Accelerating Post-Quantum Secure zkSNARKs for Privacy-Preserving Frameworks

by

Mohammadtaghi Badakhshan

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2025

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner:	Liqun Chen Professor, Dept. of Computer Science and Electronic Engineering, University of Surrey
Supervisor:	Guang Gong Professor, Dept. of Electrical and Computer Engineering, University of Waterloo
Internal Member:	Anwar Hasan Professor, Dept. of Electrical and Computer Engineering, University of Waterloo
Internal Member:	Mahesh Tripunitara Professor, Dept. of Electrical and Computer Engineering, University of Waterloo
Internal-External Member:	Douglas Stebila Professor, Dept. of Combinatorics and Optimization, University of Waterloo

Author's Declaration

Statement of Contributions

I am the sole author of Chapters 1, 6, 7, and 8. Chapters 2, 3, 4, 5, 6, 7 include material from both published and unpublished papers that I have authored or co-authored. Specific contributions and collaborations are clearly indicated at the beginning of each chapter.

Abstract

Zero-Knowledge Succinct Non-Interactive Arguments of Knowledge (zkSNARKs) are gaining widespread adoption across various applications. Despite significant progress in developing post-quantum secure zkSNARKs, current schemes still encounter notable challenges, particularly regarding computational complexity. This thesis proposes utilizing the Cantor special basis to enhance post-quantum secure zkSNARKs operating over binary extension fields. By adopting this basis, we optimize the additive fast Fourier transform (FFT) algorithm employed in Aurora, a plausible post-quantum secure zkSNARK, replacing the previously utilized Gao-Mateer FFT with the more efficient Cantor FFT. Our implementation demonstrates substantial reductions in computation time for Aurora, indicating potential performance improvements for other zkSNARK systems reliant on additive FFTs. We offer a thorough theoretical analysis of the computational complexity of the Cantor FFT algorithm, providing precise counts of required additions, multiplications, and precomputation overhead. Additionally, we examine the FFT call complexity within the Rank-1 Constraint System encoding specifically for Aurora. Furthermore, this thesis includes an extensive analysis of the algorithms within Polaris, a plausible post-quantum zkSNARK protocol, by systematically decomposing its components for detailed evaluation. Recognizing the critical need for efficient real-world implementations, we propose a concrete GKR arithmetic circuit for integration into Polaris. We also enhance the efficiency of the FRI protocol within Polaris by eliminating costly field inversion operations. Finally, as an illustrative example of a privacy-preserving protocol utilizing zkSNARKs, we propose a novel framework called Anonymous Authentication Token (AAT). This framework supports the unlinkable transfer of token ownership (AAT Ownership Transfer or AATOT), including the merging and dividing of tokens in an unlinkable manner. Our construction leverages zkSNARK protocols to ensure robust anonymity, unlinkability, and authentication. Building upon this foundation, we introduce Zupply, a decentralized system designed to maintain directed acyclic graphs (DAGs) of authentic data records. Zupply operates atop a permissionless blockchain equipped with smart contracts, offering a trustless environment that preserves participant anonymity and unlinkability. Simultaneously, it ensures the integrity and authenticity of data records across the entire supply chain ecosystem. We design and implement optimized arithmetic circuits within the Zupply framework to minimize proof sizes and verification costs. We concurrently explore post-quantum secure solutions to future-proof the framework against quantum computing advancements. Our implementation of Zupply utilizes C++ and Solidity, leveraging two distinct zkSNARK protocols: Groth16 and Aurora. The Groth16 zkSNARK, while vulnerable to quantum attacks, provides computational efficiency and reduced operational costs, demonstrating Zupply's practicality for real-world decentralized supply chain management (SCM) systems.

Conversely, the Aurora zkSNARK is designed to be plausibly secure against quantum-capable adversaries. We conducted a comparative analysis of computation efficiency and proof sizes between these two zkSNARK variants,

Acknowledgements

I would like to sincerely thank my supervisor, Professor Guang Gong, for her unwavering support and guidance throughout my PhD. Her expertise in cryptography and communications introduced me to the fascinating world of zero-knowledge proofs, an area I knew very little about before starting this journey. She gave me the freedom to explore my own research directions, while always being there to offer advice and encouragement during difficult times. Her kindness and support went beyond academics and created an environment where I could grow both professionally and personally. It has been an honor to be her student, and I will always be grateful for everything I have learned from her.

I am truly grateful to Professor Liqun Chen, Professor Anwar Hasan, Professor Mahesh Tripunitara, and Professor Douglas Stebila for serving on my thesis committee. Their thoughtful feedback and valuable suggestions helped improve the quality of this work. I deeply appreciate the time and effort they dedicated to reviewing my thesis and supporting my progress.

I would like to thank my parents for their unconditional love, support, and belief in me throughout every step of my life. Their sacrifices and values have shaped who I am today. Even from afar, they have always been with me, giving me strength and reminding me to keep going with honesty and perseverance. I dedicate this achievement to them with love and deep appreciation.

Last but not the least, I am incredibly grateful to my girlfriend, Maryam Mottaghi, for her endless support, patience, and love during this journey. She stood by me through all the ups and downs, encouraged me when I struggled, and believed in me when I needed it most. Her presence has been a source of peace and motivation, and sharing this chapter of life with her has been a true gift.

Dedication

To my parents.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

AAT Anonymous Authentication Token 3–5, 90–97, 99–108, 110–118, 121, 123, 124, 139

AATOT Anonymous Authentication Token Ownership Transfer 3, 4, 90–93, 99, 108, 118, 135, 139

AIR Algebraic Intermediate Representation 52

BLE Bluetooth low energy 44

BP Blockchain Platform 108

CID Content Identifier 98

CRH Collision-Resistant Hash 19, 21

CRS Common Reference String 20, 21

DAG Directed Acyclic Graph 3, 4, 45, 89–94, 103, 111, 113, 115, 116, 118, 120, 139, 140

dApp Decentralized Application 42

DCS Decentralized Cloud Storag 92, 94, 98, 125, 141

DFT Discrete Fourier Transform 10

ECRH Extractable Collision-Resistant Hash 19

EoL End of Life 96, 97, 103–106, 111, 112

EVM Ethereum Virtual Machine 124

FDA Food and Drug Administration 47

FFT Fast Fourier Transform 1–4, 10, 36, 38, 40, 41, 51–54, 134, 138–140

FRI Fast Reed-Solomon Interactive Oracle Proof of Proximity 3, 4, 21, 23–25, 128, 139

G3C Graph 3-Colorability 16

IFFT Inverse Fast Fourier Transform 4

IOP Interactive Oracle Proof 21, 24, 128

IoT Internet of Things 44, 45, 47

IP Interactive Proof System 12–18, 24

IPFS InterPlanetary File System 46, 47, 90, 125, 141

LDT Low Degree Testing 25

MSLAG Multilayered Linkable Spontaneous Anonymous Group 135

MSME Micro, Small and Medium Enterprise 44–46, 90

MSP Membership Service Provider 46

NP Nondeterministic Polynomial-time 13, 100–102, 120, 121, 127–129

PI Proof of Inclusion 97, 100

PPT Probabilistic Polynomial-time 12–17, 20, 110

R1CS Rank-1 Constraint Systems 3, 8, 9, 24, 25, 120, 121, 123, 126–132, 134, 136, 139, 140, 165

RFID Radio-Frequency Identification 44, 47

RS Reed-Solomon 3, 10, 24, 25, 140

SCM Supply Chain Management 3, 43–48, 89–91, 93, 115, 118, 121, 129, 138, 140

SMPC Secure Multi-Party Computation 126

SNARG Succinct Non-Interactive Argument 19

SNARK Succinct Non-Interactive Argument of Knowledge 19

UTXO Unspent Transaction Output 41

VCS Version Control System 89, 90, 140

ZKP Zero-Knowledge Proof 3–5, 8, 12, 14, 16, 41, 52, 90–93, 97, 98, 100–107, 114, 116, 118, 120–124, 129, 138

zkSNARK Zero-Knowledge Succinct Non-Interactive Argument of Knowledge 1–5, 19–21, 24, 42, 51–53, 82, 103, 110, 120, 126, 128–132, 134, 138–140

Chapter 1

Introduction

Zero-knowledge succinct non-interactive argument of knowledge (zkSNARK) protocols are cryptographic schemes involving a prover and a verifier. The prover generates a publicly verifiable and succinct proof that demonstrates knowledge of a witness vector (secret inputs) satisfying a given constraint system ($\mathcal{NP}$ statement). The proof does not reveal any information about the witness itself. These protocols have a wide range of applications, such as post-quantum secure digital signature algorithms [24, 59, 62, 111], privacy-preserving applications over blockchains [31, 117, 163, 48, 180], blockchain scalability solutions using rollups [14, 57, 170, 146, 187, 162], data availability proofs for lightweight clients in Ethereum [99], the proof of replication in Filecoin [40], verifying the computation of neural networks [166, 60, 178], and verifying the authenticity of edited images [72]. The efficiency of the prover and verifier algorithms, along with the proof size of the zkSNARK protocol, are crucial factors influencing the cost-effectiveness and overall practicality of the aforementioned applications. Additionally, two other important factors are: first, whether the zkSNARK protocol relies on a trusted setup or employs a transparent setup; and second, whether it offers plausible post-quantum security against a malicious prover with quantum capabilities.

With the increasing adoption of zkSNARKs, optimizing or accelerating the implementation of algorithms within these protocols has become an active and extensively studied research area [29, 28, 70, 128, 107, 105]. Fast Fourier Transform (FFT) algorithms are central to the performance of zkSNARK constructions, as both post-quantum vulnerable [95, 65, 53, 82, 102] and post-quantum secure [9, 27, 35, 66, 81] variants depend on polynomial commitment schemes. These commitments are applied to polynomials interpolated from the secret and public values in the constraint system.

The FFT algorithm used in a zkSNARK protocol can be either multiplicative [67, 147, 46] or additive [56, 84, 83, 123], depending on the structure of the polynomial evaluation domains. These domains are typically either multiplicative cosets in a prime field or additive cosets (affine subspaces) in a binary extension field. The choice of domain is influenced by the underlying polynomial commitment scheme employed in the protocol, which determines whether the values in the constraint system (i.e., polynomial evaluations) lie in a prime field or a binary extension field. For instance, zkSNARKs based on pairing-based polynomial commitments (e.g.,[110]) and inner-product argument schemes (e.g.,[53]) typically operate over prime fields. In contrast, zkSNARKs utilizing Merkle hash tree [134] based commitment schemes (e.g., [26]) can operate over either binary extension fields or prime fields. Merkle hash tree based schemes do not rely on cryptographic hardness assumptions that are vulnerable to quantum attacks, this property makes them attractive for constructing post-quantum secure zkSNARKs [9, 27, 35, 66, 185].

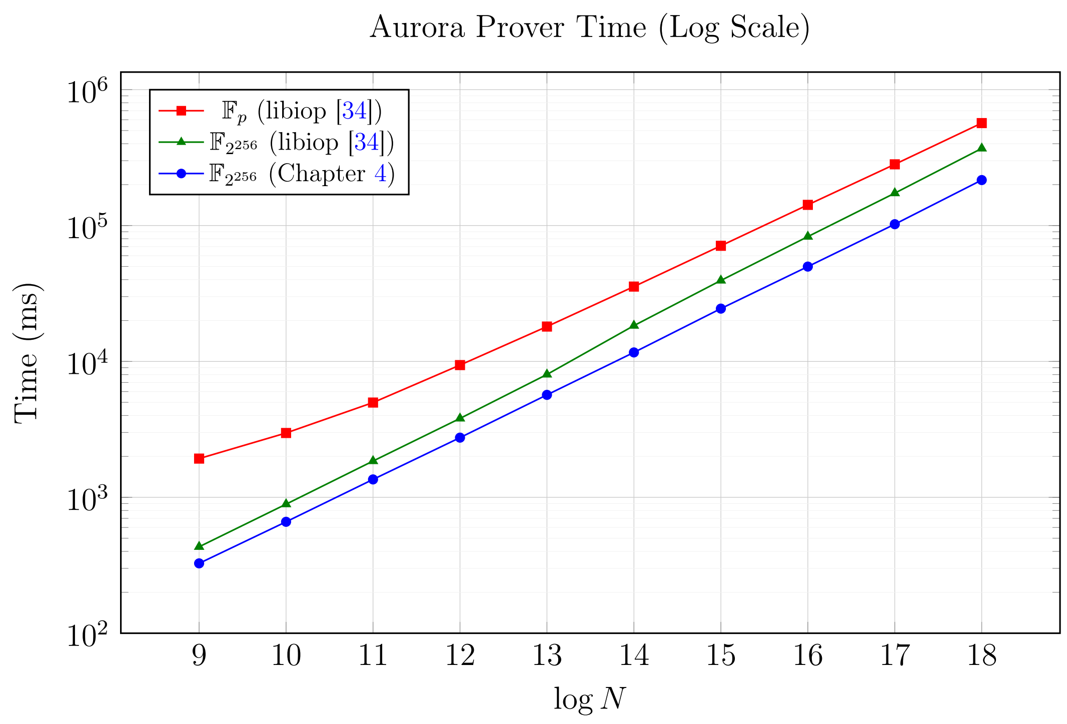


Figure 1.1: Comparison of Aurora zkSNARK [35] performance over prime fields and binary extension fields on an AMD Ryzen 9 9950X @ 5.7 GHz. The prime field is defined by a 255-bit prime p, corresponding to the scalar field of the BLS12-381 curve [22]. N denotes the number of constraints.

Additive FFTs offer distinct advantages in scenarios where multiplicative FFTs are constrained. Specifically, multiplicative FFTs require the evaluation domain of size $n = 2^m$ to contain an n-th root of unity, which is not always available. In such cases, zkSNARK protocols over prime fields must resort to slower alternatives, such as the method proposed in [47], for polynomial interpolation and evaluation, instead of using efficient FFT-based techniques. Moreover, finite field arithmetic operations (addition/subtraction, multiplication, squaring, and inversion) tend to be faster, more power-efficient, and more area-efficient in hardware when performed over binary extension fields as compared to prime fields [179, 70]. Figure 1.1 compares the runtime of the Aurora zkSNARK[35] prover algorithm implemented over two different fields: the prime field corresponding to the scalar field of the BLS12-381 curve [22], and the binary extension field $\mathbb{F}_{2^{256}}$, as implemented in libiop [34]. The figure also illustrates the performance gains achieved through the acceleration techniques for additive FFT algorithms presented in Chapter 4.

While FFT algorithms play a crucial role in accelerating zkSNARKs and numerous other applications [44, 63, 121, 42, 41], the optimization of post-quantum secure zkSNARKs can be pursued through various approaches. The fast Reed-Solomon (RS) interactive oracle proof of proximity (FRI) serves as the foundation for many post-quantum secure zkSNARKs, including [27, 35, 66, 81]. In addition, the Polaris zkSNARK [81] leverages the GKR protocol [89] to offload certain computations to the prover. Chapter 5 proposes an optimization technique to accelerate the FRI protocol and introduces an efficient method for instantiating the arithmetic circuit used in Polaris's GKR protocol.

As previously mentioned, zkSNARK protocols are widely used in privacy-preserving applications and typically rely on the rank-1 constraint system (R1CS), which is well-suited for representing arithmetic circuits. This structure is particularly advantageous because it allows the prover to generate a zero-knowledge proof (ZKP) of knowledge of the preimage of a leaf node in a Merkle hash tree [134], where the tree is computed using the circuit representation of a hash function (e.g., SHA-256). Several applications, such as those presented in [31, 117, 180, 48, 164], leverage Merkle hash trees to preserve user privacy. Chapter 6 presents a novel anonymous authentication token (AAT) framework that supports unlinkable AAT ownership transfer (AATOT), including the ability to merge and divide AATs in an unlinkable manner. This construction utilizes a Merkle hash tree to store AATs and facilitates their transfer over a public blockchain. This approach enables the creation of off-chain, anonymously maintained, decentralized data records organized as a directed acyclic graph (DAG), with practical applications in supply chain management (SCM). Chapter 7 presents the implementation of the proposed framework using two zkSNARKs. The first is the Groth16 zkSNARK [95], which requires a one-time trusted setup and relies on cryptographic assumptions that are vulnerable to quantum attacks.

The second is the Aurora zkSNARK[35], which does not require any trusted setup and is plausibly secure against quantum adversaries.

In the remainder of this thesis, Chapter 2 provides the definitions and mathematical preliminaries essential to the presented work, Chapter 3 reviews the literature relevant to this research. The contributions of the thesis are presented in Chapters 4,5,6, and 7, and are summarized as follows:

1. Chapter 4 presents a comprehensive study and optimization of additive FFT algorithms, focusing on the Cantor [56] and Gao-Mateer [83] methods. In the Cantor algorithm, a novel theoretical analysis of vanishing polynomials is provided, where the number of terms is precisely determined based on Hamming-weight, allowing for accurate estimation of additions, improving upon previously reported upper bounds. The vanishing polynomials and multiplication factors are efficiently computed. Modular building blocks for the Cantor FFT are proposed and integrated into the Aurora zkSNARK [34], resulting in significant performance improvements over the Gao-Mateer algorithm. In the Gao-Mateer method, the Expand and Aggregate modules are analyzed, the Cantor special basis is incorporated, and two levels of precomputation are introduced to reduce computational overhead. The FFT/IFFT call complexity in Aurora is analyzed, with emphasis on how the choice of shift elements in affine subspaces reduces space complexity in the Cantor FFT. Optimized C++ implementations of all algorithms and precomputation techniques are provided, and a detailed comparative evaluation is performed.

