish.

- The constant term does not include cross-terms, so all monomials are linearly independent and the expression cancels only if $v_0 = v_{\sigma,i} = v_{\tau,i} = v_{i,j} = 0$.
- The linear term is formed by pairwise distinct monomials which are all independent; no allowed choice of indices i, j, i', j' or i, i' produces a monomial in the linear span of any others. In particular, note that variables in $(S_{i'}/S_i)T_i$ only cancel for $i = i'$ which is not in the sum. Also, the denominator of $(S_{i'}T_{j'})/(S_i T_j)$ only cancels if $(i, j) = (i', j')$ which is also not in the sum.
- For the quadratic term, we reason analogously to conclude that all terms are independent.

It follows that all coefficients of $p_u(\boldsymbol{S}, \boldsymbol{T}, H)$ and $p_v(\boldsymbol{S}, \boldsymbol{T}, H)$ must be zero, so $p_u = p_v = 0$ and the assumption holds.

□

4.6.2 Our CFC Construction

As defined in the previous section we express $f \in \mathcal{F}_{\mathsf{quad}}$ through a set of matrices $\mathbf{F}^{(h)} \in \mathbb{F}^{\ell\times\ell}$ and $\mathbf{G}^{(h,h')} \in \mathbb{F}^{\ell\times\ell^2}$, and a vector $\boldsymbol{e} \in \mathbb{F}^{\ell}$ such that

$$f(\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}) = \boldsymbol{e} + \sum_{h\in\mathcal{S}_1(f)} \mathbf{F}^{(h)} \cdot \boldsymbol{x}^{(h)} + \sum_{(h,h')\in\mathcal{S}_2^{\otimes}(f)} \mathbf{G}^{(h,h')} \cdot (\boldsymbol{x}^{(h)} \otimes \boldsymbol{x}^{(h')}) \tag{4.5}$$

For the sake of defining the succinctness of our CFC we parametrize the class $\mathcal{F}_{\mathsf{quad}}$ by the size of the quadratic support of f. Formally, let $\mathcal{K} = \{0, 1, \dots, m(m+1)/2\}$. Then we partition $\mathcal{F}_{\mathsf{quad}}$ as $\{\mathcal{F}_{\mathsf{quad},\kappa}\}_{\kappa\in\mathcal{K}}$ where each $\mathcal{F}_{\mathsf{quad},\kappa} = \{f \in \mathcal{F}_{\mathsf{quad}} : \mathcal{S}_2^{\otimes}(f) = \kappa\}$. Note that the parametrization extends naturally to the class $\mathcal{F}_{\mathsf{level}}$ as described in Section 4.5. Due to the definition of $\mathcal{F}_{\mathsf{level}}$, in that case we have at most m partitions, i.e,. $\mathcal{F}_{\mathsf{level}} = \{\mathcal{F}_{\mathsf{level},\kappa}\}_{\kappa=0}^{m}$.

Setup$(1^\lambda, 1^\ell)$ Let $\ell \geq 1$ be an integer representing the width of each of the inputs of the functions to be computed at opening time. Generate a bilinear group description $\mathsf{bgp} := (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2) \leftarrow \mathcal{BG}(1^\lambda)$, and let $\mathbb{F} := \mathbb{Z}_q$.

Next, sample random $\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma} \leftarrow\$\ \mathbb{F}^\ell$, $\eta_\alpha, \eta_\beta, \eta_\gamma \leftarrow\$\ \mathbb{F}$, and output

$$\mathsf{ck} := \begin{pmatrix} [\boldsymbol{\alpha}]_1, [\boldsymbol{\alpha}]_2, [\boldsymbol{\beta}]_1, [\boldsymbol{\gamma}]_1, [\boldsymbol{\alpha} \otimes \boldsymbol{\beta}]_1, [\eta_\alpha]_2, [\eta_\beta]_2, [\eta_\gamma]_2 \\ \left\{\left[\alpha_i \frac{\gamma_{i'}}{\gamma_i}\eta_\alpha\right]_1, \left[\frac{\alpha_{i'}}{\alpha_i}\beta_i\eta_\beta\right]_1\right\}_{\substack{i,i' \in [\ell] \\ i \neq i'}} \left\{\left[\frac{\alpha_{i'}\beta_{j'}}{\alpha_i\beta_j}\gamma_k\eta_\gamma\right]_1\right\}_{\substack{i,j,i',j',k \in [\ell] \\ (i,j) \neq (i',j')}} \\ \left\{\left[\frac{\alpha_i\eta_\alpha}{\gamma_i}\right]_2, \left[\frac{\beta_i\eta_\beta}{\alpha_i}\right]_2\right\}_{i \in [\ell]}, \left\{\left[\frac{\gamma_k\eta_\gamma}{\alpha_i}\right]_2\right\}_{i,k \in [\ell]} \left\{, \left[\frac{\gamma_k\eta_\gamma}{\alpha_i\beta_j}\right]_2\right\}_{i,j,k \in [\ell]} \end{pmatrix}.$$

Com$(\mathsf{ck}, \boldsymbol{x})$ output $\mathsf{com} := [\langle \boldsymbol{x}, \boldsymbol{\alpha} \rangle]_1$ and $\mathsf{aux} = \boldsymbol{x}$.

FuncProve$(\mathsf{ck}, (\mathsf{aux}_i)_{i \in [m]}, f) \to \pi$ Let $\mathbf{F}^{(h)} \in \mathbb{F}^{\ell \times \ell}$ for $h \in \mathcal{S}_1(f)$, $\mathbf{G}^{(h,h')} \in \mathbb{F}^{\ell \times \ell^2}$ for $(h, h') \in \mathcal{S}_2^\otimes(f)$, and $\boldsymbol{e} \in \mathbb{F}^\ell$ be the matrices and vectors associated to $f : \mathbb{F}^{mn} \to \mathbb{F}^\ell$. The functional opening algorithm computes the output $\boldsymbol{y} = f(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)})$ and proceeds as follows.

- For every $h \in \mathcal{S}_2(f)$: compute $\left[X_h^{(2)}\right]_2 := [\langle \boldsymbol{x}^{(h)}, \boldsymbol{\alpha} \rangle]_2$, $\left[X_h^{(\beta)}\right]_1 := [\langle \boldsymbol{x}^{(h)}, \boldsymbol{\beta} \rangle]_1$, which are commitments to $\boldsymbol{x}^{(h)}$ under $\boldsymbol{\alpha}$ in $\mathbb{G}_2$ and under $\boldsymbol{\beta}$ in $\mathbb{G}_1$, resp.
- For every $h \in \mathcal{S}_2(f)$: compute a linear map opening proof for the identity function, to show that the input commitment $[X_h]_1$ and $\left[X_h^{(\beta)}\right]_1$ open to the same value:

$$\left[\pi_h^{(\beta)}\right]_1 := \sum_{\substack{i,i' \in [\ell] \\ i \neq i'}} x_{i'}^{(h)} \cdot \left[\frac{\alpha_{i'}}{\alpha_i}\beta_i\eta_\beta\right]_1$$

- For every pair of inputs $\boldsymbol{x}^{(h)}, \boldsymbol{x}^{(h')}$ such that $(h, h') \in \mathcal{S}_2^\otimes(f)$, compute a commitment to their tensor products as follows:

$$\left[Z_{h,h'}\right]_1 := \sum_{i,j \in [\ell]} x_i^{(h)} x_j^{(h')} \cdot [\alpha_i\beta_j]_1 = [\langle \boldsymbol{x}^{(h)} \otimes \boldsymbol{x}^{(h')}, \boldsymbol{\alpha} \otimes \boldsymbol{\beta} \rangle]_1.$$

- Compute a linear map opening proof to show that the vector $\boldsymbol{y}$ satisfies equation (4.5), with respect to all the inputs $\boldsymbol{x}^{(h)}$ committed in $[X_h]_1$ and the inputs $\boldsymbol{x}^{(h)} \otimes \boldsymbol{x}^{(h')}$ committed in $\left[Z_{h,h'}\right]_1$:

$$\begin{aligned} \left[\pi^{(\gamma)}\right]_1 := & \sum_{h \in \mathcal{S}_1(f)} \sum_{\substack{i,i',k \in [\ell] \\ i \neq i'}} F_{k,i}^{(h)} \cdot x_{i'}^{(h)} \cdot \left[\frac{\alpha_{i'}}{\alpha_i}\gamma_k\eta_\gamma\right]_1 \\ & + \sum_{(h,h') \in \mathcal{S}_2^\otimes(f)} \sum_{\substack{i,j,i',j',k \in [\ell] \\ (i,j) \neq (i',j')}} G_{k,(i,j)}^{(h,h')} \cdot x_{i'}^{(h)} x_{j'}^{(h')} \cdot \left[\frac{\alpha_{i'}\beta_{j'}}{\alpha_i\beta_j}\gamma_k\eta_\gamma\right]_1 \end{aligned}$$

  Note that $\left[\pi^{(\gamma)}\right]_1$ is in fact a proof for the vector $\boldsymbol{e} - \boldsymbol{t}$; the linear shift will be addressed by the verifier in equation (4.11).

- Commit to the output $\boldsymbol{y}$ under $\boldsymbol{\gamma}$ by computing $\left[Y^{(\gamma)}\right]_1 := [\langle \boldsymbol{y}, \boldsymbol{\gamma}\rangle]_1$. Then, compute a linear map opening proof for the identity function, to show that $\left[Y^{(\gamma)}\right]_1$ and the commitment to the output $\mathsf{com}_y \leftarrow \mathsf{Com}(\mathsf{ck}, \boldsymbol{y})$ (which is under $\boldsymbol{\alpha}$) open to the same value:
$$\left[\pi^{(\alpha)}\right]_1 := \sum_{\substack{i,i' \in [\ell] \\ i \neq i'}} y_{i'} \cdot \left[\alpha_i \frac{\gamma_{i'}}{\gamma_i} \eta_\alpha\right]_1$$
- Return
$$\pi := \left(\left\{\left[X_h^{(2)}\right]_2, \left[X_h^{(\beta)}\right]_1, \left[\pi_h^{(\beta)}\right]_1\right\}_{h \in \mathcal{S}_2(f)}, \left\{[Z_{h,h'}]_1\right\}_{(h,h') \in \mathcal{S}_2^{\otimes}(f)}, \left[Y^{(\gamma)}\right]_1, \left[\pi^{(\alpha)}\right]_1, \left[\pi^{(\gamma)}\right]_1\right).$$

$\underline{\mathsf{FuncVer}(\mathsf{ck}, (\mathsf{com}_i)_{i \in [m]}, \mathsf{com}_y, f, \pi) \to b \in \{0,1\}}$ Parse the proof π as above and set $[X_h]_1 := \mathsf{com}_h$. Output 1 if all the following checks pass and 0 otherwise:

- Verify the consistency of all the commitments. Namely, verify that each $[X_h]_1$ and $\left[X_h^{(2)}\right]_2$ are commitments to the same value in $\mathbb{G}_1$ and $\mathbb{G}_2$:
$$\forall h \in \mathcal{S}_2(f): \quad [X_h]_1 \cdot [1]_2 \stackrel{?}{=} [1]_1 \cdot \left[X_h^{(2)}\right]_2 \tag{4.6}$$
- Verify the linear map commitment proofs $\left[\pi_h^{(\beta)}\right]_1$ that both $\left[X_h^{(\beta)}\right]_1, [X_h]_1$ commit to the same value in different sets of parameters:
$$\forall h \in \mathcal{S}_2(f): \quad [X_h]_1 \cdot \sum_{i \in [\ell]} \left[\frac{\beta_i \eta_\beta}{\alpha_i}\right]_2 \stackrel{?}{=} \left[\pi_h^{(\beta)}\right]_1 \cdot [1]_2 + \left[X_h^{(\beta)}\right]_1 \cdot [\eta_\beta]_2 \tag{4.7}$$
- Verify the consistency of the commitments to the tensor products, i.e., verify that $[Z_{h,h'}]_1$ is a commitment to $\boldsymbol{x}^{(h)} \otimes \boldsymbol{x}^{(h')}$:
$$\forall (h,h') \in \mathcal{S}_2^{\otimes}(f): \quad [Z_{h,h'}]_1, [1]_2 \stackrel{?}{=} \left[X_{h'}^{(\beta)}\right]_1 \cdot \left[X_h^{(2)}\right]_2 \tag{4.8}$$
- Verify the linear map commitment proof $\left[\pi^{(\alpha)}\right]_1$ that both com_y, $\left[Y^{(\gamma)}\right]_1$ commit to the same value in different sets of parameters:
$$\left[Y^{(\gamma)}\right]_1 \cdot \left(\sum_{i \in [\ell]} \left[\frac{\alpha_i \eta_\alpha}{\gamma_i}\right]_2\right) \stackrel{?}{=} \left[\pi^{(\alpha)}\right]_1 \cdot [1]_2 + \mathsf{com}_y \cdot [\eta_\alpha]_2 \tag{4.9}$$
- Verify the linear map commitment proof to check that, intuitively, $\left[Y^{(\gamma)}\right]_1$ is a commitment under $\boldsymbol{\gamma}$ to the output of f, computed from the inputs committed in $[X_h]_1$ and $[Z_{h,h'}]_1$. To this end, compute the encoding of the matrices $\mathbf{F}^{(h)}$ for $h \in \mathcal{S}_1(f)$, $\mathbf{G}^{(h,h')}$ for $(h,h') \in \mathcal{S}_2^{\otimes}(f)$ and the vector $\boldsymbol{e}$ as follows. Let $[\Theta]_1 = [\langle \boldsymbol{e}, \boldsymbol{\gamma}\rangle]_1$ and
$$[\Phi_h]_2 := \sum_{i,k \in [\ell]} F_{k,i}^{(h)} \cdot \left[\frac{\gamma_k \eta_\gamma}{\alpha_i}\right]_2, \quad [\Gamma_{h,h'}]_2 := \sum_{i,j,k \in [\ell]} G_{k,(i,j)}^{(h,h')} \cdot \left[\frac{\gamma_k \eta_\gamma}{\alpha_i \beta_j}\right]_2 \tag{4.10}$$
and then verify that

$$\sum_{h\in\mathcal{S}_1(f)} [X_h]_1\cdot[\Phi_h]_2 + \sum_{(h,h')\in\mathcal{S}_2^{\otimes}(f)} \left[Z_{h,h'}\right]_1\cdot\left[\Gamma_{h,h'}\right]_2 \overset{?}{=} \left[\pi^{(\gamma)}\right]_1\cdot[1]_2 + \left(\left[Y^{(\gamma)}\right]_1 - [\Theta]_1\right)\cdot[\eta_\gamma]_2. \tag{4.11}$$

Theorem 4.17. *Assume that the ℓ-HiKer assumption holds for a bilinear group setting generated by $\mathcal{BG}$. Then the construction* CFC *described above is an evaluation binding CFC scheme for the class $\mathcal{F}_{\mathsf{quad}}$ of quadratic functions over any $m = \mathsf{poly}(\lambda)$ vectors of length $\leq \ell$, that has efficient verification and is additively homomorphic. Considering the partitioning of $\mathcal{F}_{\mathsf{quad}} = \{\mathcal{F}_{\mathsf{quad},\kappa}\}_{\kappa=0}^{m(m+1)/2}$,* CFC *is $s(\ell,m,\kappa)$-succinct for $s(\ell,m,\kappa) = (\kappa+3m+3)$. Furthermore, when executed on the class of functions $\mathcal{F}_{\mathsf{level}} \subset \mathcal{F}_{\mathsf{quad}}$ introduced in Section 4.5.4 and partitioned as $\mathcal{F}_{\mathsf{level}} = \{\mathcal{F}_{\mathsf{level},\kappa}\}_{\kappa=0}^{m}$, then* CFC *is $(4\kappa+3)$-succinct.*

4.6.3 Correctness

To prove correctness, consider honestly generated input commitments $[X_h]_1 = \left[\langle \boldsymbol{x}^{(h)}, \boldsymbol{\alpha}\rangle\right]_1$ for $h\in[m]$ and an honestly generated opening

$$\pi := \left(\left\{\left[X_h^{(2)}\right]_2, \left[X_h^{(\beta)}\right]_1, \left[\pi_h^{(\beta)}\right]_1\right\}_{h\in\mathcal{S}_2(f)}, \{\left[Z_{h,h'}\right]_1\}_{(h,h')\in\mathcal{S}_2^{\otimes}(\boldsymbol{f})}, \left[Y^{(\gamma)}\right]_1, \left[\pi^{(\alpha)}\right]_1, \left[\pi^{(\gamma)}\right]_1\right)$$

for a quadratic function f represented by the matrices $\boldsymbol{e}, \mathbf{F}^{(h)}, \mathbf{G}^{(h,h')}$ for $h\in\mathcal{S}_1(f)$ and $(h,h')\in\mathcal{S}_2^{\otimes}(f)$.

The correctness of equations (4.6) and (4.8) follows easily by construction since

$$\begin{aligned}
[X_h]_1\cdot[1]_2 &= \left[\langle \boldsymbol{x}^{(h)},\boldsymbol{\alpha}\rangle\right]_1\cdot[1]_2 = [1]_1\cdot\left[\langle \boldsymbol{x}^{(h)},\boldsymbol{\alpha}\rangle\right]_2 = [1]_1\cdot\left[X_h^{(2)}\right]_2,\\
\left[Z_{h,h'}\right]_1\cdot[1]_2 &= \left[\langle \boldsymbol{x}^{(h)}\otimes\boldsymbol{x}^{(h')},\boldsymbol{\alpha}\otimes\boldsymbol{\beta}\rangle\right]_1\cdot[1]_2 = \left[\sum_{i,j\in[\ell]} x_i^{(h)}x_j^{(h')}\alpha_i\beta_j\right]_1\cdot[1]_2\\
&= \left[\langle \boldsymbol{x}^{(h')},\boldsymbol{\beta}\rangle\right]_1\cdot\left[\langle \boldsymbol{x}^{(h)},\boldsymbol{\alpha}\rangle\right]_2 = \left[X_{h'}^{(\beta)}\right]_1\cdot\left[X_h^{(2)}\right]_2.
\end{aligned}$$

The correctness of equation (4.7) can be seen as follows. Given $h\in\mathcal{S}_2(f)$, we have that

$$\begin{aligned}
[X_h]_1\cdot\sum_{i\in[\ell]}\left[\frac{\beta_i\eta_\beta}{\alpha_i}\right]_2 &= \left[\left(\sum_{i\in[\ell]} x_i^{(h)}\alpha_i\right)\cdot\left(\sum_{i\in[\ell]}\frac{\beta_i\eta_\beta}{\alpha_i}\right)\right]_T = \left[\sum_{i,i'\in[\ell]} x_{i'}^{(h)}\frac{\alpha_{i'}}{\alpha_i}\beta_i\eta_\beta\right]_T\\
&= \left[\sum_{\substack{i,i'\in[\ell]\\ i\neq i'}} x_{i'}^{(h)}\frac{\alpha_{i'}}{\alpha_i}\beta_i\eta_\beta + \sum_{i\in[\ell]} x_i^{(h)}\beta_i\eta_\beta\right]_T = \left[\pi_h^{(\beta)}\right]_1\cdot[1]_2 + \left[X_h^{(\beta)}\right]_1\cdot[\eta_\beta]_2.
\end{aligned}$$

Similarly, for equation (4.9) we have that

$$\left[Y^{(\gamma)}\right]_1 \cdot \sum_{i\in[\ell]} \left[\frac{\alpha_i \eta_\alpha}{\gamma_i}\right]_2 = \left[\left(\sum_{i\in[\ell]} y_i\gamma_i\right)\cdot\left(\sum_{i\in[\ell]}\frac{\alpha_i\eta_\alpha}{\gamma_i}\right)\right]_T = \left[\sum_{i,i'\in[\ell]} y_{i'}\frac{\gamma_{i'}}{\gamma_i}\alpha_i\eta_\alpha\right]_T$$
$$= \left[\sum_{\substack{i,i'\in[\ell]\\ i\neq i'}} y_{i'}\alpha_i\frac{\gamma_{i'}}{\gamma_i}\eta_\alpha + \sum_{i\in[\ell]} y_i\alpha_i\eta_\alpha\right]_T = \left[\pi^{(\alpha)}\right]_1 \cdot [1]_2 + \mathsf{com}_y \cdot [\eta_\alpha]_2.$$

Finally, the correctness of equation (4.11) can be proven in an analogous way. First of all, we expand the pairing coefficients on the LHS in $\mathbb{G}_T$,

$$[X_h]_1 \cdot [\Phi_h]_2 = \left[\sum_{k\in[\ell]}\left(\sum_{i\in[\ell]} F^{(h)}_{k,i}\cdot x^{(h)}_i\right)\gamma_k\eta_\gamma + \sum_{\substack{i,i',k=1\\ i\neq i'}}^{\ell} F^{(h)}_{k,i}\cdot x^{(h)}_{i'}\cdot\frac{\alpha_{i'}}{\alpha_i}\gamma_k\eta_\gamma\right]_T$$

$$\left[Z_{h,h'}\right]_1 \cdot \left[\Gamma_{h,h'}\right]_2$$
$$= \left[\sum_{k\in[\ell]}\left(\sum_{i,j\in[\ell]} G^{(h,h')}_{k,(i,j)}\cdot x^{(h)}_i x^{(h')}_j\right)\gamma_k\eta_\gamma + \sum_{\substack{i,j,i',j',k\in[\ell]\\ (i,j)\neq(i',j')}} G^{(h,h')}_{k,(i,j)}\cdot x^{(h)}_{i'}x^{(h')}_{j'}\cdot\frac{\alpha_{i'}\beta_{j'}}{\alpha_i\beta_j}\gamma_k\eta_\gamma\right]_T.$$

By using the identities above and equation (4.5), we have

$$\sum_{h\in\mathcal{S}_1(f)} [X_h]_1\cdot[\Phi_h]_2 \cdot \sum_{(h,h')\in\mathcal{S}_2^{\otimes}(f)} \left[Z_{h,h'}\right]_1\cdot\left[\Gamma_{h,h'}\right]_2$$
$$= \left[\sum_{\substack{h\in\mathcal{S}_1(f)\\ i,k\in[\ell]}} F^{(h)}_{k,i}\cdot x^{(h)}_i\cdot\gamma_k\eta_\gamma + \sum_{\substack{(h,h')\in\mathcal{S}_2^{\otimes}(f)\\ i,j,k\in[\ell]}} G^{(h,h')}_{k,(i,j)}\cdot x^{(h)}_i x^{(h')}_j\cdot\gamma_k\eta_\gamma\right]_T + \left[\pi^{(\gamma)}\right]_1\cdot[1]_2$$
$$= \left[\sum_{k\in[\ell]}(y_k - e_k)\gamma_k\eta_\gamma\right]_T + \left[\pi^{(\gamma)}\right]_1\cdot[1]_2 = \left(\mathsf{com}_y - [\Theta]_1\right)\cdot\left[\eta_\gamma\right]_2 + \left[\pi^{(\gamma)}\right]_1\cdot[1]_2\,.$$

Note that from the equations above it also follows that CFC is additively homomorphic.

4.6.4 Succinctness

An opening proof π to a given function $f \in \mathcal{F}_{\mathsf{quad},\kappa}$ includes $\left|\mathcal{S}_2^{\otimes}(f)\right| = \kappa$ commitments to tensored inputs $\tilde{X}_{h,h'}$, and the triples of elements $\left\{\left[X^{(2)}_h\right]_2, \left[X^{(\beta)}_h\right]_1, \left[\pi^{(\beta)}_h\right]_1\right\}_{h\in\mathcal{S}_2(f)}$, which are $3\cdot|\mathcal{S}_2(f)|$ group elements. Finally, π includes three additional group elements $\left[Y^{(\gamma)}\right]_1, \left[\pi^{(\alpha)}\right]_1, \left[\pi^{(\gamma)}\right]_1$. Hence, the proof consists of $\kappa + 3\cdot|\mathcal{S}_2(f)| + 3$ group elements, and essentially ranges from $O(1)$ (in fact π has only 3 elements if f is a linear function) to $O\left(m^2\right)$ depending on the quadratic support of f. Precisely, considering a fixed polynomial $p(\lambda)$ that upper bounds the size of a group element from $\mathbb{G}_1$ or $\mathbb{G}_2$, our CFC is $O(\kappa)$-succinct.

When the CFC is executed on functions from the class $\mathcal{F}_{\mathsf{level}} = \{\mathcal{F}_{\mathsf{level},\kappa}\}$ introduced in Section 4.5.4 we have that $|\mathcal{S}_2(f)| = \kappa \leq m$. In this case a CFC functional opening contains $4\kappa + 3$ group elements and our CFC is also $O(m)$-succinct.

4.6.5 Resulting Instantiations of FC for Circuits

We summarize the FC schemes that result from instantiating our generic construction of Section 4.5 with our pairing-based CFC.

Corollary 4.18. *Assume that the ℓ-HiKer assumption holds for $\mathcal{BG}$. Then the following statements hold:*

1. *There exists an FC scheme for the class $\mathcal{F}_n = \{\mathcal{F}_{(d,t,w)}\}$ of arithmetic circuits of width $w \leq \ell$ that is $O(d \cdot t)$-succinct. In particular, the FC is $O(d^2)$-succinct for an* arbitrary *arithmetic circuit of multiplicative depth d, and is $O(d)$-succinct for a* layered *arithmetic circuit of multiplicative depth d.*
2. *There exists an FC scheme for the class $\mathcal{F}_n = \{\mathcal{F}_{(d,t,w)}\}$ of arithmetic circuits of width $w \leq \ell$ that is $O\big(d^2 \cdot w \cdot \ell^{-1}\big)$-succinct.*
3. *There exists an FC scheme for the class of arithmetic circuits of size $\leq S$, that is $O(d)$-succinct where d is the multiplicative depth of the circuit.*
4. *For any $w_0 \geq 2$, there exists an FC scheme for the class $\mathcal{F} = \{\mathcal{F}_{(d,t,w)}\}$ of circuits of arbitrary width $w > w_0$ that is $O\big(d \cdot t \cdot (w/w_0)^2\big)$-succinct.*

Proof. Consider the FC construction in Section 4.5 instantiated with our pairing-based CFC for quadratic functions. More precisely, we consider arithmetic circuits following the model described in Section 4.5.4 which allows us to use CFC only with quadratic functions in $\mathcal{F}_{\mathsf{level}}$. The statements of the corollary can be obtained by combining the following observations.

1. For arbitrary circuits, note that the in-degree t of the circuit upper bounds the number m of inputs used in the CFC, and thus an FC proof consists of d CFC proofs, which makes a total of $4dt + 3d$ group elements. $O(d^2)$-succinctness for arbitrary arithmetic circuits follows from the fact that an arbitrary arithmetic circuit of depth d may have in-degree up to d, while $O(d)$-succinctness for layered circuits follows from the in-degree being 1 in such circuits.
2. The statement follows from the transformation that we present in Proposition 4.13.
3. To see this statement, let us consider the folklore transformation from arbitrary to layered arithmetic circuits (which is a special case of the transformation in Proposition 4.13). If one starts from a circuit C of width ℓ and depth d, the circuit C' resulting from this transformation has the same depth, but width $\leq \ell \cdot d$, which is upper bounded by the circuit size S.
4. The statement follows directly from Proposition 4.14, where w_0 is the maximum width supported by the parameters of the given FC.

□

4.6.6 Proof of Security

In this section, we prove that our CFC satisfies evaluation binding.

Consider an adversary $\mathcal{A}$ who returns a tuple $((\mathsf{com}_h)_{h\in[m]}, \mathsf{com}_y, f, \pi, \tilde{\mathsf{com}}_y, \tilde{\pi})$ that breaks evaluation binding, set $[X_h]_1 := \mathsf{com}_h$, and parse the proofs as follows

$$\pi := \left(\left\{\left[X_h^{(2)}\right]_2, \left[X_h^{(\beta)}\right]_1, \left[\pi_h^{(\beta)}\right]_1\right\}_{h\in\mathcal{S}_2(f)}, \{[Z_{h,h'}]_1\}_{(h,h')\in\mathcal{S}_2^{\otimes}(f)}, \left[Y^{(\gamma)}\right]_1, \left[\pi^{(\alpha)}\right]_1, \left[\pi^{(\gamma)}\right]_1\right)$$
$$\tilde{\pi} := \left(\left\{\left[\tilde{X}_h^{(2)}\right]_2, \left[\tilde{X}_h^{(\beta)}\right]_1, \left[\tilde{\pi}_h^{(\beta)}\right]_1\right\}_{h\in\mathcal{S}_2(f)}, \{\tilde{Z}_{h,h'}\}_{(h,h')\in\mathcal{S}_2^{\otimes}(f)}, \left[\tilde{Y}^{(\gamma)}\right]_1, \left[\tilde{\pi}^{(\alpha)}\right]_1, \left[\tilde{\pi}^{(\gamma)}\right]_1\right)$$

Recall that by definition of evaluation binding, if $\mathcal{A}$'s attack is successful, both proofs must verify for the same function f, the same input commitments $[X_h]_1$ for $h \in [m]$, and for different output commitments $\mathsf{com}_y \neq \tilde{\mathsf{com}}_y$.

The intuition of the proof is that $\mathcal{A}$ can cheat in three possible ways, for which we define three events E_1, E_2, E_3 as follows:

- E_1 is the event that $\left[Y^{(\gamma)}\right]_1 = \left[\tilde{Y}^{(\gamma)}\right]_1$. As $\mathsf{com}_y \neq \tilde{\mathsf{com}}_y$, this implies an evaluation binding break in the linear map commitment proof in equation (4.9).
- E_2 is the event that E_1 does not happen (i.e., $\left[Y^{(\gamma)}\right]_1 \neq \left[\tilde{Y}^{(\gamma)}\right]_1$) and that $\left[X_{h^*}^{(\beta)}\right]_1 \neq \left[\tilde{X}_{h^*}^{(\beta)}\right]_1$ for some $h^* \in \mathcal{S}_2(f)$. This means that the proofs $\left[\pi_{h^*}^{(\beta)}\right]_1, \left[\tilde{\pi}_{h^*}^{(\beta)}\right]_1$ open the commitment com_{h^*} to two different output commitments for the identity function, which breaks evaluation binding in equation (4.7).
- E_3 is the event that neither E_1 nor E_2 occur. In this case, we will show that evaluation binding breaks in equation (4.11).

For any of these events, we will use $\mathcal{A}$'s output to break the ℓ-HiKer assumption if this is embedded into ck. For this embedding, $\mathcal{B}$ makes a guess $\hat{s} \in \{0, 1\}$ such that $\hat{s} = 0$ corresponds to a guess that event E_1 occurs while $\hat{s} = 1$ corresponds to a guess that either E_2 or E_3 will occur. This $\hat{s}$ is perfectly hidden to $\mathcal{A}$.

Next we describe how to build $\mathcal{B}$ out of $\mathcal{A}$.

Commitment key generation. Let $\mathcal{B}$ be an adversary against the ℓ-HiKer assumption. $\mathcal{B}$ uniformly samples a value $\hat{s} \leftarrow\$ \{0, 1\}$ and simulates ck as follows.

Case $\hat{s} = 0$. $\mathcal{B}$ samples $\boldsymbol{\alpha}, \boldsymbol{\beta} \leftarrow\$ \mathbb{F}^{\ell}, \eta_\beta, \eta_\gamma \leftarrow\$ \mathbb{F}$ and implicitly sets $\boldsymbol{\gamma} := \boldsymbol{\sigma}$ and $\eta_\alpha := \eta$ from the input of the assumption. It is easy to see that this implicit setting allows $\mathcal{B}$ to compute all the elements in the first row of ck, namely:

$$[\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \boldsymbol{\alpha} \otimes \boldsymbol{\beta}]_1, [\boldsymbol{\alpha}, \eta_\alpha, \eta_\beta, \eta_\gamma]_2$$

We show how $\mathcal{B}$ can simulate the remaining elements in the second and third rows of ck starting from the inputs from the ℓ-HiKer assumption as follows:

$$\begin{aligned}
\forall i, i' \in [\ell], i \neq i' : \quad & \alpha_i \left[\tfrac{\eta \sigma_{i'}}{\sigma_i}\right]_1 &&= \left[\alpha_i \frac{\gamma_{i'}}{\gamma_i} \eta_\alpha\right]_1 \\
& \tfrac{\alpha_{i'}}{\alpha_i} \beta_i \eta_\beta [1]_1 &&= \left[\frac{\alpha_{i'}}{\alpha_i} \beta_i \eta_\beta\right]_1 \\
\forall i, j, i', j', k \in [\ell] : (i,j) \neq (i',j') : \quad & \tfrac{\alpha_{i'}\beta_{j'}}{\alpha_i \beta_j} \eta_\gamma [\sigma_k]_1 &&= \left[\frac{\alpha_{i'}\beta_{j'}}{\alpha_i \beta_j} \gamma_k \eta_\gamma\right]_1 \\
\forall i \in [\ell] : \quad & \alpha_i \left[\tfrac{\eta}{\sigma_i}\right]_2 &&= \left[\frac{\alpha_i \eta_\alpha}{\gamma_i}\right]_2 \\
& \tfrac{\beta_i \eta_\beta}{\alpha_i} [1]_2 &&= \left[\frac{\beta_i \eta_\beta}{\alpha_i}\right]_2 \\
\forall i, k \in [\ell] : \quad & \tfrac{\eta_\gamma}{\alpha_i} [\sigma_k]_2 &&= \left[\frac{\gamma_k \eta_\gamma}{\alpha_i}\right]_2 \\
\forall i, j, k \in [\ell] : \quad & \tfrac{\eta_\gamma}{\alpha_i \beta_j} [\sigma_k]_2 &&= \left[\frac{\gamma_k \eta_\gamma}{\alpha_i \beta_j}\right]_2
\end{aligned}$$

As one can notice, in this case of $\hat{s} = 0$ we embed in the commitment key only a subset of the elements of the assumption. This means that the reduction for adversaries causing event E_1 can actually be done based on a weaker version of the assumption which includes only the subset of the elements that we need for this case.

Case $\hat{s} = 1$. $\mathcal{B}$ samples $\eta_\alpha, r_\beta, r_\gamma \leftarrow\$ \mathbb{F}$ and $\boldsymbol{\gamma} \leftarrow\$ \mathbb{F}^\ell$ and implicitly sets $\boldsymbol{\alpha} := \boldsymbol{\sigma}, \boldsymbol{\beta} := \boldsymbol{\tau}, \eta_\beta := r_\beta \cdot \eta, \eta_\gamma := r_\gamma \cdot \eta$. As for the case of $\hat{s} = 0$, it is easy to see that this implicit setting allows $\mathcal{B}$ to compute all the elements in the first row of ck, namely $[\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \boldsymbol{\alpha} \otimes \boldsymbol{\beta}]_1, [\boldsymbol{\alpha}, \eta_\alpha, \eta_\beta, \eta_\gamma]_2$.

Next, we show how $\mathcal{B}$ can simulate the remaining elements in the second and third rows of ck starting from the inputs from the ℓ-HiKer assumption as follows:

$$\begin{aligned}
\forall i, i' \in [\ell], i \neq i' : \quad & \tfrac{\gamma_{i'}}{\gamma_i} \eta_\alpha [\sigma_i]_1 &&= \left[\alpha_i \frac{\gamma_{i'}}{\gamma_i} \eta_\alpha\right]_1 \\
& r_\beta \left[\eta \tfrac{\sigma_{i'}}{\sigma_i} \tau_i\right]_1 &&= \left[\frac{\alpha_{i'}}{\alpha_i} \beta_i \eta_\beta\right]_1 \\
\forall i, j, i', j', k \in [\ell] : (i,j) \neq (i',j') : \quad & r_\gamma \gamma_k \left[\eta \tfrac{\sigma_{i'}\tau_{j'}}{\sigma_i \tau_j}\right]_1 &&= \left[\frac{\alpha_{i'}\beta_{j'}}{\alpha_i \beta_j} \gamma_k \eta_\gamma\right]_1 \\
\forall i \in [\ell] : \quad & \tfrac{\eta_\alpha}{\gamma_i} [\sigma_i]_2 &&= \left[\frac{\alpha_i \eta_\alpha}{\gamma_i}\right]_2 \\
& r_\beta \left[\tfrac{\eta \tau_i}{\sigma_i}\right]_2 &&= \left[\frac{\beta_i \eta_\beta}{\alpha_i}\right]_2 \\
\forall i, k \in [\ell] : \quad & r_\gamma \gamma_k \left[\tfrac{\eta}{\sigma_i}\right]_2 &&= \left[\frac{\gamma_k \eta_\gamma}{\alpha_i}\right]_2 \\
\forall i, j, k \in [\ell] : \quad & r_\gamma \gamma_k \left[\tfrac{\eta}{\sigma_i \tau_j}\right]_2 &&= \left[\frac{\gamma_k \eta_\gamma}{\alpha_i \beta_j}\right]_2
\end{aligned}$$

Execution of $\mathcal{A}$. Once having generated ck as described above, $\mathcal{B}$ runs $\mathcal{A}(\mathsf{ck})$, receives the output $((\mathsf{com}_h)_{h \in [m]}, \mathsf{com}_y, f, \pi, \tilde{\mathsf{com}}_y, \tilde{\pi})$ and parses the proofs as before. Notice that

ck is perfectly distributed as the one generated by Setup and thus the value $\hat{s}$ is perfectly hidden to $\mathcal{A}$.

The reduction proceeds differently according to the output produced by $\mathcal{A}$, that we split in the events E_1, E_2, E_3 as defined above.

E_1 **occurs:** If $\hat{s} \neq 0$, then $\mathcal{B}$ aborts. Otherwise it proceeds as follows. Recall that in this case we have that as $\left[Y^{(\gamma)}\right]_1 = \left[\tilde{Y}^{(\gamma)}\right]_1$, then $\left[\pi^{(\alpha)}\right]_1, \left[\tilde{\pi}^{(\alpha)}\right]_1$ open to different $\mathsf{com}_y, \tilde{\mathsf{com}}_y$. Therefore, by equation (4.9) we have that

$$\left[\pi^{(\alpha)}\right]_1, [1]_2\, \mathsf{com}_y, \left[\eta_\alpha\right]_2 = \left[Y^{(\gamma)}\right]_1 \cdot \sum_{i\in[\ell]} \left[\frac{\alpha_i \eta_\alpha}{\gamma_i}\right]_2 = \left[\tilde{\pi}^{(\alpha)}\right]_1 \cdot [1]_2\, \tilde{\mathsf{com}}_y \cdot \left[\eta_\alpha\right]_2$$

Then, $\mathcal{B}$ returns (U, V) such that

$$[U]_1 := \tilde{\mathsf{com}}_y - \mathsf{com}_y,\ [V]_1 := \left[\pi^{(\alpha)}\right]_1 - \left[\tilde{\pi}^{(\alpha)}\right]_1.$$

If $\mathcal{B}$ did not abort, then $\hat{s} = 0$. Thus, $\eta_\alpha = \eta$ and $[U]_1 \cdot \left[\eta\right]_2 = [V]_1 \cdot [1]_2$.

E_2 **occurs:** If $\hat{s} \neq 1$, then $\mathcal{B}$ aborts. Otherwise, let h^* be some index such that $\left[X^{(\beta)}_{h^*}\right]_1 \neq \left[\tilde{X}^{(\beta)}_{h^*}\right]_1$; note that h^* must exist by definition of event E_2. Similarly as before, from equation (4.7) we have that

$$\left[\pi^{(\beta)}_{h^*}\right]_1, [1]_2 \left[X^{(\beta)}_{h^*}\right]_1, \left[\eta_\beta\right]_2 = X_{h^*}, \sum_{i\in[\ell]} \left[\frac{\beta_i \eta_\beta}{\alpha_i}\right]_2 = \left[\tilde{\pi}^{(\beta)}_{h^*}\right]_1, [1]_2 \left[\tilde{X}^{(\beta)}_{h^*}\right]_1, \left[\eta_\beta\right]_2.$$

Then, $\mathcal{B}$ returns $([U]_1, V)$ such that

$$[U]_1 := r_\beta \cdot \left(\left[\tilde{X}^{(\beta)}_{h^*}\right]_1 - \left[X^{(\beta)}_{h^*}\right]_1\right),\ [V]_1 := \left[\pi^{(\beta)}_{h^*}\right]_1 - \left[\tilde{\pi}^{(\beta)}_{h^*}\right]_1.$$

If $\mathcal{B}$ did not abort, then $\hat{s} = 1$. Thus, $\eta_\beta = r_\beta \cdot \eta$ and $[U]_1 \cdot [\eta]_2 = [V]_1 \cdots [1]_2$.

E_3 **occurs:** If $\hat{s} \neq 1$, then $\mathcal{B}$ aborts. Otherwise, $\mathcal{B}$ proceeds as follows. First, note that since E_1 and E_2 did not occur, then $\left[Y^{(\gamma)}\right]_1 = \left[\tilde{Y}^{(\gamma)}\right]_1$ and $\left[X^{(\beta)}_h\right]_1 = \left[\tilde{X}^{(2)}_h\right]_2$ for every $h \in \mathcal{S}_2(f)$. Also, by equation (4.6) and by the non-degeneracy of the pairing, we have

$$[X_h]_1 \cdot [1]_2 = [1]_1 \cdot \left[X^{(2)}_h\right]_2 = [1]_1 \cdot \left[\tilde{X}^{(2)}_h\right]_2 \quad \text{which implies that } \left[X^{(2)}_h\right]_2 = \left[\tilde{X}^{(2)}_h\right]_2.$$

From the equality above we can use equation (4.8) to also conclude that $\left[Z_{h,h'}\right]_1 = \tilde{Z}_{h,h'}$ for all $(h, h') \in \mathcal{S}^{\otimes}_2(f)$. Then, since both proofs satisfy equation (4.11), we have

$$\begin{aligned}\left[\pi^{(\gamma)}\right]_1 \cdot [1]_2 \left(\left[Y^{(\gamma)}\right]_1 - [\Theta]_1\right) \cdot [\eta_\gamma]_2 &= \sum_{h\in\mathcal{S}_1(f)} [X_h]_1, [\Phi_h]_2 \cdot \sum_{(h,h')\in\mathcal{S}^{\otimes}_2(f)} \left[Z_{h,h'}\right]_1 \cdot \left[\Gamma_{h,h'}\right]_2 \\ &= \left[\tilde{\pi}^{(\gamma)}\right]_1 \cdot [1]_2 \left(\left[\tilde{Y}^{(\gamma)}\right]_1 - [\Theta]_1\right) \cdot [\eta_\gamma]_2.\end{aligned}$$

The reduction returns $([U]_1, [V]_1)$ computed as follows:

$$[U]_1 := r_\gamma \cdot \left(\left[\tilde{Y}^{(\gamma)}\right]_1 - \left[Y^{(\gamma)}\right]_1\right),\ [V]_1 := \left(\left[\pi^{(\gamma)}\right]_1 - \left[\tilde{\pi}^{(\gamma)}\right]_1\right).$$

If $\mathcal{B}$ did not abort, then $\hat{s} = 1$ and $\eta_\gamma = r_\gamma \cdot \eta$. Thus, $[U]_1 \cdot [\eta]_2 = [V]_1 \cdot [1]_2$. Since $\hat{s}$ is perfectly hidden $\mathcal{B}$ aborts with probability $1/2$. Hence, if $\mathcal{A}$ is successful with probability ϵ, then $\mathcal{B}$ breaks the assumption with probability $\epsilon/2$.

4.6.7 Efficient Verification

Our chainable functional commitment scheme CFC supports amortized efficient verification. We define the algorithms PreFuncVer and EffFuncVer below, following Theorem 4.10.

$\underline{\mathsf{PreFuncVer}(\mathsf{ck}, f)}$ Parse ck and compute the encodings $[\Theta]_1, [\Phi_h]_2, [\Gamma_{h,h'}]_2$ of f as done in the CFC.FuncVer algorithm following equation (4.10). Also, compute the encodings in equations (4.7) and (4.9), $[\Psi^{(\beta)}]_2 = \sum_{i\in[\ell]} \left[\frac{\beta_i \eta_\beta}{\alpha_i}\right]_2$ and $[\Psi^{(\alpha)}]_2 = \sum_{i\in[\ell]} \left[\frac{\alpha_i \eta_\alpha}{\gamma_i}\right]_2$.
Output $\mathsf{ck}_f := (\{[\Theta]_1, [\Phi_h]_2, [\Gamma_{h,h'}]_2\}_{(h,h')\in\mathcal{S}_2^{\otimes}(f)}, [\Psi^{(\alpha)}]_2, [\Psi^{(\beta)}]_2)$.

$\underline{\mathsf{EffFuncVer}(\mathsf{ck}_f, (\mathsf{com}_h)_{h\in[m]}, \mathsf{com}_y, \pi)}$ Parse ck_f, π and carry out all the pairing checks in the FuncVer algorithm, i.e., verify equations (4.6), (4.7), (4.8), (4.9), (4.11).

Following the description of succinctness in Section 4.6.4, given any $f \in \mathcal{F}_{\mathsf{quad},\kappa}$ then EffFuncVer needs to parse a proof that has $O(\kappa)$ group elements. Then, it verifies $\omega \le \kappa$ pairing checks in equations (4.6) and (4.7), κ checks in equation (4.8), a single check in equation (4.7), and a single check involving $\kappa + \omega$ products in equation (4.11). Assuming that the running time of each pairing computation is bounded by some polynomial $p(\lambda) = \mathsf{poly}(\lambda)$, the running time of EffFuncVer is therefore $O(p(\lambda)\cdot|\kappa|) = O(p(\lambda)\cdot|\pi|)$, which is essentially optimal.

4.6.8 Commitment Hiding

Our CFC construction can be made perfectly commitment hiding (as in Theorem 3.1) by adding randomness to the commitment. We describe the transformation $\widetilde{\mathsf{CFC}} = (\widetilde{\mathsf{Setup}}, \widetilde{\mathsf{Com}}, \widetilde{\mathsf{FuncProve}}, \widetilde{\mathsf{FuncVer}})$ below.

$\underline{\widetilde{\mathsf{Setup}}(1^\lambda, 1^\ell)}$ Output $\tilde{\mathsf{ck}} \leftarrow \mathsf{Setup}(1^\lambda, 1^{\ell+1})$.

$\underline{\widetilde{\mathsf{Com}}(\tilde{\mathsf{ck}}, \boldsymbol{x})}$ Let $r \leftarrow\$\ \mathbb{F}$. Output $(\mathsf{com}, \mathsf{aux})$ where $\mathsf{com} \leftarrow \mathsf{Com}(\tilde{\mathsf{ck}}, \boldsymbol{x}) + r \cdot [\alpha_{i+1}]_1$ and $\mathsf{aux} = (\boldsymbol{x}, r)$.

$\underline{\widetilde{\mathsf{FuncProve}}(\tilde{\mathsf{ck}}, (\mathsf{aux}_i)_{i\in[m]}, f)}$ Let $\mathsf{aux}_i = (\boldsymbol{x}^{(i)}, r^{(i)})$. Output $\mathsf{FuncProve}(\tilde{\mathsf{ck}}, (\mathsf{aux})_{i\in[m]}, f')$ where $f' = (f, 0)$.

$\underline{\widetilde{\mathsf{FuncVer}}(\tilde{\mathsf{ck}}, (\mathsf{com}_i)_{i\in[m]}, \mathsf{com}_y, f, \pi)}$ Output $\mathsf{FuncVer}(\tilde{\mathsf{ck}}, (\mathsf{com}_i)_{i\in[m]}, \mathsf{com}_y, f, \pi)$.

For the above scheme, it is easy to construct a simulator Sim as follows.

$\underline{\mathsf{Sim}_{\mathsf{Setup}}(1^\lambda, \ell)}$ Sample $\boldsymbol{\alpha} \leftarrow\$\ \mathbb{F}^{\ell+1}$ and generate $\tilde{\mathsf{ck}}$ as in $\mathsf{Setup}(1^\lambda, \ell+1)$, sampling additional field elements when necessary. Output $(\tilde{\mathsf{ck}}, \mathsf{td})$ where $\mathsf{td} = \boldsymbol{\alpha}$.

$\underline{\mathsf{Sim}_{\mathsf{Com}}(\mathsf{td})}$ Sample $r \leftarrow\$\ \mathbb{F}$ and output $(\mathsf{com}, \mathsf{aux})$ where $\mathsf{com} = r \cdot [\alpha_{\ell+1}]_1$ and $\mathsf{aux} = (\mathbf{0}, r)$.

$\underline{\mathsf{Sim}_{\mathsf{Equiv}}(\mathsf{td}, \mathsf{com}, \mathsf{aux}, \boldsymbol{x})}$ The algorithm uses the field elements in $\boldsymbol{\alpha}$ to find a value $r' \in \mathbb{F}$ such that $\mathsf{com} = \widetilde{\mathsf{Com}}(\boldsymbol{x}, r')$. It simply obtains the solution r' of the linear equation $\langle \boldsymbol{x}, \boldsymbol{\alpha}\rangle + \alpha_{\ell+1} r' = \alpha_{\ell+1} r$ and outputs $\mathsf{aux} = (\boldsymbol{x}, r')$.

4.7 Lattice-based CFC for Quadratic Functions

In this section, we present a lattice-based construction of a CFC for quadratic functions. Our construction can be seen as a lattice-analogue of the pairing-based scheme presented in Section 4.6 obtained via a slight generalisation of the translation technique in [ACL$^+$22]. We recall that a background on lattices is provided in Section 3.3.

4.7.1 Hardness Assumptions

The k-R-ISIS assumption family[14] was recently introduced in [ACL$^+$22] as a natural extension of the standard short integer solution (SIS) assumption and a natural lattice-analogue of a certain class of pairing-based assumptions. The k-R-ISIS family was accompanied by a translation technique outlined in [ACL$^+$22] for translating pairing-based schemes and assumptions to their lattice-analogues.

For instance, a certain k-R-ISIS assumption could be parametrised by a set $\mathcal{G}$ of monomials. It states that even when given short preimages $\boldsymbol{u}_g$ satisfying $\mathbf{A} \cdot \boldsymbol{u}_g = \boldsymbol{t} \cdot g(\boldsymbol{v}) \bmod q$ for all $g \in \mathcal{G}$, it is hard to find a short non-zero preimage $\boldsymbol{u}^*$ satisfying $\mathbf{A} \cdot \boldsymbol{u}^* = \mathbf{0} \bmod q$.

Applying the translation technique in [ACL$^+$22] to the pairing-based assumption (Definition 4.15) which underlies the security of the pairing-based CFC construction, we encounter an obstacle that there is no translation for the term $[\eta]_2$ in the challenge relation $e(U, [\eta]_2) = e(V, [1]_2)$.

To overcome the above obstacle, in the following, we introduce (a special case of) a generalisation of the k-R-ISIS assumption which we call the Twin-k-R-ISIS assumption. In a nutshell, instead of a single set $\mathcal{G}$ of monomials, we now have two (or in general more) sets $\mathcal{G}_A$ and $\mathcal{G}_B$ of non-overlapping monomials. The Twin-k-R-ISIS assumption states that even when given short preimages $\boldsymbol{u}_g$ satisfying $\mathbf{A} \cdot \boldsymbol{u}_g = \boldsymbol{t} \cdot g(\boldsymbol{v}) \bmod q$ for all $g \in \mathcal{G}_A$ and short preimages $\boldsymbol{w}_g$ satisfying $\mathbf{B} \cdot \boldsymbol{u}_g = \boldsymbol{t} \cdot g(\boldsymbol{v}) \bmod q$ for all $g \in \mathcal{G}_B$, it is hard to find a short non-zero preimage $(\boldsymbol{u}^*, \boldsymbol{w}^*)$ satisfying $\mathbf{A} \cdot \boldsymbol{u}^* + \mathbf{B} \cdot \boldsymbol{w}^* = \mathbf{0} \bmod q$. We stress that the non-overlapping requirement of $\mathcal{G}_A$ and $\mathcal{G}_B$ is crucial, for otherwise $(\boldsymbol{u}_g, -\boldsymbol{w}_g)$ would be a trivial solution for any $g \in \mathcal{G}_A \cap \mathcal{G}_B$. Other than this trivial attack (which is ruled out), the (failed) attack strategies discussed in [ACL$^+$22] against the k-R-ISIS assumption also fail against the Twin-k-R-ISIS assumption.[15]

Definition 4.19 (Twin-k-R-ISIS Assumption)**.** *Let $\zeta, \eta \in \mathbb{N}$, q be a rational prime, $\beta, \beta^* \in \mathbb{R}^+$,*

$$\mathcal{G}_A := \left\{ \frac{X_{i'}}{X_i} \cdot \bar{X}_k,\ \frac{X_{i'}}{X_i} \cdot \check{X}_k,\ \frac{\bar{X}_{i'}}{\bar{X}_i} \cdot X_k \right\}_{i,i',k \in [\ell], i \neq i'} \cup \left\{ \frac{X_{i'} \cdot \check{X}_{j'}}{X_i \cdot \check{X}_j} \cdot \bar{X}_k \right\}_{\substack{i,i',j,j',k \in [\ell] \\ i \neq i', j \neq j'}},$$

$\mathcal{G}_B := \left\{ X_k, \bar{X}_k, \check{X}_k \right\}_{k \in [\ell]}$, and $\mathcal{G} := \mathcal{G}_A \cup \mathcal{G}_B$. Let $\mathcal{D}$ be a distribution over $\mathcal{R}^\zeta$. Write $\mathsf{pp} :=$ *$(\mathcal{R}_q, \eta, \zeta, n, \beta, \beta^*, \mathcal{G}_A, \mathcal{G}_B, \mathcal{D})$. The k-R-$\mathsf{ISIS}_{\mathsf{pp}}$ assumption states that for any PPT adversary $\mathcal{A}$*

[14] We use k-R-ISIS to refer to both the ring and module version. In [ACL$^+$22], the module version is given the name k-M-ISIS.

[15] A subsequent work by Albrecht, Fenzi, Lapiha and Nguyen [AFLN24] actually proves that there exists a reduction from the k-R-ISIS assumption to the Twin-k-R-ISIS assumption.

we have $\mathsf{Adv}^{\text{k-r-isis}}_{\mathsf{pp},\mathcal{A}}(\lambda) \leq \mathsf{negl}(\lambda)$, *where* $\mathsf{Adv}^{\text{k-r-isis}}_{\mathsf{pp},\mathcal{A}}(\lambda)$ *is given by*

$$\Pr\left[\begin{array}{l|l} & \mathbf{A} \leftarrow\$ \mathcal{R}_q^{\eta\times\zeta} \bmod q;\ \mathbf{B} \leftarrow\$ \mathcal{R}_q^{\eta\times\zeta} \bmod q \\ & \boldsymbol{t} \leftarrow\$ (\mathcal{R}_q^{\times})^{\eta};\ \boldsymbol{v}, \bar{\boldsymbol{v}}, \check{\boldsymbol{v}} \leftarrow\$ (\mathcal{R}^{\times})^{n} \\ \mathbf{A}\cdot\boldsymbol{u}^* + \mathbf{B}\cdot\boldsymbol{w}^* \equiv \mathbf{0} \bmod q & \boldsymbol{u}_g \leftarrow\$ \mathcal{D} : \mathbf{A}\cdot\boldsymbol{u}_g \equiv \boldsymbol{t}\cdot g(\boldsymbol{v},\bar{\boldsymbol{v}},\check{\boldsymbol{v}}) \bmod q,\ \forall g \in \mathcal{G}_A \\ \wedge\ 0 < \|(\boldsymbol{u}^*, \boldsymbol{w}^*)\| \leq \beta^* & \boldsymbol{w}_g \leftarrow\$ \mathcal{D} : \mathbf{B}\cdot\boldsymbol{w}_g \equiv \boldsymbol{t}\cdot g(\boldsymbol{v},\bar{\boldsymbol{v}},\check{\boldsymbol{v}}) \bmod q,\ \forall g \in \mathcal{G}_B \\ & (\boldsymbol{u}^*, \boldsymbol{v}^*) \leftarrow \mathcal{A}\left(\mathbf{A},\mathbf{B},\boldsymbol{t},\boldsymbol{v},\bar{\boldsymbol{v}},\check{\boldsymbol{v}},\left\{\boldsymbol{u}_{\mathcal{G}_A},\boldsymbol{w}_{\mathcal{G}_B}\right\}\right) \end{array}\right].$$

4.7.2 Construction

In the following, we construct a lattice-based chainable functional commitment scheme. Our construction is parametrised by a ring $\mathcal{R}$, dimensions η, ζ, modulus q, norm bound β, an input length ℓ, and the number of inputs m. Before describing the construction, we first introduce the following shorthands and notation.

For a quadratic polynomial map $f : \mathcal{R}^{mn} \to \mathcal{R}^{\ell}$, we express $f(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m)$

$$f(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m) = \boldsymbol{e} + \sum_{h\in\mathcal{S}_1(f)} \mathbf{F}_h \cdot \boldsymbol{x}_h + \sum_{(h,h')\in\mathcal{S}_2^{\otimes}(f)} \mathbf{G}_{h,h'} \cdot (\boldsymbol{x}_h \otimes \boldsymbol{x}_{h'})$$

for some $\mathbf{G}_{h,h'} \in \mathcal{R}^{\ell\times\ell^2}$, $\mathbf{F}_h \in \mathcal{R}^{\ell\times\ell}$, and $\boldsymbol{e} \in \mathcal{R}^{\ell}$, similarly to previous sections.

Different from the pairing-based construction, our lattice-based construction is additionally parametrised by a norm bound $\alpha \in \mathbb{R}^+$. We assume that messages $\boldsymbol{x}$ and each coefficient of any quadratic polynomial map f to be opened have norm at most α, and f is such that for any $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m$ of norm at most α, it holds that $\left\|f(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m)\right\| \leq \alpha$.

For a vector $\boldsymbol{v} \in (\mathcal{R}_q^{\times})^{\ell}$, denote its component-wise inverse by $\boldsymbol{v}^{\dagger} := (v_i^{-1})_{i=1}^{\ell}$. Define $\mathbf{Z}_{\boldsymbol{v}} := \boldsymbol{v}^{\dagger} \cdot \boldsymbol{v}^{\mathtt{T}} - \mathbf{I} = (z_{i,j})_{i,j}$ where

$$z_{i,j} = \begin{cases} 0 & i = j \\ v_i^{-1} \cdot v_j & i \neq j \end{cases}.$$

We are now ready to describe the construction as follows.

<u>$\mathsf{Setup}(1^{\lambda}, 1^{\ell})$</u>

- Sample trapdoored matrices $(\mathbf{A}, \mathsf{td}_{\mathbf{A}}), (\mathbf{B}, \mathsf{td}_{\mathbf{B}}) \leftarrow \mathsf{TrapGen}(\mathcal{R}, 1^{\eta}, 1^{\zeta}, q, \beta)$.
- Sample submodule generator $\boldsymbol{t} \leftarrow\$ (\mathcal{R}_q^{\times})^{\eta}$.
- Sample commitment key vectors $\boldsymbol{v}, \bar{\boldsymbol{v}}, \check{\boldsymbol{v}} \leftarrow\$ \mathcal{R}_q^{\ell}$.
- Sample a short preimage $\boldsymbol{u}_g \leftarrow \mathsf{SampPre}(\mathsf{td}_{\mathbf{A}}, \boldsymbol{t} \cdot g(\boldsymbol{v}, \bar{\boldsymbol{v}}, \check{\boldsymbol{v}}) \bmod q)$ for each $g \in \mathcal{G}_A$, where

$$\mathcal{G}_A := \left\{\frac{X_{i'}}{X_i}\cdot\bar{X}_k,\ \frac{X_{i'}}{X_i}\cdot\check{X}_k,\ \frac{\bar{X}_{i'}}{\bar{X}_i}\cdot X_k\right\}_{i,i',k\in[\ell],i\neq i'} \cup \left\{\frac{X_{i'}\cdot\check{X}_{j'}}{X_i\cdot\check{X}_j}\cdot\bar{X}_k\right\}_{\substack{i,i',j,j',k\in[\ell]\\ i\neq i',j\neq j'}}$$

- Sample a short preimage $\boldsymbol{w}_g \leftarrow \mathsf{SampPre}(\mathsf{td}_{\mathbf{B}}, \boldsymbol{t} \cdot g(\boldsymbol{v}, \bar{\boldsymbol{v}}, \check{\boldsymbol{v}}) \bmod q)$ for each $g \in \mathcal{G}_B$, where
$$\mathcal{G}_B := \left\{ \mathrm{X}_k, \bar{\mathrm{X}}_k, \check{\mathrm{X}}_k \right\}_{k \in [\ell]}.$$
- Output $\mathsf{ck} := \left(\mathbf{A}, \mathbf{B}, \boldsymbol{t}, \boldsymbol{v}, \bar{\boldsymbol{v}}, \check{\boldsymbol{v}}, (\boldsymbol{u}_g)_{g \in \mathcal{G}_A}, (\boldsymbol{w}_g)_{g \in \mathcal{G}_B} \right)$.

<u>$\mathsf{Com}(\mathsf{ck}, \boldsymbol{x})$</u>

- Compute $c := \langle \boldsymbol{v}, \boldsymbol{x} \rangle \bmod q$.
- Output $\mathsf{com} = c$ and $\mathsf{aux} = \boldsymbol{x}$.

<u>$\mathsf{FuncProve}(\mathsf{ck}, (\mathsf{aux}_h)_{h \in [m]}, f)$</u>

- Parse aux_h as $\boldsymbol{x}_h$ for all $h \in [m]$ and let $\boldsymbol{y} := f(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m)$.
- Compute $\boldsymbol{v}_1 := \mathsf{vec}(\mathbf{Z}_{\boldsymbol{v}}) \otimes \bar{\boldsymbol{v}}$ and $\boldsymbol{v}_2 := \mathsf{vec}((\mathbf{I} + \mathbf{Z}_{\boldsymbol{v}}) \otimes (\mathbf{I} + \mathbf{Z}_{\check{\boldsymbol{v}}}) - \mathbf{I}) \otimes \bar{\boldsymbol{v}}$.
- Pack the preimages vectors given in the public parameters as columns of the following matrices:
  - $\mathbf{U}_i$ such that $\mathbf{A} \cdot \mathbf{U}_i = \boldsymbol{t} \cdot \boldsymbol{v}_i^{\mathrm{T}} \bmod q$ for $i \in [2]$.
    For example, for $i = 1$, the first few columns of the R.H.S. of the equation are of the form
    $$\boldsymbol{t} \cdot \boldsymbol{v}_1^{\mathrm{T}} = \boldsymbol{t} \cdot \begin{pmatrix} 0 & \frac{v_1}{v_2} \cdot \bar{v}_1 & \frac{v_1}{v_3} \cdot \bar{v}_1 & \ldots \end{pmatrix}.$$
    Notice that each column is either $\mathbf{0} \in \mathcal{R}_q^{\eta}$, for which $\mathbf{0} \in \mathcal{R}^{\zeta}$ is a trivial preimage, or of the form $\boldsymbol{t} \cdot \frac{v_{i'}}{v_i} \cdot \bar{v}_k$ for some $i, i', k \in [\ell]$ with $i \neq i'$, for which a preimage is given in ck.
  - $\bar{\mathbf{U}}$ such that $\mathbf{A} \cdot \bar{\mathbf{U}} = \boldsymbol{t} \cdot \boldsymbol{v}^{\mathrm{T}} \cdot \mathbf{Z}_{\bar{\boldsymbol{v}}} \bmod q$.
  - $\check{\mathbf{U}}$ such that $\mathbf{A} \cdot \check{\mathbf{U}} = \boldsymbol{t} \cdot \check{\boldsymbol{v}}^{\mathrm{T}} \cdot \mathbf{Z}_{\boldsymbol{v}} \bmod q$.
  - $\mathbf{W}$ such that $\mathbf{B} \cdot \mathbf{W} = \boldsymbol{t} \cdot \boldsymbol{v}^{\mathrm{T}} \bmod q$.
  - $\bar{\mathbf{W}}$ such that $\mathbf{B} \cdot \bar{\mathbf{W}} = \boldsymbol{t} \cdot \bar{\boldsymbol{v}}^{\mathrm{T}} \bmod q$.
  - $\check{\mathbf{W}}$ such that $\mathbf{B} \cdot \check{\mathbf{W}} = \boldsymbol{t} \cdot \check{\boldsymbol{v}}^{\mathrm{T}} \bmod q$.
- Compute $\boldsymbol{u} := \sum_{h \in \mathcal{S}_1(f)} \mathbf{U}_1 \cdot \mathsf{vec}(\boldsymbol{x}_h^{\mathrm{T}} \otimes \mathbf{F}_h) + \sum_{(h,h') \in \mathcal{S}_2^{\otimes}(f)} \mathbf{U}_2 \cdot \mathsf{vec}((\boldsymbol{x}_h^{\mathrm{T}} \otimes \boldsymbol{x}_{h'}^{\mathrm{T}}) \otimes \mathbf{G}_{h,h'})$.
- Compute $\boldsymbol{w}_0 := \mathbf{W} \cdot \boldsymbol{y}$.
- Compute $\bar{\boldsymbol{u}}_0 := \bar{\mathbf{U}} \cdot \boldsymbol{y}$ and $\bar{\boldsymbol{w}}_0 := \bar{\mathbf{W}} \cdot \boldsymbol{y}$.
- Compute $\check{\boldsymbol{u}}_h := \check{\mathbf{U}} \cdot \boldsymbol{x}_h$ and $\check{\boldsymbol{w}}_h := \check{\mathbf{W}} \cdot \boldsymbol{x}_h$ for $h \in \mathcal{S}_2(f)$.
- Output $(\boldsymbol{u}, \boldsymbol{w}_0, \bar{\boldsymbol{u}}_0, \bar{\boldsymbol{w}}_0, (\check{\boldsymbol{u}}_h, \check{\boldsymbol{w}}_h)_{h \in \mathcal{S}_2(f)})$.

<u>$\mathsf{FuncVer}(\mathsf{ck}, (\mathsf{com}_h)_{h \in [m]}, \mathsf{com}_0, f, \pi)$</u>

- Define $\hat{f}(C_1, \ldots, C_m, \check{C}_1, \ldots, \check{C}_m)$
$$:= \bar{\boldsymbol{v}}^{\mathrm{T}} \cdot \left(\sum_{(h,h') \in \mathcal{S}_2(f)} \mathbf{G}_{h,h'} \cdot (\boldsymbol{v}^{\dagger} \otimes \check{\boldsymbol{v}}^{\dagger}) \cdot C_h \cdot \check{C}_{h'} + \sum_{h \in \mathcal{S}_1(f)} \mathbf{F}_h \cdot \boldsymbol{v}^{\dagger} \cdot C_h + \boldsymbol{e}^{\mathrm{T}} \right).$$
- Check if $\|\boldsymbol{w}_0\| \leq \beta^*$ and $\|\bar{\boldsymbol{w}}_0\| \leq \beta^*$.

- For $h \in [m] \setminus \mathcal{S}_2(f)$, set $\check{c}_h = 0$ and check if $\|\check{\boldsymbol{w}}_h\| \le \beta^*$.
- Check if $\mathbf{B} \cdot \boldsymbol{w}_0 = \boldsymbol{t} \cdot c_0 \bmod q$.
- Check if there exists (unique) $\bar{c}_0$ such that $\mathbf{B} \cdot \bar{\boldsymbol{w}}_0 = \boldsymbol{t} \cdot \bar{c}_0 \bmod q$.
- Check if there exists (unique) $\check{c}_h$ such that $\mathbf{B} \cdot \check{\boldsymbol{w}}_h = \boldsymbol{t} \cdot \check{c}_h \bmod q$ for $h \in \mathcal{S}_2(f)$.
- Check if $\mathbf{A} \cdot \boldsymbol{u} = \boldsymbol{t} \cdot (\hat{f}(c_1, \ldots, c_m, \check{c}_1, \ldots, \check{c}_m) - \bar{c}_0) \bmod q$ and $\|\boldsymbol{u}\| \le \beta^*$.
- Check if $\mathbf{A} \cdot \bar{\boldsymbol{u}}_0 = \boldsymbol{t} \cdot (\boldsymbol{v}^{\mathrm{T}} \cdot \bar{\boldsymbol{v}}^{\dagger} \cdot \bar{c}_0 - c_0) \bmod q$ and $\|\bar{\boldsymbol{u}}_0\| \le \beta^*$.
- Check if $\mathbf{A} \cdot \check{\boldsymbol{u}}_h = \boldsymbol{t} \cdot (\check{\boldsymbol{v}}^{\mathrm{T}} \cdot \boldsymbol{v}^{\dagger} \cdot c_h - \check{c}_h) \bmod q$ and $\|\check{\boldsymbol{u}}_h\| \le \beta^*$ for $h \in \mathcal{S}_2(f)$.
- Accept, i.e. output 1, if all checks pass. Otherwise, output 0.

Theorem 4.20. *Let $\zeta \ge \mathsf{lhl}(\mathcal{R}, \eta, q, \beta)$, $\beta^* \ge 2 \cdot \ell^4 \cdot \hat{m}^2 \cdot \alpha^3 \cdot \beta \cdot \gamma_{\mathcal{R}}^3$, and $\mathcal{D} = \mathsf{SampD}(\mathcal{R}, 1^\eta, 1^\zeta, q, \beta)$, and assume that the twin-k-R-$\mathsf{ISIS}_{\mathcal{R}_q,\eta,\zeta,n,\beta,\beta^*,\mathcal{G}_A,\mathcal{G}_B,\mathcal{D}}$ assumption holds.*

Then, the construction CFC described above is an evaluation binding CFC for the class $\mathcal{F}_{\mathsf{quad}}$ of quadratic functions over any $m \le \hat{m}$ vectors of length $\le \ell$, has efficient verification, and is (almost) additively homomorphic. For a function $f \in \mathcal{F}_{\mathsf{quad}}$, the proof size of CFC is $|\pi| = |\mathcal{S}_2(f)| \cdot \log^2(m \cdot \ell) \cdot \mathsf{poly}(\lambda)$, and for the class $\mathcal{F}_{\mathsf{level}} = \{\mathcal{F}_{\mathsf{level},\kappa}\}$, our CFC is $s(\ell, m, \kappa)$-succinct where $s(\ell, m, \kappa) = \kappa \cdot \log^2(m \cdot \ell)$. Furthermore, by setting $\hat{m} = \lambda^{\omega(1)}$ the CFC supports quadratic functions over any $m = \mathsf{poly}(\lambda)$ vectors and is $\kappa \cdot \log^2(\ell)$-succinct.

In the following sections we prove the theorem.

4.7.3 Correctness

To prove correctness, we first state a claim which abstracts away most of the tedious calculations.

Claim 4.21. *Let $f(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m) = \boldsymbol{y}$. For $h \in \mathcal{S}(f)$, let $c_h = \langle \boldsymbol{v}, \boldsymbol{x}_h \rangle \bmod q$. For $h \in \mathcal{S}_2(f)$, let $\check{c}_h = \langle \check{\boldsymbol{v}}, \boldsymbol{x}_h \rangle \bmod q$. For $h \in [m] \setminus \mathcal{S}_2(f)$, let $\check{c}_h = 0$. Let $c_0 = \langle \boldsymbol{v}, \boldsymbol{y} \rangle \bmod q$ and $\bar{c}_0 = \langle \bar{\boldsymbol{v}}, \boldsymbol{y} \rangle \bmod q$. Let $\boldsymbol{v}_2 = \mathsf{vec}((\mathbf{I} + \mathbf{Z}_{\boldsymbol{v}}) \otimes (\mathbf{I} + \mathbf{Z}_{\check{\boldsymbol{v}}}) - \mathbf{I}) \otimes \bar{\boldsymbol{v}}$ and $\boldsymbol{v}_1 = \mathsf{vec}(\mathbf{Z}_{\boldsymbol{v}}) \otimes \bar{\boldsymbol{v}}$. It holds that*

$$\hat{f}(c_1, \ldots, c_m, \check{c}_1, \ldots, \check{c}_m) - \bar{c}_0 = \sum_{(h,h') \in \mathcal{S}_2^{\otimes}(f)} \boldsymbol{v}_2^{\mathrm{T}} \cdot \mathsf{vec}(\boldsymbol{x}_h^{\mathrm{T}} \otimes \boldsymbol{x}_{h'}^{\mathrm{T}} \otimes \mathbf{G}_{h,h'}) + \sum_{h \in \mathcal{S}_1(f)} \boldsymbol{v}_1^{\mathrm{T}} \cdot \mathsf{vec}(\boldsymbol{x}_h^{\mathrm{T}} \otimes \mathbf{F}_h),$$

$$\boldsymbol{v}^{\mathrm{T}} \cdot \bar{\boldsymbol{v}}^{\dagger} \cdot \bar{c}_0 - c_0 = \boldsymbol{v}^{\mathrm{T}} \cdot \mathbf{Z}_{\bar{\boldsymbol{v}}} \cdot \boldsymbol{y}, \text{ and}$$

$$\check{\boldsymbol{v}}^{\mathrm{T}} \cdot \boldsymbol{v}^{\dagger} \cdot c_h - \check{c}_h = \check{\boldsymbol{v}}^{\mathrm{T}} \cdot \mathbf{Z}_{\boldsymbol{v}} \cdot \boldsymbol{y} \text{ for all } h \in \mathcal{S}_2(f).$$

Proof. The proof of the claim relies on the following fact about Kronecker products and vectorization.

Lemma 4.22. *Let $\mathbf{L}, \mathbf{Z}$ be matrices and $\boldsymbol{v}, \boldsymbol{x}$ be vectors of compatible dimensions so that the product $\boldsymbol{v}^{\mathrm{T}} \cdot \mathbf{L} \cdot \mathbf{Z} \cdot \boldsymbol{x}$ is well-defined. It holds that*

$$\boldsymbol{v}^{\mathrm{T}} \cdot \mathbf{L} \cdot \mathbf{Z} \cdot \boldsymbol{x} = (\mathsf{vec}(\mathbf{Z})^{\mathrm{T}} \otimes \boldsymbol{v}^{\mathrm{T}}) \cdot \mathsf{vec}(\boldsymbol{x}^{\mathrm{T}} \otimes \mathbf{L}).$$

Proof. The proof involves repeated applications of the identities $\mathsf{vec}(\mathbf{ABC}) = (\mathbf{C}^{\mathrm{T}} \otimes \mathbf{A}) \cdot \mathsf{vec}(\mathbf{B})$ and $\mathsf{vec}(\boldsymbol{x}) = \boldsymbol{x}$. We observe the following:

$$\begin{aligned}\boldsymbol{v}^{\mathrm{T}} \cdot \mathbf{L} \cdot \mathbf{Z} \cdot \boldsymbol{x} &= \boldsymbol{v}^{\mathrm{T}} \cdot \mathsf{vec}(\mathbf{L} \cdot \mathbf{Z} \cdot \boldsymbol{x}) = \boldsymbol{v}^{\mathrm{T}} \cdot (\boldsymbol{x}^{\mathrm{T}} \otimes \mathbf{L}) \cdot \mathsf{vec}(\mathbf{Z}) \\ &= \mathsf{vec}(\boldsymbol{v}^{\mathrm{T}} \cdot (\boldsymbol{x}^{\mathrm{T}} \otimes \mathbf{L}) \cdot \mathsf{vec}(\mathbf{Z})) = (\mathsf{vec}(\mathbf{Z})^{\mathrm{T}} \otimes \boldsymbol{v}^{\mathrm{T}}) \cdot \mathsf{vec}(\boldsymbol{x}^{\mathrm{T}} \otimes \mathbf{L})\end{aligned}$$

□

We are now ready to prove the claim in the correctness proof. We prove it by directly calculating

$$\begin{aligned}
&\hat{f}(c_1,\dots,c_m,\check{c}_1,\dots,\check{c}_m)\\
&= \bar{\boldsymbol{v}}^{\mathsf{T}} \cdot \left(\sum_{h,h' \in [m]} \mathbf{G}_{h,h'} \cdot (\boldsymbol{v}^{\dagger} \otimes \check{\boldsymbol{v}}^{\dagger}) \cdot c_h \cdot \check{c}_{h'} + \sum_{i \in [m]} \mathbf{F}_h \cdot \boldsymbol{v}^{\dagger} \cdot c_h + \boldsymbol{e} \right)\\
&= \bar{\boldsymbol{v}}^{\mathsf{T}} \cdot \left(\sum_{h,h' \in [m]} \mathbf{G}_{h,h'} \cdot (\boldsymbol{v}^{\dagger} \otimes \check{\boldsymbol{v}}^{\dagger}) \cdot (\boldsymbol{v}^{\mathsf{T}} \otimes \check{\boldsymbol{v}}^{\mathsf{T}}) \cdot (\boldsymbol{x}_h \otimes \boldsymbol{x}_{h'}) + \sum_{i \in [m]} \mathbf{F}_h \cdot \boldsymbol{v}^{\dagger} \cdot \boldsymbol{v}^{\mathsf{T}} \cdot \boldsymbol{x}_h + \boldsymbol{e} \right)\\
&= \bar{\boldsymbol{v}}^{\mathsf{T}} \cdot \left(\sum_{h,h' \in [m]} \mathbf{G}_{h,h'} \cdot ((\boldsymbol{v}^{\dagger} \cdot \boldsymbol{v}^{\mathsf{T}}) \otimes (\check{\boldsymbol{v}}^{\dagger} \cdot \check{\boldsymbol{v}}^{\mathsf{T}})) \cdot (\boldsymbol{x}_h \otimes \boldsymbol{x}_{h'}) + \sum_{i \in [m]} \mathbf{F}_h \cdot \boldsymbol{v}^{\dagger} \cdot \boldsymbol{v}^{\mathsf{T}} \cdot \boldsymbol{x}_h + \boldsymbol{e} \right)\\
&= \bar{\boldsymbol{v}}^{\mathsf{T}} \cdot \left(\sum_{h,h' \in [m]} \mathbf{G}_{h,h'} \cdot ((\mathbf{I} + \mathbf{Z}_{\boldsymbol{v}}) \otimes (\mathbf{I} + \mathbf{Z}_{\check{\boldsymbol{v}}})) \cdot (\boldsymbol{x}_h \otimes \boldsymbol{x}_{h'}) + \sum_{i \in [m]} \mathbf{F}_h \cdot (\mathbf{I} + \mathbf{Z}_{\boldsymbol{v}}) \cdot \boldsymbol{x}_h + \boldsymbol{e} \right)\\
&= \bar{c}_0 + \bar{\boldsymbol{v}}^{\mathsf{T}} \cdot \sum_{h,h' \in [m]} \mathbf{G}_{h,h'} \cdot ((\mathbf{I} + \mathbf{Z}_{\boldsymbol{v}}) \otimes (\mathbf{I} + \mathbf{Z}_{\check{\boldsymbol{v}}}) - \mathbf{I}) \cdot (\boldsymbol{x}_h \otimes \boldsymbol{x}_{h'}) + \bar{\boldsymbol{v}}^{\mathsf{T}} \cdot \sum_{i \in [m]} \mathbf{F}_h \cdot \mathbf{Z}_{\boldsymbol{v}} \cdot \boldsymbol{x}_h\\
&= \bar{c}_0 + \sum_{h,h' \in [m]} (\mathsf{vec}((\mathbf{I} + \mathbf{Z}_{\boldsymbol{v}}) \otimes (\mathbf{I} + \mathbf{Z}_{\check{\boldsymbol{v}}}) - \mathbf{I})^{\mathsf{T}} \otimes \bar{\boldsymbol{v}}^{\mathsf{T}}) \cdot \mathsf{vec}(\boldsymbol{x}_h^{\mathsf{T}} \otimes \boldsymbol{x}_{h'}^{\mathsf{T}} \otimes \mathbf{G}_{h,h'})\\
&\quad + \sum_{i \in [m]} (\mathsf{vec}(\mathbf{Z}_{\boldsymbol{v}})^{\mathsf{T}} \otimes \bar{\boldsymbol{v}}^{\mathsf{T}}) \cdot \mathsf{vec}(\boldsymbol{x}_h^{\mathsf{T}} \otimes \mathbf{F}_h),
\end{aligned}$$

where the last equality follows from Lemma 4.22,

$$\begin{aligned}
\bar{\boldsymbol{v}}^{\mathsf{T}} \cdot \boldsymbol{v}^{\dagger} \cdot \boldsymbol{v}^{\mathsf{T}} \cdot \boldsymbol{y} &= \bar{\boldsymbol{v}}^{\mathsf{T}} \cdot (\mathbf{I} + \mathbf{Z}_{\boldsymbol{v}}) \cdot \boldsymbol{y} = \bar{c}_0 + \bar{\boldsymbol{v}}^{\mathsf{T}} \cdot \mathbf{Z}_{\boldsymbol{v}} \cdot \boldsymbol{y}, \text{ and}\\
\check{\boldsymbol{v}}^{\mathsf{T}} \cdot \boldsymbol{v}^{\dagger} \cdot \boldsymbol{v}^{\mathsf{T}} \cdot \boldsymbol{y} &= \check{\boldsymbol{v}}^{\mathsf{T}} \cdot (\mathbf{I} + \mathbf{Z}_{\boldsymbol{v}}) \cdot \boldsymbol{y} = \check{c}_0 + \check{\boldsymbol{v}}^{\mathsf{T}} \cdot \mathbf{Z}_{\boldsymbol{v}} \cdot \boldsymbol{y}.
\end{aligned}$$

□

To continue the correctness proof, recall that

$$\begin{aligned}
\boldsymbol{u} &= \sum_{(h,h') \in \mathcal{S}_2^{\otimes}(f)} \mathbf{U}_2 \cdot \mathsf{vec}(\boldsymbol{x}_h^{\mathsf{T}} \otimes \boldsymbol{x}_{h'}^{\mathsf{T}} \otimes \mathbf{G}_{h,h'}) + \sum_{h \in \mathcal{S}_1(f)} \mathbf{U}_1 \cdot \mathsf{vec}(\boldsymbol{x}_h^{\mathsf{T}} \otimes \mathbf{F}_h),\\
\bar{\boldsymbol{u}}_0 &= \bar{\mathbf{U}} \cdot \boldsymbol{y}, \text{ and}\\
\check{\boldsymbol{u}}_h &= \check{\mathbf{U}} \cdot \boldsymbol{x}_h \text{ for } h \in \mathcal{S}_2(f)
\end{aligned}$$

are computed using $(\mathbf{U}_2, \mathbf{U}_1, \bar{\mathbf{U}}, \check{\mathbf{U}})$ satisfying

$$\begin{aligned}
\mathbf{A} \cdot \mathbf{U}_h &= \boldsymbol{t} \cdot \boldsymbol{v}_h^{\mathsf{T}} \bmod q,\\
\mathbf{A} \cdot \bar{\mathbf{U}} &= \boldsymbol{t} \cdot \bar{\boldsymbol{v}}^{\mathsf{T}} \cdot \mathbf{Z}_{\boldsymbol{v}} \bmod q, \text{ and}\\
\mathbf{A} \cdot \check{\mathbf{U}} &= \boldsymbol{t} \cdot \bar{\boldsymbol{v}}^{\mathsf{T}} \cdot \mathbf{Z}_{\boldsymbol{v}} \bmod q.
\end{aligned}$$

It follows that

$$\begin{aligned}\mathbf{A}\cdot\boldsymbol{u} &= \boldsymbol{t}\cdot(\hat{f}(c_1,\dots,c_m,\check{c}_1,\dots,\check{c}_m)-\bar{c}_0) \bmod q,\\ \mathbf{A}\cdot\bar{\boldsymbol{u}}_0 &= \boldsymbol{t}\cdot(\boldsymbol{v}^{\mathrm{T}}\cdot\bar{\boldsymbol{v}}^{\dagger}\cdot\bar{c}_0 - c_0) \bmod q, \text{ and}\\ \mathbf{A}\cdot\check{\boldsymbol{u}}_h &= \boldsymbol{t}\cdot(\check{\boldsymbol{v}}^{\mathrm{T}}\cdot\boldsymbol{v}^{\dagger}\cdot c_h - \check{c}_h) \bmod q \text{ for all } h\in\mathcal{S}_2(f).\end{aligned}$$

It remains to analyse the norms of the preimages. The norms of $\boldsymbol{w}_0$, $\bar{\boldsymbol{w}}_0$, $(\check{\boldsymbol{w}}_h)_{h\in\mathcal{S}_2(f)}$ are easy to verify. By the properties discussed in Section 3.3.2, each column in the matrices $\mathbf{U}_2$, $\mathbf{U}_1$, $\bar{\mathbf{U}}$, and $\check{\mathbf{U}}$ has norm as most β. By our choice of parameters, each entry in $\mathbf{G}_{h,h'}$, $\mathbf{F}_h$, $\boldsymbol{x}_1,\dots,\boldsymbol{x}_m$ and $\boldsymbol{y}$ has norm at most α. It follows that

$$\begin{aligned}\|\boldsymbol{u}\| &\le \ell^4\cdot\mathcal{S}_2^{\otimes}(f)\cdot\alpha^3\cdot\beta\cdot\gamma_{\mathcal{R}}^3 + \ell^3\cdot\mathcal{S}_1(f)\cdot\alpha^2\cdot\beta\cdot\gamma_{\mathcal{R}}^2 < \beta^*,\\ \|\bar{\boldsymbol{u}}\|_0 &\le \ell\cdot\alpha\cdot\beta\cdot\gamma_{\mathcal{R}} < \beta^*, \text{ and}\\ \|\check{\boldsymbol{u}}\|_h &\le \ell\cdot\alpha\cdot\beta\cdot\gamma_{\mathcal{R}} < \beta^* \ \forall\, h\in\mathcal{S}_2(f).\end{aligned}$$

Additive homomorphism. As is common in the lattice setting, our construction is almost additively homomorphic in the following sense: Although the commitment function $\boldsymbol{x}\mapsto\langle\boldsymbol{v},\boldsymbol{x}\rangle \bmod q$ is a linear function, the bounded-norm restriction on messages could be violated since $\|\boldsymbol{x}\|\le\alpha$ and $\|\boldsymbol{x}'\|\le\alpha$ in general do not imply $\|\boldsymbol{x}+\boldsymbol{x}'\|\le\alpha$. As such, correctness is only guaranteed after homomorphic evaluation if $\|\boldsymbol{x}+\boldsymbol{x}'\|\le\alpha$.

4.7.4 Succinctness

We measure the succinctness of our construction. A commitment consists of a single $\mathcal{R}_q$ element. A functional opening proof consists of $2\,\mathcal{S}_2(f)+3$ vectors in $\mathcal{R}^{\zeta}$ each of norm at most β^*. Setting $\zeta = \mathsf{lhl}(\mathcal{R},\eta,q,\beta) = \log q\cdot\mathsf{poly}(\lambda)$ for the guarantees of lattice trapdoor algorithms, $\beta^* = 2\cdot\ell^4\cdot m^2\cdot\alpha^3\cdot\beta\cdot\gamma_{\mathcal{R}}^3 = \ell^4\cdot m^2\cdot\mathsf{poly}(\lambda)$ so that correctness holds, and $q=\beta^*\cdot\mathsf{poly}(\lambda)$ to be large enough so that the Twin-k-R-ISIS assumption plausibly holds, a commitment can be described with $\log q\cdot\mathsf{poly}(\lambda) = (\log\ell+\log m)\cdot\mathsf{poly}(\lambda)$ bits, while an opening proof for a function $f\in\mathcal{F}_{\mathsf{quad}}$ can be described with $(2\,|\mathcal{S}_2(f)|+3)\cdot\zeta\cdot\log\beta^*\cdot\mathsf{poly}(\lambda) = |\mathcal{S}_2(f)|\cdot\log^2(m\cdot\ell)\cdot\mathsf{poly}(\lambda)$ bits. Note that for $f\in\mathcal{F}_{\mathsf{level}}$, then $|\mathcal{S}_2(f)| = |\mathcal{S}_2^{\otimes}(f)| = \kappa$. Hence, our CFC is $s(\ell,m,\kappa)$-succinct for the class $\mathcal{F}_{\mathsf{level}} = \{\mathcal{F}_{\mathsf{level},\kappa}\}$, where $s(\ell,m,\kappa)=\kappa\cdot\log^2(m\cdot\ell)$.

Remark 4.23 (Removing the dependence on m)**.** *According to the choice of parameters above, commitments and functional openings have a logarithmic dependence on the number of inputs m (in addition to the input length ℓ). More importantly, for correctness to hold, one should fix q depending on the largest m to be supported. This is a limitation, especially when plugging this CFC in the FC transformation as there m is in the worst case the depth of the circuit. However, since the dependence is only logarithmic we can actually set $\beta^* = 2\cdot\ell^4\cdot\hat{m}^2\cdot\alpha^3\cdot\beta\cdot\gamma_{\mathcal{R}}^3$ where $\hat{m}=\lambda^{\omega(1)}$ is superpolynomial in the security parameter, in such a way that correctness holds for any $m=\mathsf{poly}(\lambda)$. This change makes $q=\lambda^{\omega(1)}$ (a choice that does not affect the plausibility of the assumption according to the analysis of [ACL+22]) and makes the CFC scheme $\kappa\cdot\log^2(\ell)$-succinct.*

4.7.5 Resulting Instantiations of FC for Circuits

As in the previous section, we summarize the FC schemes that result from instantiating our generic construction of Section 4.5 with our lattice-based CFC.

Corollary 4.24. *Assume that all the conditions of Theorem 4.20 are satisfied. Then the following statements hold:*

1. *There exists an FC scheme for the class $\mathcal{F}_n = \{\mathcal{F}_{(d,t,w)}\}$ of arithmetic circuits of width w bounded by $\le \ell$ and in-degree bounded by $\le t_{\max}$ that is $O(d \cdot \log^2(t_{\max} \cdot \ell))$-succinct.*
2. *Using the choice of parameters of Remark 4.23, there exists an FC scheme for $\mathcal{F}_n = \{\mathcal{F}_{(d,t,w)}\}$ of width $w \le \ell$ that is $O(d)$-succinct.*
3. *For any $w_0 \ge 2$, there exists an FC scheme for the class $\mathcal{F} = \{\mathcal{F}_{(d,t,w)}\}$ of circuits of arbitrary width $w > w_0$ that is $O\big(d \cdot (w/w_0)^2\big)$-succinct.*

Case (1) follows by observing that in the FC construction from CFCs the number of CFC inputs is bounded by the in-degree of the admissible circuits. In case (1) we fix a concrete $m = t_{\max}$ in the choice of $q = \beta^* \cdot \mathsf{poly}(\lambda)$ while in points (2)–(3) we consider the parameters choice of Remark 4.23 that let us support any in-degree $t = \mathsf{poly}(\lambda)$.

As opposed to our pairing-based construction, the linear dependency on the depth does not follow from a black-box application of our FC from CFC construction. In fact, Theorem 4.11 gives a proof size of $O\big(d \cdot t \cdot \log^2(t_{\max} \cdot \ell)\big)$. We can supress the t factor by noticing that, for each circuit layer h, the *same* vectors $(\check{\boldsymbol{u}}_h, \check{\boldsymbol{w}}_h)$ are included in the openings at every layer h' such that $h \in \mathcal{S}_2(f^{(h')})$.

The result follows by including them only once in the FC opening proof.

We observe that the resulting lattice-based FC schemes yield shorter proofs (with respect to circuit depth) than their pairing-based counterparts. This feature can be seen as a natural consequence of the additional capability to perform computations over encrypted (in this case, committed) data that lattices provide. Indeed, in our pairing-based construction, the prover needs to provide $O(d \cdot t)$ commitments $\left[X_{h,h'}\right]_1$ to the tensor product of every pair of layers in the circuit. This is avoided in our lattice-based scheme, as the verifier can multiply commitments $C_h \cdot \check{C}_{h'}$ by herself.

4.7.6 Proof of Security

Suppose there exists a PPT adversary $\mathcal{A}$ against evaluation binding of the CFC construction, we construct a PPT algorithm $\mathcal{B}$ for the Twin-k-R-ISIS problem as follows. Given a Twin-k-R-ISIS instance ck, $\mathcal{B}$ passes ck to $\mathcal{A}$. The adversary $\mathcal{A}$ returns input commitments $(c_h)_{h\in[m]}$, a quadratic function f, two output commitments c_0 and c_0', and two functional opening proofs π and π', where $\pi = (\boldsymbol{u}, \boldsymbol{w}_0, \bar{\boldsymbol{u}}_0, \bar{\boldsymbol{w}}_0, (\check{\boldsymbol{u}}_h, \check{\boldsymbol{w}}_h)_{h\in\mathcal{S}(f)})$ and $\pi' = (\boldsymbol{u}', \boldsymbol{w}_0', \bar{\boldsymbol{u}}_0', \bar{\boldsymbol{w}}_0', (\check{\boldsymbol{u}}_h', \check{\boldsymbol{w}}_h')_{h\in\mathcal{S}_2(f)})$. By our assumption on $\mathcal{A}$, with non-negligible probability, π (and analogously π') satisfies

$$\begin{aligned}\mathbf{A}\cdot\boldsymbol{u} &= \boldsymbol{t}\cdot(\hat{f}(c_1,\ldots,c_m,\check{c}_1,\ldots,\check{c}_m)-\bar{c}_0) \bmod q,\\ \mathbf{A}\cdot\bar{\boldsymbol{u}}_0 &= \boldsymbol{t}\cdot(\boldsymbol{v}^{\mathrm{T}}\cdot\bar{\boldsymbol{v}}^{\dagger}\cdot\bar{c}_0-c_0) \bmod q, \text{ and}\\ \mathbf{A}\cdot\check{\boldsymbol{u}}_h &= \boldsymbol{t}\cdot(\check{\boldsymbol{v}}^{\mathrm{T}}\cdot\boldsymbol{v}^{\dagger}\cdot c_h-\check{c}_h) \bmod q \text{ for all } h\in\mathcal{S}_2(f),\end{aligned}$$

where $\mathbf{B}\cdot\bar{\boldsymbol{w}}_0=\boldsymbol{t}\cdot\bar{c}_0 \bmod q$ and $\mathbf{B}\cdot\check{\boldsymbol{w}}_h=\boldsymbol{t}\cdot\check{c}_h \bmod q$.

For any $h\in\mathcal{S}_2(f)$, suppose $\check{\boldsymbol{w}}_h\neq\check{\boldsymbol{w}}'_h$, then from the third equation $(\check{\boldsymbol{u}}_h-\check{\boldsymbol{u}}'_h,\check{\boldsymbol{w}}_h-\check{\boldsymbol{w}}'_h)$ would be a non-zero vector of norm at most $2\beta^*$ satisfying $\mathbf{A}\cdot(\check{\boldsymbol{u}}_h-\check{\boldsymbol{u}}'_h)+\mathbf{B}\cdot(\check{\boldsymbol{w}}_h-\check{\boldsymbol{w}}'_h)=\mathbf{0} \bmod q$, contradicting the twin-k-R-ISIS assumption. We therefore have $\check{\boldsymbol{w}}_h=\check{\boldsymbol{w}}'_h$ and hence $\check{c}_h=\check{c}'_h$ for all $h\in\mathcal{S}_2(f)$.

Next, suppose $\bar{\boldsymbol{w}}_0\neq\bar{\boldsymbol{w}}'_0$, then from the first equation $(\boldsymbol{u}-\boldsymbol{u}',\bar{\boldsymbol{w}}_0-\bar{\boldsymbol{w}}'_0)$ would be a non-zero vector of norm at most $2\beta^*$ satisfying $\mathbf{A}\cdot(\boldsymbol{u}-\boldsymbol{u}')+\mathbf{B}\cdot(\bar{\boldsymbol{w}}_0-\bar{\boldsymbol{w}}'_0)=\mathbf{0} \bmod q$, contradicting the twin-k-R-ISIS assumption. We therefore have $\bar{\boldsymbol{w}}_0=\bar{\boldsymbol{w}}'_0$ and hence $\bar{c}_0=\bar{c}'_0$.

Finally, suppose $\boldsymbol{w}_0\neq\boldsymbol{w}'_0$, then from the second equation $(\bar{\boldsymbol{u}}_0-\bar{\boldsymbol{u}}'_0,\boldsymbol{w}_0-\boldsymbol{w}'_0)$ would be a non-zero vector of norm at most $2\beta^*$ satisfying $\mathbf{A}\cdot(\bar{\boldsymbol{u}}_0-\bar{\boldsymbol{u}}'_0)+\mathbf{B}\cdot(\boldsymbol{w}_0-\boldsymbol{w}'_0)=\mathbf{0} \bmod q$, contradicting the twin-k-R-ISIS assumption. We therefore have $\boldsymbol{w}_0=\boldsymbol{w}'_0$ and hence $c_0=c'_0$, meaning that $\mathcal{A}$ cannot be a successful adversary against evaluation binding.

4.7.7 Efficient Verification

Our CFC construction also supports amortized efficient verification. We observe that in our construction the FuncVer algorithm can be split into an offline preprocessing step and an online verification step:

- $\mathsf{PreFuncVer}(\mathsf{ck},f)$: Compute the polynomials $\hat{f}$, $\bar{\mathsf{id}}$, and $\check{\mathsf{id}}$, output $\mathsf{ck}_f:=(\mathbf{A},\mathbf{B},\boldsymbol{t},\hat{f},\bar{\mathsf{id}},\check{\mathsf{id}})$.
- $\mathsf{EffFuncVer}(\mathsf{vk}_f,(\mathsf{com}_h)_{h\in[m]},\mathsf{com}_0,\pi)$: Perform all the checks described in FuncVer.

Clearly, the runtime of EffFuncVer is $(\mathcal{S}_2^{\otimes}(f)+\mathcal{S}_1(f))\cdot\log q\cdot\mathsf{poly}\leq m^2\cdot\log(m\cdot\ell)\cdot\mathsf{poly}(\lambda)$, which is logarithmic in ℓ.

4.7.8 Commitment Hiding

Commitment hiding can be achieved by extending the dimension of the input vector and dedicating some entries for commitment randomness. We outline such a transformation in the following.

First, we modify the setup so that the vectors $\boldsymbol{v},\bar{\boldsymbol{v}},\check{\boldsymbol{v}}$ are now sampled from $\mathcal{R}_q^{\ell+\zeta}$. The sets $\mathcal{G}_A$ and $\mathcal{G}_B$ of monomials are adjusted accordingly. To commit to $\boldsymbol{x}\in\mathcal{R}^{\ell}$, sample a uniformly random vector $\boldsymbol{r}\leftarrow\$\ \mathcal{R}^{\zeta}$ with $\|\boldsymbol{r}\|\leq\alpha$, and compute $c:=\left\langle\boldsymbol{v},\begin{pmatrix}\boldsymbol{x}\\ \boldsymbol{r}\end{pmatrix}\right\rangle \bmod q$. Opening and verifying are almost identical as in the base scheme, except that f is treated as a polynomial on $(\boldsymbol{x}_1,\boldsymbol{r}_1,\ldots,\boldsymbol{x}_m,\boldsymbol{r}_m)$ but with zero coefficients for all terms involving any entry of $(\boldsymbol{r}_1,\ldots,\boldsymbol{r}_m)$. It can be verified that the modified scheme retains correctness and evaluation binding. For $\zeta\geq\mathsf{lhl}(\mathcal{R},\eta,q,\beta)$, which we anyway need for correctness, commitment hiding is immediate from the leftover hash lemma.

To make the verification more friendly to zero-knowledge arguments, we need to make one more minor change to the scheme: The opening algorithm additionally includes the commitments $(\bar{c}_0, (\check{c}_h)_{h \in \mathcal{S}_2(f)})$ in an opening proof. This makes the verification NIZK-friendly, since it boils down to proving the following SIS relations in zero-knowledge: There exists $(\boldsymbol{u}, \boldsymbol{w}_0, \bar{\boldsymbol{u}}_0, \bar{\boldsymbol{w}}_0, (\check{\boldsymbol{u}}_h, \check{\boldsymbol{w}}_h)_{h \in \mathcal{S}_2(f)}) \in (\mathcal{R}^{\zeta})^{2\mathcal{S}_2(f)+3}$ such that

$$\begin{cases} \mathbf{A} \cdot \boldsymbol{u} = \boldsymbol{t} \cdot (\hat{f}(c_1, \ldots, c_m, \check{c}_1, \ldots, \check{c}_m) - \bar{c}_0) \bmod q & \wedge \ \|\boldsymbol{u}\| \leq \beta^* \\ \mathbf{A} \cdot \bar{\boldsymbol{u}}_0 = \boldsymbol{t} \cdot (\boldsymbol{v}^{\mathrm{T}} \cdot \bar{\boldsymbol{v}}^{\dagger} \cdot \bar{c}_0 - c_0) \bmod q & \wedge \ \|\bar{\boldsymbol{u}}_0\| \leq \beta^* \\ \mathbf{A} \cdot \check{\boldsymbol{u}}_h = \boldsymbol{t} \cdot (\check{\boldsymbol{v}}^{\mathrm{T}} \cdot \boldsymbol{v}^{\dagger} \cdot c_h - \check{c}_h) \bmod q & \wedge \ \|\check{\boldsymbol{u}}_h\| \leq \beta^* \quad \forall\, h \in \mathcal{S}_2(f) \\ \mathbf{B} \cdot \boldsymbol{w}_0 = \boldsymbol{t} \cdot c_0 \bmod q & \wedge \ \|\boldsymbol{w}_0\| \leq \beta^* \\ \mathbf{B} \cdot \bar{\boldsymbol{w}}_0 = \boldsymbol{t} \cdot \bar{c}_0 \bmod q & \wedge \ \|\bar{\boldsymbol{w}}_0\| \leq \beta^* \\ \mathbf{B} \cdot \check{\boldsymbol{w}}_h = \boldsymbol{t} \cdot \check{c}_h \bmod q & \wedge \ \|\check{\boldsymbol{w}}_h\| \leq \beta^* \quad \forall\, h \in \mathcal{S}_2(f). \end{cases}$$

By slightly adjusting the parameters of the k-R-ISIS assumption, the scheme remains evaluation binding even if the NIZK argument can only guarantee that the norm of the witness is bounded by some $\beta^{**} > \beta^*$ (although the prover has a witness of norm bounded by β^*). This allows to use efficient NIZK (e.g. [Lyu09]) for proving SIS relations with relaxed soundness.

4.8 Generic Transformations for Functional Commitments

We introduce two transformations that allow one to boost the properties of functional commitment schemes. These results appeared in [ABF24] and will be used in Chapter 7 to construct succinct multi-key homomorphic signature schemes.

4.8.1 From Succinct to Compact FC

Succinctness is defined with respect to both the input length ℓ and the output length ℓ_y – which we name input-succinctness and output-succinctness. Some FC constructions in the literature, such as [dCP23, WW23b] are nevertheless not output-succinct. To address this generically, we introduce a result that allows one to obtain output-succinctness from any FC. The same transformation applies to (multi-key) homomorphic signatures as those in Chapter 7 (the adaptation is straightforward and omitted here).

Theorem 4.25. *Let* FC *be an evaluation binding FC for ℓ-to-1 functions in the class $\mathcal{F}$. Let $\mathcal{F}'$ be the class of functions where each $f : \mathcal{M}^{\ell} \to \mathcal{M}^{\ell_y}$ in $\mathcal{F}'$ is such that each of its ℓ_y projections is a function in $\mathcal{F}$. Let $H : \mathcal{M}^{\ell_y} \to \mathcal{M}^{\ell}$ with $\ell = \mathsf{poly}(\lambda)$ be a collision resistant hash function. Then, for a suitably expressive $\mathcal{F}$, there exists an evaluation binding* FC′ *for the class $\mathcal{F}'$.*

Proof. The idea is to execute FC on the composed function $f_H := H \odot f : \mathcal{M}^{\ell} \to \mathcal{M}^{\ell}$ in order to generate an opening for each of the ℓ output values of f_H. The verifier who knows the output $\boldsymbol{y} \in \mathcal{M}^{\ell_y}$ runs the FC verification with the function f_H and the output $H(\boldsymbol{y})$. Precisely, since we assume that FC supports only ℓ-to-1 functions, we consider an instantiation for ℓ-to-ℓ functions obtained by running the opening and verification algorithms ℓ times, for the functions $\{f_{H,i}\}_{i=1..\ell_y}$ that return the i-th output bit of f_H. As one can see, the size of the opening proof of this construction is $\ell \cdot |\pi|$ where $|\pi|$ is the size of an opening in FC.

For this transformation to be correct we need the FC scheme to be sufficiently expressive in order to support the functions $H \odot f$, which may be in a class of functions larger than $\mathcal{F}$. For example, for FCs that support circuits of bounded depth one needs to increase the bound by $d_H(\ell_y)$ (i.e., the depth of H on inputs of length ℓ_y). Technically, we need that each projection $f_{H,j}$, for $j = 1$ to ℓ, is in the class $\mathcal{F}$ supported by FC.

The security of this transformation relies on the evaluation binding of FC and the collision resistance of H. A proof sketch follows. Consider any adversary breaking evaluation binding of FC′. Recall that this means that we have two valid openings for $\boldsymbol{y}$ and $\boldsymbol{y}' \neq \boldsymbol{y}$. Then there are two possible cases: $H(\boldsymbol{y}) = H(\boldsymbol{y}')$ or not. In the former case we can break collision resistance of H. In the second case, there is at least an index j such that $H(\boldsymbol{y})_j \neq H(\boldsymbol{y}')_j$ and there are two valid proofs for these values w.r.t. the same function $f_{H,j}$. This case can be reduced to the evaluation binding of FC. □

An interesting special case. Interestingly, the idea of this transformation can be applied even to very limited FCs, such as ones for linear maps, by means of linear hash functions such as Ajtai's. In turn, this method can be applied to existing functional commitments from lattices [dCP23, WW23b, WW23a] to obtain output-succinctness efficiently.

Let FC be an FC for ℓ-to-1 linear forms over a ring $\mathbb{Z}_q$. Precisely, let $\mathcal{F}$ be a set of functions $\mathcal{F} = \{f : \mathbb{Z}_q^\ell \to \mathbb{Z}_q^{\ell_y}\}$ where outputs are small integers bounded (in absolute value) by some $\beta < q$. Consider Ajtai's hash function $H_\mathbf{A} : \mathbb{Z}^{\ell_y} \to \mathbb{Z}_q^\ell$ defined by $H_\mathbf{A}(\boldsymbol{y}) := \mathbf{A} \cdot \boldsymbol{y} \mod q$ for $\mathbf{A} \in \mathbb{Z}_q^{\ell_y \times \ell}$, which is collision-resistant for vectors of small norm. For any $f \in \mathcal{F}$ define $f_H := H_\mathbf{A} \odot f$, i.e., $f_H(\boldsymbol{x}) := \mathbf{A} \cdot f(\boldsymbol{x})$. Notice that $f_H : \mathbb{Z}_q^\ell \to \mathbb{Z}_q^\ell$ is a linear map. Thus, we can run FC for linear forms ℓ times, one for every output.

4.8.2 From FCs to Chainable FCs

Next, we present a generic result that allows one to construct a chainable functional commitment from any (suitably expressive) FC.[16] The idea is simple: for a committed $\boldsymbol{x}$, instead of opening to $\boldsymbol{y} = f(\boldsymbol{x})$ we open to $\mathsf{com}_y = \mathsf{FC.Com}(\mathsf{ck}, f(\boldsymbol{x}))$, which can be expressed as $\mathsf{com}_y = f'(\boldsymbol{x})$ for a function $f' = g \odot f$ (i.e., the sequential composition of f followed by g), where $g(\cdot)$ is the circuit that computes the commitment algorithm $\mathsf{FC.Com}(\mathsf{ck}, \cdot)$.

We remark that, by applying the generic CFC-to-FC transformation in Theorem 4.26 for the special case of layered circuits, this result boosts any FC for bounded-depth circuits into a FC′ for *unbounded-depth* circuits, albeit the proof size of FC′ grows linearly with the circuit depth.

[16]Precisely, we can build a CFC supporting a single input commitment; this is however sufficient for many applications, such as the composable MKHS in Chapter 7.

Theorem 4.26. *Let* FC *be a functional commitment scheme for a class of circuits* $\mathcal{F}$ *and whose commitment algorithm* $\mathsf{FC.Com}$ *can be computed by a circuit* $g \in \mathcal{F}$. *Then there exists a CFC scheme* CFC *for the class of circuits* $\mathcal{F}' = \{f : g \odot f \in \mathcal{F}\}$.

Proof. To obtain a chainable FC scheme CFC from an FC scheme FC, we define CFC as follows:

- $\mathsf{CFC.Setup}(1^\lambda, 1^\ell) = \mathsf{FC.Setup}(1^\lambda, 1^\ell)$
- $\mathsf{CFC.Com}(\mathsf{ck}, \boldsymbol{x}) = \mathsf{FC.Com}(\mathsf{ck}, \boldsymbol{x})$
- $\mathsf{CFC.FuncProve}(\mathsf{ck}, \mathsf{aux}, f) = \mathsf{FC.FuncProve}(\mathsf{ck}, \mathsf{aux}, g \odot f)$
- $\mathsf{CFC.FuncVer}(\mathsf{ck}, \mathsf{com}_x, \mathsf{com}_y, f, \pi) = \mathsf{FC.FuncVer}(\mathsf{ck}, \mathsf{com}_x, \mathsf{com}_y, g \odot f, \pi)$.

Correctness is immediate by construction and by the definition of the class $\mathcal{F}'$.

For evaluation binding, assume by contradiction that an adversary $\mathcal{A}$ outputs a tuple $(\mathsf{com}_x, f, \mathsf{com}_y, \pi, \mathsf{com}'_y, \pi')$ that breaks the evaluation binding of CFC. Then, by construction, the tuple $(\mathsf{com}_x, g \odot f, \mathsf{com}_y, \pi, \mathsf{com}'_y, \pi')$ breaks the evaluation binding of FC.

Let $g \in \mathcal{F}_{\kappa_g}$ and $f \in \mathcal{F}_{\kappa_f}$. Then, the CFC scheme has succinctness parametrized by $s_{\mathsf{CFC}}(\ell, \ell_y, \kappa_f) = s_{\mathsf{FC}}(\ell, \ell_y, \kappa_{g \odot f})$, which by succinctness of FC is $= \mathsf{poly}(\log \ell, \log \ell_y, o(|g \odot f|))$. To argue that this yields succinctness, i.e., $s_{\mathsf{CFC}}(\ell, \ell_y, \kappa_f) = \mathsf{poly}(\log \ell, \log \ell_y, o(|f|))$, we need that $|g| = o(|f|)$. Concretely, for the sake of existing FCs it can be enough to assume that g is a circuit of depth $\mathsf{polylog}(\ell)$. $\square$

5

Pairing-based Functional Commitments with Shorter Parameters

In this chapter, we introduce a pairing-based construction of chainable functional commitments for circuits which has a shorter commitment key than the constructions in Chapter 4. The results are based on the article *"Pairing-based Chainable Functional Commitments for Circuits, Revisited"* [BFL25b], which is in submission at the time of writing.

The chapter is structured as follows. In Section 5.1, we summarize the main contributions, followed by a technical overview in Section 5.2. In Section 5.3, we introduce the pairing-based assumptions that the security of our construction relies on, as well as a building block called type chaining proofs. Finally, in Section 5.4 we introduce our main constructions of chainable functional commitments.

5.1 Contributions

We continue the study of functional commitments for circuits that we initiated in Chapter 4. Our main contribution is an algebraic pairing-based chainable functional commitment for quadratic functions $\mathbb{F}^{m\cdot\ell} \to \mathbb{F}^{\ell}$ with the following characteristics. First, our CFC has constant-sized $O(\lambda)$ commitments and proofs of size $O(\lambda m^2)$, similarly to the pairing-based construction from Chapter 4. In particular, for functions that take as an input a single vector of size ℓ, proofs are also of size $O(\lambda)$. Second, it has public parameters that are of size $O(\lambda \ell^3)$ where ℓ is the input size. Third, it is additively homomorphic and supports efficient verification with pre-processing. The security of the scheme relies on three newly introduced pairing-based assumptions that can be seen as kernel assumptions with hints in the spirit of the HiKer assumption (Theorem 4.15).

Our scheme is seamlessly compatible with all the generic trade-offs and the transformations from Section 4.5. In particular, this yields an algebraic pairing-based (chainable) functional commitment for unbounded-depth d circuits of bounded width w, where the public parameters grow as $O(\lambda w^3)$ and the proof size scales as $O(\lambda d^2)$. For a detailed comparison to the state of the art, we recall Table 4.1. Essentially, the construction has the same asymptotic costs as the pairing-based construction from Chapter 4 does, except that the size of the public parameters is reduced from quintic to cubic on the circuit width. This is a clear improvement on our previous scheme that presents no additional drawbacks. Notably, the public parameter size is the same as in the scheme by Catalano, Fiore and Tucker for the specific case of quadratic polynomials [CFT22] – however, their scheme is not chainable and cannot be compiled into an FC for circuits. We also remark that the compiler from FCs to homomorphic signatures from [CFT22] naturally yields an algebraic homomorphic signature scheme for unbounded-depth circuits with smaller public parameters than the state-of-the-art.

In Section 5.2, we introduce a technical overview of how our CFC construction works, but we anticipate two features of special interest below.

- *The return of the power basis.* Several previous works [LRY16, LM19, CFT22] built functional commitments for restricted classes of functions using the so-called power basis for group elements. In the power basis, also known as monomial basis, the commitment key includes group elements of the form $[\alpha], [\alpha^2], \ldots, [\alpha^\ell]$ for some random $\alpha \leftarrow\$ \mathbb{F}$. Then, commitments are computed as $\mathsf{com} = \sum_{i=1}^{\ell} x_i \cdot [\alpha^i]$. The advantage of the power basis is that one can generally obtain more succinct commitment keys than in the multilinear basis, where the commitment key encodes uncorrelated terms $[\alpha_1], [\alpha_2], \ldots, [\alpha_n]$ such that $\alpha_i \leftarrow\$ \mathbb{F}$. However, whether one could obtain a chainable scheme from a power basis was unclear. In our scheme, we revisit the power basis approach and overcome this challenge by introducing a series of commit-and-prove gadgets for switching between different types of commitments, i.e., in different power basis. We call these gadgets type chaining proofs.
- *Building FCs from type chaining proofs.* Our construction is built in a black-box way following a modular abstraction which relies on (a) several commitment algorithms in different power basis, and (b) linear and quadratic type chaining proofs between them. This abstraction provides a general blueprint for constructing functional commitment schemes. In particular, it is possible to observe the constructions in Chapter 4 from the lens of this abstraction.

A natural question is whether our techniques in the power basis can be extended to the setting of the FC for circuits with fully-succinct proofs from [WW24b], with the goal of reducing the (quintic on the circuit size) public parameters of that scheme. The main challenge seems to be on embedding projective spaces into the power basis. We discuss projective spaces in Chapter 6, but we do not address this issue and leave it as an open question.

5.2 Technical Overview

Our CFC for quadratic functions from pairings is built modularly on top of a family of commitment schemes. We rely on three types of commitments on different basis ($[c], [\hat{c}], [d]$) and three different type chaining proof systems $\Pi_{\mathsf{pow}}, \Pi_{\mathsf{quad}}, \Pi_{\mathsf{eq}}$ that prove different relations between these commitments. We depict them in Figure 5.1. In this technical overview, we describe all of these components of our construction and how they fit together.

5.2.1 A Family of Commitment Schemes in a Power Basis

Our starting point for our CFC construction from pairings is a commitment scheme which, as opposed to previous constructions of chainable functional commitments [BCFL23, WW24b], is based on a power monomial basis instead of a multilinear basis. Given a bilinear group $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ over a base field $\mathbb{F}$, we define our *primary* basis as a set of ℓ elements $\{[\alpha^i]_1\}_{i=1}^{\ell}$. Then, given a vector $\boldsymbol{x} = (x_1, \dots, x_\ell) \in \mathbb{F}^\ell$, we commit to $\boldsymbol{x}$ as $\mathsf{com} = [c]_1 = \sum_{i=1}^{\ell} x_i[\alpha^i]_1$. This corresponds to the left node $[c]_1$ of the diagram in Figure 5.1.

Besides this *primary* commitment scheme, we introduce two additional commitment algorithms that allow us to commit to vectors in different monomial basis. The *power* commitment algorithm allows us to commit to a vector $\boldsymbol{x}$ in a power basis $[\hat{c}]_1 = \sum_{i=1}^{\ell} x_i[\alpha^{\ell(i-1)}]_1$. Note that this basis is a monomial basis also based on powers of α, but it is not the same as the primary basis, as the exponents are "spaced out" by a factor of ℓ. Looking ahead, if we pair a primary commitment to $\boldsymbol{x}$ with a ($\mathbb{G}_2$ version of a) power commitment to $\boldsymbol{x}'$, we obtain a commitment to the tensor product $\boldsymbol{x} \otimes \boldsymbol{x}'$ in the natural α-power basis in the target group, namely $[c]_1 \cdot [\hat{c}]_2 = \left[\sum_{i,j=1}^{\ell} x_i x_j' \alpha^{\ell(j-1)+i}\right]_T$. The *alternate* commitment algorithm allows us to commit to a vector $\boldsymbol{y}$ in a different monomial basis as $[d]_1 = \sum_{i=1}^{\ell} y_i[\beta^i]_1$. We will use alternate commitments to place the outputs of quadratic relations evaluated on the primary commitment.

We remark that all of our commitments are deterministic.

5.2.2 Building a CFC

Chainability implies that our CFC must be able to prove relations between committed values *in the same monomial basis*. This is, given a commitment $\mathsf{com}_x = \sum_{i=1}^{\ell} x_i[\alpha^i]_1$, our goal is to make a proof that com_x opens to $\mathsf{com}_y = \sum_{i=1}^{\ell} y_i[\alpha^i]_1$ under f, where $\boldsymbol{y} = f(\boldsymbol{x})$. We achieve this by using a combination of three proof systems that we call *type chaining proof systems*, as they allow for switching between bases while proving relations between them. Each of these proof systems requires us to include additional cross-terms in the commitment key. Their security property is *evaluation binding* in the same sense of CFC, meaning that it is hard to open the same input commitment com_x to two different output commitments com_y and com_y' under the same function f. These proof systems are as follows:

- The first proof system Π_{pow} is a *basis change proof* that proves *equality* between a primary commitment $[c]_1 = \sum_{i=1}^{\ell} x_i[\alpha^i]_1$ and a power commitment $[\hat{c}]_1 = \sum_{i=1}^{\ell} x_i[\alpha^{\ell(i-1)}]_1$.
- The second proof system Π_{quad} is a *quadratic relation proof* that proves *quadratic relations* between a pair of primary and power commitments $([c]_1, [\hat{c}]_1)$ to $\boldsymbol{x}$, and an alternate commitment $[d]_1 = \sum_{i=1}^{\ell} y_i[\beta^i]_1$.

$$\begin{tikzcd}
\text{primary} & \text{power} & \text{alternate} \\
{[c]_1} \arrow[r, "\Pi_{\mathsf{pow}}"] & ([c]_1, [\hat{c}]_1) \arrow[r, "\Pi_{\mathsf{quad}}"] & {[d]_1} \arrow[dll, bend left, "\Pi_{\mathsf{eq}}"'] \\
{[c_y]_1} & &
\end{tikzcd}$$

Figure 5.1: *Representation of the different proof systems that conform our* CFC.

- The third proof system Π_{eq} is again a *basis change proof* that proves *equality* between an alternate commitment $[d]_1 = \sum_{i=1}^{\ell} y_i[\beta^i]_1$ and a primary commitment $[c]_1 = \sum_{i=1}^{\ell} y_i[\alpha^i]_1$.

We can now combine the three proof systems to build a CFC for quadratic functions, following the steps in Figure 5.1. Given an input $\boldsymbol{x}$ and its (primary) commitment $[c]_1$ and a function f such that $\boldsymbol{y} = f(\boldsymbol{x})$, the prover does as follows:

- Commit to $\boldsymbol{x}$ in the power basis, obtaining $[\hat{c}]_1$.
- Commit to $\boldsymbol{y}$ using the alternate commitment scheme, obtaining $[d]_1$.
- Prove the equality between $[c]_1$ and a power commitment $[\hat{c}]_1$ using Π_{pow}.
- Prove that $([c]_1, [\hat{c}]_1)$ and $[d]_1$ are related by the quadratic function f using Π_{quad}.
- Prove the equality between $[d]_1$ and a primary commitment $[c_y]_1$ to $\boldsymbol{y}$ using Π_{eq}.

To verify the proof, given $[c]_1$ and a commitment $[c_y]_1$ to $\boldsymbol{y}$, the verifier checks all the proofs. Security (evaluation binding) follows from the evaluation binding of each of the proof systems. As all the functional encodings can be easily pre-computed, the CFC admits efficient verification.

In the technical sections, we describe how to extend this approach to evaluate f on multiple inputs (and multiple input commitments). We also remark that our type chaining proof abstraction perfectly captures what occurs in both of our constructions in Chapter 4, providing a cleaner framework for the results from [BCFL23].

5.2.3 Type Chaining Proof Systems

To conclude this overview, we describe the main intuition behind each of the three type chaining proof systems that we introduced above.

Power Basis Type Chaining. To switch between basis while proving equality, the idea is to ask the prover to provide, as a proof, a third commitment to a random linear combination of both basis. This is, define $[t_i]_1 = \gamma[\alpha_i]_1 + \delta[\alpha^{\ell(i-1)}]_1$ for $i \in [\ell]$, where $\gamma, \delta \in \mathbb{F}$ are random scalars. Then, the prover generates a proof $\pi_{\mathsf{pow}} = \sum_{i=1}^{\ell} x_i[t_i]$. The verifier can easily check the relation by checking the following equation, where correctness is straightforward to verify.

$$[\pi_{\mathsf{pow}}]_1 = \gamma[c]_1 + \delta[\hat{c}]_1.$$

However, this approach is only sound if the prover does not know the coefficients γ, δ in advance. To make this idea work in the non-interactive case, we leverage the pairing. We publish the terms $[t_i]_1 = [\gamma\alpha_i + \delta\alpha^{\ell(i-1)}]_1$ in the commitment key, as well as $[\gamma]_2, [\delta]_2$ (note that these are $\mathbb{G}_2$ elements). Then, the verifier checks

$$[\pi_{\mathsf{pow}}]_1 \cdot [1]_2 = [c]_1[\gamma]_2 + [\hat{c}]_1[\delta]_2.$$

The security (evaluation binding) of the proof system relies on the fact that we never give out γ, δ in $\mathbb{G}_1$ in the ck. The only $\mathbb{G}_1$ terms where these coefficients appear are the $[\gamma\alpha_i + \delta\alpha^{\ell(i-1)}]_1$ terms, where they are properly masked by the linear combination.

The security of the proof system relies on a *kernel-with-hints* assumption, in a similar flavour as the HiKer assumption from [BCFL23], that we justify with a proof in the generic group model. The assumption says that it is hard to find two $\mathbb{G}_1$ elements in the kernel of the linear subspace defined by $(\gamma, 1)$, even in the presence of hints (additional terms in the commitment key). More precisely, the adversary wins if it outputs non-zero $([U]_1, [V]_1)$ such that $[U]_1 \cdot [1]_2 = [V]_1 \cdot [\gamma]_2$.

Quadratic Type Chaining. For a quadratic function $f : \mathbb{F}^\ell \to \mathbb{F}^\ell$, we express f as

$$y_k = f_k(\boldsymbol{x}) = \sum_{i,j=1}^{\ell} f_{i,j,k} x_i x_j.$$

Which can also be seen as a linear function on the tensor product $\boldsymbol{x} \otimes \boldsymbol{x}$. As we anticipated, pairing c and $\hat{c}$ results in a commitment to the tensor product in the target group, $[c]_1 \cdot [\hat{c}]_2 = \left[\sum_{i,j=1}^{\ell} x_i x'_j \alpha^{\ell(j-1)+i}\right]_T$. Then, we ask the prover to provide a commitment to the tensor product $[\tilde{c}]_1 = \sum_{i,j=1}^{\ell} x_i x_j [\alpha^{\ell(j-1)+i}]_1$ to $\boldsymbol{x} \otimes \boldsymbol{x}$, whose correctness we can verify (in the target group) with the help of the primary and power commitments.

The next goal is to create an encoding of f and a proof that map y_k to the k-th power of β in $[d]$. For this, we require that the encoding of f satisfies the following relation:

$$\tilde{c} \cdot f = \left(\sum_{i,j=1} x_i x_j \alpha^{\ell(j-1)+i}\right) \cdot f = \sum_{i,j=1} f_{i,j,k} x_i x_j \alpha^{\ell^2+1} \beta^k + T$$

where T are cross terms that will appear in a proof π and which, crucially, *do not contain the α^{ℓ^2+1} term.* To achieve this, we encode the i, j-th coefficient of f in the opposite order to how it appears in the commitment to the tensor product, such that they add up to $\ell^2 + 1$. This leads us to the following encoding of f:

$$[f]_2 \leftarrow \sum_{i,j,k=1}^{\ell} f_{i,j,k} [\beta^k \alpha^{\ell^2+1-i-\ell(j-1)}]_2$$

By encoding all the cross-terms in a proof π_f, the verification equation looks as follows:

$$[\tilde{c}]_1 \cdot [f]_2 = [\pi_f]_1 \cdot [1]_2 + [d]_1 \cdot [\alpha^{\ell^2+1}]_2.$$

For the proof system to be sound, we need to ensure that the α^{ℓ^2+1} term is not included in the proof π_f. We do so by never including the $\mathbb{G}_1$ term $[\alpha^{\ell^2+1}]_1$ in the commitment key. The resulting assumption is a variant of a q-type assumption where all powers of α are given in both groups, except for the $\ell^2 + 1$ power in $\mathbb{G}_1$. We leave the details for the technical section, where we also generalize the approach to multiple inputs and include a degree check on the output commitment.

Equality Basis Type Chaining. Finally, this system is constructed almost identically as Π_{pow}, by asking the prover to provide a commitment to a random linear combination of both basis. We omit the details here as they are similar to the previous proof system.

5.3 Preliminaries

5.3.1 Pairing-based Assumptions

In this section, we introduce three pairing-based assumptions that we require to prove the security of our construction in Section 5.4. In particular, assumption 1 is required by the proof system Π_{pow}, Assumption 2 is required by the proof system Π_{quad}, and Assumption 3 is required by the proof system Π_{eq}. The three assumptions are closely related, as all of them are variants of a q-type assumption where the goal is to find a linear combination of elements in a kernel that is only given in $\mathbb{G}_2$. Besides stating them, we prove that all of them hold in the generic bilinear group model.

Definition 5.1 (Assumption 1)**.** *Let* $\mathsf{bgp} = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, [1]_1, [1]_2)$ *be a bilinear group setting and let* $\ell \in \mathbb{N}$. *We say that Assumption 2 holds for* bgp *on set* $\mathcal{S}_\ell$ *if for any PPT adversary* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that,*

$$\Pr\left[\begin{array}{c|c} [U]_1 \cdot [1]_2 = [V]_1 \cdot [\gamma]_2 & \alpha, \gamma, \delta \leftarrow\$ \mathbb{F} \\ \wedge \quad ([U]_1, [V]_1) \neq ([0]_1, [0]_1) & ([U]_1, [V]_1) \leftarrow \mathcal{A}(\mathsf{bgp}, \mathcal{S}_\ell(\alpha, \gamma, \delta)) \end{array}\right] = \mathsf{negl}(\lambda).$$

Where

$$\mathcal{S}_\ell(\alpha, \gamma, \delta) = \left\{ \{[\alpha^i]_1, [\alpha^i]_2\}_{\substack{i=1 \\ i \neq \ell^2+1}}^{2\ell^2+1}, [\alpha^{\ell^2+1}]_2, \left\{[\gamma\alpha^{\ell(i-1)} + \delta\alpha^i]_1\right\}_{i=1}^{\ell}, [\gamma]_2, [\delta]_2 \right\}$$

and the probability is taken over the choice of α, γ, δ *and the adversary* $\mathcal{A}$*'s random coins.*

Lemma 5.2. *Assumption 1 is sound in the generic bilinear group model.*

Proof. Intuitively, any adversary against the assumption should find a solution of the form $(U, V) = (\gamma u, u)$, this is, in the linear span of $(\gamma, 1)$. As γ only appears in $\mathbb{G}_1$ when it is randomized by δ, which is also never given in the clear in $\mathbb{G}_1$, there is no pair of elements in such space.

Formally, let $\mathcal{A}(\mathcal{S}_\ell(\alpha, \gamma, \delta))$ be an adversary against Assumption 1, which given ck returns a winning tuple $([U]_1, [V]_1)$. Then, the GGM extractor outputs two corresponding polynomials $u(X), v(X) \in \mathbb{F}[X]$ given by

$$u(X,C,D) = u_0 + \sum_{\substack{i=1\\ i\neq \ell^2+1}}^{2\ell^2+1} u_i^{(1)} X^i + \sum_{i=1}^{\ell} u_i^{(2)} (X^{\ell(i-1)}C + X^i D),$$
$$v(X,C,D) = v_0 + \sum_{\substack{i=1\\ i\neq \ell^2+1}}^{2\ell^2+1} v_i^{(1)} X^i + \sum_{i=1}^{\ell} v_i^{(2)} (X^{\ell(i-1)}C + X^i D),$$

such that $u(X,C,D) = v(X,C,D) \cdot C$. Expanding on this identity, we have that

$$u_0 + \sum_{\substack{i=1\\ i\neq \ell^2+1}}^{2\ell^2+1} u_i^{(1)} X^i + \sum_{i=1}^{\ell} u_i^{(2)} (X^{\ell(i-1)}C + X^i D) = v_0 C + \sum_{\substack{i=1\\ i\neq \ell^2+1}}^{2\ell^2+1} v_i^{(1)} X^i C + \sum_{i=1}^{\ell} v_i^{(2)} (X^{\ell(i-1)}C^2 + X^i CD).$$

As there are no monomials of degree 0 in C in $v(X,C,D)$, we immediately derive that $u(X,C,D) = 0$, since all $u_0, u_i^{(1)}, u_i^{(2)}$ appear as coefficients of some monomial of degree 0. Therefore, it also holds that $v(X,C,D) = u(X,C,D) \cdot C = 0$. □

Definition 5.3 (Assumption 2). *Let* $\mathsf{bgp} = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, [1]_1, [1]_2)$ *be a bilinear group setting and let* $\ell \in \mathbb{N}$. *We say that Assumption 1 holds for* bgp *on set* $\mathcal{S}_\ell$ *if for any PPT adversary* $\mathcal{A}$, *there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that,*

$$\Pr\left[\begin{array}{rl} & [U]_1 \cdot [1]_2 = [V]_1 \cdot [\alpha^{\ell^2+1}]_2 \\ \wedge & [V]_1 \cdot [\alpha^{2\ell^2+1}]_2 = [W]_1 \cdot [1]_2 \\ \wedge & [V]_1 \neq [0]_1 \end{array} \middle| \begin{array}{l} \alpha \leftarrow\!\$\, \mathbb{F} \\ ([U]_1, [V]_1, [W]_1) \leftarrow \mathcal{A}(\mathsf{bgp}, \mathcal{S}_\ell(\alpha)) \end{array}\right] = \mathsf{negl}(\lambda).$$

Where $\mathcal{S}_\ell(\alpha) = \left\{ \{[\alpha^i]_1, [\alpha^i]_2\}_{i=1, i\neq \ell^2+1}^{2\ell^2+1}, [\alpha^{\ell^2+1}]_2 \right\}$, *and the probability is taken over the choice of* α *and the adversary* $\mathcal{A}$*'s random coins.*

Lemma 5.4. *Assumption 2 is sound in the generic bilinear group model.*

Proof. The intuition is that, since $[V]_1 \cdot [\alpha^{2\ell^2+1}]_2 = [W]_1 \cdot [1]_2$ and $\alpha^{2\ell^2+1}$ is the highest power of α available, then V cannot contain any powers of α (in other words, it must be a constant term on the α-basis). Then, as $[U]_1 \cdot [1]_2 = [V]_1 \cdot [\alpha^{\ell^2+1}]_2$ and V is constant in α, then U must contain a power α^{ℓ^2+1}, which is never given in $\mathbb{G}_1$.

Formally, let $\mathcal{A}(\mathcal{S}_\ell(\alpha))$ be an adversary against Assumption 1, which given ck returns a winning tuple $([U]_1, [V]_1, [W]_1)$. Then, the GGM extractor outputs three corresponding polynomials $u(X), v(X), w(X) \in \mathbb{F}[X]$ given by

$$u(X) = \sum_{\substack{i=0\\ i\neq \ell^2+1}}^{2\ell^2+1} u_i X^i, \quad v(X) = \sum_{\substack{i=0\\ i\neq \ell^2+1}}^{2\ell^2+1} v_i X^i, \quad w(X) = \sum_{\substack{i=0\\ i\neq \ell^2+1}}^{2\ell^2+1} w_i X^i.$$

Moreover, u, v, w must satisfy the following system of equations:

$$\begin{cases} u(X) = v(X) \cdot X^{\ell^2+1} \\ v(X) \cdot X^{2\ell^2+1} = w(X) \end{cases}$$

Expanding the second equation implies that

$$\sum_{\substack{i=0\\ i\neq \ell^2+1}}^{2\ell^2+1} v_i X^{2\ell^2+1+i} = \sum_{\substack{i=0\\ i\neq \ell^2+1}}^{2\ell^2+1} w_i X^i,$$

which yields $v_i = 0$ for every $i \geq 1$. Hence, $v(X) = v_0$ is a constant polynomial. Then, we can rewrite the first equation as

$$\sum_{\substack{i=0\\ i\neq \ell^2+1}}^{2\ell^2+1} u_i X^i = v_0 \cdot X^{\ell^2+1}.$$

which implies that both $u(X) = v(X) = 0$.

□

Definition 5.5 (Assumption 3). *Let* $\mathsf{bgp} = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, [1]_1, [1]_2)$ *be a bilinear group setting and let* $\ell \in \mathbb{N}$. *We say that Assumption 3 holds for* bgp *on set* $\mathcal{S}_\ell$ *if for any PPT adversary* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that,*

$$\Pr\left[\begin{array}{rl} & [U]_1 \cdot [1]_2 = [V]_1 \cdot [\gamma]_2 \\ \wedge & ([U]_1, [V]_1) \neq ([0]_1, [0]_1) \end{array} \middle| \begin{array}{l} \alpha, \beta, \gamma, \delta \leftarrow\!\!\$\, \mathbb{F} \\ ([U]_1, [V]_1) \leftarrow \mathcal{A}(\mathsf{bgp}, \mathcal{S}_\ell(\alpha, \beta, \gamma, \delta)) \end{array}\right] = \mathsf{negl}(\lambda).$$

Where

$$\mathcal{S}_\ell(\alpha,\beta,\gamma,\delta) = \begin{cases} \{[\alpha^i]_1, [\alpha^i]_2\}_{\substack{i=1\\ i\neq \ell^2+1}}^{2\ell^2+1}, [\alpha^{\ell^2+1}]_2,\ \left\{[\beta^i]_1, [\beta^i]_2\right\}_{i=1}^{\ell}, \left\{[\beta^k\alpha^i]_1, [\beta^k\alpha^i]_2\right\}_{\substack{i,k=1,\\ i\neq \ell^2+1}}^{(2\ell^2+1,\ell)}, \\ \left\{[\gamma\alpha^i + \delta\beta^i]_1\right\}_{i=1}^{\ell}, [\gamma]_2, [\delta]_2 \end{cases}$$

and the probability is taken over the choice of $\alpha, \beta, \gamma, \delta$ *and the adversary* $\mathcal{A}$*'s random coins.*

Lemma 5.6. *Assumption 3 is sound in the generic bilinear group model.*

Proof. The assumption is essentially the same as Assumption 1 (Theorem 5.1) but with the addition of the β-basis elements. Hence, the proof follows the same steps as in Theorem 5.2.

□

5.3.2 Type Chaining Proofs

Recall the definition of chainable functional commitments from Section 4.4. Within a given CFC commitment key ck, there may exist multiple internal commitment keys corresponding to different commitment types. Hence, we allow multiple commitment algorithms $\mathsf{Com}^{(\mathsf{type})}$ to be used on the same ck. Their syntax is given by $\mathsf{Com}^{(\mathsf{type})}(\mathsf{ck}, \boldsymbol{x}) \rightarrow \mathsf{com}^{\mathsf{type}}$.

For increased modularity, we will split our CFC construction in this chapter into smaller proof systems that can be used independently given the same commitment key ck. These systems, that we name *type chaining proofs*, allow one to switch between the different types of commitments allowed in ck. For simplicity, we define these proof systems assuming efficient verification.

Definition 5.7 (Type Chaining Proof). *Let* Setup *be a commitment setup algorithm,* $\mathsf{Com}^{(\mathsf{x})}$ *be an (input) commit algorithm,* $\mathsf{Com}^{(\mathsf{y})}$ *be an (output) commit algorithm, and let* $\mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell)$ *be a commitment key. A type chaining proof system* Π *for the above algorithms and a class of functions* $\mathcal{F}$ *consists of a tuple of PPT algorithms* $(\mathsf{Prove}, \mathsf{PreVer}, \mathsf{Ver})$ *with the following syntax:*

$\Pi.\mathsf{Prove}(\mathsf{ck}, (\boldsymbol{x}_i)_{i\in[m]}, f) \to \pi$: *Given a commitment key* ck*, input vectors* $(\boldsymbol{x}_i)_{i\in[m]}$*, and a function* $f \in \mathcal{F}$*, output a proof* π.

$\Pi.\mathsf{PreVer}(\mathsf{ck}, f) \to \mathsf{ck}_f$: *Given a commitment key* ck *and a function* $f \in \mathcal{F}$*, output a pre-verification key* ck_f.

$\Pi.\mathsf{Ver}(\mathsf{ck}_f, (\mathsf{com}^{(\mathsf{x})}{}_i)_{i\in[m]}, \mathsf{com}^{(\mathsf{y})}, \pi) \to 0/1$: *Given input commitments* $(\mathsf{com}^{(\mathsf{x})}{}_i)_{i\in[m]}$*, output commitments* $\mathsf{com}^{(\mathsf{y})}$ *and a proof* π*, outputs 1 (accept) or 0 (reject).*

Moreover, it must satisfy:
Correctness. *For* $\lambda, \ell, m, \in \mathbb{N}$, $J_1, \ldots, J_m, J_y \in \mathcal{J}$, $f \in \mathcal{F}$ *where* $f : \prod_{i\in[m]} \mathcal{M}^{J_i} \to \mathcal{M}^{J_y}$*, and* $\boldsymbol{x}_i \in \mathcal{M}^{J_i}$ *for* $i \in [m]$*, it holds that*

$$\Pr\left[\begin{array}{r|l} & \mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell) \\ & \mathsf{com}^{(\mathsf{x})}{}_i \leftarrow \mathsf{Com}^{(\mathsf{x})}(\mathsf{ck}, J_i, \boldsymbol{x}_i) \ \forall i \in [m] \\ \mathsf{Ver}(\mathsf{ck}_f, (\mathsf{com}^{(\mathsf{x})}{}_i)_{i\in[m]}, & \mathsf{com}^{(\mathsf{y})} \leftarrow \mathsf{Com}^{(\mathsf{y})}(\mathsf{ck}, J_y, f(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m)) \\ \mathsf{com}^{(\mathsf{y})}, f, \pi) = 1 & \mathsf{ck}_f \leftarrow \mathsf{PreVer}(\mathsf{ck}, f) \\ & \pi \leftarrow \mathsf{FuncProve}(\mathsf{ck}, (\boldsymbol{x}_i)_{i\in[m]}, f) \end{array}\right] = 1.$$

The notion of security (evaluation binding) of a type chaining proof system is evaluation binding, identically to Theorem 4.9. Note that evaluation binding allows for adversarially chosen commitments, and so the notion is not parametrized by the different commitment algorithms that are used, as opposed to what occurs for correctness.

5.4 Pairing-based CFC for Quadratic Functions

In this section we describe our construction of a pairing-based chainable functional commitment scheme for quadratic functions. Our goal is to prove the following theorem.

Theorem 5.8 (Pairing-based CFC). *Let* $\mathcal{F}_{\mathsf{quad}} = \{f : \mathbb{F}^{m\cdot\ell} \to \mathbb{F}^\ell : f \text{ is quadratic}\}$ *be the family of quadratic functions with* m *input vectors of length* ℓ *and a single output vector. Let* $\mathsf{bgp} = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, [1]_1, [1]_2)$ *be a bilinear group setting. If Assumptions 1, 2 and 3 (Definitions 5.1, 5.3, 5.5) hold over* bgp*, the construction* CFC *in Figure 5.6 is a chainable functional commitment (Theorem 4.8) for* $\mathcal{F}_{\mathsf{quad}}$ *with the following properties:*

- *The commitment key is of size* $O(\lambda\ell^3)$ *group elements.*
- *The commitment contains one* $\mathbb{G}_1$ *group element.*
- *The proof size is* $|\pi| = O(\lambda m^2)$.
- *Efficient verification runs in time dominated by* $O(m^2)$ *pairing computations between* $\mathbb{G}_1$ *and* $\mathbb{G}_2$.

Moreover, CFC *is additively homomorphic and enjoys efficient verification with preprocessing, where the preprocessing key has size* $\left|\mathsf{ck}_f\right| = O(\lambda)$ *and the verifier needs to do* $O\left(m^2\right)$ *group operations.*

Proof. Correctness follows by the correctness of each of the proof systems Π_{pow}, Π_{quad} and Π_{eq}, in Lemmas 5.10, 5.13 and 5.16 respectively. Evaluation binding follows from Theorem 5.18. The remaining properties follow by the construction of the commitment key in Section 5.4.1 and the proof systems in Section 5.4.2, Section 5.4.3 and Section 5.4.4. □

The running times of the prover and the verifier are dominated by the quadratic type chaining proof, which involves the most intensive computations. We analyse these in Theorem 5.14.

Remark 5.9. *The generic transformation from a CFC for quadratic functions in [BCFL23] to a fully-fledged CFC for circuits can be applied to our construction. This yields a CFC for circuits with a setup size of* $|\mathsf{ck}| = O\left(\lambda w^3\right)$ *and a proof size of* $O\left(\lambda d^2\right)$ *for arithmetic circuits of depth d and width w.*

Outline. In the remainder of the section, we introduce our CFC for quadratic functions and prove the lemmas required in Theorem 5.8. In Section 5.4.1, we describe the setup algorithm, the base commitment scheme which is used to commit to the input vectors $\boldsymbol{x}$, and two auxiliary commitment schemes required internally by the scheme. Then, in Sections 5.4.2, 5.4.3, 5.4.4, we describe the three type chaining proof systems Π_{pow}, Π_{quad} and Π_{eq} that conform the main scheme. Finally, in Section 5.4.5, we describe the main construction of the CFC, which is a combination of the base commitment scheme and the type chaining proof systems.

5.4.1 Base Commitment Scheme

We start by describing the base algorithms of the commitment scheme, Setup and Com. These are used across the section to commit to and prove relations on the input vectors $\boldsymbol{x}$. The commitment key contains $2\ell^3\mathbb{G}_1 + 2\ell^3\mathbb{G}_2 + O(\ell^2)$ elements. The assumptions required to prove the evaluation binding of the type chaining proof systems that are associated to this commitment key are based on the hardness of finding (a multiple of) γ_e, γ_p or α^{ℓ^2+1} in the group $\mathbb{G}_1$, as introduced previously.

$\underline{\mathsf{CFC.Setup}(1^\lambda, 1^\ell)}$**:** Let ℓ be the maximum input size and λ the security parameter.

Generate a bilinear group description $\mathsf{bgp} := (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, [1]_1, [1]_2) \leftarrow \mathcal{BG}(1^\lambda)$, and let $\mathbb{F} := \mathbb{Z}_q$.

Sample $\alpha, \beta, \delta_p, \gamma_p, \delta_e, \gamma_e \leftarrow\$\ \mathbb{F}$ and encode the key as follows:

$$\mathsf{ck} = \left\{ \begin{aligned} &\{[\alpha^i]_1, [\alpha^i]_2\}_{\substack{i=1\\ i\neq \ell^2+1}}^{2\ell^2+1}, [\alpha^{\ell^2+1}]_2,\ \left\{[\beta^i]_1, [\beta^i]_2\right\}_{i=1}^{\ell}, \left\{[\beta^k\alpha^i]_1, [\beta^k\alpha^i]_2\right\}_{\substack{i,k=1,\\ i\neq \ell^2+1}}^{(2\ell^2+1,\ell)}, \\ &\left\{[\gamma_p\alpha^{\ell(i-1)} + \delta_p\alpha^i]_1, [\gamma_e\alpha^i + \delta_e\beta^i]_1\right\}_{i=1}^{\ell}, [\gamma_p]_2, [\delta_p]_2, [\gamma_e]_2, [\delta_e]_2 \end{aligned} \right\}$$

<u>$\mathsf{CFC.Com}(\mathsf{ck}, \boldsymbol{x})$:</u> Let $\boldsymbol{x} = (x_1, \ldots, x_\ell) \in \mathbb{F}^\ell$ be the input vector. Output $\mathsf{com} \leftarrow [c]_1 = \sum_{i=1}^{\ell} x_i [\alpha^i]_1$.

Beyond the primary commitment algorithm, we define two additional commitment algorithms, $\mathsf{Com}^{(\mathsf{pow})}$ and $\mathsf{Com}^{(\mathsf{eq})}$, that are used to commit to the auxiliary vectors $[\hat{c}]_1$ and $[d]_1$, respectively. These are defined as follows:

<u>$\mathsf{CFC.Com}^{(\mathsf{pow})}(\mathsf{ck}, \boldsymbol{x})$:</u> Output $[\hat{c}]_1 = \sum_{i=1}^{\ell} x_i [\alpha^{\ell(i-1)}]_1$.

<u>$\mathsf{CFC.Com}^{(\mathsf{eq})}(\mathsf{ck}, \boldsymbol{x})$:</u> Output $[d]_1 = \sum_{i=1}^{\ell} x_i [\beta^i]_1$.

5.4.2 Power Basis Type Chaining

Our first type chaining proof system is Π_{pow}, which proves the equality between two commitments to the same vector $\boldsymbol{x}$ in different bases. Π_{pow} is defined with respect to the standard commitment algorithm Com and the auxiliary commitment algorithm $\mathsf{Com}^{(\mathsf{pow})}$. We introduce the construction in Figure 5.2.

Internally, the proof system attests equality between vectors committed in $\mathsf{com}^{(\mathsf{x})} = [c]_1 = \sum_{i=1}^{\ell} x_i [\alpha^i]_1$ and $\mathsf{com}^{(\mathsf{y})} = [\hat{c}]_1 = \sum_{i=1}^{\ell} x_i [\alpha^{\ell(i-1)}]_1$. In the scheme, note that the smallest power of α that is required by the prover (regarding γ-terms) is $[\alpha^1 \gamma]_1$, as if $i = 1$ it must be that $j \geq 2$. Therefore, we never need to give out $[\gamma]_1$, which would break the scheme.

We also remark that the running time of the prover is $O(\lambda \ell^2)$, as it requires ℓ^2 multiplications powers of α to produce a proof.

<u>$\Pi_{\mathsf{pow}}.\mathsf{Prove}(\mathsf{ck}, \boldsymbol{x})$:</u>

Given a vector $\boldsymbol{x}$, compute $[\pi_{\mathsf{pow}}]_1 \leftarrow \sum_{i=1}^{\ell} x_i [\gamma_p \alpha^{\ell(i-1)} + \delta_p \alpha^i]_1$.

<u>$\Pi_{\mathsf{pow}}.\mathsf{Ver}(\mathsf{ck}, \mathsf{com}^{(\mathsf{x})}, \mathsf{com}^{(\mathsf{y})}, \pi)$:</u>

- Parse $\sigma_x = [c]_1$ and $\sigma_y = [\hat{c}]_1$.
- Check that $[\pi_{\mathsf{pow}}]_1 \cdot [1]_2 = [\hat{c}]_1 \cdot [\gamma_p]_2 + [c]_1 \cdot [\delta_p]_2$.

Figure 5.2: *Power type chaining proof* Π_{pow}.

Lemma 5.10. Π_{pow} *satisfies correctness.*

Proof. For honestly generated proofs and commitments, the verification equation (in the exponent) is given by:

$$\pi_{\mathsf{pow}} \cdot 1 = \sum_{i=1}^{\ell} x_i \gamma_p \alpha^{\ell(i-1)} + x_i \delta_p \alpha^i = \hat{c} \cdot \gamma_p + c \cdot \delta_p.$$

□

Lemma 5.11. *If Assumption 1 (Theorem 5.1) holds over* bgp, Π_{pow} *satisfies evaluation binding.*

Proof. Let $\mathcal{A}$ be an adversary against evaluation binding of Π_{pow}. We construct an adversary $\mathcal{B}$ against the assumption as follows. $\mathcal{B}(\mathsf{bgp}, \mathcal{S}(\alpha, \gamma_p, \delta_p))$ samples $\beta, \gamma_e, \delta_e \leftarrow\$ \mathbb{F}$ uniformly at random and generates a ck as follows:

$$\mathsf{ck} = \begin{cases} \{[\alpha^i]_1, [\alpha^i]_2\}_{\substack{i=1 \\ i \neq \ell^2+1}}^{2\ell^2+1}, [\alpha^{\ell^2+1}]_2, \ \{\beta^i[1]_1, \beta^i[1]_2\}_{i=1}^{\ell}, \{\beta^k[\alpha^i]_1, \beta^k[\alpha^i]_2\}_{\substack{i,k=1, \\ i \neq \ell^2+1}}^{(2\ell^2+1,\ell)}, \\ \{[\gamma_p \alpha^{\ell(i-1)} + \delta_p \alpha^i]_1, \left(\gamma_e[\alpha^i]_1 + \delta_e \beta^i[1]_1\right)\}_{i=1}^{\ell}, [\gamma_p]_2, [\delta_p]_2, \gamma_e[1]_2, \delta_e[1]_2 \Big\} \end{cases}$$

It is straightforward to see that ck is distributed identically as the original one.

Then, it calls $\mathcal{A}(\mathsf{ck})$ and parses the proofs and outputs as $([c]_1, [\hat{c}]_1, [\hat{c}']_1, [\pi_{\mathsf{pow}}]_1, [\pi'_{\mathsf{pow}}]_1)$, respectively. Note that if $\mathcal{A}$ is successful, it must be that $[\hat{c}]_1 \neq [\hat{c}']_1$. Then, $\mathcal{B}$ simply outputs $[U]_1 = [\pi_{\mathsf{pow}}]_1 - [\pi'_{\mathsf{pow}}]_1, [V]_1 = [\hat{c}]_1 - [\hat{c}']_1$

The claim follows by subtracting the verification equation satisfied by $[\hat{c}]_1$ and $[\hat{c}']_1$ for the same input $[c]_1$. Namely, as

$$\begin{aligned} [\pi_{\mathsf{pow}}]_1 \cdot [1]_2 =& [\hat{c}]_1 \cdot [\gamma_p]_2 + [c]_1 \cdot [\delta_p]_2, \\ [\pi'_{\mathsf{pow}}]_1 \cdot [1]_2 =& [\hat{c}']_1 \cdot [\gamma_p]_2 + [c]_1 \cdot [\delta_p]_2. \end{aligned}$$

We subtract both equations and obtain

$$[U]_1 \cdot [1]_2 = ([\pi_{\mathsf{pow}}]_1 - [\pi'_{\mathsf{pow}}]_1) \cdot [1]_2 = ([\hat{c}]_1 - [\hat{c}']_1) \cdot [\gamma_p]_2 = [V]_1 \cdot [\gamma_p]_2.$$

□

5.4.3 Quadratic Map Type Chaining

Our second type chaining proof system is Π_{quad}, which proves the evaluation of a quadratic function f on multiple committed input vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m$ with respect to a committed output $\boldsymbol{y} = f(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m)$. Before presenting the construction, we introduce some notation on quadratic functions.

Notation on Quadratic Functions. We consider the family of quadratic functions $\mathcal{F}_{\mathsf{quad}} = \{f : \mathbb{F}^{\ell \cdot m} \to \mathbb{F}^{\ell} : f \text{ is quadratic}\}$, where m is the number of input vectors and ℓ is the input and output size. As we show in the following lemma, we note that it is sufficient to consider homogeneous quadratic polynomials, as any quadratic polynomial $f(\boldsymbol{x})$ can be expressed as a homogeneous quadratic polynomial evaluated on $(1, \boldsymbol{x})$.

Lemma 5.12. *For any ℓ-variate quadratic polynomial $f \in \mathbb{F}[X_1, \ldots, X_\ell]^{\leq 2}$, there exists a homogeneous $\ell + 1$-variate quadratic polynomial $\bar{f} \in \mathbb{F}[X_0, X_1, \ldots, X_\ell]^2$ such that $f(\boldsymbol{x}) = \bar{f}(1, \boldsymbol{x})$ for every $\boldsymbol{x} \in \mathbb{F}^\ell$.*

Proof. Consider the ℓ-variate quadratic polynomial $f(\boldsymbol{x}) = \sum_{i=1}^{\ell} \sum_{j=1}^{\ell} f_{i,j} x_i x_j + \sum_{i=1}^{\ell} g_i x_i + h$. We define a ℓ+1-variate homogeneous quadratic polynomial $\bar{f}$ as $\bar{f}(x_0, \boldsymbol{x}) = \sum_{i=0}^{\ell} \sum_{j=0}^{\ell} \bar{f}_{i,j} x_i x_j$

such that:

$$\bar{f}_{i,j} = \begin{cases} \bar{f}_{i,j} & \text{if} \quad 1 \leq i,j \leq \ell \\ g_i & \text{if} \quad j = 0, 1 \leq i \leq \ell \\ 0 & \text{if} \quad i = 0, 1 \leq j \leq \ell \\ h & \text{if} \quad i = j = 0 \end{cases}$$

The equality of $f(\boldsymbol{x})$ and $\bar{f}(1,\boldsymbol{x})$ as polynomial functions follows by setting $x_0 = 1$, as

$$\bar{f}(1,\boldsymbol{x}) = \sum_{i,j=1}^{\ell} f_{i,j}x_ix_j + \sum_{i=1}^{\ell} \bar{f}_{i,0}x_i + \sum_{j=1}^{\ell} \bar{f}_{0,j}x_j + \bar{f}_{0,0} = \sum_{i,j=1}^{\ell} f_{i,j}x_ix_j + \sum_{i=1}^{\ell} g_ix_i + h = f(\boldsymbol{x}).$$

□

For a single-input homogeneous quadratic function $f : \mathbb{F}^{\ell} \longrightarrow \mathbb{F}^{\ell}$, we denote its coefficients by $f_{i,j,k} \in \mathbb{F}$ for $i,j,k \in [\ell]$. The k-th coordinate of $f(\boldsymbol{x})$ is given by $y_k = \sum_{i,j=1}^{\ell} f_{i,j,k}x_ix_j$. For its multi-input analogue $f : \mathbb{F}^{\ell \cdot m} \longrightarrow \mathbb{F}^{\ell}$, we denote its coefficients by $f_{i,j,k}^{(h,h')}$ where $h,h' \in [m]$. The k-th output coordinate is given by

$$y_k = f_k(\boldsymbol{x}_1,\ldots,\boldsymbol{x}_m) = \sum_{h,h'}^{m} \sum_{i,j=1}^{\ell} f_{i,j}^{(h,h')} x_i^{(h)} x_j^{(h')}.$$

Construction. We describe the type chaining proof in Figure 5.4, which entails the core of our CFC construction. In Figure 5.3 we introduce the simpler single-input case explicitly, as it captures the main properties and intuition of the proof system, albeit with a notably simpler notation.

Lemma 5.13. *The construction* Π_{quad} *satisfies correctness.*

Proof. The first three verification equations are straightforward to verify. For the quadratic function test, the LHS (in the single-input scheme in Figure 5.3) is given by:

$$\begin{aligned} \tilde{c} \cdot \mathsf{ck}_f &= \left(\sum_{i',j'=1}^{\ell} x_{i'}x_{j'}\alpha^{i'+\ell(j'-1)} \right) \left(\sum_{i,j,k=1}^{\ell} f_{i,j,k}\beta^k\alpha^{\ell^2+1-i-\ell(j-1)} \right) \\ &= \sum_{\substack{i,j,k,i',j'=1 \\ (i,j)\neq(i',j')}}^{\ell} f_{i,j,k}x_{i'}x_{j'}\beta^k\alpha^{\ell^2+1-(i'-i)-\ell(j'-j)} + \sum_{i,j,k=1}^{\ell} f_{i,j,k}x_ix_j\beta^k\alpha^{\ell^2+1} \\ &= \pi_f \cdot 1 + \sum_{k=1}^{\ell} y_k\beta^k\alpha^{\ell^2+1} \\ &= \pi_f \cdot 1 + d \cdot \alpha^{\ell^2+1}. \end{aligned}$$

Where in the second step, we separate the terms corresponding to α^{ℓ^2+1}. Note that the product only contains α^{ℓ^2+1} terms whenever $(i,j) = (i',j')$.

$\Pi_{\mathsf{quad}}.\mathsf{Prove}(\mathsf{ck}, \boldsymbol{x}, f)$:

Let $f_{i,j,k}$ be the coefficients of a quadratic form f where $y_k = \sum_{i,j} f_{i,j,k} x_i x_j$. Compute a quadratic proof as:

$$[\pi_f]_1 \leftarrow \sum_{\substack{i,j,k,i',j'=1\\(i,j)\neq(i',j')}}^{\ell} f_{i,j,k} x_{i'} x_{j'} [\beta^k \alpha^{\ell^2+1-(i'-i)-\ell(j'-j)}]_2$$

- Compute an auxiliary $[\hat{c}]_2 \leftarrow \sum_{i=1}^{\ell} x_i [\alpha^{\ell(i-1)}]_1$.
- Compute the tensor encoding $[\tilde{c}]_1 \leftarrow \sum_{i,j=1}^{\ell} x_i x_j [\alpha^{\ell(j-1)+i}]_1$
- Compute an auxiliary eq commitment $[\check{d}]_1 \leftarrow \sum_{i=1}^{\ell} y_i [\beta^i \alpha^{2\ell^2+1}]_1$
- Output $\pi_{\mathsf{quad}} = ([\hat{c}]_2, [\tilde{c}]_1, [\pi_f]_1, [\check{d}]_1)$

$\Pi_{\mathsf{quad}}.\mathsf{PreVer}(\mathsf{ck}, f)$:

Compute and output

$$[\mathsf{ck}_f]_2 \leftarrow \sum_{i,j,k=1}^{\ell} f_{i,j,k} [\beta^k \alpha^{\ell^2+1-i-\ell(j-1)}]_2$$

$\Pi_{\mathsf{quad}}.\mathsf{Ver}(\mathsf{ck}, \mathsf{com}^{(\mathsf{x})}, \mathsf{com}^{(\mathsf{y})}, \pi_{\mathsf{quad}})$:

- Parse $\mathsf{com}^{(\mathsf{x})} = ([c]_1, [\hat{c}]_1)$ and $\mathsf{com}^{(\mathsf{y})} = [d]_1$.
- Parse $\pi_{\mathsf{quad}} = ([\hat{c}]_2, [\tilde{c}]_1, [\pi_f]_1, [\check{d}]_1)$.
- Check $[1]_1 \cdot [\hat{c}]_2 = [\hat{c}]_1 \cdot [1]_2$ ($\mathbb{G}_2$ commitment equality)
- Check $[\tilde{c}]_1 \cdot [1]_1 = [c]_1 \cdot [\hat{c}]_2$ (tensor product)
- Check $[d]_1 \cdot [\alpha^{2\ell^2+1}]_2 = [\check{d}]_1 \cdot [1]_2$ (degree test for $[d]_1$)
- Check $[\tilde{c}]_1 \cdot [\mathsf{ck}_f]_2 = [\pi_f]_1 \cdot [1]_2 + [d]_1 \cdot [\alpha^{\ell^2+1}]_2$ (quadratic function)

Figure 5.3: *Quadratic type chaining proof* Π_{quad} *(single-input).*

For the multi-input scheme in Figure 5.4, the argument is identical, except that the notation is more cumbersome. The LHS is given by:

$\Pi_{\mathsf{quad}}.\mathsf{Prove}(\mathsf{ck}, (\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(h)}), f)$:

Let $f_{i,j,k}^{(h,h')}$ be the coefficients of a homogeneous quadratic polynomial f where $y_k = \sum_{i,j} f_{i,j,k}^{(h,h')} x_i^{(h)} x_j^{(h')}$. Compute a quadratic proof as:

$$[\pi_f]_1 \leftarrow \sum_{h,h'=1}^{m} \sum_{\substack{i,j,k,i',j'=1 \\ (i,j)\neq(i',j')}}^{\ell} f_{i,j,k}^{(h,h')} x_{i'}^{(h)} x_{j'}^{(h')} [\beta^k \alpha^{\ell^2+1-(i'-i)-\ell(j'-j)}]_2$$

- Compute auxiliary $[\hat{c}_h]_2 \leftarrow \sum_{i=1}^{\ell} x_i^{(h)} [\alpha^{\ell(i-1)}]_1$ for every $h \in [m]$.
- Compute the tensor encodings $[\tilde{c}_{h,h'}]_1 \leftarrow \sum_{i,j=1}^{\ell} x_i^{(h)} x_j^{(h')} [\alpha^{\ell(j-1)+i}]_1$
- Compute an auxiliary eq commitment $[\check{d}]_1 \leftarrow \sum_{i=1}^{\ell} y_i [\beta^i \alpha^{2\ell^2+1}]_1$
- Output $\pi_{\mathsf{quad}} = (([\hat{c}_h]_2)_{h\in[m]}, ([\tilde{c}_{h,h'}]_1)_{h,h'\in[m]}, [\pi_f]_1, [\check{d}]_1)$

$\Pi_{\mathsf{quad}}.\mathsf{PreVer}(\mathsf{ck}, f)$:

Compute and output

$$[\mathsf{ck}_{f,(h,h')}]_2 \leftarrow \sum_{i,j,k=1}^{\ell} f_{i,j,k}^{(h,h')} [\beta^k \alpha^{\ell^2+1-i-\ell(j-1)}]_2$$

for every $h, h' \in [m]$.

$\Pi_{\mathsf{quad}}.\mathsf{Ver}(\mathsf{ck}_f, (\mathsf{com}^{(\mathsf{x})}{}_h)_{h\in[m]}, \mathsf{com}^{(\mathsf{y})}, \pi_{\mathsf{quad}})$:

- Parse $\mathsf{ck}_f = ([\mathsf{ck}_{f,(h,h')}]_2)_{h,h'\in[m]}$.
- Parse $\mathsf{com}^{(\mathsf{x})}{}_h = ([c_h]_1, [\hat{c}_h]_1)$ for $h \in [m]$ and $\mathsf{com}^{(\mathsf{y})} = [d]_1$.
- Parse $\pi_{\mathsf{quad}} = ([\hat{c}_h]_2, [\tilde{c}_{h,h'}]_1, [\pi_f]_1, [\check{d}]_1)$.
- Check $[1]_1 \cdot [\hat{c}_h]_2 = [\hat{c}_h]_1 \cdot [1]_2$ for every $h \in [m]$ ($\mathbb{G}_2$ commitment equality)
- Check $[\tilde{c}_{h,h'}]_1 \cdot [1]_1 = [c_h]_1 \cdot [\hat{c}_h]_2$ for every $h \in [m]$ (tensor product)
- Check $[d]_1 \cdot [\alpha^{2\ell^2+1}]_2 = [\check{d}]_1 \cdot [1]_2$ (degree test for $[d]_1$)
- Check $\sum_{h,h'=1}^{m} \left([\tilde{c}_{h,h'}]_1 \cdot [\mathsf{ck}_{f,(h,h')}]_2\right) = [\pi_f]_1 \cdot [1]_2 + [d]_1 \cdot [\alpha^{\ell^2+1}]_2$ (quadratic function)

Figure 5.4: *Quadratic type chaining proof* Π_{quad} *(multi-input).*

$$
\begin{aligned}
\sum_{h,h'}^{m}\left(\tilde{c}_{h,h'}\cdot \mathsf{ck}_{f,(h,h')}\right) &= \sum_{h,h'}^{m}\left(\sum_{i',j'=1}^{\ell} x_{i'}^{(h)}x_{j'}^{(h')}\alpha^{i'+\ell(j'-1)}\right)\left(\sum_{i,j,k=1}^{\ell} f_{i,j,k}^{(h,h')}\beta^{k}\alpha^{\ell^2+1-i-\ell(j-1)}\right)\\
&= \sum_{h,h'}^{m}\sum_{\substack{i,j,k,i',j'=1\\(i,j)\neq(i',j')}}^{\ell} f_{i,j,k}^{(h,h')}x_{i'}^{(h)}x_{j'}^{(h')}\beta^{k}\alpha^{\ell^2+1-(i'-i)-\ell(j'-j)}\\
&+\sum_{h,h'}^{m}\sum_{i,j,k=1}^{\ell} f_{i,j,k}^{(h,h')}x_{i}^{(h)}x_{j}^{(h')}\beta^{k}\alpha^{\ell^2+1}\\
&= \pi_f\cdot 1+\sum_{k=1}^{\ell} y_k\beta^{k}\alpha^{\ell^2+1}\\
&= \pi_f\cdot 1+d\cdot\alpha^{\ell^2+1}.
\end{aligned}
$$

□

Before proving the security of the scheme, we include a lemma about prover and verifier efficiency. We note that for families of functions with sparse coefficients, the running times of both $\Pi_{\mathsf{quad}}.\mathsf{Prove}$ and $\Pi_{\mathsf{quad}}.\mathsf{PreVer}$ are faster. For instance, for functions where $f_{i,j,k}^{(h,h')}=0$ except if $i=j=k$, algorithm $\Pi_{\mathsf{quad}}.\mathsf{PreVer}$ requires only $O(m^2\ell)$ group operations.[1]

Lemma 5.14. *In the construction Π_{quad} (Figure 5.4), the worst-case running times of the algorithms are:*

- $\Pi_{\mathsf{quad}}.\mathsf{Prove}$ *is dominated by the time required to carry out $O(m^2\ell^3\log\ell^2)$ field operations and $O(m^2\ell^3)$ group operations.*
- $\Pi_{\mathsf{quad}}.\mathsf{PreVer}$ *is dominated by the time required to carry out $O(m^2\ell^3)$ group operations.*
- $\Pi_{\mathsf{quad}}.\mathsf{Ver}$ *is dominated by the time required to carry out $O(m^2)$ pairing operations.*

Proof. For $\Pi_{\mathsf{quad}}.\mathsf{PreVer}$, the result follows by inspection as the algorithm needs to compute m^2 pre-verification keys $[\mathsf{ck}_{f,(h,h')}]_2$ for every $h,h'\in[m]$, where for computing each of the keys one needs to sum over ℓ^3 distinct group elements from ck. Similarly, for $\Pi_{\mathsf{quad}}.\mathsf{Ver}$ the running time is dominated by the final check, which involves a sum over m^2 elements in $\mathbb{G}_T$ where each of them is the output of a pairing computation.

For $\Pi_{\mathsf{quad}}.\mathsf{Prove}$, the naïve complexity is $O(m^2\ell^5)$, but the prover can leverage the structure of the proof terms to compute $[\pi_f]_1$ faster. For fixed h,h',k, consider

$$
[\pi_{f,k}^{(h,h')}]_1=\sum_{\substack{i,j,i',j'=1\\(i,j)\neq(i',j')}}^{\ell} f_{i,j,k}^{(h,h')}x_{i'}^{(h)}x_{j'}^{(h')}[\beta^{k}\alpha^{\ell^2+1-(i'-i)-\ell(j'-j)}]_2.
$$

[1] Such classes of sparse functions actually appear naturally in some applications of functional commitents, as they capture gate functions in arithmetic circuits. In Chapter 6, we denote them by separable functions (Theorem 6.15).

Naturally, $[\pi_f]_1 = \sum_{h,h' \in [m], k \in [\ell]} [\pi_{f,k}^{(h,h')}]_1$. To compute each of the $[\pi_{f,k}^{(h,h')}]_1$ efficiently, define the univariate polynomials

$$p_f(\alpha) = \sum_{i,j \in [\ell]} f_{i,j,k} \alpha^{\ell^2+1-i-\ell j}, \quad p_x(\alpha) = \sum_{i,j \in [\ell]} x_i x_j \alpha^{i+\ell j}.$$

Both $p_f(\alpha)$ and $p_x(\alpha)$ are polynomials of degree ℓ^2 on α. Then, the prover can compute all terms on $[\pi_{f,k}^{(h,h')}]_1$ by simply calculating the product of polynomials $p_f(\alpha) \cdot p_x(\alpha)$, which gives:

$$p_f(\alpha) \cdot p_x(\alpha) = \sum_{i,j,i',j'=1}^{\ell} f_{i,j,k}^{(h,h')} x_{i'}^{(h)} x_{j'}^{(h')} \alpha^{\ell^2+1-(i'-i)-\ell(j'-j)}$$

Hence, by ignoring the term on α^{ℓ^2+1} which correspond to $(i,j) = (i',j')$, the prover can compute all the coefficients of $[\pi_{f,k}^{(h,h')}]_1$ via polynomial multiplication. Using an FFT-based polynomial multiplication algorithm, this requires $O(\ell^2 \log \ell^2)$ field operations. As this has to be done for every $h, h' \in [m]$ and for every $k \in [\ell]$ separately, the total running time of the prover is dominated by the time required to do $O(m^2\ell^3 \log \ell^2)$ field operations plus the time required to power and add all group elements, which involve $O(m^2\ell^3)$ group operations. □

Lemma 5.15. *If Assumption 2 (Theorem 5.3) holds over* bgp, Π_{quad} *satisfies evaluation binding.*

Proof. We prove security directly for the multi-input scheme, as the single-input scheme is a special case of it. Let $\mathcal{A}$ be an adversary against evaluation binding of Π_{quad}. We construct an adversary $\mathcal{B}$ against the assumption as follows. $\mathcal{B}(\mathsf{bgp}, \mathcal{S}(\alpha))$ samples $\beta, \gamma_p, \delta_p, \gamma_e, \delta_e \leftarrow\$ \mathbb{F}$ uniformly at random and generates a commitment key ck as follows:

$$\mathsf{ck} = \left\{ \begin{array}{l} \{[\alpha^i]_1, [\alpha^i]_2\}_{\substack{i=1 \\ i \neq \ell^2+1}}^{2\ell^2+1}, [\alpha^{\ell^2+1}]_2, \ \{\beta^i[1]_1, \beta^i[1]_2\}_{i=1}^{\ell}, \{\beta^k[\alpha^i]_1, \beta^k[\alpha^i]_2\}_{\substack{i,k=1, \\ i \neq \ell^2+1}}^{(2\ell^2+1,\ell)}, \\ \left\{ \left(\gamma_p[\alpha^{\ell(i-1)}]_1 + \delta_p[\alpha^i]_1 \right), (\gamma_e[\alpha^i]_1 + \delta_e \beta^i[1]_1) \right\}_{i=1}^{\ell}, \gamma_p[1]_2, \delta_p[1]_2, \gamma_e[1]_2, \delta_e[1]_2 \end{array} \right\}$$

It is straightforward to see that ck is distributed identically as the output of $\mathsf{Setup}(1^\lambda, 1^\ell)$. Then, $\mathcal{B}$ calls $\mathcal{A}(\mathsf{ck})$ and parses the proofs and outputs of $\mathcal{A}$ as

$$\begin{aligned} \pi_{\mathsf{quad}} &= \left(([\hat{c}_h]_2)_{h \in [m]}, ([\tilde{c}_{h,h'}]_1)_{h,h' \in [m]}, [\pi_f]_1, [\check{d}]_1 \right) \\ \pi'_{\mathsf{quad}} &= \left(([\hat{c}'_h]_2)_{h \in [m]}, ([\tilde{c}'_{h,h'}]_1)_{h,h' \in [m]}, [\pi'_f]_1, [\check{d}']_1 \right) \end{aligned}$$

respectively. Note that if $\mathcal{A}$ is successful, it must be that $[d]_1 \neq [d']_1$. Finally, $\mathcal{B}$ outputs $[U]_1 = [\pi_f]_1 - [\pi'_f]_1$, $[V]_1 = [d']_1 - [d]_1$, and $W = [\check{d}]_1 - [\check{d}']_1$.

Next, we show that $\mathcal{B}$ is a successful adversary against the assumption. First, due to the non-degeneracy of the pairing (in the $\mathbb{G}_2$ equality check and tensor product check), we know that $[\tilde{c}_{h,h'}]_1 = [\tilde{c}'_{h,h'}]_1$ for every $h, h' \in [m]$.

Second, due to the degree check, we have that:

$$[d]_1 \cdot [\alpha^{2\ell^2+1}]_2 = [\check{d}]_1 \cdot [1]_2 \quad \text{and} \quad [d']_1 \cdot [\alpha^{2\ell^2+1}]_2 = [\check{d}']_1 \cdot [1]_2,$$

Subtracting both sides yields $([d']_1 - [d]_1) \cdot [\alpha^{2\ell^2+1}]_2 = ([\check{d}]_1 - [\check{d}']_1) \cdot [1]_2$.

Third, due to the quadratic function check,

$$[\pi_f]_1 \cdot [1]_2 + [d]_1 \cdot [\alpha^{\ell^2+1}]_2 = \sum_{h,h' \in [m]} [\tilde{c}_{h,h'}]_1 \cdot [\mathsf{ck}_{f,(h,h')}]_2 = [\pi'_f]_1 \cdot [1]_2 + [d']_1 \cdot [\alpha^{\ell^2+1}]_2,$$

Again, subtracting elements from both sides of the equality yields that $([\pi_f]_1 - [\pi'_f]_1) \cdot [1]_2 = ([d']_1 - [d]_1) \cdot [\alpha^{\ell^2+1}]_2$, which concludes the proof.

□

5.4.4 Equality Type Chaining

Our last type chaining proof system is Π_{eq}, which is a simple equality proof system between a β-basis commitment $[d]_1 = \sum_{i=1}^{\ell} x_i [\beta^i]_1$ and a α-basis commitment $[c]_1 = \sum_{i=1}^{\ell} x_i [\alpha^i]_1$. Naturally, Π_{eq} is defined with respect to $\mathsf{Com}^{(\mathsf{eq})}$ and Com. We introduce the construction in Figure 5.5.

We remark that the prover time is linear, $O(\lambda \ell)$. Moreover, there is no need for specific efficient verification algorithms as the verifier already runs in time $O(\lambda)$. Despite Π_{eq} being the most efficient of our proof systems, note that it requires the strongest assumption, as we cannot simulate any of the terms of the commitment key in the security proof. It is an interesting open question whether the assumption can be simplified.

$\Pi_{\mathsf{eq}}.\mathsf{Prove}(\mathsf{ck}, \boldsymbol{x})$:

Compute and output $[\pi_{\mathsf{eq}}]_1 \leftarrow \sum_{i=1}^{\ell} x_i \left([\gamma_e \alpha^i + \delta_e \beta^i]_1\right)$.

$\Pi_{\mathsf{eq}}.\mathsf{Ver}(\mathsf{ck}, \mathsf{com}^{(\mathsf{x})}, \mathsf{com}^{(\mathsf{y})}, \pi_{\mathsf{eq}})$:

- Parse $\mathsf{com}^{(\mathsf{x})} = [d]_1$ and $\mathsf{com}^{(\mathsf{y})} = [c]_1$.
- Check $[\pi_{\mathsf{eq}}]_1 \cdot [1]_2 = [c]_1 \cdot [\gamma_e]_2 + [d]_1 \cdot [\delta_e]_2$.

Figure 5.5: *Equality type chaining proof* Π_{eq}.

Lemma 5.16. *The construction* Π_{eq} *satisfies correctness.*

Proof. The verification equation is given by:

$$\pi_{\mathsf{eq}} \cdot 1 = \sum_{i=1}^{\ell} x_i \gamma_e \alpha^i + \sum_{i=1}^{\ell} x_i \delta_e \beta^i = c \cdot \gamma_e + d \cdot \delta_e$$

□

Lemma 5.17. *Suppose that Assumption 3 (Theorem 5.5) holds over* bgp. *Then,* Π_{eq} *satisfies evaluation binding.*

Proof. Let $\mathcal{A}$ be an adversary against evaluation binding of Π_{eq}. We construct an adversary $\mathcal{B}$ against the assumption as follows. $\mathcal{B}(\mathsf{bgp}, \mathcal{S}(\alpha, \beta, \gamma_e, \delta_e))$ samples $\gamma_p, \delta_p \leftarrow\$ \mathbb{F}$ uniformly at random and generates a commitment key ck as follows:

$$\mathsf{ck} = \left\{ \begin{array}{l} \{[\alpha^i]_1, [\alpha^i]_2\}_{\substack{i=1 \\ i \neq \ell^2+1}}^{2\ell^2+1}, [\alpha^{\ell^2+1}]_2, \ \{[\beta^i]_1, [\beta^i]_2\}_{i=1}^{\ell}, \{[\beta^k \alpha^i]_1, [\beta^k \alpha^i]_2\}_{\substack{i,k=1, \\ i \neq \ell^2+1}}^{(2\ell^2+1,\ell)}, \\ \left\{ \left(\gamma_p[\alpha^{\ell(i-1)}]_1 + \delta_p[\alpha^i]_1\right), [\gamma_e \alpha^i + \delta_e \beta^i]_1 \right\}_{i=1}^{\ell}, \gamma_p[1]_2, \delta_p[1]_2, [\gamma_e]_2, [\delta_e]_2 \end{array} \right\}$$

Clearly, ck is distributed identically as the output of $\mathsf{Setup}(1^\lambda, 1^\ell)$. Then, $\mathcal{B}$ calls $\mathcal{A}(\mathsf{ck})$ and parses the proofs and outputs of $\mathcal{A}$ as $([d]_1, [c]_1, [c']_1, [\pi_{\mathsf{eq}}]_1, [\pi'_{\mathsf{eq}}]_1)$, respectively. Note that if $\mathcal{A}$ is successful, it must be that $[c]_1 \neq [c]_1$. Then, $\mathcal{B}$ simply outputs $[U]_1 = [\pi_{\mathsf{eq}}]_1 - [\pi'_{\mathsf{eq}}]_1, [V]_1 = [c]_1 - [c']_1$

The claim follows as in the previous proofs, by subtracting the verification equation satisfied by $[c]_1$ and $[c']_1$ for the same input $[d]_1$, which yields:

$$[U]_1 \cdot [1]_2 = ([\pi_{\mathsf{eq}}]_1 - [\pi'_{\mathsf{eq}}]_1) \cdot [1]_2 = ([c]_1 - [c']_1) \cdot [\gamma_e]_2 = [V]_1 \cdot [\gamma_e]_2.$$

□

5.4.5 CFC Construction

Finally, we present the CFC construction in Figure 5.6. Recall that the CFC algorithms Setup and Com are described in Section 5.4.1, as well as the auxiliary commitment algorithms $\mathsf{Com}^{(\mathsf{pow})}$ and $\mathsf{Com}^{(\mathsf{eq})}$.

Theorem 5.18. *If* $\Pi_{\mathsf{pow}}, \Pi_{\mathsf{quad}}, \Pi_{\mathsf{eq}}$ *satisfy evaluation binding, the construction* CFC *in Figure 5.6 also satisfies evaluation binding.*

Proof. The proof is a game-based proof which relies on the evaluation binding of the internal proof systems Π_{pow}, Π_{quad} and Π_{eq}. Let $\mathcal{A}$ be an adversary against the evaluation binding of CFC, this is, $\mathcal{A}(\mathsf{ck})$ outputs a function $f : \mathcal{M}^{\ell \cdots m} \to \mathcal{M}^{\ell} \in \mathcal{F}_{\mathsf{quad}}$, a tuple of input commitments $(\mathsf{com}_1, \ldots, \mathsf{com}_m)$, two different output commitments $\mathsf{com}_y, \mathsf{com}'_y$, and two corresponding valid opening proofs π, π'. We refer to this as the standard evaluation binding game Hyb^0.

We define a sequence of game hops in Figure 5.7. In these games, we parse the proofs as follows:

$$\begin{aligned} \pi &= \left(\{\pi_{\mathsf{pow},i}\}_{i \in [m]}, \pi_{\mathsf{quad}}, \pi_{\mathsf{eq}}, \{[\hat{c}_i]_1\}_{i \in [m]}, [d]_1\right), \\ \pi' &= \left(\{\pi'_{\mathsf{pow},i}\}_{i \in [m]}, \pi'_{\mathsf{quad}}, \pi'_{\mathsf{eq}}, \{[\hat{c}'_i]_1\}_{i \in [m]}, [d']_1\right). \end{aligned}$$

On each game hop, we add the condition that one of the internal commitments in π must be equal to the corresponding one in π'.

We now proceed to bound the advantage of any PPT adversary $\mathcal{A}$ against Hyb^0 via a series of lemmas. Note that Hyb^0 and $\mathsf{Hyb}^{1,0}$ are identical.

```
CFC.FuncProve(ck, (x_1, ..., x_m), f) :
```

- $\boldsymbol{y} \leftarrow f(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m)$.
- $[\hat{c}_i]_1 \leftarrow \mathsf{CFC.Com}^{(\mathsf{pow})}(\mathsf{ck}, \boldsymbol{x}_i)$ for every $i \in [m]$.
- $[d]_1 \leftarrow \mathsf{CFC.Com}^{(\mathsf{eq})}(\mathsf{ck}, \boldsymbol{y})$.
- $\pi_{\mathsf{pow},i} \leftarrow \Pi_{\mathsf{pow}}.\mathsf{Prove}(\mathsf{ck}, \boldsymbol{x}_i)$ for every $i \in [m]$.
- $\pi_{\mathsf{quad}} \leftarrow \Pi_{\mathsf{quad}}.\mathsf{Prove}(\mathsf{ck}, (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m), f)$.
- $\pi_{\mathsf{eq}} \leftarrow \Pi_{\mathsf{eq}}.\mathsf{Prove}(\mathsf{ck}, \boldsymbol{y})$.
- Output $\pi = (\{\pi_{\mathsf{pow},i}\}_{i\in[m]}, \pi_{\mathsf{quad}}, \pi_{\mathsf{eq}}, \{[\hat{c}_i]_1\}_{i\in[m]}, [d]_1)$.

$\underline{\mathsf{CFC.PreFuncVer}(\mathsf{ck}, f)}$:

- Compute $\mathsf{ck}_{\mathsf{quad}} \leftarrow \Pi_{\mathsf{quad}}.\mathsf{PreVer}(\mathsf{ck}, f)$.
- Compute $\mathsf{ck}_{\mathsf{eq}} \leftarrow \Pi_{\mathsf{eq}}.\mathsf{PreVer}(\mathsf{ck})$.
- Output $(\mathsf{ck}_{\mathsf{quad}}, \mathsf{ck}_{\mathsf{eq}})$.

$\underline{\mathsf{CFC.EffFuncVer}(\mathsf{ck}_f, (\mathsf{com}_1, \ldots, \mathsf{com}_m), \mathsf{com}_y, \pi)}$:

- Parse $\pi = (\{\pi_{\mathsf{pow},i}\}_{i\in[m]}, \pi_{\mathsf{quad}}, \pi_{\mathsf{eq}}, \{[\hat{c}_i]_1\}_{i\in[m]}, [d]_1)$.
- Check that $\Pi_{\mathsf{pow}}.\mathsf{Ver}(\mathsf{ck}, \mathsf{com}_i, [\hat{c}_i]_1, \pi_{\mathsf{pow},i}) = 1$ for every $i \in [m]$.
- Check that $\Pi_{\mathsf{quad}}.\mathsf{Ver}(\mathsf{ck}_{\mathsf{quad}}, \{(\mathsf{com}_1, [\hat{c}]_1)\}_{i\in[m]}, [d]_1, \pi_{\mathsf{quad}}) = 1$.
- Check that $\Pi_{\mathsf{eq}}.\mathsf{Ver}(\mathsf{ck}_{\mathsf{eq}}, [d]_1, \mathsf{com}_y, \pi_{\mathsf{eq}}) = 1$.

Figure 5.6: *CFC construction from the type chaining proof systems* Π_{pow}, Π_{quad} *and* Π_{eq}.

Lemma 5.19. *For any $i \in [m]$, there exists a PPT adversary $\mathcal{B}$ against the evaluation binding of Π_{pow} such that*

$$\left|\Pr[\mathsf{Hyb}^{1,i-1}_{\mathcal{A}}(\lambda) = 1] - \Pr[\mathsf{Hyb}^{1,i}_{\mathcal{A}}(\lambda) = 1]\right| \leq \mathsf{Adv}^{\mathrm{evbind}}_{\Pi_{\mathsf{pow}},\mathcal{B}}(\lambda).$$

Proof. Fix $i \in [m]$. We build an adversary $\mathcal{B}$ as follows. $\mathcal{B}$ receives input ck from its challenger and runs $\mathcal{A}(\mathsf{ck})$. Then, it parses the proofs π, π' of $\mathcal{A}$ and retrieves the tuples $([\hat{c}_i]_1, \pi_{\mathsf{pow},i})$ and $([\hat{c}'_i]_1, \pi'_{\mathsf{pow},i})$. Finally, $\mathcal{B}$ simply forwards $(\mathsf{com}_i, [\hat{c}_i]_1, [\hat{c}'_i]_1, \pi_{\mathsf{pow},i}, \pi'_{\mathsf{pow},i})$ to its challenger.