2. Chapter 5 presents an instantiation of the FRI protocol, in which field inversion operations are eliminated in both the Commit and Query phases to achieve improved efficiency. Additionally, an instantiation of the GKR circuit tailored to the Polaris implementation is described. The circuit is designed as a satisfiability circuit to enable the verifiable computation of values essential to the Polaris protocol while minimizing the number of gates. This design reduces communication overhead as well as the complexities of the verifier and the prover.

3. In Chapter 6, a novel AAT scheme is designed using zero-knowledge proofs (ZKPs) on smart contract-enabled public blockchains. The proposed framework, Zupply, consists of a set of algorithms and protocols that enable on-chain unlinkable AAT ownership transfers (AATOT) and off-chain anonymous data authentication. Four main properties are satisfied: (1) Anonymity is preserved for data uploaders and AAT owners, except for AATs that initiate a DAG; (2) Unlinkability is ensured by concealing the link between entities collaborating to maintain the DAG of supply chain data records; (3) Integrity is guaranteed as each data record is authenticated, remains unaltered, and is verifiably

created by the authorized entity responsible for that stage; and (4) Trustlessness is achieved by eliminating the need for trusted parties during protocol execution, relying instead on public blockchains for decentralization. In addition, storage is separated from the blockchain while maintaining a concealed link to the AATs. This design enables entities to anonymously upload data. The off-chain anonymous authentication mechanism reduces blockchain storage costs, enhances anonymity, and allows entities to choose storage strategies that best align with their decentralization and cost requirements.

4. In Chapter 7, Four arithmetic circuits, Auth, Trans, Merge, and Div, are designed and employed in the Zupply framework to express the $\mathcal{NP}$ statements required for ZKPs. In the first implementation variant of the Zupply framework, Groth16 [95], a zkSNARK known for producing succinct proofs and enabling fast verification, is integrated. This protocol is based on bilinear groups and requires a trusted setup to generate the necessary cryptographic parameters. The implementation is evaluated over two elliptic curves: BN254 [23], which offers approximately 100-bit security [20], and BLS12-381 [22], designed for 128-bit security. In the second implementation variant, Aurora [35] is integrated into the Zupply framework to achieve a transparent setup and plausible post-quantum security against quantum-capable adversarial provers. The computational performance of this variant is compared with the Groth16-based implementation over the BN254 and BLS12-381 curves.

Finally, Chapter 8 concludes the thesis by summarizing its contributions and outlining directions for future work.

Chapter 2

Preliminaries

Declaration of Contributions

This chapter contains preliminary material from an unpublished work and the paper [19], both of which I co-authored. The preliminaries in those works were written in collaboration with all co-authors. In addition, this chapter includes content from [17], of which I am the sole author.

2.1 Algebraic Foundations

2.1.1 Finite Fields

Let $\mathbb{F}$ denote a finite field and $\mathbb{F}_q$ denote the finite field with q elements, where $q = p^k$ for a prime p and a positive integer k. The elements of a finite field with characteristic p can be represented as vectors over $\mathbb{F}_p$. In other words, there exists a vector space isomorphism from $\mathbb{F}_{p^k}$ to $\mathbb{F}_p^k$ defined by

$$x = (x_0\alpha_0 + x_1\alpha_1 + \cdots + x_{k-1}\alpha_{k-1}) \mapsto (x_0, x_1, \ldots, x_{k-1}),$$

where $\{\alpha_0, \alpha_1, \ldots, \alpha_{k-1}\}$ is a basis of $\mathbb{F}_{p^k}$ over $\mathbb{F}_p$ [156].

A polynomial of degree n over a finite field $\mathbb{F}_q$ is an expression in an indeterminate x of the form

$$f(x) = \sum_{i=0}^{n} c_i x^i,$$

where $c_i \in \mathbb{F}_q$ and $c_n \neq 0$. More precisely, $f(x)$ is referred to as a *univariate polynomial.* The set of polynomial over $\mathbb{F}_q$ in the variable x is denoted as $\mathbb{F}_q[x]$.

The finite extension $\mathbb{F}_{q^m}$ of the field $\mathbb{F}_q$ forms a vector space of dimension m over the field $\mathbb{F}_q$. Let $\mathbb{F}_2$ denote the binary field, the finite field $\mathbb{F}_{2^{256}}$ is defined as

$$\mathbb{F}_{2^{256}} := \mathbb{F}_2[X]/(X^{256} + X^{10} + X^5 + X^2 + 1), \tag{2.1}$$

and is utilized in our implementation in Chapter 4.

The trace function is a mapping from $\mathbb{F}_{q^m}$ to $\mathbb{F}_q$ which will turn out to be linear.

Definition 2.1. For an element $\beta \in \mathbb{F}_{q^m}$, the trace $\mathrm{Tr}_{\mathbb{F}_{q^m}/\mathbb{F}_q}(\beta)$ over $\mathbb{F}_q$ is defined by

$$\mathrm{Tr}_{\mathbb{F}_{q^m}/\mathbb{F}_q}(\beta) = \beta + \beta^q + \cdots + \beta^{q^{m-1}}.$$

The trace function satisfies the following properties, which are relevant to our discussions in this thesis:

(i) $\mathrm{Tr}_{\mathbb{F}_{q^m}/\mathbb{F}_q}(\beta + \gamma) = \mathrm{Tr}_{\mathbb{F}_{q^m}/\mathbb{F}_q}(\beta) + \mathrm{Tr}_{\mathbb{F}_{q^m}/\mathbb{F}_q}(\gamma)$ for all $\beta, \gamma \in \mathbb{F}_{q^m}$.

(ii) $\mathrm{Tr}_{\mathbb{F}_{q^m}/\mathbb{F}_q}(c\beta) = c\,\mathrm{Tr}_{\mathbb{F}_{q^m}/\mathbb{F}_q}(\beta)$ for all $\beta \in \mathbb{F}_{q^m}$ and $c \in \mathbb{F}_q$.

(iii) $\mathrm{Tr}_{\mathbb{F}_{q^m}/\mathbb{F}_q}$ takes on each value in $\mathbb{F}_q$ equally often, i.e., q^{m-1} times.

2.1.2 Affine Subspace

Let us define the subspace W_m of $\mathbb{F}_{2^k}$ as the linear combinations of $\{\beta_0, \beta_1, \ldots, \beta_{m-1}\}$. We order the elements of the subspace W_m by $\{\eta_0 = 0, \eta_1, \eta_2, \ldots, \eta_{2^m-1}\}$ where

$$\eta_j = \sum_{i=0}^{m-1} x_i\beta_i \quad \text{and} \quad j = \sum_{i=0}^{m-1} x_i 2^i, x_i \in \mathbb{F}_2.$$

Note that for any $0 \leq m < k$, we can decompose the elements of W_{m+1} into two disjoint sets: the subspace W_m and the *affine subspace* $\beta_m + W_m$, where $\beta_m + W_m$ is the set obtained by translating (or shifting) the subspace W_m by the vector β_m. More formally, this decomposition is given by $W_{m+1} = W_m \cup (\beta_m + W_m)$, where

$$\beta_m + W_m = \{\beta_m + w : w \in W_m\}.$$

Furthermore, if $m < k - 1$ and θ is any linear combination of $\{\beta_{m+1}, \beta_{m+2}, \ldots, \beta_{k-1}\}$, then we can decompose the elements of $\theta + W_{m+1}$ into two pairwise disjoint affine subspaces $\theta + W_m$ and $\beta_m + \theta + W_m$.

2.1.3 Rank-1 Constraint System

Zero-knowledge proof (ZKP) system toolchains, such as [95, 35, 66, 9] that generate arithmetic or boolean circuits from high-level programming languages, usually use rank-1 constraint systems (R1CSs), which is so straightforward that it can be used to create efficient argument systems. R1CS has also proven to be reliable in real-world applications, such as Zcash [31]. We provide the definition of R1CS in the following:

Definition 2.2 (rank-1 constraint system (R1CS))**.** Let d_1, d_2, and d_3 be positive integers where denote the number of constraints, the number of variables, and the number of public inputs respectively. The R1CS relation consists of matrices $\mathbf{A}, \mathbf{B}, \mathbf{C} \in \mathbb{F}^{d_1 \times (d_2+1)}$ and public inputs $\vec{v} \in \mathbb{F}^{d_3}$. A vector $\vec{w} \in \mathbb{F}^{d_2-d_3}$, denoting private (auxiliary) inputs, satisfies the system if $\mathbf{A}\vec{z} \circ \mathbf{B}\vec{z} = \mathbf{C}\vec{z}$, where $\vec{z} := (1, \vec{v}, \vec{w}) \in \mathbb{F}^{d_2+1}$ and "$\circ$" denotes the Hadamard product.

Let an R1CS instance be denoted as $(\mathbb{F}, d_1, d_2, d_3, \mathbf{A}, \mathbf{B}, \mathbf{C}, \vec{v})$, and accordingly, define the R1CS relation $\mathcal{R}_{\text{R1CS}}$ as

$$\mathcal{R}_{\text{R1CS}} = \{(\mathbb{F}, d_1, d_2, d_3, \mathbf{A}, \mathbf{B}, \mathbf{C}, \vec{v}, \vec{w}) \ \mid \mathbf{A}\vec{z} \circ \mathbf{B}\vec{z} = \mathbf{C}\vec{z}, \text{ where } \vec{z} := (1, \vec{v}, \vec{w}) \in \mathbb{F}^{d_2+1}\}.$$

In this thesis, we define a notation for the R1CS parameter as $(\mathbb{F}, d_1, d_2, d_3, \mathbf{A}, \mathbf{B}, \mathbf{C})$, where the public input $\vec{v}$ has not been set yet. Therefore, this does not constitute an R1CS instance. Instead, it only specifies the constraint system, which may be derived from an arithmetic or Boolean circuit. In the following, we will explain arithmetic and Boolean citcuits and how an they can be compiled to R1CS instances.

Arithmetic and Boolean Circuits

Any computer program can be compiled into either an arithmetic or a Boolean circuit [144]. As described by Gong [92], arithmetic circuits consist of `ADD` and `MULTIPLY` gates, each with a fan-in of two. For any gate g in a circuit C, let g_l, g_r, and g_o denote its left input, right input, and output, respectively. If g is an `ADD` gate, it satisfies the equation

$$g_l + g_r - g_o = 0.$$

If it is a `MULTIPLY` gate, then

$$g_l \times g_r - g_o = 0.$$

Similarly, Boolean circuits comprise `AND` and `XOR` gates. If g is an `AND` gate, it satisfies

$$g_l \wedge g_r \oplus g_o = 0,$$

and if it is an XOR gate, then

$$g_l \oplus g_r \oplus g_o = 0.$$

In the conversion of a circuit to an R1CS representation, only MULTIPLY gates in arithmetic circuits and AND gates in Boolean circuits are considered, as other gate types can be expressed in terms of these two. The arithmetic (or Boolean) circuit is then compiled into an R1CS instance, where d_1 denotes the number of MULTIPLY (AND) gates, and $\mathbf{A}\vec{z}$, $\mathbf{B}\vec{z}$, and $\mathbf{C}\vec{z}$ represent the left input, right input, and output of these gates, respectively. Appendix A, provides an exmaple of converting a Boolean circuit to an R1CS instance.

2.1.4 Vanishing Polynomial

Definition 2.3. The polynomial $\mathbb{Z}_{W_m}(x) = \prod_{a \in W_m}(x-a)$ which is a linearized polynomial given by $\mathbb{Z}_{W_m}(x) = \sum_{i=0}^{m} c_i x^{2^i}, c_i \in \mathbb{F}_{2^k}$. This polynomial is called the *vanishing polynomial* for the subspace W_m.

Observe that the vanishing polynomial of $\epsilon + W_m$ is given by

$$\prod_{a \in \epsilon + W_m} (x-a) = \prod_{a \in W_m} (x - a - \epsilon) = \mathbb{Z}_{W_m}(x - \epsilon).$$

Now, since $W_{m+1} = W_m \cup (\beta_m + W_m)$, we have

$$\mathbb{Z}_{W_{m+1}}(x) = (\mathbb{Z}_{W_m}(x))^2 - \mathbb{Z}_{W_m}(\beta_m) \cdot \mathbb{Z}_{W_m}(x).$$

2.1.5 Univariate Polynomials Vectorial Representation

For any univariate polynomial $f(x) = \sum_{i=0}^{n-1} c_i x^i, c_i \in \mathbb{F}$, where $\deg(f) < n = 2^m$, the coefficients are represented as a vector of n elements, ordered from the constant term to the highest degree term. Namely,

$$\mathbf{f} = (c_0, ..., c_{n-1}), c_i \in \mathbb{F}_{2^k}$$

represents the polynomial $f(x)$, where $\deg(f) < n$.

2.1.6 Additive Discrete Fourier Transform

Now we will discuss the evaluation of a univariate polynomial $f(x)$ over the subspace W_m.

Definition 2.4. The evaluation of $f(x)$ at the points $\eta_0, \eta_1, \ldots, \eta_{2^m-1}$ is given by

$$\hat{\mathbf{f}} = (f(\eta_0), f(\eta_1), \ldots, f(\eta_{2^m-1}))\,.$$

This set of evaluations is referred to as the additive discrete Fourier transform (DFT) of $f(x)$ over the subspace W_m. We sometimes refer to the vector $\hat{\mathbf{f}}$ as the *discrete Fourier transform of length* $n = 2^m$ for the function $f(x)$, denoted as $\mathrm{DFT}(f, W_m)$. The additive fast Fourier transform (FFT) is an efficient method for computing $\mathrm{DFT}(f, W_m)$, which we will denote as $\mathrm{FFT}(f, W_m)$.

2.1.7 Reed-Solomon Code

Reed-Solomon (RS) code is proposed in 1960 [148]. It is a block-based error-correcting code that has several applications in digital communications and storage. Let the evaluation domain L be a subset of $\mathbb{F}$. RS$[\mathbb{F}, L, \rho]$ is the set of all codewords $\hat{\mathbf{f}} : L \to \mathbb{F}$ defined as follows:

Definition 2.5 (Reed-Solomon code)**.** Let $L \subseteq \mathbb{F}$ and $\rho \in [0, 1]$ denote the codeword domain and the rate parameter respectively. RS$[\mathbb{F}, L, \rho]$ is the set of all codewords $\hat{\mathbf{f}} : L \to \mathbb{F}$ that are the evaluation of polynomials over L with degree $< \rho|L|$.

2.2 Cryptographic Primitives

Let $\boldsymbol{\lambda} > 0$ be an integer and represent an adjustable security parameter in our scheme. A negligible function $\mathsf{negl}(\boldsymbol{\lambda})$ is a negligible in $\boldsymbol{\lambda}$. Namely, for every polynomial p, there exists an $\boldsymbol{\lambda}_0$, such that for all $\boldsymbol{\lambda} > \boldsymbol{\lambda}_0$ it holds that $\mathsf{negl}(\boldsymbol{\lambda}) < \frac{1}{p(\boldsymbol{\lambda})}$. With the security parameter and the negligible function in place, we define the cryptographic primitives:

Definition 2.6 (Collision and Preimage Resistance Hash Function)**.** A hash function $\mathcal{H} : \{0, 1\}^* \to \{0, 1\}^{O(\boldsymbol{\lambda})}$ exhibits the following properties:

- *Collision Resistance*: It is infeasible for any PPT adversary $\mathcal{A}$ to find two distinct inputs x, y such that $\mathcal{H}(x) = \mathcal{H}(y)$ [112].

- *Preimage Resistance*: It is infeasible for any PPT adversary $\mathcal{A}$ to find any input x given $\mathcal{H}(x)$.

Definition 2.7 (Statistically-hiding Commitment)**.** A statistically-hiding commitment scheme $\{\mathsf{COMM}_\rho : \{0,1\}^* \rightarrow \{0,1\}^{O(\boldsymbol{\lambda})}\}_\rho$ [31] where ρ denotes the commitment trapdoor, should hold the following two properties [112, 31]:

- *Statistically Hiding*: The commitment reveals nothing about the committed value for *every* adversary.
- *Computationally Binding*: It is impossible for all *PPT adversaries* to output commitment that can be opened in two different ways.