We argue that if $\mathcal{A}$ wins in $\mathsf{Hyb}^{1,j-1}$ but not in $\mathsf{Hyb}^{1,i}$, then $\mathcal{B}$ also wins in its corresponding game. First, as $\mathcal{A}$ wins in $\mathsf{Hyb}^{1,j-1}$, it holds that $\Pi_{\mathsf{pow}}.\mathsf{Ver}(\mathsf{ck}, \mathsf{com}_i, [\hat{c}_i]_1, \pi_{\mathsf{pow},i}) = 1$ and that $\Pi_{\mathsf{pow}}.\mathsf{Ver}(\mathsf{ck}, \mathsf{com}_i, [\hat{c}'_i]_1, \pi'_{\mathsf{pow},i}) = 1$. Finally, as $\mathcal{A}$ does not win in $\mathsf{Hyb}^{1,j}$, it must be that $[\hat{c}_i]_1 \neq [\hat{c}'_i]_1$. Hence, $\mathcal{B}$ breaks evaluation binding of Π_{pow}. □

Lemma 5.20. *There exists a PPT adversary $\mathcal{B}$ against the evaluation binding of Π_{quad} such that*

$$\Pr[\mathsf{Hyb}^{1,m}_{\mathcal{A}}(\lambda) = 1] \leq \Pr[\mathsf{Hyb}^{2}_{\mathcal{A}}(\lambda) = 1] + \mathsf{Adv}^{\mathrm{evbind}}_{\Pi_{\mathsf{quad}},\mathcal{B}}(\lambda).$$

Proof. The proof is analogous to that of Theorem 5.19. $\mathcal{B}$ calls $\mathcal{A}(\mathsf{ck})$ and parses the proofs π, π'. Note that if $\mathcal{A}$ wins in $\mathsf{Hyb}^{1,m}$ but not in Hyb^{2}, it must be that $(\mathsf{com}_i, [\hat{c}_i]_1) =$

```
$\mathsf{Hyb}^0_{\mathcal{A}}(\lambda)$:
$\mathsf{ck} \leftarrow \mathsf{CFC.Setup}(1^\lambda, 1^\ell)$
$(f, (\mathsf{com}_i)_{i\in[m]}, \mathsf{com}_y, \mathsf{com}'_y, \pi, \pi') \leftarrow \mathcal{A}(\mathsf{ck})$
Parse $\pi = (\{\pi_{\mathsf{pow},i}\}_{i\in[m]}, \pi_{\mathsf{quad}}, \pi_{\mathsf{eq}}, \{[\hat{c}_i]_1\}_{i\in[m]}, [d]_1)$
Parse $\pi' = (\{\pi'_{\mathsf{pow},i}\}_{i\in[m]}, \pi'_{\mathsf{quad}}, \pi'_{\mathsf{eq}}, \{[\hat{c}'_i]_1\}_{i\in[m]}, [d']_1)$
assert $\mathsf{com}_y \neq \mathsf{com}'_y$
assert $\mathsf{CFC.Ver}(\mathsf{ck}, f, (\mathsf{com}_i)_{i\in[m]}, \mathsf{com}_y, \pi) = 1$
assert $\mathsf{CFC.Ver}(\mathsf{ck}, f, (\mathsf{com}_i)_{i\in[m]}, \mathsf{com}'_y, \pi') = 1$
return 1

$\mathsf{Hyb}^{1,i}_{\mathcal{A}}(\lambda), 0 \le i \le m$:
    // identical to $\mathsf{Hyb}^0_{\mathcal{A}}(\lambda)$ until before "return 1"
assert $[\hat{c}_j]_1 = [\hat{c}'_j]_1 \ \forall j \in [i]$
return 1

$\mathsf{Hyb}^2_{\mathcal{A}}(\lambda)$:
    // identical to $\mathsf{Hyb}^{1,m}_{\mathcal{A}}(\lambda)$ until before "return 1"
assert $[d]_1 = [d']_1$
return 1
```

Figure 5.7: *Games* $\mathsf{Hyb}^0, \mathsf{Hyb}^{1,i}, \mathsf{Hyb}^2$ *for the proof of evaluation binding of* CFC*. We highlight changes between games.*

$(\mathsf{com}_i, [\hat{c}'_i]_1)$ for every $i \in [m]$, and that $[d]_1 \neq [d']_1$. Hence, $\mathcal{B}$ simply forwards the tuple $((\mathsf{com}_i, [\hat{c}_i]_1)_{i\in[m]}, [d]_1, [d']_1, \pi_{\mathsf{quad}}, \pi'_{\mathsf{quad}})$ from $\mathcal{A}(\mathsf{ck})$'s outputs to its challenger, breaking evaluation binding of Π_{quad}. □

Lemma 5.21. *There exists a PPT adversary $\mathcal{B}$ against the evaluation binding of Π_{eq} such that*

$$\Pr[\mathsf{Hyb}^2_{\mathcal{A}}(\lambda) = 1] \le \mathsf{Adv}^{\mathrm{evbind}}_{\Pi_{\mathsf{eq}},\mathcal{B}}(\lambda).$$

Proof. The proof is again analogous to that of Theorem 5.19. $\mathcal{B}$ calls $\mathcal{A}(\mathsf{ck})$ and parses the proofs π, π'. Note that if $\mathcal{A}$ wins in Hyb^2, it must be that $[d]_1 = [d']_1$ and that $\mathsf{com}_y \neq \mathsf{com}'_y$. Hence, $\mathcal{B}$ simply forwards $([d]_1, \mathsf{com}_y, \mathsf{com}'_y, \pi_{\mathsf{eq}}, \pi'_{\mathsf{eq}})$ from $\mathcal{A}(\mathsf{ck})$'s outputs to its challenger, breaking evaluation binding of Π_{eq}. □

The overall proof follows by aggregating the bounds from Lemmas 5.19, 5.20 and 5.21, as we obtain that

$$\Pr[\mathsf{Hyb}^0_{\mathcal{A}}(\lambda) = 1] \le n \cdot \mathsf{Adv}^{\mathrm{evbind}}_{\Pi_{\mathsf{pow}},\mathcal{B}}(\lambda) + \mathsf{Adv}^{\mathrm{evbind}}_{\Pi_{\mathsf{quad}},\mathcal{B}}(\lambda) + \mathsf{Adv}^{\mathrm{evbind}}_{\Pi_{\mathsf{eq}},\mathcal{B}}(\lambda).$$

□

6

Circuit-Succinct Algebraic Batch Arguments for NP

In this chapter, we present the first algebraic construction of a batch argument for NP that is *circuit-succinct*, this is, such that the proof size is $\mathcal{O}(\lambda \cdot |\mathtt{w}|)$ where $\mathtt{w}$ is the NP witness. The chapter is based on results from the article *"Circuit-Succinct Algebraic Batch Argument from Standard Assumptions"*[BFL25a], which is in submission at the time of writing.

The chapter is structured as follows. In Section 6.1, we present a succinct summary of contributions, followed by an extensive technical overview in Section 6.2. In Section 6.4, we introduce a novel notion of projective chainable functional commitments (PCFC) and our circuit model. In Section 6.5 and Section 6.6, we present our basic and circuit-succinct compilers, respectively, which transform a PCFC into a BARG. Finally, in Section 6.7 we construct an algebraic PCFC from the MDDH assumption over bilinear groups with the properties required by our compilers.

6.1 Contributions

We continue the line of work established by [WW22, GLWW24] on obtaining algebraic constructions of batch arguments for NP. We present a new construction of BARG for NP directly over bilinear pairings under the matrix decisional Diffie-Hellman (MDDH) assumption, which achieves proofs of size $\mathcal{O}(\lambda \cdot |\mathtt{w}|)$ with matching online verification time. This improves over the schemes in [WW22, GLWW24] which present a proof size of $|\pi| = \mathcal{O}(\lambda \cdot |C|)$, and in which an online verifier runs in time $\Omega(\lambda \cdot |C|)$.

Additionally, our BARG natively supports proving k different circuits, one for each statement. In prior work, this property is achieved with the overhead of using a universal circuit $\mathcal{U}$ to define the batch language and including the description of each circuit in the statement, i.e., by proving that $\mathcal{U}((C_i, \mathtt{x}_i), \mathtt{w}_i) = 1$.

To achieve the above result, we rely on a new notion that we call *projective chainable functional commitments (PCFC)*, which is a combination of projective commitments (PC) [GZ21, WW24b] and chainable functional commitments (CFC) [BCFL23], together with two technical ingredients:

1. A black-box compiler that transforms a PCFC with certain properties into a BARG; and
2. An algebraic construction of such a PCFC based on the MDDH assumption.

We believe that our black-box compiler might be of independent interest for understanding the connection between functional commitments (FCs) [LRY16] and BARGs, which relax SNARGs by forgoing soundness and succinctness respectively. It might also pave the way towards obtaining other direct algebraic constructions of BARGs from different assumptions, such as lattices, which is an open problem.

6.2 Technical Overview

The main result of this work is an algebraic construction of BARG with proof size and online verification time $O(\lambda \cdot |\mathtt{w}|)$. We obtain this result by compiling, in a black-box manner, a new commitment primitive that we call *projective chainable functional commitment* (PCFC), for which we give an algebraic construction over bilinear groups.

To give a high-level overview of our approach, in Section 6.2.1, we begin by recalling and generalizing the notions of projective commitments (PC), [GZ21, WW24b] and chainable functional commitments (CFC) [BCFL23], and introducing the notion of PCFC. We then revisit in Section 6.2.2 the algebraic BARG construction of Waters and Wu [WW22] by viewing it as a black-box compilation from a PCFC to a BARG. From this abstract perspective, it becomes clear how, given a PCFC with stronger properties, the compiler can be tweaked to improve the proof size and online verification time from $O(\lambda \cdot |C|)$ to $O(\lambda \cdot |\mathtt{w}|)$. We thus present in Section 6.2.3 how to obtain such a PCFC from pairing groups based on the MDDH assumption. Finally, in Section 6.2.4 we conclude with a summary of our results and an outlook on future work.

6.2.1 Projective Chainable Functional Commitments (PCFC)

A (deterministic) commitment scheme allows the generation of a short commitment com_x which is computationally binding to a vector $\boldsymbol{x} \in \mathcal{M}^\ell$ potentially much longer than com_x. Many variations of the notion exist that extend the basic functionality and security of a commitment scheme. Our new notion, PCFC, which we formalise in Section 6.4, is a combination of two such variations: projective commitments (PC) and chainable functional commitments (CFC).

Projective Commitments (PC). In a *Projective commitment (PC)* scheme, introduced by Wee and Wu [WW24b][1], the commitment key ck of the scheme can be sampled in two modes: the *normal mode* and the *projective mode*. Sampling the key in normal mode leads to a standard commitment key ck that computationally binds a vector $\boldsymbol{x} \in \mathcal{M}^\ell$ to a commitment com. In projective mode, the commitment key ck is sampled together with a trapdoor td that encodes a hidden index set $I \subseteq [\ell]$. Here, ck hides a *projective key*, which is essentially a secondary commitment key that is sampled only at the positions indexed by

[1] Similar properties for commitments appear in [GZ21] under the name of G-extractability.

I. The trapdoor td allows one to project any commitment com to $\boldsymbol{x} \in \mathcal{M}$ to a commitment pcom to the subvector $\boldsymbol{x}_I \in \mathcal{M}^I$.[2] Without knowing td, both modes are computationally indistinguishable.

In more detail, a PC features the following algorithms (among others):

- $\mathsf{ProjSetup}(1^\lambda, 1^\ell, I)$: Generates an ordinary-looking commitment key ck together with a trapdoor td which encodes a secret index set $I \subseteq [\ell]$.
- $\mathsf{ProjCom}(\mathsf{td}, \boldsymbol{x}) \to \mathsf{pcom}$: Generate a projective commitment pcom of the subvector $\boldsymbol{x}_I$.
- $\mathsf{Proj}(\mathsf{td}, \mathsf{com}) \to \mathsf{pcom}$: Project a commitment com into a projective commitment pcom.

Projective commitments are closely related to somewhere extractable commitments [HW15, CJJ21], which essentially allow one to generate commitment keys so that a commitment com_x is *statistically binding* at some positions $I \subset [\ell]$. The trapdoor td allows one to actually extract $\boldsymbol{x}_I \in \bar{\mathcal{M}}$ (where $\bar{\mathcal{M}} \subseteq \mathcal{M}$ is a smaller extraction space) leveraging the statistical binding property. In this work, we enhance the original notion of projective commitments to also provide somewhere extractability. This is, our commitments also feature the following algorithm:

- $\mathsf{ProjExt}(\mathsf{td}, \mathsf{com})$: Given com, which is guaranteed[3] to be committing to a vector whose positions indexed by I belong to the projective subspace $\bar{\mathcal{M}}^I$, extract $\boldsymbol{x}_I \in \bar{\mathcal{M}}^I$ such that $\mathsf{ProjCom}(\mathsf{td}, \boldsymbol{x}_I) = \mathsf{Proj}(\mathsf{td}, \mathsf{com})$.

To achieve the extractability outlined in the above simplified description, a commitment com has to be of size linear in $|I|$, which will be too large for our application of constructing circuit-succinct BARGs. Instead, we further augment the syntax of the commitment algorithm Com (among others) to take as input another index set J chosen from some admissible set $\mathcal{J} \subseteq 2^{[\ell]}$ so that the resulting commitment com is committing to a subsector $\boldsymbol{x}_J \in \mathcal{M}^J$. Now, the extractability guarantee is only required to hold at the positions $I \cap J$ for any $J \in \mathcal{J}$. By picking $\mathcal{J}$ appropriately, we can ensure that $|I \cap J|$ and hence $|\mathsf{com}|$ is never too large which eventually allows us to achieve circuit-succinctness.

Chainable Functional Commitments (CFC). While ordinary commitments are "all-or-nothing", functional commitments (FC) [LRY16] enable the generation of a succinct (weakly sound) proof asserting the correctness of $\boldsymbol{y} = f(\boldsymbol{x})$ for the vector $\boldsymbol{x}$ committed in com_x. *Chainable functional commitments (CFC)* [BCFL23] extend this functionality and allow to generate proofs for $\boldsymbol{y} = f(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k)$ where both the inputs $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k$ and the output $\boldsymbol{y}$ are committed. A CFC is "chainable" in the sense that it allows one to prove relations between a sequence of committed inputs and outputs, in a "commit-and-prove" fashion. A (C)FC usually satisfies a relaxed notion of soundness called evaluation binding – no efficient prover could produce valid proofs for $(f, \mathsf{com}_x, \mathsf{com}_y)$ and $(f, \mathsf{com}_x, \mathsf{com}'_y)$ for $\mathsf{com}_y \neq \mathsf{com}'_y$. In this work, however, we deviate from this usual notion and require a stronger property that we introduce next.

[2] The subvector $\boldsymbol{x}_I$ is obtained by taking the entries of $\boldsymbol{x}$ indexed by I.
[3] To be (weakly) asserted by another algorithm called $\mathsf{ProjProve}$ and verified by $\mathsf{ProjVer}$.

Projective Chainable Functional Commitments (PCFC). Although PC and CFC are useful notions in their own rights, combining them into a PCFC makes the result more powerful than the sum of the two, since it allows us to reason about the functional relations between projective commitments and their extracted messages, whose spaces are hidden from a malicious prover. Such a property already resembles the functionality of a BARG, as both allow to extract part of the inputs to the batch computation while verifying its correctness.

In more detail, recall that a PC commitment key ck hides a secret index set $I \subseteq [\ell]$ while a CFC allows the generation of functional proofs. To glue these two notions together, we consider a class of functions called "I-separable functions" and define a property called *projective chain binding* with respect to these functions.

I-Separable Functions. We say that a function $f : (\mathcal{M}^\ell)^k \to \mathcal{M}^\ell$ is I-separable if the coordinates in I of the output vector only depend on the coordinates in I of each input vector.[4] A simple example is when f is the parallel application of another function $f_I : \mathcal{M}^{\ell/k} \to \mathcal{M}^{\ell/k}$ such that $f(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k) = (f_I(\boldsymbol{x}_1), \ldots, f_I(\boldsymbol{x}_k))$.

Projective Chain Binding. We say that a PCFC is projective chain binding for an I-separable function f if no PPT adversary, given both a commitment key ck, which is trapdoored to be projective at I, *and the trapdoor* td, can produce tuples $(\mathsf{com}, \mathsf{com}_y, \pi)$ and $(\mathsf{com}', \mathsf{com}'_y, \pi')$ where

- π and π' are valid CFC proofs for $f(\mathsf{com}) = \mathsf{com}_y$ and $f(\mathsf{com}') = \mathsf{com}'_y$, respectively, and
- $\mathsf{Proj}(\mathsf{td}, \mathsf{com}) = \mathsf{Proj}(\mathsf{td}, \mathsf{com}')$, but
- $\mathsf{Proj}(\mathsf{td}, \mathsf{com}_y) \neq \mathsf{Proj}(\mathsf{td}, \mathsf{com}'_y)$.

Essentially, the projective chain binding property allows a security reduction to transfer the evaluation binding property of the CFC to hold also in the projective subspace. This is especially powerful if we leverage the somewhere extractability property of PC on the projective subspace. It is worth noting that some of the proof systems that are internally used to build the FC in [WW24b] achieve a similar notion to projective chain binding (although the notion is not attained by the FC itself). In their case, however, the projection trapdoor is not available to the adversary. As we discuss in the next section, our strengthening is crucial when proving the security of our circuit-succinct BARG construction.

6.2.2 Black-Box Compiler from PCFC to BARG

Recall that in a BARG the prover aims to convince the verifier that k NP statements $\mathtt{x}_1, \ldots, \mathtt{x}_k$ are valid. That is, the prover knows witnesses $\mathtt{w}_1, \ldots, \mathtt{w}_k$ such that $C(\mathtt{x}_i, \mathtt{w}_i) = 1$ for all $i \in [k]$. The main security property of BARGs, known as somewhere extractability, requires the existence of an efficient extractor which, given a valid proof, can extract a valid witness for the i^*-th instance $\mathtt{w}_{i^*}$, where i^* is a hidden index programmed into the common reference string.

[4] Our formal definition generalizes to allow different separability sets for inputs and outputs.

Suppose the circuit C is such that the input-output relation of each gate $g \in C$ is captured by a gate function χ_g represented as a quadratic polynomial over a field $\mathbb{F}$, and write $x_{i,g}$ for the output value from the g-th gate (this is, the g-th circuit wire) in the evaluation of $C(\mathtt{x}_i, \mathtt{w}_i)$. In the following, let $\mathcal{G}_{\mathsf{stmt}}, \mathcal{G}_{\mathsf{wit}}, \mathcal{G}_{\mathsf{out}}$ be the sets of wires of C corresponding to the statement input, the witness input and all gate outputs, respectively.

Basic Compiler. To obtain an improved algebraic BARG, a natural starting point is the algebraic BARG of Waters and Wu [WW22]. Our first step is to view[5] their construction as black-box compiler which turns a PCFC into a BARG. We formalize this compiler in Section 6.5 and give a high-level overview here.

Basic Compiler (Abstraction of [WW22]):

1. The prover PCFC-commits to the evaluation of $C(\mathtt{x}_i, \mathtt{w}_i)$ in a wire-by-wire fashion across all k circuits, i.e. for each wire $g \in \mathcal{G}_{\mathsf{wit}} \cup \mathcal{G}_{\mathsf{out}}$, they commit to the values $\boldsymbol{x}_g = (x_{1,g}, \ldots, x_{k,g})$ of the g-th wire as com_g.
2. For each witness wire $g \in \mathcal{G}_{\mathsf{wit}}$, compute a PC proof for the binariness of $\boldsymbol{x}_g$ committed in com_g, i.e. $\boldsymbol{x}_g \in \{0,1\}^k$, where $\{0,1\} = \bar{\mathcal{M}}$ plays the role of the projective subspace.
3. For each gate output wire $g \in \mathcal{G}_{\mathsf{out}}$, compute a CFC proof for $\boldsymbol{x}_g$ committed in com_g being computed correctly with χ_g w.r.t. the commitments of the input wires.

Instantiating the PCFC with the implicit PCFC for quadratic functions from [WW22] then yields a BARG with proof size $O(\lambda \cdot |C|)$, where the linear dependency on $|C|$ is due to the generation of commitments and proofs in a wire-by-wire fashion.[6]

Circuit-Succinct Compiler. Without going yet into formal details, one aspect of the security proof of the basic compiler stands out as an obstacle for reducing the proof size to be sublinear in $|C|$: The security reduction has to extract the values of all $|C|$ wires in one of the k circuits, and verify the functional proof of each of the gate functions. The extractability of all $|C|$ wires of a circuit information-theoretically implies that the BARG proof necessarily has size linear in $|C|$.

Towards improving the proof size and online verification time from $O(\lambda \cdot |C|)$ to $O(\lambda \cdot |\mathtt{w}|)$, a crucial observation is that the security reduction does not necessarily need to extract all $|C|$ wires of (the i^*-th copy of) the circuit C but only its input $\mathtt{w}_{i^*}$. That is, if the PCFC is expressive enough to support generation of CFC proofs for arbitrary circuits, then the burden of checking that the circuit is evaluated correctly can be delegated to the

[5] Although Waters and Wu presented the same intuition (obliviously using different terminologies than PCFCs) when explaining their construction in [WW22], formally, they constructed the BARG directly in terms of algebraic operations. This results in a construction with many intertwined components, making it difficult to dissect and improve upon.

[6] Here, we glossed over the detail that the somewhere extractability of the BARG obtained above is proven based on another property of PCFC that we call *functional extractability* instead of projective chain binding. We refer to Section 6.5 for details.

prover. Based on this observation, we obtain the following circuit-succinct compiler, which is detailed in Section 6.6.

Circuit-Succinct Compiler:

1. The prover PCFC-commits to the *witness input wires* of $C(\mathtt{x}_i, \mathtt{w}_i)$ across all k circuits, i.e. for each witness input wire $g \in \mathcal{G}_{\mathsf{wit}}$, they commit to the values $\boldsymbol{x}_g = (x_{1,g}, \ldots, x_{k,g})$ of the g-th wire as com_g.
2. For each witness wire $g \in \mathcal{G}_{\mathsf{wit}}$, compute a PC proof for the binariness of $\boldsymbol{x}_g$ committed in com_g, i.e. $\boldsymbol{x}_g \in \{0,1\}^k$, where $\{0,1\} = \bar{\mathcal{M}}$ plays the role of the projective subspace.
3. Compute a CFC proof for $f_C(\boldsymbol{x}_g : g \in \mathcal{G}_{\mathsf{stmt}} \cup \mathcal{G}_{\mathsf{wit}}) = \mathbf{1}_k$ where f_C consists of k parallel copies of C.

Assuming that the commitments $(\mathsf{com}_g)_{g \in \mathcal{G}_{\mathsf{wit}}}$ and the CFC proof for f_C are sufficiently succinct, we can conclude by inspection that the BARG is circuit-succinct, i.e. a proof is of size $O(\lambda \cdot |\mathtt{w}|)$.

Security of the Circuit-Succinct Compiler. Assuming that the underlying PCFC is projective chain binding, we can prove that the above black-box compiler yields a somewhere-extractable BARG. In the security experiment, we consider a trapdoored setup which programs the PCFC commitment key so that it is projective on the subspace I_{i^*} defined by the hidden index i^* of the BARG extractor. The set I_{i^*} contains the location of the statement and witness of the i^*-th instance, i.e. $(\mathtt{x}_{i^*}, \mathtt{w}_{i^*})$. Recall that a BARG proof contains commitments com_g for each witness input wire $g \in \mathcal{G}_{\mathsf{wit}}$ across all k circuits, along with their projection proofs. The security proof proceeds in two broad steps.[7]

1. Appeal to PC somewhere extractability for each com_g, which ensures the extraction of a binary witness $\mathtt{w}_{i^*} \in \{0,1\}^{\ell_{\mathtt{w}}}$ and that $\mathsf{ProjCom}(\mathsf{td}, \boldsymbol{x}_g) = \mathsf{Proj}(\mathsf{td}, \mathsf{com}_g)$ for each $g \in \mathcal{G}_{\mathsf{wit}}$.
2. Use projective chain binding to ensure that $C(\mathtt{x}_{i^*}, \mathtt{w}_{i^*}) = 1$.

To explain the second step in more detail, in the case $C(\mathtt{x}_{i^*}, \mathtt{w}_{i^*}) = 0$, we can construct a reduction $\mathcal{B}$ from projective chain binding. For simplicity, let us ignore the statement wires $g \in \mathcal{G}_{\mathsf{stmt}}$ (we can think of them being hardwired in f_C) as they are not important for the reduction. The reduction $\mathcal{B}$ runs the malicious prover $\mathcal{A}$ to obtain the commitments com_g for all witness input wires $g \in \mathcal{G}_{\mathsf{wit}}$ and a valid CFC proof π for $f_C(\boldsymbol{x}_g : g \in \mathcal{G}_{\mathsf{wit}}) = \mathbf{1}_k$. Using the trapdoor td, $\mathcal{B}$ can extract the i^*-th witness $\mathtt{w}_{i^*}$ from the commitments com_g. It then uses the extracted witness $\mathtt{w}_{i^*}$ together with dummy witnesses $\mathtt{w}_i = \mathbf{0}_{\ell_{\mathtt{w}}}$ for the other slots $i \neq i^*$ to compute the commitments $(\mathsf{com}'_g)_{g \in \mathcal{G}_{\mathsf{wit}}}$ together with another CFC proof π' for $f_C(\boldsymbol{x}'_g : g \in \mathcal{G}_{\mathsf{wit}}) = \boldsymbol{b}$ for some $\boldsymbol{b} \in \{0,1\}^k$ with $b_{i^*} = 0$. By the correctness of the PCFC, it must hold that $\mathsf{Proj}(\mathsf{td}, \mathsf{com}_g) = \mathsf{Proj}(\mathsf{td}, \mathsf{com}'_g)$ for all $g \in \mathcal{G}_{\mathsf{wit}}$ and that π' is a valid CFC proof. The reduction $\mathcal{B}$ can thus output $(\mathsf{com}_g, \mathsf{com}'_g)_{g \in \mathcal{G}_{\mathsf{wit}}}, \mathsf{com}_y, \mathsf{com}'_y, \pi, \pi'$

[7] In our formal construction there is one more step handling the aggregation of commitments. We gloss over this detail for now.

where $\mathsf{com}_y = \mathsf{Com}(\mathsf{ck}, \mathbf{1}_k)$ and $\mathsf{com}'_y = \mathsf{Com}(\mathsf{ck}, \boldsymbol{b})$. Naturally, this violates projective chain binding as $b_{i^*} = 0$ and hence $\mathsf{Proj}(\mathsf{td}, \mathsf{com}_y) \neq \mathsf{Proj}(\mathsf{td}, \mathsf{com}'_y)$. We note that it is crucial in the projective chain binding definition that the adversary is given the trapdoor td, since the reduction $\mathcal{B}$ above needs it to extract the witness $\mathtt{w}_{i^*}$.

6.2.3 Algebraic PCFC from Pairings

To complete our circuit-succinct algebraic BARG construction, it remains to construct an algebraic PCFC for circuits which is compatible with our compiler. Since explaining details of the construction is tedious and does not help much in gaining intuition, we focus on structural-level ideas and refer to Section 6.7 for the formal details.

Our construction is loosely based on two existing schemes that we introduced earlier in the overview:

1. the implicit PCFC for separable quadratic functions extracted from the Waters-Wu BARG [WW22], and
2. the projective functional commitment (PFC) scheme for circuits by Wee and Wu [WW24b], which is the only existing algebraic FC for circuits with constant size proofs.

Unfortunately, these schemes are not readily available to fulfill the requirements of our compiler. In particular, a major technical challenge is to achieve our projective chain binding property in which the adversary gets access to the projection trapdoor. Notably, this property is stronger than the chain binding of [WW24b] where the adversary only sees the commitment key. Indeed, the Wee and Wu's PFC construction cannot be proven secure if the adversary has the trapdoor.

Melding Waters-Wu's PCFC with Wee-Wu's PFC. The PCFC implicit in Waters-Wu's BARG [WW22] and Wee-Wu's PFC [WW24b] each already satisfy some of our requirements. In particular:

- Waters-Wu's PCFC supports chainable functional proofs for separable quadratic functions and allows to program the commitment key with a singleton set $I = \{i^*\}$ at which the committed message is extractable (given that it belongs to the projective subspace, which in their case is the boolean set $\bar{\mathcal{M}} = \{0, 1\}$).
- Wee-Wu's PFC supports non-chainable[8] functional proofs for arbitrary circuits and allows for projecting commitments on prefix sets $I = \{1, 2, \ldots, i^*\}$. The functional proofs are built from several specialized proof sub-systems that prove linear and quadratic relations between different types of commitments, ensuring consistency over the projective space. However, the scheme is not extractable and cannot achieve projective chain binding.

Conveniently, both schemes are pairing-based and are similarly structured. Our first step is therefore to meld these two schemes in a compatible way. At a high level, we proceed as follows.

[8] The authors discussed chainability but did not formalise the idea.

- We redesign the Waters-Wu's PCFC to embed it with a projective key that allow us to reason about functional binding in the projective subspace. In the original scheme, the size of the commitments is $\kappa + 1$ group elements for a small parameter κ (which parameterizes the κ-Lin assumption). This is sufficient for somewhere extractability, but seems insufficient to reason about functional relations in the projective subspace. We extend it to size 2κ, where, intuitively, we need κ group elements acting on the entire set $[\ell]$ for the normal-mode key and κ group elements acting only on I for the projective key.
- We structure the 2κ-sized key to enable the generation of functional proofs for arbitrary circuits, making it fully compatible with the proof sub-systems that compose the Wee-Wu's PFC.[9]

Overall, we obtain a PCFC which supports functional proofs for arbitrary circuits, projecting commitments onto arbitrary (bounded cardinality) index sets $I \subseteq [\ell]$, and allows extraction at $I \cap J$ for commitments to subvectors given by J as long as $|I \cap J| = 1$. Note that although this PCFC is correct, it does not yet satisfy the projective chain binding property. To satisfy this property, we design a novel functional proof system that partially relies on two of the specialized proof sub-systems in the Wee-Wu's PFC.

Achieving Projective Chain Binding. As mentioned above, the main technical challenge that we tackle is to achieve projective chain binding against adversaries with access to the projection trapdoor td. In more detail, in the reduction from the MDDH assumption to projective chain binding, we need to employ a "sliding window" argument, which is quite common in the BARG literature (e.g. [CJJ21, KLVW23]), where we gradually shift the hidden index set I programmed in the trapdoor td from the input wires to the output wires. However, without modifying the construction, this technique contradicts with the requirement that the adversary is given access to td, since then the adversary can notice that the hidden index set I changes.

To circumvent this issue, our construction introduces three commitment sub-keys $\mathsf{ck} = (\mathsf{ck}_0, \mathsf{ck}_1, \mathsf{ck}_2)$ with their corresponding sub-trapdoors $\mathsf{td}_0, \mathsf{td}_1, \mathsf{td}_2$. The first key ck_0 is the primary commitment key that is trapdoored to be projective at I (and which also enables somewhere extractability). ck_1 and ck_2 are functional keys that correspond to the so-called type-I and type-II commitments, respectively, which are required by the functional proof sub-systems of the Wee-Wu's PFC. The idea is to hand td_0 to the adversary, while td_1 and td_2 are only used in the security proof and never given to the adversary. Then, we can enable the "sliding window" argument only over ck_1 and ck_2, without ever modifying ck_0. To execute this idea, we have to augment the PCFC construction to include additional consistency proofs between commitments generated against the primary key ck_0 and the functional keys $\mathsf{ck}_1, \mathsf{ck}_2$.

[9] In turn, this would enable us to support the functional proofs of the Wee-Wu's PFC and obtain standard CFC evaluation binding, although this is not our end goal.

$$\begin{tikzcd}
\mathsf{com} = \mathsf{Com}(\mathsf{ck}_0, \boldsymbol{x}) \arrow[dr, "\pi_{\mathsf{in}}, p_x"] & & \\
& \mathsf{Com}(\mathsf{ck}_2, \boldsymbol{z}) \arrow[r, bend left, "\pi_{\mathsf{Quad}}, w"] \arrow[dl, "\pi_{\mathsf{out}}, p_y"] & \mathsf{Com}(\mathsf{ck}_1, \boldsymbol{z}) \arrow[l, bend left, "\pi_{\mathsf{Lin}}, id"] \\
\mathsf{com}_y = \mathsf{Com}(\mathsf{ck}_0, \boldsymbol{y}) & &
\end{tikzcd}$$

Figure 6.1: *Overview of the sub-proofs and commitments involved in the PCFC functional proofs.*

Constructing our Functional Proofs. In Figure 6.1 we give an overview of the functional proofs and commitments involved in our PCFC construction. Recall that the end goal is to prove the evaluation of a function f (interpreted as an arithmetic circuit) given an input commitment com and an output commitment com_y, both of which are committed under ck_0. For the description below, we let $\boldsymbol{z} \in \mathcal{M}^{|f|}$ be the computation trace of $f(\boldsymbol{x})$. This is, $\boldsymbol{z}$ contains the values of the $|f|$ circuit wires when f is evaluated on $\boldsymbol{x}$. For simplicity, we omit details related to f being a separable function that evaluates multiple copies of a circuit, and refer only to the single-circuit case. The proof contains four main components:

- A linear map sub-proof π_{in} that proves that a linear function s holds between $\mathsf{com} = \mathsf{Com}(\mathsf{ck}_0, \boldsymbol{x})$ and a type-II commitment $\mathsf{Com}(\mathsf{ck}_2, \boldsymbol{z})$. The map s is a projection that maps the first ℓ coordinates of the input (i.e., $\boldsymbol{x}$) to the first ℓ coordinates of the output (i.e., the prefix of $\boldsymbol{z}$ corresponding to the input).

- A linear map sub-proof π_{Lin} for the identity function id between the type-I commitment $\mathsf{Com}(\mathsf{ck}_1, \boldsymbol{z})$ and the type-II commitment $\mathsf{Com}(\mathsf{ck}_2, \boldsymbol{z})$.

- A quadratic map sub-proof π_{Quad} for the *next wire* function w, which given the first t coordinates of $\boldsymbol{z}$ (i.e., the first t wires of $f(\boldsymbol{x})$), computes the $t+1$-th gate output wire. This is done between the type-II commitment $\mathsf{Com}(\mathsf{ck}_2, \boldsymbol{z})$ and the type-I commitment $\mathsf{Com}(\mathsf{ck}_1, \boldsymbol{z})$.

- Finally, a sub-proof π_{out} that proves an output-projection map p which maps the last ℓ coordinates of $\boldsymbol{z}$ to the first ℓ coordinates of $\boldsymbol{y}$, which is done between $\mathsf{Com}(\mathsf{ck}_2, \boldsymbol{z})$ and $\mathsf{com}_y = \mathsf{Com}(\mathsf{ck}_0, \boldsymbol{y})$.

At a high level, the proof of projective chain binding goes as follows. The sub-proof π_{in} allows one to ensure matching projections between the input commitment com and the type-II commitment $\mathsf{Com}(\mathsf{ck}_2, \boldsymbol{z})$ on the first ℓ coordinates (i.e., $I = [\ell]$). Then, π_{Lin} and π_{Quad} enable a sliding window argument where I gradually increases from $[\ell]$ to $[|f|]$, ensuring matching projections of their respective commitments. Finally, π_{out} ensures that the projection (onto $[\ell]$) of the output commitment com_y matches the projection over the last ℓ coordinates of the type-II commitment $\mathsf{Com}(\mathsf{ck}_2, \boldsymbol{z})$. We refer to Section 6.7 for the details.

6.2.4 Summary

The techniques introduced above lead to a construction of PCFC over pairing groups that we summarize in the following (simplified) theorem:

Theorem 6.33 (simplified). *Assuming the Ker-DH assumption and the MDDH assumption over bilinear groups, the construction* PCFC *satisfies:*

- *Somewhere extractability for a message space $\bar{\mathcal{M}} = \{0, 1\}$ if the subvector set J and the projective set I satisfy $|I \cap J| = 1$.*
- *Projective chain binding for the class of functions $\mathcal{F}_{k,\ell_C}$ which evaluate up to k parallel circuits C_i, each of size at most ℓ_C.*
- *The commitment key size and the worst-case prover running time are $O\left(\lambda \cdot k^5 \cdot \ell_C^5\right)$.*
- *Commitments and proofs have size $O(\lambda)$.*
- *Efficient verification runs in time $O(\lambda)$.*

Overall, by combining the PCFC with the circuit-succinct compiler, we obtain a fully algebraic construction of BARGs from bilinear pairings, which we summarize as follows:

Corollary 6.34. *Under the assumptions of Theorem 6.33, there exists an algebraic somewhere extractable BARG for NP such that:*

- *The setup size $|\mathsf{crs}|$ and the prover running time are $O\left(\lambda \cdot k^5 \cdot \ell_C^5\right)$.*
- *The proof size is $|\pi| = O(\lambda \cdot |\mathtt{w}| \cdot)$.*
- *Efficient verification runs in time $O(\lambda \cdot |\mathtt{w}|)$.*

Our BARG construction is almost concretely efficient, in the sense that all complexity measures are explicitly expressible. In particular, the resulting online verification time and proof size are $O(\lambda \cdot |\mathtt{w}|)$ with small constants. The main efficiency bottleneck that remains is the size of the commitment key, which also impacts the prover time. We therefore see shrinking the commitment key size as one of the next immediate future directions.

6.3 Preliminaries and Circuit Model

6.3.1 Standard Assumptions on Bilinear Groups

We recall the matrix Diffie-Hellman assumption (MDDH) and the kernel Diffie-Hellman assumption (KerDH), which we adapt from [MRV16]. For MDDH we directly introduce the bilateral variant where the challenge is encoded in both $\mathbb{G}_1$ and $\mathbb{G}_2$, following the notation from [GZ21, WW24b].

Definition 6.1 (Kernel Diffie-Hellman Assumption). *Let* $\mathsf{bgp} = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, [1]_1, [1]_2)$ *be a bilinear group setting and let $\kappa, \ell, d \in \mathbb{N}$. We say that the kernel Diffie-Hellman assumption*

$\mathsf{KerDH}_{\kappa,\ell}$ *holds in* $\mathbb{G}_1$ *for* bgp *if for any PPT adversary* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that*

$$\Pr\left[\mathbf{A}\boldsymbol{x} = 0 \wedge \boldsymbol{x} \neq 0 \;\middle|\; \begin{array}{l}\mathbf{A} \leftarrow\!\!\$\; \mathbb{F}_q^{\kappa\times\ell} \\ [\boldsymbol{x}]_2 \leftarrow \mathcal{A}(\mathsf{bgp}, [\mathbf{A}]_1)\end{array}\right] = \mathsf{negl}(\lambda).$$

where the probability is taken over the choice of $\mathbf{A} \leftarrow\!\!\$\; \mathbb{F}_q^{\kappa\times\ell}$ *and the adversary* $\mathcal{A}$*'s random coins. We define* $\mathsf{KerDH}_{\kappa,\ell}$ *over* $\mathbb{G}_2$ *analogously.*

Definition 6.2 (Matrix Diffie-Hellman Assumption). *Let* $\mathsf{bgp} = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, [1]_1, [1]_2)$ *be a bilinear group setting and let* $\kappa, \ell, d \in \mathbb{N}$*. We say that the (bilateral) Matrix Diffie-Hellman assumption* $\mathsf{MDDH}_{\kappa,\ell,d}$ *holds over* bgp *if for any PPT adversary* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that*

$$\left|\begin{array}{l}\Pr\left[\mathcal{A}(\mathsf{bgp}, [\mathbf{A}]_1, [\mathbf{A}]_2, [\mathbf{S}\cdot\mathbf{A}]_1, [\mathbf{S}\cdot\mathbf{A}]_2) \to 1\right] \\ -\Pr\left[\mathcal{A}(\mathsf{bgp}, [\mathbf{A}]_1, [\mathbf{A}]_2, [\mathbf{U}]_1, [\mathbf{U}]_2) \to 1\right]\end{array}\right| \leq \mathsf{negl}(\lambda),$$

where the probability is taken over the choice of $\mathbf{A} \leftarrow\!\!\$\; \mathbb{F}_q^{\kappa\times\ell}, \mathbf{S} \leftarrow\!\!\$\; \mathbb{F}_q^{d\times\kappa}, \mathbf{U} \leftarrow\!\!\$\; \mathbb{F}_q^{d\times\ell}$ *and the adversary* $\mathcal{A}$*'s random coins.*

6.3.2 Batch Arguments for NP

A Batch Argument for NP (BARG) is a proof system for a particular subclass of NP, which is the conjunction of k NP statements corresponding to the same NP language. More precisely, given an NP language $\mathcal{L}$, the prover attests that k statements $\mathtt{x}_1, \ldots, \mathtt{x}_k$ belong to $\mathcal{L}$. In terms of efficiency, BARGs produce proofs π whose size is sublinear in the number of instances k. As opposed to SNARGs, the size of a BARG proof is only required to be sublinear in k, but could be linear in the size of the circuit $C(\mathtt{x}_i, \mathtt{w}_i)$ that decides the language given an input $\mathtt{x}_i$ and an NP witness $\mathtt{w}_i$.

We adopt the notion of (somewhere extractable) BARGs for circuit satisfiability from [CJJ22, KLVW23], where we generically consider circuits $C : \mathcal{M}^{\ell_{\mathtt{x},\mathtt{w}}} \to \mathcal{M}$. Such BARGs have two security properties. The first is setup indistinguishability: one can sample a trapdoored BARG crs at some index i^*, but this should be indistinguishable from a non-trapdoored crs. The second is somewhere extractability: given a valid proof for $\mathtt{x}_1, \ldots, \mathtt{x}_k$ on a crs trapdoored at i^*, the trapdoor allows one to extract a valid witness $\mathtt{w}^*$ such that $C(\mathtt{x}_{i^*}, \mathtt{w}^*) = 1$.

Definition 6.3 (BARG for NP [CJJ22, KLVW23]). *Let statement length* $\ell_{\mathtt{x}} = \mathsf{poly}(\lambda)$ *and witness length* $\ell_{\mathtt{w}} = \mathsf{poly}(\lambda)$ *be fixed. A somewhere extractable batch argument (BARG) for NP is a tuple of algorithms* $\mathsf{BARG} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Ver}, \mathsf{SetupTd}, \mathsf{Ext})$*:*

$\mathsf{Setup}(1^\lambda, 1^k, 1^{\ell_C}) \to \mathsf{crs}$: *on input the security parameter* λ*, a number of instances* k*, a circuit size* ℓ_C*, a statement length* $\ell_{\mathtt{x}}$*, and a witness length* $\ell_{\mathtt{w}}$*, outputs a common reference string* crs*. Below, we implicitly assume that circuits* $C : \mathcal{M}^{\ell_{\mathtt{x},\mathtt{w}}} \to \mathcal{M}$ *are of size* ℓ_C*, statements* $\mathtt{x} \in \mathcal{M}^{\ell_{\mathtt{x}}}$*, and witnesses* $\mathtt{w} \in \mathcal{M}^{\ell_{\mathtt{w}}}$*.*

$\mathsf{Prove}(\mathsf{crs}, C, \{(\mathtt{x}_i, \mathtt{w}_i)\}_{i\in[k]}) \to \pi$: *on input the common reference string* crs*, a circuit* C*, and a batch of input-witness pairs* $\{(\mathtt{x}_i, \mathtt{w}_i)\}_{i\in[k]}$*, outputs a proof* π*.*

$\mathsf{Ver}(\mathsf{crs}, C, \{\mathtt{x}_i\}_{i\in[k]}, \pi) \to b$: *on input* crs, *a circuit C of size ℓ_C, a batch of statements $\{\mathtt{x}_i\}_{i\in[k]}$, and a proof π, accepts ($b = 1$) or rejects ($b = 0$).*

$\mathsf{SetupTd}(1^\lambda, 1^k, 1^{\ell_C}, i^*) \to (\mathsf{crs}, \mathsf{td})$ *works as* Setup, *and additionally outputs a trapdoor* td *associated to the index i^* that is given as input.*

$\mathsf{Ext}(\mathsf{td}, C, \{\mathtt{x}_i\}_{i\in[k]}, \pi) \to \mathtt{w}^*$ *On input a trapdoor* td, *a circuit C, a batch of statements $\{\mathtt{x}_i\}_{i\in[k]}$ and a proof π, outputs a witness $\mathtt{w}^*$ corresponding to the position specified by the trapdoor.*

A BARG must satisfy the following properties:

Completeness. *For any $\lambda, k, \ell_C \in \mathbb{N}$, any circuit C, all statement-witness pairs $(\mathtt{x}_i, \mathtt{w}_i)_{i\in[k]}$ such that $C(\mathtt{x}_i, \mathtt{w}_i) = 1$ for all $i \in [k]$, $\mathsf{crs} \in \mathsf{Setup}(1^\lambda, 1^k, 1^{\ell_C})$ and $\pi \in \mathsf{Prove}(\mathsf{crs}, C, \{(\mathtt{x}_i, \mathtt{w}_i)\}_{i\in[k]})$, it holds that $\mathsf{Ver}(\mathsf{crs}, C, \{\mathtt{x}_i\}_{i\in[k]}, \pi) = 1$.*

Succinctness. *For any admissible set of parameters as before, $|\pi| \leq \mathsf{poly}(\lambda, \log k, \ell_C)$. Furthermore, we say that* BARG *is* circuit-succinct *if $|\pi| \leq \mathsf{poly}(\lambda, \log k, \ell_{\mathtt{w}})$.*

Setup Indistinguishability. *For any PPT adversary $\mathcal{A}$, any $k, \ell_C = \mathsf{poly}(\lambda)$, and index $i^* \in [k]$, the distributions of* crs *induced by $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{SetupTd}(1^\lambda, 1^k, 1^{\ell_C}, i^*)$ and $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^k, 1^{\ell_C})$ are computationally indistinguishable.*

Somewhere Extractability. *For any PPT adversary $\mathcal{A}$, any $k, \ell_C = \mathsf{poly}(\lambda)$, and index $i^* \in [k]$,*

$$\Pr\left[\begin{array}{c|l} \mathsf{Ver}(\mathsf{crs}, C, \{\mathtt{x}_i\}_{i\in[k]}, \pi) = 1 & (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{SetupTd}(1^\lambda, 1^k, 1^{\ell_C}, i^*) \\ \wedge\, C(\mathtt{x}_{i^*}, \mathtt{w}^*) \neq 1 & (C, \{\mathtt{x}_i\}_{i\in[k]}, \pi) \leftarrow \mathcal{A}(\mathsf{crs}) \\ & \mathtt{w}^* \leftarrow \mathsf{Ext}(\mathsf{td}, \{\mathtt{x}_i\}_{i\in[k]}, C, \pi) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

Remark 6.4. *One can also consider parametric notions of succinctness for BARGs following a similar notation as for functional commitments in Theorem 4.1, where there exists a function s_{BARG} such that $|\pi| \leq s_{\mathsf{BARG}}(\lambda, k, \ell_C)$. We will use this notation in Chapter 7.*

Definition 6.5 (Efficient Verification for BARG). *A BARG for NP* BARG *has efficient verification with preprocessing (also called split verification in [WW22]) if there exists a pair of algorithms:*

$\mathsf{PreVer}(\mathsf{crs}, C, \{\mathtt{x}_i\}_{i\in[k]}) \to \mathsf{vk}$: *On input the common reference string* crs, *a circuit C, and a batch of k statements $\{\mathtt{x}_i\}_{i\in[k]}$, it creates a succinct verification key* vk *such that $|\mathsf{vk}| \leq \mathsf{poly}(\lambda, \log k, \ell_C)$*

$\mathsf{EffVer}(\mathsf{vk}, \pi) \to b$ *On input the succinct verification key* vk *and a proof π, accepts ($b = 1$) or rejects ($b = 0$) in time bounded by $\mathsf{poly}(\lambda, \log k, \ell_C)$.*

Furthermore, if for $\ell_C \leq 2^{\mathsf{poly}(\lambda)}$ the size of the verification key $|\mathsf{vk}|$ and the runtime of $\mathsf{EffVer}(\mathsf{vk}, \pi)$ are bounded by $\mathsf{poly}(\lambda, \log k, \ell_{\mathtt{w}})$, we say that BARG *has a circuit-succinct verification with preprocessing.*

Remark 6.6. *In Theorem 6.5 we consider a stronger notion of efficient verification where the circuit C can be preprocessed by* PreVer. *This allows us to define the circuit-succinct verifier where the verifier runtime is sublinear in $|C|$. In prior definitions (e.g. [WW22]), the circuit C is not preprocessed by* PreVer *but is instead passed as an input to* EffVer. *In this case, it is impossible for the online verifier to have circuit-succinctness.*

6.3.3 Circuit model

Throughout this work, we model circuits $C : \mathcal{M}^{\ell_{\mathtt{x},\mathtt{w}}} \to \mathcal{M}$, where $\ell_{\mathtt{x},\mathtt{w}} = \ell_{\mathtt{x}} + \ell_{\mathtt{w}}$, to take as input a statement $\mathtt{x} \in \mathcal{M}^{\ell_{\mathtt{x}}}$ and a witness $\mathtt{w} \in \mathcal{M}^{\ell_{\mathtt{w}}}$, for some message space $\mathcal{M}$ of cardinality $|\mathcal{M}| \leq 2^{\mathsf{poly}(\lambda)}$. A circuit can be arithmetic (e.g. $\mathcal{M} = \mathbb{Z}_q$) or Boolean (i.e. $\mathcal{M} = \{0, 1\}$). A circuit supports arbitrary fan-in and fan-out arithmetic (resp. Boolean) gates over $\mathcal{M}$. We adopt the convention that $1 \in \mathcal{M}$ and we say that an input $(\mathtt{x}, \mathtt{w})$ is accepted by C if $C(\mathtt{x}, \mathtt{w}) = 1$.

We measure the size of a circuit by its total number of wires ℓ_C, which includes the statement and witness input wires, as well as the output wire. We associate each wire to the output of a gate $g \in [\ell_C]$. For notational convenience, and such that the number of gates and that of wires are both ℓ_C, we consider that all inputs come from an "input gate" $g \in [\ell_{\mathtt{x},\mathtt{w}}]$ which simply outputs the hardwired g-th input. Given a circuit computation $C(\mathtt{x}, \mathtt{w})$, the value of the g-th wire (or the output of the g-th gate) is denoted by x_g. Each gate $g \in [\ell_C]$ computes a function $\chi_g : \mathcal{M}^{|T_g|} \to \mathcal{M}$ on an arbitrary set of input *wires* denoted by T_g. Each gate has a single output wire, even if this can be input to multiple other gates. Finally, note that χ_g is a quadratic map, as it represents a single gate computation on a Boolean or arithmetic circuit.

To be descriptive, we use $\mathcal{G} = [\ell_C] = \mathcal{G}_{\mathsf{stmt}} \cup \mathcal{G}_{\mathsf{wit}} \cup \mathcal{G}_{\mathsf{out}}$ to denote the set of all gates/wires partitioned into statement, witness, internal and output wires respectively. More precisely, we enumerate the circuit gates and wires as follows:

- $\mathcal{G}_{\mathsf{stmt}} := \{1, \ldots, \ell_{\mathtt{x}}\}$ are the input wires corresponding to the values in $\mathtt{x}$.
- $\mathcal{G}_{\mathsf{wit}} := \{\ell_{\mathtt{x}} + 1, \ldots, \ell_{\mathtt{x},\mathtt{w}}\}$ are the input wires corresponding to the values in $\mathtt{w}$.
- $\mathcal{G}_{\mathsf{out}} := \{\ell_{\mathtt{x},\mathtt{w}} + 1, \ldots, \ell_C\}$ correspond to the output wires of all (non-input) gates. We arrange these gates that the g-th gate only takes input wires corresponding to smaller indices, namely $T_g \subseteq [g - 1]$.

We note that the subscripts in ℓ_C, $\ell_{\mathtt{x}}$, $\ell_{\mathtt{w}}$, $\mathcal{G}_{\mathsf{stmt}}$, $\mathcal{G}_{\mathsf{wit}}$, $\mathcal{G}_{\mathsf{out}}$, etc. are symbolic, e.g. ℓ_C should not be thought of as a function of a circuit C. We sometimes abuse notation and write $\ell_C = |C|$, $\ell_{\mathtt{x}} = |\mathtt{x}|$, $\ell_{\mathtt{w}} = |\mathtt{w}|$, etc. to denote the quantities.

6.4 Projective Chainable Functional Commitments

We define several variants of commitments, in particular projective chainable functional commitments (PCFC), and their properties. Recall from Theorem 3.1 that we allow the commitment scheme algorithms to input an index set $J \in \mathcal{J} \subseteq 2^{[\ell]}$ to refer to subvectors $\boldsymbol{x} \in \mathcal{M}^J$.

6.4.1 Projective Commitments (PC)

In [WW24b], Wee and Wu introduce the notion of projective commitments. We generalise their notion to allow for arbitrary projective subspaces, but not only prefixes, along with the flexibility of committing to subvectors. We adopt the notation that $\mathcal{I}$ is a family of admissible index sets $I \in \mathcal{I} \in 2^{[\ell]}$. We write $\boldsymbol{x}_I$ to refer to a subvector of $\mathcal{M}^\ell$ that contains only the coordinates in I, namely $\boldsymbol{x}_I \in \mathcal{M}^I$.

Definition 6.7 (Projective Commitments (PC)). *Let $\mathcal{I}, \mathcal{J}$ be families of index sets parametrised by $\ell \in \mathbb{N}$. A projective commitment (PC) scheme for $(\mathcal{I}, \mathcal{J})$ consists of a commitment scheme* $(\mathsf{Setup}, \mathsf{Com})$ *for $\mathcal{J}$ and additional PPT algorithms* $(\mathsf{ProjSetup}, \mathsf{ProjCom}, \mathsf{Proj})$ *with the following syntax:*

$\mathsf{ProjSetup}(1^\lambda, 1^\ell, I) \to (\mathsf{ck}, \mathsf{td})$: *On input the security parameter λ, vector length ℓ, and index set $I \in \mathcal{I}$, output a commitment key* ck *and a trapdoor* td.

$\mathsf{ProjCom}(\mathsf{td}, J, \boldsymbol{x}) \to \mathsf{pcom}$: *On input the trapdoor* td, *an index set $J \in \mathcal{J}$ and a vector $\boldsymbol{x} \in \mathcal{M}^{I \cap J}$, output a projective commitment* pcom.

$\mathsf{Proj}(\mathsf{td}, \mathsf{com}) \to \mathsf{pcom}$: *On input the trapdoor* td *and a commitment* com, *output a projective commitment* pcom.[10]

The index set J is taken as $J = [\ell]$ when omitted. A PC must satisfy the following properties:

Projection Correctness. *For any index sets $I \in \mathcal{I}$, $J \in \mathcal{J}$, any vector $\boldsymbol{x} \in \mathcal{M}^{I \cap J}$, and any* $(\mathsf{ck}, \mathsf{td}) \leftarrow \mathsf{ProjSetup}(1^\lambda, 1^\ell, I)$, *then* $\Pr[\mathsf{Proj}(\mathsf{td}, \mathsf{Com}(\mathsf{ck}, J, \boldsymbol{x})) = \mathsf{ProjCom}(\mathsf{td}, J, \boldsymbol{x})] = 1$.

Setup Indistinguishability. *For any PPT adversary $\mathcal{A}$, and subspace $I \in \mathcal{I}$, the distributions of* ck *induced by* $(\mathsf{ck}, \mathsf{td}) \leftarrow \mathsf{ProjSetup}(1^\lambda, 1^\ell, I)$ *and* $\mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell)$ *are computationally indistiguishable.*

Extending the notion of aggregatability, we define the notion of projective aggregatability which states that not only ordinary commitments but also projective commitments are aggregatable.

Definition 6.8 (PC Projective Aggregatability). *A PC* $(\mathsf{Setup}, \mathsf{Com}, \mathsf{ProjSetup}, \mathsf{ProjCom}, \mathsf{Proj})$ *for $(\mathcal{I}, \mathcal{J})$ is projective aggregatable if, given $\hat{\mathcal{J}} \subset \mathcal{J}$,*

- $(\mathsf{Setup}, \mathsf{Com})$ *is aggregatable with an aggregation algorithm* Agg,
- *there exists a projective aggregation algorithm* $\mathsf{ProjAgg}(\mathsf{ck}, (\mathsf{pcom}_J)_{J \in \hat{\mathcal{J}}}) \to \mathsf{pcom}$ *inputting the commitment key* ck *and tuples of projective commitments* $(\mathsf{pcom}_J)_{J \in \hat{\mathcal{J}}}$ *and outputs a projective commitment* pcom,
- *if $J \cap J' = \emptyset$ for all distinct $J, J' \in \hat{\mathcal{J}}$, then for any $I \in \mathcal{I}$ and any $\boldsymbol{x}_J \in \mathcal{M}^{I \cap J}$ for $J \in \hat{\mathcal{J}}$, the following property is satisfied:*

$$\Pr\left[\mathsf{ProjAgg}(\mathsf{ck}, (\mathsf{pcom}_J)_{J \in \hat{\mathcal{J}}}) = \mathsf{pcom} \;\middle|\; \begin{array}{l} (\mathsf{ck}, \mathsf{td}) \leftarrow \mathsf{ProjSetup}(1^\lambda, 1^\ell, I) \\ \mathsf{pcom}_J \leftarrow \mathsf{ProjCom}(\mathsf{td}, J, \boldsymbol{x}_J) \;\; \forall J \in \hat{\mathcal{J}} \\ \mathsf{pcom} \leftarrow \mathsf{ProjCom}\left(\mathsf{ck}, \bigcup_{J \in \hat{\mathcal{J}}} J, \bigcup_{J \in \hat{\mathcal{J}}} \boldsymbol{x}_J\right) \end{array}\right] = 1.$$

[10]Note that we define Proj not to input the (alleged) index set J of the commitment com. This leads to stronger properties related to projection. Clearly, the syntax and properties can be relaxed to allow Proj input J.

Remark 6.9 (Projection commutativity)**.** *Projection correctness (Theorem 6.7) and projective aggregatability (Theorem 6.8), which implies aggregatability (Theorem 3.5), together imply "projection commutativity". In more detail, suppose*

$$\begin{aligned} &\mathsf{com}_J = \mathsf{Com}(\mathsf{ck}, J, \boldsymbol{x}_J), && \mathsf{pcom}_J = \mathsf{ProjCom}(\mathsf{td}, J, \boldsymbol{x}_J), \\ &\mathsf{com} = \mathsf{Com}\left(\mathsf{ck}, \textstyle\bigcup_{J \in \hat{\mathcal{J}}} J, \bigcup_{J \in \hat{\mathcal{J}}} \boldsymbol{x}_J\right), \quad \text{and} && \mathsf{pcom} = \mathsf{ProjCom}\left(\mathsf{td}, \textstyle\bigcup_{J \in \hat{\mathcal{J}}} J, \bigcup_{J \in \hat{\mathcal{J}}} \boldsymbol{x}_J\right). \end{aligned}$$

The above three properties imply the following:

- *projection correctness:* $\mathsf{Proj}(\mathsf{td}, \mathsf{com}_J) = \mathsf{pcom}_J$ *and* $\mathsf{Proj}(\mathsf{td}, \mathsf{com}) = \mathsf{pcom}$.
- *aggregatability:* $\mathsf{Agg}(\mathsf{ck}, (\mathsf{com}_J)_{J \in \hat{\mathcal{J}}}) = \mathsf{com}$.
- *projective aggregatability:* $\mathsf{ProjAgg}(\mathsf{ck}, (\mathsf{pcom}_J)_{J \in \hat{\mathcal{J}}}) = \mathsf{pcom}$.

Putting the above together, we obtain

$$\mathsf{ProjAgg}(\mathsf{ck}, (\mathsf{Proj}(\mathsf{td}, \mathsf{com}_J))_{J \in \hat{\mathcal{J}}}) = \mathsf{Proj}(\mathsf{td}, \mathsf{Agg}(\mathsf{ck}, (\mathsf{com}_J)_{J \in \hat{\mathcal{J}}})).$$

For certain message subspaces $\bar{\mathcal{M}} \subset \mathcal{M}$ and vectors $\boldsymbol{x} \in \bar{\mathcal{M}}^\ell$, it might be possible to augment a projective commitment scheme with an embedded proof system that provides somewhere extractability. It means that, if the commitment key is extractable at set I and an adversary provides a commitment com and a proof π that verifies, then one can extract a subvector $\boldsymbol{x}_I \in \bar{\mathcal{M}}^I$ such that $\mathsf{Proj}(\mathsf{td}, \mathsf{com}) = \mathsf{ProjCom}(\mathsf{td}, \boldsymbol{x}_I)$ except with negligible probability.

To illustrate this property, consider the example of a commitment scheme such that the message space $\mathcal{M} = \mathbb{F}_q$ for a prime $q > 2$. Moreover, the commitment supports an embedded "booleanity" proof that ensures that a committed value $\boldsymbol{x}$ satisfies $x_i(x_i - 1) = 0\ \forall i \in [\ell]$, namely that $x_i \in \{0, 1\} = \bar{\mathcal{M}}$. Then, our somewhere extractability notion ensures that we can extract a vector $\boldsymbol{x}_I \in \{0, 1\}^I$.

Definition 6.10 (PC Somewhere Extractability)**.** *Let $\mathcal{I}, \mathcal{J}$ be index sets families parametrised by $\ell \in \mathbb{N}$ and $\bar{\mathcal{M}} \subset \mathcal{M}$ be a message subspace. A somewhere extractable PC for $(\mathcal{I}, \mathcal{J}, \bar{\mathcal{M}})$ consists of a PC for $(\mathcal{I}, \mathcal{J})$ and additional PPT algorithms* $(\mathsf{ProjProve}, \mathsf{ProjVer}, \mathsf{ProjExt})$ *with the following syntax:*

$\mathsf{ProjProve}(\mathsf{ck}, J, \boldsymbol{x}) \to \pi$**:** *On input the commitment key* ck*, an index set $J \in \mathcal{J}$ and a vector $\boldsymbol{x} \in \mathcal{M}^J$, output a subspace proof π.*

$\mathsf{ProjVer}(\mathsf{ck}, J, \mathsf{com}, \pi) \to 0/1$**:** *On input the commitment key* ck*, an index set $J \in \mathcal{J}$, a commitment* com *and a subspace proof π, accepts or rejects.*

$\mathsf{ProjExt}(\mathsf{td}, J, \mathsf{com}) \to \boldsymbol{x}$**:** *On input the trapdoor* td*, an index set $J \in \mathcal{J}$ and a commitment* com*, outputs a value $\boldsymbol{x} \in \bar{\mathcal{M}}^{I \cap J} \cup \{\bot\}$.*

Moreover, these algorithms must satisfy the following properties:

Succinctness. *For any honestly generated commitment key* ck*, index set $J \in \mathcal{J}$ and vector $\boldsymbol{x} \in \mathcal{M}^J$, a projection proof $\pi \in \mathsf{ProjProve}(\mathsf{ck}, J, \boldsymbol{x})$ satisfies $|\pi| \le \mathsf{poly}(\lambda, \log \ell, \log |J|)$.*

Somewhere Completeness. *For any index sets $I \in \mathcal{I}$ and $J \in \mathcal{J}$ and any vector $\boldsymbol{x} \in \mathcal{M}^J$:*

$$\Pr\left[\mathsf{ProjVer}(\mathsf{ck}, J, \mathsf{com}, \pi) = 1 \,\middle|\, \begin{array}{l} (\mathsf{ck}, \mathsf{td}) \leftarrow \mathsf{ProjSetup}(1^\lambda, 1^\ell, I) \\ \mathsf{com} \leftarrow \mathsf{Com}(\mathsf{ck}, J, \boldsymbol{x}) \\ \pi \leftarrow \mathsf{ProjProve}(\mathsf{ck}, J, \boldsymbol{x}) \end{array} \right] = 1.$$

Somewhere Extractability. *For any PPT*[11] *adversary $\mathcal{A}$ and index set $I \in \mathcal{I}$, the advantage $\mathsf{Adv}^{\mathrm{se}}_{\mathsf{PCFC},\mathcal{A}}(\lambda)$ defined below is $\mathsf{negl}(\lambda)$:*

$$\Pr\left[\begin{array}{l} \mathsf{ProjVer}(\mathsf{ck}, J, \mathsf{com}, \pi) = 1 \\ \wedge\ \big(\mathsf{ProjCom}(\mathsf{td}, J, \boldsymbol{x}) \neq \mathsf{Proj}(\mathsf{td}, \mathsf{com}) \vee \boldsymbol{x} \notin \bar{\mathcal{M}}^{I \cap J}\big) \end{array} \,\middle|\, \begin{array}{l} (\mathsf{ck}, \mathsf{td}) \leftarrow \mathsf{ProjSetup}(1^\lambda, 1^\ell, I) \\ (J, \mathsf{com}, \pi) \leftarrow \mathcal{A}(\mathsf{ck}) \\ \boldsymbol{x} \leftarrow \mathsf{ProjExt}(\mathsf{td}, J, \mathsf{com}) \\ J \in \mathcal{J} \end{array} \right].$$

Remark 6.11 (Somewhere extractability for aggregated commitments.). *Naturally, the above definition of somewhere extractability (Theorem 6.10) can be extended to aggregated commitments. If $\mathsf{com} = \mathsf{Agg}(\mathsf{ck}, (\mathsf{com}_J)_{J \in \hat{\mathcal{J}}})$ for some non-overlapping family $\hat{\mathcal{J}} \subset \mathcal{J}$, then it follows that $\mathsf{ProjExt}(\mathsf{td}, \bigcup_{J' \in \hat{\mathcal{J}}} J, \mathsf{com}) = \bigcup_{J \in \hat{\mathcal{J}}} \mathsf{ProjExt}(\mathsf{td}, J', \mathsf{com}_J)$. This may enable extraction on aggregated commitments com where running the extractor $\mathsf{ProjExt}$ on com directly is not possible, such as when $I \cap J$ is too large.*

Definition 6.12 (PC Efficient Verification). *A somewhere extractable PC admits efficient verification with preprocessing if there exists a pair of algorithms:*

$\mathsf{PreProjVer}(\mathsf{ck}, J) \to \mathsf{ck}_J$ *on input the commitment key ck and an index set $J \in \mathcal{J}$, outputs a projective key ck_J of size $|\mathsf{ck}_J| \leq \mathsf{poly}(\lambda, \log \ell, \log |J|)$.*

$\mathsf{EffProjVer}(\mathsf{ck}_J, \mathsf{com}, \pi) \to b \in \{0, 1\}$ *on input a projective key ck_J, a commitment com, and a projection proof π, accepts ($b = 1$) or rejects ($b = 0$) in time bounded by $\mathsf{poly}(\lambda, \log \ell, \log |J|)$.*

Furthermore, the following function equivalence holds:

$$\mathsf{ProjVer}(\mathsf{ck}, J, \mathsf{com}, \pi) \equiv \mathsf{EffProjVer}(\mathsf{PreProjVer}(\mathsf{ck}, J), \mathsf{com}, \pi).$$

6.4.2 Projective Chainable Functional Commitments (PCFC)

We consider projective commitments that also admit the syntax of chainable functional commitments (Theorem 4.8). We call these commitments *projective chainable functional commitments* (PCFC).

Definition 6.13 (Projective Chainable Functional Commitment (PCFC)). *A (somewhere extractable) projective CFC ((se-)PCFC) for a class of functions $\mathcal{F}$, families of admissible index*

[11]The property may also hold statistically against unbounded adversaries.

sets $(\mathcal{I}, \mathcal{J})$ (and a message subspace $\bar{\mathcal{M}}$) is a tuple of PPT algorithms consisting of some shared $\mathsf{Setup}, \mathsf{Com}$ *and*

$$\overbrace{(\underbrace{\mathsf{ProjSetup}, \mathsf{ProjCom}, \mathsf{Proj}}_{\textit{PC for } (\mathcal{I},\mathcal{J})}, (\mathsf{ProjProve}, \mathsf{ProjVer}, \mathsf{ProjExt})}^{\textit{se-PC for } (\mathcal{I},\mathcal{J},\bar{\mathcal{M}})}, \underbrace{\mathsf{FuncProve}, \mathsf{FuncVer}}_{\textit{CFC for } (\mathcal{F},\mathcal{J})}).$$

Definition 6.14 (PCFC Efficient Verification). *A PCFC admits efficient verification with preprocessing if the underlying CFC and PC admit efficient verification with preprocessing, i.e. there exists additional algorithms* $(\mathsf{PreFuncVer}, \mathsf{EffFuncVer}, \mathsf{PreProjVer}, \mathsf{EffProjVer})$ *with properties defined in Theorems 4.10 and 6.12.*

We introduce the two main security notions for PCFC: functional extractability and projective chain binding. To define these notions, we first introduce separable functions.

Definition 6.15 (Separable Function). *Let $I_1, \ldots, I_\ell, I_y \subseteq [\ell]$. We say that a function $f : \prod_{i \in [m]} \mathcal{M}^{J_i} \to \mathcal{M}^{J_y}$ is $I = (I_1, \ldots, I_\ell, I_y)$-separable whenever the coordinates in $I_y \cap J_y$ of the output vector only depend on the coordinates in $I_i \cap J_i$ of each of the i-th input vectors. The restriction of f to I is denoted by $f_I : \prod_{i \in [m]} \mathcal{M}^{I_i \cap J_i} \to \mathcal{M}^{I_y \cap J_y}$. For a class of functions $\mathcal{F}$, we write $\mathcal{F}_I$ for the set of functions $f \in \mathcal{F}$ that are I-separable. If f is I-separable for all $I \in \mathcal{I}^{\ell+1}$ for some set $\mathcal{I}$, we say that f is $\mathcal{I}$-separable. If $I_1 = \ldots = I_\ell = I_y$, we abuse notation and write $I = I_1 = \ldots = I_\ell = I_y$.*

Example 6.16. *The function $f : \mathbb{Z}^2 \times \mathbb{Z}^2 \to \mathbb{Z}^2$ given by $f((x_1, x_2), (y_1, y_2)) = (x_1 + y_1, x_1 + x_2 + y_2)$ is $\mathcal{I}$-separable for $\mathcal{I} = \{\{1\}, \{1, 2\}\}$.*

Intuitively, a PCFC is functional extractable if, given a CFC opening π for a separable function f, then the extracted values from the commitments are guaranteed to evaluate f_I correctly.

Definition 6.17 (PCFC Functional Extractability). *A somewhere extractable PCFC satisfies functional somewhere extractability for a class of $\mathcal{I}$-separable functions $\mathcal{F}$ and index set families $(\mathcal{I}, \mathcal{J})$ if for any PPT adversary $\mathcal{A}$ and set $I \in \mathcal{I}$, the advantage $\mathsf{Adv}^{\mathsf{fe}}_{\mathsf{PCFC},\mathcal{A}}(\lambda)$ defined below is* $\mathsf{negl}(\lambda)$*:*

$$\Pr\left[\begin{array}{l|l} & (\mathsf{ck}, \mathsf{td}) \leftarrow \mathsf{ProjSetup}(1^\lambda, 1^\ell, I) \\ & (f, (\mathsf{com}_i)_{i\in[m]}, \mathsf{com}_y, \pi) \leftarrow \mathcal{A}(\mathsf{ck}) \\ \mathsf{FuncVer}(\mathsf{ck}, (\mathsf{com}_i)_{i\in[m]}, \mathsf{com}_y, f, \pi) = 1 & f : \prod_{i\in[m]} \mathcal{M}^{J_i} \to \mathcal{M}^{J_y} \\ \wedge\ f_I(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m) \neq \boldsymbol{y} & \boldsymbol{x}_i \leftarrow \mathsf{ProjExt}(\mathsf{td}, J_i, \mathsf{com}_i)\ \ \forall i \in [m] \\ \wedge\ f \in \mathcal{F}_I & \boldsymbol{y} \leftarrow \mathsf{ProjExt}(\mathsf{td}, J_y, \mathsf{com}_y) \\ & \boldsymbol{x}_i \in \mathcal{M}^{I \cap J_i}\ \ \forall i \in [m] \\ & \boldsymbol{y} \in \mathcal{M}^{I \cap J_y} \end{array}\right].$$

Remark 6.18. *Due to the somewhere extractability of PC (Theorem 6.10), an alternative way of defining Functional Extractability for PCFC is to let the adversary output $\boldsymbol{x}_i$ and $\boldsymbol{y}$, and to let the experiment run* Proj *instead of* $\mathsf{ProjExt}$.

Finally, we introduce projective chain binding for PCFC which generalises evaluation binding (Theorem 4.9). In particular, if a PCFC for a class of $\mathcal{I}$-separable functions $\mathcal{F}$ and index set families $(\mathcal{I}, \mathcal{J})$ is projective chain binding, where $\mathcal{I}$ contains $[\ell]$, then the PCFC is also evaluation binding for $\mathcal{F}$.

Definition 6.19 (PCFC Projective Chain Binding)**.** *A PCFC for a class of $\mathcal{I}$-separable functions $\mathcal{F}$ and index set families $(\mathcal{I}, \mathcal{J})$ satisfies projective chain binding if for any PPT adversary $\mathcal{A}$ and set $I \in \mathcal{I}$, the advantage $\mathsf{Adv}^{\mathrm{pcb}}_{\mathsf{PCFC},\mathcal{A}}(\lambda)$ defined below is $\mathsf{negl}(\lambda)$.*

$$\Pr\left[\begin{array}{c|l} & (\mathsf{ck}, \mathsf{td}) \leftarrow \mathsf{ProjSetup}(1^\lambda, 1^\ell, I) \\ \mathsf{Proj}(\mathsf{td}, \mathsf{com}_y) & (f, (\mathsf{com}_i, \mathsf{com}'_i)_{i\in[m]}, \mathsf{com}_y, \mathsf{com}'_y, \pi, \pi') \leftarrow \mathcal{A}(\mathsf{ck}, \mathsf{td}) \\ \neq \mathsf{Proj}(\mathsf{td}, \mathsf{com}'_y) & \mathsf{FuncVer}(\mathsf{ck}, (\mathsf{com}_i)_{i\in[m]}, \mathsf{com}_y, f, \pi) = 1 \\ \wedge\ f \in \mathcal{F}_I & \mathsf{FuncVer}(\mathsf{ck}, (\mathsf{com}'_i)_{i\in[m]}, \mathsf{com}'_y, f, \pi') = 1 \\ & \mathsf{Proj}(\mathsf{td}, \mathsf{com}_i) = \mathsf{Proj}(\mathsf{td}, \mathsf{com}'_i)\ \ \forall i \in [m] \end{array}\right].$$

Note that the adversary not only gets access to the commitment key ck, but also to the projection trapdoor td. One may consider a weaker variant where td is not given to the adversary.

On functional extractability, projective chain binding, and partial input soundness. Partial input soundness [KLVW23] is a security notion that arises in the context of RAM SNARGs, more specifically in *flexible RAM SNARGs*. A flexible RAM SNARG with partial input soundness intuitively guarantees soundness if some part of the RAM memory can be extracted from the hash of the memory tape, as long as the RAM computation depends only on the part that was extracted. The notion is defined with respect to a somewhere extractable hash function that is programmed on a memory part I. Then, partial input soundness is stated roughly as follows: given a somewhere extractable hash key programmed on I, it is hard for any adversary to output a valid SNARG proof π and a hash digest rt such that the RAM machine $\mathcal{R}$ reads only I and the extracted $\boldsymbol{x}_I$ satisfies that $\mathcal{R}(\boldsymbol{x}_I) = 0$.

Functional extractability (Theorem 6.17) and partial input soundness are essentially the same property in different environments, as both notions refer to local computations where a part of the input can be extracted. Their main differences are the following two. First, functional extractability is not defined with respect to any external building block such as a somewhere extractable hash function, as it is an intrinsic property of PCFC schemes. Second, the property is defined for separable functions over circuits, instead of RAM machines that read only a part of their memory tape.

On the other hand, projective chain binding (Theorem 6.19) is a stronger notion than both functional extractability and partial input soundness, as it must hold even if the adversary knows the trapdoor of the scheme. Moreover, projective chain binding holds for projections, which may or may not be (somewhere) extractable. In case the projected input commitments are indeed extractable, it is not hard to see that projective chain binding implies functional extractability.

```
$\mathsf{Setup}(1^\lambda, 1^k, 1^{\ell_C})$
    $\mathsf{ck} \leftarrow \mathsf{PCFC.Setup}(1^\lambda, 1^k)$
    return $\mathsf{crs} := \mathsf{ck}$

$\mathsf{SetupTd}(1^\lambda, 1^k, 1^{\ell_C}, i^*)$
    $(\mathsf{ck}, \mathsf{td}) \leftarrow \mathsf{PCFC.ProjSetup}(1^\lambda, 1^k, i^*)$
    return $(\mathsf{crs}, \mathsf{td}) := (\mathsf{ck}, \mathsf{td})$

$\mathsf{Ext}(\mathsf{td}, C, \{\mathtt{x}_i\}_{i \in [k]}, \pi)$
    $\mathtt{w}_{i^*,g} \leftarrow \mathsf{PCFC.ProjExt}(\mathsf{td}, \mathsf{com}_g) \ \forall g \in \mathcal{G}_{\mathsf{wit}}$
    return $\mathtt{w}_{i^*} := (\mathtt{w}_{i^*,g})_{g \in \mathcal{G}_{\mathsf{wit}}}$

$\mathsf{Prove}(\mathsf{crs}, C, \{(\mathtt{x}_i, \mathtt{w}_i)\}_{i \in [k]})$
    $(\chi_g)_{g \in \mathcal{G}_{\mathsf{out}}} \leftarrow C$
    $x_{i,g} := \chi_g(x_{i,j} : j \in T_g) \ \forall i \in [k], g \in \mathcal{G}_{\mathsf{out}}$
    $\boldsymbol{x}_g := (x_{1,g}, \ldots, x_{k,g}) \ \forall g \in \mathcal{G}_{\mathsf{out}}$
    $\mathsf{com}_g \leftarrow \mathsf{PCFC.Com}(\mathsf{ck}, \boldsymbol{x}_g) \ \forall g \in \mathcal{G}_{\mathsf{wit}} \cup \mathcal{G}_{\mathsf{out}}$
    $\pi_g^b \leftarrow \mathsf{PCFC.ProjProve}(\mathsf{ck}, \boldsymbol{x}_g) \ \forall g \in \mathcal{G}_{\mathsf{wit}}$
    $\pi_g^\chi \leftarrow \mathsf{PCFC.FuncProve}(\mathsf{ck}, (\boldsymbol{x}_t)_{t \in T_g}, f_{\chi_g})$
        $\forall g \in \mathcal{G}_{\mathsf{out}}$
    $\pi := ((\mathsf{com}_g)_{g \in \mathcal{G}_{\mathsf{wit}} \cup \mathcal{G}_{\mathsf{out}}}, (\pi_g^b)_{g \in \mathcal{G}_{\mathsf{wit}}}, (\pi_g^\chi)_{g \in \mathcal{G}_{\mathsf{out}}})$
    return $\pi$

$\mathsf{Ver}(\mathsf{crs}, C, \{\mathtt{x}_i\}_{i \in [k]}, \pi)$
    $(\chi_g)_{g \in \mathcal{G}_{\mathsf{out}}} \leftarrow C$
    $\boldsymbol{x}_g := (x_{1,g}, \ldots, x_{k,g}) \ \forall g \in \mathcal{G}_{\mathsf{stmt}}$
    $\mathsf{com}_g \leftarrow \mathsf{PCFC.Com}(\mathsf{ck}, \boldsymbol{x}_g) \ \forall g \in \mathcal{G}_{\mathsf{stmt}}$
    assert $\mathsf{PCFC.ProjVer}(\mathsf{ck}, \mathsf{com}_g, \pi_g^b) = 1$
        $\forall g \in \mathcal{G}_{\mathsf{wit}}$
    assert $\mathsf{PCFC.FuncVer}(\mathsf{ck}, (\mathsf{com}_i)_{i \in T_g},$
        $\mathsf{com}_g, f_{\chi_g}, \pi_g^\chi) = 1 \ \forall g \in \mathcal{G}_{\mathsf{out}}$
    return 1

$\mathsf{PreVer}(\mathsf{crs}, C, \{\mathtt{x}_i\}_{i \in [k]})$
    $(\chi_g)_{g \in \mathcal{G}_{\mathsf{out}}} \leftarrow C$
    $\boldsymbol{x}_g := (x_{1,g}, \ldots, x_{k,g}) \ \forall g \in \mathcal{G}_{\mathsf{stmt}}$
    $\mathsf{com}_g \leftarrow \mathsf{PCFC.Com}(\mathsf{ck}, \boldsymbol{x}_g) \ \forall g \in \mathcal{G}_{\mathsf{stmt}}$
    $\mathsf{ck}_{[k]} \leftarrow \mathsf{PCFC.PreProjVer}(\mathsf{ck}, [k])$
    $\mathsf{ck}_g \leftarrow \mathsf{PCFC.PreFuncVer}(\mathsf{ck}, f_{\chi_g})$
        $\forall g \in \mathcal{G}_{\mathsf{out}}$
    $\mathsf{vk} := ((\mathsf{com}_g)_{g \in \mathcal{G}_{\mathsf{stmt}}}, \mathsf{ck}_{[k]}, (\mathsf{ck}_g)_{g \in \mathcal{G}_{\mathsf{out}}})$
    return $\mathsf{vk}$

$\mathsf{EffVer}(\mathsf{vk}, \pi)$
    assert $\mathsf{PCFC.EffProjVer}(\mathsf{ck}_{[k]}, \mathsf{com}_g, \pi_g^b)$
        $\forall g \in \mathcal{G}_{\mathsf{wit}}$
    assert $\mathsf{PCFC.EffFuncVer}(\mathsf{ck}_g, (\mathsf{com}_i)_{i \in T_g},$
        $\mathsf{com}_g, \pi_g^\chi) \ \forall g \in \mathcal{G}_{\mathsf{out}}$
    return 1
```

Figure 6.2: *Basic Compiler from PCFC to BARG.*

6.5 Wire-by-Wire Compiler: From Functional Extractability to BARGs

We provide a generic compiler to build a BARG for NP from somewhere extractable projective chainable funcitonal commitments (se-PCFC), as defined in Section 6.4. Our compiler can be seen as an abstraction of the construction by Waters and Wu [WW22]. The resulting BARG can evaluate circuits $C : \bar{\mathcal{M}}^\ell \to \bar{\mathcal{M}}$ with arbitrary gates, proving that $C(\mathtt{x}_i, \mathtt{w}_i) = 1$ for every $i \in [k]$. Notably, the compiler only makes black-box use of the underlying PCFC. Therefore, if the PCFC is algebraic, so does the resulting BARG.

6.5.1 Compiler construction

Our compiler is described in Figure 6.2. Given a quadratic gate function $\chi : \bar{\mathcal{M}}^m \to \bar{\mathcal{M}}$ for (arithmetic or Boolean) circuits $C : \bar{\mathcal{M}}^{\ell_{\mathtt{x},\mathtt{w}}} \to \bar{\mathcal{M}}$, we define $f_\chi : \bar{\mathcal{M}}^{k \cdot m} \to \bar{\mathcal{M}}^k$ as $f_\chi(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k) = (\chi(\boldsymbol{x}_1), \ldots, \chi(\boldsymbol{x}_k))$. Namely, f_χ is a k-separable function that runs χ in parallel k times, and such that $m < \ell_C$. The compiler turns a PCFC which 1) supports the functions f_χ, 2) is somewhere extractable and 3) is functional extractable into a BARG for NP. The formal requirements and implication are stated in the following Theorem 6.20. Note that in our construction the crs is independent of the circuit size ℓ_C.

Theorem 6.20 (Basic Compiler). *Let $\mathcal{I} := \{\{i\} : i \in [k]\}$ (i.e. singletons of indices in $[k]$), $\bar{\mathcal{M}} \supseteq \{0,1\}$ be some message space,* PCFC *be a projective chainable functional commitment scheme (Theorem 6.13) for the $\mathcal{I}$-separable functions $\mathcal{F} = \{f_\chi : \bar{\mathcal{M}}^{k \cdot m} \to \bar{\mathcal{M}}^k : \chi \text{ is a gate function}\}$ which satisfies*

- *(Theorem 6.10) somewhere extractability for $(\mathcal{I}, \bar{\mathcal{M}})$ and*
- *(Theorem 6.17) functional extractability for $(\mathcal{F}, \mathcal{I})$.*

Then, the construction BARG *in Figure 6.2 is a somewhere extractable batch argument (Theorem 6.3) for NP whose proof size is $|\pi| = \ell_C \cdot |\mathsf{com}| + \ell_{\mathtt{w}} \cdot |\pi^b| + (\ell_C - \ell_{\mathtt{x},\mathtt{w}}) \cdot |\pi^\chi|$ where π^b is a PC subspace proof, and π^χ is a PCFC functional proof for $f \in \mathcal{F}$. Moreover,* BARG *admits efficient verification with preprocessing (Theorem 6.5).*

Proof. The completeness, succinctness and setup indistinguishability of BARG follow by inspection, given the correctness, succinctness and setup indistinguishability of PCFC. The somewhere extractability of BARG follows from Theorem 6.21. Its efficient verification property follows from Theorem 6.24.

□

In the remainder of the section, we prove somewhere extractability in Section 6.5.2 and efficient verification in Section 6.5.3.

6.5.2 Proof of somewhere extractability

Lemma 6.21. *Let* PCFC *and* BARG *be as defined in Theorem 6.20. It holds that* BARG *has somewhere extractability.*

Proof. We proceed by a series of hybrid games $\mathsf{Hyb}^0, \mathsf{Hyb}^{1,0}, \ldots, \mathsf{Hyb}^{1,\ell_{\mathtt{w}}}, \mathsf{Hyb}^{2,0}, \ldots, \mathsf{Hyb}^{2,\ell_{\mathrm{out}}}$ which progressively include additional winning conditions. The formal descriptions of the games are given in Figure 6.3 with explanations given below.

$\underline{\mathsf{Hyb}^0}$**:** This is the standard somewhere extractability game (Theorem 6.3) with the algorithms in BARG inlined. To recall, in this game the adversary outputs a proof π and the experiment runs $\mathtt{w}_{i^*} \leftarrow \mathsf{BARG.Ext}(\mathsf{td}, C, \{\mathtt{x}_i\}_{i \in [k]}, \pi)$. The adversary wins if the proof verifies yet $C(\mathtt{x}_{i^*}, \mathtt{w}_{i^*}) \neq 1$.

```
$\mathsf{Hyb}^0_{\mathcal{A}}(\lambda)$:
$(\mathsf{ck}, \mathsf{td}_{i^*}) \leftarrow \mathsf{PCFC.ProjSetup}(1^\lambda, 1^k, i^*)$
$(C, \{\mathtt{x}_i\}_{i\in[k]}, \pi) \leftarrow \mathcal{A}(\mathsf{ck})$
$\mathsf{com}_g \leftarrow \mathsf{PCFC.Com}(\mathsf{ck}, (x_{1,g}, \dots, x_{k,g}))\ \ \forall g \in \mathcal{G}_{\mathsf{stmt}}$
$x^*_g \leftarrow \mathsf{PCFC.ProjExt}(\mathsf{td}, \mathsf{com}_g)\ \ \forall g \in \mathcal{G}_{\mathsf{wit}}$
$\mathtt{w}^* := (x^*_g)_{g\in\mathcal{G}_{\mathsf{wit}}}$
$\textbf{assert } \mathsf{com}_{\ell_C} = \mathsf{PCFC.Com}(\mathsf{ck}, \mathbf{1}_k)$
$\textbf{assert } \mathsf{PCFC.ProjVer}(\mathsf{ck}, \mathsf{com}_g, \pi^b_g) = 1\ \ \forall g \in \mathcal{G}_{\mathsf{wit}}$
$\textbf{assert } \mathsf{PCFC.FuncVer}(\mathsf{ck}, (\mathsf{com}_i)_{i\in T_g}, \mathsf{com}_g, f_{\chi_g}, \pi^\chi_g) = 1\ \ \forall g \in \mathcal{G}_{\mathsf{out}}$
$\textbf{assert } C(\mathtt{x}_{i^*}, \mathtt{w}^*) \neq 1$
$\textbf{return } 1$

$\mathsf{Hyb}^{1,j}_{\mathcal{A}}(\lambda), 0 \le j \le \ell_{\mathtt{w}}$:
    // identical to $\mathsf{Hyb}^0_{\mathcal{A}}(\lambda)$ until before "$\textbf{return } 1$"
$\textbf{assert } x^*_g \in \bar{\mathcal{M}}\ \ \forall g \in \mathcal{G}_{\mathsf{wit}} \cap [\ell_{\mathtt{x}} + j]$
$\textbf{assert } \mathsf{PCFC.ProjCom}(\mathsf{td}_{i^*}, x^*_g) = \mathsf{PCFC.Proj}(\mathsf{td}_{i^*}, \mathsf{com}_g)\ \ \forall g \in \mathcal{G}_{\mathsf{wit}} \cap [\ell_{\mathtt{x}} + j]$
$\textbf{return } 1$

$\mathsf{Hyb}^{2,j}_{\mathcal{A}}(\lambda), 0 \le j \le \ell_{\mathsf{out}}$:
    // identical to $\mathsf{Hyb}^{1,\ell_{\mathtt{w}}}_{\mathcal{A}}(\lambda)$ until before "$\textbf{return } 1$"
$\boldsymbol{x}^* \leftarrow \mathsf{Wires}(C, \mathtt{x}_{i^*}, \mathtt{w}^*)$
$\textbf{assert } x^*_g = \mathsf{PCFC.ProjExt}(\mathsf{td}_{i^*}, \mathsf{com}_g)\ \ \forall g \in \mathcal{G}_{\mathsf{out}} \cap [\ell_{\mathtt{x},\mathtt{w}} + j]$
$\textbf{return } 1$
```

Figure 6.3: *Games* $\mathsf{Hyb}^0, \mathsf{Hyb}^{1,j}, \mathsf{Hyb}^{2,j}$ *for the proof of BARG somewhere extractability. We highlight changes between games. We define the function* $\boldsymbol{x} \leftarrow \mathsf{Wires}(C, \mathtt{x}_i, \mathtt{w}_i)$ *to output the value of all the internal values of a circuit computation* $C(\mathtt{x}_i, \mathtt{w}_i)$*. Namely,* $x_j \leftarrow \chi_j((x_t)_{t\in T_j})$ *where the initial values are given by* $\boldsymbol{x} = \mathtt{x}_i|\mathtt{w}_i|\mathbf{0}$ *and* $\boldsymbol{x}$ *is recomputed iteratively* $\forall j \in [N]$*.*

$\mathsf{Hyb}^{1,j}$: $\mathsf{Hyb}^{1,0}$ is identical to Hyb^0. In $\mathsf{Hyb}^{1,j}$, the game additionally checks that the first j entries of the extracted witness, i.e. those indexed by g belonging to the first j values in $\mathcal{G}_{\mathsf{wit}}$, resides in the message subspace $\bar{\mathcal{M}}$, i.e. $x^*_g \in \bar{\mathcal{M}}^{\ell_{\mathtt{w}}}$, and that they are consistent with the projected commitments, i.e. $\mathsf{PCFC.ProjCom}(\mathsf{td}_{i^*}, x^*_g) = \mathsf{PCFC.Proj}(\mathsf{td}_{i^*}, \mathsf{com}_g)$.

$\mathsf{Hyb}^{2,j}$: $\mathsf{Hyb}^{2,0}$ is identical to $\mathsf{Hyb}^{1,\ell_{\mathtt{w}}}$. To define the rest of the games, we let $\boldsymbol{x}^* \in \bar{\mathcal{M}}^{\ell_C}$ be the (honestly computed) values of the circuit wires of instance i^* that result from the evaluation of $C(\mathtt{x}_{i^*}, \mathtt{w}^*)$. Note that $\boldsymbol{x}^*$ is guaranteed to satisfy $\mathtt{w}^* \in \bar{\mathcal{M}}^{\ell_{\mathtt{w}}}$ by the winning condition. In $\mathsf{Hyb}^{2,j}$, the game additionally checks that $x^*_g = \mathsf{PCFC.ProjExt}(\mathsf{td}_{i^*}, \mathsf{com}_g)$ for all g belonging to the first j values in $\mathcal{G}_{\mathsf{out}}$.

We now proceed to bound the advantage of any PPT adversary $\mathcal{A}$ against somewhere extractability, i.e. Hyb^0, by a series of lemmas. We remark that the proof also works for unbounded adversaries (proving statistical somewhere extractability of BARG) if PCFC satisfies (1) statistical functional somewhere extractability and (2) statistical projective somewhere extractability.

Lemma 6.22. *For any $j \in [\ell_{\mathsf{w}}]$, there exists a PPT adversary $\mathcal{B}$ against the somewhere extractability for $(\mathcal{I}, \bar{\mathcal{M}})$ of* PCFC *such that*

$$\left|\Pr[\mathsf{Hyb}^{1,j-1}_{\mathcal{A}}(\lambda) = 1] - \Pr[\mathsf{Hyb}^{1,j}_{\mathcal{A}}(\lambda) = 1]\right| \leq \mathsf{Adv}^{\mathsf{se}}_{\mathsf{PCFC},\mathcal{B}}(\lambda).$$

Proof. Fix $j \in [\ell_{\mathsf{w}}]$. We build an adversary $\mathcal{B}$ as follows. $\mathcal{B}$ receives input ck from its challenger and runs $\mathcal{A}(\mathsf{ck})$. Then, it parses the proof π of $\mathcal{A}$ and retrieves the tuple $(\mathsf{com}_{\ell_{\mathsf{x}}+j}, \pi^b_{\ell_{\mathsf{x}}+j})$. Finally, $\mathcal{B}$ simply forwards these to its challenger.

We argue that if $\mathcal{A}$ wins in $\mathsf{Hyb}^{1,j-1}$ but not in $\mathsf{Hyb}^{1,j}$, then $\mathcal{B}$ also wins the game. First, as $\mathcal{A}$ wins in $\mathsf{Hyb}^{1,j}$, it holds that $\mathsf{PCFC.ProjVer}(\mathsf{ck}, \mathsf{com}_{\ell_{\mathsf{x}}+j}, \pi^b_{\ell_{\mathsf{x}}+j}) = 1$. Also, note that $x^*_{\ell_{\mathsf{x}}+j} \leftarrow \mathsf{PCFC.ProjExt}(\mathsf{td}_{i^*}, \mathsf{com}_{\ell_{\mathsf{x}}+j})$. Therefore, all the preconditions for $\mathcal{B}$ in the PC somewhere extractability game are met. Finally, as $\mathcal{A}$ does not win in $\mathsf{Hyb}^{1,j}$, it must be that either $\mathsf{PCFC.Proj}(\mathsf{td}_{i^*}, \mathsf{com}_{\ell_{\mathsf{x}}+j}) \neq \mathsf{PCFC.ProjCom}(\mathsf{td}_{i^*}, x^*_{\ell_{\mathsf{x}}+j})$ or $x^*_{\ell_{\mathsf{x}}+j} \notin \bar{\mathcal{M}}$.

□

Lemma 6.23. *For any $j \in [\ell_{\mathsf{out}}]$, there exists a PPT adversary $\mathcal{B}$ against the functional extractability for $(\mathcal{F}, \mathcal{I})$ of* PCFC *such that*

$$\left|\Pr[\mathsf{Hyb}^{2,j-1}_{\mathcal{A}}(\lambda) = 1] - \Pr[\mathsf{Hyb}^{2,j}_{\mathcal{A}}(\lambda) = 1]\right| \leq \mathsf{Adv}^{\mathsf{fe}}_{\mathsf{PCFC},\mathcal{B}}(\lambda).$$

Proof. Fix $j \in [\ell_{\mathsf{out}}]$. We build $\mathcal{B}$ from $\mathcal{A}$ as follows:

- Receives input ck from its challenger and runs $(C, \{\mathsf{x}_i\}_{i\in[k]}, \pi) \leftarrow \mathcal{A}(\mathsf{ck})$.
- Parses C to obtain the gate function χ_j and the input wire set T_j.
- Parses the proof π and obtains $\{\mathsf{com}_g\}_{g \in T_j \setminus [\ell_{\mathsf{x}}]}, \pi^{\chi}_j, \mathsf{com}_j$.
- Outputs $(f_{\chi_j}, \{\mathsf{com}_g\}_{g\in T_j}, \mathsf{com}_j, \pi^{\chi}_j)$ where the com_g for $g \in \mathcal{G}_{\mathsf{stmt}} \cap T_g$ are computed honestly as $\mathsf{com}_g \leftarrow \mathsf{PCFC.Com}(\mathsf{ck}, (x_{1,g}, \ldots, x_{k,g}))$.

We argue that if $\mathcal{A}$ wins in $\mathsf{Hyb}^{2,j-1}$ but not in $\mathsf{Hyb}^{2,j}$, then $\mathcal{B}$ is a successful adversary against CFC functional extractability.

- By the winning condition (inherited from Hyb^0), it holds that the functional proof verifies, $\mathsf{FuncVer}(\mathsf{ck}, (\mathsf{com}_g)_{g\in T_j}, f_{\chi_j}, \mathsf{com}_j, \pi^{\chi}_j) = 1$.
- Let $\tilde{x}_g \leftarrow \mathsf{PCFC.ProjExt}(\mathsf{td}_{i^*}, \mathsf{com}_g)$ be the value extracted from the commitments for any $g \in \mathcal{G}$.
- As $\mathcal{A}$ wins in $\mathsf{Hyb}^{2,j-1}$, we have that $\tilde{x}_g = x^*_g = \chi_j((x^*_g)_{g\in T_j})$ for every $g \in [\ell_{\mathsf{x},\mathsf{w}} + j - 1]$, and therefore for every $g \in T_j$. By the definition of $\boldsymbol{x}^*$, it also follows that $\tilde{x}_g \in \bar{\mathcal{M}}$ for every $g \in T_j$.
- As $\mathcal{A}$ does not win in $\mathsf{Hyb}^{2,j}$, then $\tilde{x}_j \neq x^*_j$.

Hence, $\chi_j((\tilde{x}_g)_{g\in T_j}) \neq \tilde{x}_j$ and the output of $\mathcal{B}$ is a successful break of CFC functional extractability.

□

Note that the advantage of $\mathcal{A}$ in $\mathsf{Hyb}^{2,\ell_{\mathsf{out}}}$ is zero, as the condition $C(\mathtt{x}_{i^*}, w^*) \neq 1$ is incompatible with $x^*_{\ell_C} = \mathsf{PCFC.ProjExt}(\mathsf{td}_{i^*}, \mathsf{com}_{\ell_C}) = 1$, which holds as com_{ℓ_C} is computed honestly by the verifier. We collect the bounds of Theorem 6.22 and Theorem 6.23, and conclude that the advantage of $\mathcal{A}$ in the original Hyb^0 is bounded by

$$\Pr[\mathsf{Hyb}^0_{\mathcal{A}}(\lambda) = 1] \leq \ell_{\mathtt{w}} \cdot \mathsf{Adv}^{\mathsf{se}}_{\mathsf{PCFC},\mathcal{B}}(\lambda) + \ell_{\mathsf{out}} \cdot \mathsf{Adv}^{\mathsf{fe}}_{\mathsf{PCFC},\mathcal{B}}(\lambda).$$

□

6.5.3 Proof of efficient verification

Lemma 6.24. *Let* PCFC *and* BARG *be as defined in Theorem 6.20. If further* PCFC *has efficient verification with preprocessing (Theorem 6.14), then* BARG *has efficient verification with preprocessing (Theorem 6.5).*

Proof. As per Theorem 6.5, it suffices to show that $|\mathsf{vk}|$ and the runtime of $\mathsf{BARG.EffVer}(\lambda, \mathsf{vk})$ are at most $\mathsf{poly}(\lambda, \log k, \ell_C)$. We actually prove below a stronger claim that these quantities are at most $\mathsf{poly}(\lambda, \log k, \log \ell_C) \cdot \ell_C$.

We first examine $|\mathsf{vk}|$. Recall that $\mathsf{vk} = ((\mathsf{com}_g)_{g\in\mathcal{G}_{\mathsf{stmt}}}, \mathsf{ck}_{[k]}, (\mathsf{ck}_g)_{g\in\mathcal{G}_{\mathsf{out}}})$. Note that by Theorem 3.1 fresh commitments are of size at most $\mathsf{poly}(\lambda, \log k)$ and by Theorem 3.5 aggregated commitments are identical to fresh ones. Therefore, $|\mathsf{com}_g| \leq \mathsf{poly}(\lambda, \log k)$. Next, by Theorem 4.5 it holds that $|\mathsf{ck}_g| \leq \mathsf{poly}(\lambda, \log k, \log m, \log |\chi_g|)$ where $m = |T_g| \leq \ell_C$ in the current context and $|\chi_g| \leq \ell_C$. Thus, $|\mathsf{ck}_g| \leq \mathsf{poly}(\lambda, \log k, \log \ell_C)$. Finally, by Theorem 6.12 it holds that $|\mathsf{ck}_{[k]}| \leq \mathsf{poly}(\lambda, \log k)$. Putting everything together, we conclude that $|\mathsf{vk}| \leq \mathsf{poly}(\lambda, \log k, \log \ell_C)$.

For the runtime of $\mathsf{BARG.EffVer}(\lambda, \mathsf{vk})$, by Theorem 3.5, each of the $\ell_{\mathtt{w}}$ calls to $\mathsf{PCFC.EffProjVer}$ runs in time at most $\mathsf{poly}(\lambda, \log k)$ according to Theorem 6.12. Also, each of the $|\mathcal{G}_{\mathsf{out}}| \leq \ell_C$ calls to $\mathsf{PCFC.EffFuncVer}$ runs in time at most $\mathsf{poly}(\lambda, \log k, \log m, \log |\chi_g|)$ which as analysed above is at most $\mathsf{poly}(\lambda, \log k, \log \ell_C)$. We thus conclude that $\mathsf{BARG.EffVer}(\lambda, \mathsf{vk})$ runs in time at most $\mathsf{poly}(\lambda, \log k, \log \ell_C) \cdot \ell_C$.

□

Note: in precomputation we could fix all wires of the statements corresponding to gates (when precomputing the function) by modifying the gates directly. The circuit would simply become a single-input $C(\mathtt{w})$.

6.6 Circuit-Succinct Compiler from PCFC to BARG

In this section we construct our black-box compiler that turns a PCFC (Theorem 6.13) into a circuit-succinct BARG for NP, this is, a BARG such that the proof size grows linearly on the size of the witness $\ell_{\mathtt{w}}$. Our BARG can prove batch relations $C(\mathtt{x}_i, \mathtt{w}_i)$ for $i \in [k]$, where $C : \bar{\mathcal{M}}^{\ell_{\mathtt{x},\mathtt{w}}} \to \bar{\mathcal{M}}$ are (possibly distinct, see Theorem 6.27) arithmetic circuits of bounded size $|C| \leq \ell_C$.

We briefly describe our circuit-succinct compiler in Figure 6.4. The prover starts by committing to each of the witness wires across the k batch instances, obtaining commitments $\mathsf{com}_{\ell_{\mathtt{x}}+1}, \ldots, \mathsf{com}_{\ell_{\mathtt{x},\mathtt{w}}}$. The prover also provides subspace proofs on each of these commitments. Until this point, the compiler is analogous to the first step of the BARG construction in [WW22] (which we abstract in our wire-by-wire compiler in Section 6.5). Then, instead of carrying out functional proofs for each circuit wire, the prover generates a PCFC functional opening as follows. First, it defines a function f_C to be the parallel evaluation of C k times. Then, it sets an input $\boldsymbol{x}$ which is the concatenation of $(\mathtt{x}_i, \mathtt{w}_i)$ for every $i \in [k]$. Finally, it provides a functional proof of $f_C(\boldsymbol{x}) = (C(\mathtt{x}_1, \mathtt{w}_1), \ldots, C(\mathtt{x}_k, \mathtt{w}_k)) = \mathbf{1}_k$. Along with this proof, the prover provides an input commitment com and an output commitment com_y which commits to $\mathbf{1}_k$. The validity of com can be verified as it must match the aggregation of the commitments to the statement wires, which are computed by the verifier, along with the aggregation of $\mathsf{com}_{\ell_{\mathtt{x}}+1}, \ldots, \mathsf{com}_{\ell_{\mathtt{x},\mathtt{w}}}$.

The proof of BARG somewhere extractability for instance i^* intuitively works as follows. First, PC somewhere extractability enables the extraction of a binary witness $\mathtt{w}_{i^*}$ from each of the $\ell_{\mathtt{w}}$ commitment wires, where $\mathtt{w}_{i^*}$ is guaranteed to match the projections of these commitments. Second, we apply PC projective aggregatability to ensure that the projection of com indeed commits to $(\mathtt{x}_{i^*}, \mathtt{w}_{i^*})$. Finally, we crucially rely on PCFC projective chain binding to ensure that $C(\mathtt{x}_{i^*}, \mathtt{w}_{i^*}) = 1$.

Notation and parameters. For an (arithmetic or Boolean) circuit $C : \bar{\mathcal{M}}^{\ell_{\mathtt{x},\mathtt{w}}} \to \bar{\mathcal{M}}$, we define $f_C : \bar{\mathcal{M}}^{k \cdot \ell_{\mathtt{x},\mathtt{w}}} \to \bar{\mathcal{M}}^{k \cdot \ell_{\mathtt{x},\mathtt{w}}}$ as $f_C(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k) = (C(\boldsymbol{x}_1), \ldots, C(\boldsymbol{x}_k))$. Let $\mathcal{F}$ be the family of such functions $\mathcal{F} = \{f_C : C : \bar{\mathcal{M}}^{\ell_{\mathtt{x},\mathtt{w}}} \to \bar{\mathcal{M}}\}$, and we let the input length $\ell = k \cdot \ell_{\mathtt{x},\mathtt{w}}$.

To express $(i-1)\ell_{\mathtt{x},\mathtt{w}} + g \in [\ell]$ for $i \in [k], g \in \ell_{\mathtt{x},\mathtt{w}}$, we abuse notation and denote it by $(i, g) \in [\ell]$. That is, we use a bijective mapping between $[k] \times [\ell_{\mathtt{x},\mathtt{w}}]$ and $[\ell]$ given by $(i, g) \mapsto (i-1)\ell_{\mathtt{x},\mathtt{w}} + g$. This allows us to directly map values to their corresponding instance and wire. We define the index subsets $I_i := \{(i, g) : g \in \mathcal{G}_{\mathsf{stmt}} \cup \mathcal{G}_{\mathsf{wit}}\}$ for $i \in [k]$ and $J_g := \{(i, g) : i \in [k]\}$ for $g \in \mathcal{G}_{\mathsf{stmt}} \cup \mathcal{G}_{\mathsf{wit}}$. Note that $I_i \cap J_g = \{(i, g)\}$ isolates a single index $(i, g) \in [\ell]$. By construction, all functions in $\mathcal{F}$ are $\mathcal{I}$-separable for the family of index sets $\mathcal{I} = \{I_i\}_{i \in [k]}$.

In Table 6.1, we show how the BARG inputs are arranged with respect to instances and subvectors. Recall that, for a given batch $i \in [k]$, we denote $x_{i,j} \leftarrow \mathtt{x}_{i,j}$ for $j = 1, \ldots, \ell_{\mathtt{x}}$, and $x_{i,j} \leftarrow \mathtt{w}_{i,(j-\ell_{\mathtt{x}})}$ for $j = \ell_{\mathtt{x}} + 1, \ldots, \ell_{\mathtt{x},\mathtt{w}}$.

Theorem 6.25 (Circuit-Succinct Compiler). *Let $\mathcal{I} := \{I_i : i \in [k]\}$ where $I_i := \{(i, g) : g \in \mathcal{G}_{\mathsf{stmt}} \cup \mathcal{G}_{\mathsf{wit}}\}$, $\mathcal{J} := \{J_g : g \in \mathcal{G}_{\mathsf{stmt}} \cup \mathcal{G}_{\mathsf{wit}}\}$ where $J_g := \{(i, g) : i \in [k]\}$, $\mathcal{F} = \{f_C : C : \bar{\mathcal{M}}^{\ell_{\mathtt{x},\mathtt{w}}} \to \bar{\mathcal{M}}\}$ where $f_C(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k) = (C(\boldsymbol{x}_1), \ldots, C(\boldsymbol{x}_k))$, $\bar{\mathcal{M}} \supseteq \{0, 1\}$ be some message space,* PCFC *be a projective chainable functional commitment scheme (Theorem 6.13) for the $\mathcal{I}$-separable functions $\mathcal{F}$ and index set families $(\mathcal{I}, \mathcal{J})$ which satisfies*

- *(Theorem 6.8) PC projective aggregatability,*

```
$\mathsf{Setup}(1^\lambda, k, 1^{\ell_C})$
    $\ell := k \cdot \ell_{\mathtt{x},\mathtt{w}}$
    $\mathsf{ck} \leftarrow \mathsf{PCFC.Setup}(1^\lambda, 1^\ell)$
    $\mathsf{com}_{\mathsf{out}} \leftarrow \mathsf{Com}(\mathsf{ck}, 1^\ell)$
    $\mathbf{return}\ \mathsf{crs} := (\mathsf{ck}, \mathsf{com}_{\mathsf{out}})$

$\mathsf{SetupTd}(1^\lambda, k, 1^{\ell_C}, i^*)$
    $\ell := k \cdot \ell_{\mathtt{x},\mathtt{w}}$
    $(\mathsf{ck}, \mathsf{td}) \leftarrow \mathsf{PCFC.ProjSetup}(1^\lambda, 1^\ell, I_{i^*})$
    $\mathsf{com}_{\mathsf{out}} \leftarrow \mathsf{Com}(\mathsf{ck}, 1^\ell)$
    $\mathbf{return}\ (\mathsf{crs}, \mathsf{td}) := ((\mathsf{ck}, \mathsf{com}_{\mathsf{out}}), \mathsf{td})$

$\mathsf{Ext}(\mathsf{td}, C, \{\mathtt{x}_i\}_{i \in [k]}, \pi)$
    $\mathtt{w}_{i^*,g} \leftarrow \mathsf{PCFC.ProjExt}(\mathsf{td}, J_g, \mathsf{com}_g)\ \ \forall g \in \mathcal{G}_{\mathsf{wit}}$
    $\mathbf{return}\ \mathtt{w}_{i^*} := (\mathtt{w}_{i^*,g})_{g \in \mathcal{G}_{\mathsf{wit}}}$

$\mathsf{PreVer}(\mathsf{crs}, C, \{\mathtt{x}_i\}_{i \in [k]})$
    $\boldsymbol{x}_g := (x_{1,g}, \dots, x_{k,g})\ \ \forall g \in \mathcal{G}_{\mathsf{stmt}}$
    $\mathsf{com}_g \leftarrow \mathsf{PCFC.Com}(\mathsf{ck}, J_g, \boldsymbol{x}_g)\ \ \forall g \in \mathcal{G}_{\mathsf{stmt}}$
    $\mathsf{com}_{\mathcal{G}_{\mathsf{stmt}}} \leftarrow \mathsf{PCFC.Agg}(\mathsf{ck}, (\mathsf{com}_g)_{g \in \mathcal{G}_{\mathsf{stmt}}})$
    $\mathsf{ck}_{f_C} \leftarrow \mathsf{PCFC.PreFuncVer}(\mathsf{ck}, f_C)$
    $\mathsf{ck}_{J_g} \leftarrow \mathsf{PCFC.PreProjVer}(\mathsf{ck}, J_g)\ \ \forall g \in \mathcal{G}_{\mathsf{wit}}$
    $\mathsf{vk} := (\mathsf{com}_{\mathcal{G}_{\mathsf{stmt}}}, \mathsf{ck}_{f_C}, (\mathsf{ck}_{J_g})_{g \in \mathcal{G}_{\mathsf{wit}}})$
    $\mathbf{return}\ \mathsf{vk}$
```

```
$\mathsf{Prove}(\mathsf{crs}, C, \{(\mathtt{x}_i, \mathtt{w}_i)\}_{i \in [k]})$
    $\boldsymbol{x}_g := (x_{1,g}, \dots, x_{k,g})\ \ \forall g \in \mathcal{G}_{\mathsf{stmt}} \cup \mathcal{G}_{\mathsf{wit}}$
    $\boldsymbol{x} := (\boldsymbol{x}_g)_{g \in \mathcal{G}_{\mathsf{stmt}} \cup \mathcal{G}_{\mathsf{wit}}}$
    $\mathsf{com}_g \leftarrow \mathsf{PCFC.Com}(\mathsf{ck}, J_g, \boldsymbol{x}_g)\ \ \forall g \in \mathcal{G}_{\mathsf{stmt}} \cup \mathcal{G}_{\mathsf{wit}}$
    $\pi_g^b \leftarrow \mathsf{PCFC.ProjProve}(\mathsf{ck}, J_g, \boldsymbol{x}_g)\ \ \forall g \in \mathcal{G}_{\mathsf{wit}}$
    $\pi^X \leftarrow \mathsf{PCFC.FuncProve}(\mathsf{ck}, \boldsymbol{x}, f_C)$
    $\mathbf{return}\ \pi := ((\mathsf{com}_g, \pi_g^b)_{g \in \mathcal{G}_{\mathsf{wit}}}, \pi^X)$

$\mathsf{Ver}(\mathsf{crs}, C, \{\mathtt{x}_i\}_{i \in [k]}, \pi)$
    $\boldsymbol{x}_g := (x_{1,g}, \dots, x_{k,g})\ \ \forall g \in \mathcal{G}_{\mathsf{stmt}}$
    $\mathsf{com}_g \leftarrow \mathsf{PCFC.Com}(\mathsf{ck}, J_g, \boldsymbol{x}_g)\ \ \forall g \in \mathcal{G}_{\mathsf{stmt}}$
    $\mathsf{com} \leftarrow \mathsf{PCFC.Agg}(\mathsf{ck}, (\mathsf{com}_g)_{g \in \mathcal{G}_{\mathsf{stmt}} \cup \mathcal{G}_{\mathsf{wit}}})$
    $\mathbf{assert}\ \mathsf{PCFC.ProjVer}(\mathsf{ck}, J_g, \mathsf{com}_g, \pi_g^b)\ \ \forall g \in \mathcal{G}_{\mathsf{wit}}$
    $\mathbf{assert}\ \mathsf{PCFC.FuncVer}(\mathsf{ck}, \mathsf{com}, \mathsf{com}_{\mathsf{out}}, f_C, \pi^X)$
    $\mathbf{return}\ 1$

$\mathsf{EffVer}(\mathsf{vk}, \pi)$
    $\mathsf{com}_{\mathcal{G}_{\mathsf{wit}}} \leftarrow \mathsf{PCFC.Agg}(\mathsf{ck}, (\mathsf{com}_g)_{g \in \mathcal{G}_{\mathsf{wit}}})$
    $\mathsf{com} \leftarrow \mathsf{PCFC.Agg}(\mathsf{ck}, (\mathsf{com}_{\mathcal{G}_{\mathsf{stmt}}}, \mathsf{com}_{\mathcal{G}_{\mathsf{wit}}}))$
    $\mathbf{assert}\ \mathsf{PCFC.EffProjVer}(\mathsf{ck}_{J_g}, \mathsf{com}_g, \pi_g^b)\ \ \forall g \in \mathcal{G}_{\mathsf{wit}}$
    $\mathbf{assert}\ \mathsf{PCFC.EffFuncVer}(\mathsf{ck}_{f_C}, \mathsf{com}, \mathsf{com}_{\mathsf{out}}, \pi^X)$
    $\mathbf{return}\ 1$
```

Figure 6.4: *Compiler from PCFC to fully succinct BARG.*

- *(Theorem 6.10) PC somewhere extractability for $(\mathcal{I}, \mathcal{J}, \bar{\mathcal{M}})$ and*
- *(Theorem 6.19) PCFC projective chain binding for $(\mathcal{F}, \mathcal{I}, \mathcal{J})$.*

Then, the construction BARG *in Figure 6.4 is a somewhere extractable batch argument (Theorem 6.3) for NP whose proof size is $|\pi| = \ell_{\mathtt{w}} \cdot (|\mathsf{com}| + |\pi^b|) + |\pi^X|$ where π^b is a PC subspace proof, and π^X is a PCFC functional proof for $f \in \mathcal{F}$. Moreover,* BARG *admits efficient verification with preprocessing.*

Proof. The completeness, full succinctness and setup indistinguishability of BARG follow by inspection, given the correctness, succinctness and setup indistinguishability of PCFC. The somewhere extractability of BARG follows from Theorem 6.28. Its efficient verification property follows from Theorem 6.32.

□

Remark 6.26. *The projective aggregatability property required by Theorem 6.25 is optional – its only purpose is so that a BARG proof does not need to include $|\mathcal{G}_{\mathsf{wit}}|$ commitments. If we were*

	statements				witnesses					wires			
	J_1	J_2	$\cdots$	$J_{\ell_\mathtt{x}}$	$J_{\ell_\mathtt{x}+1}$	$\cdots$	$J_{\ell_\mathtt{x}+j}$	$\cdots$	$J_{\ell_{\mathtt{x},\mathtt{w}}}$	$J_{\ell_{\mathtt{x},\mathtt{w}}+1}$	$J_{\ell_{\mathtt{x},\mathtt{w}}+2}$	$\cdots$	J_{ℓ_C}
I_1	$\mathtt{x}_{1,1}$	$\mathtt{x}_{1,2}$	$\cdots$	$\mathtt{x}_{1,\ell_\mathtt{x}}$	$\mathtt{w}_{1,1}$	$\cdots$	$\mathtt{w}_{1,j}$	$\cdots$	$\mathtt{w}_{1,\ell_\mathtt{w}}$	$x_{1,\ell_{\mathtt{x},\mathtt{w}}+1}$	$x_{1,\ell_{\mathtt{x},\mathtt{w}}+2}$	$\cdots$	x_{1,ℓ_C}
$\vdots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$		$\vdots$		$\vdots$	$\vdots$	$\vdots$		$\vdots$
I_i	$\mathtt{x}_{i,1}$	$\mathtt{x}_{i,2}$	$\cdots$	$\mathtt{x}_{i,\ell_\mathtt{x}}$	$\mathtt{w}_{i,1}$	$\cdots$	$\mathtt{w}_{i,j}$	$\cdots$	$\mathtt{w}_{i,\ell_\mathtt{w}}$	$x_{i,\ell_{\mathtt{x},\mathtt{w}}+1}$	$x_{i,\ell_{\mathtt{x},\mathtt{w}}+2}$	$\cdots$	x_{i,ℓ_C}
$\vdots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$		$\vdots$		$\vdots$	$\vdots$	$\vdots$		$\vdots$
I_k	$\mathtt{x}_{k,1}$	$\mathtt{x}_{k,2}$	$\cdots$	$\mathtt{x}_{k,\ell_\mathtt{x}}$	$\mathtt{w}_{k,1}$	$\cdots$	$\mathtt{w}_{k,j}$	$\cdots$	$\mathtt{w}_{k,\ell_\mathtt{w}}$	$x_{k,\ell_{\mathtt{x},\mathtt{w}}+1}$	$x_{k,\ell_{\mathtt{x},\mathtt{w}}+2}$	$\cdots$	x_{k,ℓ_C}
	$\downarrow$ com_1	$\downarrow$ com_2		$\downarrow$ $\mathsf{com}_{\ell_\mathtt{x}}$	$\downarrow$ $\mathsf{com}_{\ell_\mathtt{x}+1}$		$\downarrow$ $\mathsf{com}_{\ell_\mathtt{x}+j}$		$\downarrow$ $\mathsf{com}_{\ell_{\mathtt{x},\mathtt{w}}}$				

Table 6.1: *Structure of the committed values in Figure 6.4. The rows are associated to projection sets $I_i \in \mathcal{I}$. The columns are associated to the subvector sets $J_j \in \mathcal{J}$.*

to drop the projective aggregatability property, the compiler needs to be slightly modified where FuncProve *and* FuncVer *instead take as input multiple vectors and input commitments respectively.*

Remark 6.27. *Although we instantiate our compiler for the standard batch relation defined by $C(\mathtt{x}_i, \mathtt{w}_i)$ for $i \in [k]$, where the circuit is the same for all batch instances, it actually supports batch proofs for distinct circuits $C_i(\mathtt{x}_i, \mathtt{w}_i)$ for $i \in [k]$* without any additional overhead. *The only requirements are that each $|C_i| \leq \ell_C$ and that the PCFC supports the consequently broader family of functions $\mathcal{F}_{k,\ell_f}$ (defined in Section 6.7), which is the case for our algebraic PCFC introduced there. This property can be useful in applications such as the aggregation of signature schemes with different verification equations.*

In the remainder of the section, we prove somewhere extractability in Section 6.6.1 and efficient verification in Section 6.6.2.

6.6.1 Proof of Somewhere Extractability

Lemma 6.28. *Let* PCFC *and* BARG *be as defined in Theorem 6.25. It holds that* BARG *has somewhere extractability.*

Proof. We proceed by a series of hybrid games $\mathsf{Hyb}^0, \mathsf{Hyb}^{1,0}, \ldots, \mathsf{Hyb}^{1,\ell_\mathtt{w}}, \mathsf{Hyb}^2$ which progressively include additional winning conditions. The formal descriptions of the games are given in Figure 6.5 with explanations given below.

$\underline{\mathsf{Hyb}^0}$**:** This is the standard somewhere extractability game (Theorem 6.3) with the algorithms in BARG inlined. To recall, in this game the adversary outputs a proof π and the experiment runs $\mathtt{w}_{i^*} \leftarrow \mathsf{BARG.Ext}(\mathsf{td}, C, \{\mathtt{x}_i\}_{i\in[k]}, \pi)$. The adversary wins if the proof verifies yet $C(\mathtt{x}_{i^*}, \mathtt{w}_{i^*}) \neq 1$.

$\underline{\mathsf{Hyb}^{1,j}}$**:** $\mathsf{Hyb}^{1,0}$ is identical to Hyb^0. In $\mathsf{Hyb}^{1,j}$, the game additionally checks that the first j entries of the extracted witness, i.e. those indexed by g belonging to the first j values in $\mathcal{G}_{\mathsf{wit}}$, resides in the message subspace $\bar{\mathcal{M}}$, i.e. $x^*_g \in \bar{\mathcal{M}}^{\ell_\mathtt{w}}$, and that they are consistent with the projected commitments, i.e. $\mathsf{PCFC.ProjCom}(\mathsf{td}, J_g, x^*_g) = \mathsf{PCFC.Proj}(\mathsf{td}, \mathsf{com}_g)$.

$\underline{\mathsf{Hyb}^2}$**:** This game simply adds the winning condition that

$$\mathsf{PCFC.Proj}(\mathsf{td}, \mathsf{com}) = \mathsf{PCFC.ProjCom}(\mathsf{td}, I_{i^*}, (\mathtt{x}_{i^*}, \mathtt{w}^*)).$$

```
$\mathsf{Hyb}^0_{\mathcal{A}}(\lambda)$:
$(\mathsf{ck}, \mathsf{td}) \leftarrow \mathsf{PCFC.ProjSetup}(1^\lambda, 1^k, I_{i^*})$
$\mathsf{com}_{\mathsf{out}} \leftarrow \mathsf{Com}(\mathsf{ck}, 1^\ell)$
$(C, \{\mathtt{x}_i\}_{i\in[k]}, ((\mathsf{com}_g, \pi^b_g)_{g\in\mathcal{G}_{\mathsf{wit}}}, \pi^{\mathcal{X}})) \leftarrow \mathcal{A}(\mathsf{ck}, \mathsf{com}_{\mathsf{out}})$
$\mathsf{com}_g \leftarrow \mathsf{PCFC.Com}(\mathsf{ck}, J_g, \boldsymbol{x}_g)\ \forall g \in \mathcal{G}_{\mathsf{stmt}}$
$\mathsf{com} \leftarrow \mathsf{PCFC.Agg}(\mathsf{ck}, (\mathsf{com}_g)_{g\in\mathcal{G}_{\mathsf{stmt}}\cup\mathcal{G}_{\mathsf{wit}}})$
$x^*_g \leftarrow \mathsf{PCFC.ProjExt}(\mathsf{td}, J_g, \mathsf{com}_g)\ \forall g \in \mathcal{G}_{\mathsf{wit}}$
$\mathtt{w}^* := (x^*_g)_{g\in\mathcal{G}_{\mathsf{wit}}}$
assert $\mathsf{PCFC.ProjVer}(\mathsf{ck}, J_g, \mathsf{com}_g, \pi^b_g) = 1\ \forall g \in \mathcal{G}_{\mathsf{wit}}$
assert $\mathsf{PCFC.FuncVer}(\mathsf{ck}, \mathsf{com}, \mathsf{com}_{\mathsf{out}}, f_C, \pi^{\mathcal{X}}) = 1$
assert $C(\mathtt{x}_{i^*}, \mathtt{w}^*) \neq 1$
return 1

$\mathsf{Hyb}^{1,j}_{\mathcal{A}}(\lambda), 0 \le j \le \ell_{\mathtt{w}}$:
    // identical to $\mathsf{Hyb}^0_{\mathcal{A}}(\lambda)$ until before "return 1"
assert $x^*_g \in \bar{\mathcal{M}}\ \forall g \in \mathcal{G}_{\mathsf{wit}} \cap [\ell_{\mathtt{x}} + j]$
assert $\mathsf{PCFC.ProjCom}(\mathsf{td}, J_g, x^*_g) = \mathsf{PCFC.Proj}(\mathsf{td}, \mathsf{com}_g)\ \forall g \in \mathcal{G}_{\mathsf{wit}} \cap [\ell_{\mathtt{x}} + j]$

$\mathsf{Hyb}^2_{\mathcal{A}}(\lambda), 0 \le j \le \ell_{\mathsf{out}}$:
    // identical to $\mathsf{Hyb}^{1,\ell_{\mathtt{w}}}_{\mathcal{A}}(\lambda)$ until before "return 1"
assert $\mathsf{PCFC.ProjCom}(\mathsf{td}, I_{i^*}, (\mathtt{x}_{i^*}, \mathtt{w}^*)) = \mathsf{PCFC.Proj}(\mathsf{td}, \mathsf{com})$
```

Figure 6.5: *Games* $\mathsf{Hyb}^0, \mathsf{Hyb}^{1,j}, \mathsf{Hyb}^2$ *for the proof of BARG somewhere extractability. All games output* 1 *by default. We highlight changes between games.*

We now proceed to bound the advantage of any PPT adversary $\mathcal{A}$ against Hyb^0 by a series of lemmas.

Lemma 6.29. *For any* $j \in [\ell_{\mathtt{w}}]$*, there exists a PPT adversary* $\mathcal{B}$ *against the somewhere extractability for* $(\mathcal{I}, \mathcal{J}, \bar{\mathcal{M}})$ *of* PCFC *such that*

$$\left|\Pr[\mathsf{Hyb}^{1,j-1}_{\mathcal{A}}(\lambda) = 1] - \Pr[\mathsf{Hyb}^{1,j}_{\mathcal{A}}(\lambda) = 1]\right| \le \mathsf{Adv}^{\mathrm{se}}_{\mathsf{PCFC},\mathcal{B}}(\lambda).$$

Proof. Fix $j \in [\ell_{\mathtt{w}}]$. We build an adversary $\mathcal{B}$ as follows. $\mathcal{B}$ receives input ck from its challenger and runs $\mathcal{A}(\mathsf{ck})$. Then, it parses the proof π of $\mathcal{A}$ and retrieves the tuple $(\mathsf{com}_{\ell_{\mathtt{x}}+j}, \pi^b_{\ell_{\mathtt{x}}+j})$. Finally, $\mathcal{B}$ simply forwards these to its challenger.

We argue that if $\mathcal{A}$ wins in $\mathsf{Hyb}^{1,j-1}$ but not in $\mathsf{Hyb}^{1,j}$, then $\mathcal{B}$ also wins the game. First, as $\mathcal{A}$ wins in $\mathsf{Hyb}^{1,j}$, it holds that $\mathsf{PCFC.ProjVer}(\mathsf{ck}, \mathsf{com}_{\ell_{\mathtt{x}}+j}, \pi^b_{\ell_{\mathtt{x}}+j}) = 1$. Also, note that $x^*_{\ell_{\mathtt{x}}+j} \leftarrow \mathsf{PCFC.ProjExt}(\mathsf{td}_{i^*}, \mathsf{com}_{\ell_{\mathtt{x}}+j})$. Therefore, all the preconditions for $\mathcal{B}$ in the PC somewhere extractability game are met. Finally, as $\mathcal{A}$ does not win in $\mathsf{Hyb}^{1,j}$, it must be that either $\mathsf{PCFC.Proj}(\mathsf{td}_{i^*}, \mathsf{com}_{\ell_{\mathtt{x}}+j}) \neq \mathsf{PCFC.ProjCom}(\mathsf{td}_{i^*}, x^*_{\ell_{\mathtt{x}}+j})$ or $x^*_{\ell_{\mathtt{x}}+j} \notin \bar{\mathcal{M}}$.

□

Lemma 6.30. *It holds that*

$$\Pr[\mathsf{Hyb}^{1,\ell_w}_{\mathcal{A}}(\lambda) = 1] = \Pr[\mathsf{Hyb}^{2}_{\mathcal{A}}(\lambda) = 1].$$

Proof. Immediate from projective aggregatability (Theorem 6.8).

□

Lemma 6.31. *There exists a PPT adversary $\mathcal{B}$ against projective chain binding for $(\mathcal{F}, \mathcal{I}, \mathcal{J})$ of* PCFC *such that*

$$\Pr[\mathsf{Hyb}^{2}_{\mathcal{A}}(\lambda) = 1] \le \mathsf{Adv}^{\mathrm{pcb}}_{\mathsf{PCFC},\mathcal{B}}(\lambda).$$

Proof. We build the adversary $\mathcal{B}$ against projective chain binding from $\mathcal{A}$ against Hyb^2 as follows. Our adversary $\mathcal{B}$ receives $(\mathsf{ck}, \mathsf{td})$ from its challenger, computes $\mathsf{com}_{\mathsf{out}} = \mathsf{PCFC.Com}(\mathsf{ck}, 1^{\ell})$ and runs $\mathcal{A}$ on $(\mathsf{ck}, \mathsf{com}_{\mathsf{out}})$ (note that $\mathcal{A}$ does not receive td as input). It obtains $(f_C, \mathsf{com}_{\mathsf{out}}, (\mathsf{com}_g)_{g \in \mathcal{G}_{\mathsf{wit}}}, \pi^{\mathcal{X}})$ from $\mathcal{A}$.

Next, $\mathcal{B}$ extracts the witness values $\mathtt{w}^*_g \leftarrow \mathsf{PCFC.ProjExt}(\mathsf{td}, J_g, \mathsf{com}_g)$ one by one from the commitments $(\mathsf{com}_g)_{g \in \mathcal{G}_{\mathsf{wit}}}$ and generates dummy circuit inputs $\tilde{\boldsymbol{x}}_i = \mathtt{x}_i | \tilde{\mathtt{w}}_i$ as follows:

$$\tilde{\boldsymbol{x}}_i \leftarrow \mathtt{x}_i | \mathbf{0}_{\ell_w} \text{ if } i \ne i^*; \quad \mathtt{x}_i | \mathtt{w}^* \text{ if } i = i^*.$$

It commits to the dummy inputs, $\tilde{\mathsf{com}} \leftarrow \mathsf{PCFC.Com}(\mathsf{ck}, (\tilde{\boldsymbol{x}}_1, \ldots, \tilde{\boldsymbol{x}}_k))$. Let also $\tilde{y}_i \leftarrow C(\tilde{\boldsymbol{x}}_i)$ for all $i \in [k]$, and let $\tilde{\mathsf{com}}_{\mathsf{out}} \leftarrow \mathsf{PCFC.Com}(\mathsf{ck}, (\tilde{y}_1, \ldots, \tilde{y}_k))$. Finally, $\mathcal{B}$ generates an honestly computed opening proof that $f_C(\tilde{\boldsymbol{x}}_1, \ldots, \tilde{\boldsymbol{x}}_k) = (y_1, \ldots, y_k)$ by running $\tilde{\pi}^{\mathcal{X}} \leftarrow \mathsf{PCFC.FuncProve}(\mathsf{ck}, \tilde{\boldsymbol{x}}, f_C)$. Then, it returns the tuple

$$(f_C, \mathsf{com}, \tilde{\mathsf{com}}, \mathsf{com}_{\mathsf{out}}, \tilde{\mathsf{com}}_{\mathsf{out}}, \pi^{\mathcal{X}}, \tilde{\pi}^{\mathcal{X}}).$$

Clearly, $\mathcal{B}$ runs in polynomial time. We next analyse its success probability of violating the projective chain binding property. Let $(\mathsf{com}_g)_{g \in \mathcal{G}_{\mathsf{wit}}}, \pi^{\mathcal{X}}$ be parsed from the outputs of $\mathcal{A}$. Let $\mathsf{com}_g \leftarrow \mathsf{PCFC.Com}(\mathsf{ck}, J_g, \boldsymbol{x}_g)$ for $g \in \mathcal{G}_{\mathsf{stmt}}$ and $\mathsf{com} \leftarrow \mathsf{PCFC.Agg}(\mathsf{ck}, (\mathsf{com}_g)_{g \in \mathcal{G}_{\mathsf{stmt}} \cup \mathcal{G}_{\mathsf{wit}}})$. By the correctness of PCFC,

- $\mathsf{PCFC.FuncVer}(\mathsf{ck}, \tilde{\mathsf{com}}, \tilde{\mathsf{com}}_{\mathsf{out}}, f_C, \tilde{\pi}^{\mathcal{X}}) = 1$,
- $\mathsf{PCFC.Proj}(\mathsf{td}, \tilde{\mathsf{com}}) = \mathsf{PCFC.ProjCom}(\mathsf{td}, I_{i^*}, (\mathtt{x}_{i^*}, \mathtt{w}^*))$, and
- $\mathsf{PCFC.Proj}(\mathsf{td}, \tilde{\mathsf{com}}_{\mathsf{out}}) = \mathsf{PCFC.ProjCom}(\mathsf{td}, I_{i^*}, 1^{\ell_{x,w}} \cdot \tilde{y}_{i^*})$, $\tilde{y}_{i^*} = C(\mathtt{x}_{i^*}, \mathtt{w}^*) \ne 1$.

On the other hand, with probability $\Pr[\mathsf{Hyb}^2_{\mathcal{A}}(\lambda) = 1]$, it holds that

- $\mathsf{PCFC.FuncVer}(\mathsf{ck}, \mathsf{com}, \mathsf{com}_{\mathsf{out}}, f_C, \pi^{\mathcal{X}}) = 1$,
- $\mathsf{PCFC.Proj}(\mathsf{td}, \mathsf{com}) = \mathsf{PCFC.ProjCom}(\mathsf{td}, I_{i^*}, (\mathtt{x}_{i^*}, \mathtt{w}^*))$, and
- $\mathsf{PCFC.Proj}(\mathsf{td}, \tilde{\mathsf{com}}_{\mathsf{out}}) = \mathsf{PCFC.ProjCom}(\mathsf{td}, I_{i^*}, 1^{\ell_{x,w}})$.

In other words, $\mathcal{B}$ successfully wins the PCFC projective chain binding game with probability $\Pr[\mathsf{Hyb}^2_{\mathcal{A}}(\lambda) = 1]$. □

We collect the bounds of Theorem 6.29, Theorem 6.30, and Theorem 6.31, conclude that the advantage of $\mathcal{A}$ in the original Hyb^0 is bounded by

$$\Pr[\mathsf{Hyb}^0_{\mathcal{A}}(\lambda) = 1] \le \ell_{\mathtt{w}} \cdot \mathsf{Adv}^{\mathrm{se}}_{\mathsf{PCFC},\mathcal{B}}(\lambda) + \mathsf{Adv}^{\mathrm{pcb}}_{\mathsf{PCFC},\mathcal{B}}(\lambda).$$

□

6.6.2 Efficient Verification

Lemma 6.32. *Let* PCFC *and* BARG *be as defined in Theorem 6.25. If further* PCFC *has efficient verification with preprocessing (Theorem 6.14), then* BARG *has circuit-succinct verification with preprocessing (Theorem 6.5).*

Proof. As per Theorem 6.5, it suffices to show that $|\mathsf{vk}|$ and the runtime of $\mathsf{BARG.EffVer}(\lambda, \mathsf{vk})$ are at most $\mathsf{poly}(\lambda, \log k, \ell_{\mathtt{w}})$. We actually prove below a stronger claim that these quantities are at most $\mathsf{poly}(\lambda, \log k) \cdot \ell_{\mathtt{w}}$. Note that since $\ell = k \cdot \ell_{\mathtt{x,w}}$ and $\ell_{\mathtt{x}}$ and $\ell_{\mathtt{w}}$ are fixed $\mathsf{poly}(\lambda)$, we have $\mathsf{poly}(\lambda, \log \ell) = \mathsf{poly}(\lambda, \log k)$. Therefore, in the below we express all complexities in terms of ℓ instead of k.

We first examine $|\mathsf{vk}|$. Recall that $\mathsf{vk} = (\mathsf{com}_{\mathcal{G}_{\mathsf{stmt}}}, \mathsf{ck}_{f_C}, (\mathsf{ck}_{J_g})_{g \in \mathcal{G}_{\mathsf{wit}}})$. Note that by Theorem 3.1 fresh commitments are of size at most $\mathsf{poly}(\lambda, \log \ell)$ and by Theorem 3.5 aggregated commitments are identical to fresh ones. Therefore, $|\mathsf{com}_{\mathcal{G}_{\mathsf{stmt}}}| \le \mathsf{poly}(\lambda, \log \ell)$. Next, by Theorem 4.5 it holds that $|\mathsf{ck}_{f_C}| \le \mathsf{poly}(\lambda, \log \ell, \log m, \log |f_C|)$ where $m = 1$ in the current context and $|f_C| \le 2^{\mathsf{poly}(\lambda)}$. Thus, $|\mathsf{ck}_{f_C}| \le \mathsf{poly}(\lambda, \log \ell)$. Finally, by Theorem 6.12 it holds that $|\mathsf{ck}_{J_g}| \le \mathsf{poly}(\lambda, \log \ell, \log |J_g|)$ for each $g \in \mathcal{G}_{\mathsf{wit}}$, where $|J_g| = k$ and $|\mathcal{G}_{\mathsf{wit}}| = \ell_{\mathtt{w}}$. Putting everything together, we conclude that $|\mathsf{vk}| \le \mathsf{poly}(\lambda, \log k) \cdot \ell_{\mathtt{w}}$.

For the runtime of $\mathsf{BARG.EffVer}(\lambda, \mathsf{vk})$, by Theorem 3.5, the two calls to PCFC.Agg take time at most $\mathsf{poly}(\lambda, \log \ell) \cdot |\mathcal{G}_{\mathsf{wit}}| = \mathsf{poly}(\lambda, \log \ell) \cdot \ell_{\mathtt{w}}$ and $\mathsf{poly}(\lambda, \log \ell) \cdot 2$ respectively. Then, each of the $\ell_{\mathtt{w}}$ calls to PCFC.EffProjVer runs in time at most $\mathsf{poly}(\lambda, \log \ell, |J_g|)$ according to Theorem 6.12. Finally, the call to PCFC.EffFuncVer runs in time at most $\mathsf{poly}(\lambda, \log \ell, \log m, \log |f_C|)$ which as analysed above is at most $\mathsf{poly}(\lambda, \log \ell)$. We thus conclude that $\mathsf{BARG.EffVer}(\lambda, \mathsf{vk})$ runs in time at most $\mathsf{poly}(\lambda, \log \ell) \cdot \ell_{\mathtt{w}}$.

□

6.7 Algebraic PCFC from Bilinear Groups

In this section, we give an algebraic construction of a somewhere extractable projective chainable functional commitment scheme PCFC that is sufficient to instantiate Theorem 6.25, yielding a circuit-succinct BARG for circuit satisfiability.

Let $\mathcal{F}_C = \{C : \mathbb{F}^\ell \to \mathbb{F}^\ell : |C| \le \ell_C\}$ be the family of arithmetic circuits of size bounded by ℓ_C, where without loss of generality we let the input space be equal to the output space. We define the function family

$$\mathcal{F}_{k,\ell_C} = \{f : \mathbb{F}^{k\ell} \to \mathbb{F}^{k\ell}; f(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k) = (C_1(\boldsymbol{x}_1), \ldots, C_k(\boldsymbol{x}_k)) \ \wedge \ C_i \in \mathcal{F}_C \ \forall i \in [k]\}.$$

That is, $\mathcal{F}_{k,\ell_C}$ consists of all functions which input k vectors $(\boldsymbol{x}_i)_{i\in[k]}$ and evaluate some circuit $C_i \in \mathcal{F}_C$ on $\boldsymbol{x}_i$ for each $i \in [k]$.

The section is structured in two parts, where we incrementally construct PCFC using only generic operations over pairing groups. We first construct a somewhere extractable PC in Section 6.7.1 and then equip it with CFC algorithms in Section 6.7.2 turning it into a PCFC, while ensuring the compatibility of all properties.

The results of this section are summarised in the following theorem:

Theorem 6.33. *Let* $\mathsf{bgp} := (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, [1]_1, [1]_2)$ *be a pairing setting description, and let* $M = \mathbb{F} = \mathbb{Z}_q$. *Assume that the bilateral* $\mathsf{MDDH}_{\kappa,\ell,d}$ *assumption and the* $\mathsf{KerDH}_{\kappa,\ell}$ *assumption hold over* bgp *for some* $\kappa \geq 2$ *and* $\ell, d = \mathsf{poly}(\lambda)$. *Then, the construction* $\mathsf{PCFC} = (\mathsf{PC}, \mathsf{CFC})$ *is a somewhere extractable PCFC with the following properties:*

- *(Projective) aggregatable.*
- PC *is somewhere extractable for* $(\mathcal{I}, \mathcal{J}, \bar{\mathcal{M}})$, *where* $\bar{\mathcal{M}} = \{0, 1\}$ *and* $\mathcal{I}, \mathcal{J} \subseteq 2^{[\ell]}$ *are arbitrary families of sets such that* $|I \cap J| \leq 1$ *for every* $I \in \mathcal{I}, J \in \mathcal{J}$.
- CFC *is projective chain binding for* $(\mathcal{F}_{k,\ell_C}, \mathcal{I}')$ *where* $\mathcal{I}' = \{\{i(\ell-1)+1, \ldots, i\ell\} : i \in [k]\}$.
- *The commitment key size* $|\mathsf{ck}|$ *and the prover running time are* $O\left(\lambda \cdot \kappa^3 \cdot k^5 \cdot \ell_C^5\right)$.
- *Commitments have size* $|\mathsf{com}| = O(\lambda \cdot \kappa)$.
- *The* PC *subspace proofs have size* $\left|\pi^b\right| = O(\lambda \cdot \kappa^2)$.
- *The* PCFC *functional proofs have size* $\left|\pi^f\right| = O(\lambda \cdot \kappa^2)$.
- PCFC *efficient verification runs in time* $O(\lambda \cdot \kappa^2)$.

Proof. The proof follows from the constructions in Section 6.7.1 and Section 6.7.2 and from the series of lemmas proven there. We state the different properties: *a*) Binding follows from Theorem 6.35. *b*) Projection correctness follows from Theorem 6.36. *c*) Setup indistinguishability follows from both Theorem 6.37 and Theorem 6.48. *d*) Somewhere completeness follows from Theorem 6.38. *e*) Somewhere extractability follows from Theorem 6.39. *f*) Aggregatability and projective aggregatability follow from Theorem 6.41. *g*) Projective chain binding follows from Theorem 6.52. *h*) Efficient verification follows from both Theorem 6.42 and Theorem 6.50.

□

As κ is a small constant (e.g., $\kappa = 2$), the combination of Theorem 6.25 and Theorem 6.33 yields the following corollary:

Corollary 6.34. *Under the assumptions of Theorem 6.33, there exists an algebraic somewhere extractable BARG for NP such that:*

- *The setup size* $|\mathsf{crs}|$ *and the prover running time are* $O\left(\lambda \cdot k^5 \cdot \ell_C^5\right)$.
- *The proof size is* $|\pi| = O(\lambda \cdot \ell_{\mathtt{w}})$.
- *Efficient verification runs in time* $O(\lambda \cdot \ell_{\mathtt{w}})$.

6.7.1 Algebraic PC from Bilinear Groups

We introduce a construction of a somewhere extractable PC. Conceptually, our PC is an adaptation of the proof systems in the WW22 BARG [WW22] to work over a larger projective commitment key that is compatible with the WW24 FC proof systems [WW24b]. Notably, the projective space in our PC has dimension κ, as opposed to 1 in the commitments in the WW22 BARG. Therefore, our commitments are vectors of length 2κ instead of $\kappa+1$. Besides, our scheme explicitly supports subvector commitments, as well as their aggregation. We remark that all algorithms are well defined for any $J \in \mathcal{J} = 2^{[\ell]}$. The message space is defined as $\mathcal{M} = \mathbb{Z}_q = \mathbb{F}$ and the small message space as $\bar{\mathcal{M}} = \{0,1\}$. All vectors $\boldsymbol{x} \in \mathbb{F}^\ell$ are column vectors. As in Section 6.7.2 we borrow algorithms from the [WW24b] CFC, we follow their notation closely.

Our PC **construction.**

<u>$\mathsf{Setup}(1^\lambda, 1^\ell)$:</u>

- Generate a bilinear group description $\mathsf{bgp} := (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, [1]_1, [1]_2) \leftarrow \mathcal{BG}(1^\lambda)$, and let $\mathbb{F} := \mathbb{Z}_q$ and κ be the κ-Lin parameter.
- Sample random invertible $\hat{\mathbf{B}}, \check{\mathbf{B}} \leftarrow\$ \mathbb{F}^{2\kappa\times 2\kappa}$ and let $\hat{\mathbf{B}}^* = \hat{\mathbf{B}}^{-1}$ and $\check{\mathbf{B}}^* = \check{\mathbf{B}}^{-1}$.
- Split $\hat{\mathbf{B}} = \begin{bmatrix} \hat{\mathbf{B}}_1 \\ \hat{\mathbf{B}}_2 \end{bmatrix}$ where $\hat{\mathbf{B}}_1, \hat{\mathbf{B}}_2 \in \mathbb{F}^{\kappa\times 2\kappa}$ and split $\hat{\mathbf{B}}^* = \begin{bmatrix} \hat{\mathbf{B}}_1^* & \hat{\mathbf{B}}_2^* \end{bmatrix}$ where $\hat{\mathbf{B}}_1^*, \hat{\mathbf{B}}_2^* \in \mathbb{F}^{2\kappa\times\kappa}$. By construction, $\hat{\mathbf{B}}_1 \cdot \hat{\mathbf{B}}_1^* = \hat{\mathbf{B}}_2 \cdot \hat{\mathbf{B}}_2^* = \mathbf{I}_\kappa$ and $\hat{\mathbf{B}}_1 \cdot \hat{\mathbf{B}}_2^* = \hat{\mathbf{B}}_2 \cdot \hat{\mathbf{B}}_1^* = \mathbf{0}_\kappa$. Define $\check{\mathbf{B}}_1, \check{\mathbf{B}}_2 \in \mathbb{F}^{\kappa\times 2\kappa}$ and $\check{\mathbf{B}}_1^*, \check{\mathbf{B}}_2^* \in \mathbb{F}^{2\kappa\times\kappa}$ analogously.
- Sample $\hat{\mathbf{S}}_1, \check{\mathbf{S}}_1 \leftarrow\$ \mathbb{F}^{\kappa\times\ell}$, and define $\hat{\mathbf{T}} = \hat{\mathbf{B}}_1^* \cdot \hat{\mathbf{S}}_1 \in \mathbb{F}^{2\kappa\times\ell}$ and $\check{\mathbf{T}} = \check{\mathbf{B}}_1^* \cdot \check{\mathbf{S}}_1 \in \mathbb{F}^{2\kappa\times\ell}$. Let also $\hat{\boldsymbol{t}}_i \in \mathbb{F}^{2\kappa}$ be each of the column vectors in $\hat{\mathbf{T}}$, and similarly for $\check{\boldsymbol{t}}_i \in \mathbb{F}^{2\kappa}$ and $\check{\mathbf{T}}$.
- Finally, sample $\mathbf{R}_{i,j} \leftarrow\$ \mathbb{F}^{\kappa\times\kappa}$ and set

$$\hat{\mathbf{W}}_{i,j} \leftarrow -\hat{\mathbf{B}}_1^* \mathbf{R}_{i,j}^T, \quad \check{\mathbf{W}}_{i,j} \leftarrow \check{\mathbf{B}}_1^* (\check{s}_{1,i} \hat{s}_{1,j}^T + \mathbf{R}_{i,j}).$$

Output the following commitment key:

$$\mathsf{ck} := \left([\check{\mathbf{T}}]_1, [\hat{\mathbf{T}}]_2, [\check{\mathbf{B}}_1^*]_1, [\hat{\mathbf{B}}_1^*]_2, \{[\check{\mathbf{W}}_{i,j}]_1, [\hat{\mathbf{W}}_{i,j}]_2\}_{i,j\in[\ell]} \right).$$

Given any subset $J \subseteq [\ell]$, we let $\hat{\boldsymbol{t}}_J \leftarrow \sum_{i\in J} \hat{\boldsymbol{t}}_i$ and $\check{\boldsymbol{t}}_J \leftarrow \sum_{i\in J} \check{\boldsymbol{t}}_i$. Note that $\left[\check{\boldsymbol{t}}_J\right]_1, \left[\hat{\boldsymbol{t}}_J\right]_2$ can be computed from ck for any $J \subseteq [\ell]$.

<u>$\mathsf{ProjSetup}(1^\lambda, 1^\ell, I)$:</u> Let $\mathbf{P}_I \in \mathbb{F}^{\ell\times\ell}$ be the (diagonal) projection matrix associated to I, defined by $\mathbf{P}_{I i,i} = 1$ if $i \in I$ and $\mathbf{P}_{I i,j} = 0$ elsewhere. Sample all matrices as before, and additionally sample $\hat{\mathbf{S}}_2, \check{\mathbf{S}}_2 \leftarrow\$ \mathbb{F}^{\kappa\times\ell}$. Then, set

$$\hat{\mathbf{T}} \leftarrow \hat{\mathbf{B}}_1^* \cdot \hat{\mathbf{S}}_1 + \hat{\mathbf{B}}_2^* \cdot \hat{\mathbf{S}}_2 \cdot \mathbf{P}_I, \quad \check{\mathbf{T}} \leftarrow \check{\mathbf{B}}_1^* \cdot \check{\mathbf{S}}_1 + \check{\mathbf{B}}_2^* \cdot \check{\mathbf{S}}_2 \cdot \mathbf{P}_I$$

$$\hat{\mathbf{W}}_{i,j} \leftarrow \begin{cases} -\hat{\mathbf{B}}_1^* \mathbf{R}_{i,j}^T + \hat{\boldsymbol{t}}_i \check{s}_{1,j}^T & \text{if } \; j \in I \\ -\hat{\mathbf{B}}_1^* \mathbf{R}_{i,j}^T & \text{if } \; j \notin I \end{cases}, \quad \check{\mathbf{W}}_{i,j} \leftarrow \begin{cases} \check{\mathbf{B}}_1^* \mathbf{R}_{i,j} & \text{if } \; j \in I \\ \check{\mathbf{B}}_1^* \mathbf{R}_{i,j} + \check{\boldsymbol{t}}_i \hat{s}_{1,j}^T & \text{if } \; j \notin I \end{cases}.$$

Output the commitment key as before, and output $\mathsf{td} := (I, \check{\mathbf{B}}_2, \check{\mathbf{S}}_2, \hat{\mathbf{B}}_2, \hat{\mathbf{S}}_2)$.

$\underline{\mathsf{Com}(\mathsf{ck}, J, \boldsymbol{x})\text{:}}$ Output $[\hat{\boldsymbol{c}}]_2 \leftarrow \sum_{i \in J} \left[\hat{\boldsymbol{t}}_i\right]_2 \cdot x_i$.

$\underline{\mathsf{ProjCom}(\mathsf{td}, J, \boldsymbol{x})\text{:}}$ Output $\left[\hat{\boldsymbol{d}}\right]_2 \leftarrow \sum_{i \in I \cap J} [\hat{\boldsymbol{s}}_{2,i}]_2 \cdot x_i$.

$\underline{\mathsf{Proj}(\mathsf{td}, \mathsf{com})\text{:}}$ Parse $\hat{\mathbf{B}}_2$ from td, and parse com $= [\hat{\boldsymbol{c}}]_2$. Output $\left[\hat{\boldsymbol{d}}\right]_2 \leftarrow \hat{\mathbf{B}}_2 \cdot [\hat{\boldsymbol{c}}]_2$.

$\underline{\mathsf{ProjProve}(\mathsf{ck}, J, \boldsymbol{x})\text{:}}$

- Compute the twin commitment $[\check{\boldsymbol{c}}]_1 \leftarrow \sum_{i \in J} \left[\check{\boldsymbol{t}}_i\right]_1 x_i$.
- Compute the booleanity proofs:

$$\left[\check{\mathbf{\Pi}}_1\right]_1 \leftarrow \sum_{i,j \in J, i \neq j} \left[\check{\mathbf{W}}_{i,j}\right]_1 \cdot x_i(x_j - 1), \quad \left[\hat{\mathbf{\Pi}}_1\right]_2 \leftarrow \sum_{i,j \in J, i \neq j} \left[\hat{\mathbf{W}}_{i,j}\right]_2 \cdot x_i(x_j - 1),$$

$$\left[\check{\mathbf{\Pi}}_2\right]_1 \leftarrow \sum_{i,j \in J, i \neq j} \left[\check{\mathbf{W}}_{i,j}\right]_1 \cdot x_j(x_i - 1), \quad \left[\hat{\mathbf{\Pi}}_2\right]_2 \leftarrow \sum_{i,j \in J, i \neq j} \left[\hat{\mathbf{W}}_{i,j}\right]_2 \cdot x_j(x_i - 1).$$

- Output $\pi \leftarrow \left([\check{\boldsymbol{c}}]_1, \left[\check{\mathbf{\Pi}}_1\right]_1, \left[\hat{\mathbf{\Pi}}_1\right]_2, \left[\check{\mathbf{\Pi}}_2\right]_1, \left[\hat{\mathbf{\Pi}}_2\right]_2\right)$.

$\underline{\mathsf{ProjVer}(\mathsf{ck}, J, \mathsf{com}, \pi)\text{:}}$ Parse $\pi = \left([\check{\boldsymbol{c}}]_1, \left[\check{\mathbf{\Pi}}_1\right]_1, \left[\hat{\mathbf{\Pi}}_1\right]_2, \left[\check{\mathbf{\Pi}}_2\right]_1, \left[\hat{\mathbf{\Pi}}_2\right]_2\right)$ and com $= [\hat{\boldsymbol{c}}]_2$. Compute $\left[\check{\boldsymbol{t}}_J\right]_1, \left[\hat{\boldsymbol{t}}_J\right]_2$. Output 1 if and only if both of the following hold:

$$[\check{\boldsymbol{c}}]_1 \cdot \left[\hat{\boldsymbol{t}}_J^T\right]_2 = [\check{\boldsymbol{c}}]_1 \cdot \left[\hat{\boldsymbol{c}}^T\right]_2 + \left[\check{\mathbf{B}}_1^*\right]_1 \cdot \left[\hat{\mathbf{\Pi}}_1^T\right]_2 + \left[\check{\mathbf{\Pi}}_1\right]_1 \cdot \left[\hat{\mathbf{B}}_1^{*T}\right]_2. \tag{6.1}$$

$$\left[\check{\boldsymbol{t}}_J\right]_1 \cdot \left[\hat{\boldsymbol{c}}^T\right]_2 = [\check{\boldsymbol{c}}]_1 \cdot \left[\hat{\boldsymbol{c}}^T\right]_2 + \left[\check{\mathbf{B}}_1^*\right]_1 \cdot \left[\hat{\mathbf{\Pi}}_2^T\right]_2 + \left[\check{\mathbf{\Pi}}_2\right]_1 \cdot \left[\hat{\mathbf{B}}_1^{*T}\right]_2. \tag{6.2}$$

$\underline{\mathsf{PreProjVer}(\mathsf{ck}, J)\text{:}}$ Output $\mathsf{ck}_J \leftarrow \left(\left[\check{\boldsymbol{t}}_J\right]_1, \left[\hat{\boldsymbol{t}}_J\right]_2, \left[\check{\mathbf{B}}_1^*\right]_1, \left[\hat{\mathbf{B}}_1^{*T}\right]_2\right)$.

$\underline{\mathsf{EffProjVer}(\mathsf{ck}_J, \mathsf{com}, \pi)\text{:}}$ Check equations 6.1, 6.2 from ProjVer and output 1 if and only if both hold.

$\underline{\mathsf{ProjExt}(\mathsf{td}, J, \mathsf{com})\text{:}}$ Parse $(I, \hat{\mathbf{S}}_2)$ from td and compute $\left[\tilde{\boldsymbol{d}}\right]_2 \leftarrow \mathsf{Proj}(\mathsf{td}, \mathsf{com})$. Then, find a vector $\tilde{\boldsymbol{x}} \in \{0,1\}^{I \cap J}$ such that $\left[\tilde{\boldsymbol{d}}\right]_2 = \sum_{i \in I \cap J} [\hat{\boldsymbol{s}}_{2,i}]_2 \cdot \tilde{x}_i$. Output $\tilde{\boldsymbol{x}}$ if found. Otherwise, output $\perp$.

Note that this algorithm runs in time $O\left(2^{|I \cap J|} \cdot \mathsf{poly}(\lambda)\right)$.

$\underline{\mathsf{Agg}(\mathsf{ck}, (\mathsf{com}_J)_{J \in \mathcal{J}'}) \rightarrow \mathsf{com} :}$ Output $[\hat{\boldsymbol{c}}]_2 \leftarrow \sum_{J \in \mathcal{J}'} \left[\hat{\boldsymbol{c}}_J\right]_2$.

$\underline{\mathsf{ProjAgg}(\mathsf{ck}, (\mathsf{pcom}_J)_{J \in \mathcal{J}'}) \rightarrow \mathsf{pcom} :}$ Output $\left[\hat{\boldsymbol{d}}\right]_2 \leftarrow \sum_{J \in \mathcal{J}'} \left[\hat{\boldsymbol{d}}_J\right]_2$.

Proofs of the properties of PC. Next, we prove that PC satisfies all the properties required by Theorem 6.33. We remark that most of the properties and the proofs follow the blueprints of [WW22] with minor modifications. Essentially, we adapt their proofs to work over a projective space of dimension κ, as the commitments in the WW22 BARG implicitly have a projective space of dimension 1.

Lemma 6.35. *Assume that the* $\mathsf{KerDH}_{\kappa,\ell}$ *assumption holds over* bgp. *Then,* se-PC *is binding (Theorem 3.1).*

Proof. Let $\mathcal{A}(\mathsf{ck})$ be a PPT algorithm against binding that returns $\boldsymbol{x}, \boldsymbol{x}' \in \mathbb{F}^J$ such that $\boldsymbol{x} \neq \boldsymbol{x}'$ and $\mathsf{Com}(\mathsf{ck}, J, \boldsymbol{x}) = \mathsf{Com}(\mathsf{ck}, J, \boldsymbol{x}')$. Note that by construction $\left[\hat{\mathbf{T}}_J\right]_2 \cdot (\boldsymbol{x} - \boldsymbol{x}') = [\mathbf{0}]_2$.

We show how to use $\mathcal{A}$ to build an adversary $\mathcal{B}$ against $\mathsf{KerDH}_{\kappa,\ell}$. $\mathcal{B}$ receives $\left[\hat{\mathbf{S}}_1\right]_2 \in \mathbb{G}_2^{\kappa\times\ell}$ from its challenger, generates all other matrices involved in Setup including $\hat{\mathbf{B}}_1^* \in \mathbb{F}^{2\kappa\times\kappa}$, and generates a commitment key ck which sends to $\mathcal{A}$. $\mathcal{B}$ receives $\boldsymbol{x}, \boldsymbol{x}'$ from $\mathcal{A}(\mathsf{ck})$ and outputs the vector $\boldsymbol{u} = \boldsymbol{x} - \boldsymbol{x}' \neq \mathbf{0}$. We argue that $\boldsymbol{u}$ is in the kernel of $\left[\hat{\mathbf{S}}_1\right]_2$. Let $\boldsymbol{v} = \hat{\mathbf{S}}_1 \cdot \boldsymbol{u}$. Note that $\hat{\mathbf{B}}_1^*\boldsymbol{v} = \mathbf{0}$ if and only if $\boldsymbol{v} = \mathbf{0}$, as $\hat{\mathbf{B}}_1^*$ has maximal rank κ. Therefore, as $\left[\hat{\mathbf{T}}\right]_2 \boldsymbol{u} = \left[\hat{\mathbf{B}}_1^*\right]_2 \boldsymbol{v} = [\mathbf{0}]_2$, we conclude that $\boldsymbol{v} = \mathbf{0}$, and $\mathcal{B}$ successfully wins the $\mathsf{KerDH}_{\kappa,\ell}$ game. □

Lemma 6.36. se-PC *satisfies projection correctness (Theorem 6.7).*

Proof. Let $\boldsymbol{x} \in \mathbb{F}^J$, $J \subseteq [\ell]$, $I \subseteq [\ell]$ and $\mathsf{ck} \leftarrow \mathsf{ProjSetup}(1^\lambda, 1^\ell, I)$. Then,

$$\begin{aligned}
\mathsf{Proj}(\mathsf{td}, \mathsf{Com}(\mathsf{ck}, J, \boldsymbol{x})) &= \hat{\mathbf{B}}_2 \cdot \left[\sum_{i\in J} \hat{\boldsymbol{t}}_i \cdot x_i\right]_1 = \hat{\mathbf{B}}_2 \cdot \left[\hat{\mathbf{T}} \cdot \boldsymbol{x}\right]_1 \\
&= \hat{\mathbf{B}}_2 \cdot \left[\hat{\mathbf{B}}_1^* \cdot \hat{\mathbf{S}}_1 + \hat{\mathbf{B}}_2^* \cdot \hat{\mathbf{S}}_2 \cdot \mathbf{P}_I\right]_1 \cdot \boldsymbol{x} = \left[\hat{\mathbf{S}}_2 \cdot \mathbf{P}_I \cdot \boldsymbol{x}\right]_1 \\
&= \left[\sum_{i\in I\cap J} \hat{s}_{2,i} \cdot x_i\right]_1 .
\end{aligned}$$

Where the simplification follows by construction of the commitment key, as $\hat{\mathbf{B}}_2 \cdot \hat{\mathbf{B}}_2^* = \mathbf{I}_k$ and $\hat{\mathbf{B}}_2 \cdot \hat{\mathbf{B}}_1^* = \mathbf{0}_k$. □

Lemma 6.37. *Assume that the* $\mathsf{MDDH}_{\kappa,\ell,2\kappa}$ *assumption holds over* bgp. *Then,* PCFC *satisfies setup indistinguishability (Theorem 6.7).*

Proof. We define a series of game hops as follows. Note that we display each columnn vectors $\check{\boldsymbol{t}}_i, \hat{\boldsymbol{t}}_i$ of the matrices $\check{\mathbf{T}}, \hat{\mathbf{T}}$ separately.

- In Hyb^0, we have the key in normal mode, where

$$\hat{\boldsymbol{t}}_i \leftarrow \hat{\mathbf{B}}_1^* \hat{\boldsymbol{s}}_{1,i}, \quad \check{\boldsymbol{t}}_i \leftarrow \check{\mathbf{B}}_1^* \check{\boldsymbol{s}}_{1,i}.$$

$$\hat{\mathbf{W}}_{i,j} \leftarrow -\hat{\mathbf{B}}_1^* \mathbf{R}_{i,j}^T, \quad \check{\mathbf{W}}_{i,j} \leftarrow \check{\mathbf{B}}_1^* (\check{\boldsymbol{s}}_{1,i} \hat{\boldsymbol{s}}_{1,j}^T + \mathbf{R}_{i,j})$$

- In Hyb^1, we change the distribution of $\hat{\mathbf{W}}_{i,j}, \check{\mathbf{W}}_{i,j}$. Namely, we sample $\hat{\mathbf{S}}_2, \check{\mathbf{S}}_2 \leftarrow\!\!\$\ \mathbb{F}^{\kappa\times\ell}$ and set

$$\hat{\mathbf{W}}_{i,j} \leftarrow \begin{cases} -\hat{\mathbf{B}}_1^* \mathbf{R}_{i,j}^T + \hat{\boldsymbol{t}}_i \check{\boldsymbol{s}}_{1,j}^T & \text{if} \quad j \in I \\ -\hat{\mathbf{B}}_1^* \mathbf{R}_{i,j}^T & \text{if} \quad j \notin I \end{cases}, \quad \check{\mathbf{W}}_{i,j} \leftarrow \begin{cases} \check{\mathbf{B}}_1^* \mathbf{R}_{i,j} & \text{if} \quad j \in I \\ \check{\mathbf{B}}_1^* \mathbf{R}_{i,j} + \check{\boldsymbol{t}}_i \hat{\boldsymbol{s}}_{1,j}^T & \text{if} \quad j \notin I \end{cases}.$$

- In Hyb^2, we change the way that $\hat{\boldsymbol{t}}_i$ is sampled. Namely, we set $\hat{\mathbf{W}}_{i,j}, \check{\mathbf{W}}_{i,j}$ as in Hyb^1, sample $\hat{\boldsymbol{u}}_i \leftarrow\!\!\$\ \mathbb{F}^{2\kappa}$, and set

$$\hat{\boldsymbol{t}}_i \leftarrow \begin{cases} \hat{\boldsymbol{u}}_i & \text{if} \quad i \in I \\ \hat{\mathbf{B}}_1^* \cdot \hat{\boldsymbol{s}}_{1,i} & \text{if} \quad i \notin I \end{cases}, \quad \check{\boldsymbol{t}}_i \leftarrow \check{\mathbf{B}}_1^* \cdot \check{\boldsymbol{s}}_{1,i}$$

- In Hyb^3, we sample $\hat{\boldsymbol{t}}_i$ in trapdoor mode. Namely, we set $\hat{\mathbf{W}}_{i,j}, \hat{\mathbf{W}}_{i,j}$ as in Hyb^2, and then

$$\hat{\boldsymbol{t}}_i \leftarrow \begin{cases} \hat{\mathbf{B}}_1^* \cdot \hat{\boldsymbol{s}}_{1,i} + \hat{\mathbf{B}}_2^* \cdot \hat{\boldsymbol{s}}_{2,i} & \text{if} \quad i \in I \\ \hat{\mathbf{B}}_1^* \cdot \hat{\boldsymbol{s}}_{1,i} & \text{if} \quad i \notin I \end{cases}, \quad \check{\boldsymbol{t}}_i \leftarrow \check{\mathbf{B}}_1^* \cdot \check{\boldsymbol{s}}_{1,i}$$

- In Hyb^4, we change the way that $\check{\boldsymbol{t}}_i$ is sampled. Namely, we set $\hat{\mathbf{W}}_{i,j}, \hat{\mathbf{W}}_{i,j}$ as in Hyb^3, sample $\check{\boldsymbol{u}}_i \leftarrow\!\!\$\, \mathbb{F}^{2\kappa}$, and set

$$\hat{\boldsymbol{t}}_i \leftarrow \begin{cases} \hat{\mathbf{B}}_1^* \cdot \hat{\boldsymbol{s}}_{1,i} + \hat{\mathbf{B}}_2^* \cdot \hat{\boldsymbol{s}}_{2,i} & \text{if} \quad i \in I \\ \hat{\mathbf{B}}_1^* \cdot \hat{\boldsymbol{s}}_{1,i} & \text{if} \quad i \notin I \end{cases}, \quad \check{\boldsymbol{t}}_i \leftarrow \begin{cases} \check{\boldsymbol{u}}_i & \text{if} \quad i \in I \\ \check{\mathbf{B}}_1^* \cdot \check{\boldsymbol{s}}_{1,i} & \text{if} \quad i \notin I \end{cases}$$

- In Hyb^5, we sample the full key in trapdoor mode. Namely, we set $\hat{\mathbf{W}}_{i,j}, \hat{\mathbf{W}}_{i,j}$ as in Hyb^4, and set

$$\hat{\boldsymbol{t}}_i \leftarrow \begin{cases} \hat{\mathbf{B}}_1^* \cdot \hat{\boldsymbol{s}}_{1,i} + \hat{\mathbf{B}}_2^* \cdot \hat{\boldsymbol{s}}_{2,i} & \text{if} \quad i \in I \\ \hat{\mathbf{B}}_1^* \cdot \hat{\boldsymbol{s}}_{1,i} & \text{if} \quad i \notin I \end{cases}, \quad \check{\boldsymbol{t}}_i \leftarrow \begin{cases} \check{\mathbf{B}}_1^* \cdot \check{\boldsymbol{s}}_{1,i} + \check{\mathbf{B}}_2^* \cdot \check{\boldsymbol{s}}_{2,i} & \text{if} \quad i \in I \\ \check{\mathbf{B}}_1^* \cdot \check{\boldsymbol{s}}_{1,i} & \text{if} \quad i \notin I \end{cases}$$

We proceed to bound the advantage between the different game hops.

$\mathsf{Hyb}^0 \to \mathsf{Hyb}^1$: . This is a syntactic relabeling that follows from generating $\mathbf{R}_{i,j}$ as follows: Sample $\bar{\mathbf{R}}_{i,j} \leftarrow\!\!\$\, \mathbb{F}^{\kappa\times\kappa}$, and let

$$\mathbf{R}_{i,j} \leftarrow \begin{cases} \bar{\mathbf{R}}_{i,j} - \check{\boldsymbol{s}}_{1,i}\hat{\boldsymbol{s}}_{1,j}^T & \text{if} \quad j \in I \\ \bar{\mathbf{R}}_{i,j} & \text{if} \quad j \notin I \end{cases}$$

Clearly, $\mathbf{R}_{i,j}$ still follows a uniform distribution over $\mathbb{F}^{\kappa\times\kappa}$, and so the view of the adversary in both games is identical. Therefore,

$$\mathsf{Adv}^0_{\mathsf{se\text{-}PC},\mathcal{A}}(\lambda) \le \mathsf{Adv}^1_{\mathsf{se\text{-}PC},\mathcal{A}}(\lambda).$$

$\mathsf{Hyb}^1 \to \mathsf{Hyb}^2$: This follows by a sequence of $|I|$ sub-hybrids where we change each $\hat{\boldsymbol{t}}_j$ at a time. In each of the sub-hybrids, we carry out a reduction to $\mathsf{MDDH}_{\kappa,1,2\kappa}$ where the challenge matrix is $[\hat{\mathbf{B}}_1^*]_2$. We specify the first sub-hybrid, where we change the smallest index $i^* \in I$. Let $\mathcal{A}$ be an adversary against the first sub-hybrid.

- Let $\mathcal{B}$ be an adversary against $\mathsf{MDDH}_{\kappa,1,2\kappa}$ which receives a challenge matrix $[\hat{\mathbf{B}}_1^*]_2 \in \mathbb{G}_2^{2\kappa\times\kappa}$ and a challenge vector $[\boldsymbol{z}]_2 \in \mathbb{G}_2^{2\kappa}$.
- $\mathcal{B}$ generates a ck as follows (we specify only the modified terms):

$$[\hat{\boldsymbol{t}}_i]_2 \leftarrow \begin{cases} [\boldsymbol{z}]_2 & \text{if} \quad i = i^* \\ [\hat{\mathbf{B}}_1^*]_2 \cdot \hat{\boldsymbol{s}}_{1,i} & \text{if} \quad i \neq i^* \end{cases},$$

$$[\hat{\mathbf{W}}_{i,j}]_2 \leftarrow \begin{cases} -[\hat{\mathbf{B}}_1^*]_2 \mathbf{R}_{i,j}^T + [\hat{\boldsymbol{t}}_i]_2 \check{\boldsymbol{s}}_{1,j}^T & \text{if} \quad j = i^* \\ -[\hat{\mathbf{B}}_1^*]_2 \mathbf{R}_{i,j}^T & \text{if} \quad j \neq i^* \end{cases}.$$

Note, in particular, that the terms $\hat{\boldsymbol{s}}_{1,j}$ are always used in $\hat{\boldsymbol{t}}_i$ for $j \notin I$, and so $\check{\mathbf{W}}_{i,j}$ does never change its distribution.

- The distribution of the key is identical to Hyb^1 whenever $\boldsymbol{z} = \hat{\mathbf{B}}_1^* \cdot \hat{\boldsymbol{s}}_{1,i^*}$ as $\hat{\boldsymbol{s}}_{1,i^*}$ is sampled uniformly at random. On the other hand, it is identical to Hyb^2 whenever $\boldsymbol{z}$ is uniform.

Hence, $\mathcal{B}$ simply gives the key to $\mathcal{A}$ and outputs whatever $\mathcal{A}$ outputs. We conclude that

$$\mathsf{Adv}^1_{\mathsf{se\text{-}PC},\mathcal{A}}(\lambda) \leq \mathsf{Adv}^2_{\mathsf{se\text{-}PC},\mathcal{A}}(\lambda) + \mathsf{Adv}^{\mathsf{mddh}}_{\mathcal{B}}(\lambda).$$

$\mathsf{Hyb}^2 \longrightarrow \mathsf{Hyb}^3$**:** This is just a syntactic relabeling, as the distribution of $\left[\hat{\boldsymbol{t}}_i\right]_2$ is identical in both games. To see this, note that $\hat{\mathbf{B}}_1^*$ and $\hat{\mathbf{B}}_2^*$ together form a basis of the 2κ-dimensional vector space, and that $\hat{\boldsymbol{s}}_{1,i}, \hat{\boldsymbol{s}}_{2,i}$ are sampled at random. Hence,

$$\mathsf{Adv}^2_{\mathsf{se\text{-}PC},\mathcal{A}}(\lambda) \leq \mathsf{Adv}^3_{\mathsf{se\text{-}PC},\mathcal{A}}(\lambda).$$

$\mathsf{Hyb}^3 \longrightarrow \mathsf{Hyb}^4$**:** The argument is the same as in between Hyb^1 and Hyb^2, so we omit it. It follows that

$$\mathsf{Adv}^3_{\mathsf{se\text{-}PC},\mathcal{A}}(\lambda) \leq \mathsf{Adv}^4_{\mathsf{se\text{-}PC},\mathcal{A}}(\lambda) + \mathsf{Adv}^{\mathsf{mddh}}_{\mathcal{B}}(\lambda).$$

$\mathsf{Hyb}^4 \longrightarrow \mathsf{Hyb}^5$**:** The argument is the same as in between Hyb^2 and Hyb^3, so we omit it. It follows that

$$\mathsf{Adv}^4_{\mathsf{se\text{-}PC},\mathcal{A}}(\lambda) \leq \mathsf{Adv}^5_{\mathsf{se\text{-}PC},\mathcal{A}}(\lambda).$$

The lemma follows by collecting the different advantages.

□

Lemma 6.38. se-PC *satisfies somewhere completeness (Theorem 6.10).*

Proof. Let $I, J \subseteq [\ell]$ and $\boldsymbol{x} \in \{0,1\}^J$. Note that $x_i^2 = x_i$ for every $i \in J$. First, note that for any $i, j \in [\ell]$, the following equality holds

$$\check{\mathbf{B}}_1^* \cdot \hat{\mathbf{W}}_{i,j}^T + \check{\mathbf{W}}_{i,j} \cdot \hat{\mathbf{B}}_1^{*T} = -\check{\mathbf{B}}_1^* \cdot \mathbf{R}_{i,j} \cdot \hat{\mathbf{B}}_1^{*T} + \check{\mathbf{B}}_1^* \cdot (\check{\boldsymbol{s}}_{1,i}\hat{\boldsymbol{s}}_{1,j}^T + \mathbf{R}_{i,j}^T) \cdot \hat{\mathbf{B}}_1^{*T} = \check{\boldsymbol{t}}_i \hat{\boldsymbol{t}}_j^T$$

We start on the RHS of the first verification equation in the ProjVer algorithm.

$$\begin{aligned}
\check{\boldsymbol{c}} \cdot \hat{\boldsymbol{c}}^T + \hat{\mathbf{B}}_1^* \cdot \hat{\boldsymbol{\Pi}}_1^T + \check{\boldsymbol{\Pi}}_1 \cdot \mathbf{B}_1^{*T} = & \\
&= \sum_{i,j \in J} \check{\boldsymbol{t}}_i \hat{\boldsymbol{t}}_j^T x_i x_j + \sum_{i,j \in J, i \neq j} \left(\check{\mathbf{B}}_1^* \cdot \hat{\mathbf{W}}_{i,j}^T + \check{\mathbf{W}}_{i,j} \cdot \hat{\mathbf{B}}_1^{*T}\right) x_i (x_j - 1) \\
&= \sum_{i,j \in J} \check{\boldsymbol{t}}_i \hat{\boldsymbol{t}}_j^T x_i x_j + \sum_{i,j \in J, i \neq j} \check{\boldsymbol{t}}_i \hat{\boldsymbol{t}}_j^T x_i (x_j - 1) \\
&= \sum_{i \in J} \check{\boldsymbol{t}}_i \hat{\boldsymbol{t}}_i^T x_i^2 + \sum_{i,j \in J, i \neq j} \check{\boldsymbol{t}}_i \hat{\boldsymbol{t}}_j^T x_i \\
&= \left(\sum_{i \in J} \check{\boldsymbol{t}}_i x_i\right) \cdot \left(\sum_{j \in J} \hat{\boldsymbol{t}}_j^T\right) \\
&= \check{\boldsymbol{c}} \cdot \hat{\boldsymbol{t}}_J^T
\end{aligned}$$

□

Correctness for the second verification equation follows analogously.

Lemma 6.39. se-PC *satisfies statistical somewhere extractability (Theorem 6.10) for any sets $I \in \mathcal{I}$, $J \in \mathcal{I}$ such that $|I \cap J| = 1$.*

Proof. Let $I, J \subseteq [\ell]$ such that $I \cap J = \{i^*\}$. Our goal is to prove that the extraction is correct and matches the projection of com except with negligible probability against unbounded adversaries. The proof follows the blueprint of the proof of [WW22, Lemma 4.12], although the trapdoor setup in our construction differs. We start by proving the following lemma.

Lemma 6.40. *With respect to the matrices defined in* ProjSetup, *any vector $\boldsymbol{v} \in \mathbb{F}^{2\kappa}$ can be* uniquely *written as*

$$\boldsymbol{v} = \hat{\mathbf{B}}_1^* \cdot \hat{\boldsymbol{u}}_1 + \hat{\mathbf{B}}_2^* \cdot \hat{\boldsymbol{u}}_2, \quad \text{and} \quad \boldsymbol{v} = \check{\mathbf{B}}_1^* \cdot \check{\boldsymbol{u}}_1 + \check{\mathbf{B}}_2^* \cdot \check{\boldsymbol{u}}_2$$

for some vectors $\hat{\boldsymbol{u}}_1, \hat{\boldsymbol{u}}_2, \check{\boldsymbol{u}}_1, \check{\boldsymbol{u}}_2 \in \mathbb{F}^{\kappa}$.

Proof. Note that $\hat{\mathbf{B}}^* = \begin{bmatrix} \hat{\mathbf{B}}_1^* & \hat{\mathbf{B}}_2^* \end{bmatrix}$ where $\hat{\mathbf{B}}_1^*, \hat{\mathbf{B}}_2^* \in \mathbb{F}^{2\kappa \times \kappa}$ form a basis of $\mathbb{F}^{2\kappa \times 2\kappa}$. Hence, any vector can be written uniquely as a combination of 2κ basis vectors, i.e.,

$$\boldsymbol{v} = \begin{bmatrix} \hat{\mathbf{B}}_1^* & \hat{\mathbf{B}}_2^* \end{bmatrix} \cdot \begin{bmatrix} \hat{\boldsymbol{u}}_1 \\ \hat{\boldsymbol{u}}_2 \end{bmatrix}.$$

And similarly for the basis formed by $\check{\mathbf{B}}_1^*, \check{\mathbf{B}}_2^*$.

□

Let $\left([\check{\boldsymbol{c}}]_1, [\hat{\boldsymbol{c}}]_2, [\check{\boldsymbol{\Pi}}_1]_1, [\hat{\boldsymbol{\Pi}}_1]_2, [\check{\boldsymbol{\Pi}}_2]_1, [\hat{\boldsymbol{\Pi}}_2]_2\right)$ be commitments and proof terms that pass the two verification equations. Following the lemma, let $\check{\boldsymbol{u}}_1, \check{\boldsymbol{u}}_2, \hat{\boldsymbol{u}}_1, \hat{\boldsymbol{u}}_2 \in \mathbb{F}^{\kappa}$ be the unique vectors such that $\check{\boldsymbol{c}} = \check{\mathbf{B}}_1^* \cdot \check{\boldsymbol{u}}_1 + \check{\mathbf{B}}_2^* \cdot \check{\boldsymbol{u}}_2$ and $\hat{\boldsymbol{c}} = \hat{\mathbf{B}}_1^* \cdot \hat{\boldsymbol{u}}_1 + \hat{\mathbf{B}}_2^* \cdot \hat{\boldsymbol{u}}_2$. When we rewrite the following terms that appear in the verification relation in the above form, we obtain

$$\begin{aligned}
\check{\boldsymbol{c}} \cdot \hat{\boldsymbol{t}}_J^T &= \left(\check{\mathbf{B}}_1^* \cdot \check{\boldsymbol{u}}_1 + \check{\mathbf{B}}_2^* \cdot \check{\boldsymbol{u}}_2\right) \cdot \left(\sum_{i \in J} \hat{\mathbf{B}}_1^* \cdot \hat{\boldsymbol{s}}_{1,i} + \sum_{i \in I \cap J} \hat{\mathbf{B}}_2^* \cdot \hat{\boldsymbol{s}}_{2,i}\right)^T, \\
\check{\boldsymbol{c}} \cdot \hat{\boldsymbol{c}}^T &= \left(\check{\mathbf{B}}_1^* \cdot \check{\boldsymbol{u}}_1 + \check{\mathbf{B}}_2^* \cdot \check{\boldsymbol{u}}_2\right) \cdot \left(\hat{\mathbf{B}}_1^* \cdot \hat{\boldsymbol{u}}_1 + \hat{\mathbf{B}}_2^* \cdot \hat{\boldsymbol{u}}_2\right)^T, \\
\check{\boldsymbol{t}}_J \cdot \hat{\boldsymbol{c}}^T &= \left(\sum_{i \in J} \check{\mathbf{B}}_1^* \cdot \check{\boldsymbol{s}}_{1,i} + \sum_{i \in I \cap J} \check{\mathbf{B}}_2^* \cdot \check{\boldsymbol{s}}_{2,i}\right) \cdot \left(\hat{\mathbf{B}}_1^* \cdot \hat{\boldsymbol{u}}_1 + \hat{\mathbf{B}}_2^* \cdot \hat{\boldsymbol{u}}_2\right)^T.
\end{aligned}$$

Now, consider the two verification equations:

$$\begin{aligned}
[\check{\boldsymbol{c}}]_1 \cdot \left[\hat{\boldsymbol{t}}_J^T\right]_2 &= [\check{\boldsymbol{c}}]_1 \cdot \left[\hat{\boldsymbol{c}}^T\right]_2 + \left[\check{\mathbf{B}}_1^*\right]_1 \cdot \left[\hat{\boldsymbol{\Pi}}_1^T\right]_2 + \left[\check{\boldsymbol{\Pi}}_1\right]_1 \cdot \left[\hat{\mathbf{B}}_1^{*T}\right]_2, \\
\left[\check{\boldsymbol{t}}_J\right]_1 \cdot \left[\hat{\boldsymbol{c}}^T\right]_2 &= [\check{\boldsymbol{c}}]_1 \cdot \left[\hat{\boldsymbol{c}}^T\right]_2 + \left[\check{\mathbf{B}}_1^*\right]_1 \cdot \left[\hat{\boldsymbol{\Pi}}_2^T\right]_2 + \left[\check{\boldsymbol{\Pi}}_2\right]_1 \cdot \left[\hat{\mathbf{B}}_1^{*T}\right]_2.
\end{aligned}$$

If we now project each term to the projective subspace by multiplying each term by $\check{\mathbf{B}}_2$ on the left and $\hat{\mathbf{B}}_2^T$ on the right, we get

$$\check{\mathbf{B}}_2 \left[\check{\boldsymbol{c}}\right]_1 \cdot \left[\hat{\boldsymbol{t}}_J^T\right]_2 \hat{\mathbf{B}}_2^T = \check{\mathbf{B}}_2 \left[\check{\boldsymbol{c}}\right]_1 \cdot \left[\hat{\boldsymbol{c}}^T\right]_2 \hat{\mathbf{B}}_2^T,$$
$$\check{\mathbf{B}}_2 \left[\check{\boldsymbol{t}}_J\right]_1 \cdot \left[\hat{\boldsymbol{c}}^T\right]_2 \hat{\mathbf{B}}_2^T = \check{\mathbf{B}}_2 \left[\check{\boldsymbol{c}}\right]_1 \cdot \left[\hat{\boldsymbol{c}}^T\right]_2 \hat{\mathbf{B}}_2^T.$$

Where we have used that $\check{\mathbf{B}}_2 \cdot \check{\mathbf{B}}_1^* = \hat{\mathbf{B}}_2 \cdot \hat{\mathbf{B}}_1^* = \mathbf{0}_k$, and so the terms with a $\mathbf{B}_1^*$-type matrix are canceled out. Plugging in the expressions for the remaining terms, and using that $I \cap J = \{i^*\}$, we derive

$$\check{\boldsymbol{u}}_2 \cdot \hat{\boldsymbol{s}}_{2,i^*}^T = \check{\boldsymbol{u}}_2 \cdot \hat{\boldsymbol{u}}_2^T,$$
$$\check{\boldsymbol{s}}_{2,i^*} \cdot \hat{\boldsymbol{u}}_2^T = \check{\boldsymbol{u}}_2 \cdot \hat{\boldsymbol{u}}_2^T,$$

Where we have used that $\check{\mathbf{B}}_2 \cdot \check{\mathbf{B}}_2^* = \hat{\mathbf{B}}_2 \cdot \hat{\mathbf{B}}_2^* = \mathbf{1}_k$. For the above equations to hold, we must have that:

$$\check{\boldsymbol{u}}_2 \cdot (\hat{\boldsymbol{s}}_{2,i^*}^T - \hat{\boldsymbol{u}}_2^T) = \mathbf{0}_k = (\check{\boldsymbol{s}}_{2,i^*} - \check{\boldsymbol{u}}_2) \cdot \hat{\boldsymbol{u}}_2^T.$$

We analyze the first equality. As these are outer products, we have that, coordinate-wise, $(\check{u}_2)_i \cdot ((\hat{s}_{2,i^*})_j - (\hat{u}_2)_j) = 0$ for every $i, j \in [\kappa]$. Hence, we conclude that either $\check{\boldsymbol{u}}_2 = \mathbf{0}$, or $\hat{\boldsymbol{u}}_2 = \hat{\boldsymbol{s}}_{2,i^*}$. A similar analysis can be carried out for the second equation, which allows us to conclude that $\check{\boldsymbol{u}}_2 = \check{\rho} \cdot \check{\boldsymbol{s}}_{2,i^*}$ and that $\hat{\boldsymbol{u}}_2 = \hat{\rho} \cdot \hat{\boldsymbol{s}}_{2,i^*}$ for some $\check{\rho}, \hat{\rho} \in \{0,1\}$.