Definition 2.8 (Strongly-unforgeable Digital Signature)**.** A digital signature scheme Sig is defined as a tuple of algorithms $\mathsf{Sig} = (\mathcal{G}_{\mathsf{sig}}, \mathcal{K}_{\mathsf{sig}}, \mathcal{S}_{\mathsf{sig}}, \mathcal{V}_{\mathsf{sig}})$ such that:

- $\mathcal{G}_{\mathsf{sig}}(1^{\boldsymbol{\lambda}}) \rightarrow \textsc{pp}_{\mathsf{sig}}$. Given a security parameter, outputs public parameters $\textsc{pp}_{\mathsf{sig}}$ for the digital signature scheme.
- $\mathcal{K}_{\mathsf{sig}}(\textsc{pp}_{\mathsf{sig}}) \rightarrow (\mathrm{PKsig}, \mathrm{SKsig})$. Given public parameters $\textsc{pp}_{\mathsf{sig}}$, outputs a public key and a secret key for a single user.
- $\mathcal{S}_{\mathsf{sig}}(\mathrm{SKsig}, m) \rightarrow \sigma$. Given a secret key SKsig and a message m, signs m and outputs the signature σ.
- $\mathcal{V}_{\mathsf{sig}}(\mathrm{PKsig}, m, \sigma) \rightarrow \{0,1\}$. Given a public key PKsig, message m, and a signature σ, outputs 1 if the signature σ is valid for message m. Otherwise, outputs 0.

The signature scheme Sig satisfies the following security property:

- *Strong Unforgeability under Chosen Message Attack (SUF-CMA)*: The adversary cannot only produce a signature for a message that the key owner has not previously signed, but also cannot create an alternative signature for a message that has already been signed [51].

Definition 2.9 (Symmetric-key Encryption)**.** A symmetric-key encryption scheme SymEnc is defined as a tuple of algorithm $\mathsf{SymEnc} = (\mathcal{K}_{\mathsf{sym}}, \mathcal{E}_{\mathsf{sym}}, \mathcal{D}_{\mathsf{sym}})$, such that:

- $\mathcal{K}_{\mathsf{sym}}(1^{\boldsymbol{\lambda}}) \rightarrow \textsc{k}$ Given a security parameter, outputs a symmetric key $\textsc{k}$ for the encryption scheme.

- $\mathcal{E}_{\mathsf{sym}}(\mathrm{K}, m) \rightarrow c$. Given a symmetric key K and a message m, encrypts m and outputs the cipher-text c.
- $\mathcal{D}_{\mathsf{sym}}(\mathrm{K}, c) \rightarrow m$. Given a symmetric key K and a cipher-text c, decrypts c and outputs the plain-text m. $\mathcal{D}$ outputs $\perp$ if decryption fails.

The symmetric-key encryption scheme SymEnc satisfies the *Indistinguishability under Chosen Plaintext Attack (IND-CPA)* security property [112].

2.3 Proof Systems

The construction of zero-knowledge proof (ZKP) protocols is based on the interactive proof system (IP) introduced in 1985 by Goldwasser, Micali, and Rackoff in [90]. This section includes the introduction of IP, ZKP property, non-interactive proves.

2.3.1 Interactive Proof System

An IP consists of an interactive pair of a computationally unbounded prover (P) and a probabilistic polynomial time (PPT) verifier (V). Goldwasser et. al. [90] modeled each of P and V as an *interactive Turing machines*, which is a Turing machine with a read-only input tape, a read/write work tape, a read-only communication tape, a write-only communication tape, and a read-only random tape, where the random tape contains an infinit sequnce of random bits. P and V share the same input tape denoted as x. The write-only communication tape of P is the read-only communication tape of V and vice versa. In the original description of the interactive pair of Turing machines presented in [90], V is initially active, after which P and V alternate their activation. However, in the scenarios discussed later, P initiates the interaction by sharing a commitment to certain values. Each machine uses input tape, work tape, communication tapes and random tapes during its ith active stage, then it writes its ith message on its write-only communication tape and becomes deactive. V may terminate the protocol and either accept or reject the input x in its kth active stage. Here we provide a definition of IP:

Definition 2.10 (Interactive Proof System)**.** Let $L \in \{0,1\}^*$ be a language, and let P and V be an interactive pair of Turing machines. We say that P and V form an IP for L if P is computationally unbounded, V is PPT, and a common input $x \in \{0,1\}^n$, with sufficiently large n, is given to both P and V. They must satisfy the following properties:

1. For each k and any $x \in L$, V halts and accepts with probability at least $1 - \frac{1}{n^k}$.
2. For each k and any $x \notin L$, for any interactive Turing machine P^*, V halts and accepts with probability at most $\frac{1}{n^k}$.

Here, the probabilities are taken over V's internal randomness.

An interactive polynomial-time is a class of languages, denoted as $\mathcal{IP}$, posessing an interactive proof system (IP), where the membership of a common input $x \in \{0,1\}^n$ in the language is decided (verified) by a PPT verifier through k rounds of message exchange between the prover (P) and the verifier (V). Since V is probabilistic, its messages may depend on its internal randomness, often described as the verifier tossing a coin. Additionally, the prover's probabilistic behavior is crucial for achieving the zero-knowledge property [90], which we will discuss later.

Let (P, V) denote an IP which includes a computationally unbounded P and a PPT V. Let $\langle P, V(r_V)\rangle(x) \in \{0,1\}$ denote the output of the verifier V when interacting with a *deterministic* prover P, where r_V represents the internal randomness of V.

Definition 2.11. An IP (P, V) for a language L, given a public input $x \in \{0,1\}^n$, is said to have *completeness error* δ_c and *soundness error* δ_s if it satisfies the following conditions:

1. **Completeness**: For any $x \in L$, the prover P convinces V with probability
$$\Pr[\langle P, V(r_V)\rangle(x) = 1] \geq 1 - \delta_c.$$
2. **Soundness**: For any $x \notin L$, every cheating deterministic prover P^* convinces V with probability at most
$$\Pr[\langle P^*, V(r_V)\rangle(x) = 1] \leq \delta_s.$$

The language L belongs to the class $\mathcal{IP}$ if and only if $\delta_c, \delta_s \leq \frac{1}{3}$.

Remark 2.1. Every language in the nondeterministic polynomial-time class (NP), denoted as $L \in \mathcal{NP}$, has an interactive proof system (IP) in which both the prover and the verifier operate deterministically. Namely, $\mathcal{NP} \subseteq \mathcal{IP}$.

Perfect Completeness

As stated in [169], for any IP where $\delta_c < 0$, the system satisfies *perfect completeness*, meaning that $\delta_c = 0$. In all the IP systems discussed in this thesis, we assume $\delta_c = 0$, i.e., P always convinces V when $x \in L$.

Argument Systems and Computational Soundness

Definition 2.11 requires *statistical* (or *information-theoretic*) soundness, meaning that soundness holds against computationally unbounded provers in IP. However, *argument systems*, a special case of IP introduced in [49], require *computational* soundness, where soundness is guaranteed only against PPT provers [36]. This implies that an infinitely powerful prover could cheat in an argument system. Such systems typically assume that a PPT prover cannot cheat under a given cryptographic assumption. Through $\Theta(k)$ repetitions of the protocol, the soundness error δ_s can be reduced to δ_s^k.

2.3.2 Zero-knowledge Proof and Proof of Knowledge System

A zero-knowledge proof (ZKP) system for verifying whether a common input x belongs to a language L is an augmented IP in which the verifier is convinced of x's membership in L without gaining any additional knowledge beyond $x \in L$. The prover in ZKP systems is also prababilistic, meaning that we assume a read-only random tape, denoted as r_P for the prover machine.

Auxilary Input Tapes

Before defining ZKP systems, we introduce a secret tape for each interactive Turing machine described in Section 2.3.1, called the *read-only auxiliary input tape*. These tapes represent prior information available to P and V and may depend on the common input x, whose membership in L is to be verified by V through the IP protocol [87]. Let w denote the auxiliary input for P and z denote the auxiliary input for V. The input w may contain information that helps the prover efficiently perform its tasks. For example, for a language $L \in \mathcal{NP}$, w may be the witness for $x \in L$. On the other hand, z represents additional information about x that may be used by a dishonest verifier V^* to extract knowledge (e.g., beyond the mere fact that $x \in L$) through interaction with P. A ZKP system must guarantee that whatever knowledge V^* can extract from x, z, and interaction with P, can also be extracted from x and z alone, without any interaction with P.

Computational Indistinguishability and Simulation Paradigm

The required property of ZKP systems is defined through the *simulation paradigm*. Let the view of any verifier V^* during its interaction with the prover P be denoted as $\mathsf{View}_{V^*(r_V,z)}^{P(r_P,w)}(x)$.

This view consists of the verifier's random tape (r_V), auxiliary input tape (z), and the messages exchanged between the prover and verifier over the k interactions of the IP protocol. The verifier's view is often referred to as the *transcript* of the interactions in IP and is formally defined as

$$\mathsf{View}_{V^*(r_V,z)}^{P(r_P,w)}(x) := (r_V, z, p_1, \ldots, p_k),$$

where p_i denotes the prover's ith message.

Let $\{X_i\}_{i \in I}$ denote a sequence of random variables, also known as an *ensemble*, indexed by I, where each X_i is a random variable [87]. Let present the definition of computational indistinguishability, following [87, Definition 3.2.2]:

Definition 2.12 (Computational Indistinguishability)**.** Two ensembles $\{X_s\}_{s\in S}$ and $\{Y_s\}_{s\in S}$ are computationally indistinguishable if for any PPT distinguisher algorithm D, and all suffucuently long $s \in S$,

$$|\Pr[D(X_s, s) = 1] - \Pr[D(Y_s, s) = 1]| < \mathsf{negl}(s).$$

Now, suppose there exists a PPT simulator S^*, which, given the common input x and the verifier's auxiliary input z, produces an output $S^*(x, z)$ such that the ensembles

$$\{\mathsf{View}_{V^*(r_V,z)}^{P(r_P,w)}(x)\}_{x\in L, z\in\{0,1\}^*} \text{ and } \{S^*(x, z)\}_{x\in L, z\in\{0,1\}^*},$$

for any arbitrary w that satisfies the completeness condition of $x \in L$, are *computationally indistinguishable*, as defined in Definition 2.12. Since the same transcript can be generated by S^* without any interaction with P, the existence of such a simulator implies that V^* does not gain any additional knowledge from interacting with P.

Zero-Knowledge Proof

We now present the definition of the computational zero-knowledge IP, following [87, Definition 4.3.10].

Definition 2.13 (Zero-Knowledge Proof Property)**.** Assume an IP (P, V) for a language L, as defined in Definition 2.10. Let w and z denote the auxiliary input tapes of P and V, respectively, where $w \in W_L(x)$. Here, $W_L(x)$ denotes the set of auxiliary inputs that may be used to satisfy the completeness condition for $x \in L$. The IP (P, V) is zero-knowledge if for any PPT machine V^*, there exist a PPT algorithm S^*, such that the following two ensembles are computationally indistinguishable, as defined in Definition 2.12:

1. $\{\mathsf{View}^{P(r_P,w)}_{V^*(r_V,z)}(x)\}_{x\in L, z\in\{0,1\}^*}$ for any $w \in W_L(x)$,
2. $\{S^*(x,z)\}_{x\in L, z\in\{0,1\}^*}$.

Namely, for all sufficiently long $x \in L$, every $w \in W_L(x)$, and $z \in \{0,1\}^*$,

$$\left|\Pr[D(x,z,\mathsf{View}^{P(r_P,w)}_{V^*(r_V,z)}(x)) = 1] - \Pr[D(x,z,S^*(x,z)) = 1]\right| < \mathsf{negl}(|x|).$$

Zero-Knowledge Proofs for $\mathcal{NP}$ Languages

Let R be a binary relation for a language $L \in \mathcal{NP}$, where:

$$L = \{x \mid \exists w \text{ such that } (x,w) \in R\}.$$

Moreover, there exists a PPT algorithm that decides membership in R. Any language $L \in \mathcal{NP}$ is polynomial-time reducible to an $\mathcal{NP}$-complete language, such as the Graph 3-Colorability (G3C) problem. Goldreich, Micali, and Wigderson [88] presented a zero-knowledge IP protocol for G3C under the assumption of a secure *bit commitment scheme*, which can be instantiated using any one-way function. Shortly afterward, Brassard and Crépeau [50] developed a zero-knowledge IP protocol for Boolean circuit satisfiability, another well-known $\mathcal{NP}$-complete problem. Since any language $L \in \mathcal{NP}$ can be reduced to $\mathcal{NP}$-complete, we state the following theorem based on [87, Theorem 4.4.11]:

Theorem 2.1. *If a commitment scheme exists as defined in Definition 2.7, then every language $L \in \mathcal{NP}$ admits a zero-knowledge IP system.*

This thesis focuses exclusively on ZKP protocols for languages in the complexity class $\mathcal{NP}$. Unless explicitly stated otherwise (e.g., zero-knowledge IPs), any reference to ZKPs throughout this work adheres to the following definition:

Definition 2.14 (Zero-Knowledge Proof)**.** A zero-knowledge proof (ZKP) is a protocol consisting of a prover P, a verifier V, and an $\mathcal{NP}$ language $\mathcal{L}_{\mathbb{x}}$ corresponding to $\mathcal{NP}$-staement $\mathbb{x}$. The prover P possesses an instance (public input) x and a witness (private or auxiliary input) w such that $(x,w) \in \mathcal{R}_{\mathbb{x}}$, where $\mathcal{R}_{\mathbb{x}}$ is a polynomial-time decidable binary relation associated with $\mathcal{L}_{\mathbb{x}}$. The prover aims to convince the verifier that $x \in \mathcal{L}_{\mathbb{x}}$ without revealing any information about w.

Proof of Knowledge for $\mathcal{NP}$ Statements

Bellare and Goldreich [25] introduced a formal definition for the notion of proof of knowledge. Here, we summarize their central idea. Let the *prover's knowledge* refer to the auxiliary input (witness) w provided to the prover. Consider a PPT oracle machine $\mathcal{E}$, which has oracle access to $\mathcal{O}^{P^*(x,r_P,w)}(\bar{m})$. This oracle outputs messages generated by the prover $P^*(x, r_P, w)$ upon receiving previous message history $\bar{m}$, and it can be viewed as a *rewindable transcript generator*. Let

$$\Pr\left[\langle P^*, V\rangle(x) = 1\right]$$

denote the probability that P^* convinces V, and δ_k denote the probability that V accepts even if P^* did not know a witness w. $\mathcal{E}$ outputs a w^* during its interation with $\mathcal{O}^{P^*(x,r_P,w)}(\bar{m})$, where $(x, w^*) \in R$ with probability at least

$$\Pr\left[\langle P^*, V\rangle(x) = 1\right] - \delta_k.$$

2.3.3 Non-Interactive Proof System

Any IP defined in Definition 2.10 can be transformed into a non-interactive proof system using the Fiat-Shamir transformation [77]. Before discussing this transformation, we first introduce the concepts of private-coin versus public-coin IP protocols and the random oracle model.

Private Coin versus Public Coin

In the IP proposed by Goldwasser et al. [90], the verifier V does not reveal the random bits from its random tape to the prover P; that is, V's random tape remains private. Instead, V transmits only the results of computations performed on these random bits to P. This model is commonly referred to as a *private-coin* IP. In contrast, the IP introduced by Babai [15], known as the Arthur-Merlin proof system, allows P full access to V's random bits, leading to what is referred to as a *public-coin* IP. In this protocol, V does not perform any computations on the random bits but instead transmits them directly. Any subsequent deterministic computations that V would typically perform can instead be carried out by P. Goldwasser and Sipser later proved in [91] that these two models are equivalent, demonstrating that any private-coin IP can be efficiently simulated by a public-coin IP. Representing an IP as a public-coin protocol enables the technique of the Fiat-Shamir transformation [77], allowing the IP to be converted into a non-interactive proof.

Random Oracle Model

Let $R : \{0,1\}^* \rightarrow \{0,1\}^\kappa$ be a function that maps a binary string input to a sufficiently large, uniformly distributed random string. R represents a random oracle, accessible to both P and V. For each query $q \in \{0,1\}^*$ made by either P or V, the oracle responds with $R(q)$, which is independently and uniformly sampled from $\{0,1\}^\kappa$. The oracle maintains a record of its responses to ensure that if q is queried again, it returns the same value as before [169].

Fiat-Shamir Transformation

As mentioned earlier, any private-coin IP can be transformed into a public-coin IP, where V only sends public random bit strings. The Fiat-Shamir transformation replaces V's random bits with values derived from the random oracle R. Given that x is the common public input to the IP, the ith random value, denoted as r_i, which would have been sent by V, is instead computed as $r_i = R(q_i)$, where

$$q_i = (x, i, r_i, p_1, \ldots, p_{i-1})$$

is the query to the random oracle R, where p_i is the P's ith message in the interaction with V. This eliminates any interaction with V. Consequently, P can provide the transcript of the entire protocol, replacing the verifier's random bits with the corresponding outputs from R. As a result, this transcript is also *publicly verifiable*, meaning that anyone who sees the transcript can verify whether x belongs to L. Throughout this thesis, such a publicly verifiable transcript is referred to as a *proof* and is denoted by π.

Since the instantiation of the ideal random oracle R is impractical in real-world scenarios, it is replaced with hash functions in practice. This substitution enables a *publicly verifiable, non-interactive argument system* [169].