To conclude the proof, note that the extractor runs $\left[\hat{\boldsymbol{d}}\right]_1 \leftarrow \mathsf{Proj}(\mathsf{td}, \mathsf{com})$, which by definition of Proj and by the previous argument, is

$$\begin{aligned}\left[\hat{\boldsymbol{d}}\right]_2 &= \hat{\mathbf{B}}_2 \cdot [\hat{\boldsymbol{c}}]_2 \\ &= \hat{\mathbf{B}}_2 \cdot \left[\hat{\mathbf{B}}_1^* \cdot \hat{\boldsymbol{u}}_1 + \hat{\mathbf{B}}_2^* \cdot \hat{\boldsymbol{u}}_2\right]_2 \\ &= [\hat{\boldsymbol{u}}_2]_2 = \hat{\rho} \cdot [\hat{\boldsymbol{s}}_{2,i^*}]_2\end{aligned}$$

Hence, by construction of the extractor, the extracted value $x_{i^*} = \hat{\rho} \in \{0,1\}$. □

Lemma 6.41. se-PC *satisfies aggregatability (Theorem 3.5) and projective aggregatability (Theorem 6.8).*

Proof. Both properties follow from the algebraic structure of the commitments, which are linearly homomorphic. The proof is straightforward and is omitted. □

Lemma 6.42. se-PC *satisfies efficient verification (Theorem 6.12) where* EffProjVer *runs in time* $O(\lambda \cdot \kappa^2)$.

Proof. Follows directly from the descriptions of $\mathsf{PreProjVer}(\mathsf{ck}, J)$ and $\mathsf{EffProjVer}(\mathsf{ck}_J, \mathsf{com}, \pi)$. □

Remark 6.43. *Given $\mathcal{I} = \{\{i\} : i \in [\ell]\}$, it is straightforward to extend* se-PC *to build a PCFC for $\mathcal{I}$-separable quadratic functions that achieves functional extractability (Theorem 6.17). This is necessary to build a BARG via Theorem 6.20. The extension consists of adding CFC prove and verify algorithms that run as described in [WW22, Remark 4.16], and we refer to their work for the details. Then, the proof of functional extractability follows a similar pattern as the proof of Theorem 6.39.*

6.7.2 Algebraic PCFC from Bilinear Groups

We construct a (somewhere extractable) projective chainable functional commitment PCFC (Theorem 6.13) by augmenting the PC from the previous section with CFC algorithms. Our starting point, which we argue is the most natural choice given that it is the only fully-succinct algebraic FC in the literature, is the WW24 CFC on its chainable form [WW24b, Remark 5.18]. Unfortunately, this scheme does not satisfy projective chain binding (Theorem 6.19), which we require for our circuit-succinct compiler to a BARG.

The WW24 scheme has multiple internal algorithms for committing and proving relations on commitments. Our PCFC will also build on (two of) their algorithms, although we will introduce modifications towards achieving projective chain binding. We start with a description of the internals of WW24, where we already introduce some generalizations. We remark that these algorithms only serve the purpose of modularizing the scheme and the proofs. When the PCFC is used in a black-box way as in our compiler, they are hidden behind the PCFC interface.

Base commitment scheme. There are two families of setup algorithms in WW24: those that generate "base" commitment keys, and those that extend the commitment key with additional terms to enable a proof system. The so-called base setup algorithm $\mathsf{ck}_{\mathsf{base}} \leftarrow \mathsf{SetupBase}(1^\lambda, 1^\ell, 1^\ell_C)$ is parameterized by the maximum input length ℓ and the maximum circuit size ℓ_C. Such a setup supports arbitrary arithmetic circuits $C : \mathbb{F}^\ell \to \mathbb{F}^\ell$ that have at most ℓ_C arithmetic gates.[12] The algorithm generates two independent projective commitment keys. The type-I key $\left[\hat{\mathbf{T}}\right]_2 \in \mathbb{G}_2^{2\kappa\times\ell_C}$ intentionally coincides with the primary commitment key of PC, and its terms are always represented with a hat accent. The type-II key is defined by two random matrices $\mathbf{T}_A, \mathbf{T}_B \leftarrow\!\!\$\ \mathbb{F}^{2\kappa\times\ell_C}$. To commit to an input $\boldsymbol{x} \in \mathbb{F}^\ell$, and an output $\boldsymbol{y} \in \mathbb{F}^\ell$ via $\mathsf{PCFC.Com}$, one uses the type-I key on the set $[\ell]$ or $[\ell]$, respectively. The type-II key is used internally to enable interactions between the functional proof algorithms.

We remark that, similarly to our PC construction, both keys are projective. One can sample them in trapdoor mode by specifying two (independent) projection sets $I_1, I_2 \subseteq [\ell_C]$ for the type-I and type-II keys, respectively.[13] We denote this operation by $(\mathsf{ck}_{\mathsf{base}}, \mathsf{td}) \leftarrow \mathsf{ProjSetupBase}(1^\lambda, 1^\ell, 1^\ell_C, (I_1, I_2))$.

Functional proof systems. For functional openings, there are three proof systems that build on top of the type-I and type-II keys: prefix proofs (Pre), linear map proofs (Lin), and quadratic map proofs (Quad). We describe the last two in the generalized form we use in our construction, where the separability sets can be arbitrary and not necessarily prefixes. The prefix proof system is not needed in our construction, so we omit its details.

<u>Lin:</u> is a tuple of algorithms ($\mathsf{SetupLin}, \mathsf{OpenLin}, \mathsf{VerLin}$) that extend the base FC algorithms. $\mathsf{SetupLin}(\mathsf{ck}_{\mathsf{base}}, \mathcal{S})$ extends the base commitment key, requiring a separability set $\mathcal{S} \subseteq$

[12] We remind that, without loss of generality and to simplify the notation, we assume that the input space and the output space are the same.

[13] In the original WW24 scheme, I_1, I_2 are always prefix sets $[j_1], [j_2]$ for $j_1, j_2 \le \ell_C$.

$2^{[\ell_C]\times[\ell_C]}$ and outputting an extended ck. Let σ_1 be a type-I commitment to a vector $\boldsymbol{x} \in \mathbb{F}_C^\ell$ and let σ_2 be a type-II commitment to $y \in \mathbb{F}_C^\ell$. Let also $f : \mathbb{F}_C^\ell \to \mathbb{F}_C^\ell$ be a (I_1, I_2)-separable linear function such that $f(\boldsymbol{x}) = \boldsymbol{y}$. Then, $\mathsf{OpenLin}(\mathsf{ck}, \boldsymbol{x}, f)$ outputs a linear map proof π. The proof can be verified using $\mathsf{VerLin}(\mathsf{ck}, \sigma_1, \sigma_2, f, \pi)$. The proof system also admits algorithms ($\mathsf{PreVerLin}$, $\mathsf{EffVerLin}$) for efficient verification with pre-processing, which run in time $O(\lambda \cdot \kappa^2)$.

Its associated security notion is called *linear chain binding* (Theorem 6.44) and can be seen as a restriction of our projective chain binding (Theorem 6.19) for linear functions.[14] Given a trapdoored setup at $(I_1, I_2) \in \mathcal{S}$, it is hard for any PPT adversary to output two input commitments σ_1, σ_1', output commitments σ_2, σ_2', a (I_1, I_2)-local function f and proofs π, π' such that (1) both proofs verify, (2) $\mathsf{Proj}(\mathsf{td}, \sigma_1) = \mathsf{Proj}(\mathsf{td}, \sigma_1')$, and (3) $\mathsf{Proj}(\mathsf{td}, \sigma_2) \neq \mathsf{Proj}(\mathsf{td}, \sigma_2')$.

Quad: is a tuple of algorithms ($\mathsf{SetupQuad}$, $\mathsf{OpenQuad}$, $\mathsf{VerQuad}$). Its properties are identical to those of Lin, except that it admits (I_1, I_2)-separable quadratic functions $f : \mathbb{F}_C^\ell \to \mathbb{F}_C^\ell$, and except that the roles of σ_1 and σ_2 are reversed. Its security notion, *quadratic chain binding* (Theorem 6.45, which generalizes [WW24b, Definition 4.35]), is analogous to linear chain binding except that it considers quadratic functions.

We next introduce the formal definitions of our generalizations of linear chain binding and quadratic chain binding, as well as the corresponding definitions of projective matrices that we use in our construction.

Generalization of chain binding properties. For our generalizations of linear chain binding and quadratic chain binding, we refer to dual-type keys generated by $\mathsf{ProjSetupBase}$ as described before, where σ_1 is a type-I commitment and σ_2 is a type-II commitment. We also allow the adversary to have the trapdoor of the scheme, which is needed in our security reductions. Note that the definitions of linear and quadratic chain binding differ only on the ordering of the commitments.

Definition 6.44 (Generalized Linear Chain Binding for Lin). *Let $\mathcal{S}_{\mathsf{Lin}} \subseteq 2^{[\ell_C]\times[\ell_C]}$ and let $\mathcal{F}_{\mathsf{Lin}}^{\mathcal{S}}$ be the family of linear functions $f : \mathbb{F}_C^\ell \to \mathbb{F}_C^\ell$ such that f is $\mathcal{S}_{\mathsf{Lin}}$-separable. A PC augmented with* Lin *satisfies generalized linear chain binding if for any PPT adversary $\mathcal{A}$, for any $\ell \in \mathbb{N}$, and for any $(I_1, I_2) \in \mathcal{S}_{\mathsf{Lin}}$,*

$$\Pr\left[\begin{array}{c|l} & (\mathsf{ck}_{\mathsf{base}}, \mathsf{td}) \leftarrow \mathsf{ProjSetupBase}(1^\lambda, 1_C^\ell, (I_1, I_2)) \\ \mathsf{Proj}(\mathsf{td}, \sigma_2) & \mathsf{ck} \leftarrow \mathsf{SetupLin}(\mathsf{ck}_{\mathsf{base}}, \mathcal{S}_{\mathsf{Lin}}) \\ \neq \mathsf{Proj}(\mathsf{td}, \sigma_2') & (f, \sigma_1, \sigma_1', \sigma_2, \sigma_2', \pi, \pi') \leftarrow \mathcal{A}(\mathsf{ck}, \mathsf{td}) \\ \wedge\, f \in \mathcal{F}_{\mathsf{Lin}}^{\mathcal{S}} & \mathsf{VerLin}(\mathsf{ck}, \sigma_1, \sigma_2, f, \pi) = 1 \\ & \mathsf{VerLin}(\mathsf{ck}, \sigma_1', \sigma_2', f, \pi') = 1 \\ & \mathsf{Proj}(\mathsf{td}, \sigma_1) = \mathsf{Proj}(\mathsf{td}, \sigma_1') \end{array}\right] \leq \mathsf{negl}(\lambda)$$

[14]This is a generalization of the original chain binding notion [WW24b, Definition 4.20] where I_1, I_2 are required to be prefixes.

Definition 6.45 (Generalized Quadratic Chain Binding for Quad). *Let $\mathcal{S}_{\mathsf{Quad}} \subseteq 2^{[\ell_C]\times[\ell_C]}$ and let $\mathcal{F}^{\mathcal{S}}_{\mathsf{Quad}}$ be the family of quadratic functions $f : \mathbb{F}^{\ell}_C \to \mathbb{F}^{\ell}_C$ such that f is $\mathcal{S}_{\mathsf{Quad}}$-separable. A PC augmented with* Lin *satisfies generalized quadratic chain binding if for any PPT adversary $\mathcal{A}$, for any $\ell \in \mathbb{N}$, and for any $(I_1, I_2) \in \mathcal{S}_{\mathsf{Quad}}$,*

$$\Pr\left[\begin{array}{c|l} & (\mathsf{ck}_{\mathsf{base}}, \mathsf{td}) \leftarrow \mathsf{ProjSetup}(1^\lambda, 1^\ell_C, I_1, I_2) \\ \mathsf{Proj}(\mathsf{td}, \sigma_1) & \mathsf{ck} \leftarrow \mathsf{SetupQuad}(\mathsf{ck}_{\mathsf{base}}, \mathcal{S}_{\mathsf{Quad}}) \\ \neq \mathsf{Proj}(\mathsf{td}, \sigma'_1) & (f, \sigma_1, \sigma'_1, \sigma_2, \sigma'_2, \pi, \pi') \leftarrow \mathcal{A}(\mathsf{ck}, \mathsf{td}) \\ \wedge f \in \mathcal{F}^{\mathcal{S}}_{\mathsf{Quad}} & \mathsf{VerQuad}(\mathsf{ck}, \sigma_2, \sigma_1, f, \pi) = 1 \\ & \mathsf{VerQuad}(\mathsf{ck}, \sigma'_2, \sigma'_1, f, \pi') = 1 \\ & \mathsf{Proj}(\mathsf{td}, \sigma_2) = \mathsf{Proj}(\mathsf{td}, \sigma'_2) \end{array}\right] \leq \mathsf{negl}(\lambda)$$

Projection matrices. We describe the projection matrices $\mathbf{P}^{(\mathcal{S}_{\mathsf{Lin}})}, \mathbf{P}^{(\mathcal{S}_{\mathsf{Quad}})}$ for separable linear and quadratic functions below, which we adapt from [WW24b, Definitions 4.21, 4.36]. Recall that P_I are diagonal projection matrices such that $P_{I_{i,i}} = 1$ if $i \in I$ and $P_{I_{i,i}} = 0$ otherwise.

Definition 6.46 (Projection matrix for a separable linear function). *Let $\ell_C \in \mathbb{N}$ be an input length. For sets $I_1, I_2 \subseteq [\ell_C]$, we define the projection matrix $\mathbf{P}^{(I_1,I_2)}_{\mathsf{Lin}}$ to be*

$$\mathbf{P}^{(I_1,I_2)}_{\mathsf{Lin}} := \mathbf{I}_{\ell_C^2} - (\mathbf{I}_{\ell_C} - \mathbf{P}_{I_1}) \otimes \mathbf{P}_{I_2} \in \{0,1\}^{\ell_C^2 \times \ell_C^2}.$$

For a separability set $\mathcal{S} \subseteq 2^{[\ell_C]\times[\ell_C]}$, we define the linear projection matrix for $\mathcal{S}$ to be

$$\mathbf{P}^{(\mathcal{S})} := \prod_{(I_1,I_2)\in\mathcal{S}} \mathbf{P}^{(I_1,I_2)}_{\mathsf{Lin}} \in \{0,1\}^{\ell_C^2 \times \ell_C^2}.$$

Definition 6.47 (Projection matrix for a separable quadratic function). *Let $\ell_C \in \mathbb{N}$ be an input length. For sets $I_1, I_2 \in [\ell_C]$, we define the projection matrix $\mathbf{P}^{(I_1,I_2)}_{\mathsf{Quad}}$ to be*

$$\mathbf{P}^{(I_1,I_2)}_{\mathsf{Quad}} := \mathbf{I}_{\ell_C^3} - (\mathbf{I}_{\ell_C^2} - (\mathbf{P}_{I_1} \otimes \mathbf{P}_{I_1})) \otimes \mathbf{P}_{I_2} \in \{0,1\}^{\ell_C^3 \times \ell_C^3}.$$

For a separability set $\mathcal{S} \subseteq 2^{[\ell_C]\times[\ell_C]}$, we define the quadratic projection matrix for $\mathcal{S}$ to be

$$\mathbf{P}^{(\mathcal{S})} := \prod_{(I_1,I_2)\in\mathcal{S}} \mathbf{P}^{(I_1,I_2)}_{\mathsf{Quad}} \in \{0,1\}^{\ell_C^3 \times \ell_C^3}.$$

Our PCFC **construction.** Our PCFC is designed for the function family $\mathcal{F}_{k,\ell_C}$ defined in Theorem 6.33. We additionally define ProjSetup that takes two arguments (I_0, I_1, I_2) – a necessity for granularity in the projective chain binding proof. In our construction, the setup of Lin, Quad is modified with respect to WW24 to express functions in $\mathcal{F}_{k,\ell_C}$ accurately, such that there are no cross-wires between any two parallel "circuit blocks".

The construction essentially boils down to three commitment keys and four functional proofs (recall Figure 6.1 from the technical overview). The primary commitment key ck_0 is the same as in PC described in Section 6.7.1. Then, we rely on auxiliary commitment keys $\mathsf{ck}_1, \mathsf{ck}_2$ to realise the following functional proofs:

- A linear proof π_{in} for an input-projection map s that maps (the first ℓ coordinates of) the input commitment com under ck_0 to a commitment σ_2 under ck_2 to the circuit trace.
- A linear proof π_{Lin} between σ_1 (under ck_1) and σ_2 (under ck_2) for the identity function id.
- A quadratic proof π_{Quad} between σ_2 and σ_1 for a next-wire function w.
- A quadratic proof π_{out} for an output-projection map p that maps (the last ℓ coordinates of) σ_2 into (the first ℓ coordinates of) com_y.

<u>$\mathsf{Setup}(1^\lambda, 1^{k\ell})$:</u>

- Generate $\mathsf{ck}_0 \leftarrow \mathsf{PC.Setup}(1^\lambda, 1^{k\ell_C})$.
- Sample matrices $\mathbf{T}_1, \mathbf{T}_A, \mathbf{T}_B \leftarrow\!\!\$\, \mathbb{F}^{2\kappa \times \ell_C}$ at random. Let $\mathbf{T}_* \leftarrow \mathbf{T}_A \otimes \mathbf{T}_B \in \mathbb{F}^{4\kappa^2 \times \ell_C^2}$ and let $\mathsf{ck}_{\mathsf{base}} \leftarrow ([\mathbf{T}_1]_2, [\mathbf{T}_A]_1, [\mathbf{T}_A]_2, [\mathbf{T}_B]_2, [\mathbf{T}_*]_2)$
For convenience, we name $\mathsf{ck}' \leftarrow \left([\hat{\mathbf{T}}]_2, [\mathbf{T}_A]_1, [\mathbf{T}_A]_2, [\mathbf{T}_B]_2, [\mathbf{T}_*]_2\right) \subseteq \mathsf{ck}$. Note that the first term is (the type-I key) taken from ck_0 and the remaining terms correspond to the type-II key from $\mathsf{ck}_{\mathsf{base}}$.
- Define the following sets and families of sets:

$$\begin{aligned} I_{i,j} &= \{(i-1)\ell_C + 1, \ldots, (i-1)\ell_C + j\}, \\ \mathcal{S}_{\mathsf{in}} &= \{(I_{i,j}, I_{i,j}) : i \in [k], j \in [\ell]\}, \\ \mathcal{S}_{\mathsf{Lin}} &= \{(I_{i,j}, I_{i,j}) : i \in [k], j \in [\ell_C]\}, \\ \mathcal{S}_{\mathsf{Quad}} &= \{(I_{i,j}, I_{i,j+1}) : i \in [k], j \in [\ell_C - 1]\}, \\ \mathcal{S}_{\mathsf{out}} &= \{(I_{i,j+\ell_C-\ell}, I_{i,j}) : i \in [k], j \in [\ell]\}. \end{aligned}$$

- Run $\mathsf{SetupLin}(\mathsf{ck}', \mathcal{S}_{\mathsf{in}})$ as follows. Let $\mathbf{P}_{\mathsf{Lin}} = \mathbf{P}^{(\mathcal{S}_{\mathsf{Lin}})}$ defined as in Theorem 6.46. Then, run SetupLin as defined in [WW24b, Construction 4.23] with the above $\mathbf{P}_{\mathsf{Lin}}$. Obtain $\mathsf{ck}_{\mathsf{in}}$.
- Run $\mathsf{SetupLin}(\mathsf{ck}_{\mathsf{base}}, \mathcal{S}_{\mathsf{Lin}})$ as before but over all terms on $\mathsf{ck}_{\mathsf{base}}$. Obtain $\mathsf{ck}_{\mathsf{Lin}}$.
- Run $\mathsf{SetupQuad}(\mathsf{ck}_{\mathsf{base}}, \mathcal{S}_{\mathsf{Quad}})$ as follows. Let $\mathbf{P}_{\mathsf{Quad}} = \mathbf{P}^{(\mathcal{S}_{\mathsf{Quad}})}$ defined as in Theorem 6.47. Then, run SetupQuad as defined in [WW24b, Construction 4.38] with the above $\mathbf{P}_{\mathsf{Quad}}$. Obtain $\mathsf{ck}_{\mathsf{Quad}}$.
- Finally, run $\mathsf{SetupQuad}(\mathsf{ck}', \mathcal{S}_{\mathsf{out}})$ (again over ck', the of terms from ck_0 and $\mathsf{ck}_{\mathsf{base}}$), Obtain $\mathsf{ck}_{\mathsf{out}}$.

Output

$$\mathsf{ck} := \mathsf{ck}_0 \cup \mathsf{ck}_{\mathsf{base}} \cup \mathsf{ck}_{\mathsf{in}} \cup \mathsf{ck}_{\mathsf{Lin}} \cup \mathsf{ck}_{\mathsf{Quad}} \cup \mathsf{ck}_{\mathsf{out}}.$$

<u>$\mathsf{ProjSetup}(1^\lambda, 1^{k\ell}, I)$:</u> As $I \in \mathcal{I}'$ as defined in Theorem 6.33, we have that $I = \{i(\ell-1)+1, \ldots, i\ell\}$ for some $i \in [k]$.

Run $(\mathsf{ck}, \mathsf{td}_0, \mathsf{td}_1, \mathsf{td}_2) \leftarrow \mathsf{ProjSetup}(1^\lambda, 1^{k\ell}, (I_{i,\ell}, I_{i,\ell}, I_{i,\ell}))$ as defined below. Then, output $(\mathsf{ck}, \mathsf{td} = \mathsf{td}_0)$.

$\underline{\mathsf{ProjSetup}(1^\lambda, 1^{k\ell}, (I_0, I_1, I_2))\text{:}}$

- Run $(\mathsf{ck}_0, \mathsf{td}_0) \leftarrow \mathsf{PC.ProjSetup}(1^\lambda, 1^{k\ell_C}, I_0)$.
- Sample random $\mathbf{B}_1, \mathbf{B}_A, \mathbf{B}_B \leftarrow\!\!\$\ \mathbb{F}^{2\kappa\times 2\kappa}$ and let $\mathbf{B}^*_A = \mathbf{B}_A^{-1}$, $\mathbf{B}^*_B = \mathbf{B}_B^{-1}$ and $\mathbf{B}^*_1 = \mathbf{B}_1^{-1}$
- Split $\mathbf{B}_A = \begin{bmatrix} \mathbf{B}_{A,1} \\ \mathbf{B}_{A,2} \end{bmatrix}$ where $\mathbf{B}_{A,1}, \mathbf{B}_{A,2} \in \mathbb{F}^{\kappa\times 2\kappa}$. Similarly, split $\mathbf{B}^*_A = \begin{bmatrix} \mathbf{B}^*_{A,1} & \mathbf{B}^*_{A,2} \end{bmatrix}$ where $\mathbf{B}^*_{A,1}, \mathbf{B}^*_{A,2} \in \mathbb{F}^{2\kappa\times\kappa}$. Note that, by construction, $\mathbf{B}_{A,1}\cdot\mathbf{B}^*_{A,1} = \mathbf{B}_{A,2}\cdot\mathbf{B}^*_{A,2} = \mathbf{I}_\kappa$ and that $\mathbf{B}_{A,1}\cdot \mathbf{B}^*_{A,2} = \mathbf{B}_{A,2}\cdot\mathbf{B}^*_{A,1} = \mathbf{0}_\kappa$. Define $\mathbf{B}_{1,1}, \mathbf{B}_{1,2}, \mathbf{B}_{B,1}, \mathbf{B}_{B,2} \in \mathbb{F}^{\kappa\times 2\kappa}$ and $\mathbf{B}^*_{1,1}, \mathbf{B}^*_{1,2}, \mathbf{B}^*_{B,1}, \mathbf{B}^*_{B,2} \in \mathbb{F}^{2\kappa\times\kappa}$ analogously.
- Sample $\mathbf{S}_{1,1}, \mathbf{S}_{A,1}, \mathbf{S}_{B,1}, \mathbf{S}_{1,2}, \mathbf{S}_{A,2}, \mathbf{S}_{B,2} \leftarrow\!\!\$\ \mathbb{F}^{\kappa\times k\ell_C}$. Let $\mathbf{P}_{I_1}, \mathbf{P}_{I_2}$ be the projection matrices for the sets I_1, I_2, respectively.

$$\begin{aligned} \mathbf{T}_1 &\leftarrow \mathbf{B}^*_{1,1}\cdot\mathbf{S}_{1,1} + \mathbf{B}^*_{1,2}\cdot\mathbf{S}_{1,2}\cdot\mathbf{P}_{I_1}, \\ \mathbf{T}_A &\leftarrow \mathbf{B}^*_{A,1}\cdot\mathbf{S}_{A,1} + \mathbf{B}^*_{A,2}\cdot\mathbf{S}_{A,2}\cdot\mathbf{P}_{I_2}, \\ \mathbf{T}_B &\leftarrow \mathbf{B}^*_{B,1}\cdot\mathbf{S}_{B,1} + \mathbf{B}^*_{B,2}\cdot\mathbf{S}_{B,2}\cdot\mathbf{P}_{I_2}. \end{aligned}$$

- Finally, let $\mathbf{T}_* \leftarrow \mathbf{T}_A \otimes \mathbf{T}_B \in \mathbb{F}^{4\kappa^2\times\ell_C^2}$.

The remaining terms are computed from ck_0 and the above terms exactly as in Setup. We also define, as construction-specific trapdoors, $\mathsf{td}_1 \leftarrow \mathbf{B}_{1,2}$ and $\mathsf{td}_2 \leftarrow (\mathbf{B}_{A,2}, \mathbf{B}_{B,2})$. These trapdoors are linked to I_1, I_2 respectively and we make them explicit as they play a role in the security proof. However, they not used for extractability or projection when considering the scheme as a whole. Output the commitment key ck and trapdoors $(\mathsf{td}_0, \mathsf{td}_1, \mathsf{td}_2)$.

$\underline{\mathsf{FuncProve}(\mathsf{ck}, \boldsymbol{x}, f \in \mathcal{F}_{k,\ell_C})\text{:}}$

- Parse $f = (C_1, \ldots, C_k)$ and split $\boldsymbol{x} = (\boldsymbol{x}_i)_{i\in[k]}$. Evaluate $\boldsymbol{y}_i \leftarrow C_i(\boldsymbol{x}_i)$ and let $\boldsymbol{z}_i \in \mathbb{F}^\ell_C$ be the trace of the i-th computation, i.e., the values of all wires when $C_i(\boldsymbol{x}_i)$ is evaluated. Note that $\boldsymbol{z}_i = \boldsymbol{x}_i|\boldsymbol{z}_{i,C}|\boldsymbol{y}_i$ where $\boldsymbol{z}_{i,C} \in \mathbb{F}^{\ell_C - 2\ell}$.
- Commit to the computation trace of $\boldsymbol{z} \leftarrow \boldsymbol{z}_1|\cdots|\boldsymbol{z}_k$ in the type-I and type-II keys,

$$\sigma_1 \leftarrow [\mathbf{T}_1]_2\,\boldsymbol{z}, \quad \sigma_2 \leftarrow ([\mathbf{T}_A]_1\,\boldsymbol{z}, [\mathbf{T}_B]_2\,\boldsymbol{z}).$$

- Let $s_i : \mathbb{F}^\ell_C \to \mathbb{F}^\ell_C$ be given by $s(\boldsymbol{z}) = (z_1, \ldots, z_\ell, \mathbf{0}_{\ell_C-\ell})$ be a function that projects the first ℓ coordinates of $\boldsymbol{z}$ into the first ℓ coordinates. Define $s : \mathbb{F}^{k\ell_C} \to \mathbb{F}^{k\ell_C}$ to be the parallel evaluation of s_i, which is a $\mathcal{S}_{\mathsf{in}}$-separable linear function. Let now $\boldsymbol{z}_{\mathsf{in}} = \boldsymbol{x}_1|\mathbf{0}_{\ell_C-\ell_{\mathsf{x,w}}}|\cdots|\boldsymbol{x}_k|\mathbf{0}_{\ell_C-\ell_{\mathsf{x,w}}}$ be a concatenation of input vectors padded appropriately. Compute an input projection proof, given by $\pi_{\mathsf{in}} \leftarrow \mathsf{OpenLin}((\mathsf{ck}', \mathsf{ck}_{\mathsf{in}}), \boldsymbol{z}_{\mathsf{in}}, s)$.
- Compute a linear proof for consistency between σ_1 and σ_2. This proof is computed as $\pi_{\mathsf{Lin}} \leftarrow \mathsf{OpenLin}((\mathsf{ck}_{\mathsf{base}}, \mathsf{ck}_{\mathsf{Lin}}), \boldsymbol{z}, id)$ where $id : \mathbb{F}^{k\ell_C} \to \mathbb{F}^{k\ell_C}$ is the identity function. Note that id is $\mathcal{S}_{\mathsf{Lin}}$-separable.

- Let $w_i : \mathbb{F}_C^\ell \to \mathbb{F}_C^\ell$ be the "next wire function" of circuit C_i, such that the j-th output of $w_i(z_i)$ is the result of evaluating the j-th gate of C_i on the $j-1$ first inputs. Define $w : \mathbb{F}^{k\ell_C} \to \mathbb{F}^{k\ell_C}$ as $w(z_1, \ldots, z_k) = (w_1(z_1), \ldots, w_k(z_k))$. Note that w is a $\mathcal{S}_{\mathsf{Quad}}$-separable quadratic function. Compute a quadratic gate proof $\pi_{\mathsf{Quad}} \leftarrow \mathsf{OpenQuad}((\mathsf{ck}_{\mathsf{base}}, \mathsf{ck}_{\mathsf{Quad}}), z, w)$.
- Let $p_i : \mathbb{F}_C^\ell \to \mathbb{F}_C^\ell$ be given by $p(z) = (z_{\ell_C-\ell+1}, \ldots, z_{\ell_C}, \mathbf{0}_{\ell_C-\ell})$ be a function that projects the last ℓ coordinates of z into the first ℓ coordinates. Define $p : \mathbb{F}^{k\ell_C} \to \mathbb{F}^{k\ell_C}$ to be the parallel evaluation of p_i. Note that p is a $\mathcal{S}_{\mathsf{out}}$-separable linear (and therefore also quadratic) function. Compute an output projection proof from the type, given by $\pi_{\mathsf{out}} \leftarrow \mathsf{OpenQuad}((\mathsf{ck}', \mathsf{ck}_{\mathsf{out}}), z, p)$.

Output $\pi \leftarrow (\sigma_1, \sigma_2, \pi_{\mathsf{in}}, \pi_{\mathsf{Lin}}, \pi_{\mathsf{Quad}}, \pi_{\mathsf{out}})$.

<u>$\mathsf{FuncVer}(\mathsf{ck}, \mathsf{com}, \mathsf{com}_y, f, \pi)$:</u>

Parse $\pi = (\sigma_1, \sigma_2, \pi_{\mathsf{in}}, \pi_{\mathsf{Lin}}, \pi_{\mathsf{Quad}}, \pi_{\mathsf{out}})$ and output 1 if all the following checks pass:

- $\mathsf{VerLin}((\mathsf{ck}', \mathsf{ck}_{\mathsf{in}}), \mathsf{com}, \sigma_2, s, \pi_{\mathsf{in}})$.
- $\mathsf{VerLin}((\mathsf{ck}_{\mathsf{base}}, \mathsf{ck}_{\mathsf{Lin}}), \sigma_1, \sigma_2, id, \pi_{\mathsf{Lin}})$.
- $\mathsf{VerQuad}((\mathsf{ck}_{\mathsf{base}}, \mathsf{ck}_{\mathsf{Quad}}), \sigma_2, \sigma_1, w, \pi_{\mathsf{Quad}})$.
- $\mathsf{VerQuad}((\mathsf{ck}', \mathsf{ck}_{\mathsf{out}}), \sigma_2, \mathsf{com}_y, p, \pi_{\mathsf{out}})$.

<u>$\mathsf{PreFuncVer}(\mathsf{ck}, f)$:</u>

Run the preprocessing algorithms for s, id, w and p, respectively:

- $\mathsf{ck}_{f,\mathsf{in}} \leftarrow \mathsf{PreVerLin}((\mathsf{ck}', \mathsf{ck}_{\mathsf{in}}), s)$.
- $\mathsf{ck}_{f,\mathsf{Lin}} \leftarrow \mathsf{PreVerLin}((\mathsf{ck}_{\mathsf{base}}, \mathsf{ck}_{\mathsf{Lin}}), id)$.
- $\mathsf{ck}_{f,\mathsf{Quad}} \leftarrow \mathsf{PreVerQuad}((\mathsf{ck}_{\mathsf{base}}, \mathsf{ck}_{\mathsf{Quad}}), w)$.
- $\mathsf{ck}_{f,\mathsf{out}} \leftarrow \mathsf{PreVerQuad}((\mathsf{ck}', \mathsf{ck}_{\mathsf{out}}), p)$.

Output $\mathsf{ck}_f \leftarrow (\mathsf{ck}_{f,\mathsf{in}}, \mathsf{ck}_{f,\mathsf{Lin}}, \mathsf{ck}_{f,\mathsf{Quad}}, \mathsf{ck}_{f,\mathsf{out}})$

<u>$\mathsf{EffFuncVer}(\mathsf{ck}_f, \mathsf{com}, \mathsf{com}_y, \pi)$:</u>

Parse $\pi = (\sigma_1, \sigma_2, \pi_{\mathsf{in}}, \pi_{\mathsf{Lin}}, \pi_{\mathsf{Quad}}, \pi_{\mathsf{out}})$ and output 1 if all the following checks pass:

- $\mathsf{EffVerLin}(\mathsf{ck}_{f,\mathsf{in}}, \mathsf{com}, \sigma_2, \pi_{\mathsf{in}})$.
- $\mathsf{EffVerLin}(\mathsf{ck}_{f,\mathsf{Lin}}, \sigma_1, \sigma_2, \pi_{\mathsf{Lin}})$.
- $\mathsf{EffVerQuad}(\mathsf{ck}_{f,\mathsf{Quad}}, \sigma_2, \sigma_1, \pi_{\mathsf{Quad}})$.
- $\mathsf{EffVerQuad}(\mathsf{ck}_{f,\mathsf{out}}, \sigma_2, \mathsf{com}_y, \pi_{\mathsf{out}})$.

Lemma 6.48. *Assume that the* $\mathsf{MDDH}_{\kappa,k\ell_C,2\kappa}$ *holds over* bgp. *Then,* PCFC *satisfies setup indistinguishability.*

Proof. First, note that $\mathsf{ck}_{\mathsf{base}}$ satisfies setup indistinguishability as proven in Theorem 6.37 and [WW24b, Theorem 4.9]. Second, note that the extensions to the commitment key by the algorithms Lin and Quad can be sampled in a black-box way from $\mathsf{ck}_{\mathsf{base}}$ and do not vary in projective mode. Therefore, ck satisfies setup indistinguishability. □

Lemma 6.49. PCFC *satisfies CFC correctness (Theorem 4.8).*

Proof. The proof follows from [WW24b] as we rely on the correctness of Lin and Quad, and our modifications on the projective matrices (Theorems 6.46 and 6.47) do not affect any of their correctness properties. We omit the details. □

Lemma 6.50. PCFC *satisfies CFC efficient verification (Theorem 4.5) where* EffFuncVer *runs in time* $O(\lambda \cdot \kappa^2)$.

Proof. Follows from the descriptions of $\mathsf{PreFuncVer}(\mathsf{ck}, J)$ and $\mathsf{EffFuncVer}(\mathsf{ck}_J, \mathsf{com}, \pi)$, as well as from the efficient verification properties of the Lin and Quad proof systems. □

Before proving projective chain binding, we state a short lemma about the individual proof systems.

Lemma 6.51. *Let* Lin *and* Quad*, be as defined in [WW24b] and modified in our* PCFC *construction. Then,* Lin *satisfies generalized linear chain binding (Theorem 6.44) and* Quad *satisfies generalized quadratic chain binding (Theorem 6.45).*

Proof. The proof follows from [WW24b] and from our definitions of projective matrices. The main difference with respect to the original notions is that we allow the adversary to have the trapdoor of the commitment keys. We argue that this does not invalidate the security of the proof systems, as every computational step in the original proofs of linear chain binding ([WW24b, Theorem 4.25]) and quadratic chain binding ([WW24b, Theorem 4.39]) relies on a MDDH-type assumption whose challenge is on a matrix that is crafted in a black-box way from the original commitment key ck_0. More in detail, we have that for the proof of linear chain binding:

- The first computational step (Lemma 4.28) uses $\mathbf{A}$ as a Ker-DH challenge, which is sampled uniformly as $\mathbf{A} \leftarrow \mathbb{F}^{\kappa\times(\kappa+1)}$.
- The second computational step (Lemma 4.31) uses $(\hat{\mathbf{S}}_2, \mathbf{Z}_\alpha)$ as a MDDH challenge. $\hat{\mathbf{S}}_2$ is not included in td, and $\mathbf{Z}_\alpha$ is sampled black-box from ck_0.

For the proof of quadratic chain binding:

- The first computational step (Lemma 4.43) uses $\mathbf{A}$ as a Ker-DH challenge, exactly as before.
- The second computational step (Lemma 4.46) uses $(\mathbf{S}_{B,2}, \mathbf{Z})$ as a MDDH challenge. As before, $\mathbf{S}_{2,2}$ is not included in td, and $\mathbf{Z}$ is sampled black-box from ck_0.

□

Theorem 6.52. *Assume that the* $\mathsf{KerDH}_{\kappa,k\ell_C}$ *and the* $\mathsf{MDDH}_{\kappa,k\ell_C,2\kappa}$ *assumptions hold over* bgp*. Then, our construction* PCFC *satisfies projective chain binding for the class* $f \in \mathcal{F}_{k,\ell_C}$ *(c.f. Theorem 6.33).*

```
$\mathsf{Hyb}^{\mathsf{pcb}}_{\mathcal{A}}(\lambda)$:
$(\mathsf{ck}, \mathsf{td}_0, \mathsf{td}_1, \mathsf{td}_2) \leftarrow \mathsf{ProjSetup}(1^\lambda, 1^{k\ell_C}, (I_\ell, I_\ell, I_\ell))$
$(f, \mathsf{com}, \mathsf{com}', \mathsf{com}_y, \mathsf{com}'_y, \pi, \pi') \leftarrow \mathcal{A}(\mathsf{ck}, \mathsf{td}_0)$
$(\sigma_1, \sigma_2, \pi_{\mathsf{in}}, \pi_{\mathsf{Lin}}, \pi_{\mathsf{Quad}}, \pi_{\mathsf{out}}) \leftarrow \pi$
$(\sigma'_1, \sigma'_2, \pi'_{\mathsf{in}}, \pi'_{\mathsf{Lin}}, \pi'_{\mathsf{Quad}}, \pi'_{\mathsf{out}}) \leftarrow \pi'$
assert $\mathsf{FuncVer}(\mathsf{ck}, \mathsf{com}, \mathsf{com}_y, f, \pi) = 1$
assert $\mathsf{FuncVer}(\mathsf{ck}, \mathsf{com}', \mathsf{com}'_y, f, \pi') = 1$
assert $\mathsf{Proj}(\mathsf{td}_0, \mathsf{com}) = \mathsf{Proj}(\mathsf{td}_0, \mathsf{com}')$
assert $\mathsf{Proj}(\mathsf{td}_0, \mathsf{com}_y) \neq \mathsf{Proj}(\mathsf{td}_0, \mathsf{com}'_y)$
  // Checkpoint line
return 1

$\mathsf{Hyb}^{\mathsf{in}}_{\mathcal{A}}(\lambda)$:
  // as $\mathsf{Hyb}^{\mathsf{pcb}}_{\mathcal{A}}(\lambda)$ until "checkpoint line"
assert $\mathsf{Proj}(\mathsf{td}_2, \sigma_2) = \mathsf{Proj}(\mathsf{td}_2, \sigma'_2)$
return 1
```

Figure 6.6: *First set of games for the proof of Theorem 6.52. We highlight changes between games.*

Proof. We start by defining the projection sets that we will use in the proof

$$I_j = \{(i^* - 1)\ell_C + 1, \ldots, (i^* - 1)\ell_C + j\}.$$

Note that $|I_j| = j$. We also recall that $\pi = (\sigma_1, \sigma_2, \pi_{\mathsf{in}}, \pi_{\mathsf{Lin}}, \pi_{\mathsf{Quad}}, \pi_{\mathsf{out}})$. We define a sequence of hybrid games in Figure 6.6, Figure 6.7. The initial hybrid game, $\mathsf{Hyb}^{\mathsf{pcb}}_{\mathcal{A}}(\lambda)$, is the projective chain binding game from Theorem 6.19. Throughout the hybrid games, the adversary obtains the primary trapdoor td_0, but not the internal trapdoors $\mathsf{td}_1, \mathsf{td}_2$, which are only available to the challenger. This is crucial in the proof as if the adversary had $\mathsf{td}_1, \mathsf{td}_2$, setup indistinguishability for the corresponding subsets I_1, I_2 would not hold, and several game transitions rely on this property. On the other hand, the projection set I_0 is fixed throughout the entire sequence of games. We progress through the hybrids as follows.

$\mathsf{Hyb}^{\mathsf{pcb}}_{\mathcal{A}}(\lambda) \rightarrow \mathsf{Hyb}^{\mathsf{in}}_{\mathcal{A}}(\lambda)$**:** This step follows by the linear binding property of Lin (Theorem 6.51) for the relation in, i.e., the identity map between the base key (ck_0) and the type-II key commitments to the internal circuit wires. To see this, note that $\mathsf{ck}_{\mathsf{in}}$ is sampled on $\mathcal{S}_{\mathsf{in}}$ and $(I_\ell, I_\ell) \in \mathcal{S}_{\mathsf{in}}$. Next, we provide the details of the reduction for this game hop — the remaining steps follow an identical pattern and we will only give a proof sketch.

Let $\mathcal{A}$ be a PPT adversary that wins $\mathsf{Hyb}^{\mathsf{pcb}}_{\mathcal{A}}(\lambda)$ but does not win $\mathsf{Hyb}^{\mathsf{in}}_{\mathcal{A}}(\lambda)$. Then, we can construct a PPT adversary $\mathcal{B}_{\mathsf{in}}$ that wins $\mathsf{Hyb}^{\mathsf{in}}_{\mathcal{A}}(\lambda)$ as follows:

- $\mathcal{B}_{\mathsf{Lin}}$ receives the key $(\mathsf{ck}', \mathsf{ck}_{\mathsf{Lin}})$ and the trapdoor td_0.
- $\mathcal{B}_{\mathsf{Lin}}$ simulates the remaining part of the commitment key given $\mathsf{ck}', \mathsf{ck}_{\mathsf{Lin}}$ and generates a complete ck. Then, it calls $\mathcal{A}(\mathsf{ck}, \mathsf{td}_0)$ and receives the proofs π, π'. It parses the commitments $\mathsf{com}, \mathsf{com}', \sigma_2, \sigma'_2$ and the Lin proofs $\pi_{\mathsf{in}}, \pi'_{\mathsf{in}}$ from their respective proofs.
- $\mathcal{B}_{\mathsf{Lin}}$ simply outputs $(s, \mathsf{com}, \mathsf{com}', \sigma_2, \sigma'_2, \pi_{\mathsf{in}}, \pi'_{\mathsf{in}})$ to its challenger, where s is the identity function for the first ℓ coordinates (clearly $s \in \mathcal{F}^{\mathcal{S}}_{\mathsf{in}}$).

```
$\mathsf{Hyb}^{j,0}_{\mathcal{A}}(\lambda), \ell \leq j \leq \ell_C$:
$(\mathsf{ck}, \mathsf{td}_0, \mathsf{td}_1, \mathsf{td}_2) \leftarrow$
    $\mathsf{ProjSetup}(1^\lambda, 1^{k\ell_C}, (I_\ell, I_j, I_j))$
  // then, as $\mathsf{Hyb}^{\mathsf{pcb}}_{\mathcal{A}}(\lambda)$ until "checkpoint line"
assert $\mathsf{Proj}(\mathsf{td}_2, \sigma_2) = \mathsf{Proj}(\mathsf{td}_2, \sigma'_2)$
return 1
```

```
$\mathsf{Hyb}^{j,1}_{\mathcal{A}}(\lambda), \ell \leq j \leq \ell_C - 1$:
$(\mathsf{ck}, \mathsf{td}_0, \mathsf{td}_1, \mathsf{td}_2) \leftarrow$
    $\mathsf{ProjSetup}(1^\lambda, 1^{k\ell_C}, (I_\ell, I_{j+1}, I_j))$
  // then, as $\mathsf{Hyb}^{\mathsf{pcb}}_{\mathcal{A}}(\lambda)$ until "checkpoint line"
assert $\mathsf{Proj}(\mathsf{td}_2, \sigma_2) = \mathsf{Proj}(\mathsf{td}_2, \sigma'_2)$
return 1
```

```
$\mathsf{Hyb}^{j,2}_{\mathcal{A}}(\lambda), \ell \leq j \leq \ell_C - 1$:
$(\mathsf{ck}, \mathsf{td}_0, \mathsf{td}_1, \mathsf{td}_2) \leftarrow$
    $\mathsf{ProjSetup}(1^\lambda, 1^{k\ell_C}, (I_\ell, I_{j+1}, I_j))$
  // then, as $\mathsf{Hyb}^{\mathsf{pcb}}_{\mathcal{A}}(\lambda)$ until "checkpoint line"
assert $\mathsf{Proj}(\mathsf{td}_2, \sigma_2) = \mathsf{Proj}(\mathsf{td}_2, \sigma'_2)$
assert $\mathsf{Proj}(\mathsf{td}_1, \sigma_1) = \mathsf{Proj}(\mathsf{td}_1, \sigma'_1)$
return 1
```

```
$\mathsf{Hyb}^{j,3}_{\mathcal{A}}(\lambda), \ell \leq j \leq \ell_C - 1$:
$(\mathsf{ck}, \mathsf{td}_0, \mathsf{td}_1, \mathsf{td}_2) \leftarrow$
    $\mathsf{ProjSetup}(1^\lambda, 1^{k\ell_C}, (I_\ell, I_{j+1}, I_j))$
  // then, as $\mathsf{Hyb}^{\mathsf{pcb}}_{\mathcal{A}}(\lambda)$ until "checkpoint line"

assert $\mathsf{Proj}(\mathsf{td}_1, \sigma_1) = \mathsf{Proj}(\mathsf{td}_1, \sigma'_1)$
return 1
```

```
$\mathsf{Hyb}^{j,4}_{\mathcal{A}}(\lambda), \ell \leq j \leq \ell_C - 1$:
$(\mathsf{ck}, \mathsf{td}_0, \mathsf{td}_1, \mathsf{td}_2) \leftarrow$
    $\mathsf{ProjSetup}(1^\lambda, 1^{k\ell_C}, (I_\ell, I_{j+1}, I_{j+1}))$
  // then, as $\mathsf{Hyb}^{\mathsf{pcb}}_{\mathcal{A}}(\lambda)$ until "checkpoint line"

assert $\mathsf{Proj}(\mathsf{td}_1, \sigma_1) = \mathsf{Proj}(\mathsf{td}_1, \sigma'_1)$
return 1
```

```
$\mathsf{Hyb}^{j,5}_{\mathcal{A}}(\lambda), \ell \leq j \leq \ell_C - 1$:
$(\mathsf{ck}, \mathsf{td}_0, \mathsf{td}_1, \mathsf{td}_2) \leftarrow$
    $\mathsf{ProjSetup}(1^\lambda, 1^{k\ell_C}, (I_\ell, I_{j+1}, I_{j+1}))$
  // then, as $\mathsf{Hyb}^{\mathsf{pcb}}_{\mathcal{A}}(\lambda)$ until "checkpoint line"
assert $\mathsf{Proj}(\mathsf{td}_2, \sigma_2) = \mathsf{Proj}(\mathsf{td}_2, \sigma'_2)$
assert $\mathsf{Proj}(\mathsf{td}_1, \sigma_1) = \mathsf{Proj}(\mathsf{td}_1, \sigma'_1)$
return 1
```

```
$\mathsf{Hyb}^{\mathsf{out0}}_{\mathcal{A}}(\lambda)$:
$(\mathsf{ck}, \mathsf{td}_0, \mathsf{td}_1, \mathsf{td}_2) \leftarrow$
    $\mathsf{ProjSetup}(1^\lambda, 1^{k\ell_C}, (I_\ell, I_\ell, I_{\ell_C}))$
  // then, as $\mathsf{Hyb}^{\mathsf{pcb}}_{\mathcal{A}}(\lambda)$ until "checkpoint line"
assert $\mathsf{Proj}(\mathsf{td}_2, \sigma_2) = \mathsf{Proj}(\mathsf{td}_2, \sigma'_2)$
return 1
```

```
$\mathsf{Hyb}^{\mathsf{out1}}_{\mathcal{A}}(\lambda)$:
$(\mathsf{ck}, \mathsf{td}_0, \mathsf{td}_1, \mathsf{td}_2) \leftarrow$
    $\mathsf{ProjSetup}(1^\lambda, 1^{k\ell_C}, (I_\ell, I_\ell, I_{\ell_C}))$
  // then, as $\mathsf{Hyb}^{\mathsf{pcb}}_{\mathcal{A}}(\lambda)$ until "checkpoint line"
assert $\mathsf{Proj}(\mathsf{td}_2, \sigma_2) = \mathsf{Proj}(\mathsf{td}_2, \sigma'_2)$
assert $\mathsf{Proj}(\mathsf{td}_0, \mathsf{com}_y) = \mathsf{Proj}(\mathsf{td}_0, \mathsf{com}'_y)$
return 1
```

Figure 6.7: *Second set of games for the proof of Theorem 6.52. We highlight changes between games.*

It is evident that $\mathcal{B}_{\mathsf{Lin}}$ runs in polynomial time. If $\mathcal{A}$ wins in $\mathsf{Hyb}^{\mathsf{pcb}}_{\mathcal{A}}(\lambda)$ but not in $\mathsf{Hyb}^{\mathsf{in}}_{\mathcal{A}}(\lambda)$, the commitments $\mathsf{com}, \mathsf{com}'$ and proofs $\pi_{\mathsf{in}}, \pi'_{\mathsf{in}}$ output by $\mathcal{A}$ must satisfy:

- $\mathsf{Proj}(\mathsf{td}_0, \mathsf{com}) = \mathsf{Proj}(\mathsf{td}_0, \mathsf{com}')$,
- $\mathsf{Proj}(\mathsf{td}_2, \sigma_2) \neq \mathsf{Proj}(\mathsf{td}_2, \sigma'_2)$,
- $\mathsf{VerLin}((\mathsf{ck}', \mathsf{ck}_{\mathsf{in}}), \mathsf{com}, \sigma_2, s, \pi_{\mathsf{in}})$,
- $\mathsf{VerLin}((\mathsf{ck}', \mathsf{ck}_{\mathsf{in}}), \mathsf{com}', \sigma'_2, s, \pi'_{\mathsf{in}})$.

Hence, $\mathcal{B}_{\mathsf{Lin}}$ wins the generalized linear binding game (Theorem 6.44) if $\mathcal{A}$ is successful. We conclude:

$$\mathsf{Adv}^{0}_{\mathsf{PCFC},\mathcal{A}}(\lambda) \leq \mathsf{Adv}^{\mathsf{in}}_{\mathsf{PCFC},\mathcal{A}}(\lambda) + \mathsf{Adv}^{\mathrm{lbind}}_{\mathsf{Lin},\mathcal{B}_{\mathsf{Lin}}}(\lambda).$$

$\mathsf{Hyb}^{\mathsf{in}}_{\mathcal{A}}(\lambda) \to \mathsf{Hyb}^{\ell,0}_{\mathcal{A}}(\lambda)$: This is only a renaming step; the games are identical for the case $j = \ell$ that we consider. Therefore,

$$\mathsf{Adv}^{\mathsf{in}}_{\mathsf{PCFC},\mathcal{A}}(\lambda) = \mathsf{Adv}^{\ell,0}_{\mathsf{PCFC},\mathcal{A}}(\lambda).$$

$\mathsf{Hyb}^{j,0}_{\mathcal{A}}(\lambda) \to \mathsf{Hyb}^{j,1}_{\mathcal{A}}(\lambda)$: For any $j \in \{\ell, \ldots, \ell_C - 1\}$, this step readily follows by the setup indistinguishability property of PCFC (Theorem 6.48). We have that, for some PPT adversary $\mathcal{B}_{\mathsf{sind}}$,

$$\mathsf{Adv}^{\mathsf{j},0}_{\mathsf{PCFC},\mathcal{A}}(\lambda) \leq \mathsf{Adv}^{\mathsf{j},1}_{\mathsf{PCFC},\mathcal{A}}(\lambda) + \mathsf{Adv}^{\mathrm{sbind}}_{\mathsf{PCFC},\mathcal{B}_{\mathsf{sind}}}(\lambda).$$

$\mathsf{Hyb}^{j,1}_{\mathcal{A}}(\lambda) \to \mathsf{Hyb}^{j,2}_{\mathcal{A}}(\lambda)$: For any $j \in \{\ell, \ldots, \ell_C - 1\}$, this step follows by the generalized quadratic chain binding of Quad (Theorem 6.51), as $\mathsf{ck}_{\mathsf{Quad}}$ is sampled on $\mathcal{S}_{\mathsf{Quad}}$ and $(I_j, I_{j+1}) \in \mathcal{S}_{\mathsf{Quad}}$. We have that, for some PPT adversary $\mathcal{B}_{\mathsf{Quad}}$,

$$\mathsf{Adv}^{\mathsf{j},1}_{\mathsf{PCFC},\mathcal{A}}(\lambda) \leq \mathsf{Adv}^{\mathsf{j},2}_{\mathsf{PCFC},\mathcal{A}}(\lambda) + \mathsf{Adv}^{\mathrm{qbind}}_{\mathsf{Quad},\mathcal{B}_{\mathsf{Quad}}}(\lambda).$$

$\mathsf{Hyb}^{j,2}_{\mathcal{A}}(\lambda) \to \mathsf{Hyb}^{j,3}_{\mathcal{A}}(\lambda)$: This step is a simplification of $\mathsf{Hyb}^{j,2}_{\mathcal{A}}(\lambda)$ where we drop a verification condition on σ_2. Therefore,

$$\mathsf{Adv}^{\mathsf{j},2}_{\mathsf{PCFC},\mathcal{A}}(\lambda) \leq \mathsf{Adv}^{\mathsf{j},3}_{\mathsf{PCFC},\mathcal{A}}(\lambda).$$

$\mathsf{Hyb}^{j,3}_{\mathcal{A}}(\lambda) \to \mathsf{Hyb}^{j,4}_{\mathcal{A}}(\lambda)$: For any $j \in \{\ell, \ldots, \ell_C - 1\}$, this step again follows by the setup indistinguishability property of PCFC (Theorem 6.48). We have that, for some PPT adversary $\mathcal{B}_{\mathsf{sind}}$,

$$\mathsf{Adv}^{\mathsf{j},3}_{\mathsf{PCFC},\mathcal{A}}(\lambda) \leq \mathsf{Adv}^{\mathsf{j},4}_{\mathsf{PCFC},\mathcal{A}}(\lambda) + \mathsf{Adv}^{\mathrm{sbind}}_{\mathsf{PCFC},\mathcal{B}_{\mathsf{sind}}}(\lambda).$$

$\mathsf{Hyb}^{j,4}_{\mathcal{A}}(\lambda) \to \mathsf{Hyb}^{j,5}_{\mathcal{A}}(\lambda)$: For any $j \in \{\ell, \ldots, \ell_C - 1\}$, this step follows by the generalized linear chain binding of Lin (Theorem 6.51), as $\mathsf{ck}_{\mathsf{Lin}}$ is sampled on $\mathcal{S}_{\mathsf{Lin}}$ and $(I_j, I_j) \in \mathcal{S}_{\mathsf{Lin}}$. We have that, for some PPT adversary $\mathcal{B}_{\mathsf{Lin}}$,

$$\mathsf{Adv}^{\mathsf{j},4}_{\mathsf{PCFC},\mathcal{A}}(\lambda) \leq \mathsf{Adv}^{\mathsf{j},5}_{\mathsf{PCFC},\mathcal{A}}(\lambda) + \mathsf{Adv}^{\mathrm{lbind}}_{\mathsf{Lin},\mathcal{B}_{\mathsf{Lin}}}(\lambda).$$

$\mathsf{Hyb}^{j,5}_{\mathcal{A}}(\lambda) \to \mathsf{Hyb}^{j+1,0}_{\mathcal{A}}(\lambda)$: This step is a simplification of $\mathsf{Hyb}^{j,5}_{\mathcal{A}}(\lambda)$ where we drop a verification condition on σ_1. Therefore,

$$\mathsf{Adv}^{\mathrm{j},5}_{\mathsf{PCFC},\mathcal{A}}(\lambda) \leq \mathsf{Adv}^{\mathrm{j}+1,0}_{\mathsf{PCFC},\mathcal{A}}(\lambda).$$

Next, we bound the final sequence of games, starting from $\mathsf{Hyb}^{\ell_C,0}_{\mathcal{A}}(\lambda)$.

$\mathsf{Hyb}^{\ell_C,0}_{\mathcal{A}}(\lambda) \to \mathsf{Hyb}^{\mathsf{out0}}_{\mathcal{A}}(\lambda)$: In this game, we change the extraction sets from $(I_\ell, I_{\ell_C}, I_{\ell_C})$ to $(I_\ell, I_\ell, I_{\ell_C})$. Again, the step follows by the setup indistinguishability property of PCFC (Theorem 6.48). We have that, for some PPT adversary $\mathcal{B}_{\mathsf{sind}}$,

$$\mathsf{Adv}^{\ell_\mathrm{J},0}_{\mathsf{PCFC},\mathcal{A}}(\lambda) \leq \mathsf{Adv}^{\mathsf{out0}}_{\mathsf{PCFC},\mathcal{A}}(\lambda) + \mathsf{Adv}^{\mathrm{sind}}_{\mathsf{PCFC},\mathcal{B}_{\mathsf{sind}}}(\lambda).$$

$\mathsf{Hyb}^{\mathsf{out0}}_{\mathcal{A}}(\lambda) \to \mathsf{Hyb}^{\mathsf{out1}}_{\mathcal{A}}(\lambda)$: Finally, this step follows by the generalized quadratic chain binding of Quad (Theorem 6.51) for the output vector, as $\mathsf{ck}_{\mathsf{out}}$ is sampled on $\mathcal{S}_{\mathsf{out}}$ and $(I_\ell, I_{\ell_C}) \in \mathcal{S}_{\mathsf{out}}$. We have that, for some PPT adversary $\mathcal{B}_{\mathsf{Quad}}$,

$$\mathsf{Adv}^{\mathsf{out0}}_{\mathsf{PCFC},\mathcal{A}}(\lambda) \leq \mathsf{Adv}^{\mathsf{out1}}_{\mathsf{PCFC},\mathcal{A}}(\lambda) + \mathsf{Adv}^{\mathrm{qbind}}_{\mathsf{Quad},\mathcal{B}_{\mathsf{Quad}}}(\lambda).$$

$\mathsf{Hyb}^{\mathsf{out1}}_{\mathcal{A}}(\lambda)$: In this game, we have the following contradicting conditions:

- $\mathsf{Proj}(\mathsf{td}_1, \mathsf{com}_y) \neq \mathsf{Proj}(\mathsf{td}_1, \mathsf{com}'_y)$ (present since $\mathsf{Hyb}^{\mathsf{pcb}}_{\mathcal{A}}(\lambda)$), and
- $\mathsf{Proj}(\mathsf{td}_1, \mathsf{com}_y) = \mathsf{Proj}(\mathsf{td}_1, \mathsf{com}'_y)$.

Hence, the advantage of any adversary in this game is $\mathsf{Adv}^{\mathsf{out1}}_{\mathsf{PCFC},\mathcal{A}}(\lambda) = 0$.

The theorem follows by collecting all the bounds and noting the relevant computational assumptions that are used at each step. □

7 Fully-Succinct Multi-Key Homomorphic Signatures

In this chapter, we present *the first fully-succinct multi-key homomorphic signature that is secure under standard falsifiable assumptions*. The results are based on the article "Fully-Succinct Multi-Key Homomorphic Signatures from Standard Assumptions" [ABF24].

The chapter is structured as follows. In Section 7.1 we introduce a summary of contributions, followed by a technical overview in Section 7.2. In Section 7.3, we introduce multi-key homomorphic signatures and their properties. In Section 7.4, we describe our construction of a fully-succinct MKHS and prove its security. Finally, in Section 7.5,we extend our construction to support multi-hop evaluation and discuss several possible instantiations for our MKHS from diverse choices of primitives.

7.1 Contributions

Our construction of fully-succinct multi-key homomorphic signatures relies on a novel combination of standard digital signatures, succinct functional commitments (FC) [LRY16], and batch arguments for NP (BARG) [KPY19, CJJ21]. Our MKHS allows the evaluation of the same functions supported by the FC scheme, and inherits succinctness from the succinctness of the FC and of the BARG. We present a simplified version of our main theorem below.

Theorem 7.11 (simplified). *Let* FC *be a functional commitment scheme for a class of functions* $\mathcal{F}$*,* BARG *a somewhere-extractable batch argument for NP,* SEC *a somewhere extractable commitment, and* Σ *a digital signature scheme. Then, there exists an adaptively-secure multi-key homomorphic signature* MKHS *for* $\mathcal{F}$*. Moreover, if the* BARG *generates proofs of size* s_{BARG} *and the* FC *generates proofs of size* s_{FC}*, then the signatures produced by* MKHS *have size* $s_{\mathsf{MKHS}} \approx s_{\mathsf{FC}} + s_{\mathsf{BARG}}$.

Both BARGs and FCs have been in the spotlight in recent years and currently offer several instantiations from different (falsifiable) assumptions, which in turn yield MKHS for all functions from a variety of assumptions. For instance, we can instantiate our MKHS from building blocks based on correlation-intractable hash functions and probabilistic checkable proofs, such as the BARGs from [CJJ21, CJJ22, CGJ+23] and an FC for circuits based on the SNARG for P from [KLVW23], to obtain constructions from standard assumptions such as LWE or subexponential DDH. Alternatively, we can use the algebraic BARG of [WW22] based on the k-Lin assumption, and the algebraic FC from [BCFL23] based on the (falsifiable) HiKer assumption, obtaining a pairing-based construction for unbounded-depth circuits. We summarize these instantiations below.

Corollary 7.18 (simplified). *Assuming the hardness of either (1) subexponential DDH, or (2) learning with errors, there exists a multi-key homomorphic signature* MKHS *for boolean circuits of unbounded depth d with public parameter size* $\mathsf{poly}(\lambda, \log \ell)$ *and signature size* $\mathsf{poly}(\lambda, \log \ell) \cdot d$.

Corollary 7.19 (simplified). *Assuming the hardness of HiKer and k-Lin for $k \geq 2$, there exists a multi-key homomorphic signature* MKHS *for arithmetic circuits of unbounded depth d and bounded width w from algebraic building blocks, with public parameter size $O(w^5)$ and signature size $O(\lambda \cdot d^2) + \mathsf{poly}(\lambda)$.*

Alternatively, using the FC from [WW24b] one obtains $\mathsf{poly}(\lambda)$ sized signatures from the bilateral k-Lin assumption, with public parameters that grow with the circuit size as $O(|f|^5)$. It is possible to reduce the signature size further, to $O(\lambda^2)$, by also using the BARG from Chapter 6.

Additional Properties. Compared to the weakly succinct scheme of [FMNP16], our MKHS schemes achieve a variety of useful properties. First, we do not need to bound a priori the number of values to be signed, but only the class of functions (to the extent required by the FC); this feature is useful in applications where one computes on portions of very large data (e.g., sliding-window statistics on unbounded data streams). Second, our schemes are secure against adversaries that can adaptively corrupt users, whereas [FMNP16] can only handle non-adaptive corruptions. Third, our MKHS have efficient verification time, after preprocessing the function; this is similar to [FMNP16] though we support a more flexible preprocessing model (see Section 7.3.2). Furthermore, all our instantiations allow multi-hop sequential composition of different functions (Section 7.5.1) and can be compiled to provide context-hiding via a generic NIZK-based technique (Theorem 7.10).

Beyond our result for MKHS, the techniques underlying our construction present, to the best of our knowledge, a novel approach for building an advanced cryptographic primitive that was only known to be (with full succinctness) realizable from SNARKs. We expect that our techniques can be applied in other settings, leading to further constructions of advanced primitives from standard assumptions.

7.2 Technical Overview

Background: Labeled Programs. In a multi-key homomorphic signature scheme, the evaluator must declare the evaluated function as a *labeled program* [GW13]. A labeled program is specified by a tuple $(f, \ell_1, \ldots, \ell_\ell)$ where $f : \mathcal{M}^\ell \rightarrow \mathcal{M}^{\ell_y}$ is a function represented by an arithmetic or boolean circuit, and the τ_i are labels of the inputs. Without loss of generality, we assume that $\tau_i := (\mathsf{id}_i, \tau_i)$, where id_i is an identity and τ_i an arbitrary string.

Upon evaluating the homomorphic signature, the ℓ messages $m_1, \ldots, m_\ell \in \mathcal{M}$ that are collected by the evaluator are each uniquely associated to labels $\tau_1, \ldots, \tau_\ell$, and therefore to identities $\mathsf{id}_1, \ldots, \mathsf{id}_\ell$ (not necessarily all distinct). Additionally, each of these identities is associated to a public key vk_i, that can be used to verify the authenticity of a message-label pair (m_i, τ_i). Program labelling is required to properly define both correctness and security of MKHS, since e.g. otherwise the order in which the m_i's are input to f is unspecified.

Warm-Up: Aggregating Signatures with BARGs . The initial inspiration for our MKHS construction lies in the mechanism to construct aggregate signatures from Waters and Wu [WW22] which is based on BARGs for NP. In an *aggregate signature* scheme [BGLS03], an aggregator (or also evaluator) can take multiple message-signature pairs $(m_1, \sigma_1), \ldots, (m_\ell, \sigma_\ell)$ from different users, and compress all the signatures into a succinct σ_{Agg}.

To construct aggregate signatures, [WW22] start from any digital signature scheme $\Sigma = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Ver})$. Then, to sign messages, aggregate signatures, and verify the aggregation proof, their algorithm broadly proceeds as follows:

- Aggregate: To aggregate ℓ message-signature pairs, $(m_1, \sigma_1), \ldots, (m_\ell, \sigma_\ell)$, let $\mathtt{x}_i = (m_i, \mathsf{vk}_i)$ be the statements and $\mathtt{w}_i = \sigma_i$ be the witnesses for the circuit $C(\mathtt{x}_i, \mathtt{w}_i)$ that checks:

  $$\Sigma.\mathsf{Ver}(\mathsf{vk}_i, m_i, \sigma_i) = 1$$

  Then, the aggregate signature σ_{Agg} is a BARG proof on $(C, \{\mathtt{x}_i\}, \{\mathtt{w}_i\})$.
- Verify: To verify σ_{Agg}, one runs the BARG verification algorithm on $(C, \{\mathtt{x}_i\})$.

A mechanism to aggregate signatures can be seen as the "first step" of the evaluation of a fully-fledged MKHS. Indeed, similarly to aggregate signatures, in MKHS one also needs to prove that all signatures and messages are valid in a succinct manner. However, while in aggregate signatures the verifier knows all the messages $(m_1, \ldots, m_\ell)$, in MKHS one only knows the result $\boldsymbol{y} = f(m_1, \ldots, m_\ell)$. Therefore the evaluation step must additionally prove in a succinct manner the correctness of f's computation. This is what makes the realization of fully succinct MKHS challenging. A natural attempt to deal with this problem is to extend the BARG circuit by placing m_i in the witness, and by additionally proving that $\boldsymbol{y} = f(m_1, \ldots, m_\ell)$. An example of such a language could be the following:

$$\Sigma.\mathsf{Ver}(\mathsf{vk}_i, m_i | \tau_i, \sigma_i) = 1 \ \wedge \ \boldsymbol{y} = f(m_1, \ldots, m_\ell),$$

where $\mathtt{x}_i = (\tau_i, \mathsf{vk}_i, f, \boldsymbol{y})$ and $\mathtt{w}_i = (m_i, \sigma_i)$. Unfortunately, the computation of f can be "global", i.e., involve messages from all the witnesses, and thus the language is not compatible with that supported by BARGs.

Proving $f(m_1, \ldots, m_\ell)$: Functional Commitments. To address the problem of f being global, our second idea is to resort to *functional commitments* (FCs). As described in Chapter 4, a functional commitment scheme [LRY16] allows an entity to first commit to some input $\boldsymbol{x}$ in com, and then open com to $f(\boldsymbol{x})$ for some function $f \in \mathcal{F}$, where $\mathcal{F}$ is the class of functions supported by the scheme. Importantly for our goal, FCs ensure commitments and openings to be succinct and can be realized from falsifiable assumptions.

By using FCs, the evaluator could create a commitment com to the inputs of the computation $(m_1, \dots, m_\ell)$ and then use the opening feature to prove the evaluation $\boldsymbol{y} = f(m_1, \dots, m_\ell)$. This way we can take this "global" task outside of the BARG. Unfortunately, doing two separate proofs, one for the BARG and one for the FC, does not suffice. The issue is that there is no connection between the m_i committed inside the FC and the messages whose signatures are verified in the BARG circuit C, i.e., in $\Sigma.\mathsf{Ver}(\mathsf{vk}_i, m_i|\tau_i, \sigma_i)$. To integrate FCs and BARGs into a working solution, we need to be able to link the commitment com to the messages m_i that are in the witnesses $\mathtt{w}_i$ of C. One natural example of such a connection may consist of showing that, at every local step i, com opens to message m_i at position i (i.e., à la vector commitment), a local check that could be easily integrated in the circuit $C(\mathtt{x}_i, \mathtt{w}_i)$ and proven with a BARG. This approach, while giving correctness, is unsuccessful for the security proof. At a very high level, the MKHS adversary produces a forgery which contains a commitment com^* and a functional-opening π^* to $\boldsymbol{y}^* \neq f(m_1, \dots, m_\ell)$. To break the security of the FC we would need to come up with another functional-opening to a different value, say the honest output $f(m_1, \dots, m_\ell)$. The reduction could compute this by itself if we had the guarantee that com^* is a commitment to $(m_1, \dots, m_\ell)$ but this is not ensured; we can only use the BARG to extract, for a single index i at a time, a position-opening to a validly signed m_i at position i in com^*. This is however not enough to break the evaluation binding of the FC.

Our Solution: Proving FC updates in the BARG. To get around the above problem, our approach consists of *iteratively computing* com *inside the BARG circuit*. We start by defining a sequence of partial commitments $\mathsf{com}_0, \dots, \mathsf{com}_\ell$, where the i-th commitment commits to the first i messages. Namely, let $\mathsf{com}_i \leftarrow \mathsf{FC.Com}(\mathsf{ck}, (m_1, \dots, m_i, 0, \dots, 0))$. Then, at step i of the BARG proof, $C(\mathtt{x}_i, \mathtt{w}_i)$ verifies a proof π_i that com_i and com_{i-1} only differ on m_i at position i. In other words, that if we update com_{i-1} with m_i at position i, then we obtain com_i.

For this idea to work, we require two properties from our FC. One, *determinism*, such that we can compare commitments without having to open them. Two, *local updatability*, such that there exists an efficient update verification algorithm $\mathsf{FC.VerUpd}$ that runs in constant (or at least sublinear) time in ℓ. Moreover, update verification should only require a succinct section ck_i of the commitment key. We describe a simplified version of the resulting BARG circuit in Figure 7.1.

Given the description of C, our construction of MKHS can be summarized as follows:

- Sign: To sign a message m_i with label τ_i under key sk_i, compute and output $\sigma_i \leftarrow \Sigma.\mathsf{Sign}(\mathsf{sk}_i, m_i|\tau_i)$.
- Evaluate: To evaluate $(f, \tau_1, \dots, \tau_\ell)$ on ℓ message-signature pairs $(m_1, \sigma_1), \dots, (m_\ell, \sigma_\ell)$, compute:
  - An FC commitment $\mathsf{com} \leftarrow \mathsf{FC.Com}(\mathsf{ck}, (m_1, \dots, m_\ell))$.
  - A BARG proof π_σ for $C(\mathtt{x}_1, \mathtt{w}_1) \wedge \cdots \wedge C(\mathtt{x}_\ell, \mathtt{w}_\ell)$.
  - An FC opening proof π_f that com opens to $\boldsymbol{y} = f(m_1, \dots, m_\ell)$ on f.

Description of $C(\mathtt{x}, \mathtt{w})$ (simplified):
Statement: $\mathtt{x} = (\mathsf{vk}_i, \mathsf{ck}_i, \tau_i, i)$
Witness: $\mathtt{w} = (m_i, \sigma_i, \pi_i, \mathsf{com}_{i-1}, \mathsf{com}_i)$
Circuit:

- If $i = 1$, check that $\mathsf{com}_{i-1} = \mathsf{FC.Com}(\mathsf{ck}, \mathbf{0})$.
- If $i = \ell$, check that $\mathsf{com}_i = \mathsf{com}$.
- Check that:

$$\begin{aligned} &\Sigma.\mathsf{Ver}(\mathsf{vk}_i, m_i | \tau_i, \sigma_i) = 1 \\ \wedge\ &\mathsf{FC.VerUpd}(\mathsf{ck}_i, i, \mathsf{com}_{i-1}, 0, \mathsf{com}_i, m_i, \pi_i) = 1 \end{aligned}$$

Figure 7.1: *Simplified description of the BARG circuit C in our MKHS construction. The commitment* com *is hardwired into the circuit.*

Then, the output signature is $\sigma_{f,y} = (\mathsf{com}, \pi_\sigma, \pi_f)$.

- Verify: To verify $\sigma_{f,y}$, simply check the BARG and FC proofs w.r.t. com.

We note that our actual construction in Section 7.4 is slightly more complex, as it additionally involves a somewhere extractable commitment scheme (SEC) which we require to connect the consecutive steps $i-1$ and i of the BARG and for the security proof to go through.

Security and Proof Strategy. The security notion for MKHS considers adversaries that can make signing queries for messages and labels of their choice and it captures that it should be hard for the adversary to (1) claim valid messages and signatures that were never received from the signing oracle, and (2) forge the output of the computation of the labeled program $(f, \tau_1, \ldots, \tau_\ell)$. The notion is adaptive as the adversary may arbitrarily expose parties' secret keys, yet compromised keys cannot be involved in a forgery.

Our security proof proceeds by partitioning the winning condition in multiple events, according to the type of forgery that is produced by the adversary, and then handles each event separately. The most interesting component of the proof, and arguably the hardest technical challenge of this work, is to deal with the event when the adversary produces a forgery for $\boldsymbol{y} \neq f(m_1, \ldots, m_\ell)$, where the (deterministic) commitment to the messages com^* output by $\mathcal{A}$ is dishonest, $\mathsf{com}^* \neq \mathsf{FC.Com}(\mathsf{ck}, (m_1, \ldots, m_\ell))$.

To bound the probability of this event, the general proof strategy is to show that all partial commitments com_i for $i \in [\ell]$ must have been computed honestly. We define multiple hybrids for each index i, which implement a "sliding window" strategy[1] where we roughly: (1) extract from both the BARG and the SEC at step i, (2) compare the extracted com_i to their honest counterparts, and (3) extract the message m_i and signature σ_i (a potential forgery) from the adversary's output, such that we can certify the validity of the i-th update. Then, we "reboot" the BARG and SEC extraction and start again at step

[1] This strategy is similar than the one followed by the proof of security of our PCFC in Chapter 6

$i + 1$. From the above proof strategy, steps (1) and (2) follow the blueprint of a line of work on succinct delegation schemes (also known as SNARGs for P) [KPY19, GZ21, KVZ21, CJJ22, KLVW23], whereas step (3) requires to go a few steps beyond. Notably, in contrast to delegation schemes where the proven computation is deterministic, in our MKHS scheme the statement includes a non-deterministic part, messages and signatures, that are not available to the verifier.

7.3 Multi-Key Homomorphic Signatures

In this section, we recall the definition of Multi-Key Homomorphic Signatures (MKHS) [FMNP16].

As explained above, a MKHS allows each signer to sign a set of messages $\{m_{\mathsf{id},i}\}$ so that an evaluator can compute a function f on messages signed by different users and to produce a signature that certifies the correctness of the result. Since the verifier does not see the original inputs one must carefully define what does it mean that a value y is the correct output of a function f on *some* signed messages. Following the work of Gennaro and Wichs [GW13] on (single-key) homomorphic authenticators, even in the multi-key setting one can use the notion of *labeled programs*. Informally speaking, this means that a user id signs each message $m_{\mathsf{id},i}$ along with a "tag" tg_i and, in the MKHS case, her identity id. The pair $\tau_i = (\mathsf{id}, \mathsf{tg}_i)$ is called the "label" and is a unique identifier of the signed message. To verify an output y, one checks the signature not only w.r.t. the function f but also with the labels (ℓ_i) of its inputs—what is called a labeled program $\mathcal{P}$. This way, a successful verification of the tuple $(\mathcal{P} = (f, \tau_1, \dots, \tau_\ell), y, \sigma_{f,y})$ means that y is the correct output of f on some messages signed by the corresponding user with label $\tau_1, \dots, \tau_\ell$ respectively.

Definition 7.1 (Labeled Programs for MKHS [FMNP16])**.** *A labeled program $\mathcal{P}$ is a tuple $(f, \tau_1, \dots, \tau_\ell)$ such that $f : \mathcal{M}^\ell \to \mathcal{M}^{\ell_y}$ is a function of ℓ variables (e.g., a circuit) and $\tau_i \in \mathcal{L}$ is a label for the i-th input of f. Let $f_{id} : \mathcal{M} \to \mathcal{M}$ be the identity function and $\tau \in \mathcal{L}$ be any label. We denote by $\mathcal{I}_\tau = (f_{id}, \tau)$ the identity program with label τ. Labeled programs can be composed as follows: given $\mathcal{P}_1, \dots, \mathcal{P}_k$ and a function $g : \mathcal{M}^t \to \mathcal{M}^{\ell_y}$, the composed program, denoted $\mathcal{P}^* = g(\mathcal{P}_1, \dots, \mathcal{P}_k)$, is the one obtained by evaluating g on the collection of t outputs of $\mathcal{P}_1, \dots, \mathcal{P}_k$. The labeled inputs of $\mathcal{P}^*$ are the distinct labeled inputs of $\mathcal{P}_1, \dots \mathcal{P}_k$, where inputs with the same label are converted to a single input. A program $\mathcal{P} = (f, \tau_1, \dots, \tau_\ell)$ can be expressed as the composition of ℓ identity programs, i.e., $\mathcal{P} = f(\mathcal{I}_{\tau_1}, \dots, \mathcal{I}_{\tau_\ell})$.*

In MKHS, each label τ is a pair $(\mathsf{id}, \mathsf{tg})$ where $\mathsf{id} \in \mathcal{ID}$ is a user's identity and $\mathsf{tg} \in \mathcal{T}$ is a tag; thus the label space is $\mathcal{L} = \mathcal{ID} \times \mathcal{T}$. We denote $\mathsf{id} \in \mathcal{P}$ if there is at least one label of the program $\mathcal{P}$ with identity id, i.e., for $\mathcal{P} = (f, \tau_1, \dots, \tau_\ell)$, $\mathsf{id} \in \mathcal{P}$ iff there exists $\tau_i = (\mathsf{id}_i, \mathsf{tg}_i)$ such that $\mathsf{id}_i = \mathsf{id}$.

Definition 7.2 (Multi-Key Homomorphic Signature)**.** *Let $\mathcal{F}$ be a family of functions, $\mathcal{ID}$ an identity space, and $\mathcal{T}$ a tag space. A Multi-Key Homomorphic Signature scheme for a family* *of functions $\mathcal{F}$, identity space $\mathcal{ID}$, and tag space $\mathcal{T}$ is a tuple of algorithms* MKHS = (Setup, KeyGen, Sign, Eval, Ver) *such that:*

Setup($1^\lambda, \mathcal{F}, \mathcal{ID}, \mathcal{T}$) $\longrightarrow$ pp**:** *On input the security parameter λ and descriptions of $\mathcal{F}, \mathcal{ID}, \mathcal{T}$, the setup algorithm outputs public parameters* pp*. We assume* pp *to be an input of all subsequent algorithms, even if not specified.*

KeyGen(pp) $\longrightarrow$ (sk, vk)**:** *On input the public parameters* pp*, the key generation algorithm outputs a* secret *signing key* sk *and a* public *verification key* vk*.*

Sign(sk, m, τ) $\longrightarrow \sigma$**:** *On input a signing key* sk*, a label $\tau = (\mathsf{id}, \mathsf{tg}) \in \mathcal{L}$, and a message $m \in \mathcal{M}$, the signing algorithm outputs a signature σ.*

Eval($f, (\mathcal{P}_i, \{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id} \in \mathcal{P}_i}, m_i, \sigma_i)_{i \in [\ell]}$) $\longrightarrow \sigma_{f,y}$**:** *Given a function $f \in \mathcal{F}$ with ℓ inputs, and for each input i a triple consisting of a labeled program $\mathcal{P}_i$, the set of corresponding verification keys $\{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id} \in \mathcal{P}_i}$, a message m_i and a signature σ_i, the evaluation algorithm outputs a new signature $\sigma_{f,y}$.*

Ver($\mathcal{P}, \{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id} \in \mathcal{P}}, \boldsymbol{y}, \sigma_{f,y}$) $\longrightarrow b$**:** *On input a labeled program $\mathcal{P} = (f, \tau_1, \dots, \tau_\ell)$, the set of verification keys $\{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id} \in \mathcal{P}}$ of the users involved in $\mathcal{P}$, a value $\boldsymbol{y} \in \mathcal{M}^{\ell_y}$, and a signature $\sigma_{f,y}$, the verification algorithm outputs 0 (reject) or 1 (accept).*

A MKHS scheme should have authentication and evaluation correctness. The former says that a freshly generated signature on (τ, m) verifies correctly for m as the output of the identity program $\mathcal{I}_\tau$.

Definition 7.3 (Authentication correctness)**.** *For all public parameters* pp $\longleftarrow$ Setup($1^\lambda, \mathcal{F}, \mathcal{ID}, \mathcal{T}$), *keypair* (sk, vk) $\longleftarrow$ KeyGen(pp)*, label $\tau \in \mathcal{L}$, message $m \in \mathcal{M}$, and identity program $\mathcal{I}_\tau$, if* $\sigma \longleftarrow$ Sign(sk, m, τ) *then* Ver($\mathcal{I}_\tau$, vk, m, σ) = 1 *holds with overwhelming probability.*

Evaluation correctness instead says, roughly, that running Eval with a function f on a tuple of valid signatures produces a new valid signature for the output. We consider two classes of MKHS schemes: *single-hop* and *multi-hop*. Single-hop MKHS are schemes where Eval can only be executed on signatures produced by Sign. In this case, evaluation correctness ensures that, given a function f and signatures $(\sigma_1, \dots, \sigma_\ell)$ such that each σ_i verifies for m_i as the output of $\mathcal{I}_{\tau_i}$, Eval produces a signature that verifies for $\boldsymbol{y} = f(m_1, \dots, m_\ell)$ as the output of the labeled program $\mathcal{P} = (f, \tau_1, \dots, \tau_\ell)$.

Definition 7.4 (Single-Hop Evaluation correctness)**.** *Consider any public parameters* pp $\longleftarrow$ Setup($1^\lambda, \mathcal{F}, \mathcal{ID}, \mathcal{T}$)*, any set $\{(\mathsf{vk}_i, \sigma_i, m_i, \tau_i)\}_{i \in [\ell]}$ such that, for every $i \in [\ell]$, vk_i is honestly generated and* Ver($\mathcal{I}_{\tau_i}$, $\mathsf{vk}_i, m_i, \sigma_i$) = 1*, and any function $f \in \mathcal{F}$. If $\boldsymbol{y} = f(m_1, \dots, m_\ell)$, $\mathcal{P} = (f, \tau_1, \dots, \tau_\ell)$, and $\sigma_{f,y}$ =* Eval($f, (\mathcal{I}_{\tau_i}, \mathsf{vk}_i, m_i, \sigma_i)_{i \in [\ell]}$) *then* Ver($\mathcal{P}, \{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id} \in \mathcal{P}}, \boldsymbol{y}, \sigma_{f,y}$) = 1 *with overwhelming probability.*

Multi-hop MKHS instead allow to execute Eval on signatures produced by previous executions of Eval. In this case, evaluation correctness ensures that, given a function f and triples $(\sigma_1, \dots, \sigma_\ell)$ such that each σ_i verifies for m_i as the output of $\mathcal{I}_{\tau_i}$, Eval produces a signature that verifies for $\boldsymbol{y} = f(m_1, \ldots, m_\ell)$ as the output of the labeled program $\mathcal{P} = (f, \tau_1, \ldots, \tau_\ell)$.

Definition 7.5 (Multi-Hop Evaluation correctness). *Consider any public parameters* $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{F}, \mathcal{ID}, \mathcal{T})$*, any* $(\mathcal{P}_i, \{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id} \in \mathcal{P}_i}, m_i, \sigma_i)_{i \in [\ell]}$ *such that all the verification keys are honestly generated and, for every* $i \in [\ell]$, $\mathsf{Ver}((\mathcal{P}_i, \{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id} \in \mathcal{P}_i}, m_i, \sigma_i)_{i \in [\ell]}) = 1$*, and any function* $f \in \mathcal{F}$*. If* $\boldsymbol{y} = f(m_1, \ldots, m_\ell)$*,* $\mathcal{P} = f(\mathcal{P}_1, \ldots, \mathcal{P}_\ell)$*, and* $\sigma_{f,y} = \mathsf{Eval}(f, (\mathcal{P}_i, \{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id} \in \mathcal{P}_i}, m_i, \sigma_i)_{i \in [\ell]})$ *then with overwhelming probability* $\mathsf{Ver}(\mathcal{P}, \{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id} \in \mathcal{P}}, \boldsymbol{y}, \sigma_{f,y}) = 1$.

Next, we define *succinctness*, which is the property that makes MKHS a nontrivial primitive to realize. Intuitively, a MKHS is succinct if the size of the signatures generated by Eval is much shorter than the input size of the evaluated function, e.g., polylogarithmic. Our notion is parametric, as for FCs in Theorem 4.1.

Definition 7.6 (Succinctness). *Let* $s_{\mathsf{MKHS}} : \mathbb{N}^4 \to \mathbb{N}$ *be a function. A MKHS scheme* MKHS *for a class of functions* $\mathcal{F}$ *is* s_{MKHS}*-succinct if for every honestly generated parameters* pp*, keys and signatures, and any function* $f : \mathcal{M}^\ell \to \mathcal{M}^{\ell_y}$*,* $f \in \mathcal{F}$*, the output* $\sigma_{f,y}$ *of* $\mathsf{Eval}(f, \cdot)$ *is of size* $|\sigma_{f,y}| \leq s_{\mathsf{MKHS}}(\lambda, \ell, \ell_y, |f|)$*. Additionally, we say that* MKHS *is* succinct *if there exists a fixed function* $s_{\mathsf{MKHS}}(\lambda, \ell, \ell_y, |f|) = \mathsf{poly}(\lambda, \log \ell, \log \ell_y, o(|f|))$*.*

We note that our succinctness definition is stronger than the one originally proposed in [FMNP16] which allowed signatures to grow linearly (or polynomially) in the number t of distinct users involved in the computation, but still logarithmically in the total number of inputs.

7.3.1 Security

The security notion of multi-key homomorphic signatures intuitively models the fact that an adversary, who can query signatures on messages of its choice to multiple users, can only produce valid signatures that are either the ones it received, or ones that are obtained by correctly executing the evaluation algorithm on genuine signatures. The adversary may also corrupt users to obtain their secret keys, yet the alleged forgery cannot involve verification keys of corrupted users.

Definition 7.7 (Unforgeability). *Consider the security experiment* $\mathsf{HomUF\text{-}CMA}_{\mathcal{A},\mathsf{MKHS}}(1^\lambda)$ *in Figure 7.2 between an adversary* $\mathcal{A}$ *and a challenger. A* MKHS *scheme is unforgeable (*HomUF-CMA*-secure) if, for all PPT adversaries* $\mathcal{A}$*, we have* $\Pr[\mathsf{HomUF\text{-}CMA}_{\mathcal{A},\mathsf{MKHS}}(1^\lambda) = 1] \leq \mathsf{negl}(\lambda)$*.*

The above notion of security, introduced by Fiore et al. [FMNP16], is *adaptive* insofar as the adversary can make corruption queries at any point in the game. This notion is stronger than the non-adaptive security achieved by the construction in [FMNP16], where the adversary can perform corruption queries only on identities for which no signature query had already been performed.

Game $\mathsf{HomUF\text{-}CMA}_{\mathcal{A},\mathsf{MKHS}}(1^\lambda)$:

Setup: The challenger proceeds as follows:

- Initialize empty lists $\mathsf{L}_{\mathsf{ID}}, \mathsf{L}_{\mathsf{Corr}}, \mathsf{L}_{\mathsf{Sig}} \longleftarrow \emptyset$ and generate $\mathsf{pp} \longleftarrow \mathsf{Setup}(1^\lambda, \mathcal{F}, \mathcal{ID}, \mathcal{T})$.
- Run $\mathcal{A}(\mathsf{pp})$. Next, $\mathcal{A}$ can make the following queries adaptively.

KeyGen queries $O^{\mathsf{KeyGen}}(\mathsf{id})$**:** If $\mathsf{id} \notin \mathsf{L}_{\mathsf{ID}}$, generate $(\mathsf{vk}_{\mathsf{id}}, \mathsf{sk}_{\mathsf{id}}) \longleftarrow \mathsf{KeyGen}(\mathsf{pp})$, update $\mathsf{L}_{\mathsf{ID}} = \mathsf{L}_{\mathsf{ID}} \cup \{\mathsf{id}\}$, and return $\mathsf{vk}_{\mathsf{id}}$ to $\mathcal{A}$.

Signing queries $O^{\mathsf{Sign}}(\tau, m)$**:** Given $\tau = (\mathsf{id}, \mathsf{tg})$:

- If $(\tau, \cdot) \notin \mathsf{L}_{\mathsf{Sig}}$, compute $\sigma_\tau \longleftarrow \mathsf{Sign}(\mathsf{sk}_{\mathsf{id}}, \tau, m)$, update $\mathsf{L}_{\mathsf{Sig}} := \mathsf{L}_{\mathsf{Sig}} \cup (\tau, m)$, and return σ_τ to $\mathcal{A}$.
- Else, if $(\tau, \cdot) \in \mathsf{L}_{\mathsf{Sig}}$, ignore the query.

Corruption queries $O^{\mathsf{Corr}}(\mathsf{id})$**:** if $\mathsf{id} \in \mathsf{L}_{\mathsf{ID}}$ and $\mathsf{id} \notin \mathsf{L}_{\mathsf{Corr}}$, update $\mathsf{L}_{\mathsf{Corr}} \longleftarrow \mathsf{L}_{\mathsf{Corr}} \cup \mathsf{id}$, and return $\mathsf{sk}_{\mathsf{id}}$

Forgery: At the end of the game, $\mathcal{A}$ returns a tuple $(\mathcal{P}^*, \boldsymbol{y}^*, \sigma^*)$ where $\mathcal{P}^* = (f^*, \tau_1^*, \dots, \tau_\ell^*)$.

Game output: Return 1 if and only if $\mathsf{Ver}(\mathcal{P}^*, \{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id} \in \mathcal{P}^*}, \boldsymbol{y}^*, \sigma^*) = 1$, $\{\mathsf{id} \in \mathcal{P}^*\} \cap \mathsf{L}_{\mathsf{Corr}} = \emptyset$, and one of the following cases occurs:

- Type 1: $\exists j \in [\ell]$ such that $(\tau_j^*, \cdot) \notin \mathsf{L}_{\mathsf{Sig}}$ (i.e., $\mathcal{A}$ never made a query with label τ_j^*).
- Type 2: $\forall i \in [\ell] : (\tau_i^*, m_i) \in \mathsf{L}_{\mathsf{Sig}}$ but $\boldsymbol{y}^* \neq f^*(m_1, \dots, m_\ell)$.

Figure 7.2: *Security experiment* $\mathsf{HomUF\text{-}CMA}_{\mathcal{A},\mathsf{MKHS}}(1^\lambda)$.

7.3.2 Amortized efficiency

We give a definition of amortized efficiency for MKHS schemes. The issue is that in the basic syntax of MKHS (and HS too) the verifier should read the description of the program $\mathcal{P}$ which may take the same running time as the computation to be verified, especially in a model of computation such as circuits. To address this, we consider the case in which one can preprocess the labeled program $\mathcal{P}$, independently of the signature to be verified, and to reuse it. However, we observe that preprocessing the entire tuple $\mathcal{P} = (f, \tau_1, \dots, \tau_\ell)$ would not give any benefit because in MKHS labels are unique, and thus preprocessing a function f for the evaluation on a set of labels $\tau_1, \dots, \tau_\ell$ cannot be reusable. Therefore we model preprocessing via two algorithms: one for the function f and one for the input labels $\tau_1, \dots, \tau_\ell$ and verification keys $\{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id} \in \mathcal{P}}$, which benefits when running the same function f on different set of signed inputs or when executing different functions on the same set of signed inputs.

Definition 7.8 (Amortized efficiency)**.** *An MKHS scheme satisfies amortized efficiency if there is a triple of algorithms* $(\mathsf{PrepFunc}, \mathsf{PrepLabels}, \mathsf{EffVer})$ *such that:*

- *For any labeled program* $\mathcal{P} = (f, \tau_1, \ldots, \tau_\ell)$*, verification keys* $\{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id} \in \mathcal{P}}$*, output* $\boldsymbol{y}$ *and signature* $\sigma_{f,y}$ *such that* $\mathsf{Ver}(\mathcal{P} = (f, \tau_1, \ldots, \tau_\ell), \{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id} \in \mathcal{P}}, \boldsymbol{y}, \sigma_{f,y}) = 1$ *it holds that:*

$$\mathsf{EffVer}(\mathsf{PrepLabels}(\mathsf{pp}, \{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id} \in \mathcal{P}}, (\tau_1, \ldots, \tau_\ell)), \mathsf{PrepFunc}(\mathsf{pp}, f), \boldsymbol{y}, \sigma_{f,y}) = 1$$

- *Given* $\mathsf{vk}_\tau \leftarrow \mathsf{PrepLabels}(\mathsf{pp}, \{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id} \in \mathcal{P}}, (\tau_1, \ldots, \tau_\ell))$ *and* $d_f \leftarrow \mathsf{PrepFunc}(\mathsf{pp}, f)$*, the running time of* $\mathsf{EffVer}(\mathsf{vk}_\tau, d_f, \boldsymbol{y}, \sigma_{f,y})$ *is bounded by* $s_{\mathsf{MKHS}}(\lambda, \ell, \ell_y, |f|) \cdot \ell_y = \mathsf{poly}(\lambda, \log \ell, \ell_y, o(|f|))$.

Finally, we note that previous work on (single-key) homomorphic signatures [CFW14, GVW15] used a different preprocessing approach based on assuming that labels have a structure $\tau = (\Delta, \mathsf{tg})$ consisting of a dataset identifier Δ (e.g., a filename) and a tag.[2] Then they allow preprocessing the circuit along with tags in order to reuse it to verify computations on different datasets. In comparison, our preprocessing notion is more flexible and, by allowing arbitrary labels, implies the one from previous work.

7.3.3 Context Hiding

Informally speaking, a MKHS is context-hiding if signatures on outputs do not reveal information on the inputs of the function. In our work, we adapt to the multi-key setting the context-hiding definition for HS of [CFN15, full version], which in turn generalizes the one in [GVW15].

Definition 7.9 (Context-Hiding MKHS). *A MKHS supports context-hiding if there exist additional PPT procedures* $\tilde{\sigma} \leftarrow \mathsf{Hide}(\mathcal{P}, \{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id} \in \mathcal{P}}, y, \sigma)$ *and* $\mathsf{HVer}(\mathcal{P}, \{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id} \in \mathcal{P}}, y, \sigma)$ *such that:*

- *Correctness: For any tuple* $(\mathcal{P}, \{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id} \in \mathcal{P}}, y, \sigma)$ *such that* $\{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id} \in \mathcal{P}}$ *are honestly generated and* $\mathsf{Ver}(\mathcal{P}, \{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id} \in \mathcal{P}}, y, \sigma) = 1$*, we have that* $\mathsf{HVer}(\mathcal{P}, \{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id} \in \mathcal{P}}, y, \mathsf{Hide}(\mathcal{P}, \{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id} \in \mathcal{P}}, y, \sigma)) = 1$.
- *Unforgeability: The signature scheme is secure when we replace the original verification algorihtm* Ver *with* HVer *in the security game.*
- *Context-Hiding: There is a simulator* $\mathsf{Sim} = (\mathsf{Sim}_{\mathsf{Setup}}, \mathsf{Sim}_{\mathsf{Sig}})$ *such that for any PPT (stateful) distinguisher* $\mathcal{D}$ *running in the experiments* $\{\mathsf{CtxtHiding}^b\}_{b=0,1}$ *defined in Figure 7.3, it holds*

$$\left|\Pr[\mathsf{CtxtHiding}^0_{\mathcal{D},\mathsf{MKHS}}(\lambda) = 1] - \Pr[\mathsf{CtxtHiding}^1_{\mathcal{D},\mathsf{MKHS}}(\lambda) = 1]\right| \leq \mathsf{negl}(\lambda)$$

Generic Context-Hiding solution via NIZKs. We state a simple result showing that any MKHS with amortized verification can be compiled, via the use of a NIZK scheme, into one that has context-hiding.

Theorem 7.10. *Let* MKHS *be a MKHS scheme with amortized efficiency, and let* Π *be a knowledge-sound NIZK for the NP relation* $R_{\mathsf{MKHS}} = \{((\mathsf{vk}_\tau, d_f, y); \sigma_{f,y}) : \mathsf{EffVer}(\mathsf{vk}_\tau, d_f, y, \sigma_{f,y}) = 1\}$*. Then there exists a context-hiding MKHS scheme* MKHS^* *for the same class of functions supported by* MKHS.

[2] Though not formalized, this is the same notion used in the MKHS scheme of [FMNP16].

$\mathsf{CtxtHiding}^0_{\mathcal{D},\mathsf{MKHS}}(\lambda)$	$\mathsf{CtxtHiding}^1_{\mathcal{D}}(\lambda)$
$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{F}, \mathcal{ID}, \mathcal{T})$	$(\mathsf{pp}, \mathsf{td}) \leftarrow \mathsf{Sim}_{\mathsf{Setup}}(1^\lambda, \mathcal{F}, \mathcal{ID}, \mathcal{T})$
$(f, (\mathcal{P}_i, \{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}_i}, m_i, \sigma_i)_{i\in[\ell]}) \leftarrow \mathcal{D}(\mathsf{pp})$	$(f, (\mathcal{P}_i, \{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}_i}, m_i, \sigma_i)_{i\in[\ell]}) \leftarrow \mathcal{D}(\mathsf{pp})$
$b \leftarrow \wedge_{i\in[\ell]}\mathsf{Ver}(\mathcal{P}_i, \{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}_i}, m_i, \sigma_i)$	$b \leftarrow \wedge_{i\in[\ell]}\mathsf{Ver}(\mathcal{P}_i, \{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}_i}, m_i, \sigma_i)$
$y \leftarrow f(m_1, \ldots, m_\ell)$	$y \leftarrow f(m_1, \ldots, m_\ell)$
$\mathcal{P} \leftarrow f(\mathcal{P}_1, \ldots, \mathcal{P}_\ell)$	$\mathcal{P} \leftarrow f(\mathcal{P}_1, \ldots, \mathcal{P}_\ell)$
$\sigma \leftarrow \mathsf{Eval}(f, (\mathcal{P}_i, \{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}_i}, m_i, \sigma_i)_{i\in[\ell]})$	
$\tilde{\sigma} \leftarrow \mathsf{Hide}(\mathcal{P}, \{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}}, y, \sigma))$	$\tilde{\sigma} \leftarrow \mathsf{Sim}_{\mathsf{Sig}}(\mathsf{td}, \mathcal{P}, \{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}}, y))$
$b' \leftarrow \mathcal{D}(\tilde{\sigma})$	$b' \leftarrow \mathcal{D}(\tilde{\sigma})$
return $b \wedge b'$	**return** $b \wedge b'$

Figure 7.3: *Security experiments* $\{\mathsf{CtxtHiding}^b\}_{b=0,1}$ *for context-hiding.*

The proof is rather straightforward and based on the idea of using the NIZK to prove the existence of a valid signature. The amortized efficiency requirement ensures that the scheme remains succinct even if the NIZK is not succinct. A proof sketch is given below.

Sketch. The algorithms of MKHS* are the same as those of MKHS except that MKHS*.Setup runs MKHS.Setup and additionally generates a common reference string for Π. Then the algorithm Hide runs Π's prover on a valid $((\mathsf{vk}_\tau, d_f, y); \sigma_{f,y}) \in R_{\mathsf{MKHS}}$ and sets $\tilde{\sigma}$ as the resulting NIZK proof. In turn, HVer executes Π's verifier on $(\mathsf{vk}_\tau, d_f, y)$ and $\tilde{\sigma}$. Correctness is straightforward. The succinctness of MKHS* is based on the succinctness of MKHS and the fact that EffVer running time is $\mathsf{poly}(\lambda, \log \ell, \log |f|)$; therefore, even if the size of the NIZK proof depended on the size of the statement, it would be still succinct. For the unforgeability of MKHS* we rely on the fact that Π is an argument of knowledge, which allows us to use its extractor to get a MKHS signature σ from the NIZK proof $\tilde{\sigma}$ so that from a forgery for MKHS* we can get one for MKHS. Finally, the context-hiding property follows by the zero-knowledge property of Π. □

7.4 Our MKHS Construction

In this section we present our main result, that is the construction of a fully succinct MKHS.

Our scheme MKHS relies on four building blocks: a functional commitment FC, a digital signature scheme Σ, a somewhere extractable BARG for NP BARG, and a somewhere extractable commitment SEC. MKHS allows the evaluation of the same functions supported by FC, and it supports arbitrary identities and tags, i.e., $\mathcal{T} = \mathcal{ID} = \{0,1\}^\lambda$. We denote messages by m_i and labels by τ_i and assume that $|m_i| = \mathsf{poly}(\lambda)$ for a fixed polynomial.

As described in the technical overview, the main idea of our construction is to combine a BARG proof to attest the validity of each signature-message pair, and a FC proof to show the correct evaluation of f on $m_1, \ldots, m_\ell$, which are committed in com. Moreover, to connect both proofs, our construction verifies the correctness of com inside the BARG circuit C, by starting with an empty commitment, and iteratively building a commitment to $m_1, \ldots, m_\ell$. We remark that the FC scheme must be updatable, and also deterministic, such that we can test commitment equality.

We describe the construction in Figure 7.4 and summarize its main properties in Theorem 7.11. For ease of exposition, in this scheme we focus on single-hop evaluation and do not consider context-hiding. We show in Section 7.5.1 how to achieve multi-hop sequential composition by employing chainable FCs instead of FCs. Also, we recall that context-hiding can be achieved via NIZKs following Theorem 7.10.

Theorem 7.11. *Let* FC *be a deterministic and updatable functional commitment scheme for a class of functions* $\mathcal{F} : \mathcal{M}^\ell \to \mathcal{M}^{\ell_y}$, BARG *a somewhere-extractable batch argument for NP,* SEC *a somewhere extractable commitment, and* Σ *a* EUF-CMA*-secure signature scheme for messages in* $\mathcal{M} \times \{0,1\}^{2\lambda}$*. Then, the construction* MKHS *in Figure 7.4 is an adaptively-secure multi-key homomorphic signature for* $\mathcal{F}$*.*

Moreover, given that the following conditions are satisfied:

- BARG *has proofs of size bounded by* $s_{\mathsf{BARG}}(\lambda, |C|, k)$*.*
- FC *has succinct commitments and* $s_{\mathsf{FC}}(\lambda, \ell, \ell_y, |f|)$*-succinct opening proofs, and admits succinct local verification where* VerUpd *runs in time bounded by* $s_{\mathsf{FC}}(\lambda, \ell, 1, 1)$ *for an update set of size* $|S| = 1$*.*
- SEC *admits local verification with* $s_{\mathsf{SEC}}(\lambda, \ell, B)$ *succinctness.*

Then, MKHS *has succinct signatures of size* $|\sigma_{f,y}| = s_{\mathsf{MKHS}}(\lambda, \ell, \ell_y, |f|)$*, where, for* $|C| = s_{\mathsf{FC}}(\lambda, \ell, 1, \lambda) + s_{\mathsf{SEC}}(\lambda, \ell, \lambda)$*, we have (up to constant factors),*

$$s_{\mathsf{MKHS}}(\lambda, \ell, \ell_y, |f|) = s_{\mathsf{BARG}}(\lambda, |C|, \ell) + s_{\mathsf{FC}}(\lambda, \ell, \ell_y, |f|) + s_{\mathsf{SEC}}(\lambda, \ell, \lambda).$$

Proof. Authentication correctness follows directly by the correctness of Σ. Evaluation correctness follows from the correctness of all the building blocks.

For succinctness, observe that the four additive factors in the expression for $|\sigma_{f,y}|$ correspond to the sizes of $\pi_\sigma, \pi_f, \mathsf{com}, \mathsf{com}_w$, respectively. To calculate the expression for $s_{\mathsf{MKHS}}(\lambda, \ell, \ell_y, |f|)$, note that the block size of the SEC is $B = \mathsf{poly}(\lambda, \log \ell)$. Then, note that all keys, commitments, and openings involved in C are of size $s_{\mathsf{FC}}(\lambda, \ell, 1, \lambda) + s_{\mathsf{SEC}}(\lambda, \ell, \lambda)$, as well as the running time of the $\mathsf{FC.VerUpd}$, $\mathsf{SEC.Ver}$ and $\Sigma.\mathsf{Ver}$ algorithms. Hence, $|C| = s_{\mathsf{FC}}(\lambda, \ell, 1, \lambda) + s_{\mathsf{SEC}}(\lambda, \ell, \lambda)$.

We prove security in Section 7.4.2. □

Remark 7.12. *For usual asymptotic succinctness bounds where* $s_{\mathsf{BARG}}(\lambda, |C|, k) = \mathsf{poly}(\lambda, |C|, \log k)$*,* $s_{\mathsf{FC}}(\lambda, \ell, \ell_y, |f|) = \mathsf{poly}(\lambda, \log \ell, \log \ell_y, o(|f|))$*, and* $s_{\mathsf{SEC}}(\lambda, \ell, B) = \mathsf{poly}(\lambda, \log \ell, B)$*, we have that* $s_{\mathsf{MKHS}}(\lambda, \ell, \ell_y, |f|) = \mathsf{poly}(\lambda, \log \ell, \log \ell_y, o(|f|))$*.*

7.4.1 Efficient Verification

If FC has amortized efficient verification, then it is possible to preprocess the function f. Similarly, if BARG has amortized efficient verification, it is possible to preprocess the

$\underline{\mathsf{MKHS.Setup}(1^\lambda, 1^\ell, \mathcal{F})}$:

- Calculate the required circuit size $|C|$ given $\ell, \mathcal{F}, \lambda$.
- Calculate the required block size B from λ. Note that $B = \mathsf{poly}(\lambda)$.
- $\mathsf{crs} \leftarrow \mathsf{BARG.Setup}(1^\lambda, \ell, 1^{|C|})$.
- $\mathsf{dk} \leftarrow \mathsf{SEC.Setup}(1^\lambda, \ell, B)$
- $\mathsf{ck} \leftarrow \mathsf{FC.Setup}(1^\lambda, 1^\ell)$.
- Output $\mathsf{pp} \leftarrow (\mathsf{crs}, \mathsf{dk}, \mathsf{ck})$.

$\underline{\mathsf{MKHS.KeyGen}(1^\lambda)}$: Output $(\mathsf{vk}, \mathsf{sk}) \leftarrow \Sigma.\mathsf{KeyGen}(1^\lambda)$.

$\underline{\mathsf{MKHS.Sign}(\mathsf{sk}, \tau, m)}$: Output $\sigma \leftarrow \Sigma.\mathsf{Sign}(\mathsf{sk}, m|\tau)$.

$\underline{\mathsf{MKHS.Eval}(\mathsf{pp}, f, (\tau_i, \mathsf{vk}_i, m_i, \sigma_i)_{i\in[\ell]})}$:

- Parse $\mathsf{pp} := (\mathsf{crs}, \mathsf{ck}, \mathsf{dk})$.
- $(\mathsf{com}, \mathsf{aux}) \leftarrow \mathsf{FC.Com}(\mathsf{ck}, m_1, \dots, m_\ell)$.
- $\pi_f \leftarrow \mathsf{FC.FuncProve}(\mathsf{ck}, f, \mathsf{aux})$.
- $(\mathsf{com}_0, \mathsf{aux}_0) \leftarrow \mathsf{FC.Com}(\mathsf{ck}, \mathbf{0})$.
- For $i \in [\ell]$, compute $(\mathsf{com}_i, \mathsf{aux}_i, \pi_i) \leftarrow \mathsf{FC.Upd}(\mathsf{ck}, \mathsf{aux}_{i-1}, i, m_i)$.
  Note that each com_i is a commitment to the *partial* vector $(m_1, \dots, m_i, 0, \dots, 0)$. Note also that $\mathsf{com}_\ell = \mathsf{com}$.
- Compute a somewhere extractable commitment to all partial commitments $(\mathsf{com}_w, \mathsf{aux}_w) \leftarrow \mathsf{SEC.Com}(\mathsf{dk}, (\mathsf{com}_1, \dots, \mathsf{com}_\ell))$.
- For $i \in [\ell]$, compute local openings $o_i \leftarrow \mathsf{SEC.Open}(\mathsf{dk}, \mathsf{aux}_w, i)$ to each com_i.
- Compute a BARG proof $\pi_\sigma \leftarrow \mathsf{BARG.Prove}(\mathsf{crs}, C, \{\mathtt{x}_i, \mathtt{w}_i\}_i)$ for circuit $C(\mathtt{x}_i, \mathtt{w}_i)$ as described in Figure 7.5.
- Output $\sigma_{f,y} = (\mathsf{com}, \pi_f, \pi_\sigma, \mathsf{com}_w)$.

$\underline{\mathsf{MKHS.Ver}(\mathsf{pp}, \mathcal{P}, \{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}}, \boldsymbol{y}, \sigma_{f,y})}$:

- Parse $\mathcal{P} := (f, \tau_1, \dots, \tau_\ell)$ and $\{\tau_i := (\mathsf{id}_i, \tau_i)\}$.
- If $\mathcal{P} = (f_{id}, \tau_1)$ then check that $\Sigma.\mathsf{Ver}(\mathsf{vk}_{\mathsf{id}_1}, \boldsymbol{y}|\tau_1, \sigma_{f,y}) = 1$.
- Else, parse $\sigma_{f,y} := (\mathsf{com}, \pi_f, \pi_\sigma, \mathsf{com}_w)$.
- Let $\mathsf{com}_0 \leftarrow \mathsf{FC.Com}(\mathsf{ck}, \mathbf{0})$
- Compute the circuit C in Figure 7.5, hardcoding $\mathsf{com}, \mathsf{com}_w, \mathsf{com}_0$.
- Given $\{\mathsf{vk}_i\}_i := \{\mathsf{vk}_{\mathsf{id}_i}\}_i$ and $\{\tau_i\}_i, \{\mathsf{ck}_i\}_i, \{\mathsf{dk}_i\}_i$, define $\mathtt{x}_i = (\mathsf{vk}_i, \mathsf{ck}_i, \mathsf{dk}_i, \mathsf{dk}_{i-1}, \tau_i, i)$.
- Output 1 iff $\mathsf{FC.FuncVer}(\mathsf{ck}, \mathsf{com}, f, \boldsymbol{y}, \pi_f) = 1$ and $\mathsf{BARG.Ver}(\mathsf{crs}, C, \{\mathtt{x}_i\}_i, \pi_\sigma) = 1$.

Figure 7.4: *Construction of a multi-key homomorphic signature scheme* MKHS *from a functional commitment* FC, *a BARG for NP* BARG, *a somewhere extractable commitment* SEC *and a digital signature* Σ.

Description of $C(\mathtt{x}, \mathtt{w})$:
Hardwired: $\mathsf{com}, \mathsf{com}_w, \mathsf{com}_0$
Statement: $\mathtt{x} = (\mathsf{vk}_i, \mathsf{ck}_i, \mathsf{dk}_i, \mathsf{dk}_{i-1}, \tau_i, i)$
Witness: $\mathtt{w} = (m_i, \sigma_i, \pi_i, \mathsf{com}_{i-1}, \mathsf{com}_i, o_{i-1}, o_i)$
Circuit:

- If $i = 1$, check that $\mathsf{com}_{i-1} = \mathsf{com}_0$ and skip the SEC verification check for $i - 1$.
- If $i = \ell$, check that $\mathsf{com}_i = \mathsf{com}$
- Check that:

$$\begin{aligned}&\Sigma.\mathsf{Ver}(\mathsf{vk}_i, m_i|\tau_i, \sigma_i) = 1\\ \wedge\ &\mathsf{FC.VerUpd}(\mathsf{ck}_i, i, \mathsf{com}_{i-1}, 0, \mathsf{com}_i, m_i, \pi_i) = 1\\ \wedge\ &\mathsf{SEC.Ver}(\mathsf{dk}_i, \mathsf{com}_w, i, o_i, \mathsf{com}_i) = 1\\ \wedge\ &\mathsf{SEC.Ver}(\mathsf{dk}_{i-1}, \mathsf{com}_w, i-1, o_{i-1}, \mathsf{com}_{i-1}) = 1\end{aligned}$$

Figure 7.5: *Description of the BARG circuit C.*

labels τ_i and the respective verification keys. We describe the corresponding preprocessing algorithms MKHS.PrepFunc and MKHS.PrepLabels, as well as the efficient verification algorithm MKHS.EffVer, in Figure 7.6.

We summarize the efficient verification properties in the following corollary of Theorem 7.11. The proof follows from the definitions of efficient verification for BARGs (Definition 6.5) and for FCs (Definition 4.5).

Corollary 7.13. *If* BARG *and* FC *admit efficient verification, the* MKHS *scheme from Figure 7.4 with the algorithms in Figure 7.6 also satisfies efficient verification, i.e., the running time of* $\mathsf{MKHS.EffVer}(\mathsf{vk}_\tau, \mathsf{ck}_f, \boldsymbol{y}, \sigma_{f,y})$ *is bounded by* $s_{\mathsf{MKHS}}(\lambda, \ell, \ell_y, |f|) \cdot \ell_y$.

We note that the efficient verification property of our scheme is flexible, in the sense that we introduce separate algorithms for preprocessing the function PrepFunc and for the labels PrepLabels. Therefore, if the BARG satisfies efficient verification but the FC does not (or vice-versa), our MKHS admits pre-processing only the labels (or the function).

7.4.2 Proof of Security

Let $\mathcal{A}$ be an adversary in the security experiment $\mathsf{HomUF\text{-}CMA}_{\mathcal{A},\mathsf{MKHS}}(1^\lambda)$ for our MKHS construction. In this game, the adversary has access to a signing oracle $\mathcal{O}^{\mathsf{Sign}}$ such that, for $\tau = (\mathsf{id}, \mathsf{tg})$, then $\mathcal{O}^{\mathsf{Sign}}(\tau, m)$ outputs $\sigma_\tau \leftarrow \Sigma.\mathsf{Sign}(\mathsf{sk}_{\mathsf{id}}, m|\tau)$. It also has access to a key generation $\mathcal{O}^{\mathsf{KeyGen}}$ and a corruption $\mathcal{O}^{\mathsf{Corr}}$ oracle. Finally, $\mathcal{A}$ produces an alleged forgery $(\mathcal{P}^*, \boldsymbol{y}^*, \sigma^*)$ where $\mathcal{P}^* = (f^*, \tau_1^*, \ldots, \tau_\ell^*)$. We recall that there are two possible types of forgeries, that we define formally as the following events.

<u>$\mathsf{MKHS.PrepFunc}(\mathsf{pp}, f)$</u> :

- Parse $\mathsf{pp} := (\mathsf{crs}, \mathsf{ck}, \mathsf{dk})$.
- Output $\mathsf{ck}_f \leftarrow \mathsf{FC.PreFuncVer}(\mathsf{ck}, f)$.

<u>$\mathsf{MKHS.PrepLabels}(\mathsf{pp}, (\mathsf{vk}_i, \tau_i)_{i\in[\ell]})$</u> :

- Parse $\mathsf{pp} := (\mathsf{crs}, \mathsf{ck}, \mathsf{dk})$.
- Given $\{\mathsf{vk}_i\}_i$, $\{\tau_i\}_i$, $\{\mathsf{ck}\}_i$, $\{\mathsf{dk}\}_i$, define $\mathtt{x}_i = (\mathsf{vk}_i, \tau_i, \mathsf{ck}_i, \mathsf{dk}_i)$.
- Output $\mathsf{vk}_\tau \leftarrow \mathsf{BARG.PreVer}(\mathsf{crs}, \{\mathtt{x}_i\}_i)$.

<u>$\mathsf{MKHS.EffVer}(\mathsf{vk}_\tau, \mathsf{ck}_f, \boldsymbol{y}, \sigma_{f,y})$</u> :

- Parse $\sigma_{f,y} := (\mathsf{com}, \pi_f, \pi_\sigma, \mathsf{com}_w)$.
- Let $\mathsf{com}_0 \leftarrow \mathsf{FC.Com}(\mathsf{ck}, \mathbf{0})$
- Check that $\mathsf{FC.EffFuncVer}(\mathsf{ck}, \mathsf{com}, \mathsf{ck}_f, \boldsymbol{y}, \pi_f) = 1$
- Compute the BARG circuit C, hardcoding $\mathsf{com}, \mathsf{com}_w, \mathsf{com}_0$.
- Check that $\mathsf{BARG.EffVer}(\mathsf{crs}, C, \mathsf{vk}_\tau, \pi_\sigma) = 1$
- Output 1 iff both checks pass.

Figure 7.6: *Efficient verification algorithms for our construction of a multi-key homomorphic signature scheme* MKHS.

- $\mathsf{TYPE}_1 := \exists j \in [\ell], (\tau_j^*, \cdot) \notin \mathsf{L}_{\mathsf{Sig}}$. Namely, there exists some index j such that $\mathcal{A}$ never queried $(\tau_j^*, \cdot)$ to the signing oracle.
- $\mathsf{TYPE}_2 := \forall i \in [\ell], (\tau_i^*, m_i) \in \mathsf{L}_{\mathsf{Sig}} \ \wedge\ \boldsymbol{y}^* \neq f^*(m_1, \dots, m_\ell)$. Namely, $\mathcal{A}$ asked all queries (τ_i^*, m_i) to the signing oracle, but cheated at computing $\boldsymbol{y}^*$.

For both types of forgeries we can partition on whether the forgery is a fresh signature (i.e., $\mathcal{P} = \mathcal{I}_\tau$) or an evaluated one. In the event of type 2 forgeries, for our scheme we can also partition over the event '$\mathsf{com}^* = \mathsf{FC.Com}(\mathsf{ck}, m_1, \dots, m_\ell)$', where com^* is the (deterministic) commitment included in σ^*.

Let also VER be the event that verification passes and that no user involved in a labeled program is corrupted, i.e.,

$$\mathsf{VER} := \mathsf{MKHS.Ver}(\mathsf{pp}, \mathcal{P}^*, \{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}^*}, \boldsymbol{y}^*, \sigma^*) = 1 \wedge \{\mathsf{id} \in \mathcal{P}^*\} \cap \mathsf{L}_{\mathsf{Corr}} = \emptyset.$$

We define 4 experiments, UF_1, UF_2, UF_3, and UF_4:

- UF_1 outputs 1 iff $\mathsf{VER} \wedge \mathcal{P} \neq \mathcal{I}_\tau \wedge \mathsf{TYPE}_1$.
- UF_2 outputs 1 iff $\mathsf{VER} \wedge \mathcal{P} \neq \mathcal{I}_\tau \wedge \mathsf{TYPE}_2 \wedge (\mathsf{com}^* = \mathsf{FC.Com}(\mathsf{ck}, m_1, \dots, m_\ell))$.
- UF_3 outputs 1 iff $\mathsf{VER} \wedge \mathcal{P} \neq \mathcal{I}_\tau \wedge \mathsf{TYPE}_2 \wedge (\mathsf{com}^* \neq \mathsf{FC.Com}(\mathsf{ck}, m_1, \dots, m_\ell))$.
- UF_4 outputs 1 iff $\mathsf{VER} \wedge \mathcal{P} = \mathcal{I}_\tau \wedge (\mathsf{TYPE}_1 \vee \mathsf{TYPE}_2)$.

Overall, we partitioned the probability space so that, by the union bound, for any PPT adversary $\mathcal{A}$ we have that $\Pr[\mathsf{HomUF\text{-}CMA}_{\mathcal{A},\mathsf{MKHS}}(\lambda) = 1] \leq \sum_{k=1}^{4} \Pr[\mathsf{UF}_{k,\mathcal{A}}(\lambda) = 1]$. We separate the proof in lemmas that bound the probability that $\mathcal{A}$ wins in each of the experiments.

Lemma 7.14. *For any PPT adversary $\mathcal{A}$ making at most $Q = \mathsf{poly}(\lambda)$ queries to the key generation oracle and that can produce a valid forgery in UF_1, there exist PPT adversaries $\mathcal{B}_{sind}, \mathcal{B}_{sExt}, \mathcal{B}_{\text{EUF-CMA}}$ against the BARG setup indistinguishability, somewhere extractability and the* EUF-CMA *property of the digital signature scheme Σ, such that:*

$$\Pr[\mathsf{UF}_{1,\mathcal{A}}(\lambda) = 1] \leq \\ \ell \cdot \left(\mathsf{Adv}^{\text{sind}}_{\mathsf{BARG},\mathcal{B}_{sind}}(\lambda) + \mathsf{Adv}^{\text{sext}}_{\mathsf{BARG},\mathcal{B}_{sExt}}(\lambda) + Q \cdot \mathsf{Adv}^{\text{euf-cma}}_{\Sigma,\mathcal{B}_{\text{EUF-CMA}}}(\lambda)\right).$$

Proof. We first define WIN_1 as the winning event of UF_1:

$$\mathsf{WIN}_1 := \left\{ \begin{array}{l} \exists j \in [\ell] : (\tau_j^* = (\mathsf{id}_j^*, \mathsf{tg}_j^*), \cdot) \notin \mathsf{L}_{\mathsf{Sig}} \wedge \mathsf{id}_j^* \notin \mathsf{L}_{\mathsf{Corr}} \\ \wedge\ \mathsf{BARG.Ver}(\mathsf{crs}, C, \{\mathsf{x}_i^*\}_i, \pi_\sigma^*) = 1 \end{array} \right. .$$

Notice that WIN_1 is implied by $\mathsf{VER} \wedge \mathsf{TYPE}_1$, and that we have suppressed unnecesary checks that will not be used in the proof of this lemma. Based on this winning condition, we define a series of hybrid games $\mathsf{Hyb}^0, \mathsf{Hyb}^1, \mathsf{Hyb}^2, \mathsf{Hyb}^3$ described in Figure 7.7.

$\underline{\mathsf{Hyb}^0}$**:** As described above, this game is a simplified version of UF_1 where we omit unnecessary outputs from the adversary and from the winning condition.

$\underline{\mathsf{Hyb}^1}$**:** To transition from Hyb^0 to Hyb^1, since the choice of j^* is uniform over $[\ell]$, we have that $j = j^*$ with probability $\frac{1}{\ell}$. As a result we have that:

$$\Pr[\mathsf{Hyb}^1_{\mathcal{A}}(\lambda) = 1] \geq \frac{1}{\ell} \Pr[\mathsf{Hyb}^0_{\mathcal{A}}(\lambda) = 1].$$

$\underline{\mathsf{Hyb}^2}$**:** In this game, the only difference with Hyb^1 is that the BARG setup is set in trapdoor mode at position j^*. Then, we have that if $\mathcal{A}$ interpolates between Hyb^1 and Hyb^2, we can construct an adversary $\mathcal{B}_{sind}$ against BARG setup indistinguishability property such that

$$\Pr[\mathsf{Hyb}^2_{\mathcal{A}}(\lambda) = 1] \leq \Pr[\mathsf{Hyb}^1_{\mathcal{A}}(\lambda) = 1] + \mathsf{Adv}^{\text{sind}}_{\mathsf{BARG},\mathcal{B}_{sind}}(\lambda).$$

$\underline{\mathsf{Hyb}^3}$**:** If $\mathcal{A}$ outputs 1 against Hyb^2 but outputs 0 against Hyb^3, it must be the case that:

- $\mathsf{BARG.Ver}(\mathsf{crs}, C, \{\mathsf{x}_i\}_i, \pi_{\sigma^*}) = 1$,
- $C(\mathsf{x}_{j^*}, \bar{\mathsf{w}}_{j^*}) \neq 1$ where $\bar{\mathsf{w}}_{j^*}$ is obtained from $\mathsf{BARG.Ext}(\mathsf{td}, C, \pi_\sigma^*)$.

Then, we can use $\mathcal{A}$ to construct an adversary $\mathcal{B}_{sExt}$ against BARG somewhere extractability such that:

$$\Pr[\mathsf{Hyb}^3_{\mathcal{A}}(\lambda) = 1] \leq \Pr[\mathsf{Hyb}^2_{\mathcal{A}}(\lambda) = 1] + \mathsf{Adv}^{\text{sext}}_{\mathsf{BARG},\mathcal{B}_{sExt}}(\lambda).$$

```
$\mathsf{Hyb}^0_{\mathcal{A}}(\lambda)$:
    $\mathsf{crs} \leftarrow \mathsf{BARG.Setup}()$
    $\mathsf{dk} \leftarrow \mathsf{SEC.Setup}()$
    $\mathsf{ck} \leftarrow \mathsf{FC.Setup}()$
    $(\mathcal{P}^*, \pi^*_\sigma, \{\mathtt{x}^*_i\}_i) \leftarrow \mathcal{A}^{O}(\mathsf{crs}, \mathsf{dk}, \mathsf{ck})$
    **Output 1 iff:**
        $\mathsf{WIN}_1 = 1$

$\mathsf{Hyb}^1_{\mathcal{A}}(\lambda)$:
    $j^* \leftarrow\$ [\ell]$
    $\mathsf{crs} \leftarrow \mathsf{BARG.Setup}()$
    $\mathsf{dk} \leftarrow \mathsf{SEC.Setup}()$
    $\mathsf{ck} \leftarrow \mathsf{FC.Setup}()$
    $(\mathcal{P}^*, \pi^*_\sigma, \{\mathtt{x}^*_i\}_i) \leftarrow \mathcal{A}^{O}(\mathsf{crs}, \mathsf{dk}, \mathsf{ck})$
    **Output 1 iff:**
        $\mathsf{WIN}_1 = 1$
        $\wedge (j^* = j)$

$\mathsf{Hyb}^2_{\mathcal{A}}(\lambda)$:
    $j^* \leftarrow\$ [\ell]$
    $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{BARG.TdSetup}(j^*)$
    $\mathsf{dk} \leftarrow \mathsf{SEC.Setup}()$
    $\mathsf{ck} \leftarrow \mathsf{FC.Setup}()$
    $(\mathcal{P}^*, \pi^*_\sigma, \{\mathtt{x}^*_i\}_i) \leftarrow \mathcal{A}^{O}(\mathsf{crs}, \mathsf{dk}, \mathsf{ck})$
    **Output 1 iff:**
        $\mathsf{WIN}_1 = 1$
        $\wedge (j^* = j)$

$\mathsf{Hyb}^3_{\mathcal{A}}(\lambda)$:
    $j^* \leftarrow\$ [\ell]$
    $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{BARG.TdSetup}(j^*)$
    $\mathsf{dk} \leftarrow \mathsf{SEC.Setup}()$
    $\mathsf{ck} \leftarrow \mathsf{FC.Setup}()$
    $\bar{\mathtt{w}}_{j^*} \leftarrow \mathsf{BARG.Ext}(\mathsf{td}, C, \pi^*_\sigma)$
    $(\mathcal{P}^*, \pi^*_\sigma, \{\mathtt{x}^*_i\}_i) \leftarrow \mathcal{A}^{O}(\mathsf{crs}, \mathsf{dk}, \mathsf{ck})$
    **Output 1 iff:**
        $\mathsf{WIN}_1 = 1$
        $\wedge (j^* = j)$
        $\wedge C(\mathtt{x}^*_{j^*}, \bar{\mathtt{w}}_{j^*}) = 1$
```

Figure 7.7: *Games* $\mathsf{Hyb}^0, \mathsf{Hyb}^1, \mathsf{Hyb}^2, \mathsf{Hyb}^3$ *for the proof of Lemma 7.14. We highlight changes between games and omit inputs to* Setup *for succinctness.*

Finally, we proceed to bound the advantage of $\mathcal{A}$ in Hyb^3. Recall that $\mathcal{A}$ can make at most $Q = \mathsf{poly}(\lambda)$ queries to the key generation oracle O^{KeyGen}. We use $\mathcal{A}$ to build an algorithm $\mathcal{B}_{\text{EUF-CMA}}$ that breaks the existential unforgeability of Σ. $\mathcal{B}_{\text{EUF-CMA}}$ simulates the game Hyb^3 to $\mathcal{A}$, proceeding as follows:

1. $\mathcal{B}_{\text{EUF-CMA}}$ receives the verification key vk^* from the EUF-CMA challenger.
2. $\mathcal{B}_{\text{EUF-CMA}}$ starts by uniformly sampling $j^* \leftarrow\$ [\ell]$ and $q^* \leftarrow\$ [Q]$. It initializes empty lists $\mathsf{L}_{\mathsf{ID}}, \mathsf{L}_{\mathsf{Sig}} \leftarrow \emptyset$. Then $\mathcal{B}_{\text{EUF-CMA}}$ runs the setup algorithm for $\mathsf{BARG}, \mathsf{FC}, \mathsf{MKHS}$, setting $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{BARG.TdSetup}(j^*)$. It then sends the public parameters pp to $\mathcal{A}$.
3. Whenever $\mathcal{A}$ makes a query to $O^{\mathsf{KeyGen}}(\mathsf{id})$:
   - If the query is the q^*-th query, let $\mathsf{vk}_{\mathsf{id}} = \mathsf{vk}^*$ and return $\mathsf{vk}_{\mathsf{id}}$ to $\mathcal{A}$.
   - Otherwise, let $(\mathsf{vk}_{\mathsf{id}}, \mathsf{sk}_{\mathsf{id}}) \leftarrow \Sigma.\mathsf{KeyGen}(1^\lambda)$.
4. Whenever $\mathcal{A}$ makes a query to $O^{\mathsf{Sign}}(m, \tau = (\mathsf{id}, \mathsf{tg}))$:
   - If $(\tau, m) \notin \mathsf{L}_{\mathsf{Sig}}$:

  - If $\mathsf{id} = \mathsf{id}_{q^*}$: forward the query to the EUF-CMA oracle $\sigma \leftarrow O^{\mathsf{Sign}}(m|\tau)$, update $\mathsf{L_{Sig}} := \mathsf{L_{Sig}} \cup (\tau, m)$ and then send σ to $\mathcal{A}$.
  - Otherwise, if $\mathsf{id} \neq \mathsf{id}_{q^*}$: compute $\sigma_\tau \leftarrow \mathsf{Sign}(\mathsf{sk_{id}}, \tau, m)$, update $\mathsf{L_{Sig}} := \mathsf{L_{Sig}} \cup (\tau, m)$, and return σ_τ to $\mathcal{A}$.
- Else, if $(\tau, m) \in \mathsf{L_{Sig}}$, ignore the query.

5. Whenever $\mathcal{A}$ makes a query to $O^{\mathsf{Corr}}(\mathsf{id})$:
   - if $\mathsf{id} = \mathsf{id}_{q^*}$, abort
   - else, if $\mathsf{id} \in \mathsf{L_{ID}}$ and $\mathsf{id} \notin \mathsf{L_{Corr}}$, update $\mathsf{L_{Corr}} \leftarrow \mathsf{L_{Corr}} \cup \mathsf{id}$, and return $\mathsf{sk_{id}}$.
6. At the end of the game $\mathcal{A}$ outputs $(\mathcal{P}^*, y^*, \sigma^*)$. $\mathcal{B}_{\text{EUF-CMA}}$ checks that $\mathsf{BARG.Ver}(\mathsf{crs}, C, \{\mathsf{x}_i^*\}_i, \pi_\sigma^*) = 1$ and that $(\tau_{j^*}^* = (\mathsf{id}_{j^*}^*, \mathsf{tg}_{j^*}^*), \cdot) \notin \mathsf{L_{Sig}}$ and $\mathsf{id}_{j^*}^* \notin \mathsf{L_{Corr}}$. Additionally, it checks that $\mathsf{id}_{j^*}^* = \mathsf{id}_{q^*}$.

   If any of these checks does not pass, $\mathcal{B}_{\text{EUF-CMA}}$ aborts. Otherwise it computes $\bar{\mathsf{w}}_{j^*} \leftarrow \mathsf{BARG.Ext}(\mathsf{td}, C, \{\mathsf{x}_i^*\}_{i\in[\ell]}, \pi_\sigma^*)$ and parses $\bar{\sigma}_{j^*}$ and $\bar{m}_{j^*}$ from $\bar{\mathsf{w}}_{j^*}$. At the end $\mathcal{B}_{\text{EUF-CMA}}$ outputs $(\bar{m}_{j^*}|\tau_{j^*}^*, \bar{\sigma}_{j^*})$ as its forgery.

By construction, conditioned on $\mathsf{id}_{j^*}^* = \mathsf{id}_{q^*}$, algorithm $\mathcal{B}_{\text{EUF-CMA}}$ perfectly simulates an execution of Hyb^3 to $\mathcal{A}$. Note that, overall, the probability of not aborting during the simulation is at least $1/Q$, since the winning condition guarantees that there exists at least one identity that remains uncorrupted. If all guesses are correct, as C is explicitly checking $\Sigma.\mathsf{Ver}(\mathsf{vk}^*, \bar{m}_{j^*}|\tau_{j^*}^*, \bar{\sigma}_{j^*}) = 1$, it means that $\bar{\sigma}_{j^*}$ is a valid signature on $\bar{m}_{j^*}|\tau_{j^*}^*$.

Thus with probability at least $\frac{1}{Q}\cdot\Pr[\mathsf{Hyb}^3_{\mathcal{A}}(\lambda) = 1]$, $\mathcal{B}_{\text{EUF-CMA}}$ outputs a valid EUF-CMA forgery. In summary,

$$\Pr[\mathsf{Hyb}^3_{\mathcal{A}}(1^\lambda) = 1] \leq Q \cdot \mathsf{Adv}^{\text{euf-cma}}_{\Sigma,\mathcal{B}_{\text{EUF-CMA}}}(\lambda).$$

□

Lemma 7.15. *For any PPT adversary $\mathcal{A}$ that can produce a valid forgery against UF_2, there exists a PPT adversary $\mathcal{B}$ against evaluation binding of the functional commitment scheme FC such that*

$$\Pr[\mathsf{UF}_{2,\mathcal{A}}(\lambda) = 1] \leq \mathsf{Adv}^{\text{evbind}}_{\mathsf{FC},\mathcal{B}}(\lambda).$$

Proof. As in the proof of Lemma 7.14, we first define a wining event WIN_2 as a simplification of the winning condition of UF_2 which only includes the checks that are relevant for the reduction.

$$\mathsf{WIN}_2 := \begin{cases} \forall i \in [\ell], (\tau_i^*, m_i) \in \mathsf{L_{Sig}} \\ \wedge\ \boldsymbol{y}^* \neq f^*(m_1, \ldots, m_\ell) \\ \wedge\ \mathsf{com}^* = \mathsf{FC.Com}(\mathsf{ck}, (m_1, \ldots, m_\ell)) \\ \wedge\ \mathsf{FC.FuncVer}(\mathsf{ck}, \mathsf{com}^*, \boldsymbol{y}^*, f^*, \pi_{f^*}^*) = 1 \end{cases}.$$

We describe how to build an efficient algorithm $\mathcal{B}$ that breaks the evaluation binding of FC.

1. $\mathcal{B}$ receives a commitment key ck by the challenger of the evaluation binding game.
2. $\mathcal{B}$ initialize empty lists $\mathsf{L}_{\mathsf{ID}}, \mathsf{L}_{\mathsf{Sig}} \leftarrow \emptyset$. Then $\mathcal{B}$ runs $\mathsf{crs} \leftarrow \mathsf{BARG.Setup}(1^\lambda, \ell, 1^{|C|})$ and $\mathsf{dk} \leftarrow \mathsf{SEC.Setup}(1^\lambda, \ell, B)$ and sends $\mathsf{pp} \leftarrow (\mathsf{crs}, \mathsf{dk}, \mathsf{ck})$ to $\mathcal{A}$.
3. $\mathcal{B}$ simulates all of $\mathcal{A}$'s queries to $\mathcal{O}^{\mathsf{KeyGen}}, \mathcal{O}^{\mathsf{Sign}}, \mathcal{O}^{\mathsf{Corr}}$ by using knowledge of the secret keys, and updates the list $\mathsf{L}_{\mathsf{Sig}}$ every time a fresh $\mathcal{O}^{\mathsf{Sign}}(\tau^*, m)$ query is made by $\mathcal{A}$.
4. At the end of the simulation, $\mathcal{A}$ outputs $(\mathcal{P}^*, \mathsf{com}^*, \pi^*_{f^*})$ (we ignore the remaining outputs). $\mathcal{B}$ parses $(f^*, (\tau_1^*, \dots, \tau_\ell^*))$ from $\mathcal{P}^*$, and retrieves the messages $m_1, \dots, m_\ell$ associated to labels $\tau_1^*, \dots, \tau_\ell^*$ from $\mathsf{L}_{\mathsf{Sig}}$.
5. Finally, $\mathcal{B}$ computes the honest output $\boldsymbol{y} = f^*(m_1, \dots, m_\ell)$, and an honest FC opening proof to $\boldsymbol{y}$ as $\pi_{f^*} \leftarrow \mathsf{FC.FuncProve}(\mathsf{ck}, (m_1, \dots, m_\ell), f^*)$. Then, $\mathcal{B}$ outputs $(\mathsf{com}^*, f^*, \boldsymbol{y}, \pi_{f^*}, \boldsymbol{y}^*, \pi^*_{f^*})$.

By construction, $\mathcal{B}$ perfectly simulates an execution of the MKHS game for $\mathcal{A}$. Also, note that if $\mathcal{A}$ is a successful adversary against UF_2, then by the WIN_2 event, the messages $m_1, \dots, m_\ell$ retrieved from $\mathsf{L}_{\mathsf{Sig}}$ must be the same ones that are committed under com^*. As com^* and $\boldsymbol{y}$ are honest, we have that $\mathsf{FC.FuncVer}(\mathsf{ck}, \mathsf{com}^*, \boldsymbol{y}, f^*, \pi_{f^*}) = 1$.

Thus, $\mathcal{B}$'s output is a valid output in the FC evaluation binding game. To summarize,

$$\Pr[\mathsf{UF}_{2,\mathcal{A}}(\lambda) = 1] \leq \mathsf{Adv}^{\mathrm{evbind}}_{\mathsf{FC},\mathcal{B}}(\lambda).$$

□

Lemma 7.16. *For any PPT adversary $\mathcal{A}$ that wins in the* UF_3 *game, there exists a tuple of PPT adversaries* $(\mathcal{B}_1, \dots, \mathcal{B}_6)$ *such that*

$$\Pr[\mathsf{UF}_{3\mathcal{A}}(\lambda) = 1] \leq \ell \cdot \Big[\mathsf{Adv}^{\mathrm{sind}}_{\mathsf{BARG},\mathcal{B}_1}(\lambda) + \mathsf{Adv}^{\mathrm{sext}}_{\mathsf{BARG},\mathcal{B}_2}(\lambda) + 2 \cdot \mathsf{Adv}^{\mathrm{sind}}_{\mathsf{SEC},\mathcal{B}_3}(\lambda) + \mathsf{Adv}^{\mathrm{sext}}_{\mathsf{SEC},\mathcal{B}_4}(\lambda) + Q \cdot \mathsf{Adv}^{\mathrm{eufcma}}_{\Sigma,\mathcal{B}_5}(\lambda) + \mathsf{Adv}^{\mathrm{updbind}}_{\mathsf{FC},\mathcal{B}_6}(\lambda)\Big].$$

Proof. For $\mathcal{A}$ to win in event UF_3, it must have crafted a type 2 forgery $\boldsymbol{y}^* \neq f^*(m_1, \dots, m_\ell)$ such that $\forall i \in [\ell], (\tau_i^*, m_i) \in \mathsf{L}_{\mathsf{Sig}}$, and such that $\{\mathsf{id} \in \mathcal{P}^*\} \cap \mathsf{L}_{\mathsf{Corr}} = \emptyset$. Besides, the commitment com^* to the messages must not be honestly computed, $\mathsf{com}^* \neq \mathsf{FC.Com}(\mathsf{ck}, \boldsymbol{m})$. We will prove the lemma through a long sequence of hybrid sub-games $\mathsf{Hyb}^{1,0}, \dots, \mathsf{Hyb}^{\ell,8}, \mathsf{Hyb}^{\ell,8*}$.

First of all, we describe the following winning event:

$$\mathsf{WIN}_3 := \begin{cases} \forall i \in [\ell], (\tau_i^*, m_i) \in \mathsf{L}_{\mathsf{Sig}} \\ \wedge \{\mathsf{id} \in \mathcal{P}^*\} \cap \mathsf{L}_{\mathsf{Corr}} = \emptyset \\ \wedge \mathsf{BARG.Ver}(\mathsf{crs}, C, \{\mathrm{x}_i\}_i, \pi_\sigma) = 1 \\ \wedge \mathsf{com}^* \neq \mathsf{FC.Com}(\mathsf{ck}, \boldsymbol{m}) \end{cases}.$$

As in previous lemmas, note that WIN_3 only includes a subset of the checks in $\mathsf{MKHS.Ver}$, as the other conditions (in particular, FC verification) are not relevant for this lemma. Based on this winning condition, we introduce an initial $\mathsf{Hyb}^{1,0}$ in Figure 7.8 as a simplification of UF_3 where we omit the outputs $\pi_f, \boldsymbol{y}^*$ from the adversary. As the winning condition in

$\mathsf{Hyb}^{1,0}$ is less strict than in UF_3 while the pre-conditions remain the same, any adversary winning in UF_3 also wins in $\mathsf{Hyb}^{1,0}$. Hence, $\Pr[\mathsf{UF}_{3\mathcal{A}}(\lambda) = 1] \leq \Pr[\mathsf{Hyb}^{1,0}_{\mathcal{A}}(\lambda) = 1]$.

Games $\mathsf{Hyb}^{1,j}$: We formally introduce the hybrid games in Figures 7.8, 7.9, 7.10, and 7.11. We progress through the hybrids below.

$\mathsf{Hyb}^{1,0}_{\mathcal{A}}(\lambda)$:	$\mathsf{Hyb}^{1,1}_{\mathcal{A}}(\lambda)$:
$\mathsf{crs} \leftarrow \mathsf{BARG.Setup}()$	$(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{BARG.TdSetup}(1)$
$\mathsf{dk} \leftarrow \mathsf{SEC.Setup}()$	$\mathsf{dk} \leftarrow \mathsf{SEC.Setup}()$
$\mathsf{ck} \leftarrow \mathsf{FC.Setup}()$	$\mathsf{ck} \leftarrow \mathsf{FC.Setup}()$
$(\mathcal{P}^*, \{\mathtt{x}^*_i\}_i, \mathsf{com}^*, \pi_\sigma, \mathsf{com}_w)$	$(\mathcal{P}^*, \{\mathtt{x}^*_i\}_i, \mathsf{com}^*, \pi_\sigma, \mathsf{com}_w)$
$\quad\leftarrow \mathcal{A}^{O}(\mathsf{crs}, \mathsf{dk}, \mathsf{ck})$	$\quad\leftarrow \mathcal{A}^{O}(\mathsf{crs}, \mathsf{dk}, \mathsf{ck})$
Output 1 iff $\mathsf{WIN}_3 = 1$	**Output 1 iff** $\mathsf{WIN}_3 = 1$

Figure 7.8: *Games* $\mathsf{Hyb}^{1,0}$, $\mathsf{Hyb}^{1,1}$ *for the proof of Lemma 7.16. We highlight changes between games and omit inputs to* Setup *for succinctness.*

$\mathsf{Hyb}^{1,2}_{\mathcal{A}}(\lambda)$:	$\mathsf{Hyb}^{1,3}_{\mathcal{A}}(\lambda)$:
$(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{BARG.TdSetup}(1)$	$(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{BARG.TdSetup}(1)$
$\mathsf{dk} \leftarrow \mathsf{SEC.Setup}()$	$(\mathsf{dk}, \mathsf{td}_c) \leftarrow \mathsf{SEC.TdSetup}(1)$
$\mathsf{ck} \leftarrow \mathsf{FC.Setup}()$	$\mathsf{ck} \leftarrow \mathsf{FC.Setup}()$
$(\mathcal{P}^*, \{\mathtt{x}^*_i\}_i, \mathsf{com}^*, \pi_\sigma, \mathsf{com}_w)$	$(\mathcal{P}^*, \{\mathtt{x}^*_i\}_i, \mathsf{com}^*, \pi_\sigma, \mathsf{com}_w)$
$\quad\leftarrow \mathcal{A}^{O}(\mathsf{crs}, \mathsf{dk}, \mathsf{ck})$	$\quad\leftarrow \mathcal{A}^{O}(\mathsf{crs}, \mathsf{dk}, \mathsf{ck})$
$\bar{\mathtt{w}}_1 \leftarrow \mathsf{BARG.Ext}(\mathsf{td}, C, \pi_\sigma)$	$\bar{\mathtt{w}}_1 \leftarrow \mathsf{BARG.Ext}(\mathsf{td}, C, \pi_\sigma)$
Output 1 iff:	**Output 1 iff:**
$\quad\mathsf{WIN}_3 = 1$	$\quad\mathsf{WIN}_3 = 1$
$\quad\wedge\, C(\mathtt{x}_1, \bar{\mathtt{w}}_1) = 1$	$\quad\wedge\, C(\mathtt{x}_1, \bar{\mathtt{w}}_1) = 1$

Figure 7.9: *Games* $\mathsf{Hyb}^{1,2}$, $\mathsf{Hyb}^{1,3}$ *for the proof of Lemma 7.16.*

$\underline{\mathsf{Hyb}^{1,1}}$**:** The transition from $\mathsf{Hyb}^{1,0}$ to $\mathsf{Hyb}^{1,1}$, where we switch $\mathsf{BARG.Setup}$ to trapdoor mode $\mathsf{BARG.TdSetup}(1)$ at index 1, follows easily by the setup indistinguishability property of BARG. We have that there exists a PPT adversary $\mathcal{B}_1$ against BARG setup indistinguishability such that

$$\Pr[\mathsf{Hyb}^{1,0}_{\mathcal{A}}(\lambda) = 1] \leq \Pr[\mathsf{Hyb}^{1,1}_{\mathcal{A}}(\lambda) = 1] + \mathsf{Adv}^{\mathrm{sind}}_{\mathsf{BARG},\mathcal{B}_1}(\lambda).$$

$\underline{\mathsf{Hyb}^{1,2}}$**:** In this step, we additionally extract from BARG at position 1 and abort if $C(\mathtt{x}_1, \bar{\mathtt{w}}_1) \neq 1$ for the extracted witness $\bar{\mathtt{w}}_1$. The witness is given by $\bar{\mathtt{w}}_1 = (\bar{m}_1, \bar{\sigma}_1, \bar{\pi}_1, \bar{\mathsf{com}}_0, \bar{\mathsf{com}}_1, \bar{o}_0, \bar{o}_1)$, where $\bar{\mathsf{com}}_0$ and $\bar{o}_0$ are irrelevant for the proof. It follows that there exists a PPT adversary $\mathcal{B}_2$ against BARG somewhere extractability such that

$$\Pr[\mathsf{Hyb}^{1,1}_{\mathcal{A}}(\lambda) = 1] \leq \Pr[\mathsf{Hyb}^{1,2}_{\mathcal{A}}(\lambda) = 1] + \mathsf{Adv}^{\mathrm{sext}}_{\mathsf{BARG},\mathcal{B}_2}(\lambda).$$

$\underline{\mathsf{Hyb}^{1,3}}$**:** In this game, we set SEC.Setup extractable at index 1. We have that there exists an adversary $\mathcal{B}_4$ against SEC setup indistinguishability such that

$$\Pr[\mathsf{Hyb}^{1,2}_{\mathcal{A}}(\lambda) = 1] \leq \Pr[\mathsf{Hyb}^{1,3}_{\mathcal{A}}(\lambda) = 1] + \mathsf{Adv}^{\mathrm{sind}}_{\mathsf{SEC}}(\lambda).$$

$\mathsf{Hyb}^{1,4}_{\mathcal{A}}(\lambda)$:	$\mathsf{Hyb}^{1,5}_{\mathcal{A}}(\lambda)$:
$(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{BARG.TdSetup}(1)$	$(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{BARG.TdSetup}(1)$
$(\mathsf{dk}, \mathsf{td}_c) \leftarrow \mathsf{SEC.TdSetup}(1)$	$(\mathsf{dk}, \mathsf{td}_c) \leftarrow \mathsf{SEC.TdSetup}(1)$
$\mathsf{ck} \leftarrow \mathsf{FC.Setup}()$	$\mathsf{ck} \leftarrow \mathsf{FC.Setup}()$
$(\mathcal{P}^*, \{\mathtt{x}^*_i\}_i, \mathsf{com}^*, \pi_\sigma, \mathsf{com}_w)$	$(\mathcal{P}^*, \{\mathtt{x}^*_i\}_i, \mathsf{com}^*, \pi_\sigma, \mathsf{com}_w)$
$\quad \leftarrow \mathcal{A}^{O}(\mathsf{crs}, \mathsf{dk}, \mathsf{ck})$	$\quad \leftarrow \mathcal{A}^{O}(\mathsf{crs}, \mathsf{dk}, \mathsf{ck})$
$\bar{\mathtt{w}}_1 \leftarrow \mathsf{BARG.Ext}(\mathsf{td}, C, \pi_\sigma)$	$\bar{\mathtt{w}}_1 \leftarrow \mathsf{BARG.Ext}(\mathsf{td}, C, \pi_\sigma)$
$\hat{\mathsf{com}}_1 \leftarrow \mathsf{SEC.Ext}(\mathsf{td}_c, \mathsf{com}_w)$	$\hat{\mathsf{com}}_1 \leftarrow \mathsf{SEC.Ext}(\mathsf{td}_c, \mathsf{com}_w)$
Output 1 iff:	**Output 1 iff:**
$\quad \mathsf{WIN}_3 = 1$	$\quad \mathsf{WIN}_3 = 1$
$\quad \wedge\, C(\mathtt{x}_1, \bar{\mathtt{w}}_1) = 1$	$\quad \wedge\, C(\mathtt{x}_1, \bar{\mathtt{w}}_1) = 1$
$\quad \wedge\, \hat{\mathsf{com}}_1 = \bar{\mathsf{com}}_1$	$\quad \wedge\, \hat{\mathsf{com}}_1 = \bar{\mathsf{com}}_1$
	$\quad \wedge\, \bar{m}_1 = m_1$

Figure 7.10: *Games* $\mathsf{Hyb}^{1,4}, \mathsf{Hyb}^{1,5}$ *for the proof of Lemma 7.16.*

$\underline{\mathsf{Hyb}^{1,4}}$**:** In this game, we extract $\hat{\mathsf{com}}_1 \leftarrow \mathsf{SEC.Ext}(\mathsf{td}_c, \mathsf{com}_w)$ and abort if $\hat{\mathsf{com}}_1 \neq \bar{\mathsf{com}}_1$. To prove the transition from the previous game, note that, by definition, $C(\mathtt{x}_1, \bar{\mathtt{w}}_1) = 1$ implies that $\mathsf{SEC.Ver}(\mathsf{dk}_1, \mathsf{com}_w, 1, \bar{\mathsf{com}}_1, \bar{o}_1) = 1$. Hence, if $\mathsf{SEC.Ext}(\mathsf{td}_c, \mathsf{com}_w, i^*) \neq \bar{\mathsf{com}}_{i^*}$, then we can create an adversary $\mathcal{B}_3$ against SEC somewhere extractability. We have that

$$\Pr[\mathsf{Hyb}^{1,3}_{\mathcal{A}}(\lambda) = 1] \leq \Pr[\mathsf{Hyb}^{1,4}_{\mathcal{A}}(\lambda) = 1] + \mathsf{Adv}^{\mathrm{sext}}_{\mathsf{SEC},\mathcal{B}_3}(\lambda).$$

$\underline{\mathsf{Hyb}^{1,5}}$**:** In this game, we add the requirement that the extracted $\bar{m}_1 \in \bar{\mathtt{w}}_1$ equals the honest m_1, where m_1 is the message that $\mathcal{A}$ queries to the O^{Sign} oracle on label $\tau^*_1 \in \mathcal{P}^*$. By definition of C, we have that $C(\mathtt{x}_1, \bar{\mathtt{w}}_1) = 1$ and therefore $\Sigma.\mathsf{Ver}(\mathsf{vk}_1, \bar{m}_1|\tau^*_1, \bar{\sigma}_1) = 1$. If $m_1 \neq \bar{m}_1$, then $\mathcal{A}$ must have produced a signature forgery $(\bar{m}_1|\tau^*_1, \bar{\sigma}_1)$ for key vk_1.

In a more careful analysis, we bound the probability of this event by constructing an adversary $\mathcal{B}_5$ against the unforgeability of the signature scheme Σ. $\mathcal{B}_5$ runs on input a verification key vk^* from the EUF-CMA challenger; it chooses a random index $q^* \leftarrow\!\$\, [Q]$, where $Q = \mathsf{poly}(\lambda)$ is the number of queries that $\mathcal{A}$ can make to the O^{KeyGen} oracle, and then it adaptively simulates all O^{KeyGen}, O^{Sign} and O^{Corr} queries for $\mathcal{A}$ as follows:

- For the i-th query to $O^{\mathsf{KeyGen}}(\mathsf{id})$, if $i = q^*$, return $\mathsf{vk}_{\mathsf{id}} = \mathsf{vk}^*$, otherwise generate a keypair $(\mathsf{vk}_{\mathsf{id}}, \mathsf{sk}_{\mathsf{id}}) \leftarrow \Sigma.\mathsf{KeyGen}(1^\lambda)$ and return $\mathsf{vk}_{\mathsf{id}}$.
- All $O^{\mathsf{Sign}}((\mathsf{id}, \cdot), \cdot)$ queries such that $\mathsf{id} = \mathsf{id}_{q^*}$ are answered using the $O^{\mathsf{Sign}}(\cdot)$ oracle of the EUF-CMA challenger, where all the remaining queries are answered by using the secret key $\mathsf{sk}_{\mathsf{id}}$, which is known to $\mathcal{B}_5$.

- If $\mathcal{A}$ makes a query $\mathcal{O}^{\mathsf{Corr}}(\mathsf{id})$ such that $\mathsf{id} = \mathsf{id}_{q^*}$ abort, otherwise return the corresponding secret key $\mathsf{sk}_{\mathsf{id}}$.

After $\mathcal{A}$ outputs $(\mathcal{P}^*, \{\mathtt{x}_i^*\}_i, \mathsf{com}^*, \pi_\sigma, \mathsf{com}_w)$, $\mathcal{B}_5$ parses the labels $\tau_1^* = (\mathsf{id}_1^*, \cdot)$ in $\mathcal{P}^*$ and aborts if $\mathsf{id}_1^* \neq \mathsf{id}_{q^*}$.

If $\mathcal{B}_5$ does not abort, then the simulation is perfect and it must be that $(\bar{m}_1|\tau_1^*, \bar{\sigma}_1)$ is a valid forgery for the EUF-CMA game. This holds as the WIN_3 condition enforces that $\mathcal{A}$ is only allowed to output labels τ_i^* such that (τ_i^*, m_i) was queried to $\mathcal{O}^{\mathsf{Sign}}$. Moreover, WIN_3 also enforces that vk_1 is not a corrupted key. Namely, id_1^* is one of the non-corrupted keys and thus the probability that $\mathcal{B}_5$ does not abort is $1/Q$. Thus, we conclude:

$$\Pr[\mathsf{Hyb}_{\mathcal{A}}^{1,4}(\lambda) = 1] \leq \Pr[\mathsf{Hyb}_{\mathcal{A}}^{1,5}(\lambda) = 1] + Q \cdot \mathsf{Adv}_{\Sigma,\mathcal{B}_5}^{\mathrm{eufcma}}(\lambda).$$

```
$\mathsf{Hyb}_{\mathcal{A}}^{1,6}(\lambda)$:
$(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{BARG.TdSetup}(1)$
$(\mathsf{dk}, \mathsf{td}_c) \leftarrow \mathsf{SEC.TdSetup}(1)$
$\mathsf{ck} \leftarrow \mathsf{FC.Setup}()$
$(\mathcal{P}^*, \{\mathtt{x}_i^*\}_i, \mathsf{com}^*, \pi_\sigma, \mathsf{com}_w)$
    $\leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{crs}, \mathsf{dk}, \mathsf{ck})$
$\bar{\mathtt{w}}_1 \leftarrow \mathsf{BARG.Ext}(\mathsf{td}, C, \pi_\sigma)$
$\hat{\mathsf{com}}_1 \leftarrow \mathsf{SEC.Ext}(\mathsf{td}_c, \mathsf{com}_w)$
$\mathsf{com}_1 \leftarrow \mathsf{FC.Com}(\mathsf{ck}, (m_1, \mathbf{0}))$
Output 1 iff:
    $\mathsf{WIN}_3 = 1$
    $\wedge\ C(\mathtt{x}_1, \bar{\mathtt{w}}_1) = 1$
    $\wedge\ \hat{\mathsf{com}}_1 = \bar{\mathsf{com}}_1 = \mathsf{com}_1$
    $\wedge\ \bar{m}_1 = m_1$
```

```
$\mathsf{Hyb}_{\mathcal{A}}^{1,7}(\lambda)$:
$(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{BARG.TdSetup}(1)$
$(\mathsf{dk}, \mathsf{td}_c) \leftarrow \mathsf{SEC.TdSetup}(1)$
$\mathsf{ck} \leftarrow \mathsf{FC.Setup}()$
$(\mathcal{P}^*, \{\mathtt{x}_i^*\}_i, \mathsf{com}^*, \pi_\sigma, \mathsf{com}_w)$
    $\leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{crs}, \mathsf{dk}, \mathsf{ck})$

$\hat{\mathsf{com}}_1 \leftarrow \mathsf{SEC.Ext}(\mathsf{td}_c, \mathsf{com}_w)$
$\mathsf{com}_1 \leftarrow \mathsf{FC.Com}(\mathsf{ck}, (m_1, \mathbf{0}))$
Output 1 iff:
    $\mathsf{WIN}_3 = 1$
    $\wedge\ \hat{\mathsf{com}}_1 = \mathsf{com}_1$
```

Figure 7.11: *Games* $\mathsf{Hyb}^{1,6}$, $\mathsf{Hyb}^{1,7}$ *for the proof of Lemma 7.16.*

$\underline{\mathsf{Hyb}^{1,6}}$**:** In this game, we compute the honest partial commitment $\mathsf{com}_1 \leftarrow \mathsf{FC.Com}(\mathsf{ck}, (m_1, 0, \ldots, 0))$, and require that $\bar{\mathsf{com}}_1 = \mathsf{com}_1$, and by extension, that $\hat{\mathsf{com}}_1 = \mathsf{com}_1$. This step follows easily by the updatability property of FC, since $C(\mathtt{x}_1, \bar{\mathtt{w}}_1) = 1$ only holds if $\mathsf{FC.VerUpd}(\mathsf{ck}_1, 1, \bar{\mathsf{com}}_0, 0, \bar{\mathsf{com}}_1, \bar{m}_1, \bar{\pi}_1) = 1$. As $\bar{m}_1 = m_1$, if $\bar{\mathsf{com}}_1 \neq \mathsf{com}_1$ then we break FC updatability soundness. Hence, there exists an adversary $\mathcal{B}_6$ against FC updatability such that

$$\Pr[\mathsf{Hyb}_{\mathcal{A}}^{1,5}(\lambda) = 1] \leq \Pr[\mathsf{Hyb}_{\mathcal{A}}^{1,6}(\lambda) = 1] + \mathsf{Adv}_{\mathsf{FC},\mathcal{B}_6}^{\mathrm{updbind}}(\lambda).$$

$\underline{\mathsf{Hyb}^{1,7}}$**:** This game is a simplification of $\mathsf{Hyb}^{1,6}$ where we no longer extract from BARG, and hence we no longer have $C(\mathtt{x}_1, \bar{\mathtt{w}}_1) = 1 \wedge \bar{m}_1 = m_1 \wedge \hat{\mathsf{com}}_1 = \bar{\mathsf{com}}_1$ in the winning condition.

We have that

$$\Pr[\mathsf{Hyb}^{1,6}_{\mathcal{A}}(\lambda) = 1] = \Pr[\mathsf{Hyb}^{1,7}_{\mathcal{A}}(\lambda) = 1 \wedge C(\mathtt{x}_1, \bar{\mathtt{w}}_1) = 1 \wedge \bar{m}_1 = m_1 \wedge \hat{\mathsf{com}}_1 = \bar{\mathsf{com}}_1]$$
$$\leq \Pr[\mathsf{Hyb}^{1,7}_{\mathcal{A}}(\lambda) = 1].$$

Note, this simplification makes the winning condition of $\mathsf{Hyb}^{1,7}$ independent of BARG extraction, which is crucial for changing the extraction index in the subsequent hybrid.

Games $\mathsf{Hyb}^{i,j}$ for $2 \leq i < \ell$: We introduce the hybrid games in Figures 7.12, 7.13, 7.14. First of all, we analyze the step from $\mathsf{Hyb}^{1,7}$ to $\mathsf{Hyb}^{2,1}$. Then, we analyze the generic step from $\mathsf{Hyb}^{i-1,9}$ to $\mathsf{Hyb}^{i,1}$ for $i > 2$, and proceed with the remaining hybrids.

$\mathsf{Hyb}^{i,1}_{\mathcal{A}}(\lambda)$:	$\mathsf{Hyb}^{i,2}_{\mathcal{A}}(\lambda)$:	$\mathsf{Hyb}^{i,3}_{\mathcal{A}}(\lambda)$:
$(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{BARG.TdSetup}(i)$	$(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{BARG.TdSetup}(i)$	$(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{BARG.TdSetup}(i)$
$(\mathsf{dk}, \mathsf{td}_c) \leftarrow \mathsf{SEC.Setup}(i-1)$	$(\mathsf{dk}, \mathsf{td}_c) \leftarrow \mathsf{SEC.Setup}(i-1)$	$(\mathsf{dk}, \mathsf{td}_c) \leftarrow \mathsf{SEC.Setup}(i-1)$
$\mathsf{ck} \leftarrow \mathsf{FC.Setup}()$	$\mathsf{ck} \leftarrow \mathsf{FC.Setup}()$	$\mathsf{ck} \leftarrow \mathsf{FC.Setup}()$
$(\mathcal{P}^*, \{\mathtt{x}^*_i\}_i, \mathsf{com}^*, \pi_\sigma, \mathsf{com}_w)$	$(\mathcal{P}^*, \{\mathtt{x}^*_i\}_i, \mathsf{com}^*, \pi_\sigma, \mathsf{com}_w)$	$(\mathcal{P}^*, \{\mathtt{x}^*_i\}_i, \mathsf{com}^*, \pi_\sigma, \mathsf{com}_w)$
$\leftarrow \mathcal{A}^{O}(\mathsf{crs}, \mathsf{dk}, \mathsf{ck})$	$\leftarrow \mathcal{A}^{O}(\mathsf{crs}, \mathsf{dk}, \mathsf{ck})$	$\leftarrow \mathcal{A}^{O}(\mathsf{crs}, \mathsf{dk}, \mathsf{ck})$
$\hat{\mathsf{com}}_{i-1} \leftarrow \mathsf{SEC.Ext}(\mathsf{td}_c, \mathsf{com}_w)$	$\bar{\mathtt{w}}_i \leftarrow \mathsf{BARG.Ext}(\mathsf{td}, C, \pi_\sigma)$	$\bar{\mathtt{w}}_i \leftarrow \mathsf{BARG.Ext}(\mathsf{td}, C, \pi_\sigma)$
$\mathsf{com}_{i-1} \leftarrow$	$\hat{\mathsf{com}}_{i-1} \leftarrow \mathsf{SEC.Ext}(\mathsf{td}_c, \mathsf{com}_w)$	$\hat{\mathsf{com}}_{i-1} \leftarrow \mathsf{SEC.Ext}(\mathsf{td}_c, \mathsf{com}_w)$
$\mathsf{FC.Com}(\mathsf{ck}, (\boldsymbol{m}_{[1:i-1]}, \mathbf{0}))$	$\mathsf{com}_{i-1} \leftarrow$	$\mathsf{com}_{i-1} \leftarrow$
Output 1 iff:	$\mathsf{FC.Com}(\mathsf{ck}, (\boldsymbol{m}_{[1:i-1]}, \mathbf{0}))$	$\mathsf{FC.Com}(\mathsf{ck}, (\boldsymbol{m}_{[1:i-1]}, \mathbf{0}))$
$\mathsf{WIN}_3 = 1$	**Output 1 iff:**	**Output 1 iff:**
$\wedge\ \hat{\mathsf{com}}_{i-1} = \mathsf{com}_{i-1}$	$\mathsf{WIN}_3 = 1$	$\mathsf{WIN}_3 = 1$
	$\wedge\ C(\mathtt{x}_i, \bar{\mathtt{w}}_i) = 1$	$\wedge\ C(\mathtt{x}_i, \bar{\mathtt{w}}_i) = 1$
	$\wedge\ \hat{\mathsf{com}}_{i-1} = \mathsf{com}_{i-1}$	$\wedge\ \hat{\mathsf{com}}_{i-1} = \bar{\mathsf{com}}_{i-1} = \mathsf{com}_{i-1}$

Figure 7.12: *Games* $\mathsf{Hyb}^{i,1}, \mathsf{Hyb}^{i,2}, \mathsf{Hyb}^{i,3}$ *for the proof of Lemma 7.16.*

$\mathsf{Hyb}^{2,1}$: In the transition from $\mathsf{Hyb}^{1,7}$ to $\mathsf{Hyb}^{2,1}$, we simply switch $\mathsf{BARG.TdSetup}(1)$ to the following index $\mathsf{BARG.TdSetup}(2)$. The step again follows by the setup indistinguishability of BARG. We have that

$$\Pr[\mathsf{Hyb}^{1,7}_{\mathcal{A}}(\lambda) = 1] \leq \Pr[\mathsf{Hyb}^{2,1}_{\mathcal{A}}(\lambda) = 1] + \mathsf{Adv}^{\mathrm{sind}}_{\mathsf{BARG},\mathcal{B}_1}(\lambda).$$

$\mathsf{Hyb}^{i,1}$: In the transition from $\mathsf{Hyb}^{i-1,9}$ to $\mathsf{Hyb}^{i,1}$, we also simply switch $\mathsf{BARG.TdSetup}(i-1)$ to the following index $\mathsf{BARG.TdSetup}(i)$. As above, the setup indistinguishability of BARG implies that

$$\Pr[\mathsf{Hyb}^{i,1}_{\mathcal{A}}(\lambda) = 1] \leq \Pr[\mathsf{Hyb}^{i-1,9}_{\mathcal{A}}(\lambda) = 1] + \mathsf{Adv}^{\mathrm{sind}}_{\mathsf{BARG},\mathcal{B}_1}(\lambda).$$

$\mathsf{Hyb}^{i,2}, \mathsf{Hyb}^{1,3}$: These steps are identical to the respective steps for $\mathsf{Hyb}^{1,2}, \mathsf{Hyb}^{1,3}$,

$$\Pr[\mathsf{Hyb}^{i,1}_{\mathcal{A}}(\lambda) = 1] \leq \Pr[\mathsf{Hyb}^{i,3}_{\mathcal{A}}(\lambda) = 1] + \mathsf{Adv}^{\mathrm{sext}}_{\mathsf{BARG},\mathcal{B}_2}(\lambda) + \mathsf{Adv}^{\mathrm{sind}}_{\mathsf{SEC},\mathcal{B}_3}(\lambda).$$

$\mathsf{Hyb}^{i,4}_{\mathcal{A}}(\lambda)$:	$\mathsf{Hyb}^{i,5}_{\mathcal{A}}(\lambda)$:	$\mathsf{Hyb}^{i,6}_{\mathcal{A}}(\lambda)$:
$(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{BARG.TdSetup}(i)$	$(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{BARG.TdSetup}(i)$	$(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{BARG.TdSetup}(i)$
$(\mathsf{dk}, \mathsf{td}_c) \leftarrow \mathsf{SEC.Setup}(i-1)$	$(\mathsf{dk}, \mathsf{td}_c) \leftarrow \mathsf{SEC.TdSetup}(i)$	$(\mathsf{dk}, \mathsf{td}_c) \leftarrow \mathsf{SEC.TdSetup}(i)$
$\mathsf{ck} \leftarrow \mathsf{FC.Setup}()$	$\mathsf{ck} \leftarrow \mathsf{FC.Setup}()$	$\mathsf{ck} \leftarrow \mathsf{FC.Setup}()$
$(\mathcal{P}^*, \{\mathtt{x}_i^*\}_i, \mathsf{com}^*, \pi_\sigma, \mathsf{com}_w)$	$(\mathcal{P}^*, \{\mathtt{x}_i^*\}_i, \mathsf{com}^*, \pi_\sigma, \mathsf{com}_w)$	$(\mathcal{P}^*, \{\mathtt{x}_i^*\}_i, \mathsf{com}^*, \pi_\sigma, \mathsf{com}_w)$
$\leftarrow \mathcal{A}^{O}(\mathsf{crs}, \mathsf{dk}, \mathsf{ck})$	$\leftarrow \mathcal{A}^{O}(\mathsf{crs}, \mathsf{dk}, \mathsf{ck})$	$\leftarrow \mathcal{A}^{O}(\mathsf{crs}, \mathsf{dk}, \mathsf{ck})$
$\bar{\mathtt{w}}_i \leftarrow \mathsf{BARG.Ext}(\mathsf{td}, C, \pi_\sigma)$	$\bar{\mathtt{w}}_i \leftarrow \mathsf{BARG.Ext}(\mathsf{td}, C, \pi_\sigma)$	$\bar{\mathtt{w}}_i \leftarrow \mathsf{BARG.Ext}(\mathsf{td}, C, \pi_\sigma)$
		$\hat{\mathsf{com}}_i \leftarrow \mathsf{SEC.Ext}(\mathsf{td}_c, \mathsf{com}_w)$
$\mathsf{com}_{i-1} \leftarrow$	$\mathsf{com}_{i-1} \leftarrow$	$\mathsf{com}_{i-1} \leftarrow$
$\mathsf{FC.Com}(\mathsf{ck}, (\boldsymbol{m}_{[1:i-1]}, \mathbf{0}))$	$\mathsf{FC.Com}(\mathsf{ck}, (\boldsymbol{m}_{[1:i-1]}, \mathbf{0}))$	$\mathsf{FC.Com}(\mathsf{ck}, (\boldsymbol{m}_{[1:i-1]}, \mathbf{0}))$
Output 1 iff:	**Output 1 iff:**	**Output 1 iff:**
$\mathsf{WIN}_3 = 1$	$\mathsf{WIN}_3 = 1$	$\mathsf{WIN}_3 = 1$
$\wedge\ C(\mathtt{x}_i, \bar{\mathtt{w}}_i) = 1$	$\wedge\ C(\mathtt{x}_i, \bar{\mathtt{w}}_i) = 1$	$\wedge\ C(\mathtt{x}_i, \bar{\mathtt{w}}_i) = 1$
$\wedge\ \hat{\mathsf{com}}_{i-1} = \mathsf{com}_{i-1}$	$\wedge\ \hat{\mathsf{com}}_{i-1} = \mathsf{com}_{i-1}$	$\wedge\ \hat{\mathsf{com}}_{i-1} = \mathsf{com}_{i-1}$
		$\wedge\ \hat{\mathsf{com}}_i = \hat{\mathsf{com}}_i$

Figure 7.13: *Games* $\mathsf{Hyb}^{i,4}$, $\mathsf{Hyb}^{i,5}$, $\mathsf{Hyb}^{i,6}$ *for the proof of Lemma 7.16.*

$\mathsf{Hyb}^{i,4}$: This game is a simplification of $\mathsf{Hyb}^{i,3}$ where we no longer extract from SEC, and therefore, the winning condition $\mathsf{com}_{i-1} = \hat{\mathsf{com}}_{i-1}$ also vanishes. Similarly to the proof for $\mathsf{Hyb}^{1,7}$, it follows that

$$\Pr[\mathsf{Hyb}^{i,3}_{\mathcal{A}}(\lambda) = 1] \leq \Pr[\mathsf{Hyb}^{i,4}_{\mathcal{A}}(\lambda) = 1].$$

$\mathsf{Hyb}^{i,5}$, $\mathsf{Hyb}^{i,6}$: These steps are nearly identical to $\mathsf{Hyb}^{1,3}$ and $\mathsf{Hyb}^{1,4}$, where we switch the extraction index of SEC (to index i), and then use the corresponding SEC extractor. By the setup indistinguishability and somewhere extractability of SEC, it follows that we can construct adversaries $\mathcal{B}_3$ and $\mathcal{B}_4$ such that

$$\Pr[\mathsf{Hyb}^{i,4}_{\mathcal{A}}(\lambda) = 1] \leq \Pr[\mathsf{Hyb}^{i,6}_{\mathcal{A}}(\lambda) = 1] + \mathsf{Adv}^{\mathrm{sind}}_{\mathsf{SEC},\mathcal{B}_3}(\lambda) + \mathsf{Adv}^{\mathrm{sext}}_{\mathsf{SEC},\mathcal{B}_4}(\lambda).$$

$\mathsf{Hyb}^{i,7}$: This game transition is as for $\mathsf{Hyb}^{1,5}$, where we rely on the unforgeability of Σ. The only difference is that in the reduction we must replace vk_1 by vk_i in the abort condition defined in the guessing argument. In an analog manner, it follows that

$$\Pr[\mathsf{Hyb}^{i,6}_{\mathcal{A}}(\lambda) = 1] \leq \Pr[\mathsf{Hyb}^{i,7}_{\mathcal{A}}(\lambda) = 1] + Q \cdot \mathsf{Adv}^{\mathrm{eufcma}}_{\Sigma,\mathcal{B}_5}(\lambda).$$

$\mathsf{Hyb}^{i,8}$, $\mathsf{Hyb}^{i,9}$: These steps are identical to those for $\mathsf{Hyb}^{1,6}$ and $\mathsf{Hyb}^{1,7}$, respectively. We have that, by FC updatability soundness, and by the simplification of the winning condition, there exists an adversary $\mathcal{B}_6$ such that

$$\Pr[\mathsf{Hyb}^{1,7}_{\mathcal{A}}(\lambda) = 1] \leq \Pr[\mathsf{Hyb}^{1,9}_{\mathcal{A}}(\lambda) = 1] + \mathsf{Adv}^{\mathrm{upbind}}_{\mathsf{FC},\mathcal{B}_6}(\lambda).$$

$\mathsf{Hyb}^{i,7}_{\mathcal{A}}(\lambda)$:	$\mathsf{Hyb}^{i,8}_{\mathcal{A}}(\lambda)$:	$\mathsf{Hyb}^{i,9}_{\mathcal{A}}(\lambda)$:
$(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{BARG.TdSetup}(i)$	$(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{BARG.TdSetup}(i)$	$(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{BARG.TdSetup}(i)$
$(\mathsf{dk}, \mathsf{td}_c) \leftarrow \mathsf{SEC.TdSetup}(i)$	$(\mathsf{dk}, \mathsf{td}_c) \leftarrow \mathsf{SEC.TdSetup}(i)$	$(\mathsf{dk}, \mathsf{td}_c) \leftarrow \mathsf{SEC.TdSetup}(i)$
$\mathsf{ck} \leftarrow \mathsf{FC.Setup}()$	$\mathsf{ck} \leftarrow \mathsf{FC.Setup}()$	$\mathsf{ck} \leftarrow \mathsf{FC.Setup}()$
$(\mathcal{P}^*, \{\mathtt{x}^*_i\}_i, \mathsf{com}^*, \pi_\sigma, \mathsf{com}_w)$	$(\mathcal{P}^*, \{\mathtt{x}^*_i\}_i, \mathsf{com}^*, \pi_\sigma, \mathsf{com}_w)$	$(\mathcal{P}^*, \{\mathtt{x}^*_i\}_i, \mathsf{com}^*, \pi_\sigma, \mathsf{com}_w)$
$\leftarrow \mathcal{A}^{O}(\mathsf{crs}, \mathsf{dk}, \mathsf{ck})$	$\leftarrow \mathcal{A}^{O}(\mathsf{crs}, \mathsf{dk}, \mathsf{ck})$	$\leftarrow \mathcal{A}^{O}(\mathsf{crs}, \mathsf{dk}, \mathsf{ck})$
$\bar{\mathtt{w}}_i \leftarrow \mathsf{BARG.Ext}(\mathsf{td}, C, \pi_\sigma)$	$\bar{\mathtt{w}}_i \leftarrow \mathsf{BARG.Ext}(\mathsf{td}, C, \pi_\sigma)$	
$\widehat{\mathsf{com}}_i \leftarrow \mathsf{SEC.Ext}(\mathsf{td}_c, \mathsf{com}_w)$	$\widehat{\mathsf{com}}_i \leftarrow \mathsf{SEC.Ext}(\mathsf{td}_c, \mathsf{com}_w)$	$\widehat{\mathsf{com}}_i \leftarrow \mathsf{SEC.Ext}(\mathsf{td}_c, \mathsf{com}_w)$
$\mathsf{com}_{i-1} \leftarrow$	$\mathsf{com}_{i-1} \leftarrow$	
$\mathsf{FC.Com}(\mathsf{ck}, (\boldsymbol{m}_{[1:i-1]}, \mathbf{0}))$	$\mathsf{FC.Com}(\mathsf{ck}, (\boldsymbol{m}_{[1:i-1]}, \mathbf{0}))$	$\mathsf{com}_i \leftarrow \mathsf{FC.Com}(\mathsf{ck}, (\boldsymbol{m}_{[1:i]}, \mathbf{0}))$
Output 1 iff:	$\mathsf{com}_i \leftarrow \mathsf{FC.Com}(\mathsf{ck}, (\boldsymbol{m}_{[1:i]}, \mathbf{0}))$	**Output 1 iff:**
$\mathsf{WIN}_3 = 1$	**Output 1 iff:**	$\mathsf{WIN}_3 = 1$
$\wedge\ C(\mathtt{x}_i, \bar{\mathtt{w}}_i) = 1$	$\mathsf{WIN}_3 = 1$	$\wedge\ \bar{\mathsf{com}}_i = \mathsf{com}_i$
$\wedge\ \bar{\mathsf{com}}_{i-1} = \mathsf{com}_{i-1}$	$\wedge\ C(\mathtt{x}_i, \bar{\mathtt{w}}_i) = 1$	
$\wedge\ \bar{\mathsf{com}}_i = \widehat{\mathsf{com}}_i$	$\wedge\ \bar{\mathsf{com}}_{i-1} = \mathsf{com}_{i-1}$	
$\wedge\ \bar{m}_i = m_i$	$\wedge\ \bar{\mathsf{com}}_i = \widehat{\mathsf{com}}_i = \mathsf{com}_i$	
	$\wedge\ \bar{m}_i = m_i$	

Figure 7.14: *Games* $\mathsf{Hyb}^{i,7}$, $\mathsf{Hyb}^{i,8}$, $\mathsf{Hyb}^{i,9}$ *for the proof of Lemma 7.16.*

Games $\mathsf{Hyb}^{\ell,j}$: Games $\mathsf{Hyb}^{\ell,1}$ to $\mathsf{Hyb}^{\ell,8}$ are defined as the games $\mathsf{Hyb}^{i,1}$ to $\mathsf{Hyb}^{i,8}$, for $i = \ell$, in Figures 7.12, 7.13, 7.14, and the reduction steps are identical for these cases. To analyze the advantage of the adversary in $\mathsf{Hyb}^{\ell,8}$, we introduce an additional $\mathsf{Hyb}^{\ell,8*}$, that we compare to the former in Figure 7.15.

Observe that $\mathsf{Hyb}^{\ell,8*}$ is just a simplification of game $\mathsf{Hyb}^{\ell,8}$ with an easier winning condition. Hence,

$$\Pr[\mathsf{Hyb}^{\ell,8}_{\mathcal{A}}(\lambda) = 1] \leq \Pr[\mathsf{Hyb}^{\ell,8*}_{\mathcal{A}}(\lambda) = 1]$$

Finally, note that the conditions WIN_3, $C(\mathtt{x}_\ell, \bar{\mathtt{w}}_\ell) = 1$, and $\bar{\mathsf{com}}_\ell = \mathsf{com}_\ell$ cannot occur simultaneously. The circuit $C(\mathtt{x}_\ell, \bar{\mathtt{w}}_\ell) = 1$ (Figure 7.5) checks, for $i = \ell$, that $\bar{\mathsf{com}}_\ell = \mathsf{com}^*$. Therefore, $\mathsf{com}^* = \mathsf{com}_\ell$ is honestly computed, which contradicts the winning condition WIN_3. We conclude that

$$\Pr[\mathsf{Hyb}^{\ell,8*}_{\mathcal{A}}(\lambda) = 1] = 0.$$

Proof summary. Putting all the intermediate bounds together, we obtain the following final bound:

$$\begin{aligned}\Pr[\mathsf{UF}_{3\mathcal{A}}(\lambda) = 1] \leq\ & \ell \cdot \mathsf{Adv}^{\mathrm{sind}}_{\mathsf{BARG},\mathcal{B}_1}(\lambda) + \ell \cdot \mathsf{Adv}^{\mathrm{sext}}_{\mathsf{BARG},\mathcal{B}_2}(\lambda) \\ & + (2n-1) \cdot \mathsf{Adv}^{\mathrm{sind}}_{\mathsf{SEC},\mathcal{B}_3}(\lambda) + \ell \cdot \mathsf{Adv}^{\mathrm{sext}}_{\mathsf{SEC},\mathcal{B}_4}(\lambda) \\ & + \ell \cdot Q \cdot \mathsf{Adv}^{\mathrm{eufcma}}_{\Sigma,\mathcal{B}_5}(\lambda) + \ell \cdot \mathsf{Adv}^{\mathrm{updbind}}_{\mathsf{FC},\mathcal{B}_6}(\lambda).\end{aligned}$$

$\mathsf{Hyb}^{\ell,8}_{\mathcal{A}}(\lambda)$:	$\mathsf{Hyb}^{\ell,8*}_{\mathcal{A}}(\lambda)$:
$(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{BARG.TdSetup}(\ell)$	$(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{BARG.TdSetup}(\ell)$
$(\mathsf{dk}, \mathsf{td}_c) \leftarrow \mathsf{SEC.TdSetup}(\ell)$	$(\mathsf{dk}, \mathsf{td}_c) \leftarrow \mathsf{SEC.TdSetup}(\ell)$
$\mathsf{ck} \leftarrow \mathsf{FC.Setup}()$	$\mathsf{ck} \leftarrow \mathsf{FC.Setup}()$
$(\mathcal{P}^*, \{\mathtt{x}^*_i\}_i, \mathsf{com}^*, \pi_\sigma, \mathsf{com}_w)$	$(\mathcal{P}^*, \{\mathtt{x}^*_i\}_i, \mathsf{com}^*, \pi_\sigma, \mathsf{com}_w)$
$\leftarrow \mathcal{A}^{O}(\mathsf{crs}, \mathsf{dk}, \mathsf{ck})$	$\leftarrow \mathcal{A}^{O}(\mathsf{crs}, \mathsf{dk}, \mathsf{ck})$
$\bar{\mathtt{w}}_\ell \leftarrow \mathsf{BARG.Ext}(\mathsf{td}, C, \pi_\sigma)$	$\bar{\mathtt{w}}_\ell \leftarrow \mathsf{BARG.Ext}(\mathsf{td}, C, \pi_\sigma)$
$\hat{\mathsf{com}}_\ell \leftarrow \mathsf{SEC.Ext}(\mathsf{td}_c, \mathsf{com}_w)$	$\hat{\mathsf{com}}_\ell \leftarrow \mathsf{SEC.Ext}(\mathsf{td}_c, \mathsf{com}_w)$
$\mathsf{com}_{\ell-1} \leftarrow \mathsf{FC.Com}(\mathsf{ck}, (\boldsymbol{m}_{[1:\ell-1]}, \mathbf{0}))$	
$\mathsf{com}_{\ell} \leftarrow \mathsf{FC.Com}(\mathsf{ck}, (\boldsymbol{m}_{[1:\ell]}, \mathbf{0}))$	$\mathsf{com}_{\ell} \leftarrow \mathsf{FC.Com}(\mathsf{ck}, (\boldsymbol{m}_{[1:\ell]}, \mathbf{0}))$
Output 1 iff:	**Output 1 iff:**
$\mathsf{WIN}_3 = 1$	$\mathsf{WIN}_3 = 1$
$\wedge\ C(\mathtt{x}_\ell, \bar{\mathtt{w}}_\ell) = 1$	$\wedge\ C(\mathtt{x}_\ell, \bar{\mathtt{w}}_\ell) = 1$
$\wedge\ \bar{\mathsf{com}}_{\ell-1} = \mathsf{com}_{\ell-1}$	$\wedge\ \bar{\mathsf{com}}_{\ell} = \mathsf{com}_{\ell}$
$\wedge\ \bar{\mathsf{com}}_{\ell} = \hat{\mathsf{com}}_{\ell} = \mathsf{com}_{\ell}$	
$\wedge\ \bar{m}_\ell = m_\ell$	

Figure 7.15: *Games* $\mathsf{Hyb}^{\ell,8}$, $\mathsf{Hyb}^{\ell,8*}$ *for the proof of Lemma 7.16.*

□

Lemma 7.17. *For any PPT adversary $\mathcal{A}$ making at most $Q = \mathsf{poly}(\lambda)$ queries to the key generation oracle and that can produce a valid forgery in UF_4, there exists a PPT adversary $\mathcal{B}_{\text{EUF-CMA}}$ against the* EUF-CMA *property of the digital signature scheme Σ, such that*

$$\Pr[\mathsf{UF}_{4,\mathcal{A}}(\lambda) = 1] \leq Q \cdot \mathsf{Adv}^{\text{euf-cma}}_{\Sigma,\mathcal{B}_{\text{EUF-CMA}}}(\lambda).$$

Proof. The proof of this lemma follows virtually the same reduction strategy to unforgeability as in the previous lemmas (the proof of the last hybrid in Lemma 7.14 and the proofs for hybrids $\mathsf{Hyb}^{1,4} \approx \mathsf{Hyb}^{1,5}$ in Lemma 7.16). Therefore we only give a sketch to highlight the main differences. The reduction starts by making a guess (which is correct with probability $1/Q$) about the index of the key generation query that gives the verification key that will be used in the forgery. If the guess is correct, a MKHS forgery (τ^*, y^*, σ^*) gives a signature on the message $y^*|\tau^*$. If the MKHS forgery is of type 1, then the message is new since no message with suffix τ^* was asked to the signing oracle (as in Lemma 7.14). If instead it is a MKHS forgery of type 2 then the message is new since the signing oracle was queried on $m|\tau^*$ for $m \neq y^*$, and on no other message with label τ^* due to the rule of the MKHS security game (as in Lemma 7.16). □

7.5 Extensions and Instantiations

In this section, we extend our base MKHS construction to support sequential multi-hop evaluation. Later, we describe a variety of instantiations of MKHS from falsifiable (and standard) assumptions obtained through BARGs, FCs and SECs introduced in previous works.