2.3.4 Succinct Non-Interactive Argument of Knowledge

In statistically sound IPs of languages in $\mathcal{NP}$, the communications is as much the size of the witness [141]. However, in argument systems relying on cryptographic assumptions (as discussed in Section 2.3.1), it is sufficient to achieve computational soundness against computationally bounded (i.e., polynomial-time) provers. In the following, we discuss methods for reducing the communication complexity (proof size) in such argument systems.

Succinct Non-Interactive Argument

In [114], Kilian introduced the notion of a *notarized envelope* to allow the prover to avoid revealing a random subset of committed bits. This approach enables the prover to reuse the same commitment to the bits across multiple rounds, rather than generating new commitments for each new permutation of bits, all while preserving the zero-knowledge property discussed in Section 2.3.2. Kilian, showed that this approach can be constructed via collision-resistant hash (CRH) functions, as defined in Definition 2.6. This construction facilitates a *succinct* interactive argument system, reducing communication overhead and significantly improving the verifier's efficiency compared to traditional $\mathcal{NP}$ verification methods. Namely, the membership of $x \in L$, where $L \in \mathcal{NP}$ can be verified by V in logarithm the time originally is required for $\mathcal{NP}$ verification. By applying Fiat-Shamir transformation [77], discussed in Section 2.3.3, we can have a succinct non-interactive argument (SNARG) system. To further reduce the proof size, the prover can commit to a polynomial constructed from witness values using a hiding and binding polynomial commitment scheme, rather than committing to each witness individually [169].

SNARG of Knowledge

The prover P in a SNARG system proves that there exists a witness w for the "$x \in L$" statement (the statement is true), where $L \in \mathcal{NP}$. However, it is not enough in many applications. Usually, P must prove that she knows that witness too (argument of knowledge). The SNARGs in which P is able to prove that she knows the witness are called succinct non-interactive argument of knowledge (SNARK). In [45], Bitansky et al. proved that if and only if extractable collision-resistant hash (ECRH) functions exist, verifier of SNARKs will exist. The prover can use an *extractable* polynomial commitment to shorten the proofs, while extractability ensures that the prover *knows* the polynomial constructed from the witnesses [169]. SNARKs can be modified so that a verifier does not learn anything about the witness by converting that to a zero-knowlege succinct non-interactive argument of knowledge (zkSNARK).

In summary, zkSNARK is a non-interactive zero-knowledge (NIZK) argument of knowledge that has succinct proof size and the proofs are publicly verifiable. Here, we provide a formal definition of zkSNARK algorithms and properties, which will be referenced throughout this thesis:

Definition 2.15 (zkSNARK Algorithms)**.** zkSNARK consists of a triple of polynomial-time algorithms which are defined as follows:

- $\mathsf{KeyGen}(1^{\boldsymbol{\lambda}}, \mathbb{x}) \rightarrow (\mathrm{pk}_{\mathbb{x}}, \mathrm{vk}_{\mathbb{x}})$. Given a security parameter $\boldsymbol{\lambda}$ and an $\mathcal{NP}$ statement $\mathbb{x}$, KeyGen outputs a proving key $\mathrm{pk}_{\mathbb{x}}$ and a verification key $\mathrm{vk}_{\mathbb{x}}$. The proving and verifying keys are both published as public parameters. They can be used, any number of times, to prove membership in $\mathcal{L}_{\mathbb{x}}$ and verify the proof, respectively.

- $\mathsf{Prove}(\mathrm{pk}_{\mathbb{x}}, x, w) \rightarrow \pi$. Given a proving key $\mathrm{pk}_{\mathbb{x}}$ and any $(x, w) \in \mathcal{R}_{\mathbb{x}}$, the algorithm outputs a non-interactive publicly verifiable proof π for the instance $x \in \mathcal{L}_{\mathbb{x}}$.

- $\mathsf{Verify}(\mathrm{vk}_{\mathbb{x}}, x, \pi) \rightarrow \{0, 1\}$. Given a verification key $\mathrm{vk}_{\mathbb{x}}$, public input x, and a proof π, the algorithm outputs 1 if proof π can convince that $x \in \mathcal{L}_{\mathbb{x}}$.

Depending on the commitment scheme used in the zkSNARK construction, the KeyGen algorithm may need to be executed by a trusted third party. In the literature, $\mathrm{pk}_{\mathbb{x}}$ and $\mathrm{vk}_{\mathbb{x}}$ are sometimes collectively referred to as the public parameters (PP) or the common reference string (CRS).

Definition 2.16 (zkSNARK Properties)**.** zkSNARK satisfies the following properties:

- *Perfect Completeness*: A prover P who knows a witness w for $x \in \mathcal{L}_{\mathbb{x}}$ can convince a verifier V with probablility 1.

- *Soundness*: If $x \notin \mathcal{L}_{\mathbb{x}}$, a dishonest PPT prover P^* can convince a verifier with probability less than $\delta_s < \mathsf{negl}(\boldsymbol{\lambda})$.

- *Knowledge Soundness (proof of knowledge) with error δ_k*: For every dishonest PPT prover P^* and every problem statement $\mathbb{x}$ and x, if there exist a PPT extractor $\mathcal{E}_{\mathbb{x}}$ algorithm, the probability of $\mathcal{E}_{\mathbb{x}}$ extracting the witness w given oracle access to $\mathcal{O}^{P^*}$, must be at least as high as the success probability of P^* in convincing the verifier.

$$\begin{aligned}\Pr\left[(x, w) \in \mathcal{R}_{\mathbb{x}} : w \leftarrow \mathcal{E}_{\mathbb{x}}^{\mathcal{O}^{P^*}(\bar{m})}(x)\right] \geq \\ \Pr\left[\langle P^*, V\rangle(x) = 1\right] - \delta_k\end{aligned}$$

Where the oracle access to P^* is denoted as $\mathcal{E}_{\mathcal{O}^{P^*}(\bar{m})}$ and $\delta_k < \mathsf{negl}(\boldsymbol{\lambda})$.

- *Succinctness*: The size of a publicly verifiable proof π and the verification algorithm $\mathsf{Verify}(\mathrm{vk}_{\mathbb{x}}, x, \pi)$ run-time is at most linear in the size of the instance x $(O_{\boldsymbol{\lambda}}(|x|))$.

- *Computational Zero-knowledge*: For a polynomial-time simulator Sim that can produce a proof π^*, and for all stateful disinguishers $\mathcal{D}$, the probability that $\mathcal{D}(\pi) = 1$ on an honest proof is equal to the probability that $\mathcal{D}(\pi^*) = 1$ on a simulated proof.

Different zkSNARK constructions result in varying proof sizes. For instance, Groth's scheme [95] generates constant-size proofs, while the Aurora zkSNARK[35], which we discuss in Section2.4.3, produces proofs whose sizes grow linearly with the instance size.

2.3.5 Trusted vs. Transparent Setup zkSNARKs

The extractable polynomial commitment scheme underlying a zkSNARK determines whether the protocol requires a trusted or transparent setup. Protocols using polynomial commitments such as the KZG scheme, proposed by Kate, Zaverucha, and Goldberg [110], require a one-time trusted setup to generate a structured common reference string (CRS). In contrast, polynomial commitment schemes like Bulletproofs [53] do not require such a trusted setup. However, the security of both KZG commitments and Bulletproofs relies fundamentally on the discrete logarithm assumption.

Conversely, polynomial commitment schemes based on interactive oracle proofs (IOP) of proximity, such as FRI [26], enable transparent setups. These schemes avoid reliance on the discrete logarithm assumption and other cryptographic assumptions known to be vulnerable to quantum computing attacks. Instead, their security relies on the existence and security properties of cryptographic hash functions (CRHs). This transparent polynomial commitment scheme is detailed further in Section 2.4.2.

2.4 Relevant Proof Protocols

This section presents various protocols that are used in this thesis.

2.4.1 GKR Protocol

GKR [89] is a public coin interactive proof protocol for any language computable by a log-space uniform layered (fan-in 2) arithmetic circuit $\mathcal{C}$, in which a prover $\mathcal{P}$ can run the computation and interactively prove the correctness of the result (output gate(s)) to a verifier $\mathcal{V}$. Consider a circuit with depth denoted by d and size represented by S, where the size is defined as the total number of gates. For any layer ℓ within the circuit, S_ℓ indicates the number of gates at that layer. Specifically, $\ell = 0$ corresponds to the output layer, while $\ell = d$ represents the input layer. Additionally, ν is the size of the input to the circuit $\mathcal{C}$. The communication cost is $O(S_0 + d\log(S))$, the cost of $\mathcal{V}$ is $O(\nu + d\log(S))$, and the runtime of the $\mathcal{P}$ is bounded by $O(S^3)$. In the following, we briefly describe the GKR protocol following Thaler's presentation of the protocol in [169].

Circuit Encoding. At each layer ℓ, the gates are numerically labeled in binary from 0 to $S_\ell - 1$, assuming S_ℓ is a power of two (expressed as $S_\ell = 2^{k_\ell}$). The functions $\mathsf{in1}_\ell$ and $\mathsf{in2}_\ell$, each defined as $\mathsf{in1}_\ell, \mathsf{in2}_\ell : \{0,1\}^{k_\ell} \to \{0,1\}^{k_{\ell+1}}$, map a binary gate label at layer ℓ to its input gates at layer $\ell+1$. This mapping explicitly encodes the wiring—how outputs from gates at layer $\ell+1$ serve as inputs to a gate at layer ℓ. Accordingly, the functions add_ℓ and mult_ℓ, representing addition and multiplication gates at layer ℓ, are defined as

$$\mathsf{add}_\ell, \mathsf{mult}_\ell : \{0,1\}^{k_\ell} \times \{0,1\}^{k_{\ell+1}} \times \{0,1\}^{k_{\ell+1}} \to \{0,1\}.$$

For a gate labeled a at layer ℓ, these functions take as input the labels of three gates (a,b,c), and return 1 if and only if $(b,c) = (\mathsf{in1}_\ell(a), \mathsf{in2}_\ell(a))$. $\widetilde{\mathsf{add}}_\ell$ and $\widetilde{\mathsf{mult}}_\ell$ denote the Multilinear Extension (MLE) of add_ℓ and mult_ℓ. Additionally, the function $W_\ell : \{0,1\}^{k_\ell} \to \mathbb{F}$, maps gate at layer ℓ to the outputted value of the gate. Accordingly, $\widetilde{W}_\ell$ denote the MLE of W_ℓ. The following equation describes how $\widetilde{W}_\ell$ can be derived form $\widetilde{W}_{\ell+1}$, $\widetilde{\mathsf{add}}_\ell$, and $\widetilde{\mathsf{mult}}_\ell$:

$$\begin{aligned}\widetilde{W}_\ell(z) = \sum_{b,c \in \{0,1\}^{k_{\ell+1}}} \Big(& \widetilde{\mathsf{add}}_\ell(z,b,c) \left(\widetilde{W}_{\ell+1}(b) + \widetilde{W}_{\ell+1}(c) \right) \\ & + \widetilde{\mathsf{mult}}_\ell(z,b,c) \left(\widetilde{W}_{\ell+1}(b) \cdot \widetilde{W}_{\ell+1}(c) \right) \Big).\end{aligned}$$

Multivariate Sum-check. The GKR protocol consists of an iteration for each layer. In the iteration corresponding to layer $\ell < d$ of the circuit, $\mathcal{P}$ claims a specific value for $\widetilde{W}_\ell(r_\ell)$, with $r_\ell \in \mathbb{F}^{k_\ell}$ being a randomly selected point. Note that r_ℓ may have non-Boolean entries. In order to check the claim, $\mathcal{P}$ and $\mathcal{V}$ cooperate in a multivariate sum-check protocol [127] to the polynomial $f_{r_\ell}^{(\ell)}$ defined as

$$\begin{aligned}f_{r_\ell}^{(\ell)}(b,c) = & \widetilde{\mathsf{add}}_\ell(r_\ell,b,c) \left(\widetilde{W}_{\ell+1}(b) + \widetilde{W}_{\ell+1}(c) \right) \\ & + \widetilde{\mathsf{mult}}_\ell(r_\ell,b,c) \left(\widetilde{W}_{\ell+1}(b) \cdot \widetilde{W}_{\ell+1}(c) \right).\end{aligned}$$

Given that $\mathcal{V}$ does not know the polynomial $f_{r_\ell}^{(\ell)}$, to evaluate $f_{r_\ell}^{(\ell)}(b^*,c^*)$ in the final round of the sum-check protocol at a randomly chosen point $(b^*,c^*) \in \mathbb{F}^{k_{\ell+1}} \times \mathbb{F}^{k_{\ell+1}}$, $\mathcal{V}$ asks $\mathcal{P}$ to provide $z_1 = \widetilde{W}_{\ell+1}(b^*)$ and $z_2 = \widetilde{W}_{\ell+1}(c^*)$, which are then verified in the subsequent iteration $(i+1)$ through the *round consistency check* process. However, $\mathcal{V}$ can independently evaluate $\widetilde{\mathsf{add}}_\ell(r_\ell, b^*, c^*)$ and $\widetilde{\mathsf{mult}}_\ell(r_\ell, b^*, c^*)$, according to the circuit's structure.

Round Consistency Check. To verify z_1 and z_2, $\mathcal{V}$ aims to *reduce* these verifications into a single task: validating the $\mathcal{P}$'s claim of $\widetilde{W}_{\ell+1}(r_{\ell+1})$. This claim represents the summation outcome at the next layer denoted as

$$\widetilde{W}_{\ell+1}(r_{\ell+1}) = \sum_{b,c \in \{0,1\}^{k_{\ell+2}}} f^{(\ell+1)}_{r_{\ell+1}}(b, c).$$

To do so, let define the unique line $\lambda : \mathbb{F} \to \mathbb{F}^{k_{\ell+1}}$, such that $\lambda(0) = b^*$ and $\lambda(1) = c^*$. $\mathcal{P}$ then sends the polynomial $q = \widetilde{W}_{\ell+1}\big|_\lambda$ to $\mathcal{V}$, representing the restriction of $\widetilde{W}_{\ell+1}$ to the line λ. Upon receiving the polynomial, $\mathcal{V}$ first verifies that $q(0) = z_1$ and $q(1) = z_2$. Subsequently, $\mathcal{V}$ selects a random $r^* \in \mathbb{F}$ and sets $r_{\ell+1} = \lambda(r^*)$, then checks if $q(r^*) = \widetilde{W}_{\ell+1}(r_{\ell+1})$, which is the claim made by $\mathcal{P}$ for the next round.

Final Round Check. In the final round, $\mathcal{V}$ independently checks $\widetilde{W}_d(r_d)$.

2.4.2 FRI Protocol

Fast Reed-Solomon interactive oracle proof of proximity (FRI) protocol is a low-degree test for polynomials. It is used to determine whether a polynomial f is low-degree with respect to the size of the evaluation domain, without actually knowing f itself [26].

The FRI protocol begins with a polynomial $f_0(X)$ and its evaluation domain L_0, which is an affine subspace in $\mathbb{F}$. The polynomial $f_0(X)$ and domain L_0 undergo a stepwise reduction process using a random folding procedure, resulting in a sequence of polynomials

$$f_0(X), f_1(X), \cdots, f_{\mathsf{r}}(X) \in \mathbb{F}[X], \tag{2.2}$$

and a sequence of domains

$$L_0 \supseteq L_1 \supseteq \cdots \supseteq L_{\mathsf{r}}. \tag{2.3}$$

Suppose d_k is the upper bound of polynomial's degree, i.e., $\deg f_k(X) < d_k$. We assume the degrees decrease with the same ratio as the domains, which means the quotients

$$\frac{d_k}{d_{k+1}} = \frac{|L_k|}{|L_{k+1}|} \tag{2.4}$$

are the same, called *reduction factors*, and in this papar we always assume the reduction factors to be 2.

Let $f^{(k)}$ ($0 \leq k \leq \mathsf{r}$) be a Reed Solomn (RS) codeword defined as follows,

$$\begin{aligned} f^{(k)} : L_k \to & \quad \mathbb{F}, \\ x \mapsto & \quad f_k(x). \end{aligned} \tag{2.5}$$

The notation caveat should be noted here that $f_k(X)$ represents a polynomial, while $f^{(k)}$ is an array of points representing a RS codeword. The interpolant of $f^{(k)}$ would construct the polynomial $f_k(x)$. The rate of the RS codeword is defined as

$$\rho = \frac{d_k}{|L_k|}. \tag{2.6}$$

FRI is an IP containing a Commit phase and a Query phase, running for r rounds.

Commit Phase. During the k-th round of the Commit phase $(0 \leq k \leq \mathsf{r} - 1)$, the prover commits to $f^{(k)}$ and the verifier has oracle access to $f^{(k)}$. In this paper, Merkle tree commitment is employed.

During the last round $k = \mathsf{r}$, prover sends the $f^{(\mathsf{r})}$ to the verifier. By this point, the degree of $f_{\mathsf{r}}(X)$, which is the interpolant of $f^{(\mathsf{r})}$, should be no more than $\rho \cdot |L_{\mathsf{r}}| - 1$.

Query Phase. The verifier will validate that the prover adheres to the prescribed procedures.