7.5.1 Multi-Hop Evaluation

We show how to adapt our MKHS construction in Figure 7.4 to support multi-hop evaluation of sequential functions $f^{(h)}(f^{(h-1)}(\cdots f^{(1)}(\cdot)))$. This construction relies on the same primitives as before, except that we require a *chainable* functional commitment CFC instead of a FC. We remark that, by applying Theorem 4.26, we can generically turn any FC for circuits into a CFC for circuits. The scheme supports the same labels and messages as the single-hop scheme.

First, we define a param structure for the input taken by the Eval algorithm with function f, such that we can distinguish whether f does a first-hop evaluation, or whether we compute over a previous output of Eval.

$$\mathsf{param} = \begin{cases} (\tau_i, \mathsf{vk}_i, m_i, \sigma_i)_{i\in[\ell]} & \text{if } h = 1 \\ (\mathcal{P}, \{\mathsf{vk}_i\}_{i\in\mathcal{P}}, \boldsymbol{m}^{(h-1)}, \sigma) & \text{if } h > 1 \end{cases}.$$

We introduce the scheme in Figure 7.16. The Setup, KeyGen, and Sign algorithms remain as in Figure 7.4. For security, note that the signature $\sigma_{f,y}$ is as in our single-hop MKHS, except that it includes multiple CFC commitments and opening proofs $(\pi_f^{(j)}, \mathsf{com}^{(j-1)})_{j\in[h]}$. The key observation is that we can see $\bar{\pi}_f := (\pi_f^{(j)}, \mathsf{com}^{(j-1)})_{j\in[h]}$ as the opening proof for $f^{(h)} \odot \cdots \odot f^{(1)}$ on $\mathsf{com}^{(0)}$ in the generic CFC-to-FC construction from [BCFL23, Theorem 2]. Hence, from the security standpoint we can interpret the multi-hop scheme as our single-hop one instantiated with a different FC; thus the same security proof applies.

7.5.2 Instantiations of MKHS for all functions

We describe several instantiations for our construction in Section 7.4 that we obtain by instantiating its main building blocks. We focus on MKHS for *all functions*, that we model as either boolean or arithmetic *circuits of unbounded depth*. We discuss the properties of the resulting schemes, in particular their succinctness and the underlying assumptions.

We give two families of MKHS instantiations: those that use non-algebraic FCs and BARGs (internally relying on correlation-intractable hash functions (CIHs) and probabilistic checkable proofs (PCPs)), and those that use algebraic constructions of these schemes. CIH + PCP based constructions offer nearly-optimal asymptotic succinctness, but the concrete parameters suffer from an impractical blow-up. Algebraic BARGs and FCs have smaller concrete parameters, and although our MKHS construction makes non-black-box use of them, we believe that instantiations based on algebraic building blocks present a more promising avenue towards fully-algebraic, concretely-efficient future MKHS constructions.

```
$\underline{\mathsf{MKHS.Eval}(\mathsf{pp}, f := f^{(h)}, \mathsf{param}, h)}$ :
  If $h = 1$:
  • Parse $\mathsf{param} := (\tau_i, \mathsf{vk}_i, m_i, \sigma_i)_{i\in[\ell]}$.
  • Output $(\mathsf{com}_w, \pi_\sigma, \pi_f^{(1)}, \mathsf{com}^{(0)}) \leftarrow \mathsf{MKHS.Eval}_0(\mathsf{pp}, f^{(1)}, (\tau_i, \mathsf{vk}_i, m_i, \sigma_i)_{i\in[\ell]})$.
  If $h > 1$:
  • Parse $\mathsf{param} := (\mathcal{P}, \{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}}, \boldsymbol{m}^{(h-1)}, \sigma)$.
  • Parse $\sigma := (\mathsf{com}_w, \pi_\sigma, (\pi_f^{(j)}, \mathsf{com}^{(j-1)})_{j\in[h-1]})$.
  • Parse $\mathsf{pp} := (\mathsf{crs}, \mathsf{ck}, \mathsf{dk})$.
  • Compute $\boldsymbol{m}^{(h)} \leftarrow f^{(h)}(\boldsymbol{m}^{(h-1)})$.
  • $\mathsf{com}^{(h-1)} \leftarrow \mathsf{CFC.Com}(\mathsf{ck}, \boldsymbol{m}^{(h-1)})$.
  • $\pi_f^{(h)} \leftarrow \mathsf{CFC.FuncProve}(\mathsf{ck}, \boldsymbol{m}^{(h-1)}, f^{(h)})$.
  • Output $\sigma_{f,y} = (\mathsf{com}_w, \pi_\sigma, (\pi_f^{(j)}, \mathsf{com}^{(j-1)})_{j\in[h]})$.
$\underline{\mathsf{MKHS.Ver}(\mathsf{pp}, \mathcal{P}, \{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}}, \boldsymbol{y}, \sigma_{f,y})}$ :
  • Parse $\mathcal{P} := (f, \tau_1, \ldots, \tau_\ell)$ and $\{\tau_i := (\mathsf{id}_i, \tau_i)\}$.
  • If $\mathcal{P} = (f_{id}, \tau_1)$ then check that $\Sigma.\mathsf{Ver}(\mathsf{vk}_{\mathsf{id}_1}, \boldsymbol{y}|\tau_1, \sigma_{f,y}) = 1$.
  • Else, parse $\sigma_{f,y} := (\mathsf{com}_w, \pi_\sigma, (\pi_f^{(j)}, \mathsf{com}^{(j-1)})_{j\in[h]})$.
  • Parse $\mathsf{pp} := (\mathsf{crs}, \mathsf{ck}, \mathsf{dk})$.
  • Parse $f := (f^{(1)}, \ldots, f^{(h)})$.
  • Compute $\mathsf{com}^{(h)} \leftarrow \mathsf{CFC.Com}(\mathsf{ck}, \boldsymbol{y})$.
  • Compute $\mathsf{com}_0 \leftarrow \mathsf{FC.Com}(\mathsf{ck}, \mathbf{0})$.
  • Compute the BARG circuit $C$ described in Figure 7.5, hardcoding
    $\mathsf{com}^{(0)}, \mathsf{com}_w, \mathsf{com}_0$.
  • Given $\{\mathsf{vk}_i\}_i := \{\mathsf{vk}_{\mathsf{id}_i}\}_i$ and $\{\tau_i\}_i, \{\mathsf{ck}_i\}_i, \{\mathsf{dk}_i\}_i$, define $\mathrm{x}_i = (\mathsf{vk}_i, \mathsf{ck}_i, \mathsf{dk}_i, \mathsf{dk}_{i-1}, \tau_i, i)$.
  • $\forall j \in [h]$, check that $\mathsf{CFC.FuncVer}(\mathsf{ck}, \mathsf{com}^{(j-1)}, f^{(j)}, \mathsf{com}^{(j)}, \pi_f^{(j)}) = 1$.
  • Check that $\mathsf{BARG.Ver}(\mathsf{crs}, C, \{\mathrm{x}_i\}_i, \pi_\sigma) = 1$.
  • Output 1 if all checks pass.
```

Figure 7.16: MKHS.Eval *and* MKHS.Ver *algorithms of a multi-hop succinct multi-key homomorphic signature scheme* MKHS *constructed from a chainable functional commitment* CFC, *a BARG for NP* BARG, *a somewhere extractable commitment* SEC *and a digital signature* Σ. Eval_0 *is the single-hop* Eval *from Figure 7.4.*

MKHS for unbounded-depth circuits from CIH and PCPs. The natural choices for this family of BARGs are the constructions in [CGJ+23, CJJ22] from either subexponential DDH or LWE, respectively.[3] Their efficiency is later refined in [KLVW23].

For functional commitments, the asymptotically optimal choice is to extend the SNARG for RAM computations from [KLVW23], which can be seen as an FC for single-output boolean circuits $C : \{0,1\}^\ell \to \{0,1\}$. Such an FC can be constructed generically from BARGs, and hence from the same assumptions as before. Extending their SNARG to a fully-fledged FC for unbounded depth-circuits is not straightforward and requires a series of observations:

- The commitment scheme underlying their SNARG is deterministic and supports efficient local updatability as it is implemented as a Merkle tree.
- Their SNARG satisfies FC evaluation binding for RAM computations with a bounded number of steps, which can be represented by single-output boolean circuits $f : \{0,1\}^\ell \to \{0,1\}$ of bounded depth $d_{\max}$. To boost their scheme, we can apply our generic transformation to obtain a CFC from any FC (Theorem 4.26). Since committing to a Merkle tree can be carried out by a circuit of $\mathsf{poly}(\lambda, \log \ell)$ depth, the transformation yields a CFC for boolean circuits $f : \{0,1\}^\ell \to \{0,1\}^\ell$ of bounded depth $d'_{\max} \lesssim d_{\max}$, where the opening proofs have size $|\pi_f| = \mathsf{poly}(\lambda, \log \ell)$.
- Given such a CFC for boolean circuits of bounded depth, one can obtain a FC for circuits of unbounded depth d by applying the generic CFC-to-FC transformation from [BCFL23], which imposes a multiplicative overhead of d on the opening size. Overall, $|\pi_f| = \mathsf{poly}(\lambda, \log \ell) \cdot d$.

Corollary 7.18. *Assuming the hardness of either (1) subexponential decisional Diffie-Hellman (DDH), or (2) learning with errors, there exists a multi-key homomorphic signature for unbounded-depth boolean circuits $\mathcal{F} = \{f : \{0,1\}^\ell \to \{0,1\}^{\ell_y}\}$ with the following properties:*

- ***Public parameters size:*** $|\mathsf{pp}| = \mathsf{poly}(\lambda, \log \ell)$.
- ***Signature size:*** $|\sigma_{f,y}| = \mathsf{poly}(\lambda, \log \ell, \log \ell_y) \cdot d$.
- ***Efficient verification:*** *Both the labels and the function can be preprocessed. The online efficient verification algorithm runs in time* $\mathsf{poly}(\lambda, \log \ell, \ell_y) \cdot d$.
- ***Multi-hop evaluation*** *and* ***Context-hiding****.*

MKHS for unbounded-depth circuit from algebraic schemes. Our MKHS can be instantiated over bilinear groups by using the algebraic BARG from [WW22], which relies either on the subgroup decision assumption or on the k-Lin assumption for any $k \geq 2$. In [WW22], they also present companion constructions of somewhere extractable commitments from the same assumptions. For the FC, the most natural pairing-based choice are either the algebraic scheme from [BCFL23], which relies on the HiKer assumption, or the [WW24b], which relies on bilateral k-Lin. Notably, both FC schemes admit efficient local updatability, deterministic commitments, and supports efficient verification with preprocessing.

[3] In the same works, SECs are constructed from the same assumptions as a building block for BARGs.

Corollary 7.19. *Assuming the hardness of HiKer and either the subgroup decision assumption or k-Lin for $k \geq 2$, there exists a pairing-based MKHS for unbounded-depth arithmetic circuits $\mathcal{F} = \{f : \mathcal{M}^{\ell} \to \mathcal{M}^{\ell_y}\}$ of bounded width w with the following properties:*

- ***Public parameters size:*** $|\mathsf{pp}| = O(w^5)$
- ***Signature size:*** $|\sigma_{f,y}| = O(\lambda \cdot d^2) + \mathsf{poly}(\lambda)$. *In particular, the signature is fully succinct on both ℓ and ℓ_y.*
- ***Efficient verification:*** *Both the labels and the function can be preprocessed. The online efficient verification algorithm runs in time $O(\lambda \cdot d^2) + \mathsf{poly}(\lambda)$.*
- ***Multi-hop evaluation*** *and* ***Context-hiding****.*

Towards a lattice-based algebraic instatiation, we remark that no lattice-based algebraic BARGs exist up to date. For FCs, a natural choice may be the lattice-based (C)FC in [BCFL23], or the scheme that results after applying the transformation of Theorem 4.26 to the FC in [WW23a].

Improvements due to recent progress on FCs and BARGs. By applying two results that appeared after our results were published in [ABF24], we can reduce the succinctness of our MHKS scheme further. First, the succinct functional commitment by Wee and Wu [WW24b] allows us to replace the HiKer assumption from [BCFL23] by bilateral k-Lin, obtaining a MKHS for arithmetic circuits where the signature size and the efficient verification time do not grow with the circuit depth, yielding $|\sigma_{f,y}| = \mathsf{poly}(\lambda)$. On the other hand, the public parameters grow with the circuit size as $|\mathsf{pp}| = O\left(|f|^5\right)$.

Second, the circuit-succinct BARG from [BFL25a] that we introduce in Chapter 6 allow us to remove the dependency in the BARG circuit size. For some instantiations, particularly those with $O(\lambda)$-sized (a) commitments, signatures and somewhere extractable commitment openings, the BARG circuit also has a witness of size $O(\lambda)$. Assembling all pieces together, this yields a MKHS with evaluated signatures of size $|\sigma_{f,y}| = O(\lambda^2)$. The quadratic blow-up comes from committing to the witness of the BARG circuit (a few group elements, i.e., of size $O(\lambda)$) as boolean values, generating $O(\lambda)$ commitments each of $O(\lambda)$ size. This comes at the cost of increasing the public parameters to $|\mathsf{pp}| = O\left(\ell^5 |f|^5\right)$. An interesting open question is whether we can reduce the MKHS signatures even further, where a natural lower is $|\sigma_{f,y}| = O(\lambda)$.

Part II

Efficient Proof Systems and Applications

8

Modular Sumcheck Proofs and Applications to Machine Learning and Image Processing

This chapter is based on results from the article *"Modular Sumcheck Proofs with Applications to Machine Learning and Image Processing"* [BFG$^+$23], which was published at ACM CCS 2023.

The chapter is structured as follows. In Section 8.1, we introduce a summary of the contributions of this work. In Section 8.2, we present our modular framework for sumcheck-based proofs, which allows us to design and compose proof systems for specific functionalities. In Section 8.3, we show how to instantiate our framework based on multilinear polynomials, capturing known proof systems. In Section 8.4, we introduce a battery of composable proof systems for usual machine learning computations, including neural networks. Finally, in Section 8.5, we implement, analyze and evaluate our solutions both theoretically and empirically. We also released an open-source library for our sumcheck-based proofs which is available at `https://github.com/imdea-software/MSCProof`.

8.1 Contributions

In previous chapters of this thesis, we built proof systems from standard assumptions, where the focus was more on feasibility and generality than in actual practicality. In this chapter, we instead tackle the construction of practically efficient proof systems. With this goal in mind, we introduce a new framework for the modular design of sumcheck-based proof systems, and we use it to develop new efficient protocols for verifiable machine learning and image processing. Our solutions aim to combine the efficiency of dedicated protocols and the versatility of general-purpose schemes. More specifically, our contributions are the following:

A modular framework for sumcheck-based proofs. We develop our framework by identifying and abstracting away the key properties of a variety of proof systems based on the sumcheck protocol, including the well-established GKR protocol [GKR08]. Briefly speaking, these protocols proceed in a layer-by-layer fashion so that at each layer the prover starts by making a "promise" about the output, and later the verifier ends with a "promise" about the input. Their security guarantee is that if the input's "promise" is correct then the initial output's "promise" must be correct too.

We define our framework abstracting these protocols as follows:

- We introduce the notions of *fingerprinting scheme* and *verifiable evaluation scheme on fingerprinted data* (VE). Fingerprinting schemes characterize the aforementioned notion of "promise" and are essentially a mechanism that allows prover and verifier to succinctly represent vectors of inputs/outputs. VEs are interactive protocols in which the verifier works by only knowing fingerprints of inputs and outputs (and thus can run sublinear in the input/output size).
- We show a *generic composition theorem*: given two VEs for functions f_1 and f_2 and compatible fingerprints, one can build a VE for their (partial or total) composition $f(x, y) = f_2(f_1(x), y)$.
- We show that a *VE can be lifted to become an interactive proof* if the verifier computes the fingerprints of the inputs and outputs of the computation (but not of intermediate steps). We also show that a *VE can be compiled into a succinct argument* by using commit-and-prove arguments for the evaluation of fingerprints (instantiatable with polynomial commitments [KZG10]).
- We instantiate our fingerprints as evaluations of multilinear polynomials, and then we show how to capture a large class of existing protocols—such as the multilinear sumcheck protocol of [XZZ+19], GKR, and the efficient matrix multiplication from [Tha13]—under our framework.

By combining these results, we obtain a way to easily design sumcheck-based proof systems in a modular way. Following the principle of modularity, one needs only to focus on designing VE schemes for specific functionalities, a task that likely results in more lightweight solutions (as we confirm below). In particular, we may take advantage of many years of great research in the field, as our modular design allows us to nicely integrate previous tools and gadgets. Furthermore, the practicality of modular VEs is not only evident at design time, but also at implementation time, since the code can be designed in blocks, in a "Lego" manner.

Applications to verifiable machine learning and image processing. We apply our approach to construct efficient proofs of computation for (convolutional) neural networks and image processing. Both processes have a layered structure that is amenable to our modular framework. Therefore, we build a VE protocol for the full computation by composing several "gadgets" VEs for each layer (including existing and new VEs that we develop – see below), and then we use a multilinear polynomial commitment to compile it into an argument of knowledge. Following the modularity principle, then we focus on designing efficient VEs for the main subroutines needed by these applications.

In this application context, our main contribution is a new VE scheme for convolution operations which is amenable to multiple input-output channels and also to prediction batching. Convolution is a challenging operation in proof systems, as it is represented by arithmetic circuits with complex wiring (and up to $O(n^3)$ size for convolutions over a $n \times n$ matrix) which is expensive for general purpose solutions. The most efficient dedicated protocol in the literature appears in zkCNN [LXZ21], which proposes a fast proving technique for Fast Fourier Transform (FFT), achieving asymptotically optimal $O(n^2)$ proving time. Nevertheless, their approach requires proving an FFT, a Hadamard product and an inverse FFT, which increase concrete proof size and prover time. Moreover, in their case the convolution kernel, which is often small in applications, needs to be padded to the input size.

We overcome these limitations by designing a compact matrix encoding of the convolution operation to which we apply the efficient matrix multiplication prover in [Tha13]. Crucially, we optimize our technique to efficiently support multiple channels (both input and output), which is when our solutions improve even more over the zkCNN's approach. Notably, our convolution VE achieves proof size and verifier time that are independent from the input size and the number of output channels.

We obtain further improvements by designing VE gadgets that extend techniques originally proposed in the context of the GKR protocol for arithmetic circuits. Notably, we propose a VE for "many-to-one reductions" for input fingerprints that extends the GKR-specific technique of [ZLW$^+$21], and we generalize the blueprint from Hyrax [WTs$^+$18] in order to efficiently batch the executions of the same VE on different inputs, e.g., $Y_i = F(X_i)$ for $i = 1$ to N.

Finally, we leverage our framework to construct the first dedicated proof system for recurrent neural networks.

Implementation and evaluation. We implement and benchmark our efficient convolution prover in Rust and confirm the concrete improvements (in overall efficiency and proof size) over the state-of-the-art [LXZ21] for common sets of parameters. Even for a single-channel convolution, our VE improves over previous solutions by a factor of 5-10× in proof size, and by a similar factor in prover time for small kernel sizes. We additionally benchmark the prover time for the convolutional layers of the VGG11 model.

8.2 Composition Framework for Interactive Proofs

We introduce a framework for building interactive proofs from the composition of function-specific protocols. Our framework consists of three main components: (1) fingerprinting schemes, that are a mechanism with which prover and verifier can succinctly represent inputs and outputs of the computation; (2) verifiable evaluation schemes on fingerprinted data (VE), that are the function-specific protocols in which the verifier works by only knowing fingerprints of inputs and outputs; (3) a composition theorem which shows how to compose VEs, in such a way that the verifier only needs to compute fingerprints for the main input and output of the computation, but not for the intermediate inputs of the sequential steps.

In this section, we define the syntax and the security property of these objects, state and prove the composition and finally also show how to compile a VE scheme into succinct arguments.

Definition 8.1 (Fingerprint)**.** *Let $\mathcal{X}$ be a data space, $\mathcal{D}_{\mathcal{X}}$ a distribution over a randomness space $\mathcal{R}_{\mathcal{X}}$, and C a finite set. A randomized fingerprint (with fingerprint space C) is a function $\mathsf{H} : \mathcal{X} \times \mathcal{R}_{\mathcal{X}} \to C$. Given $x \in \mathcal{X}, r \in \mathcal{R}_{\mathcal{X}}$, we call $c_x \leftarrow \mathsf{H}(x, r)$ the fingerprint of x on r. Furthermore, we say that a fingerprint H is (statistically) sound for $\mathcal{D}_{\mathcal{X}}$ if for any pair $x, x^* \in \mathcal{M}$ such that $x \neq x^*$, we have*

$$\Pr_{r \leftarrow\$ \mathcal{R}_{\mathcal{X}}}[\mathsf{H}(x, r) = \mathsf{H}(x^*, r)] = \mathsf{negl}(\lambda).$$

For vectors of inputs $\boldsymbol{x} \in \prod_{i=1}^{M} \mathcal{M}_i$ and randomness $\boldsymbol{r} \in \prod_{i=1}^{M} \mathcal{R}_{\mathcal{M}_i}$, we use the compact notation $\mathsf{H}(\boldsymbol{x}, \boldsymbol{r}) := (\mathsf{H}(x_1, r_1), \ldots, \mathsf{H}(x_M, r_M))$.

The distribution $\mathcal{D}_{\mathcal{M}}$ is an abstraction that allows us to capture sampling (e.g. via a uniform distribution) from a space which is yet undefined. The randomness space $\mathcal{R}_{\mathcal{M}}$ may depend on the data space $\mathcal{M}$ and on the security parameter λ of the scheme, that will generally be implicit. For instance, large domains may require large randomness spaces[1].

Fingerprints and CRHFs Even though their syntax presents similarities, fingerprints are strictly weaker objects than collision-resistant hash functions (CRHFs). Fingerprints are only guaranteed to be sound if the randomness r is randomly sampled, as opposed to controlled by the adversary. Also, the input x has to be chosen by the adversary before seeing r. The closest notion to our fingerprints are universal hash functions (when instantiated over an exponentially large output space).

8.2.1 Verifiable Evaluation Schemes on Fingerprinted Data

For our framework we consider a class of interactive proofs for the language $\mathcal{L}_{\mathcal{F}} = \{(f, x, y) : f \in \mathcal{F} \wedge f(x) = y\}$, which have the following structure (cf. Figure 8.1):

1. Prover and verifier agree on a common fingerprint $c_y = \mathsf{H}(y, r_y)$. As an example, the verifier samples and sends randomness $r_y \leftarrow\$ \mathcal{D}_{\mathcal{Y}}$ to the prover, and both parties compute c_y independently.
2. Prover and verifier interact on common input (f, c_y, r_y) through subroutines $\mathsf{VE.P}(x)$ and $\mathsf{VE.V}(r_x)$ respectively. Notably, neither x nor y are used by the verifier in this part of the interaction. At the end of a successful interaction, both parties agree on a common fingerprint c_x and randomness r_x.
3. The verifier checks that $c_x = \mathsf{H}(x, r_x)$ and rejects otherwise.

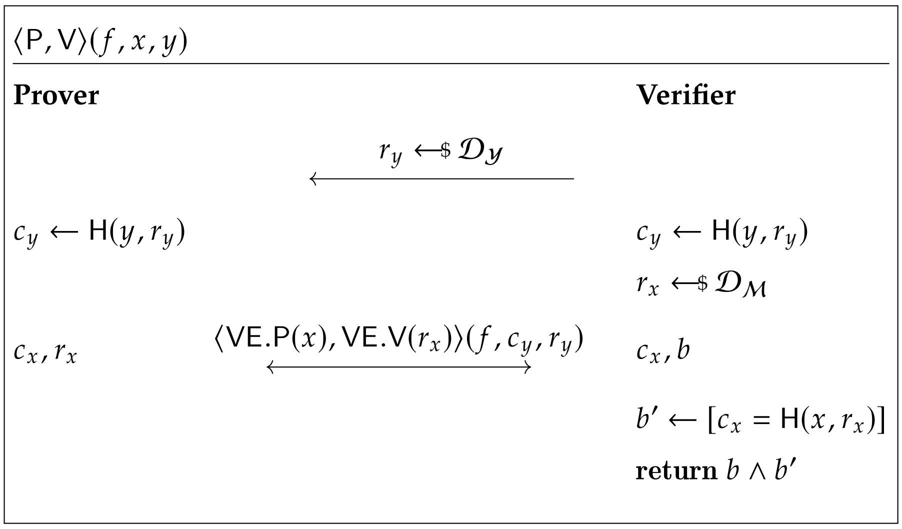

Figure 8.1: *Interactive proof constructed from a verifiable evaluation scheme on fingerprinted data* VE *and a fingerprinting scheme* H.

In other words, these are interactive proofs that manage to reduce the check $f(x) = y$ into a simpler verification that only involves the fingerprints of the output (computed in step (1)) and of the input (computed in step (3)). In this work, we formalize the primitive that takes place in step (2), that we call (interactive) *verifiable evaluation scheme on fingerprinted data* (VE). The goal of a VE scheme is to prove that, given an admissible function f and fingerprints c_x, c_y, then c_x is a valid fingerprint to the input x and c_y is a valid fingerprint to $f(x)$. Contrary to the intuitive setting where the interaction starts with both parties having a common input x (or fingerprint c_x) and finishes on $f(x)$ (or c_y), VE interactions start at a common output fingerprint c_y and finish with both parties agreeing on an input fingerprint c_x.

Definition 8.2. *A* verifiable evaluation scheme on fingerprinted data VE *for a family of functions $\mathcal{F}$ is a pair of interactive algorithms* $(\mathsf{VE.P}, \mathsf{VE.V})$ *that, given as prover input $\boldsymbol{x}$; as verifier input randomness $\mathbf{r}_{\boldsymbol{x}}$; and as common input fingerprints $\mathbf{c}_{\boldsymbol{y}}$, randomness $\mathbf{r}_{\boldsymbol{y}}$, and a function $f \in \mathcal{F}$, the interaction outputs*

$$(\mathbf{c}_{\boldsymbol{x}}; \mathbf{r}_{\boldsymbol{x}}; b) \leftarrow\$ \langle \mathsf{VE.P}(\boldsymbol{x}), \mathsf{VE.V}(\mathbf{r}_{\boldsymbol{x}}) \rangle (\mathbf{c}_{\boldsymbol{y}}, \mathbf{r}_{\boldsymbol{y}}, f)$$

Where $\mathbf{c}_{\boldsymbol{x}}$ is a common output, $\mathbf{r}_{\boldsymbol{x}}$ a prover output, and b a verifier output. Furthermore, the verifier $\mathsf{VE.V}$ *is public-coin.*

The scheme VE *is* ***correct*** *if for any valid pair $(f, \boldsymbol{x})$ and randomness $\mathbf{r}_{\boldsymbol{x}}, \mathbf{r}_{\boldsymbol{y}}$, we have that*

$$\Pr\left[\begin{array}{l} \mathbf{c}_{\boldsymbol{x}} = \mathsf{H}(\boldsymbol{x}, \mathbf{r}_{\boldsymbol{x}}) \\ \wedge\ b \end{array} \middle| \begin{array}{c} \mathbf{c}_{\boldsymbol{y}} \leftarrow \mathsf{H}(f(\boldsymbol{x}), \mathbf{r}_{\boldsymbol{y}}) \\ (\mathbf{c}_{\boldsymbol{x}}; \mathbf{r}_{\boldsymbol{x}}; b) \leftarrow\$ \langle \mathsf{VE.P}(\boldsymbol{x}), \mathsf{VE.V}(\mathbf{r}_{\boldsymbol{x}}) \rangle \\ (\mathbf{c}_{\boldsymbol{y}}, \mathbf{r}_{\boldsymbol{y}}, f) \end{array}\right] = 1$$

Our definition considers families of functions with multiple inputs and outputs, and also with multiple input-output fingerprints. Inputs and outputs may correspond one-to-one with fingerprints, but it is also possible that several fingerprints (computed on

[1] We write $\mathcal{D}_{\mathcal{F}}$ to refer to $\mathcal{D}_{\mathcal{M}}$ when the domain $\mathcal{M}$ is defined by a family of functions $\mathcal{F}$.

different randomness) correspond to a single input or output. For compactness, we write vectors $\mathbf{c}_{\boldsymbol{x}}, \mathbf{r}_{\boldsymbol{x}}$ (respectively $\mathbf{c}_{\boldsymbol{y}}, \mathbf{r}_{\boldsymbol{y}}$) where $\mathsf{c}_{x,i} \in C$ corresponds to $\mathsf{r}_{x,i} \in \mathcal{R}_{\mathcal{M}}$.

The security that is required for VEs is that, if $\mathbf{c}_{\boldsymbol{x}}$ are valid fingerprints of $\boldsymbol{x}$ and the verifier accepts, then $\mathbf{c}_{\boldsymbol{y}}$ are guaranteed to be valid fingerprints of $f(\boldsymbol{x})$ (except with negligible probability). As we will show later, this property is very useful for composing VEs. We remark that security only holds when the fingerprints of the inputs $\mathbf{c}_{\boldsymbol{x}}$ are honest.

Definition 8.3 (VE Soundness)**.** *A VE scheme* VE *is statistically (resp. computationally) sound if for any stateful unbounded (resp. PPT) adversary $\mathcal{A}$, the following probability is* $\mathsf{negl}(\lambda)$*:*

$$\Pr\left[\begin{array}{l|l} & \mathbf{r}_{\boldsymbol{x}}, \mathbf{r}_{\boldsymbol{y}} \leftarrow\!\!\$\ \mathcal{D}_{\mathcal{F}} \\ \mathbf{c}^*_{\boldsymbol{y}} \neq \mathsf{H}(f(\boldsymbol{x}), \mathbf{r}_{\boldsymbol{y}}) & (\mathbf{c}^*_{\boldsymbol{y}}, \boldsymbol{x}, f) \leftarrow \mathcal{A}(\mathbf{r}_{\boldsymbol{y}}) \\ \wedge\, b & (\mathbf{c}^*_{\boldsymbol{x}}; \mathbf{r}_{\boldsymbol{x}}; b) \leftarrow\!\!\$\ \langle \mathcal{A}(\boldsymbol{x}), \mathsf{VE.V}(\mathbf{r}_{\boldsymbol{x}})\rangle \\ & \qquad\qquad (\mathbf{c}^*_{\boldsymbol{y}}, \mathbf{r}_{\boldsymbol{y}}, f) \\ & \mathbf{c}^*_{\boldsymbol{x}} = \mathsf{H}(\boldsymbol{x}, \mathbf{r}_{\boldsymbol{x}}) \end{array}\right]$$

where the probability is taken over the choices of $\mathbf{r}_{\boldsymbol{x}}, \mathbf{r}_{\boldsymbol{y}}$, the randomness of $\mathcal{A}$ and any additional randomness used by VE.V.

Next, we show that VE security is indeed sufficient for building a sound interactive proof as described in Figure 8.1.

Proposition 8.4. *The protocol in Figure 8.1 is an interactive proof.*

Proof. For simplicity, we consider the single-input and single-fingerprint case; the general case follows easily. Completeness follows from the correctness of the VE and the fingerprint. For soundness, let $\mathcal{A}$ be an adversarial prover that makes the verifier accept for $(f, x, y^*) \notin \mathcal{L}_{\mathcal{F}}$ where $f(x) = y \neq y^*$. We want to show that then one can use $\mathcal{A}$ to break either the soundness of the fingerprint or the soundness of the verifiable evaluation scheme.

We consider two events. E_1 is the event that V accepts and $c_y = c^*_y$, where $c_y = \mathsf{H}(y, r_y)$ and $c^*_y = \mathsf{H}(y^*, r_y)$, and E_2 is the event that V accepts and $c_y \neq c^*_y$. E_1 and E_2 are complimentary and clearly

$$\Pr[\mathcal{A} \text{ wins}] = \Pr[E_1] + \Pr[E_2].$$

If E_1 occurs, then we have a collision on the output fingerprint where $r_y \leftarrow\!\!\$\ \mathcal{D}_{\mathcal{Y}}$. By fingerprint soundness, $\Pr[E_1] \leq \Pr[c_y = c^*_y] = \mathsf{negl}(\lambda)$.

If E_2 occurs, it is easy to construct an adversary $\mathcal{B}$ that breaks VE soundness. On input $r_y \leftarrow\!\!\$\ \mathcal{D}_{\mathcal{Y}}$, $\mathcal{B}$ runs the interactive proof in Figure 8.1 using $\mathcal{A}$ as a prover and on randomness r_y. Note that r_y is sampled from the same distribution in the interactive proof and in the VE security game. Then, $\mathcal{B}$ outputs the fingerprint c^*_x output by $\mathcal{A}$ after the interaction. Since the verifier accepts, we have that $b = 1$ and that $c_x = \mathsf{H}(x, r_x)$. Hence,

$$\begin{aligned} \Pr[E_2] &= \Pr[\mathsf{V} \to 1 \ \wedge\ c_y \neq c^*_y] \\ &= \Pr[b = 1 \ \wedge\ \mathsf{H}(f(x), r_y) \neq c^*_y \mid c_x = \mathsf{H}(x, r_x)] \\ &= \Pr[\mathcal{B} \text{ wins VE game}] = \mathsf{negl}(\lambda). \end{aligned}$$

□

8.2.2 Composition of VEs

Next, we show that the composition of VEs that use the same fingerprint scheme is also a VE. This allows for constructions of modular interactive protocols for sequential functions.

Let f be composed of several sub-functions $f_1, \dots, f_n$, that can place left-to-right in a pipeline fashion (see Figure 8.2). The high-level approach of this procedure is the following: 1) start on a fingerprint of the output of f (i.e., on the right) that both prover and verifier trust; 2) run the VE schemes for the f_i in a right-to-left order (starting with f_n); while 3) collecting fingerprints to inputs of the f_i obtained throughout the interaction and using them as output fingerprints for sub-functions on the left. At the end of the interaction, the verifier needs to check one or multiple input fingerprints. In Figure 8.2, we show this procedure in a block diagram.

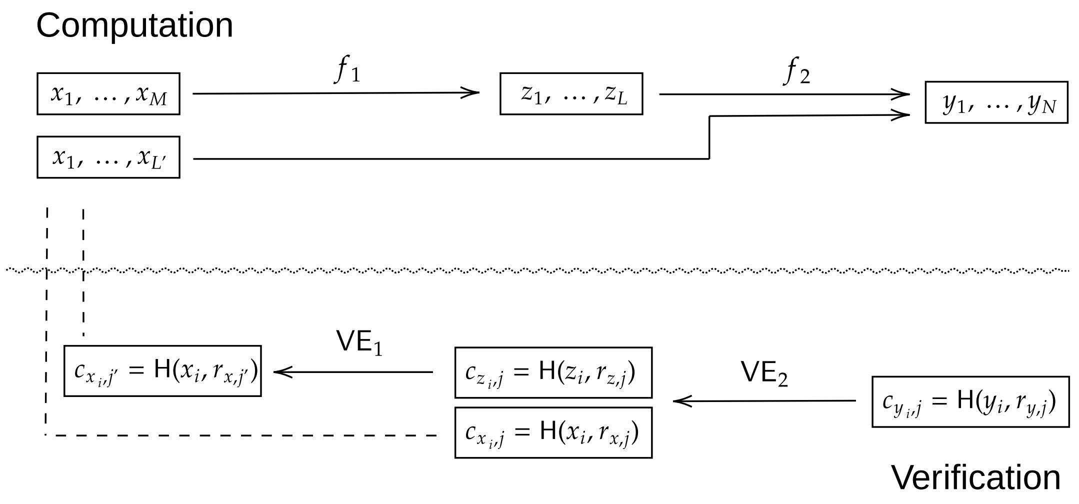
Computation
$x_1, \dots, x_M$
f_1
$z_1, \dots, z_L$
f_2
$y_1, \dots, y_N$
$x_1, \dots, x_{L'}$
$c_{x_i,j'} = \mathsf{H}(x_i, r_{x,j'})$
VE_1
$c_{z_i,j} = \mathsf{H}(z_i, r_{z,j})$
$c_{x_i,j} = \mathsf{H}(x_i, r_{x,j})$
VE_2
$c_{y_i,j} = \mathsf{H}(y_i, r_{y,j})$
Verification

Figure 8.2: *Composition of VEs for functions f_1, f_2 following Proposition 8.5. Top half: computation of $f_2(f_1(\boldsymbol{x}), \bar{\boldsymbol{x}})$, operations are left-to-right. Bottom half: composition of VE_2 and VE_1, interaction is right-to-left.*

Proposition 8.5 (Composition of VEs)**.** *Let $\mathcal{M} = \prod_{i=1}^{M} \mathcal{M}_i$, $\mathcal{Z} = \prod_{i=1}^{L} \mathcal{Z}_i$, $\bar{\mathcal{M}} = \prod_{i=1}^{L'} \bar{\mathcal{M}}_i$ and $\mathcal{Y} = \prod_{i=1}^{N} \mathcal{Y}_i$ be domains. Let also $f_1 : \mathcal{M} \to \mathcal{Z}$ and $f_2 : \mathcal{Z} \times \bar{\mathcal{M}} \to \mathcal{Y}$. Finally, let $f : \mathcal{M} \times \bar{\mathcal{M}} \to \mathcal{Y}$ be the function given by the (partial) composition $f(\boldsymbol{x}, \bar{\boldsymbol{x}}) := f_2(f_1(\boldsymbol{x}), \bar{\boldsymbol{x}})$.*

Then, given verifiable evaluation schemes VE_1 and VE_2 for f_1 and f_2 based on the same fingerprint scheme, the composition protocol VE obtained by running VE_2 and then VE_1 as in Figure 8.2 is a verifiable evaluation scheme for f.

Proof. Let $\mathcal{A}$ be a successful adversary against VE. On input $\boldsymbol{r}_y \leftarrow\$ \mathcal{D}_{\mathcal{F}}$, $\mathcal{A}$ outputs $(\{c^*_{y_i,j}\}_j, (\boldsymbol{x}, \tilde{\boldsymbol{x}}), f)$ such that, for $\boldsymbol{y} = f(\boldsymbol{x}, \tilde{\boldsymbol{x}})$, then $c^*_{y_i,j} \neq \mathsf{H}(y_i, r_{y,j})$ for some j. We use the index j to index the collection of fingerprints independently from the index i of the input they refer to. Then, the verifier $\mathsf{VE.V}$ accepts in the interaction on input $(\{c^*_{y_i,j}\}_j, \boldsymbol{r}_y, f)$.

We will show that in this case $\mathcal{A}$ must break soundness of either VE_1 or VE_2. First, let $\boldsymbol{z} = f_1(\boldsymbol{x})$ (following the notation in Figure 8.2) and $\boldsymbol{y} = f_2(\boldsymbol{z}, \tilde{\boldsymbol{x}})$. Then, we can distinguish between two events based on the behaviour of the adversary. Let E_1 be the event that $c^*_{z_i,j'} \neq \mathsf{H}(z_i, r_{z,j'})$ for some j', and let E_2 be the event that $c^*_{z_i,j} = \mathsf{H}(z_i, r_{z,j})$ for every j. Note that it is possible to determine which event occurs since the protocol is public-coin and so all honest fingerprints can be recomputed in polynomial time.

If E_1 occurs, then $\mathcal{A}$ breaks soundness of VE_1 as the output fingerprint $c^*_{z_i,j'}$ does not match its honest counterpart, while the input fingerprints are honest by assumption. If E_2 occurs, then every $c^*_{z_i,j}$ is honest. As these are the input fingerprints to VE_2, it follows that $\mathcal{A}$ must break soundness of VE_2. □

By combining Proposition 8.5 and Proposition 8.4, we obtain a framework for composing arbitrary evaluation schemes for different functions that can be later compiled into an interactive proof. Regarding efficiency, the communication complexity and running time of the resulting protocol grows additively for both prover and verifier, as VEs are run sequentially.

For clarity, in the following sections we use a parametrization for VE schemes that we define as follows.

Definition 8.6 (Parametrization of VEs). *A verifiable evaluation scheme* VE *is parametrized by:*

- *the fingerprint scheme* H*,*
- *the (family of) admissible functions* $\mathcal{F} = \{f : \mathcal{M} \to \mathcal{Y}\}$*,*
- *the input and output (vectors of) fingerprints* $\mathbf{c}_{\boldsymbol{x}}, \mathbf{c}_{\boldsymbol{y}}$*,*
- *the communication complexity* $|\pi|$ *(of prover messages, i.e., we do not consider verifier challenges)*
- *the prover and verifier running time* $\mathsf{t}_\mathsf{P}, \mathsf{t}_\mathsf{V}$*,*
- *and the soundness* ϵ*.*

8.2.3 From VEs to Arguments of Knowledge

We show how to use a VE to build a commit-and-prove interactive argument of knowledge $\Pi := (\mathsf{Setup}, \mathsf{P}, \mathsf{V})$ for the relation

$$\mathcal{R}_\Pi = \left\{(f, \mathsf{com}_x, \boldsymbol{y}; \boldsymbol{x}, o_x) : f \in \mathcal{F} \wedge f(\boldsymbol{x}) = \boldsymbol{y} \ \wedge \ \mathsf{Com.Ver}(\mathsf{ck}, \mathsf{com}_x, \boldsymbol{x}, o_x)\right\}$$

where o_x is the opening for the committed $\boldsymbol{x}$.

To enable this proof, we need the following building blocks:

- a commitment scheme $\mathsf{Com} := (\mathsf{Setup}, \mathsf{Com}, \mathsf{Open}, \mathsf{Ver})$,
- an argument of knowledge $\mathsf{AoK}_\mathsf{H} := (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Ver})$ for the relation $\mathcal{R}_\mathsf{H} = \{(\mathsf{c}_x, \mathsf{com}_x, r_x; \boldsymbol{x}, o_x) : \mathsf{Com.Ver}(\mathsf{ck}, \mathsf{com}_x, \boldsymbol{x}, o_x) = 1 \wedge \mathsf{c}_x = \mathsf{H}(\boldsymbol{x}, r_x)\}$, where $\mathsf{ck} \leftarrow \mathsf{Com.Setup}(1^\lambda)$ (note that we can instantiate this with a multilinear polynomial commitment).,

- an $\mathsf{AoK}_{\mathsf{Com}}$ for the "proof of knowledge" relation "I know the $\boldsymbol{x}$ committed in com_x" given by $\mathcal{R}_{\mathsf{PoK}} = \{(\mathsf{com}_x; \boldsymbol{x}, o_x) : \mathsf{Com.Ver}(\mathsf{ck}, \mathsf{com}_x, \boldsymbol{x}, o_x)\}$.
- and a VE scheme for a family of functions $\mathcal{F}$.

The idea, described in Figure 8.3, is a generalization of the vSQL approach [ZGK+17] and relies on the observation that in the VE protocol the verifier does not need to know neither $\boldsymbol{x}$ nor $\boldsymbol{y}$ but only their fingerprints $c_{\boldsymbol{x}}, c_{\boldsymbol{y}}$. In the VE-to-IP construction, the verifier would test if $c_{\boldsymbol{x}} = \mathsf{H}(\boldsymbol{x}, r_x)$ and $c_{\boldsymbol{y}} = \mathsf{H}(\boldsymbol{y}, r_y)$. In the AoK, the verifier instead holds a commitment com_x, and we let the prover show the correctness of the fingerprint $c_{\boldsymbol{x}}$ w.r.t. the committed $\boldsymbol{x}$.

Proposition 8.7. $\Pi := (\mathsf{Setup}, \mathsf{P}, \mathsf{V})$ *is an interactive argument of knowledge for the relation* $\mathcal{R}_\Pi$.

Proof. We have to show that for any PPT prover $\mathcal{A}$ there exists an extractor $\mathcal{E}$ that, given access to $\mathcal{A}$'s input and random tape as well as the entire transcript tr of an interaction $\langle \mathcal{A}, \Pi.\mathsf{V}\rangle(\mathsf{crs}, \mathsf{crs}')$ (which includes $\mathcal{A}$'s choice of the statement $(f, \mathsf{com}_x, \boldsymbol{y}^*)$), can extract a witness $w = (\boldsymbol{x}, o_x)$ such that

$$\Pr[\langle \mathcal{A}, \Pi.\mathsf{V}\rangle \to 1 \wedge ((f, \mathsf{com}_x, \boldsymbol{y}^*), (\boldsymbol{x}, o_x)) \notin \mathcal{R}_\Pi] = \mathsf{negl}(\lambda)$$

We proceed as follows.

First, for any adversary $\mathcal{A}$ we can build the following two adversaries $\mathcal{A}_1$ and $\mathcal{A}_2$. $\mathcal{A}_1$ is an adversary against $\mathsf{AoK}_{\mathsf{Com}}$ that, on input crs' and auxiliary input consisting of crs and a random tape $\rho_{\mathcal{A}}$, simply runs $\mathcal{A}(\mathsf{crs}, \mathsf{crs}'; \rho_{\mathcal{A}})$ until it outputs the first message π_1, and then returns π_1. $\mathcal{A}_2$ is an adversary against $\mathsf{AoK}_{\mathsf{H}}$ that, on input crs and auxiliary input consisting of crs', random tape $\rho_{\mathcal{A}}$, and a transcript tr_{V} of random public coins for an execution of VE.V, runs $\mathcal{A}(\mathsf{crs}, \mathsf{crs}'; \rho_{\mathcal{A}})$ until the end so as to obtain π_2, and then it outputs π_2.

Second, by applying the knowledge-soundness of $\mathsf{AoK}_{\mathsf{Com}}$ and $\mathsf{AoK}_{\mathsf{H}}$ we obtain that there exist corresponding extractors $\mathcal{E}_1, \mathcal{E}_2$ that return $(\boldsymbol{x}', o_x')$ and $(\boldsymbol{x}'', o_x'')$ respectively. Therefore, we construct the extractor $\mathcal{E}$ as the algorithm that, on input $(\mathsf{crs}, \mathsf{crs}', \rho_{\mathcal{A}}, tr_{\mathsf{V}})$, runs $(\boldsymbol{x}', o_x') \leftarrow \mathcal{E}_1(\mathsf{crs}'; (\mathsf{crs}, \rho_{\mathcal{A}}))$, and $(\boldsymbol{x}'', o_x'') \leftarrow \mathcal{E}_2(\mathsf{crs}; (\mathsf{crs}', \rho_{\mathcal{A}}, tr_{\mathsf{V}}))$, and returns $(\boldsymbol{x}', o_x')$.

Third, we argue that the probability that the verifier accepts and the witness returned by $\mathcal{E}$ is wrong is negligible. To this end, let us define the following events:

- For $i = 1, 2$, let bad_i be the event that $\mathcal{E}_i$ outputs an invalid witness for π_i and corresponding commitment com_x.
- Let coll be the event that both $(\boldsymbol{x}', o_x')$ and $(\boldsymbol{x}'', o_x'')$ are valid openings of com_x, but $\boldsymbol{x}' \neq \boldsymbol{x}''$.
- Let bad_y be the event that $\boldsymbol{y}^* \neq f(\boldsymbol{x}')$.

Let us define some shorthands for relevant events in the execution of $\mathcal{E}$. Let $\mathsf{Ev} := \mathsf{acc} \wedge \mathsf{bad}_{\mathcal{E}}$ where acc denotes the event that the AoK verifier accepts, "$\langle \mathcal{A}, \Pi.\mathsf{V}\rangle \to 1$", $\mathsf{bad}_{\mathcal{E}}$ is the

```
$\underline{\Pi.\mathsf{Setup}(1^\lambda, \mathsf{ck})\text{:}}$

  • $\mathsf{crs} \leftarrow \mathsf{AoK}_\mathsf{H}.\mathsf{Setup}(1^\lambda, (\mathsf{ck}, \mathcal{R}_\mathsf{H}))$
  • $\mathsf{crs}' \leftarrow \mathsf{AoK}_\mathsf{Com}.\mathsf{Setup}(1^\lambda, (\mathsf{ck}, \mathcal{R}_\mathsf{Com}))$
  • $\textbf{return } (\mathsf{crs}, \mathsf{crs}')$.

$\underline{\Pi.\mathsf{P}((\mathsf{crs}, \mathsf{crs}'), (f, \mathsf{com}_x, \boldsymbol{y}; \boldsymbol{x}, o_x))\text{:}}$

  • $\pi_1 \leftarrow \mathsf{AoK}_\mathsf{Com}.\mathsf{Prove}(\mathsf{crs}', (\mathsf{com}_x; \boldsymbol{x}, o_x))$
  • $\underline{\text{Send}}$ $\pi_1$ to V
  • $\underline{\text{Get}}$ $r_y \leftarrow\$ \mathcal{D}_\mathcal{F}$ from V
  • $c_{\boldsymbol{y}} \leftarrow \mathsf{H}(\boldsymbol{y}, r_y)$
  • $\underline{\text{Run}}$ $(c_{\boldsymbol{x}}, r_x) \leftarrow \mathsf{VE}.\mathsf{P}(\boldsymbol{x}, f, c_{\boldsymbol{y}}, r_y)$ interactively with V.
  • $\pi_2 \leftarrow \mathsf{AoK}_\mathsf{H}.\mathsf{Prove}(\mathsf{crs}, (c_{\boldsymbol{x}}, \mathsf{com}_x, r_x; \boldsymbol{x}, o_x))$
  • $\underline{\text{Send}}$ $\pi_2$ to V

$\underline{\Pi.\mathsf{V}((\mathsf{crs}, \mathsf{crs}'), (f, \mathsf{com}_x, \boldsymbol{y}))\text{:}}$

  • $\underline{\text{Get}}$ $\pi_1$ from P
  • $\underline{\text{Send}}$ $r_y \leftarrow\$ \mathcal{D}_\mathcal{F}$ to P and compute $c_{\boldsymbol{y}} \leftarrow \mathsf{H}(\boldsymbol{y}, r_y)$
  • $r_x \leftarrow\$ \mathcal{D}_\mathcal{X}$
  • $\underline{\text{Run}}$ $b_0 \leftarrow \mathsf{VE}.\mathsf{V}(r_x, f, c_{\boldsymbol{y}}, r_y)$ interactively with P.
  • $\underline{\text{Get}}$ $\pi_2$ from P
  • $b_1 \leftarrow \mathsf{AoK}_\mathsf{Com}.\mathsf{Ver}(\mathsf{crs}', \mathsf{com}_x, \pi_1)$
  • $b_2 \leftarrow \mathsf{AoK}_\mathsf{H}.\mathsf{Ver}(\mathsf{crs}, (c_{\boldsymbol{x}}, \mathsf{com}_x, r_x), \pi_2)$
  • $\textbf{return } b_0 \wedge b_1 \wedge b_2$
```

Figure 8.3: *Construction of an interactive argument of knowledge* Π *for the relation* $\mathcal{R}_\Pi$ *from a commitment scheme* Com *such that* $\mathsf{ck} \leftarrow \mathsf{Com}.\mathsf{Setup}(1^\lambda)$*, arguments of knowledge* $\mathsf{AoK}_\mathsf{Com}$ *for* $\mathcal{R}_\mathsf{PoK}$ *and* AoK_H *for* $\mathcal{R}_\mathsf{H}$*, and a* VE *scheme for* $\mathcal{F}$*.*

event "$((f, \mathsf{com}_x, \boldsymbol{y}^*), (\boldsymbol{x}', o_x')) \notin \mathcal{R}_\Pi$", and $\overline{\mathsf{bad}} := \overline{\mathsf{bad}_1} \wedge \overline{\mathsf{bad}_2}$. Note that $\mathsf{bad}_\mathcal{E} := (f(\boldsymbol{x}') \neq \boldsymbol{y}^*) \vee \mathsf{Com.Ver}(\mathsf{ck}, \mathsf{com}_x, \boldsymbol{x}', o_x') = 0$.

Then it holds

$$\begin{aligned}\Pr[\mathsf{Ev}] &\leq \sum_{i=1}^{2} \Pr[\mathsf{Ev} \wedge \mathsf{bad}_i] + \Pr[\mathsf{Ev} \wedge \overline{\mathsf{bad}}] \\ &\leq \mathsf{negl}(\lambda) + \Pr[\mathsf{Ev} \wedge \overline{\mathsf{bad}} \wedge \mathsf{coll}] + \Pr[\mathsf{Ev} \wedge \overline{\mathsf{bad}} \wedge \overline{\mathsf{coll}}] \\ &\leq \mathsf{negl}(\lambda) + \Pr[\mathsf{Ev} \wedge \overline{\mathsf{bad}} \wedge \overline{\mathsf{coll}}].\end{aligned}$$

where the first inequality follows by applying a union bound, the second one by the knowledge soundness of $\mathsf{AoK}_{\mathsf{Com}}$ and $\mathsf{AoK}_{\mathsf{H}}$, and the third one follows by the computational binding of the commitment scheme. Next, we show that $\Pr[\mathsf{Ev} \wedge \overline{\mathsf{bad}} \wedge \overline{\mathsf{coll}}]$ is negligible under the statistical soundness of the fingerprinting scheme and of the VE scheme. To this end, we partition over the event $c_{\boldsymbol{y}} = \mathsf{H}(\boldsymbol{y}, r_y) = \mathsf{H}(f(\boldsymbol{x}'), r_y)$, i.e.,

$$\begin{aligned}\Pr[\mathsf{Ev} \wedge \overline{\mathsf{bad}} \wedge \overline{\mathsf{coll}}] &\leq \Pr[\mathsf{H}(\boldsymbol{y}, r_y) = \mathsf{H}(f(\boldsymbol{x}'), r_y)] \\ &\quad + \Pr[\mathsf{Ev} \wedge \overline{\mathsf{bad}} \wedge \overline{\mathsf{coll}} \wedge c_{\boldsymbol{y}} \neq \mathsf{H}(f(\boldsymbol{x}'), r_y)].\end{aligned}$$

As $\boldsymbol{x}'$ is extracted before the random choice of r_y we obtain that $\Pr[\mathsf{H}(\boldsymbol{y}, r_y) = \mathsf{H}(f(\boldsymbol{x}'), r_y)] = \mathsf{negl}(\lambda)$ by the soundness of the fingerprinting scheme.

Since acc includes the event $b_0 = 1$ and by simplifying the events in '$\mathsf{Ev} \wedge \overline{\mathsf{bad}} \wedge \overline{\mathsf{coll}}$', we have

$$\begin{aligned}&\Pr[\mathsf{Ev} \wedge \overline{\mathsf{bad}} \wedge \overline{\mathsf{coll}} \wedge c_{\boldsymbol{y}} \neq \mathsf{H}(f(\boldsymbol{x}'), r_y)] \\ &\leq \Pr\left[b_0 \wedge c_{\boldsymbol{y}} \neq \mathsf{H}(f(\boldsymbol{x}'), r_y) \mid c_{\boldsymbol{x}} = \mathsf{H}(\boldsymbol{x}', r_x) \wedge f(\boldsymbol{x}') \neq \boldsymbol{y}'\right] = \epsilon.\end{aligned}$$

Consider an adversary $\mathcal{A}$ and its corresponding $\mathcal{E}$ shown above such that the above ϵ is non-negligible. Then we can build a VE adversary $\mathcal{B}$ that breaks soundness with probability ϵ as follows. Recall that $\mathcal{B}(r_y)$ breaks VE soundness if, before the interaction, it outputs a fingerprint $c_{\boldsymbol{y}}$ such that $c_{\boldsymbol{y}} \neq \mathsf{H}(f(\boldsymbol{x}'), r_y)$.

1. $\mathcal{B}(r_y)$, on input a random challenge r_y, honestly generates $\mathsf{crs}, \mathsf{crs}'$ and a suitable random tape $\rho_\mathcal{A}$, runs $\mathcal{A}(\mathsf{crs}, \mathsf{crs}'; \rho_\mathcal{A})$ until "send π_1 to V" to obtain $(f, \mathsf{com}_x, \boldsymbol{y})$ and the proof π_1.
2. Runs the extractors $(\boldsymbol{x}', o_x') \leftarrow \mathcal{E}_1(\mathsf{crs}'; (\mathsf{crs}, \rho_\mathcal{A}))$.
3. Compute $c_{\boldsymbol{y}} = \mathsf{H}(\boldsymbol{y}, r_y)$ and **outputs** $(c_{\boldsymbol{y}}, \boldsymbol{x}', f)$.
4. $\mathcal{B}$ sends r_y to $\mathcal{A}$ and then interacts with $\mathsf{VE.V}$ in its soundness game on common input $(c_{\boldsymbol{y}}, r_y, f)$, by forwarding all the messages from $\mathsf{VE.V}$ to $\mathcal{A}$. Namely, $\mathcal{B}$ runs $(c_{\boldsymbol{x}}, r_x; b_0) \leftarrow \langle \mathcal{A}, \mathsf{VE.V} \rangle$ and then executes the last step of $\mathcal{A}$ to obtain π_2. Let tr_V be the transcript of V's coins in this interaction (including r_y and VE.V's coins).
5. Runs the extractor $(\boldsymbol{x}'', o_x'') \leftarrow \mathcal{E}_2(\mathsf{crs}; (\mathsf{crs}', \rho_\mathcal{A}, tr_\mathsf{V}))$.
6. Aborts if either any of the events $\{\mathsf{bad}_1, \mathsf{bad}_2, \mathsf{coll}\}$ occurs. Otherwise, **returns** $(c_{\boldsymbol{x}}, r_x)$

As one can see, if $\mathcal{A}$ and $\mathcal{E}$ are such that the above event occurs with probability ϵ, then $\mathcal{B}$ wins in the VE soundness experiment with the same probability ϵ.

□

The protocol Π described above is a public-coin interactive protocol that can be easily compiled into a non-interactive argument. As usual in the literature, security is argued in the random oracle model, via the Fiat-Shamir heuristic [FS87].

Finally, we observe that, similarly to zkCNN, we can obtain a zero-knowledge AoK for $\mathcal{R}_\Pi$ by using existing approaches [CFS17, XZZ+19] based on zero-knowledge sumcheck and low-degree extensions. More precisely, starting from the (non-ZK) VE scheme, we first apply the information-theoretic compiler based on zero-knowledge sumcheck from Libra ([XZZ+19], Section 4.1). Then, we require a ZK-AoK for H in the compilation to a succinct argument. For the first step, we also need to mask the fingerprints obtained by the verifier to avoid leakage of intermediate values. This can also be done following ([XZZ+19], Section 4.2).

8.3 Verifiable Evaluation from Multilinear Polynomials

In this section, we reinterpret the line of work for the delegation of computation via sumchecks of multilinear polynomials, initiated by the GKR protocol [GKR08] and continued by [CMT12b, Tha13, XZZ+19, ZLW+21], in the framework introduced in Section 8.2. We show that the notion of verifiable evaluation scheme captures the soundness properties of these core protocols, and we provide a modular approach such that they are easily composable with function-specific VEs. This allows us to compose these existing protocols with the new VE schemes that we propose in the next section.

First of all, we define a fingerprint based on multilinear extensions. From this point, we adopt the convention that $\lambda = \lfloor \log|\mathbb{F}| \rfloor$ for a field $\mathbb{F}$.

Proposition 8.8. *Let $\mathbb{F}$ be a field, $\tilde{\boldsymbol{x}}$ be the multilinear extension of $\boldsymbol{x} \in \mathbb{F}^n$, and $\ell = \lceil \log n \rceil$. Then, the evaluation of a multilinear extension at a point $\boldsymbol{r} \in \mathbb{F}^\ell$, given by $\tilde{\boldsymbol{x}}(\boldsymbol{r}) \leftarrow \mathsf{H}_{\mathsf{MLE}}(\boldsymbol{x}, \boldsymbol{r})$, is a statistically sound fingerprint for the uniform distribution over $\mathbb{F}^\ell$.*

Proof. Given two inputs $\boldsymbol{x}, \boldsymbol{x}^*$ and $\boldsymbol{r} \leftarrow\!\$\ \mathbb{F}^d$ such that $\boldsymbol{x} \neq \boldsymbol{x}^*$, we have that

$$\Pr[\tilde{\boldsymbol{x}}(\boldsymbol{r}) = \tilde{\boldsymbol{x}}^*(\boldsymbol{r})] = \Pr[(\tilde{\boldsymbol{x}} - \tilde{\boldsymbol{x}}^*)(\boldsymbol{r}) = 0] \leq d/|\mathbb{F}|.$$

where the bound follows by the Schwartz-Zippel lemma (Theorem 3.16). □

Multinear sumcheck VE. The following result is a generalization of the multilinear sumcheck-based delegation schemes in the literature, particularly of those introduced in [Tha13, XZZ+19]. The prover time depends on the time required to compute the multilinear extension of each polynomial factor $f_{k,i}$ as described below. Note that when the multilinear sumcheck is described in the VE framework, the function f corresponds to the sum of the evaluations over $\{0, 1\}^\ell$, while the polynomial factors $f_{k,i} \in \mathbb{F}[x_1, \ldots, x_\ell]$

correspond to the input and are not necessarily known to the verifier. In most practical cases, $s = 1$ and t is a small constant (such as $t = 2$).

Proposition 8.9. *Let $\boldsymbol{x}$ be a vector of ℓ variables, $\mathbb{F}$ a finite field and $\alpha_i \in \mathbb{F}$ for $i = 1, \dots, s$. Let also*

$$f(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{s} \alpha_i \prod_{k=1}^{t} f_{k,i}(\boldsymbol{x}_{k,i}, \boldsymbol{y})$$

where each factor $f_{k,i}$ is a multilinear polynomial over $\mathbb{F}$ evaluated on a subvector $\boldsymbol{x}_{k,i} \subset \boldsymbol{x}$. Then, the multilinear sumcheck protocol $\mathsf{VE}_{\mathsf{ML}}$ in Figure 8.4 is a MLE-based VE scheme for the relation

$$f_y(\boldsymbol{r}_y) = \sum_{\boldsymbol{x} \in \{0,1\}^{\ell}} f(\boldsymbol{x}, \boldsymbol{r}_y).$$

$\mathsf{VE}_{\mathsf{SC}}$ is parametrized by one output fingerprint $\mathsf{com}_{f_y} = f_y(\boldsymbol{r}_y)$, $s \cdot t$ input fingerprints $\mathsf{com}_{k,i} = f_{k,i}(\boldsymbol{r}_{k,i}, \boldsymbol{r}_y)$ where each $\boldsymbol{r}_{k,i} \subset \boldsymbol{r} \in \mathbb{F}^{\ell}$, communication complexity $|\pi| = (\ell + s) \cdot t \cdot \lambda$, verification time $\mathsf{t_V} = O(t \cdot \ell)$, and soundness $\epsilon = t\ell/|\mathbb{F}|$. Furthermore, given that $\tau_{k,i}$ is the time required to compute the MLE of $f_{k,i}(\boldsymbol{x}_{k,i}, \cdot)$, the prover time is $\mathsf{t_P} = O\big(s \cdot t^2 \cdot \max_{k,i} \tau_{k,i}\big)$.

Proof. First, we recall that the sumcheck protocol over a field $\mathbb{F}$ for a ℓ-variate polynomial of degree t has soundness $t\ell/|\mathbb{F}|$ [LFKN92].

Correctness, communication complexity and efficiency follow from inspection of Figure 8.4 and from the efficient sumcheck and padding techniques in previous work [XZZ+19, ZLW+21]. For soundness, consider a successful adversary against VE soundness that, given an output fingerprint $\mathsf{com}^*_{f_y} \neq f_y(\boldsymbol{r}_y)$, makes $\mathsf{VE}_{\mathsf{SC}}.\mathsf{V}$ accept. Let also $g'_1(x_1), \dots, g'_\ell(x_\ell)$ be the sequence of degree t polynomials that correspond to running the protocol honestly, in addition to the constant polynomial $g'_0 = f_y(\boldsymbol{r}_y)$. By definition of VE soundness, we have that all input fingerprints are honestly computed, i.e., $\mathsf{com}_{k,i} = f_{k,i}(\boldsymbol{r})$ for every k, i. Therefore, as the final check of Figure 8.4 verifies, it must be that $\hat{g}_\ell(r_\ell) = g'_\ell(r_\ell)$. We conclude that the adversary must have found a collision during the sumcheck, which occurs with probability $\epsilon = t\ell/|\mathbb{F}|$. □

8.3.1 VE for GKR layers

In the celebrated GKR protocol [GKR08], prover and verifier interact in a series of sumchecks that take place at every layer of the circuit. Each of these sumchecks can be written as a VE scheme with multiple input fingerprints (and possibly multiple output fingerprints too). This interpretation is straightforward following Proposition 8.9; it also addresses the observation that the add and mult gate predicates can be replaced by alternative gate predicates, in order to support other operations efficiently as mentioned in [XZZ+19], or larger fan-in such as in [LXZ21].

Following the notation from Libra [XZZ+19], we write V_i for the output values at the gates of the circuit at layer i (interpreted as a function $V_i : \{0,1\}^{\ell_i} \to \mathbb{F}$) and $\tilde{V}_i$ its multilinear extension. We define the wiring predicates $\mathsf{add}_i, \mathsf{mult}_i : \{0,1\}^{\ell_i + 2\ell_{i-1}} \to \mathbb{F}$, which take one gate label $y \in \{0,1\}^{\ell_i}$ and two gate labels $x_1, x_2 \in \{0,1\}^{\ell_{i-1}}$, and output 1 if

```
VE_SC.P(com_{f_y}, r_y, f)                                   VE_SC.V(com_{f_y}, r_y, r)

• Evaluate $f_{k,i}(\boldsymbol{x}_{k,i}, \boldsymbol{r}_y)$ for all $k, i$
•                                                            $\hat{g}_0 \leftarrow c_{f_y}$
• for $j = 1 \ldots \ell$ :
•     for $d = 0 \ldots t$ :
•         $m_{j,d} \leftarrow \sum_{\boldsymbol{b} \in \{0,1\}^{\ell-j}} \sum_{i=1}^{s} \alpha_i \prod_{k=1}^{t} f_{k,i}(r_1, \ldots, r_{j-1}, d, \boldsymbol{b}, \boldsymbol{r}_y)$
•     Send $\boldsymbol{m}_j = (m_{j,0}, \ldots, m_{j,t}) \in \mathbb{F}^{t+1}$
•                                        Interpolate $\hat{g}_{j-1}$ from $\boldsymbol{m}_{j-1}$
•                                        Check $[\hat{g}_{j-1}(r_{j-1}) = m_{j,0} + m_{j,1}]$
•                                                            Send $r_j$

                              Final round:

• Send $\mathsf{com}_{k,i} = f_{k,i}(\boldsymbol{r}_{k,i}, \boldsymbol{r}_y)$, for all $k, i$
•                                        Interpolate $\hat{g}_\ell$ from $\boldsymbol{m}_\ell$
•                                        Check $\left[\hat{g}_\ell(r_\ell) = \sum_{i=1}^{s} \alpha_i \prod_{k=1}^{t} \mathsf{com}_{k,i}\right]$
•                                        Set $b \leftarrow 1$ if all checks pass

  Output $(\{c_{k,i}\}_{k,i}, \boldsymbol{r})$                        Output $(\{c_{k,i}\}_{k,i}, b)$
```

Figure 8.4: *Multilinear sumcheck protocol* $\mathsf{VE}_{\mathsf{SC}}$.

gate y is an addition (respectively a multiplication) gate that takes the outputs from gates x_1, x_2 in the previous layer. Therefore, for any $y \in \{0,1\}^{\ell_i}$, we can write V_{i+1} as

$$V_{i+1}(y) = \sum_{x_1, x_2 \in \{0,1\}^{\ell_i}} \mathsf{add}_i(y, x_1, x_2)(V_i(x_1) + V_i(x_2)) \\ + \mathsf{mult}_i(y, x_1, x_2)(V_i(x_1) \cdot V_i(x_2)). \tag{8.1}$$

In the protocol, prover and verifier start on a common fingerprint of the output $V_{i+1}(\boldsymbol{r}_y)$ and then run the multilinear sumcheck from Figure 8.4. At the end of the sumcheck, in which the prover sends a total of $2 \cdot \ell_i$ polynomials, the verifier needs to check the consistency of the prover's claims by using the wiring predicates. Namely, it needs to compute (or re-use in a layer above) the following fingerprints: $\tilde{V}_i(\boldsymbol{r}_1)$, $\tilde{V}_i(\boldsymbol{r}_2)$, $\tilde{\mathsf{add}}_i(\boldsymbol{r}_y, \boldsymbol{r}_1, \boldsymbol{r}_2)$, $\tilde{\mathsf{mult}}_i(\boldsymbol{r}_y, \boldsymbol{r}_1, \boldsymbol{r}_2)$.

The following result is a reinterpretation of [XZZ+19], and in particular the observation that the prover time is linear in 2^ℓ where $\ell = \max\{\ell_i, \ell_{i+1}\}$ due to the sparsity of $\tilde{\mathsf{add}}_i$, $\tilde{\mathsf{mult}}_i$ and Lemma 3.17. The proof follows from Proposition 8.9.

Proposition 8.10. *The interactive protocol that takes place at a GKR layer is a VE scheme* $\mathsf{VE}_{\mathsf{GKR}}$ *for all functions computable by a single-layered arithmetic circuit with gates of fan-in 2. The scheme is parametrized by 1 output fingerprint (of* V_{i+1}*), 4 input fingerprints (2 of* V_i*, 1 of* add_i*, 1 of* mult_i*), communication complexity* $|\pi| = (3 \cdot \ell + 4) \cdot \lambda$*, prover time* $\mathsf{t_P} = O(2^\ell)$*, verifier time* $\mathsf{t_V} = O(\ell)$*, and soundness* $\epsilon = 2\ell/|\mathbb{F}|$*.*

8.3.2 VE for Many-to-One Reductions

Multivariate sumcheck-based VEs often present the issue that, from a single output fingerprint, the interaction yields multiple input fingerprints to be checked by the verifier at a later time. For GKR layers, the two input fingerprints of V_i obtained shall be used as output fingerprint for layer $i-1$. To avoid an exponential blow-up on the number of fingerprints to be checked, the original GKR protocol proposes a 2-to-1 reduction protocol that, given two fingerprints of any $\boldsymbol{x}$, it reduces them to a single fingerprint. An alternative to the 2-to-1 reduction is to use a random linear combination on the sum [CFS17].

Below we formalize 2-to-1 reductions in the VE framework and generalize it to a m-to-1 reduction. The result extends GKR-specific techniques from Virgo++ [ZLW+21].

Proposition 8.11. *Let* $\boldsymbol{x} \in \mathbb{F}^n$ *and let* $\tilde{\boldsymbol{x}}(\boldsymbol{r}_1), \ldots, \tilde{\boldsymbol{x}}(\boldsymbol{r}_m)$ *be MLE fingerprints on* $\boldsymbol{r}_i \in \mathbb{F}^\ell$*. Let also* $\alpha_i \in \mathbb{F}$ *for* $i = 1, \ldots, m$*, let* $I(\boldsymbol{u}, \boldsymbol{v})$ *be the indicator function on the boolean hypercube such that* $I(\boldsymbol{u}, \boldsymbol{v}) = 1$ *if* $\boldsymbol{u} = \boldsymbol{v}$ *and is zero elsewhere, and define*

$$f(\boldsymbol{y}) = \sum_{i=1}^{m} \alpha_i \cdot \boldsymbol{x}(\boldsymbol{r}_i) = \left(\sum_{i=1}^{m} \alpha_i \cdot \tilde{I}(\boldsymbol{r}_i, \boldsymbol{y}) \right) \cdot \tilde{\boldsymbol{x}}(\boldsymbol{y}).$$

Then, running the multilinear sumcheck protocol from Figure 8.4 on $f(\boldsymbol{y})$ *yields a VE scheme* $\mathsf{VE_{m\text{-}1}}$ *parametrized by* m *output fingerprints* $\tilde{\boldsymbol{x}}(\boldsymbol{r}_i)$*,* $m+1$ *input fingerprints (*$I(\boldsymbol{r}_i, \boldsymbol{r}_y)$ *for* $i = 1, \ldots, m$ *and* $\tilde{\boldsymbol{x}}(\boldsymbol{r}_y)$*), communication complexity* $|\pi| = (3 \cdot \ell + m + 1) \cdot \lambda$*, prover time* $\mathsf{t_P} = O(m \cdot 2^\ell)$*, verifier time* $\mathsf{t_V} = O(m + \ell)$*, and soundness* $\epsilon = (2\ell + 1)/|\mathbb{F}|$*.*

Note the additional soundness loss of $1/|\mathbb{F}|$ with respect to the sumcheck, which comes from the choice of the α_i. It is straightforward to express the random linear combination approach from [CFS17] as a VE, also following Proposition 8.9. Such VE is parametrized by 2 input fingerprints (of V_{i+1}), and 6 output fingerprints (2 of V_i, 2 of add_i, 2 of mult_i).

Evaluation of mult**,** add **and structured predicates.** In all VEs introduced so far, including those in Proposition 8.10 and 8.11, the number of input fingerprints is larger than the number of output fingerprints. Some of these fingerprints correspond to unstructured data (such as the values at a circuit layer or an external input), but most of them have a regular structure such as wiring predicates mult, add and indicator functions.

When multiple VEs are composed, fingerprints coming from structured data may be checked directly by the verifier, as opposed to plugged into other VEs. There exist essentially two design choices available:

- The verifier recomputes the multilinear extensions on its own. In many cases, one can benefit from parallelism [CMT12a], or from sparsity [XZZ+19]. In [HR18], it is shown that most *simple* predicates (those expressible as read-only branching programs), including many regular wiring patterns such as indicator functions, can be evaluated in logarithmic time (i.e. polynomial in ℓ).
- The verifier performs a pre-processing phase or relies on a trusted third party to compute (multilinear) polynomial commitments to the data. Then, the prover provides an opening proof on the required point. In this setting, the evaluation is outsourced to the prover, similarly to what is done for instance in Spartan [Set20].

8.3.3 Efficient Matrix Multiplication

Among the protocols that we can capture in our framework, a notable example is the efficient interactive protocol for matrix multiplication from [Tha13]. The main idea of the protocol is to express the product of two matrices $C = A \cdot B$ where $A, B, C \in \mathbb{F}^{n \times n}$ as a polynomial identity as

$$C(\boldsymbol{x}_1, \boldsymbol{x}_2) = \sum_{\boldsymbol{y} \in \{0,1\}^\ell} A(\boldsymbol{x}_1, \boldsymbol{y}) \cdot B(\boldsymbol{y}, \boldsymbol{x}_2) \tag{8.2}$$

Then, the interaction follows the sumcheck in Figure 8.4. Namely, given $\boldsymbol{r}_1, \boldsymbol{r}_2 \in \mathbb{F}^\ell$, both parties carry out a sumcheck over

$$\tilde{C}(\boldsymbol{r}_1, \boldsymbol{r}_2) = \sum_{\boldsymbol{y} \in \{0,1\}^\ell} \tilde{A}(\boldsymbol{r}_1, \boldsymbol{y}) \cdot \tilde{B}(\boldsymbol{y}, \boldsymbol{r}_2). \tag{8.3}$$

The protocol is therefore a VE scheme parametrized by two input fingerprints $\tilde{A}(\boldsymbol{r}_1, \boldsymbol{r}_3)$, $\tilde{B}(\boldsymbol{r}_3, \boldsymbol{r}_2)$, an output fingerprint $\tilde{C}(\boldsymbol{r}_1, \boldsymbol{r}_2)$, communication complexity $|\pi| = (3 \cdot \ell + 2) \cdot \lambda$, prover time $\mathsf{t_P} = O(n^2)$, verifier time $\mathsf{t_V} = O(\ell)$ and soundness $\epsilon = 2\ell/|\mathbb{F}|$.

8.4 Verifiable Evaluation for Machine Learning

In this section, we introduce efficient proofs for common ML operations, following our VE framework. We focus on Convolutional Neural Networks (CNNs) though we note that many of these operations are also usual in image processing. We start by introducing ML preliminaries.

8.4.1 Neural Network Preliminaries

CNNs. A Convolutional Neural Network (CNN) is a layered model where the initial input X is transformed sequentially from layer to layer. Let $X = X^{(1)}$ be the array of input values and $\{X^{(k)}\}_{k=1}^{L}$ the intermediate values between layers, as defined before. Each $X^{(k)} \in \mathbb{F}^{c^{(k)} \times n^{(k)} \times n^{(k)}}$, where $c^{(k)}$ is the number of channels at layer k, and $n^{(k)} \times n^{(k)}$ is the dimension of the arrays at layer k. Namely, at each intermediate layer we have $c^{(k)}$ "parallel" arrays of the same size. An example of multiple channels in an input layer is a coloured image, which commonly has 3 channels: the red, blue, and green values of each pixel.

CNNs apply layer functions $f^{(k)}$ sequentially, such that $X^{(k+1)} = f^{(k)}(X^{(k)}, W^{(k)})$. Usually, models interleave linear layers, such as convolutional layers and fully connected layers, and nonlinear layers such as ReLU and Pooling. At some of these layers, including convolutional layers, we have parameters $W^{(k)}$ (aka weights). For convolutional layers, these are $c^{(k)} \times c^{(k+1)}$ matrices of size $m^{(k)} \times m^{(k)}$. We denote each of these matrices as $W^{(k)}_{\sigma,\tau}$ where $\sigma \in [c^{(k)}]$ and $\tau \in [c^{(k+1)}]$.

Convolution. The equation of a plain 2D convolution[2] in a CNN for a given output channel τ is

$$X^{(k+1)}_{\tau}[u, v] = \sum_{\sigma=1}^{c^{(k)}} \sum_{i,j=1}^{m^{(k)}} X^{(k)}_{\sigma}[u + i, v + j] \cdot W^{(k)}_{\sigma,\tau}[i, j]. \tag{8.4}$$

If no padding and strides (i.e. "jumps" in the convolution) are applied, the output matrix $X^{(k+1)}_{\tau}$ is a square matrix of size $n^{(k+1)} \times n^{(k+1)}$ where $n^{(k+1)} = n^{(k)} - m^{(k)} + 1$. It is very common in practice to apply a zero or mirror padding such that $n^{(k)} = n^{(k+1)}$. Convolutions can be carried out via (naive) dot products, via Fast Fourier Transforms (FFTs), via polynomial multiplication, or via matrix multiplication[3].

A related common operation is *transposed convolution*, which is an upsampling operation that increases the size of the output with respect to the input. We refer to [DV16] for a good introduction to convolution arithmetic.

Quantisation. Generally, CNNs need to be quantised to be embedded in proof systems, since these require that values belong to some finite field. Quantisation is actually used beyond verification, as typical models reach a similar accuracy on short integers (such as 8-bit). A usual quantization scheme is [JKC$^+$18], which, as shown in zkCNN [LXZ21], can be integrated into large fields easily. A possible avenue for building verifiable CNNs without quantisation consists of using proof systems with native ring arithmetic such as [CCKP19, Sor22].

8.4.2 Our VE for Convolution

In this section we present a novel approach to proving convolutions efficiently by exploiting the symmetrical structure of a convolution operation. We write convolutions as matrix multiplications, seeking a more convenient form than the commonly used Toeplitz or circulant matrices (see [Uni] for further details).

Rewriting convolution. We observe that it is possible to re-write a convolution operation in the following compact form, where we specify a convolution of a 3×3 input X by a 2×2 kernel W.

[2] Note that 1D, 2D and 3D convolutions are equivalent in practice if the arrays are arranged adequately.

[3] It may seem that FFTs are best-performing, but in some practical cases matrix multiplication is actually preferred [CWV$^+$14].

$$\begin{bmatrix} x_1 & x_2 & x_4 & x_5 \\ x_2 & x_3 & x_5 & x_6 \\ x_4 & x_5 & x_7 & x_8 \\ x_5 & x_6 & x_8 & x_9 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix} = \begin{bmatrix} w_1x_1 + w_2x_2 + w_3x_4 + w_4x_5 \\ w_1x_2 + w_2x_3 + w_3x_5 + w_4x_6 \\ w_1x_4 + w_2x_5 + w_3x_7 + w_4x_8 \\ w_1x_5 + w_2x_6 + w_3x_8 + w_4x_9 \end{bmatrix} \tag{8.5}$$

The example is easily extended to an $n_x \times n_x$ input and $m \times m$ kernel. The matrix on the left-hand side has dimensions[4] $(n-m+1)^2 \times m^2$. More generically, this is the dimension of the flattened output times the dimension of the flattened weight matrix, which is $n_y^2 \times m^2$ for a convolutional layer that has an output of size $n_y \times n_y$.

We can extend this approach to capture multiple channels in a convolutional neural network. Let us recover usual CNN notation while ignoring layer indices; let X_σ be the input with channel $\sigma \in [c]$, and let $W_{\sigma,\tau}$ be the weight matrix where $\tau \in [d]$ is the output channel. Then, in matrix form (where $\hat{X}, \hat{W}$ are the transformed matrix representations of the data and weights in the form of Equation 8.5), we have that the layer's output Y is given by

$$Y = [Y_1|\cdots|Y_d] = \sum_{\sigma=1}^{c} \hat{X}_\sigma \cdot [\hat{W}_{\sigma,1}|\cdots|\hat{W}_{\sigma,d}]. \tag{8.6}$$

Namely, for each input channel σ we have the product of a $(n_y)^2 \times m^2$ matrix and a $m^2 \times d$ matrix. Each Y_τ is a column vector of length n_y^2 (i.e., a flattened channel of the output of the layer). If we apply the efficient VE for matrix multplication at this stage, we need to prove the result of a sum of c matrix multiplications, where the size of the matrices is $(n_y)^2 \times m^2$ and $m^2 \times d$.

Combining all input channels. The main efficiency advantage of our approach is that it is straightforward to extend the sumcheck equation for matrix multiplication (eq. (8.3)) to sum over the multiple channels. To do this, we can encode both $\hat{X}$ and $\hat{W}$ as trivariate polynomials given by $\hat{X}(x,y,\sigma) := \hat{X}_\sigma(x,y)$ and $\hat{W}(x,y,\sigma) := \hat{W}_\sigma(x,y)$ for every $\sigma \in [c]$. Then, we obtain the following sumcheck equation over $\boldsymbol{x}_1, \boldsymbol{x}_2$

$$\tilde{Y}(\boldsymbol{y}_1, \boldsymbol{y}_2) = \sum_{\substack{(\boldsymbol{x}_1,\boldsymbol{x}_2)\in \\ \{0,1\}^{2\lceil \log m\rceil + \lceil \log c\rceil}}} \tilde{X}(\boldsymbol{y}_1, \boldsymbol{x}_1, \boldsymbol{x}_2) \cdot \tilde{W}(\boldsymbol{x}_1, \boldsymbol{y}_2, \boldsymbol{x}_2). \tag{8.7}$$

Proposition 8.12. *Let* $\mathsf{VE}_{\mathsf{conv}}$ *be the VE scheme for two-dimensional convolution that is obtained by running the multivariate sumcheck protocol in Figure 8.4 on Equation 8.7. Then,* $\mathsf{VE}_{\mathsf{conv}}$ *is parametrized by two input fingerprints (one for* $\hat{X}$ *and one for* $\hat{W}$*), one output fingerprint (for* Y*), communication complexity* $|\pi| = (3\cdot(2\lceil\log m\rceil + \lceil\log c\rceil) + 2)\cdot\lambda$*, prover time* $\mathsf{t_P} = O\Big(c(n_y^2 m^2 + m^2 d)\Big)$*, verifier time* $\mathsf{t_V} = O\big(\log(cm^2)\big)$*, and soundness* $\epsilon = 2\cdot(2\lceil\log m\rceil + \lceil\log c\rceil)/|\mathbb{F}|$.

Intuitively, the asymptotic benefit of our approach compared to previous work is essentially given by expressing the input channels in columns in eq. (8.6), avoiding the overhead of padding the kernels to the input size.

[4] In this explanation, we are ignoring padding and stride parameters.

We also note that it is straightforward to extend equation 8.5 to support arbitrary padding or stride settings by modifying the reshaped input $\hat{X}$, as done in our implementation. An advantage of our method is that the output Y does not need to be reshaped after the VE is applied.

Transpose Convolution

The transpose convolution operation can be re-written as in Equation 8.5. For an example, let $m = n_x = 2$ over a single input channel $X_\sigma^{(k)}$. A basic upscaling transposed convolution yields $n_y = 3$ as below.

$$\begin{bmatrix} 0 & 0 & 0 & x_1 \\ 0 & 0 & x_1 & x_2 \\ 0 & 0 & x_2 & 0 \\ 0 & x_1 & 0 & x_3 \\ x_1 & x_2 & x_3 & x_4 \\ x_2 & 0 & x_4 & 0 \\ 0 & x_3 & 0 & 0 \\ x_3 & x_4 & 0 & 0 \\ x_4 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix} = \begin{bmatrix} x_1 w_4 \\ x_1 w_3 + x_2 w_4 \\ x_2 w_3 \\ x_1 w_2 + x_3 w_4 \\ \sum_{i=1}^4 x_i w_i \\ x_2 w_1 + x_4 w_3 \\ x_3 w_2 \\ x_3 w_1 + x_4 w_2 \\ x_4 w_1 \end{bmatrix} \tag{8.8}$$

For arbitrary input channels, the output will be a $n_y^2 \times d$ matrix. As before, we need to compute the sum over all input channels $\sigma \in [c]$, which can be done by extending the sumcheck as in Equation 8.7. This yields a prover time of $\mathsf{t_P} = O\left(c(n_y^2 m^2 + m^2 d)\right)$ and a verifier time of $\mathsf{t_V} = O\left(\log(cm^2)\right)$, exactly as for convolutions.

8.4.3 VEs for Other Neural Network Layers

Neural networks, and in general many data processing algorithms, incorporate several (generally simple) steps beyond convolution. We succinctly describe efficient ways of constructing VEs for the most usual operations.

Layer reshaping and pooling. For any sequence of operations that can be expressed without any multiplication gate (such as padding, rotation, compression, averaging, or any input rearrangement –e.g., the pre-processing required for the input of $\mathsf{VE_{conv}}$), one can encode the desired pattern in a wiring predicate $P(\boldsymbol{x}, \boldsymbol{y})$ and apply the multilinear sumcheck $\mathsf{VE_{SC}}$ as follows. For an input layer X and output layer Y, let $P(\boldsymbol{x}, \boldsymbol{y}) = t$ if the value $t \cdot X(\boldsymbol{x})$ is added to $Y(\boldsymbol{y})$. Then, $\mathsf{VE_{SC}}$ can be applied over $Y(\boldsymbol{y}) = \sum_{x \in \{0,1\}^\ell} P(\boldsymbol{x}, \boldsymbol{y}) \cdot X(\boldsymbol{x})$. Note that $\tilde{P}$ is sparse for most operations (except for weighted sums of many input values).

The predicate P natively supports average pooling. For max pooling, we recall the approach using auxiliary bit decompositions by zkCNN [LXZ21], that can be expressed as a VE. We also note that the described VE for reshaping can be easily used in combination with a many-to-one VE.

Normalization and linear transformations. Point-wise normalization, and in general input re-scaling operations that can be expressed as linear transformations of the form $x \mapsto \alpha x + \beta$, where $\alpha, \beta \in \mathbb{F}$, can be verified via a linear shift without any prover work. Indeed, multilinear fingerprints satisfy that given $X, Y \in \mathbb{F}^n$ such that $Y(\boldsymbol{x}) = \alpha \cdot X(\boldsymbol{x}) + \beta$ for all $\boldsymbol{x} \in \{0,1\}^\ell$, then $c_Y = \tilde{Y}(\boldsymbol{r}) = \alpha \cdot \tilde{X}(\boldsymbol{r}) + \beta$.

Activation functions. Due to their non-linearity, the verification of activation layers is particularly challenging and essentially reduces to two possibilities:

- Dedicated VEs with additional input. For instance, zkCNN [LXZ21] introduces a protocol for ReLU that requires additional bit decomposition, and can be easily seen as a VE.
- Approximate activation functions via polynomials, as is usual in the privacy-preserving ML literature. Quadratic polynomials may already offer good approximations [ASA20]. For this approach, one can construct a VE that evaluates quadratic polynomials via the following multilinear sumcheck (which follows from a GKR-like encoding):

  $$\tilde{Y}(\boldsymbol{y}) = \sum_{\boldsymbol{x}_1, \boldsymbol{x}_2 \in \{0,1\}^\ell} \tilde{I}(\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{y}) \cdot \tilde{X}(\boldsymbol{x}_1) \cdot \tilde{X}(\boldsymbol{x}_2)$$

  For a degree d polynomial, it is possible to use a binary tree of multiplications, such that prover time, verifier time, and communication complexity scale with $\log d$.

An alternative approach is using efficient lookup arguments [ZBK+22, PK22, ZGK+22, EFG22], where one can benefit from storing all values of the activation function (for quantised inputs) in a lookup table. We leave the investigation of lookups in our VE framework as interesting future work.

8.4.4 VE-based Proof System for Neural Networks

To construct a dedicated proof system for neural networks, we build a large VE scheme (denoted by $\mathsf{VE_{NN}}$), composed by several "gadget" $\mathsf{VE_k}$ for each of the layers of the network. Then, we use a multilinear polynomial commitment scheme to build a commit-and-prove AoK that achieves succinctness and efficient verification, following the blueprint of Proposition 8.5.

Following previous notation, let $X^{(k)}$ be the input and $f^{(k)}$ the function at layer k. We consider two general kinds of layers:

- Layers $f^{(k)}(X^{(k)})$ that apply an input transformation without additional parameters. For such $f^{(k)}$ we consider $\mathsf{VE_k}$ that take output fingerprints $\boldsymbol{c}_X^{(k+1)}$ (on randomness $\boldsymbol{r}_X^{(k+1)}$) and produce input fingerprints $\boldsymbol{c}_X^{(k)}$ (on randomness $\boldsymbol{r}_X^{(k)}$) and a (possibly empty) vector of fingerprints $\boldsymbol{c}_P^{(k)}$ (on randomness $\boldsymbol{r}_P^{(k+1)}$) to an auxiliary predicate P (see below).
- Layers $f^{(k)}(X^{(k)}, W^{(k)})$ that require additional parameters, not necessarily known to the verifier. For these functions, we consider $\mathsf{VE_k}$ that take output fingerprints $(\boldsymbol{c}_X^{(k+1)}, \boldsymbol{r}_X^{(k+1)})$ and produce input fingerprints $(\boldsymbol{c}_X^{(k)}, \boldsymbol{c}_W^{(k)}, \boldsymbol{c}_P^{(k)}, \boldsymbol{r}_X^{(k)}, \boldsymbol{r}_W^{(k)}, \boldsymbol{r}_P^{(k)})$.

Additionally, we require $\mathsf{VE_k}$ to take as many output fingerprints to $X^{(k+1)}$ as input fingerprints produced by $\mathsf{VE_{k+1}}$, such that they are compatible. Note, we can always achieve compatibility as one can reduce input fingerprints by applying $\mathsf{VE_{m\text{-}1}}$ (Proposition 8.11).