Specifically, the verifier will randomly select $s^{(0)}$ from L_0, and iteratively computes a sequence of points $s^{(0)}, s^{(1)}, \cdots, s^{(\mathsf{r})}$. The verifier will query the value $f_{k+1}(s^{(k+1)})$, and two other points from $f^{(k)}$, and check the round consistency among those three points $(0 \leq k \leq \mathsf{r} - 1)$. The verifier will repeat this check for ℓ times, and accept only when all checks pass.

If verifier accepts, it means that the degree of the original polynomial $f_0(X)$ should be no more than $\rho \cdot |L_0| - 1$.

2.4.3 Aurora zkSNARK Protocol

Aurora [35] is a zkSNARK protocol for R1CS relations. It encodes an R1CS instance into entries of Reed-Solomon (RS) codewords and employs an interactive oracle proof (IOP) framework, which leverages the FRI [26] to perform a sumcheck proof on these entries. FRI is employed to verify that the given RS code is close to a low-degree polynomial, or low degree testing (LDT). It progressively reduces the size of the codeword based on a localization parameter η.

Aurora ensures the zero-knowledge property for its univariate sumcheck protocol and LDT, and consequently for the entire protocol, by employing the technique introduced in [30].

At the beginning of the protocol, the prover P and verifier V establish a finite field $\mathbb{F}$, a security parameter $\boldsymbol{\lambda}$, a rate ρ, an FRI localization parameter η, an R1CS parameters $(\mathbb{F}, d_1, d_2, d_3, \mathbf{A}, \mathbf{B}, \mathbf{C})$, and two subspaces $H_1, H_2 \subset \mathbb{F}$, where $|H_1| = d_1$ and $|H_2| = d_2 + 1$ such that $H_1 \subseteq H_2$ or $H_2 \subseteq H_1$. We can write $H_1 \cup H_2 = \{h_0, \ldots, h_{t-1}\}$, where $t = |H_1 \cup H_2| = \max(d_1, d_2 + 1)$.

First, the prover computes the low degree extension (LDE) of the unique polynomial of degree $\kappa + 1$

$$f_{(1,\mathbf{v})}(h_i) = \begin{cases} 1 & \text{for } i = 0, \\ \mathbf{v}_{i-1} & \text{for } i \in [1, \kappa]. \end{cases}$$

Accordingly, the prover computes RS codewords listed in Table 2.1, where L denotes the codeword domain. To achieve zero-knowledge against query bound $\mathtt{b}$ malicious verifier, the degree of non-masking polynomials in Table 2.1 is incremented by $\mathtt{b}$ to enable sampling the polynomials at random while it evaluates to pre-determined values in $H_1 \cup H_2$. Accordingly, any set of $\mathtt{b}$ evaluations of the polynomials outside of $H_1 \cup H_2$ are independently and uniformly distributed in $\mathbb{F}$. Hence, the codeword domain is chosen such that $L \cap (H_1 \cup H_2) = \emptyset$.

Aurora consists of λ_i rounds of *lincheck* protocol, where in round $\ell \in [1, \lambda_i]$ the verifier sends four random numbers represented as $\alpha_\ell, s_\ell^A, s_\ell^B, s_\ell^C \in_R \mathbb{F}$, then both prover and verifier compute the polynomials

$$p_{\alpha_\ell}(h_i) = \begin{cases} {\alpha_\ell}^{i+1} & \text{for } i \in [0, \eta - 1] \\ 0 & \text{for } i \in [\eta, \mu] \end{cases}, \text{ and } p_{\alpha_\ell}^{ABC}(h_j) = \sum_{M \in \{A,B,C\}} s_\ell^A p_{\alpha_\ell}^M(h_j), \tag{2.7}$$

where

$$p_{\alpha_\ell}^M(h_j) = \begin{cases} \sum_{i \in [0,\eta-1]} M_{i,j} {\alpha_\ell}^{i+1} & \text{for } j \in [0, \mu], \\ 0 & \text{for } j \in [\mu + 1, \eta]. \end{cases} \tag{2.8}$$

The prover then proceeds to prove the consistency of the known witness with the R1CS instance. This is achieved by reducing the random combination of the codewords to a

Table 2.1: Codewords generated by prover. λ_i' denote the number of repetitions of LDT. The mechanism for randomizing polynomials, denoted as *, is omitted here for the sake of simplicity.

Codeword(Oracle)	**Description**
$\hat{\mathbf{f}}_{\mathbf{w}} \in \mathrm{RS}[L, \frac{\mu-\kappa+\mathtt{b}}{\lvert L\rvert}]$	$\hat{\mathbf{f}}_{\mathbf{w}} := f^*_{\mathbf{w}}\rvert_L$, where $f^*_{\mathbf{w}}$ is a random polynomial of degree $< \mu - \kappa + \mathtt{b}$ such that for $\kappa < i \leq \mu$, $f_{\mathbf{w}}(h_i) = (\mathbf{w}_{i-\kappa-1} - f_{(1,\mathbf{v})}(h_i))/\mathbb{Z}_{\{h_0,\ldots,h_\kappa\}}$.
$\hat{\mathbf{f}}_{\mathbf{Az}} \in \mathrm{RS}[L, \frac{\eta+\mathtt{b}}{\lvert L\rvert}]$, $\hat{\mathbf{f}}_{\mathbf{Bz}} \in \mathrm{RS}[L, \frac{\eta+\mathtt{b}}{\lvert L\rvert}]$, $\hat{\mathbf{f}}_{\mathbf{Cz}} \in \mathrm{RS}[L, \frac{\eta+\mathtt{b}}{\lvert L\rvert}]$	$\hat{\mathbf{f}}_{\mathbf{Mz}} := f^*_{\mathbf{Mz}}\rvert_L$ for $\mathbf{M} \in \{\mathbf{A}, \mathbf{B}, \mathbf{C}\}$, where $f^*_{\mathbf{Mz}}$ is a random polynomial of degree $< \eta + \mathtt{b}$ such that for $i \in [0, \eta - 1]$, $f_{\mathbf{Mz}}(h_i) = \mathbf{Mz}_i$.
$\hat{\mathbf{r}}_\ell \in_R \mathrm{RS}[L, \frac{2t+\mathtt{b}-1}{\lvert L\rvert}]$	For $\ell \in [1, \lambda_i]$, $\hat{\mathbf{r}}_\ell$ is a random masking codeword for lincheck
$\hat{\mathbf{h}}_\ell \in \mathrm{RS}[L, \frac{t+\mathtt{b}}{\lvert L\rvert}]$	For $\ell \in [1, \lambda_i]$, $\hat{\mathbf{h}}_\ell := h\rvert_L$, where h is a polynomial of degree $< t + b$ such that it is the quotient of the polynomial division $r_\ell(X) + \sum_{\mathbf{M}\in\{\mathbf{A},\mathbf{B},\mathbf{C}\}} \left(f_{\mathbf{Mz}}(X)p_{\alpha_\ell}(X) - f_{\mathbf{z}}(X)p^M_{\alpha_\ell}(X)\right) = g_\ell(X) + \frac{\sum_{i\in[0,t-1]} r_\ell(h_i)}{\sum_{i\in[0,t-1]} h_i^{t-1}} \cdot X^{t-1} + \mathbb{Z}_{H_1\cup H_2} \cdot h(X)$
$\hat{\mathbf{r}}'_\ell \in_R \mathrm{RS}[L, \frac{2t+2\mathtt{b}}{\lvert L\rvert}]$	For $\ell \in [1, \lambda_i']$, $\hat{\mathbf{r}}_\ell$ is a random masking codeword for LDT

univariate sumcheck protocol, equivalent to a low-degree testing (LDT) which is realized using the FRI of proximity parameter

$$\delta = \min\left(\frac{1-2\rho}{2}, \frac{1-\rho}{3}, 1-\rho\right). \tag{2.9}$$

Furthermore, the transformation described in [36] is employed to compile the described interactive protocol into a zkSNARK. In Section 4.4, we will discuss the FFT complexity of Aurora which includes the size of the codeword domain $|L|$, and the number of input lengths FFT/IFFT calls. For example, the prover calls FFT and IFFT algorithms to compute the codewords in Table 2.1 and both prover and verifier call IFFT to interpolate the polynomials p_α and p_α^{ABC}.

2.4.4 Polaris zkSNARK Protocol

Polaris [81] is a zkSNARK protocol without a trusted setup that has quasi-linear time complexity for the prover and polylogarithmic proof size. Its verification time is relative to the size of the arithmetic circuit representing the statement to be proven. It achieves this efficiency by encoding the R1CS instance as a univariate polynomial in a quadratic arithmetic program (QAP) [85]. The main source of Polaris' efficiency is an arithmetic layered circuit design that allows the verifier to delegate query computations to the prover and verify the results using the GKR protocol [89]. Polaris combines univariate polynomial encoding described in Section 2.4.4 with univariate sumcheck protocol in Aurora [35]. By accomplishing this, the protocol constructs an interactive proof that is complete and sound, and then extends it to incorporate zero-knowledge and non-interativeness using Fiat-Shamir protocol [77]. Figure 2.1 below shows the building blocks of the Polaris protocol.

R1CS Instance

Following Definition 2.2 The R1CS instance can be represented as a tuple $\mathbb{x} = (\mathbb{F}, A, B, C, \vec{v}, m, n)$, where $\mathbb{F}$ is the finite field, A, B are input matrices and C is the output matrix of degree $m \times m$ from the R1CS construction. $\vec{v}$ represents the vector containing the instance's public parameter. There are at most n non-zero entries in each matrix.

An R1CS relation $\mathcal{R}_{\text{R1CS}}$ is said to be satisfiable if there exists a witness $\vec{w} \in \mathbb{F}^{m-|v|-1}$ consisting of the circuit's private input and wire values such that

$$(A\vec{z}) \circ (B\vec{z}) = C\vec{z},$$

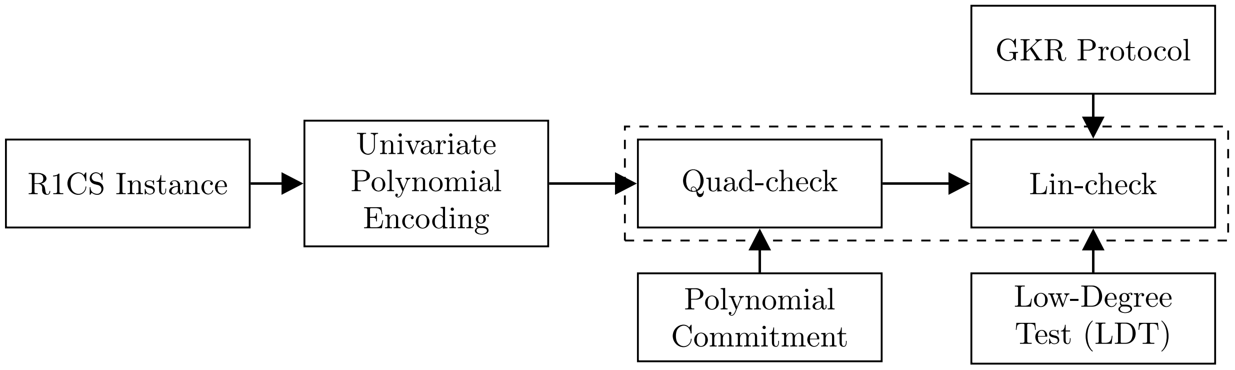


Figure 2.1: Structure of the Polaris protocol with key sub-protocols.

where $\vec{z} := (1, \vec{w}, \vec{v})$ and "$\circ$" denotes the Hadamard product of the two vectors $A\vec{z}$ and $B\vec{z}$. This relation $\mathcal{R}_{\text{R1CS}}$ represents the input and output vectors of the gates in the arithmetic circuit.

Univariate Polynomial Encoding

Let H be an s-dimensional affine space of $\mathbb{F}$ such that $|H| = m$. The vector $\mathbf{z} = (1, \mathbf{v}, \mathbf{w}) \in \mathbb{F}^H$ is interpreted as a univariate function $Z : H \to \mathbb{F}$. This allows the accessing of any element of vector $\mathbf{z}$ using an index in H. The function $F_{\mathbf{w}}(\cdot)$ encodes $\mathbf{z}$, which contains the private witness $\mathbf{w}$, into a polynomial form for use in the protocol. It is defined as

$$F_{\mathbf{w}}(X) = \left(\sum_{y \in H} A(X, y) \cdot Z(y)\right) \cdot \left(\sum_{y \in H} B(X, y) \cdot Z(y)\right) - \left(\sum_{y \in H} C(X, y) \cdot Z(y)\right).$$

A witness-instance pair $(\mathbb{x}, \mathbf{w})$ is deemed valid, i.e., $(\mathbb{x}, \mathbf{w}) \in \mathcal{R}_{\text{R1CS}}$ if and only if $F_{\mathbf{w}}(x) = 0$ for any $x \in H$. Polaris utilises the polynomial extension of $F_{\mathbf{w}}(\cdot)$, for its arithmetisation in the protocol. We assign

$$\bar{A}(X) = \sum_{y \in H} A(X, y) \cdot Z(y),$$
$$\bar{B}(X) = \sum_{y \in H} B(X, y) \cdot Z(y),$$
$$\bar{C}(X) = \sum_{y \in H} C(X, y) \cdot Z(y).$$

To find the coefficients of the three polynomials, we can compute the vector products $A\mathbf{z}, B\mathbf{z}$ and $C\mathbf{z}$, and then interpolate those vector results over H. Given that A, B and C are sparse matrices, the prover employs *sparse encoding* approach for efficiently finding the coefficient of each $\bar{A}$, $\bar{B}$, and $\bar{C}$. To do so, we define three functions for each matrix M, where $M \in \{A, B, C\}$. These functions, $\mathsf{row}, \mathsf{col} : [n] \to H$, and $\mathsf{val} : [n] \to \mathbb{F}$, respectively map from the set of indices $[n]$ to the row and column indices, and to the value of the non-zero entries within the matrix M. Here $[n] := \{1, 2, \cdots, n\}$. Thus, for every $x \in H$,

$$\bar{M}(x) = \sum_{i \in [n] \text{ s.t. } \mathsf{row}(i) = x} \mathsf{val}(i) \cdot Z(\mathsf{col}(i)).$$

We can represent any sequence of length N over $\mathbb{F}$ as a univariate polynomial by using Lagrange interpolation. For a sequence $\{f_i\}$, applying Lagrange interpolation would result in a polynomial f, where $f(\rho(i)) = f_i$. Here, $\rho(i) = i_0\alpha_0 + \cdots + i_{s-1}\alpha_{s-1}$ with $(i_0, \cdots, i_{s-1})$ as the binary representation of i. The points used in the Lagrange interpolation are $(\rho(i), f_i), 0 \le i < 2^s - 1$. In other words, we have

$$f(x) = \sum_{i=0}^{2^s-1} f_i \sigma_i(x), \tag{2.10}$$

where $\{\sigma_i \,|\, 0 \le i < 2^s\}$ is the Lagrange basis, given by

$$\sigma_i(x) = \frac{\prod_{j \neq i}(x - \rho(j))}{\prod_{j \neq i}(\rho(i) - \rho(j))}. \tag{2.11}$$

Since $f(x)$ is a polynomial with coefficients in $\mathbb{F}$, we can naturally extend f from a function mapping $N_{2^s} \to \mathbb{F}$ to a function over $\mathbb{F}$.

Given the definition of the vanishing polynomial, we can further represent $\mathbb{Z}_H(x)$, an affine linearized polynomial of $\mathbb{F}$ with dimension s, as below,

$$\mathbb{Z}_H(x) = x^{2^k} + \sum_{i=1}^{s} c_i x^{2^{i-1}} + c_0, c_i \in \mathbb{F}.$$

If H is linear, then $c_0 = 0$. Note that $c_1 \neq 0$, since $\mathbb{Z}_H(x)$ has no repeated roots. This also means that $\mathbb{Z}_H$ has degree $|H|$. To perform univariate encoding, Polaris makes use of the following bivariate polynomial:

$$\Delta_H(x, y) := \frac{\mathbb{Z}_H(x) - \mathbb{Z}_H(y)}{x - y}, \tag{2.12}$$

which is a polynomial of degree $|H|-1$ because $\mathbb{Z}_H(x)-\mathbb{Z}_H(y)$ is divisible by $x-y$. With the above notation, the Lagrange basis in 2.11 is given as

$$\sigma_i(x) = \frac{\Delta_H(x,\rho(i))}{c_1} = \frac{1}{c_1}\frac{Z_H(x)}{x-\rho(i)}, 0 \le i < 2^s,$$

and (2.10) now becomes

$$f(x) = \frac{1}{c_1}\sum_{i\in N_{2^s}} f_i \Delta_H(x,\rho(i)) = \frac{1}{c_1}\sum_{i\in N_{2^s}} f_i \frac{Z_H(x)}{x-\rho(i)}. \tag{2.13}$$

Equation (2.13) represents a univariate polynomial leveraged from bivariate interpolation.

Quadratic and Linear Checks In the Polaris protocol, the prover $\mathcal{P}$ wants to convince the verifier $\mathcal{V}$ that $(\mathbb{x},\mathbf{w}) \in \mathcal{R}_{\mathrm{R1CS}}$ which is true if and only if the univariate $\mathbb{F}_{\mathbf{w}}(x)=0$ at all points within the affine subspace H of $\mathbb{F}$. According to the factor theorem (Theorem 1 in [81]), this verification is the same as determining if there's a polynomial $G(X)$ where $\deg(F_{\mathbf{w}}) - |H| \le |H|-2$, such that

$$F_{\mathbf{w}}(X) = \mathbb{Z}_H(X)\cdot G(X). \tag{2.14}$$

This is verified through two distinct checks, namely, the Quad-Check and the Lin-Check, as presented in the following sections.

Quad-Check. In the Quad-Check protocol, $\mathcal{V}$ conducts a probabilistic check of Equation (2.14) at a randomly selected point. To realize this, $\mathcal{P}$ commits to the polynomial $G(X)$ using FRI univariate polynomial commitment and sends the commitment to $\mathcal{V}$. Section 5.2 presents the instantiation of the FRI commitment scheme in our protocol. Then, $\mathcal{V}$ sends the randomly selected point $r_x \in F\setminus H$ to $\mathcal{P}$. Then, the prover evaluates $\eta = G(r_x)$ and sends the result to $\mathcal{V}$. Finally, $\mathcal{V}$ query the commitment to verify the evaluation.

To verify $F_w(r_x) = \mathbb{Z}_H(r_x)\cdot G(r_x)$, the verifier also needs to compute $F_w(r_x)$. Recall that

$$F_w(r_x) = \bar{A}(r_x)\cdot\bar{B}(r_x) - \bar{C}(r_x).$$

The prover makes three separate claims to $\mathcal{V}$, say that $\bar{A}(r_x)=v_A$, $\bar{B}(r_x)=v_B$, and $\bar{C}(r_x)=v_C$. Then the verifier $\mathcal{V}$ can check if

$$v_A\cdot v_B - v_C = G(r_x)\cdot\mathbb{Z}_H(r_x).$$

This equation is represents the quadratic-check of the R1CS instance, or *quad-check*, owing to it checking the R1CS verifiability in $\rho(i)$, where i is within the range of the matrix degree m. But the verifier must now additionally verify three new claims from the prover $\mathcal{P}$:

$$\bar{A}(r_x) = v_A, \bar{B}(r_x) = v_B, \text{ and } \bar{C}(r_x) = v_C. \tag{2.15}$$

To do so, $\mathcal{P}$ and $\mathcal{V}$ engage in the Lin-check protocol.

Lin-Check. In the Lin-Check protocol, the three claims presented in Equation (2.15) are combined into one to be evaluated in a single claim. Accordingly, the verifier chooses r_A, r_B, $r_C \in \mathbb{F}$ uniformly at random and sends them to the prover, which can test if

$$c = r_A \cdot \bar{A}(r_x) + r_B \cdot \bar{B}(r_x) + r_C \cdot \bar{C}(r_x). \tag{2.16}$$

We can rewrite c as follows.

$$\begin{aligned} c &= r_A \cdot \sum_{y \in H} A(r_x, y) \cdot Z(y) + r_B \cdot \sum_{y \in H} B(r_x, y) \cdot Z(y) + r_C \cdot \sum_{y \in H} C(r_x, y) \cdot Z(y) \\ &= \sum_{y \in H} \big(r_A \cdot A(r_x, y) + r_B \cdot B(r_x, y) + r_C \cdot C(r_x, y)\big) \cdot Z(y). \end{aligned}$$

where $y \in H$. We denote

$$Q_{r_x}(Y) := \big(r_A \cdot A(r_x, Y) + r_B \cdot B(r_x, Y) + r_C \cdot C(r_x, Y)\big) \cdot Z(Y). \tag{2.17}$$

Consequently, to verify $c = \sum_{y \in H} Q_{r_x}(y)$, $\mathcal{P}$ and $\mathcal{V}$ engage in the univariate sumcheck protocol realized by the FRI low degree test (LDT). Section 5.2 presents our instantiation of the FRI LDT protocol. At the end of the FRI protocol, $\mathcal{V}$ needs oracle access to the evaluations of $Q_{r_x}(\cdot)$ at κ points $r_y \in L$, where L is an affine subspace $L \subseteq \mathbb{F}$ such that $|L| > 2|H|$ and $L \cap H = \varnothing$.

We denote the evaluations of $A(r_x, r_y)$, $B(r_x, r_y)$, and $C(r_x, r_y)$ in $Q_{r_x}(r_y)$ as $M(r_x, r_y)$ where $M \in \{A, B, C\}$ using $\mathsf{row}(i)$, $\mathsf{col}(i)$, and $\mathsf{val}(i)$ functions as the following equation:

$$M(r_x, r_y) = \frac{\mathbb{Z}_H(r_x) \cdot \mathbb{Z}_H(r_y)}{c_1^2} \cdot \sum_{i \in [n]} \frac{\mathsf{val}(i)}{(r_x - \mathsf{row}(i))\,(r_y - \mathsf{col}(i))}, \tag{2.18}$$

where we denote the second part as $C_M(r_x, r_y)$, such that

$$C_M(r_x, r_y) := \sum_{i \in [n]} \frac{\mathsf{val}(i)}{(r_x - \mathsf{row}(i))\,(r_y - \mathsf{col}(i))}. \tag{2.19}$$

Given that $\mathsf{row}(i), \mathsf{col}(i) \in H$, $r_x \in \mathbb{F} \setminus H$ and $r_y \in L$, such that $L \cap H = \varnothing$, the denominators in $C_M(r_x, r_y)$ are non-zero. To reduce the computational burden on the verifier, the task of evaluating $C_M(r_x, r_y)$ at κ points $r_y \in L$ is not performed locally. Instead, this computation is outsourced to the prover. The verifier then uses the GKR protocol to validate the results provided by the prover. In Section 5.3, we explain how we utilized the GKR protocol and designed the corresponding circuit for $C_M(r_x, r_y)$.

Adding Zero-Knowledge

The interactive protocol reveals information about the witness $\mathbf{w}$ when $\mathcal{P}$ sends evaluations of $G(r_x), \bar{A}(r_x), \bar{B}(r_x), \bar{C}(r_x)$ and $Z(\cdot)$, and invokes the low degree test on related polynomials of $((Q_{r_x}(Y))$. To prevent these "leakages" and achieve zero-knowledge, three main modifications are employed:

1. **Eliminating leakage of queries on $Z(\cdot)$.** The prover chooses a random polynomial $R_Z(\cdot)$ of degree κ and computes a $\widetilde{Z}(Y) := Z(Y) + \mathbb{Z}_H(Y) \cdot R_Z(Y)$. Even though $\widetilde{Z}(y) = Z(y)$ for $y \in H$, $\widetilde{Z}(\cdot)$ polynomial evaluations outside H preserve zero knowledge as $R_Z(\cdot)$ masks the information of polynomial $Z(\cdot)$.

2. **Modifications to the evaluations of $\bar{A}(\cdot), \bar{B}(\cdot), \bar{C}(\cdot)$.** The prover $\mathcal{P}$ samples some random polynomials $R_A(\cdot)$, $R_B(\cdot)$, $R_C(\cdot)$ of degree $|H| - 1$ and provides that

$$\widetilde{A}(X) := \sum_{y \in H} A(X, y) \cdot \widetilde{Z}(y) + \mathbb{Z}_H(X) \cdot \sum_{y \in H} R_A(y),$$

$$\widetilde{B}(X) := \sum_{y \in H} B(X, y) \cdot \widetilde{Z}(y) + \mathbb{Z}_H(X) \cdot \sum_{y \in H} R_B(y),$$

$$\widetilde{C}(X) := \sum_{y \in H} C(X, y) \cdot \widetilde{Z}(y) + \mathbb{Z}_H(X) \cdot \sum_{y \in H} R_C(y).$$

   As the $R(\cdot)$ are random, revealing the evaluations of $\widetilde{A}(\cdot), \widetilde{B}(\cdot), \widetilde{C}(\cdot)$ outside H does not leak information about the values in the witness w.

3. **Modifications to the polynomial $Q(r_x)(\cdot)$.** To uphold the zero-knowledge property of the $Q_{r_x}(\cdot)$ polynomial from Equation (2.17) in the univariate sumcheck phase, the prover $\mathcal{P}$ picks a random polynomial $S_Q(\cdot)$ of degree $2|H|+\kappa-1$, and sends an $s_1 = \sum_{y \in H} S_Q(y)$

to $\mathcal{V}$. To this, $\mathcal{V}$ responds with a random challenge $\alpha_1 \in \mathbb{F}$. $\mathcal{P}$ and $\mathcal{V}$ then run a sumcheck on the following linearised representation:

$$\alpha_1 \cdot \widetilde{c} + s_1 = \sum_{y \in H} (\alpha_1 \cdot \widetilde{Q}r_x(y) + S_Q(y)).$$

where $\widetilde{c} = r_A \cdot \widetilde{A}(r_x) + r_B \cdot \widetilde{B}(r_x) + r_C \cdot \widetilde{C}(r_x)$, referenced from Equation (2.16). This ensures $\widetilde{c}$ and s_1 can be correctly computed due to the random linear combination of the sumcheck, while revealing no information about $\widetilde{Q}r_x(\cdot)$ as it is masked by the random polynomial $S_Q(\cdot)$ [35][185] .

To obtain the full zero-knowledge protocol, we replace relevant components with their zero-knowledge versions, with the additional need for the prover $\mathcal{P}$ to commit to the random polynomials using Merkle tree commitments at the beginning, to be later opened at κ points by $\mathcal{V}$. A key advantage is that the GKR protocol remains unchanged, avoiding expensive cryptographic computations.

2.5 Additive Fast Fourier Transform

2.5.1 Cantor Additive FFT

The evaluation of a polynomial $f(x)$ of degree less than $n = 2^m$ over the subspace W_m using the Cantor algorithm is valid only when W_m is a subspace of the field (or subfield) $\mathbb{F}_{2^k}$, where $k = 2^\ell$. Furthermore, Cantor introduced a special basis to facilitate the efficient evaluation of $f(x)$. In [83, Appendix], the authors discuss how to easily construct such a basis. In the following, we will provide a more detailed discussion of this approach in a broader context.

The Cantor Special Basis

Consider the function $S : \mathbb{F}_{2^k} \to \mathbb{F}_{2^k}$ defined by $S(x) = x^2 + x$, and let the following sequence of functions be defined recursively:

$$S^0(x) = x \quad \text{and} \quad S^m(x) = S(S^{m-1}(x)).$$

A nonrecursive formula for $S^m(x)$ can be derived as $S^m(x) = \sum_{i=0}^{m} \binom{m}{i} x^{2^i}$, where $\binom{m}{i}$ denotes the binomial coefficient reduced modulo 2. Thus, when $m = 2^t$ for some t, we have $S^{2^t} = x^{2^t} + x$.

Now, for $1 < m \leq k$, assume that there exists $\beta_{m-1} \in \mathbb{F}_{2^k}$ such that $S^{m-1}(\beta_{m-1}) = 1$. We will now show that there exists a basis $\{\beta_0, \beta_1, \ldots, \beta_{m-1}\}$ for the subspace W_m such that $S(\beta_i) = \beta_{i-1}$ for $i = 1, \ldots, m-1$ with $\beta_0 = 1$. This basis is referred as the *Cantor special basis* , as mentioned in [130].

If we set $\beta_{m-2} = \beta_{m-1}^2 + \beta_{m-1}$, then $S(\beta_{m-1}) = \beta_{m-2}$. Similarly, choosing $\beta_{m-3} = \beta_{m-2}^2 + \beta_{m-2}$, and continuing this process, we obtain the following sequence of elements:

$$\begin{aligned}
\beta_{m-2} &= \beta_{m-1}^2 + \beta_{m-1}, \text{ i.e., } S(\beta_{m-1}) = \beta_{m-2} \\
\beta_{m-3} &= \beta_{m-2}^2 + \beta_{m-2}, \text{ i.e., } S(\beta_{m-2}) = \beta_{m-1} \\
&\vdots \\
\beta_0 &= \beta_1^2 + \beta_1, \text{ i.e., } S(\beta_1) = \beta_0.
\end{aligned} \tag{2.20}$$

Thus, we have $S^{m-1}(\beta_{m-1}) = \beta_0$, which implies that $\beta_0 = 1$.

Now, we will demonstrate that the set $\{\beta_0, \beta_1, \ldots, \beta_{m-1}\}$ forms a basis. While a proof of this can be found in [130, Appendix], we provide a detailed proof below for the sake of completeness.

Theorem 2.2. *Let $\{\beta_0, \beta_1, \ldots, \beta_{m-1}\}$ be a set of elements in $\mathbb{F}_{2^k}$ such that*

$$\beta_0 = 1 \text{ and } S(\beta_i) = \beta_i^2 + \beta = \beta_{i-1} \text{ for } i = 1, \ldots, m-1.$$

Then, the set $\{\beta_0, \beta_1, \ldots, \beta_{m-1}\}$ is linearly independent over $\mathbb{F}_2$.

Proof. We will use the mathematical induction to prove this. If $c_1 \cdot \beta_1 = 0$, then we must have $c_1 = 0$. This implies that the result is true for $i = 1$.

Now suppose that the result is true for $i = \ell$, i.e., $\beta_0, \beta_1, \ldots, \beta_\ell$ are linearly independent over $\mathbb{F}_2$. We need to show that $\beta_0, \beta_1, \ldots, \beta_\ell, \beta_{\ell+1}$ are also linearly independent over $\mathbb{F}_2$.

If possible, let $\beta_0, \beta_1, \ldots, \beta_\ell, \beta_{\ell+1}$ are linearly dependent over $\mathbb{F}_2$. So there exists some scalars c_i, not all zero, such that

$$\begin{aligned}
&c_0 \cdot \beta_0 + c_1 \cdot \beta_1 + \cdots + c_\ell \cdot \beta_\ell + c_{\ell+1} \cdot \beta_{\ell+1} = 0 \\
\Longrightarrow\ &S(c_0 \cdot \beta_0 + c_1 \cdot \beta_1 + \cdots + c_\ell \cdot \beta_\ell + c_{\ell+1} \cdot \beta_{\ell+1}) = 0 \\
\Longrightarrow\ &c_0 \cdot S(\beta_0) + c_1 \cdot S(\beta_1) + \cdots + c_\ell \cdot S(\beta_\ell) + c_{\ell+1} \cdot S(\beta_{\ell+1}) = 0 \\
\Longrightarrow\ &c_1 \cdot \beta_0 + \cdots + c_\ell \cdot \beta_{\ell-1} + c_{\ell+1} \cdot \beta_\ell = 0 \\
\Longrightarrow\ &\beta_0 = 1, \beta_1, \ldots, \beta_\ell \text{ are linearly dependent}
\end{aligned}$$

Therefore, $\beta_0 = 1, \beta_1, \ldots, \beta_{m-1}$ are linearly independent. □

Therefore, we have a basis $\{\beta_0 = 1, \beta_1, \ldots, \beta_{m-1}\}$ for W_m such that $S(\beta_i) = \beta_{i-1}$ for $i = 1, 2, \ldots, m-1$. Also, from the set of Equations 2.20, we can also check that $S^i(\beta_i) = 1$ for $i = 0, 1, \ldots, m-1$. Since $\mathbb{Z}_{W_1} = x^2 + x = S(x)$, we have

$$\mathbb{Z}_{W_2}(x) = (S(x))^2 + S(\beta_1) \cdot S(x) = S^2(x)$$

Similarly, we can show that $\mathbb{Z}_{W_3}(x) = S^3(x)$ and so on. Thus, for the Cantor special basis, we have $\mathbb{Z}_{W_i}(x) = S^i(x)$ for $i = 0, 1, \ldots, m$.

Thus, the coefficients of the vanishing polynomials $\mathbb{Z}_{W_i}(x)$ are always equal to 1. Additionally, from (2.20), we have

$$S^i(\beta_{i+\ell}) = \beta_\ell \tag{2.21}$$

for any $i, \ell \geq 0$ with $i + \ell \leq m-1$.

We know that for the Cantor special basis, $\mathbb{Z}_{W_i}(x) = S^i(x)$ for $i = 0, 1, \ldots, m$. Thus, the polynomials $S^i(x)$ are linearized polynomials with coefficients in $\mathbb{F}_2$. Hence, for any $\theta \in \mathbb{F}_{2^k}$, we have the property $S^i(x+\theta) = S^i(x) + S^i(\theta)$, which means that in $S^i(x+\theta)$, there may be at most one term that is not equal to 1. This makes the Cantor algorithm highly efficient when performing polynomial division with the polynomials $S^i(x+\theta)$. Now, we are ready to explain the Cantor algorithm.