The predicates $P^{(k)}$ englobe any additional predicate that expresses the circuit at each layer, such as the wiring predicates in Equation (8.1) or additional auxiliary input as in [LXZ21]. For both $P^{(k)}$ and $W^{(k)}$, we define $W(\boldsymbol{k}, \boldsymbol{x}) := W^{(k)}(\boldsymbol{x})$ and $P(\boldsymbol{k}, \boldsymbol{x}) := P^{(k)}(\boldsymbol{x})$ via interpolation as

$$T(\boldsymbol{x}_k, \boldsymbol{x}) = \sum_{k=0}^{L-1} I(\boldsymbol{x}_k, k) \cdot T^{(k)}(\boldsymbol{x}) \tag{8.9}$$

where $T \in \{X, W\}$ and $I(\boldsymbol{x}_k, k)$ is the indicator function on $\lceil \log L \rceil$ variables. Without loss of generality, we pad every $T^{(k)}$ to have the same number of variables. For concrete implementations, it is possible to optimize the padding.

We describe $\mathsf{VE_{NN}}$ and its compiled AoK Π_{NN} in Figure 8.5. Soundness of $\mathsf{VE_{NN}}$ follows by Proposition 8.5 and the soundness of $\mathsf{VE_k}$ and $\mathsf{VE_{m\text{-}1}}$. Π_{NN} is an instantiation of the compiler of Section 8.2.3 and Proposition 8.7. By expressing the model parameters and predicates as single polynomials, it is possible to obtain, via many-to-one reductions, a single input fingerprint for each of $X := X^{(0)}$, W, and P. These fingerprints are verified in $\Pi_{\mathsf{NN}}.\mathsf{V}$ by three polynomial commitment opening proofs.

Proposition 8.13. *The protocol* $\mathsf{VE_{NN}}$ *is a VE scheme for a neural network architecture* $F_{\mathsf{NN},P}$*, parameterized by 1 output fingerprint (of* $\boldsymbol{y}$*), and 3 input fingerprints (of* $X^{(0)}$*,* W*, and* P*). Communication complexity, prover time, verifier time, and soundness result from the sum of the respective parameters of each* $\mathsf{VE_k}$ *and* $\mathsf{VE_{m\text{-}1}}$ *on Figure 8.5.*

Besides, Π_{NN} *is an argument of knowledge for the relation*

$$\begin{aligned}\mathcal{R}_{\mathsf{NN}} = \{&(\mathsf{com}_X, \mathsf{com}_W, \mathsf{com}_P, \boldsymbol{y}; X, W, P, o_X, o_W, o_P) : \\ &F_{\mathsf{NN},P}(X, W) = \boldsymbol{y} \wedge \mathsf{Com.Ver}(\mathsf{ck}, \mathsf{com}_T, T, o_T), \forall T \in \{X, W, P\}\}\,.\end{aligned}$$

Finally, we remark that our modular approach allows verifying pre- or post-processing operations in addition to the model, such as an aggregation phase. In this case, one can extend $\mathsf{VE_{NN}}$ and compose it with additional VE schemes for these operations.

8.4.5 Proof Batching

Our techniques are amenable to efficient batching where many evaluations $Y_i = F(X_i, W)$ for $i = 1, \ldots, N$ are verified in a single step. For VE schemes that rely on the multilinear sumcheck protocol from Figure 8.4, including the convolution VE introduced in this section, it is possible to reduce the verification time and communication complexity from linear to constant in the number of instances N.

Let $X(\boldsymbol{i}, \boldsymbol{x}) \in \mathbb{F}[X_1, \ldots X_{\log N + \ell_x}]$ be defined by $X(\boldsymbol{i}, \boldsymbol{x}) := X_i(\boldsymbol{x})$, and let $Y(\boldsymbol{i}, \boldsymbol{y})$ be defined analogously following equation (8.9). Then, one can run the protocol in Figure 8.4 over $Y(\boldsymbol{r}_i, \boldsymbol{r}_y)$ where $\boldsymbol{r}_i \in \mathbb{F}^{\log N}$ and $\boldsymbol{r}_y \in \mathbb{F}^{\ell_y}$. For instance, the sumcheck on the convolution VE (equation (8.7)) can be written as

```
$\underline{\mathsf{VE}_{\mathsf{NN}}.\mathsf{P}(c_y, r_y, F, (X, W, P))}$:

• $c_X^{(L)} \leftarrow c_y, r_X^{(L)} \leftarrow r_y$
• for $k = L-1, \dots, 0$ :
•     Run $(\boldsymbol{c}_X^{(k)}, \boldsymbol{c}_W^{(k)}, \boldsymbol{c}_P^{(k)}, \boldsymbol{r}_X^{(k)}, \boldsymbol{r}_W^{(k)}, \boldsymbol{r}_P^{(k)}) \leftarrow \mathsf{VE_k.P}\left(\boldsymbol{c}_X^{(k+1)}, \boldsymbol{r}_X^{(k+1)}, F^{(k)}, (X^{(k)}, W^{(k)}, P^{(k)})\right)$
• for $T \in \{W, P\}$ :
•     Run $(c_T, r_T) \leftarrow \mathsf{VE_{m\text{-}1}.P}\left(\boldsymbol{c}_T^{(0)}, \dots, \boldsymbol{c}_T^{(L-1)}, \boldsymbol{r}_T^{(0)}, \dots, \boldsymbol{r}_T^{(L-1)}, T\right)$
• Run $(c_X, r_X) \leftarrow \mathsf{VE_{m\text{-}1}.P}\left(\boldsymbol{c}_X^{(0)}, \boldsymbol{r}_X^{(0)}, X^{(0)}\right)$
• return $(c_X, c_W, c_P, r_X, r_W, r_P)$

$\underline{\Pi_{\mathsf{NN}}.\mathsf{P}((\mathsf{crs}, \mathsf{crs}'), (\mathsf{com}_X, \mathsf{com}_W, \mathsf{com}_P, \boldsymbol{y}; X, W, P, o_X, o_W, o_P))}$:

• for $T \in \{W, P\}$ : $\pi_{1,T} \leftarrow \mathsf{AoK_{Com}.Prove}(\mathsf{crs}', \mathsf{com}_T, (T, o_T))$
• Send $\pi_1 \leftarrow (\pi_{1,X}, \pi_{1,W}, \pi_{1,P})$
• Get $r_y \leftarrow\!\!\$\, \mathcal{D}_{\boldsymbol{y}}$ from V
• $c_y \leftarrow \mathsf{H}(\boldsymbol{y}, r_y)$
• Run $(c_X, c_W, c_P, r_X, r_W, r_P) \leftarrow \mathsf{VE_{NN}.P}(c_y, r_y, F, (X, W, P))$.
• for $T \in \{W, P\}$ : $\pi_T \leftarrow \mathsf{AoK_H.Prove}(\mathsf{crs}, (c_T, \mathsf{com}_T), (T, o_T))$
• Send $(\pi_X, \pi_W, \pi_P)$ to V

$\underline{\Pi_{\mathsf{NN}}.\mathsf{V}(\mathsf{ck}, (\mathsf{com}_X, \mathsf{com}_W, \mathsf{com}_P, \boldsymbol{y}))}$:

• Get $(\pi_{1,X}, \pi_{1,W}, \pi_{1,P})$
• Send $r_y \leftarrow\!\!\$\, \mathcal{D}_{\boldsymbol{y}}$ and compute $c_y \leftarrow \mathsf{H}(\boldsymbol{y}, r_y)$
• $r_T \leftarrow\!\!\$\, \mathcal{D}_{\mathcal{T}}$ for $T \in \{X, W, P\}$
• Run $(c_X, c_W, c_P, b_0) \leftarrow \mathsf{VE_{NN}.V}(c_y, r_y, F, r_X, r_W, r_P)$
• Get $(\pi_X, \pi_W, \pi_P)$
• for $T \in \{X, W, P\}$ $b_T \leftarrow \mathsf{AoK_{Com}.Ver}(\mathsf{crs}', \mathsf{com}_T, \pi_{1,T})$
  $\wedge\ \mathsf{AoK_H.Ver}(\mathsf{crs}, (\mathsf{com}_T, c_T, r_T), \pi_T)$.
• return $b_0 \wedge b_X \wedge b_W \wedge b_T$
```

Figure 8.5: *Modular construction of* $\mathsf{VE_{NN}}$ *and compilation to an interactive argument of knowledge* Π_{NN}*. The verifier* $\mathsf{VE_{NN}.V}$ *is omitted as it simply runs* $\mathsf{VE_k.V}$ *sequentially.*

$$\tilde{Y}(\boldsymbol{i}, \boldsymbol{y}_1, \boldsymbol{y}_2) = \sum_{\substack{(\boldsymbol{x}_1, \boldsymbol{x}_2) \in \\ \{0,1\}^{2\lceil \log m \rceil + \lceil \log c \rceil}}} \tilde{X}(\boldsymbol{i}, \boldsymbol{y}_1, \boldsymbol{x}_1, \boldsymbol{x}_2) \cdot \tilde{W}(\boldsymbol{x}_1, \boldsymbol{y}_2, \boldsymbol{x}_2). \tag{8.10}$$

The resulting VE increases the prover time by a factor of $\lceil \log N \rceil$ and maintains the same soundness, communication complexity and verifier time as their single-input counterpart.

8.4.6 Verifiable Recurrent Neural Networks

As an additional application of our modular framework, we show how to construct a protocol for the verification of recurrent neural network (RNN) predictions, a problem that has not been addressed efficiently in the literature. RNNs are a type of neural network designed to process sequential data such as time series or natural language text. Unlike feedforward neural networks, which process input data in a single pass and do not maintain memory, RNNs have a loop that allows information to be passed from one time step to the next, following a cyclic computation graph.

Let T be the length of the longest cycle in the graph described by a RNN of L layers. For example, $T = 1$ in the RNN in Figure 8.6, as the only cycle is a self-loop. We construct a VE that verifies the computation of S predictions $(Y^{(1)}, \ldots, Y^{(S)})$ from (streaming) inputs $(X_0^{(1)}, \ldots, X_0^{(S)})$ as follows.

- The prover computes the predictions and stores all intermediate values $X_k^{(i)}$ for $i = 0, \ldots, S$. Then, it "unrolls" the intermediate computations of the RNN as in Figure 8.7. The resulting computation trace is a circuit of depth $D = L + S \cdot T$ with an evident layer structure.
- The prover embeds each layer of the computation trace in a multilinear polynomial $Z_k(\boldsymbol{j}, \boldsymbol{x}) := Z_k^{(\boldsymbol{j})}(\boldsymbol{x})$ as in equation (8.9), and defines W_k, P_k accordingly. In total, one obtains D multilinear polynomials, structured as the layers in Figure 8.7.
- The VE proceeds similarly to the $\mathsf{VE_{NN}}$ of Figure 8.5. Instead of obtaining fingerprints for each $X_k^{(i)}$ via separate VEs, one can work directly with the (batched) Z_k as follows. Let $g_{k,k+1}$ be the product of multilinear polynomials that relates $Z_{k+1}(\boldsymbol{i}, \boldsymbol{y})$ and $Z_k(\boldsymbol{j}, \boldsymbol{x})$. $g_{k,k+1}$ contains factors of Z_k, W_k, P_k, subsequently defined over variables $(\boldsymbol{j}, \boldsymbol{x})$. Then, we have

  $$Z_{k+1}(\boldsymbol{i}, \boldsymbol{y}) = \sum_{\boldsymbol{x}, \boldsymbol{j}} g_{k,k+1}(Z_k, W_k, P_k)(\boldsymbol{i}, \boldsymbol{j}, \boldsymbol{x}, \boldsymbol{y}).$$

  Finally, by summing over all layers that are input to Z_{k+1} and polynomials $g_{k',k+1}$ for $k' \leq k$, we can verify a fingerprint of Z_{k+1} in a single sumcheck. The evaluation of $g_{k,k+1}$ yields fingerprints of Z_k, W_k, P_k that can be handled as in $\mathsf{VE_{NN}}$.

The resulting VE scheme has communication complexity and verifier time $\mathsf{t_V} = |\pi| = O(D \cdot \log S \cdot \ell_{\max})$, where $\ell_{\max} = \max_{k=0}^{L-1} \lceil \log n_k \rceil$ and n_k is the size of $\tilde{X}_k^{(\cdot)}$. Even if the proof size scales linearly in the length of the stream S, we believe that our approach may present good concrete performance for small streams, in particular due to the batching technique.

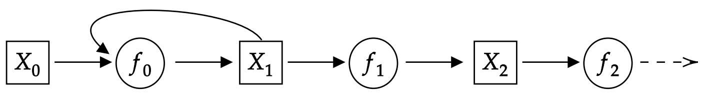

Figure 8.6: *Illustration of a RNN with a loop at layer X_1.*

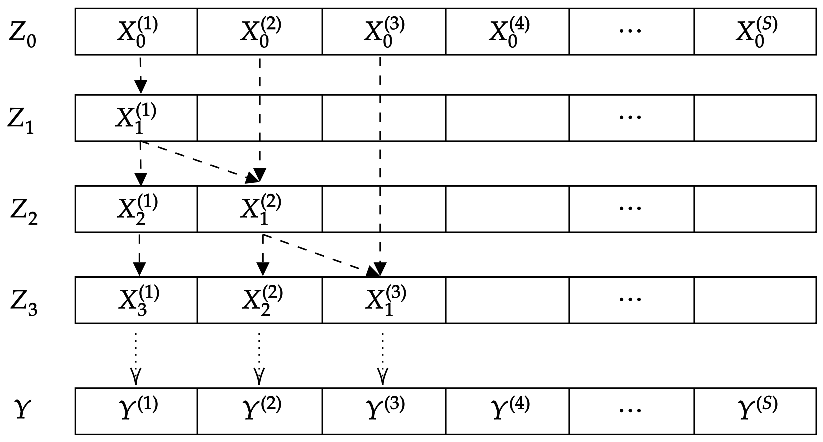

Figure 8.7: *Unrolled computation trace of a sequence of inputs $X_0^{(1)}, \dots, X_0^{(S)}$ in the RNN in Figure 8.6.*

8.4.7 VEs for Image Processing

The techniques developed in these sections find a direct application in the verification of image processing operations. For instance, convolution is used in applications such as edge detection (such as using Sobel or Canny kernels), image blurring (Gaussian blur), and feature extraction. Below we provide a brief description how to construct a VE for some common applications.

- Operations that require geometric modifications or rearrangements of the original picture, such as cropping, rotation, mirroring, padding, or partial censoring (i.e. removal or replacement of sectors of an image) can be verified following Section 8.4.3.
- For convolution-related operations, one can directly apply our $\mathsf{VE}_{\mathsf{CNN}}$ with the desired parameters.
- Multiple transformations can be merged in a single sumcheck by merging wiring predicates. For instance, rotation + cropping + input reshaping (1) and a posterior convolutional filtering (2) can be verified with only two sumchecks.

For images encoded in RGB or other multi-channel format, we can apply batching techniques for the channels as shown in equation (8.10). If negative values appear in convolution kernels, linear shifts need to be applied to avoid wrapping of field elements. We compare the performance of our approach to ZK-IMG [KHSS22] and PhotoProof [NT16] in Section 8.5.3.

8.5 Evaluation

In this section we discuss the performance of our solution and compare it to previous work. We focus the evaluation on our $\mathsf{VE}_{\mathsf{conv}}$ for convolution operations introduced in Section 8.4.2, as this is the most novel proof gadget compared to previous work.

8.5.1 Theoretical comparison

Recalling previous notation, let $n_x \times n_x$ be the input size, $n_y \times n_y$ the output size, $m \times m$ the kernel size, and c, d the number of input and output channels, respectively. We also write $n = \max\{n_x, n_y\}$. In Table 8.1, we compare the the prover and verifier running times as well as the proof size of our convolution VE to the FFT-based approach from zkCNN [LXZ21].

	$\mathsf{VE}_{\mathsf{conv}}$	zkCNN [LXZ21]
Prover $\mathsf{t_P}$	$O\left(m^2c(n_y^2+d)\right)$	$O(n^2cd)$
Verifier $\mathsf{t_V}$	$O\left(\log(n_y^2c)\right)$	$O(\log^2(n^2cd))$
Proof size $\lvert\pi\rvert$	$(6\lceil\log m\rceil + 3\lceil\log c\rceil + 2)\cdot\lambda$	$O(\log^2(n^2cd))$

Table 8.1: *Comparison between* $\mathsf{VE}_{\mathsf{conv}}$ *and the convolution proofs in zkCNN.*

We observe that our approach is always more efficient in communication complexity and verification time, while our prover is more efficient asymptotically when $m^2 \leq d$, which is often the case in practice (e.g., VGG16 presents $m = 3$ and d grows up to 512), and its running time is independent of d when the term $n_y^2m^2$ dominates in the sum. Additionally, in zkCNN, they need to either compute the FFT matrix or outsource this to the prover, thereby increasing proof size. We avoid all the complications of the multiple sumchecks in our direct approach. We also note that their FFT-sumcheck-based protocol can be easily expressed as a VE.

We note that, in many typical ML models, $n \gg m, c$ in early layers, and $c \gg n \approx m$ in "deep" intermediate layers. Hence, even if our approach does not outperform the prover time of the FFT-based polynomial multiplication approach in all parameter regimes, it will improve it for many parameter sets in intermediate layers. Based on the characteristics of the layer, one could select the most efficient VE for convolution.

8.5.2 Experimental evaluation

We implemented $\mathsf{VE}_{\mathsf{conv}}$ in Rust.[5] We use the `arkworks` library [con22] for implementing field arithmetic over the 256-bit prime field from the `bls12-381` curve, the same field used in [LXZ21]. We also utilize several components of the `arkworks` sumcheck library that implements the doubly efficient protocol in [XZZ+19].

We carry out different benchmarks in a virtual machine running Debian GNU/Linux with 8 cores Xeon-Gold-6154 at 3GHz and with 98 GB of RAM. Our implementation can be

[5] Our code is available at `https://github.com/imdea-software/MSCProof`

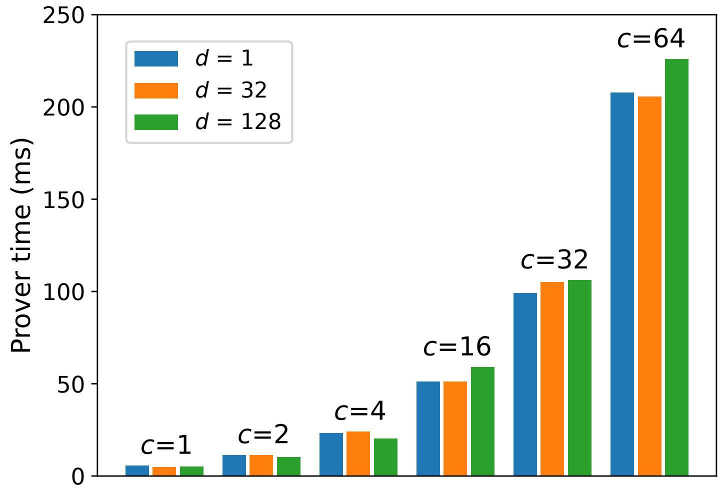

Figure 8.8: *Prover time for varying number of channels c, d and fixed $n = 64$ and $m = 4$.*

run using the natively supported parallelisation in `arkworks`, but we run our experiments on a single thread to facilitate comparison to previous work. All timings correspond to the average over 10 executions.

Single-channel convolution. Our first set of benchmarks run a single convolution with different input and kernel sizes. For small kernels $m = 4$, our VE prover requires 1.3 ms for a $n = 32$ input, and 98 ms for $n = 256$. In this parameter regime, our prover time is $5\times$ faster than the FFT prover (and also the naive prover) in [LXZ21]. Our prover also outperforms [LKKO20] by two orders of magnitude. For large convolution kernels, the prover in zkCNN remains faster.

Verification is very fast and scales logarithmically on the kernel size, as expected. Verifying a moderate-size convolution such as $n = 256$ (in fact, for any n) and $m = 8$ takes 0.157 ms, whereas large kernels $m = 128$ require 0.362 ms.

Multiple channels. Our approach is optimized for multiple convolution channels, as we show in Figure 8.8. We display our results for a small fixed kernel $m = 4$ and input $n = 64$, for c up to 64 and $d = 1, 32, 128$. As seen in the chart, the prover time is essentially constant in d since $n^2 \cdot m^2$ dominates the sum. The verifier time is also very small, ranging from 0.07 ms for $c = 1$ to 0.210 ms for $c = 64$, and also constant in d.

We do not have concrete running times for multiple channels in zkCNN, but we expect their prover time to increase linearly on $c \cdot t$.

Communication complexity. We also provide concrete figures of the communication complexity (equivalently, the proof size of the non-interactive protocol), which is deterministic for $\mathsf{VE}_{\mathsf{conv}}$ (Proposition 8.12). For the single-channel experiments, the proof size amounts to 0.64 KB for $m = 8$, and 1.4 KB for $m = 128$, for any input size. This is a $8\times$

improvement over zkCNN, which ranges from 5.6 KB to 8.4 KB for the same experiments. For the multi-channel setting in Figure 8.8, the instance $n = 64$, $m = 8$ and $c = 32$ yields a proof size of 1.12 KB for any d.

Image Processing. We benchmark a convolution proof of a 8×8 kernel (such as blurring) with a RGB image (720×480) with the goal of comparing to ZK-IMG [KHSS22], which already outperforms [NT16] by several orders of magnitude. The comparison is only approximate as their benchmarks are run on more powerful hardware than ours, and image sizes are not identical.

In this regime, $\mathsf{VE}_{\mathsf{conv}}$ takes 3.3 s of proving time, 0.12 ms of verification time and yields a proof size of 0.64 KB. In ZK-IMG, a $3\times$ larger 1280×720 convolution input involves 78 s of proving (ignoring key generation), 8.12 ms of verification, and 11 KB proof size (a $20\times$ increase).

For a 128×128 input, they report 2.7 s of proving time and 5.3 ms of verification on standard hardware. For the same size and a 8×8 kernel, our prover takes 110 ms ($25\times$ faster) and our verifier 0.117 ms.

Nevertheless, ZK-IMG implements a complete proof system, while our approach requires an additional polynomial commitment. We expect other simple transformations (cropping, padding, partial censoring...) to present similar running times.

Pre-Processing in $\mathsf{VE}_{\mathsf{conv}}$. As discussed in Sections 8.4.2 and 8.4.3, a pre-processing reshaping step, which can often be embedded into other steps such as activation layers, is required if $\mathsf{VE}_{\mathsf{conv}}$ is used to prove a standalone convolution. In that case, the sumcheck in Section 8.4.3 needs to be executed after $\mathsf{VE}_{\mathsf{conv}}$. We note that this step induces a minimal overhead as (1) the sumcheck involves strictly less variables and rounds than $\mathsf{VE}_{\mathsf{conv}}$, and (2) the prover already has the fingerprints to the reshaped input. We empirically evaluate this overhead in the VGG11 benchmark in Figure 8.9 as we discuss below.

Polynomial Commitment Overhead. A polynomial commitment is used in the AoK described in Proposition 8.13 but not at the VE level. The overhead induced by the PC depends on the chosen scheme and affects the efficiency of our solution and prior work's [LXZ21] in the same way. In the case of zkCNN, sumchecks take roughly 2/3 of the total prover time, whereas PCs take the remaining 1/3 (see [LXZ21], Table 1). Our improvements in the information-theoretic protocol significantly reduce the fraction taken by the sumchecks.

For completeness, we benchmark the multilinear KZG from HyperPlonk [CBBZ23] together with our $\mathsf{VE}_{\mathsf{conv}}$. For a single-channel convolution of $n = 256$, $m = 4$, a PC opening takes 400 ms, whereas the VE sumcheck prover takes 98 ms. The commit operation takes 191 ms. We remark that the PC opening cost gets further amortized when more VEs are composed sequentially. In general, the *deeper* the model is, the more significant the sumcheck overhead becomes.

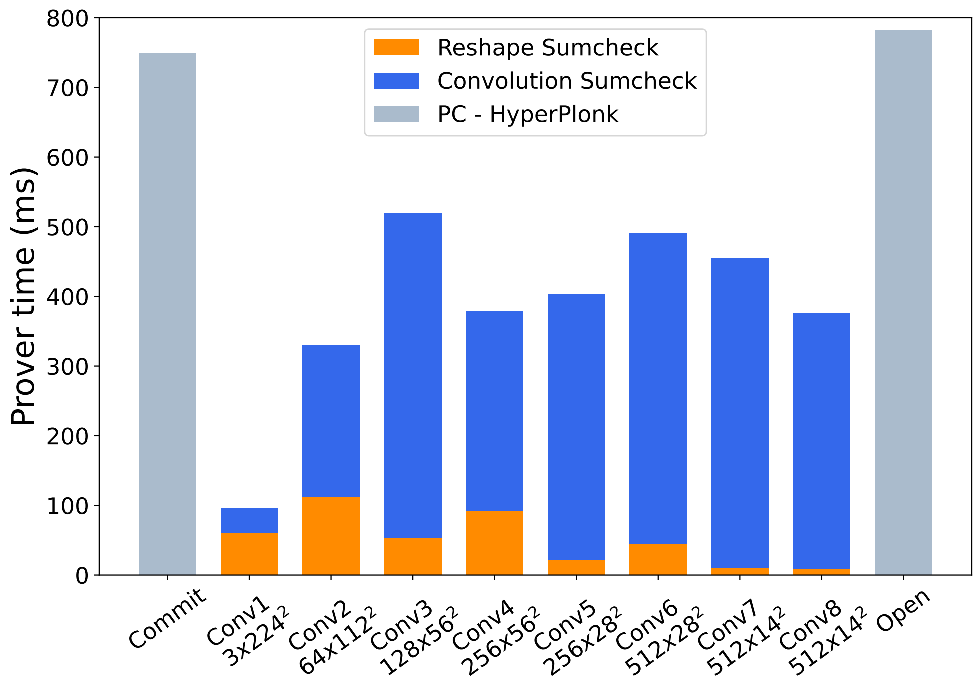

Figure 8.9: *Prover time for the convolution layers of a VGG11 network. The network presents $c \times n^2$ convolution layers as indicated in the x axis. All kernels are of size 3×3.*

CNN Evaluation We also benchmark the proof generation for the convolution layers of a VGG11 neural network (without activation and fully connected layers), that we summarize in Figure 8.9. As shown in the figure, the overhead of the polynomial commitment gets amortized across multiple layers. The figure also shows the small overhead from the reshape sumcheck compared to the convolution sumcheck. Finally, the empirical prover time for an increasing number of channels (larger c, d) and decreasing inputs (smaller n^2) remains similar, in agreement with the claimed prover complexity.

8.5.3 Discussion

Our protocols achieve, overall, faster prover times, reduced communication and faster verification times than existing solutions. As in other works [LKKO20, KHSS22, LXZ21], we found memory usage to be the main bottleneck, the reason being the dynamic programming technique used by the prover to compute the multilinear extensions. Yet, our approach allows for clearing the memory after every sequential step, as opposed to solutions such as [LKKO20] or [KHSS22] (built upon general-purpose proof systems). A solution towards improving memory bottlenecks is to trade memory usage for proving time by applying streaming algorithms for multilinear extensions [CTY11], which is an interesting direction for future work.

Part III

Conclusion

9 Conclusions and Future Work

This thesis presented several advances on succinct proof systems. In the first part (Chapters 4 to 7), we covered foundational aspects of succinct arguments, centred around the question of building *expressive succinct arguments from standard assumptions*. In the second part (Chapter 8), we adopted a more practical angle, focusing on *modularity, composability and concrete efficiency for real-world applications*. This final section aims to provide an interpretation of our results in a broader context, discuss their implications, and point out future work directions.

Functional Commitments. In Chapter 4, we introduced the first constructions of functional commitments for circuits based on falsifiable assumptions. Our construction provided a blueprint for building expressive FCs from chainable functional commitments for only quadratic equations, which we later constructed from both lattices and pairings. In hindsight, the main technical challenge that our construction overcomes is the evaluation of an unbounded number of multiplication gates, which is rooted in the concept of chainability. Both our lattice-based and our pairing-based scheme achieve chainability by committing to the circuit wires level-by-level and proving quadratic relations between them. In contrast, in (lattice-based) homomorphic computation approaches [dCP23, WW23b, WW23a] the noise of evaluated commitments grows significantly with each multiplication gate, enforcing an a-priori bound on the circuit depth at setup time.

Following the publication of our results, notable progress has been made in the area. Most prominently, [WW24b] refined our chainability approach and, by letting the prover commit to the entire computation trace of a circuit and embedding a projective space into the commitment key, achieved constant-size functional openings (i.e. $O(\lambda)$). Unfortunately, the size of the public parameters in their scheme becomes very large (quintic in the circuit *size*). Then, in Chapter 5 we introduced new techniques that enabled more compact public parameters than in Chapter 4, reducing their size from quintic to cubic in the circuit width. We see no major reason to believe that this effort cannot be pushed further in algebraic pairing-based solutions, e.g. aiming for public parameters that are quadratic in the circuit width, which we believe is an interesting open question.

An interesting observation is that the constructions in Chapter 4, Chapter 5 and

[WW24b] all seem to present a parameter-proof size trade-off. This is, for public parameters ck and a functional proof π, it holds that $|\mathsf{ck}| \cdot |\pi| \geq \lambda \cdot s$ for a circuit of size s. For example, in [WW24b] $|\mathsf{ck}| = O(\lambda s^5)$ and $|\pi| = O(1)$, whereas in Chapter 5 $|\mathsf{ck}| = O(\lambda w^5)$ and $|\pi| = O(\lambda d^2)$ for a circuit of depth d and width w (note that $s \leq d \cdot w$). Thus, building an algebraic pairing-based scheme (or showing impossibility) in which $|\mathsf{ck}| \cdot |\pi| = \mathsf{poly}(\lambda) \cdot o(s)$ is also a challenging open problem.

In general, these research directions attempt to bring functional commitments closer to the practical efficiency of SNARGs, which is a problem that is well-motivated in theory but also to some extent in practice. For instance, the use of random oracles in certain SNARG families recently faced a significant setback due to a new attack on the Fiat-Shamir transformation [KRS25]. Functional commitments offer an alternative recipe for building succinct arguments where issues with idealized models and extractability assumptions are minimized by design.

As a final remark, while some of our constructions rely on falsifiable but non-standard assumptions, there is a strong historical precedent for subsequently basing such results on standard assumptions. Notable examples include the progression of BARGs from q-type assumptions in [KPY19] to constructions based on LWE in [CJJ22], and the development of functional commitments themselves, from the HiKer assumption in [BCFL23] to MDDH in [WW24b]. Hence, we see falsifiable assumptions as a strong stepping stone towards achieving realizations from standard assumptions.

Batch Arguments. In Chapter 6, we presented the first algebraic construction of a batch argument for NP that achieves circuit-succinctness from standard pairing-based assumptions. Besides providing a formal connection between functional commitments and BARGs, our BARG construction is the first *black-box* construction from another cryptographic primitive with falsifiable security notions and standard-model realizations. We believe this connection enhances the understanding of BARGs and could result in novel efficient realizations. One notable open problem is to realize algebraic BARGs from lattice assumptions, which seems plausible since there exist several algebraic lattice-based functional commitments [ACL+22, WW23b, dCP23, BCFL23, WW23a]. Unfortunately, none of them satisfies the properties required by our FC-to-BARG compiler and enhancing them does not seem straightforward. Nevertheless, we believe that our approach provides valuable insights and a candidate construction blueprint towards this goal.

The main impediment towards making progress in this area, which is in general an important limitation of lattice-based succinct proofs, is that any meaningful form of extractability (such as somewhere extractability) seems to be hard to get without relying on random oracles (in algebraic schemes). While constructions of somewhere extractable commitments from lattices exist [HW15, CJJ21], they rely on homomorphic encryption techniques and their structure does not seem to support augmenting them with an algebraic proof system. Any significant advance in this direction, such as obtaining new algebraic constructions of somewhere extractable commitments from SIS, BASIS or k-R-ISIS, would be an important step towards practically efficient lattice-based BARGs.

Another intriguing research direction is to build concretely efficient BARGs for specific languages of interest, such as for linear relations and norm checks, for R1CS instances, or for pairing equations. As an example, the former would provide a mechanism for aggregating hash-and-sign lattice-based signatures [GPV08] such as Falcon [FHK+20] natively. Separately, it would be interesting to study whether BARGs for more expressive relations, such as for monotone policies [BBK+23, NWW24], can also be instantiated in a fully algebraic manner.

Homomorphic Signatures. Our linearly-homomorphic functional commitments for circuits in Chapter 4 and Chapter 5 resulted in algebraic instantiations of homomorphic signatures from standard assumptions that, for the first time, supported the evaluation of arbitrary computations. Later, in Chapter 7, we presented the first fully-succinct multi-key homomorphic signature that is secure under standard assumptions. This construction follows a relatively simple blueprint which relies on functional commitments and BARGs, finding a natural application for the constructions in the previous chapters. First, one commits to all signed messages and proves the evaluation of the function using a functional commitment opening. Second, one uses a BARG to (a) prove the verification of each signature and (b) prove that the commitment is well-formed. The main technical barrier that we overcome is the security proof itself, which has a notable technical complexity and forced us to introduce a somewhere extractable commitment as an additional building block.

A significant drawback of our multi-key homomporphic signature is its non-algebraic nature, as the BARG circuit needs to encode functional commitment updates and SEC verification operations. In practice, this translates into notable efficiency overheads. Hence, the main open question is whether one can build a fully algebraic scheme which could be realized efficiently. From lattice assumptions, the road ahead is obscure due to the lack of algebraic somewhere extractable primitives as described previously. From pairing assumptions, a natural approach may be to use structure-preserving tools for commitments and signatures. Unfortunately, there exist strong impossibility results on succinctly committing to group elements [AFG+16], so the road ahead is not much clearer. We remark that follow-up work on MKHS presents a more convoluted and highly non-algebraic construction [ACG24], so it does not seem likely that their approach could yield efficient solutions either.

Further directions worth exploring are building MKHS (or even HS) with advanced properties (such as supporting threshold or monotone policies on the validity of the signatures), and improving on the best possible succinctness of our instantiation, which yields evaluated signatures of size $O(\lambda^2)$. Towards this latter goal, we observe that using rate-1 BARGs [PP22] does not seem to provide any advantage, as they introduce an additive $\mathsf{poly}(\lambda)$ factor in the proof size.

Concretely Efficient Proofs. In Chapter 8, we depart from realizing new primitives and breaking asymptotic bounds and focus on building concretely efficient proofs for applications of interest. Our contributions are twofold. First, we present a modular framework that allows one to seamlessly compose sumcheck-based proofs for both general and domain-specific relations. Second, we present and implement an efficient proof for convolution operations that outperforms the state-of-the-art in both speed and succinctness. By leveraging sequential composition at the information-theoretic level, our proof generation algorithms achieve not only low computation time but also high memory efficiency. Hence, our framework enables efficient verifiable data pipelines for applications that require large amounts of data. Among these, we highlight applications in verifiable neural network inference and verifiable image processing.

Both application domains have witnessed significant research activity in recent years. Regarding verifiable machine learning, the increasing prevalence of AI in decision-making processes has increased the need for ensuring fairness, both in training [GGJ$^+$23, APKP24] and in inference [SLZ24]. Notably, zero-knowledge proofs applied to machine learning are no longer an academic quest exclusively, but have also been deployed in commercial products, in some cases following our modular sumcheck-based approach [Lag25].

Simultaneously, the surge of generative AI has strongly motivated the problem of certifying the genuine manipulation of images and videos. The applications of verifiable media processing are wide and include journalism, law enforcement, fake news detection, and even art or photography contests. Given the scale of the problem, cryptography seems to be the only viable solution, as it is not feasible to have a human certify every image or video. Recent works are designing verifiable image processing pipelines that improve our solutions in different angles [DEH25, MVVZ24, DCB25].

The construction of efficient proof systems is not necessarily incompatible with the directions taken in first part of the thesis. As an example, a recent trend for building efficient proof systems for large computations is to use proof recursion [Val08], generally through folding schemes [KST22]. As part of the recursion step, one of the most expensive operations is proving hash function evaluations, which instantiate random oracles in practice. To circumvent this, a yet unexplored pathway is to employ proof systems without random oracles, such as functional commitments and BARGs. Whether this approach can yield competitive instantiations for recursive proofs is an interesting open question.

Final Thoughts. To conclude, we would like to emphasize the importance of cryptanalysis and security research on proof systems developed for practical deployment, such as those in Chapter 8. The proof systems field is fast-moving and highly complex, yet there is a relatively short delay between the surge of academic solutions and their commercial adoption. This raises several concerns in practice.

From a cryptanalytical standpoint, the recent attacks on the Fiat-Shamir transform [KRS25] actually show that our sumcheck-based proofs may be vulnerable due to our use of random oracles. Even if concrete attacks do not seem feasible, such results reflect the difficulty of building practical proof systems that stand the test of time. Similarly, it highlights the importance of the fundamental research done in the first part of the thesis, which seeks solid security foundations for these primitives.

In parallel, there is a long road ahead in studying the security of proof systems at the level of implementation and protocol design. Despite securing billions of euros in digital assets, current implementations often lack rigorous scrutiny. Standardization efforts are scarce as companies tend to develop different proof systems, and there are few established guidelines for developers. These concerns are not limited to blockchains but extend to protocols in the digital identity and authentication domain, which often rely on tailor-made zero knowledge proofs. Governments across the world are actively developing these protocols, yet they often do so with poor scientific rigour and suboptimal security choices.

As a general stance, the field would benefit greatly from increased interaction between cryptographers and system designers, or by having more cryptographers taking inspiration from real-world problems. While proof systems do have potential to ensure trust, the cryptography for a verifiable world must still confront significant challenges to secure trust in its own foundations.

Bibliography

[ABB$^+$24] Masayuki Abe, David Balbás, Dung Bui, Miyako Ohkubo, Zehua Shang, Akira Takahashi, and Mehdi Tibouchi. Critical Rounds in Multi-Round Proofs: Proof of Partial Knowledge, Trapdoor Commitments, and Advanced Signatures. Cryptology ePrint Archive, Paper 2024/1766, 2024. URL: `https://eprint.iacr.org/2024/1766`.

[ABF24] Gaspard Anthoine, David Balbás, and Dario Fiore. Fully-succinct multi-key homomorphic signatures from standard assumptions. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part III*, volume 14922 of *LNCS*, pages 317–351. Springer, Cham, August 2024. `doi:10.1007/978-3-031-68382-4_10`.

[ABO$^+$24] Masayuki Abe, Andrej Bogdanov, Miyako Ohkubo, Alon Rosen, Zehua Shang, and Mehdi Tibouchi. CDS composition of multi-round protocols. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part IX*, volume 14928 of *LNCS*, pages 391–423. Springer, Cham, August 2024. `doi:10.1007/978-3-031-68400-5_12`.

[ACF21] Thomas Attema, Ronald Cramer, and Serge Fehr. Compressing proofs of k-out-of-n partial knowledge. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part IV*, volume 12828 of *LNCS*, pages 65–91, Virtual Event, August 2021. Springer, Cham. `doi:10.1007/978-3-030-84259-8_3`.

[ACG24] Abtin Afshar, Jiaqi Cheng, and Rishab Goyal. Leveled fully-homomorphic signatures from batch arguments. Cryptology ePrint Archive, Report 2024/931, 2024. URL: `https://eprint.iacr.org/2024/931`.

[ACL$^+$22] Martin R. Albrecht, Valerio Cini, Russell W. F. Lai, Giulio Malavolta, and Sri Aravinda Krishnan Thyagarajan. Lattice-based SNARKs: Publicly verifiable, preprocessing, and recursively composable - (extended abstract). In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part II*, volume 13508 of *LNCS*, pages 102–132. Springer, Cham, August 2022. `doi:10.1007/978-3-031-15979-4_4`.

[AEE$^+$21] Lukas Aumayr, Oguzhan Ersoy, Andreas Erwig, Sebastian Faust, Kristina Hostáková, Matteo Maffei, Pedro Moreno-Sanchez, and Siavash Riahi. Generalized channels from limited blockchain scripts and adaptor signatures. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part II*, volume 13091 of *LNCS*, pages 635–664. Springer, Cham, December 2021. doi:10.1007/978-3-030-92075-3_22.

[AFG$^+$16] Masayuki Abe, Georg Fuchsbauer, Jens Groth, Kristiyan Haralambiev, and Miyako Ohkubo. Structure-preserving signatures and commitments to group elements. *Journal of Cryptology*, 29(2):363–421, April 2016. doi:10.1007/s00145-014-9196-7.

[AFLN24] Martin R. Albrecht, Giacomo Fenzi, Oleksandra Lapiha, and Ngoc Khanh Nguyen. SLAP: Succinct lattice-based polynomial commitments from standard assumptions. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part VII*, volume 14657 of *LNCS*, pages 90–119. Springer, Cham, May 2024. doi:10.1007/978-3-031-58754-2_4.

[AH87] William Aiello and Johan Håstad. Perfect zero-knowledge languages can be recognized in two rounds. In *28th FOCS*, pages 439–448. IEEE Computer Society Press, October 1987. doi:10.1109/SFCS.1987.47.

[AL11] Nuttapong Attrapadung and Benoît Libert. Homomorphic network coding signatures in the standard model. In Dario Catalano, Nelly Fazio, Rosario Gennaro, and Antonio Nicolosi, editors, *PKC 2011*, volume 6571 of *LNCS*, pages 17–34. Springer, Berlin, Heidelberg, March 2011. doi:10.1007/978-3-642-19379-8_2.

[AL21] Martin R. Albrecht and Russell W. F. Lai. Subtractive sets over cyclotomic rings - limits of Schnorr-like arguments over lattices. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part II*, volume 12826 of *LNCS*, pages 519–548, Virtual Event, August 2021. Springer, Cham. doi:10.1007/978-3-030-84245-1_18.

[AP19] Diego F. Aranha and Elena Pagnin. The simplest multi-key linearly homomorphic signature scheme. In Peter Schwabe and Nicolas Thériault, editors, *LATINCRYPT 2019*, volume 11774 of *LNCS*, pages 280–300. Springer, Cham, October 2019. doi:10.1007/978-3-030-30530-7_14.

[APKP24] Kasra Abbaszadeh, Christodoulos Pappas, Jonathan Katz, and Dimitrios Papadopoulos. Zero-knowledge proofs of training for deep neural networks. In *ACM CCS 2024*, pages 4316–4330. ACM Press, November 2024. doi:10.1145/3658644.3670316.

[AS92] Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs; A new characterization of NP. In *33rd FOCS*, pages 2–13. IEEE Computer Society Press, October 1992. doi:10.1109/SFCS.1992.267824.

[ASA20] Ramy E. Ali, Jinhyun So, and Amir Salman Avestimehr. On Polynomial Approximations for Privacy-Preserving and Verifiable ReLU Networks. *CoRR*, abs/2011.05530, 2020. URL: `https://arxiv.org/abs/2011.05530`, `arXiv:2011.05530`.

[BBG05] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. Hierarchical identity based encryption with constant size ciphertext. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 440–456. Springer, Berlin, Heidelberg, May 2005. `doi:10.1007/11426639_26`.

[BBH25] Matilda Backendal, David Balbás, and Miro Haller. Group Key Progression: Strong Security for Shared Persistent Data. Cryptology ePrint Archive, Paper 2025/1028, 2025. URL: `https://eprint.iacr.org/2025/1028`.

[BBK$^+$23] Zvika Brakerski, Maya Farber Brodsky, Yael Tauman Kalai, Alex Lombardi, and Omer Paneth. SNARGs for monotone policy batch NP. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part II*, volume 14082 of *LNCS*, pages 252–283. Springer, Cham, August 2023. `doi:10.1007/978-3-031-38545-2_9`.

[BC24] Dan Boneh and Binyi Chen. LatticeFold: A lattice-based folding scheme and its applications to succinct proof systems. Cryptology ePrint Archive, Report 2024/257, 2024. URL: `https://eprint.iacr.org/2024/257`.

[BCFL23] David Balbás, Dario Catalano, Dario Fiore, and Russell W. F. Lai. Chainable functional commitments for unbounded-depth circuits. In Guy N. Rothblum and Hoeteck Wee, editors, *TCC 2023, Part III*, volume 14371 of *LNCS*, pages 363–393. Springer, Cham, November / December 2023. `doi:10.1007/978-3-031-48621-0_13`.

[BCG$^+$14] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474. IEEE Computer Society Press, May 2014. `doi:10.1109/SP.2014.36`.

[BCG$^+$15] Eli Ben-Sasson, Alessandro Chiesa, Matthew Green, Eran Tromer, and Madars Virza. Secure sampling of public parameters for succinct zero knowledge proofs. In *2015 IEEE Symposium on Security and Privacy*, pages 287–304. IEEE Computer Society Press, May 2015. `doi:10.1109/SP.2015.25`.

[BCG22] David Balbás, Daniel Collins, and Phillip Gajland. Analysis and Improvements of the Sender Keys Protocol for Group Messaging. In *XVII Reunión española sobre criptología y seguridad de la información. RECSI 2022*, volume 265, page 25. Ed. Universidad de Cantabria, 2022.

[BCG23] David Balbás, Daniel Collins, and Phillip Gajland. WhatsUpp with sender keys? Analysis, improvements and security proofs. In Jian Guo and Ron Steinfeld, editors, *ASIACRYPT 2023, Part V*, volume 14442 of *LNCS*, pages 307–341. Springer, Singapore, December 2023. doi:10.1007/978-981-99-8733-7_10.

[BCI$^+$13] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. Succinct non-interactive arguments via linear interactive proofs. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 315–333. Springer, Berlin, Heidelberg, March 2013. doi:10.1007/978-3-642-36594-2_18.

[BCJP24] Maya Farber Brodsky, Arka Rai Choudhuri, Abhishek Jain, and Omer Paneth. Monotone-policy aggregate signatures. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part IV*, volume 14654 of *LNCS*, pages 168–195. Springer, Cham, May 2024. doi:10.1007/978-3-031-58737-5_7.

[BCR$^+$19] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct arguments for R1CS. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 103–128. Springer, Cham, May 2019. doi:10.1007/978-3-030-17653-2_4.

[BCS16] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part II*, volume 9986 of *LNCS*, pages 31–60. Springer, Berlin, Heidelberg, October / November 2016. doi:10.1007/978-3-662-53644-5_2.

[BCV23] David Balbás, Daniel Collins, and Serge Vaudenay. Cryptographic administration for secure group messaging. In Joseph A. Calandrino and Carmela Troncoso, editors, *USENIX Security 2023*, pages 1253–1270. USENIX Association, August 2023. URL: https://www.usenix.org/conference/usenixsecurity23/presentation/balbas.

[BF11] Dan Boneh and David Mandell Freeman. Homomorphic signatures for polynomial functions. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 149–168. Springer, Berlin, Heidelberg, May 2011. doi:10.1007/978-3-642-20465-4_10.

[BFG$^+$23] David Balbás, Dario Fiore, María Isabel González Vasco, Damien Robissout, and Claudio Soriente. Modular sumcheck proofs with applications to machine learning and image processing. In Weizhi Meng, Christian Damsgaard Jensen, Cas Cremers, and Engin Kirda, editors, *ACM CCS 2023*, pages 1437–1451. ACM Press, November 2023. doi:10.1145/3576915.3623160.

[BFKW09] Dan Boneh, David Freeman, Jonathan Katz, and Brent Waters. Signing a linear subspace: Signature schemes for network coding. In Stanislaw Jarecki and

Gene Tsudik, editors, *PKC 2009*, volume 5443 of *LNCS*, pages 68–87. Springer, Berlin, Heidelberg, March 2009. doi:10.1007/978-3-642-00468-1_5.

[BFL90] László Babai, Lance Fortnow, and Carsten Lund. Non-deterministic exponential time has two-prover interactive protocols. In *31st FOCS*, pages 16–25. IEEE Computer Society Press, October 1990. doi:10.1109/FSCS.1990.89520.

[BFL25a] David Balbás, Dario Fiore, and Russell Lai. Circuit-Succinct Algebraic Batch Arguments from Projective Functional Commitments. Unpublished manuscript, 2025.

[BFL25b] David Balbás, Dario Fiore, and Russell Lai. Pairing-based Functional Commitments for Circuits, Revisited. Unpublished manuscript, 2025.

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications (extended abstract). In *20th ACM STOC*, pages 103–112. ACM Press, May 1988. doi:10.1145/62212.62222.

[BFR+25] David Balbás, Dario Fiore, Georgios Raikos, Damien Robissout, and Claudio Soriente. FHorgEt: A Cryptographic Solution for Secure Machine Unlearning. Unpublished manuscript, 2025.

[BGG+90] Michael Ben-Or, Oded Goldreich, Shafi Goldwasser, Johan Håstad, Joe Kilian, Silvio Micali, and Phillip Rogaway. Everything provable is provable in zero-knowledge. In Shafi Goldwasser, editor, *CRYPTO'88*, volume 403 of *LNCS*, pages 37–56. Springer, New York, August 1990. doi:10.1007/0-387-34799-2_4.

[BGH19] Sean Bowe, Jack Grigg, and Daira Hopwood. Halo: Recursive proof composition without a trusted setup. Cryptology ePrint Archive, Report 2019/1021, 2019. URL: https://eprint.iacr.org/2019/1021.

[BGI+01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 1–18. Springer, Berlin, Heidelberg, August 2001. doi:10.1007/3-540-44647-8_1.

[BGLS03] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 416–432. Springer, Berlin, Heidelberg, May 2003. doi:10.1007/3-540-39200-9_26.

[BHK17] Zvika Brakerski, Justin Holmgren, and Yael Tauman Kalai. Non-interactive delegation and batch NP verification from standard computational assumptions. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *49th ACM STOC*, pages 474–482. ACM Press, June 2017. doi:10.1145/3055399.3055497.

[BP04] Mihir Bellare and Adriana Palacio. The knowledge-of-exponent assumptions and 3-round zero-knowledge protocols. In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 273–289. Springer, Berlin, Heidelberg, August 2004. `doi:10.1007/978-3-540-28628-8_17`.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM CCS 93*, pages 62–73. ACM Press, November 1993. `doi:10.1145/168588.168596`.

[BS23] Ward Beullens and Gregor Seiler. LaBRADOR: Compact proofs for R1CS from module-SIS. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part V*, volume 14085 of *LNCS*, pages 518–548. Springer, Cham, August 2023. `doi:10.1007/978-3-031-38554-4_17`.

[BSW11] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In Yuval Ishai, editor, *TCC 2011*, volume 6597 of *LNCS*, pages 253–273. Springer, Berlin, Heidelberg, March 2011. `doi:10.1007/978-3-642-19571-6_16`.

[CB25] Binyi Chen and Dan Boneh. LatticeFold+: Faster, Simpler, Shorter Lattice-Based Folding for Succinct Proof Systems. In *CRYPTO 2025*. Springer-Verlag, 2025.

[CBBZ23] Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. HyperPlonk: Plonk with linear-time prover and high-degree custom gates. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part II*, volume 14005 of *LNCS*, pages 499–530. Springer, Cham, April 2023. `doi:10.1007/978-3-031-30617-4_17`.

[CCH$^+$19] Ran Canetti, Yilei Chen, Justin Holmgren, Alex Lombardi, Guy N. Rothblum, Ron D. Rothblum, and Daniel Wichs. Fiat-Shamir: from practice to theory. In Moses Charikar and Edith Cohen, editors, *51st ACM STOC*, pages 1082–1090. ACM Press, June 2019. `doi:10.1145/3313276.3316380`.

[CCKP19] Shuo Chen, Jung Hee Cheon, Dongwoo Kim, and Daejun Park. Verifiable computing for approximate computation. Cryptology ePrint Archive, Report 2019/762, 2019. URL: `https://eprint.iacr.org/2019/762`.

[CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In Yvo Desmedt, editor, *CRYPTO'94*, volume 839 of *LNCS*, pages 174–187. Springer, Berlin, Heidelberg, August 1994. `doi:10.1007/3-540-48658-5_19`.

[CF13] Dario Catalano and Dario Fiore. Vector commitments and their applications. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *PKC 2013*, volume 7778 of *LNCS*, pages 55–72. Springer, Berlin, Heidelberg, February / March 2013. `doi:10.1007/978-3-642-36362-7_5`.

[CFF$^+$21] Matteo Campanelli, Antonio Faonio, Dario Fiore, Anaïs Querol, and Hadrián Rodríguez. Lunar: A toolbox for more efficient universal and updatable zkSNARKs and commit-and-prove extensions. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part III*, volume 13092 of *LNCS*, pages 3–33. Springer, Cham, December 2021. `doi:10.1007/978-3-030-92078-4_1`.

[CFF$^+$24] Matteo Campanelli, Antonio Faonio, Dario Fiore, Tianyu Li, and Helger Lipmaa. Lookup arguments: Improvements, extensions and applications to zero-knowledge decision trees. In Qiang Tang and Vanessa Teague, editors, *PKC 2024, Part II*, volume 14602 of *LNCS*, pages 337–369. Springer, Cham, April 2024. `doi:10.1007/978-3-031-57722-2_11`.

[CFGV13] Dario Catalano, Dario Fiore, Rosario Gennaro, and Konstantinos Vamvourellis. Algebraic (trapdoor) one-way functions and their applications. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 680–699. Springer, Berlin, Heidelberg, March 2013. `doi:10.1007/978-3-642-36594-2_38`.

[CFM08] Dario Catalano, Dario Fiore, and Mariagrazia Messina. Zero-knowledge sets with short proofs. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 433–450. Springer, Berlin, Heidelberg, April 2008. `doi:10.1007/978-3-540-78967-3_25`.

[CFN15] Dario Catalano, Dario Fiore, and Luca Nizzardo. Programmable hash functions go private: Constructions and applications to (homomorphic) signatures with shorter public keys. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 254–274. Springer, Berlin, Heidelberg, August 2015. `doi:10.1007/978-3-662-48000-7_13`.

[CFQ19] Matteo Campanelli, Dario Fiore, and Anaïs Querol. LegoSNARK: Modular design and composition of succinct zero-knowledge proofs. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 2075–2092. ACM Press, November 2019. `doi:10.1145/3319535.3339820`.

[CFS17] Alessandro Chiesa, Michael A. Forbes, and Nicholas Spooner. A zero knowledge sumcheck and its applications. Cryptology ePrint Archive, Report 2017/305, 2017. URL: `https://eprint.iacr.org/2017/305`.

[CFT22] Dario Catalano, Dario Fiore, and Ida Tucker. Additive-homomorphic functional commitments and applications to homomorphic signatures. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part IV*, volume 13794 of *LNCS*, pages 159–188. Springer, Cham, December 2022. `doi:10.1007/978-3-031-22972-5_6`.

[CFW12] Dario Catalano, Dario Fiore, and Bogdan Warinschi. Efficient network coding signatures in the standard model. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *PKC 2012*, volume 7293 of *LNCS*, pages 680–696. Springer, Berlin, Heidelberg, May 2012. doi:10.1007/978-3-642-30057-8_40.

[CFW14] Dario Catalano, Dario Fiore, and Bogdan Warinschi. Homomorphic signatures with efficient verification for polynomial functions. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 371–389. Springer, Berlin, Heidelberg, August 2014. doi:10.1007/978-3-662-44371-2_21.

[CGH98] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited (preliminary version). In *30th ACM STOC*, pages 209–218. ACM Press, May 1998. doi:10.1145/276698.276741.

[CGJ$^+$23] Arka Rai Choudhuri, Sanjam Garg, Abhishek Jain, Zhengzhong Jin, and Jiaheng Zhang. Correlation intractability and SNARGs from sub-exponential DDH. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part IV*, volume 14084 of *LNCS*, pages 635–668. Springer, Cham, August 2023. doi:10.1007/978-3-031-38551-3_20.

[Cha81] David L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–90, February 1981. doi:10.1145/358549.358563.

[CHM$^+$20] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Psi Vesely, and Nicholas P. Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 738–768. Springer, Cham, May 2020. doi:10.1007/978-3-030-45721-1_26.

[CHN$^+$22] Haixia Chen, Xinyi Huang, Jianting Ning, Futai Zhang, and Chao Lin. Vils: A verifiable image licensing system. *IEEE Transactions on Information Forensics and Security*, 17:1420–1434, 2022. doi:10.1109/TIFS.2022.3162105.

[CJJ21] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. Non-interactive batch arguments for NP from standard assumptions. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part IV*, volume 12828 of *LNCS*, pages 394–423, Virtual Event, August 2021. Springer, Cham. doi:10.1007/978-3-030-84259-8_14.

[CJJ22] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. SNARGs for $\mathcal{P}$ from LWE. In *62nd FOCS*, pages 68–79. IEEE Computer Society Press, February 2022. doi:10.1109/FOCS52979.2021.00016.

[CL01] Jan Camenisch and Anna Lysyanskaya. An identity escrow scheme with appointed verifiers. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 388–407. Springer, Berlin, Heidelberg, August 2001. `doi:10.1007/3-540-44647-8_23`.

[CMT12a] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. Practical verified computation with streaming interactive proofs. In Shafi Goldwasser, editor, *ITCS 2012*, pages 90–112. ACM, January 2012. `doi:10.1145/2090236.2090245`.

[CMT12b] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. Practical verified computation with streaming interactive proofs. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, page 90–112, New York, NY, USA, 2012. Association for Computing Machinery. `doi:10.1145/2090236.2090245`.

[con22] Arkworks contributors. `arkworks` zksnark ecosystem, 2022. URL: `https://arkworks.rs`.

[Cou00] Council of European Union. Council Directive 2000/43/EC of 29 June 2000 implementing the principle of equal treatment between persons irrespective of racial or ethnic origin, 2000.
`https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32000L0043`.

[Cou04] Council of European Union. Council directive 2004/113/ec of 13 december 2004 implementing the principle of equal treatment between men and women in the access to and supply of goods and services, 2004.
`https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32004L0113`.

[CT10] Alessandro Chiesa and Eran Tromer. Proof-carrying data and hearsay arguments from signature cards. In *Innovations in Computer Science - ICS 2010, Tsinghua University, Beijing, China, January 5-7, 2010. Proceedings*, pages 310–331, 2010.

[CTY11] Graham Cormode, Justin Thaler, and Ke Yi. Verifying computations with streaming interactive proofs. *Proc. VLDB Endow.*, 5(1):25–36, sep 2011. `doi:10.14778/2047485.2047488`.

[CWV$^+$14] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*, 2014.

[Dam90] Ivan Damgård. On the existence of bit commitment schemes and zero-knowledge proofs. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 17–27. Springer, New York, August 1990. doi:10.1007/0-387-34805-0_3.

[Dam92] Ivan Damgård. Towards practical public key systems secure against chosen ciphertext attacks. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 445–456. Springer, Berlin, Heidelberg, August 1992. doi:10.1007/3-540-46766-1_36.

[Dam00] Ivan Damgård. Efficient concurrent zero-knowledge in the auxiliary string model. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 418–430. Springer, Berlin, Heidelberg, May 2000. doi:10.1007/3-540-45539-6_30.

[DCB25] Trisha Datta, Binyi Chen, and Dan Boneh. VerITAS: Verifying image transformations at scale. In *IEEE Symposium in Security and Privacy*, San Francisco, CA, USA, 2025. IEEE.

[dCP23] Leo de Castro and Chris Peikert. Functional commitments for all functions, with transparent setup and from SIS. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part III*, volume 14006 of *LNCS*, pages 287–320. Springer, Cham, April 2023. doi:10.1007/978-3-031-30620-4_10.

[DEH25] Stefan Dziembowski, Shahriar Ebrahimi, and Parisa Hassanizadeh. Vimz: Private proofs of image manipulation using folding-based zksnarks. In *Proc. Privacy Enhancing Technologies (PETS)*, 2025. URL: https://doi.org/10.56553/popets-2025-0065, doi:10.56553/POPETS-2025-0065.

[Des93] Yvo Desmedt. Computer security by redefining what a computer is. NSPW, 1993.

[DFGK14] George Danezis, Cédric Fournet, Jens Groth, and Markulf Kohlweiss. Square span programs with applications to succinct NIZK arguments. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part I*, volume 8873 of *LNCS*, pages 532–550. Springer, Berlin, Heidelberg, December 2014. doi:10.1007/978-3-662-45611-8_28.

[DGKV22] Lalita Devadas, Rishab Goyal, Yael Kalai, and Vinod Vaikuntanathan. Rate-1 non-interactive arguments for batch-NP and applications. In *63rd FOCS*, pages 1057–1068. IEEE Computer Society Press, October / November 2022. doi:10.1109/FOCS54457.2022.00103.

[DV16] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*, 2016.

[DWW24] Lalita Devadas, Brent Waters, and David J. Wu. Batching adaptively-sound SNARGs for NP. In *TCC 2024, Part II*, LNCS, pages 339–370. Springer, Cham, November 2024. doi:10.1007/978-3-031-78017-2_12.

[EFG22] Liam Eagen, Dario Fiore, and Ariel Gabizon. cq: Cached quotients for fast lookups. Cryptology ePrint Archive, Report 2022/1763, 2022. URL: `https://eprint.iacr.org/2022/1763`.

[EHK$^+$13] Alex Escala, Gottfried Herold, Eike Kiltz, Carla Ràfols, and Jorge Villar. An algebraic framework for Diffie-Hellman assumptions. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 129–147. Springer, Berlin, Heidelberg, August 2013. `doi:10.1007/978-3-642-40084-1_8`.

[Fed22] Federal Office for Information Security Germany (BSI). Auditing machine learning algorithms - a white paper for public auditors. `https://www.hhi.fraunhofer.de/fileadmin/Departments/AI/TechnologiesAndSolutions/2022-05-23-whitepaper-tuev-bsi-hhi-towards-auditable-ai-systems.pdf`, 2022.

[FHK$^+$20] Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Prest, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. Falcon: Fast-Fourier Lattice-based Compact Signatures over NTRU, Specification v1.2. `https://falcon-sign.info/falcon.pdf`, January 2020. Version 1.2.

[FKL18] Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 33–62. Springer, Cham, August 2018. `doi:10.1007/978-3-319-96881-0_2`.

[FKNP24] Giacomo Fenzi, Christian Knabenhans, Ngoc Khanh Nguyen, and Duc Tu Pham. Lova: Lattice-based folding scheme from unstructured lattices. In *ASIACRYPT 2024, Part IV*, LNCS, pages 303–326. Springer, Singapore, December 2024. `doi:10.1007/978-981-96-0894-2_10`.

[FMNP16] Dario Fiore, Aikaterini Mitrokotsa, Luca Nizzardo, and Elena Pagnin. Multi-key homomorphic authenticators. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 499–530. Springer, Berlin, Heidelberg, December 2016. `doi:10.1007/978-3-662-53890-6_17`.

[FN16] Dario Fiore and Anca Nitulescu. On the (in)security of SNARKs in the presence of oracles. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part I*, volume 9985 of *LNCS*, pages 108–138. Springer, Berlin, Heidelberg, October / November 2016. `doi:10.1007/978-3-662-53641-4_5`.

[FP18] Dario Fiore and Elena Pagnin. Matrioska: A compiler for multi-key homomorphic signatures. In Dario Catalano and Roberto De Prisco, editors, *SCN*

18, volume 11035 of *LNCS*, pages 43–62. Springer, Cham, September 2018. doi:10.1007/978-3-319-98113-0_3.

[FQZ+21] Boyuan Feng, Lianke Qin, Zhenfei Zhang, Yufei Ding, and Shumo Chu. Zen: An optimizing compiler for verifiable, zero-knowledge neural network inferences. Cryptology ePrint Archive, Report 2021/087, 2021. https://ia.cr/2021/087.

[Fre12] David Mandell Freeman. Improved security for linearly homomorphic signatures: A generic framework. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *PKC 2012*, volume 7293 of *LNCS*, pages 697–714. Springer, Berlin, Heidelberg, May 2012. doi:10.1007/978-3-642-30057-8_41.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Berlin, Heidelberg, August 1987. doi:10.1007/3-540-47721-7_12.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 169–178. ACM Press, May / June 2009. doi:10.1145/1536414.1536440.

[GGJ+23] Sanjam Garg, Aarushi Goel, Somesh Jha, Saeed Mahloujifar, Mohammad Mahmoody, Guru-Vamsi Policharla, and Mingyuan Wang. Experimenting with zero-knowledge proofs of training. In Weizhi Meng, Christian Damsgaard Jensen, Cas Cremers, and Engin Kirda, editors, *ACM CCS 2023*, pages 1880–1894. ACM Press, November 2023. doi:10.1145/3576915.3623202.

[GGP10] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 465–482. Springer, Berlin, Heidelberg, August 2010. doi:10.1007/978-3-642-14623-7_25.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 626–645. Springer, Berlin, Heidelberg, May 2013. doi:10.1007/978-3-642-38348-9_37.

[GH98] Oded Goldreich and Johan Håstad. On the complexity of interactive proofs with bounded communication. *Inf. Process. Lett.*, 67(4):205–214, August 1998. doi:10.1016/S0020-0190(98)00116-1.

[GK96] Oded Goldreich and Hugo Krawczyk. On the composition of zero-knowledge proof systems. *SIAM Journal on Computing*, 25(1):169–192, 1996. arXiv:https://doi.org/10.1137/S0097539791220688, doi:10.1137/S0097539791220688.

[GKKR10] Rosario Gennaro, Jonathan Katz, Hugo Krawczyk, and Tal Rabin. Secure network coding over the integers. In Phong Q. Nguyen and David Pointcheval, editors, *PKC 2010*, volume 6056 of *LNCS*, pages 142–160. Springer, Berlin, Heidelberg, May 2010. `doi:10.1007/978-3-642-13013-7_9`.

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: interactive proofs for muggles. In Richard E. Ladner and Cynthia Dwork, editors, *40th ACM STOC*, pages 113–122. ACM Press, May 2008. `doi:10.1145/1374376.1374396`.

[GLS$^+$23] Alexander Golovnev, Jonathan Lee, Srinath T. V. Setty, Justin Thaler, and Riad S. Wahby. Brakedown: Linear-time and field-agnostic SNARKs for R1CS. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part II*, volume 14082 of *LNCS*, pages 193–226. Springer, Cham, August 2023. `doi:10.1007/978-3-031-38545-2_7`.

[GLWW24] Rachit Garg, George Lu, Brent Waters, and David J. Wu. Reducing the CRS size in registered ABE systems. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part III*, volume 14922 of *LNCS*, pages 143–177. Springer, Cham, August 2024. `doi:10.1007/978-3-031-68382-4_5`.

[GM18] Nicholas Genise and Daniele Micciancio. Faster Gaussian sampling for trapdoor lattices with arbitrary modulus. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 174–203. Springer, Cham, April / May 2018. `doi:10.1007/978-3-319-78381-9_7`.

[GMR85] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In *17th ACM STOC*, pages 291–304. ACM Press, May 1985. `doi:10.1145/22145.22178`.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987. `doi:10.1145/28395.28420`.

[GNS23] Chaya Ganesh, Anca Nitulescu, and Eduardo Soria-Vazquez. Rinocchio: SNARKs for ring arithmetic. *Journal of Cryptology*, 36(4):41, October 2023. `doi:10.1007/s00145-023-09481-3`.

[Goy24] Rishab Goyal. Mutable batch arguments and applications. Cryptology ePrint Archive, Report 2024/737, 2024. URL: `https://eprint.iacr.org/2024/737`.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In Richard E. Ladner and Cynthia Dwork, editors, *40th ACM STOC*, pages 197–206. ACM Press, May 2008. `doi:10.1145/1374376.1374407`.

[GR19] Alonso González and Carla Ràfols. Shorter pairing-based arguments under standard assumptions. In Steven D. Galbraith and Shiho Moriai, editors, *ASIACRYPT 2019, Part III*, volume 11923 of *LNCS*, pages 728–757. Springer, Cham, December 2019. doi:10.1007/978-3-030-34618-8_25.

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 305–326. Springer, Berlin, Heidelberg, May 2016. doi:10.1007/978-3-662-49896-5_11.

[GS08] Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 415–432. Springer, Berlin, Heidelberg, April 2008. doi:10.1007/978-3-540-78967-3_24.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 75–92. Springer, Berlin, Heidelberg, August 2013. doi:10.1007/978-3-642-40041-4_5.

[GSWW22] Rachit Garg, Kristin Sheridan, Brent Waters, and David J. Wu. Fully succinct batch arguments for NP from indistinguishability obfuscation. In Eike Kiltz and Vinod Vaikuntanathan, editors, *TCC 2022, Part I*, volume 13747 of *LNCS*, pages 526–555. Springer, Cham, November 2022. doi:10.1007/978-3-031-22318-1_19.

[GU24] Romain Gay and Bogdan Ursu. On instantiating unleveled fully-homomorphic signatures from falsifiable assumptions. In Qiang Tang and Vanessa Teague, editors, *PKC 2024, Part I*, volume 14601 of *LNCS*, pages 74–104. Springer, Cham, April 2024. doi:10.1007/978-3-031-57718-5_3.

[GVW02] Oded Goldreich, Salil Vadhan, and Avi Wigderson. On interactive proofs with a laconic prover. *Comput. Complex.*, 11(1/2):1–53, June 2002. doi:10.1007/s00037-002-0169-0.

[GVW15] Sergey Gorbunov, Vinod Vaikuntanathan, and Daniel Wichs. Leveled fully homomorphic signatures from standard lattices. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th ACM STOC*, pages 469–477. ACM Press, June 2015. doi:10.1145/2746539.2746576.

[GW11] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In Lance Fortnow and Salil P. Vadhan, editors, *43rd ACM STOC*, pages 99–108. ACM Press, June 2011. doi:10.1145/1993636.1993651.

[GW13] Rosario Gennaro and Daniel Wichs. Fully homomorphic message authenticators. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part II*, volume 8270 of *LNCS*, pages 301–320. Springer, Berlin, Heidelberg, December 2013. doi:10.1007/978-3-642-42045-0_16.

[GWC19] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over Lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953, 2019. URL: https://eprint.iacr.org/2019/953.

[GZ21] Alonso González and Alexandros Zacharakis. Fully-succinct publicly verifiable delegation from constant-size assumptions. In Kobbi Nissim and Brent Waters, editors, *TCC 2021, Part I*, volume 13042 of *LNCS*, pages 529–557. Springer, Cham, November 2021. doi:10.1007/978-3-030-90459-3_18.

[HKW15] Susan Hohenberger, Venkata Koppula, and Brent Waters. Universal signature aggregators. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 3–34. Springer, Berlin, Heidelberg, April 2015. doi:10.1007/978-3-662-46803-6_1.

[HR18] Justin Holmgren and Ron Rothblum. Delegating computations with (almost) minimal time and space overhead. In Mikkel Thorup, editor, *59th FOCS*, pages 124–135. IEEE Computer Society Press, October 2018. doi:10.1109/FOCS.2018.00021.

[HW15] Pavel Hubacek and Daniel Wichs. On the communication complexity of secure function evaluation with long output. In Tim Roughgarden, editor, *ITCS 2015*, pages 163–172. ACM, January 2015. doi:10.1145/2688073.2688105.

[Imp95] Russell Impagliazzo. A personal view of average-case complexity. In *Proceedings of Structure in Complexity Theory. Tenth Annual IEEE Conference*, pages 134–147, 1995. doi:10.1109/SCT.1995.514853.

[JJ21] Abhishek Jain and Zhengzhong Jin. Non-interactive zero knowledge from subexponential DDH. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 3–32. Springer, Cham, October 2021. doi:10.1007/978-3-030-77870-5_1.

[JKC$^+$18] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2704–2713, 2018.

[JKKZ21] Ruta Jawale, Yael Tauman Kalai, Dakshita Khurana, and Rachel Yun Zhang. SNARGs for bounded depth computations and PPAD hardness from subexponential LWE. In Samir Khuller and Virginia Vassilevska Williams, editors, *53rd ACM STOC*, pages 708–721. ACM Press, June 2021. `doi:10.1145/3406325.3451055`.

[JKLM25] Zhengzhong Jin, Yael Tauman Kalai, Alex Lombardi, and Surya Mathialagan. Universal snargs for np from proofs of correctness. In *Proceedings of the 57th Annual ACM Symposium on Theory of Computing*, STOC '25, page 933–943, New York, NY, USA, 2025. Association for Computing Machinery. `doi:10.1145/3717823.3718104`.

[JMSW02] Robert Johnson, David Molnar, Dawn Xiaodong Song, and David Wagner. Homomorphic signature schemes. In Bart Preneel, editor, *CT-RSA 2002*, volume 2271 of *LNCS*, pages 244–262. Springer, Berlin, Heidelberg, February 2002. `doi:10.1007/3-540-45760-7_17`.

[KHSS22] Daniel Kang, Tatsunori Hashimoto, Ion Stoica, and Yi Sun. ZK-IMG: Attested Images via Zero-Knowledge Proofs to Fight Disinformation, 2022. `arXiv:2211.04775`.

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *24th ACM STOC*, pages 723–732. ACM Press, May 1992. `doi:10.1145/129712.129782`.

[KLNO24] Michael Klooß, Russell W. F. Lai, Ngoc Khanh Nguyen, and Michal Osadnik. RoK, paper, SISsors toolkit for lattice-based succinct arguments - (extended abstract). In *ASIACRYPT 2024, Part V*, LNCS, pages 203–235. Springer, Singapore, December 2024. `doi:10.1007/978-981-96-0935-2_7`.

[KLVW23] Yael Kalai, Alex Lombardi, Vinod Vaikuntanathan, and Daniel Wichs. Boosting batch arguments and RAM delegation. In Barna Saha and Rocco A. Servedio, editors, *55th ACM STOC*, pages 1545–1552. ACM Press, June 2023. `doi:10.1145/3564246.3585200`.

[KNYY19] Shuichi Katsumata, Ryo Nishimaki, Shota Yamada, and Takashi Yamakawa. Designated verifier/prover and preprocessing NIZKs from Diffie-Hellman assumptions. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 622–651. Springer, Cham, May 2019. `doi:10.1007/978-3-030-17656-3_22`.

[KP23] Abhiram Kothapalli and Bryan Parno. Algebraic reductions of knowledge. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part IV*, volume 14084 of *LNCS*, pages 669–701. Springer, Cham, August 2023. `doi:10.1007/978-3-031-38551-3_21`.

[KPY19] Yael Tauman Kalai, Omer Paneth, and Lisa Yang. How to delegate computations publicly. In Moses Charikar and Edith Cohen, editors, *51st ACM STOC*, pages 1115–1124. ACM Press, June 2019. `doi:10.1145/3313276.3316411`.

[KRS25] Dmitry Khovratovich, Ron D. Rothblum, and Lev Soukhanov. How to prove false statements: Practical attacks on fiat-shamir. Cryptology ePrint Archive, Report 2025/118, 2025. URL: `https://eprint.iacr.org/2025/118`.

[KST22] Abhiram Kothapalli, Srinath Setty, and Ioanna Tzialla. Nova: Recursive zero-knowledge arguments from folding schemes. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part IV*, volume 13510 of *LNCS*, pages 359–388. Springer, Cham, August 2022. `doi:10.1007/978-3-031-15985-5_13`.

[KVZ21] Yael Tauman Kalai, Vinod Vaikuntanathan, and Rachel Yun Zhang. Somewhere statistical soundness, post-quantum security, and SNARGs. In Kobbi Nissim and Brent Waters, editors, *TCC 2021, Part I*, volume 13042 of *LNCS*, pages 330–368. Springer, Cham, November 2021. `doi:10.1007/978-3-030-90459-3_12`.

[KZG10] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 177–194. Springer, Berlin, Heidelberg, December 2010. `doi:10.1007/978-3-642-17373-8_11`.

[Lag25] Lagrange Labs. Lagrange's DeepProve: AI You Can Prove. `https://www.lagrange.dev/blog/deepprove-zkml`, May 2025. Accessed: 2025-06-24.

[LFKN92] Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *Journal of the ACM*, 39(4):859–868, 1992.

[Lip13] Helger Lipmaa. Succinct non-interactive zero knowledge arguments from span programs and linear error-correcting codes. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part I*, volume 8269 of *LNCS*, pages 41–60. Springer, Berlin, Heidelberg, December 2013. `doi:10.1007/978-3-642-42033-7_3`.

[LKKO20] Seunghwa Lee, Hankyung Ko, Jihye Kim, and Hyunok Oh. vCNN: Verifiable convolutional neural network. Cryptology ePrint Archive, Report 2020/584, 2020. URL: `https://eprint.iacr.org/2020/584`.

[LM19] Russell W. F. Lai and Giulio Malavolta. Subvector commitments with application to succinct arguments. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part I*, volume 11692 of *LNCS*, pages 530–560. Springer, Cham, August 2019. `doi:10.1007/978-3-030-26948-7_19`.

[LP20] Helger Lipmaa and Kateryna Pavlyk. Succinct functional commitment for a large class of arithmetic circuits. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part III*, volume 12493 of *LNCS*, pages 686–716. Springer, Cham, December 2020. `doi:10.1007/978-3-030-64840-4_23`.

[LPJY13] Benoît Libert, Thomas Peters, Marc Joye, and Moti Yung. Linearly homomorphic structure-preserving signatures and their applications. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 289–307. Springer, Berlin, Heidelberg, August 2013. doi:10.1007/978-3-642-40084-1_17.

[LRY16] Benoît Libert, Somindu C. Ramanna, and Moti Yung. Functional commitment schemes: From polynomial commitments to pairing-based accumulators from simple assumptions. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *ICALP 2016*, volume 55 of *LIPIcs*, pages 30:1–30:14. Schloss Dagstuhl, July 2016. doi:10.4230/LIPIcs.ICALP.2016.30.

[LTWC18] Russell W. F. Lai, Raymond K. H. Tai, Harry W. H. Wong, and Sherman S. M. Chow. Multi-key homomorphic signatures unforgeable under insider corruption. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part II*, volume 11273 of *LNCS*, pages 465–492. Springer, Cham, December 2018. doi:10.1007/978-3-030-03329-3_16.

[LXZ21] Tianyi Liu, Xiang Xie, and Yupeng Zhang. zkCNN: Zero knowledge proofs for convolutional neural network predictions and accuracy. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 2968–2985. ACM Press, November 2021. doi:10.1145/3460120.3485379.

[LY10] Benoît Libert and Moti Yung. Concise mercurial vector commitments and independent zero-knowledge sets with short proofs. In Daniele Micciancio, editor, *TCC 2010*, volume 5978 of *LNCS*, pages 499–517. Springer, Berlin, Heidelberg, February 2010. doi:10.1007/978-3-642-11799-2_30.

[Lyu09] Vadim Lyubashevsky. Fiat-Shamir with aborts: Applications to lattice and factoring-based signatures. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 598–616. Springer, Berlin, Heidelberg, December 2009. doi:10.1007/978-3-642-10366-7_35.

[Mau05] Ueli M. Maurer. Abstract models of computation in cryptography (invited paper). In Nigel P. Smart, editor, *10th IMA International Conference on Cryptography and Coding*, volume 3796 of *LNCS*, pages 1–12. Springer, Berlin, Heidelberg, December 2005. doi:10.1007/11586821_1.

[MBKM19] Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. Sonic: Zero-knowledge SNARKs from linear-size universal and updatable structured reference strings. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 2111–2128. ACM Press, November 2019. doi:10.1145/3319535.3339817.

[Mer79] Ralph Charles Merkle. *Secrecy, authentication, and public key systems.* PhD Thesis, Stanford University, 1979.

[Mic94] Silvio Micali. CS proofs (extended abstracts). In *35th FOCS*, pages 436–453. IEEE Computer Society Press, November 1994. doi:10.1109/SFCS.1994.365746.

[MP12] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 700–718. Springer, Berlin, Heidelberg, April 2012. doi:10.1007/978-3-642-29011-4_41.

[MPV24] Surya Mathialagan, Spencer Peters, and Vinod Vaikuntanathan. Adaptively sound zero-knowledge SNARKs for UP. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part X*, volume 14929 of *LNCS*, pages 38–71. Springer, Cham, August 2024. doi:10.1007/978-3-031-68403-6_2.

[MRV16] Paz Morillo, Carla Ràfols, and Jorge Luis Villar. The kernel matrix Diffie-Hellman assumption. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*, pages 729–758. Springer, Berlin, Heidelberg, December 2016. doi:10.1007/978-3-662-53887-6_27.

[MVVZ24] Pierpaolo Della Monica, Ivan Visconti, Andrea Vitaletti, and Marco Zecchini. Trust nobody: Privacy-preserving proofs for edited photos with your laptop. Cryptology ePrint Archive, Report 2024/1074, 2024. URL: https://eprint.iacr.org/2024/1074.

[Nao03] Moni Naor. On cryptographic assumptions and challenges (invited talk). In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 96–109. Springer, Berlin, Heidelberg, August 2003. doi:10.1007/978-3-540-45146-4_6.

[NRBB22] Valeria Nikolaenko, Sam Ragsdale, Joseph Bonneau, and Dan Boneh. Powers-of-tau to the people: Decentralizing setup ceremonies. Cryptology ePrint Archive, Report 2022/1592, 2022. URL: https://eprint.iacr.org/2022/1592.

[NT16] Assa Naveh and Eran Tromer. PhotoProof: Cryptographic image authentication for any set of permissible transformations. In *2016 IEEE Symposium on Security and Privacy*, pages 255–271. IEEE Computer Society Press, May 2016. doi:10.1109/SP.2016.23.

[NWW23] Shafik Nassar, Brent Waters, and David J. Wu. Monotone policy BARGs from BARGs and additively homomorphic encryption. Cryptology ePrint Archive, Report 2023/1967, 2023. URL: https://eprint.iacr.org/2023/1967.

[NWW24] Shafik Nassar, Brent Waters, and David J. Wu. Monotone policy BARGs from BARGs and additively homomorphic encryption. In *TCC 2024, Part II*, LNCS, pages 399–430. Springer, Cham, November 2024. doi:10.1007/978-3-031-78017-2_14.

[NWW25] Shafik Nassar, Brent Waters, and David J. Wu. Monotone-policy BARGs and more from BARGs and quadratic residuosity. LNCS, pages 283–313. Springer, Cham, May 2025. doi:10.1007/978-3-031-91829-2_9.

[OPWW15] Tatsuaki Okamoto, Krzysztof Pietrzak, Brent Waters, and Daniel Wichs. New realizations of somewhere statistically binding hashing and positional accumulators. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part I*, volume 9452 of *LNCS*, pages 121–145. Springer, Berlin, Heidelberg, November / December 2015. doi:10.1007/978-3-662-48797-6_6.

[PHGR13] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *2013 IEEE Symposium on Security and Privacy*, pages 238–252. IEEE Computer Society Press, May 2013. doi:10.1109/SP.2013.47.

[PK22] Jim Posen and Assimakis A. Kattis. Caulk+: Table-independent lookup arguments. Cryptology ePrint Archive, Report 2022/957, 2022. URL: https://eprint.iacr.org/2022/957.

[Poe17] Andrew Poelstra. Scriptless scripts, 2017. https://download.wpsoftware.net/bitcoin/wizardry/mw-slides/2017-03-mit-bitcoin-expo/slides.pdf.

[PP22] Omer Paneth and Rafael Pass. Incrementally verifiable computation via rate-1 batch arguments. In *63rd FOCS*, pages 1045–1056. IEEE Computer Society Press, October / November 2022. doi:10.1109/FOCS54457.2022.00102.

[PP24] Christodoulos Pappas and Dimitrios Papadopoulos. Sparrow: Space-efficient zkSNARK for data-parallel circuits and applications to zero-knowledge decision trees. In *ACM CCS 2024*, pages 3110–3124. ACM Press, November 2024. doi:10.1145/3658644.3690318.

[PPS21] Chris Peikert, Zachary Pepin, and Chad Sharp. Vector and functional commitments from lattices. In Kobbi Nissim and Brent Waters, editors, *TCC 2021, Part III*, volume 13044 of *LNCS*, pages 480–511. Springer, Cham, November 2021. doi:10.1007/978-3-030-90456-2_16.

[PS19] Chris Peikert and Sina Shiehian. Noninteractive zero knowledge for NP from (plain) learning with errors. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part I*, volume 11692 of *LNCS*, pages 89–114. Springer, Cham, August 2019. doi:10.1007/978-3-030-26948-7_4.

[Rot25] Lior Rotem. Goldreich-krawczyk revisited: A note on the zero knowledge of proofs of knowledge. *CiC*, 2(1):14, 2025. doi:10.62056/avr-11fgx.

[RS09] Markus Rückert and Dominique Schröder. Aggregate and verifiably encrypted signatures from multilinear maps without random oracles. In Jong Hyuk Park, Hsiao-Hwa Chen, Mohammed Atiquzzaman, Changhoon Lee, Tai-hoon Kim, and Sang-Soo Yeo, editors, *Advances in Information Security and Assurance*, pages 750–759, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[SBB19] Lucas Schabhüser, Denis Butin, and Johannes Buchmann. Context hiding multi-key linearly homomorphic authenticators. In Mitsuru Matsui, editor, *CT-RSA 2019*, volume 11405 of *LNCS*, pages 493–513. Springer, Cham, March 2019. doi:10.1007/978-3-030-12612-4_25.

[Sch80] J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, October 1980. doi:10.1145/322217.322225.

[sci17] scipr-lab. libsnark: a C++ library for zkSNARK proofs. https://github.com/scipr-lab/libsnark, 2017.

[Set20] Srinath Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 704–737. Springer, Cham, August 2020. doi:10.1007/978-3-030-56877-1_25.

[SFVA21] Somayeh Dolatnezhad Samarin, Dario Fiore, Daniele Venturi, and Morteza Amini. A compiler for multi-key homomorphic signatures for turing machines. *Theor. Comput. Sci.*, 889:145–170, 2021. doi:10.1016/j.tcs.2021.08.002.

[Sha90] Adi Shamir. IP=PSPACE. In *31st FOCS*, pages 11–15. IEEE Computer Society Press, October 1990. doi:10.1109/FSCS.1990.89519.

[Sho97] Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 256–266. Springer, Berlin, Heidelberg, May 1997. doi:10.1007/3-540-69053-0_18.

[SLZ24] Haochen Sun, Jason Li, and Hongyang Zhang. zkLLM: Zero knowledge proofs for large language models. In *ACM CCS 2024*, pages 4405–4419. ACM Press, November 2024. doi:10.1145/3658644.3670334.

[Sor22] Eduardo Soria-Vazquez. Doubly efficient interactive proofs over infinite and non-commutative rings. In Eike Kiltz and Vinod Vaikuntanathan, editors, *TCC 2022, Part I*, volume 13747 of *LNCS*, pages 497–525. Springer, Cham, November 2022. doi:10.1007/978-3-031-22318-1_18.

[SW05] Amit Sahai and Brent R. Waters. Fuzzy identity-based encryption. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 457–473. Springer, Berlin, Heidelberg, May 2005. doi:10.1007/11426639_27.

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In David B. Shmoys, editor, *46th ACM STOC*, pages 475–484. ACM Press, May / June 2014. `doi:10.1145/2591796.2591825`.

[Tha13] Justin Thaler. Time-optimal interactive proofs for circuit evaluation. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 71–89. Springer, Berlin, Heidelberg, August 2013. `doi:10.1007/978-3-642-40084-1_5`.

[the20] the Supreme Audit Institutions of Finland, Germany, the Netherlands, Norway and the UK. Towards auditable ai systems - from principles to practice. `https://www.auditingalgorithms.net/`, 2020.

[Uni] Stanford University. Cs231: Convolutional neural networks for pattern recognition. `https://cs231n.github.io/convolutional-networks/#convert`.

[Val08] Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In Ran Canetti, editor, *TCC 2008*, volume 4948 of *LNCS*, pages 1–18. Springer, Berlin, Heidelberg, March 2008. `doi:10.1007/978-3-540-78524-8_1`.

[VSBW13] Victor Vu, Srinath T. V. Setty, Andrew J. Blumberg, and Michael Walfish. A hybrid architecture for interactive verifiable computation. In *2013 IEEE Symposium on Security and Privacy*, pages 223–237. IEEE Computer Society Press, May 2013. `doi:10.1109/SP.2013.48`.

[Wee05] Hoeteck Wee. On round-efficient argument systems. In Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *ICALP 2005*, volume 3580 of *LNCS*, pages 140–152. Springer, Berlin, Heidelberg, July 2005. `doi:10.1007/11523468_12`.

[Wee25] Hoeteck Wee. Functional Commitments and SNARGs for P/poly from SIS. In *CRYPTO 2025*. Springer-Verlag, 2025.

[WJB$^+$17] Riad S. Wahby, Ye Ji, Andrew J. Blumberg, abhi shelat, Justin Thaler, Michael Walfish, and Thomas Wies. Full accounting for verifiable outsourcing. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 2071–2086. ACM Press, October / November 2017. `doi:10.1145/3133956.3133984`.

[WTs$^+$18] Riad S. Wahby, Ioanna Tzialla, abhi shelat, Justin Thaler, and Michael Walfish. Doubly-efficient zkSNARKs without trusted setup. In *2018 IEEE Symposium on Security and Privacy*, pages 926–943. IEEE Computer Society Press, May 2018. `doi:10.1109/SP.2018.00060`.

[WW22] Brent Waters and David J. Wu. Batch arguments for NP and more from standard bilinear group assumptions. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part II*, volume 13508 of *LNCS*, pages 433–463. Springer, Cham, August 2022. `doi:10.1007/978-3-031-15979-4_15`.

[WW23a] Hoeteck Wee and David J. Wu. Lattice-based functional commitments: Fast verification and cryptanalysis. In Jian Guo and Ron Steinfeld, editors, *ASIACRYPT 2023, Part V*, volume 14442 of *LNCS*, pages 201–235. Springer, Singapore, December 2023. `doi:10.1007/978-981-99-8733-7_7`.

[WW23b] Hoeteck Wee and David J. Wu. Succinct vector, polynomial, and functional commitments from lattices. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part III*, volume 14006 of *LNCS*, pages 385–416. Springer, Cham, April 2023. `doi:10.1007/978-3-031-30620-4_13`.

[WW24a] Brent Waters and David J. Wu. Adaptively-sound succinct arguments for NP from indistinguishability obfuscation. In Bojan Mohar, Igor Shinkar, and Ryan O'Donnell, editors, *56th ACM STOC*, pages 387–398. ACM Press, June 2024. `doi:10.1145/3618260.3649671`.

[WW24b] Hoeteck Wee and David J. Wu. Succinct functional commitments for circuits from k-Lin. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part II*, volume 14652 of *LNCS*, pages 280–310. Springer, Cham, May 2024. `doi:10.1007/978-3-031-58723-8_10`.

[XZS22] Tiancheng Xie, Yupeng Zhang, and Dawn Song. Orion: Zero knowledge proof with linear prover time. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part IV*, volume 13510 of *LNCS*, pages 299–328. Springer, Cham, August 2022. `doi:10.1007/978-3-031-15985-5_11`.

[XZZ$^+$19] Tiancheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. Libra: Succinct zero-knowledge proofs with optimal prover computation. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 733–764. Springer, Cham, August 2019. `doi:10.1007/978-3-030-26954-8_24`.

[Yao82] Andrew Chi-Chih Yao. Theory and applications of trapdoor functions (extended abstract). In *23rd FOCS*, pages 80–91. IEEE Computer Society Press, November 1982. `doi:10.1109/SFCS.1982.45`.

[ZBK$^+$22] Arantxa Zapico, Vitalik Buterin, Dmitry Khovratovich, Mary Maller, Anca Nitulescu, and Mark Simkin. Caulk: Lookup arguments in sublinear time. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 3121–3134. ACM Press, November 2022. `doi:10.1145/3548606.3560646`.

[ZCa22] ZCash. The halo2 zero-knowledge proving system. https://zcash.github.io/halo2/, 2022.

[ZGK^{+}17] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. vSQL: Verifying arbitrary SQL queries over dynamic outsourced databases. In *2017 IEEE Symposium on Security and Privacy*, pages 863–880. IEEE Computer Society Press, May 2017. doi:10.1109/SP.2017.43.

[ZGK^{+}18] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. vRAM: Faster verifiable RAM with program-independent preprocessing. In *2018 IEEE Symposium on Security and Privacy*, pages 908–925. IEEE Computer Society Press, May 2018. doi:10.1109/SP.2018.00013.

[ZGK^{+}22] Arantxa Zapico, Ariel Gabizon, Dmitry Khovratovich, Mary Maller, and Carla Ràfols. Baloo: Nearly optimal lookup arguments. Cryptology ePrint Archive, Report 2022/1565, 2022. URL: https://eprint.iacr.org/2022/1565.

[Zip79] Richard Zippel. Probabilistic algorithms for sparse polynomials. In Edward W. Ng, editor, *Symbolic and Algebraic Computation*, pages 216–226, Berlin, Heidelberg, 1979. Springer Berlin Heidelberg.

[ZLW^{+}21] Jiaheng Zhang, Tianyi Liu, Weijie Wang, Yinuo Zhang, Dawn Song, Xiang Xie, and Yupeng Zhang. Doubly efficient interactive proofs for general arithmetic circuits with linear prover time. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 159–177. ACM Press, November 2021. doi:10.1145/3460120.3484767.

[ZXZS20] Jiaheng Zhang, Tiancheng Xie, Yupeng Zhang, and Dawn Song. Transparent polynomial delegation and its applications to zero knowledge proof. In *2020 IEEE Symposium on Security and Privacy*, pages 859–876. IEEE Computer Society Press, May 2020. doi:10.1109/SP40000.2020.00052.