When $k = 2^\ell$ for some ℓ: In this case, we have $k-1 = 2^\ell - 1$. Choose an element β_{k-1} from the field (or subfield) $\mathbb{F}_{2^k}$ such that $\mathrm{Tr}_{\mathbb{F}_{2^k}/\mathbb{F}_2}(\beta_{k-1}) = 1$, i.e.,

$$\beta_{k-1} + \beta_{k-1}^2 + \beta_{k-1}^{2^2} + \cdots + \beta_{k-1}^{2^{k-1}} = 1 \implies S^{k-1}(\beta_{k-1}) = 1. \tag{2.22}$$

Thus, as described above, we can find a set of Cantor special basis $B = \{\beta_0, \beta_1, \ldots, \beta_{k-1}\}$ of W_k. Now for an m-dimensional subspace, with $m \leq k$, of $\mathbb{F}_{2^k}$, we will select the first m elements $\beta_0, \beta_1, \ldots, \beta_{m-1}$ from the set B. Note that from Property (iii) of trace function, we know that exactly half of the elements have a trace value of 1. Therefore, we can easily find an element β_{k-1} such that $\mathrm{Tr}_{\mathbb{F}_{2^k}/\mathbb{F}_2}(\beta_{k-1}) = 1$.

Cantor FFT Algorithm

Let $f(x) \in \mathbb{F}_{2^k}$ be a polynomial of degree less than $n = 2^m$ and we want to evaluate $f(x)$ over the affine subspace $\theta + W_m = \theta + \langle \beta_0, \beta_1, \ldots, \beta_{m-1} \rangle$, where $\{\beta_0 = 1, \beta_1, \ldots, \beta_{m-1}\}$ is a Cantor special basis. The evaluation of $f(x)$ using the Cantor algorithm proceeds as follows: First, compute two polynomials $f_0(x)$ and $f_1(x)$ such that $f_0(x) = f(x)$ for all $x \in \theta + W_{m-1}$ and $f_1(x) = f(x)$ for all $x \in \theta + \beta_{m-1} + W_{m-1}$. Since the affine subspaces $\theta + W_{m-1}$ and $\theta + \beta_{m-1} + W_{m-1}$ correspond to the roots of the polynomials $\mathbb{Z}_{W_{m-1}}(x+\theta) = S^{m-1}(x+\theta)$ and $\mathbb{Z}_{W_{m-1}}(x+\theta+\beta_{m-1}) = S^{m-1}(x+\theta+\beta_{m-1})$, respectively, the polynomials $f_0(x)$ and $f_1(x)$ can be obtained by taking the remainders of $f(x)$ when divided by these polynomials. Specifically, we have

$$f_0(x) = f(x) \mod S^{m-1}(x+\theta) \quad \text{and} \quad f_1(x) = f(x) \mod S^{m-1}(x+\theta+\beta_{m-1}).$$

Each polynomial $f_0(x)$ and $f_1(x)$ has degree less than 2^{m-1}. We then proceed by recursively evaluating $f_0(x)$ and $f_1(x)$ over the affine subspaces $\theta + W_{m-1}$ and $\theta + \beta_{m-1} + W_{m-1}$, respectively. The recursion continues until all the resulting polynomials $f_0(x)$ and $f_1(x)$ are constants. This is summarized in Algorithm 2.1.

Algorithm 2.1 Cantor additive FFT of length $n = 2^m$

```
Require: $f(x) \in \mathbb{F}_{2^k}[x]$ of degree $< n = 2^m$, where $k = 2^\ell$, and the affine subspace
    $\theta + W_m = \theta + \langle \beta_0, \beta_1, \ldots, \beta_{m-1} \rangle$, where $\{\beta_0 = 1, \beta_1, \ldots, \beta_{m-1}\}$ is a Cantor special
    basis.
Ensure: FFT($f, \theta + W_m$).
 1: if $m = 0$ then
 2:     return $f(\theta)$
 3: end if
 4: Compute:
        $f_0(x) = f(x) \mod S^{m-1}(x+\theta)$,
        $f_1(x) = f(x) \mod S^{m-1}(x+\theta+\beta_{m-1})$.
 5: return FFT($f_0, \theta + W_{m-1}$) || FFT($f_1, \theta + \beta_{m-1} + W_{m-1}$)
```

2.5.2 Gao-Mateer Additive FFT

In [83], Gao and Mateer propose two algorithms for computing the additive FFT using Taylor expansion. In this section, we will focus on the first algorithm, which applies to lengths $n = 2^m$ for any arbitrary m. It is also important to note that this algorithm is originally formulated for a subspace $W_m = \langle \beta_0, \beta_1, \ldots, \beta_{m-1} \rangle$, but here we will discuss its application over an affine subspace $\theta + W_m$.

Consider the subspace $G = \langle \gamma_0, \gamma_1, \ldots, \gamma_{m-2} \rangle$, where $\gamma_i = \beta_i \cdot \beta_{m-1}^{-1}$ for $i = 0, 1, \ldots, m-2$, and $D = \langle \delta_0, \delta_1, \ldots, \delta_{m-2} \rangle$, where $\delta_i = \gamma_i^2 + \gamma_i$. Since $\beta_0, \beta_1, \ldots, \beta_{m-1}$ are linearly independent, we will now show that the set $\{\gamma_0, \gamma_1, \ldots, \gamma_{m-2}, 1\}$ and $\{\delta_0, \delta_1, \ldots, \delta_{m-2}\}$ also form a basis.

Theorem 2.3. *Let $\beta_0, \beta_1, \ldots, \beta_{m-1} \in \mathbb{F}_{2^m}$ are linearly independent over $\mathbb{F}_2$. Then, for any non-zero element $\alpha \in \mathbb{F}_{2^m}$, the elements $\beta_0 \cdot \alpha^{-1}, \beta_1 \cdot \alpha^{-1}, \ldots, \beta_{m-1} \cdot \alpha^{-1}$ are also linearly independent.*

Proof. Suppose for any scalars $c_0, c_1, \ldots, c_{m-1}$, we have

$$\begin{aligned} & c_0\beta_0\alpha^{-1} + c_1\beta_1\alpha^{-1} + \cdots + c_{m-1}\beta_{m-1}\alpha^{-1} = 0 \\ \iff & c_0\beta_0 + c_1\beta_1 + \cdots + c_{m-1}\beta_{m-1} = 0. \end{aligned}$$

Thus, $\beta_0, \beta_1, \ldots, \beta_{m-1}$ are linearly independent if and only if $\beta_0\alpha^{-1}, \beta_1\alpha^{-1}, \ldots, \beta_{m-1}\alpha^{-1}$ are also linearly independent. □

Corollary 2.1. *G is a subspace of dimension $m-1$.*

Proof. We have $G = \langle \gamma_0, \gamma_1, \ldots, \gamma_{m-2} \rangle$, where $\gamma_i = \beta_i \cdot \beta_{m-1}^{-1}$ for $i = 0, 1, \ldots, m-1$. Now, since $\beta_0, \beta_1, \ldots, \beta_{m-1}$ are linearly independent and $\beta_{m-1} \neq 0$, by Theorem 2.3, $\gamma_0, \gamma_1, \ldots, \gamma_{m-2}$ are also linearly independent. Therefore, G is a subspace of dimension $m-1$. □

Theorem 2.4. *D is a subspace of dimension $m-1$.*

Proof. We have $D = \langle \delta_0, \delta_1, \ldots, \delta_{m-2} \rangle$, where $\delta_i = \gamma_i^2 + \gamma_i$. We will show that $\delta_0, \delta_1, \ldots, \delta_{m-2}$ are also linearly independent. Suppose, for the sake of contradiction, that $\delta_0, \delta_1, \ldots, \delta_{m-2}$ are linearly dependent. So there exists some scalars $c_0, c_1, \ldots, c_{m-2}$ such that some $c_i \neq 0$ and

$$\begin{aligned} & c_0\delta_0 + c_1\delta_1 + \cdots + c_{m-2}\delta_{m-2} = 0 \\ \implies & c_0(\gamma_0^2 + \gamma_0) + c_1(\gamma_1^2 + \gamma_1) + \cdots + c_{m-2}(\gamma_{m-2}^2 + \gamma_{m-2}) = 0 \\ \implies & c_0 \cdot S(\gamma_0) + c_1 \cdot S(\gamma_1) + \cdots + c_{m-2} \cdot S(\gamma_{m-2}) = 0 \\ \implies & S(c_0\gamma_0 + c_1\gamma_1 + \cdots + c_{m-2}\gamma_{m-2}) = 0 \quad \text{[since } S \text{ is linearized over } \mathbb{F}_2\text{]} \\ \implies & c_0\gamma_0 + c_1\gamma_1 + \cdots + c_{m-2}\gamma_{m-2} \in \{0, 1\} \end{aligned}$$

Now, $c_0\gamma_0 + c_1\gamma_1 + \cdots + c_{m-2}\gamma_{m-2} = 0$ implies $\gamma_0, \gamma_1, \ldots, \gamma_{m-2}$ are linearly dependent which is a contradiction. Again, $c_0\gamma_0 + c_1\gamma_1 + \cdots + c_{m-2}\gamma_{m-2} = 1$ implies that $\gamma_0, \gamma_1, \ldots, \gamma_{m-2}, 1$ are linearly dependent which is again a contradiction. Thus, $\delta_0, \delta_1, \ldots, \delta_{m-2}$ are linearly independent over $\mathbb{F}_2$. Therefore, D is a subspace of dimension $m-1$. □

Now, we are ready to discuss the Gao-Mateer algorithm. Consider the function $g(x) = f(\beta_{m-1}x)$. Therefore, the evaluation of $f(x)$ over $\theta + W_m$ is equivalent to the evaluation of $g(x)$ over

$$\begin{aligned}\beta_{m-1}^{-1}(\theta + W_m) &= \beta_{m-1}^{-1}\theta + \beta_{m-1}^{-1} \cdot W_m \\ &= (\theta_0 + G) \cup (1 + \theta_0 + G),\end{aligned}$$

where $\theta_0 = \beta_{m-1}^{-1}\theta$. Therefore, we have

$$\mathrm{FFT}(f, \theta + W_m) = \mathrm{FFT}(g, \theta_0 + G) \parallel \mathrm{FFT}(g, 1 + \theta_0 + G).$$

For $\beta = \theta_0 + \alpha \in \theta_0 + G$, let $\beta' = (\theta_0 + \alpha)^2 + \theta_0 + \alpha$ be the corresponding element in $\theta_0^2 + \theta_0 + D$. Suppose that the Taylor expansion [1] of $g(x)$ at $x^2 + x$ is

$$g(x) = \sum_{i=0}^{\ell-1}(g_{i0} + g_{i1}x)(x^2 + x)^i, \text{ where } \ell = 2^{m-1} \text{ and } g_{ij} \in \mathbb{F}_{2^k}. \tag{2.23}$$

Consider the two polynomials $f_0(x) = \sum_{i=0}^{\ell-1} g_{i0}x^i$ and $f_1(x) = \sum_{i=0}^{\ell-1} g_{i1}x^i$.

Now, for any $\beta = \theta_0 + \alpha \in \theta_0 + G$ and $b \in \mathbb{F}_2$, we have $(b + \beta)^2 + b + \beta = \beta^2 + \beta = \beta'$. Thus, from (2.23), we have

$$g(b + \beta) = f_0(\beta') + \beta \cdot f_1(\beta') + b \cdot f_1(\beta'). \tag{2.24}$$

Hence, the FFT of $g(x)$ over $\beta_{m-1}^{-1}(\theta + W_m)$ can be obtained from the FFT of $f_0(x)$ and $f_1(x)$ over $\theta_0^2 + \theta_0 + D$. Let

$$\begin{aligned}&\mathrm{FFT}(f_0, \theta_0^2 + \theta_0 + D) = (u_0, \ldots, u_{\ell-1}) \text{ and} \\ &\mathrm{FFT}(f_1, \theta_0^2 + \theta_0 + D) = (v_0, \ldots, v_{\ell-1}).\end{aligned}$$

Thus, from (2.24), we have

$$\begin{aligned}&\mathrm{FFT}(g, \theta_0 + G) = (u_0 + \eta_0 v_0, \ldots, u_{\ell-1} + \eta_{\ell-1}v_{\ell-1}) \quad \text{and} \\ &\mathrm{FFT}(g, 1 + \theta_0 + G) = ((u_0 + \eta_0 v_0) + v_0, \ldots, (u_{\ell-1} + \eta_{\ell-1}v_{\ell-1}) + v_{\ell-1}),\end{aligned}$$

where η_i denotes the i-th element of $\theta_0 + G$.

By applying this reduction step again to $\mathrm{FFT}(f_0, \theta_0^2 + \theta_0 + D)$ and $\mathrm{FFT}(f_1, \theta_0^2 + \theta_0 + D)$, we continue until D has a dimension of 1. This is summarized in Algorithm 2.2.

[1] We provide a detailed discussion of the Taylor expansion in Section 4.3.1.

Algorithm 2.2 Gao-Mateer additive FFT of length $n = 2^m$

```
Require: $f(x) \in \mathbb{F}_{2^k}[x]$ of degree $< n = 2^m$ and the affine subspace $\theta + W_m = \theta + \langle \beta_0, \beta_1, \ldots, \beta_{m-1} \rangle$.
Ensure: $\mathrm{FFT}(f, \theta + W_m)$.
 1: if $m = 1$ then
 2:     return $(f(\theta), f(\theta + \beta_0))$
 3: end if
 4: Compute $g(x) = f(\beta_{m-1}x)$
 5: Compute the Taylor expansion of $g(x)$ at $x^2 + x$ to get $f_0(x)$ and $f_1(x)$.
 6: Compute $\gamma_i = \beta_i \cdot \beta_{m-1}^{-1}$ and $\delta_i = \gamma_i^2 + \gamma_i$ for $0 \le i \le m-2$.
 7: Let $\theta_0 = \beta_{m-1}^{-1}\theta$. Consider the affine subspaces:
        $\theta_0 + G = \theta_0 + \langle \gamma_0, \ldots, \gamma_{m-2} \rangle,$
        $\theta_0^2 + \theta_0 + D = \theta_0^2 + \theta_0 + \langle \delta_0, \ldots, \delta_{m-2} \rangle$
 8: Let $\ell = 2^{m-1}$ and compute:
        $\mathrm{FFT}(f_0, \theta_0^2 + \theta_0 + D) = (u_0, \ldots, u_{\ell-1}),$
        $\mathrm{FFT}(f_1, \theta_0^2 + \theta_0 + D) = (v_0, \ldots, v_{\ell-1})$
 9: for $i = 0$ to $\ell - 1$ do
10:     $\omega_i \leftarrow u_i + \eta_i \cdot v_i$
11:     $\omega_{\ell+i} \leftarrow \omega_i + v_i$
12: end for
13: return $(\omega_0, \ldots, \omega_{n-1})$
```

Chapter 3

Literature Review

Declaration of Contributions

Section 3.1 of this chapter includes material from an unpublished work that I co-authored. Additionally, other sections incorporate content from [17, 16, 18], all of which I authored independently.

3.1 Additive FFT Algorithms

The fast Fourier transform (FFT) over additive groups, known as additive FFT, over finite fields was developed in the late 1980s. Wang and Zhu [177] first introduced this concept in 1988, followed independently by Cantor [56] in 1989. These algorithms evaluate polynomials at the roots of the vanishing polynomial corresponding to a subspace or an affine subspace. In [56], Cantor introduced an FFT algorithm for the evaluation of a polynomial $f(x)$ of degree less than $n = 2^m$ over an m-dimensional affine subspace of the field (or subfield) $\mathbb{F}_{2^k}$, with $k = 2^\ell$ for some ℓ. This foundational work was later generalized to accommodate any arbitrary values of k by von zur Gathen and Gerhard [84], although their approach incurs a greater operational cost. Subsequently, Gao and Mateer [83] proposed two algorithms for computing the additive FFT utilizing the concept of Taylor expansion. The first algorithm is applicable to any arbitrary k and requires fewer operations compared to the method developed by von zur Gathen and Gerhard. Their second algorithm, designed for the additive FFT of length 2^m, where m is a power of 2, matches Cantor's in the number of multiplications while reducing the number of additions.

In 2014, Lin, Chung, and Han [124] designed a new polynomial basis, referred to as the LCH basis, constructed using the subspace polynomials over $\mathbb{F}_{2^k}$ to implement FFT. Their evaluation algorithm takes the coefficients of a polynomial in the LCH basis as input and operates with a complexity of $O(n \log n)$ additions and $O(n \log n)$ multiplications. Later, Lin et al. [123] addressed the challenge of converting between the LCH basis and the monomial basis. They proposed algorithms that perform this conversion with $O(n \log n \log \log n)$ additions and no multiplications, specifically when the subspace is generated by the Cantor special basis. The LCH additive FFT for Cantor bases has since been applied to binary polynomial multiplication in several works [63, 121]. It is important to note that when applying the additive FFT using these algorithms, for a polynomial with $(t + 1)$ coefficients (i.e., of degree t), if $t + 1$ is less than 2^m, zero-padding is required to make its size 2^m. In [42], Bernstein et al. generalized the algorithm of Gao-Mateer a to avoid wasting time on manipulating coefficients that are known to be zero. Later, Bernstein and Chou [41] introduced additional improvements to the algorithm by considering the Cantor basis and the use of tower field construction. These improvements are primarily discussed for FFT of lengths up to 64.

3.2 Blockchain Based Privacy Preserving Protocols

Bitcoin [136] is the first blockchain platform provides an authenticated tamper-proof record of transactions maintained in a peer-to-peer network. Transactions on Bitcoin are linked together. Many research works such as [150, 153, 132] showed that analyzing the transaction graphs, values, and dates of transactions helps to retrieve information that leads to attribute ownership of Bitcoin addresses. Consequently, the identity of the owners could be revealed easier than was expected. Researchers have proposed privacy-focused solutions, such as CoinJoin [131], in response to privacy concerns. However, these solutions do not necessarily provide complete privacy guarantees.

Zerocash [31] is a cryptocurrency that shares similarities with Bitcoin in terms of its blockchain technology. However, Zerocash uses zero-knowledge proofs (ZKPs) to break the link between the input of a new transaction and the unspent transaction output (UTXO) of an existing transaction on the blockchain, thereby providing enhanced privacy. Zerocash can be implemented on top of Bitcoin or other blockchains, and the use of private transactions is optional. The core of Zerocash's privacy features centers around two primary algorithms: Mint and Pour. The Mint algorithm allows a user to convert any coin value they own into a committed coin, which is then securely stored within a Merkle tree [134]. This tree maintains all such commitments. When a user wishes to transfer the committed coins, they utilize the Pour algorithm. This process enables the user to spend up to two of their committed coins, creating up to two new coin commitments or making them directly spendable as public values. The recipient of these coins is specified within the algorithm, ensuring that only they can spend these new commitments. The brilliance of the Pour algorithm lies in its ability to maintain the anonymity of both the sender and receiver. It does so by concealing the commitments being spent and encrypting the details of the new commitments with the receiver's public keys. Additionally, it breaks the traceable link between the spent coins and the newly created commitments, further enhancing privacy.

With the possibility of implementing the idea of smart contracts on Blockchain platforms (such as Ethereum [54]), we can see an increasing growth in the implementation of various financial decentralized applications (dApps) on Blockchain. Transferring financial activities to the blockchain platforms without paying attention to the privacy of users can have worrying consequences. However, Zerocash lacks support for smart contracts, which limits its adaptability for diverse applications, such as SCM. To redesign Zerocash to accommodate SCM application, one would need to develop a new layer-1 blockchain. Achieving the necessary widespread adoption for this blockchain to maintain its security is a considerable challenge, rendering such a redesign impractical. Because of this, many efforts have been made to provide methods to protect people's privacy in the blockchain. Such as, Hawk [117], Zexe [48], ZeeStar [163], etc. These methods are general approaches that aim to preserve the privacy of users.

Hawk [117] is a smart contract compiler that enables users to interact with a smart contract in a secure manner. Hawk developers implemented zkSNARKs protocols on Ethereum smart contracts. Hawk incorporates the privacy-enhancing Mint and Pour algorithms, originally from Zerocash, into smart contracts. Users wishing to execute a computation (e.g., an auction) first commit their native tokens (e.g., ETH) using the Mint algorithm in the Hawk smart contract. They then utilize the Pour algorithm to allocate a portion of these committed tokens for specific computations. Subsequently, users reveal their private inputs to a trusted manager, who executes the computation off-chain and publishes the results on the blockchain. This process ensures the confidentiality of the users' inputs while enabling the execution of complex computations. This allows the use of native cryptocurrency on smart contracts anonymously by hiding money transfers within a smart contract. However, this approach is not suitable for many dApps. For example, where users need to transfer other tokens on Ethereum anonymously, or they want to employ functionalities of smart contracts other than token transformations (Pour). Additionally, Hawk requires a protocol manager.

Bowe et. al presented Zexe [48] that addresses privacy and scalability in Ethereum blockchain. They propose decentralized private computation (DPC) scheme which extends

Zerocash. Zexe leverages DPC to enable offline computation. The offilie computations produces publicly-verifiable transactions that prove correctness of these offline executions. However, it requires cryptographic expertise for implementing new applications [163] and a separate trusted setup for each application [183]. Xiong et al. [183] proposed a new DPC scheme called VeriZexe which needs only one single universal setup to be able to support any number of applications.

ZeeStar [163] is a compiler that converts a smart contract into an anonymous smart contract where computations are performed off the blockchain on the user side. The encrypted values are then assigned to variables on the smart contract, ensuring that no one knows the actual values of the variables. To verify the accuracy of the computation done by the user, a zero-knowledge proof is uploaded along with the newly encrypted variables. This means that the new state of the smart contract is computed off chain and its correctness is verified by nodes on the Ethereum platform. To achieve this capability, the user must rewrite the smart contract and add privacy annotations to the code. Accordingly, the user specifies which parts of the code should be private and which parts can be public. ZeeStar [163] provides more applicability than Hawk [117]. However, it still needs to verify the transaction sender according to its address to determine if the sender has permission to execute the rest of the called function. Therefore, using address-based authentication to verify the transaction sender of a smart contract can lead to significant information leakage.

Narula et. al [137] proposed zkLedger which is a private but also audiable transaction ledger. Their focus is on banks that want their transactions be encrypted; but, let regulators have an insight into bank assets and trades. Auditor can query the bank and have a verifiable proof that the answer actually is true. zkLedger leverages Peterson commitment [145] for committing values. Peterson commitment enables linear functions over transaction values. For example, ratios, percentages, sums, averages, etc. zkLedger leverages zero-knowledge proofs to ensure that banks have valid transactions, i.e., they had consent to transfer the asset, they had enough asset to transfer, and the assets are neither created or destroyed.

3.3 Supply Chain Management

A supply chain management (SCM) is a system used by a business and its suppliers to make and deliver certain products to the customers. An SCM consists of different components such as inventory management, transportation and logistics, and supply chain analytics. In SCM schemes, it is necessary to monitor the history of each product to gain visibility of the product's flow from producers to consumers. At each stage, entities possessing the product must have access to its history to trace its origins. This traceability ensures the product's quality and compliance with ethical standards [167, 98, 4, 149]. Data is either captured by the Internet of things (IoT) sensors connected to products or entered manually. In the first case, the data is transmitted via different communication protocols, such as radio-frequency identification (RFID) [171, 168], ZigBee sensors [154], or Bluetooth low energy (BLE) [96] to receivers. Then, the manually entered or sensor-captured data are aggregated for real-time analysis or future research. The collected data provides valuable information for business owners. Also, it eases the mind of the final customers about their product. For example, whether a wild-caught salmon is really caught from a lake, river, etc. or it is just a farm-raised salmon.

In supply chains, the ownership of products often changes as the products move through the chain. To accommodate this, ownership models has been developed for RFID tags that enables the current owner to authenticate the tag and transfer its ownership. For example, numerous research works, such as [43, 64, 184], focus on transferring the ownership of RFID tags while maintaining privacy. However, the process of uploading measured data from the products still can potentially compromise the privacy of the product owners. It is crucial to protect personal details and trade secrets within the supply chain. This includes maintaining confidentiality about trading partners and involvement in specific supply chains. Moreover, an SCM necessitates enabling trusted interoperability among various entities, not only within a predetermined supply chain but also fostering cooperation between different supply chains [143]. This is particularly crucial in the face of unforeseen situations (e.g., pandemics and natural disasters) where the transiliency of a supply chain becomes paramount [69, 157]. Transiliency is defined as the ability to both return to the original form and simultaneously undergo transformation (through adaptation and innovation) when facing disruptions, is essential for maintaining continuity and efficiency in supply chains [69]. An SCM requires the establishment of trustworthy collaborations with Micro, Small and Medium Enterprises (MSMEs), which are defined as firms with fewer than 300 employees [104]. The involvement of MSMEs in SCMs offers several benefits, including transparency, traceability, resilience, and sustainability [181, 11, 173].

Traditional SCM systems have encountered numerous challenges, particularly in achieving comprehensive transparency and visibility across their operations. The absence of effective traceability measures, lead to significant risks of fraud and product mislabeling. For instance, a study conducted by Shehata et al. (2019) reports a mislabeling rate of 32.3% among targeted finfish species within the supply chain in southern Ontario, Canada [159]. Furthermore, research by Lechmere (2016) shows that up to one in five bottles of fine wine in the market may be counterfeit [119]. These findings highlight the critical need for enhanced traceability and verification mechanisms within an SCM to mitigate such risks. The heterogeneity of SCM platforms, the use of independent centralized databases, and the adoption of different data standards by various entities within a supply chain significantly limit transparency and traceability. Additionally, mistrust among supply chain partners further impedes collaboration [100]. Moreover, payments in current SCM systems are centrally managed by the owners according to invoices from their business partners, a process that is both error-prone and difficult, requiring entities to trust the centralized authority [79]. The lack of agility in inter-SCM collaboration also renders these networks vulnerable during crises. For instance, the COVID-19 pandemic's disruption of manufacturing and logistics activities highlighted the need for supply chains to rapidly employ alternative resources. Examples include converting passenger airplanes to carry freight in their belly cargo, consolidating freight, and implementing on-site storage solutions [165]. Furthermore, MSMEs face greater challenges than large companies in integrating with larger SCMs. They often depend on their larger partners for relation-based rents, profits generated jointly in an exchange relationship [71, 11]. Additionally, due to their lesser-known reputations, MSMEs frequently struggle to gain the trust of other companies [11].

Supply chains' data records have a sequential format since they represent the conditions of a product over time. Furthermore, two distinct supply chains might merge due to various reasons, such as using the same warehouse, the same transportation, or assembly in the manufacturing stage. In contrast, a single supply chain can be divided into two or more sub supply chains for reasons like the distribution of a product to different locations. Consequently, directed acyclic graphs (DAGs) are an optimal data structure for storing supply chain data records. For instance, a food supply chain can be seen as a DAG, where each node in the DAG signifies a data record captured by an entity responsible for moving, storing, or processing batches of food over a period of time.

3.3.1 Blockchain in Supply Chain Management

Applying blockchain technologies to secure SCM is a promising approach. Blockchain removes the need for trusting third parties; Moreover, the potential benefits that blockchain and IoT can give to SCM, such as traceability, transparency, less paper works and less code of conduct violation and fraud [6]. Blockchain platforms can be divided into two categories: permissioned and permissionless blockchains.

In permissioned blockchains, the number of participants is limited, all parties are known, and there is no anonymity. Permissioned blockchains, while providing visibility and automation for supply chain owners, impose certain limitations on their partners, who are required to adhere to the owner's platforms, protocols, and standards. This requirement undermines the agility necessary for maintaining a sustainable SCM system; notably, integrating a new partner into the system demands a time-consuming process to align with these protocols. Furthermore, the cost of permissioned blockchain solutions can be prohibitively expensive. For example, the IBM Food Trust's [103] minimum featured solution, accommodating up to five supply chain partners, costs USD 2,000 per month [3]. IBM Food Trust leverages Hyperledger Fabric [76] which is a permissioned blockchain and one of the Hyperledger projects hosted by The Linux Foundation. In Hyperledger Fabric, membership service provider (MSP) registers member who can publish and share information. Permissioned data access is an essential part of IBM Food Trust. Walmart collaborated with IBM in 2016 to build a permissioned blockchain-based system to trace the origin and transportation of food goods [101]. Their blockchain platform is also build on Hyperledger Fabric to trace over 25 products from 5 different suppliers. Blockchain technology has enabled Walmart to track a food item from the store back to its source within seconds. For instance, with the use of blockchain, the time required to trace the origin of mangoes in the U.S. has been reduced from 7 days to just 2.2 seconds [80]. However, the lack of flexibility, challenge the practicality and scalability of permissioned blockchains in dynamic supply chain environments.

Permissionless (public) blockchains provide a fully decentralized and trustless environment, enabling a wide array of global entities to collaborate, with this collaboration facilitated by smart contracts. Smart contracts offer financial guarantees through their transparent and immutable execution, significantly aiding MSMEs in building trust with their partners. By automating contract enforcement, these digital agreements ensure that all parties fulfill their obligations, such as timely payments or delivery of services or products, thereby mitigating risks associated with traditional contracts [5]. Furthermore, the integration of entities into the SCM system is facilitated by the transparent rules established by smart contracts, making the SCM more agile. Moreover, public blockchains offer enhanced security against 51% attacks. This accessibility, transparency and security pave the way for developing more sustainable solutions, making public blockchains an attractive option for future SCM innovations. This category of blockchains can provide a higher level of transparency and robustness. Also, they do not need a centralized party to grant or revoke permissions. Bitcoin and Ethereum are two well-known examples of permissionless blockchains. Permissionless blockchains can let any user participate as an entity in a supply chain. One problem in permissionless blockchains is that data storage on the blockchain is very expensive. So in some approaches data is stored on decentralized data storages like interplanetary file system (IPFS) [38].

Tracr™ [2] is an example of an SCM that uses the public blockchain Ethereum. Tracr has been introduced by the De Beers Group to monitor diamonds from the mining stage, through cutting and polishing, and finally to the jewellers. This system provides tamper-proof assurance of the diamond's source [118]. Musamih et al. [135] proposed blockchain-based system for pharmaceutical supply chain to address counterfeit drugs issue that is one of the consequences of complex healthcare supply chains structures. They use smart contract over Ethereum platform to define different functions of the different stakeholders (entities) in a supply chian. Such as initializing (manufacturing) a Food and Drug Administration (FDA) approved drug on blockchain. In their approach the manufacturer and distributors can update data captured by IoT devices on the blockchain. Their scheme provides the capability of uploading product images to the IPFS, where the hash of the image is published on Ethereum blockchain. The smart contract authenticate stakeholders using their Ethereum address.

Salah et al. [155] proposed an SCM solution based on the Ethereum Blockchain and IPFS to solve the problem of traceability in the agricultural supply chain where it is difficult to track and trace products in centralized controlled supply chain. Accordingly, in the event of contamination, identifying the source will be easier in blockchain based SCM solution. They employ Ethereum smart contracts to automate and enforce the rules and regulations of the supply chain and execute specific actions automatically when certain conditions are met. In their approach details of the product is captured and saved on IPFS as images. The details can be the time-stamped corp growth images. Hash of the stored file in IPFS is stored in the smart contract. The authentication of entities in their proposed approach is based on Ethereum address.

Toyoda et al. [172] proposed a novel product ownership management system (POMS). They have implemented POMS on a Ethereum smart contract. In their proposed approach, the manufacturer assign Electronic Product Code (EPC) to each product and write that into the RFID tag attached to the product. The product's EPC is registered on the smart contract. Furthermore, functions such as product owner transformation, incentivising entities to follow POMS protocol, and unauthorized party prevention are enabled by their smart contract. Toyoda et al. claimed PMOS system makes counterfeiters' efforts to clone real tags ineffective.

3.3.2 Blockchain Based Privacy Presetving Solutions for Supply Chain Management

Privacy, especially against public exposure, is crucial in SCM applications. It encompasses entities' anonymity—ensuring that auditors reviewing the history of a product cannot identify the creator of a specific data record unless the entity chooses to reveal its identity. Additionally, privacy involves unlinkability among entities, meaning that businesses desire to keep their partners and the supply chains they are involved in confidential. Our analysis reveals that all of the privacy preserving SCM solutions reviewed are not practical to be employed as SCM systems on public blockchains.

The importance of privacy preservation in SCMs fuels research into permissioned blockchains such as [7, 120] that offer restricted access and rely on a centralized membership service provider [10]. However, preserving the entities' anonymity while verifying the authenticity of the data they have uploaded is an important issue that needs to be resolved in permissionless blockchains. Many SCM schemes developed over permissionless blockchains, such as those mentioned above, do not preserve the privacy of their participants. Each entity has to interact with the blockchain ledger through a generated address for invoking smart contract functions and covering gas fees with the blockchain's native token (e.g., ETH), which compensate for the processing and validation of transactions. Entities typically obtain the required tokens through exchanges enforcing know your customer (KYC) protocols, or they may acquire them from other recognized addresses, with or without the involvement of an intermediary. This requirement introduces a risk: through detailed analysis of blockchain transactions, it is possible to infer the identity of the entities involved [175, 186]. Given the transparency of transactions on permissionless blockchains, these schemes could potentially compromise participants privacy. Following reviews some privacy preservation methods over public blockchains.

AlTway et al. presented Mesh [8], a supply chain solution over Ethereum smart contracts. Mesh uses group signatures to preserve the privacy of participant and employs a forward secrecy approach to preserve the confidentiality of data. Both are required to protect business secrets. However, Mesh requires a centralized server to keep the protocol going. Mesh uses Petersen's group signature [145] to authorize members of a supply chain. For each supply chain, all entities that are involved in that supply chain must build a group. The group must have a group manager which sends a list of identities of all group members to the Mesh Server and the Mesh's SupplyChain smart contract (C_{SC}) instance. For each smart contract, a separate instance of C_{SC} should be created. Only the group members of a supply chain are able to upload data to the related C_{SC}. The data is stored on Ethereum's Blockchain. To limit the upload access to the group members, Altawy et al. [8] use Petersen's group signature [145] for members' authorization. Consequently, C_{SC} will be able to verify the membership of an entity without learning the identity of that user. However, it is clear that the transaction is from a known group without knowing which group member is sending the data. Also, Petersen's scheme provides plausible anonymity which means that although group members are locally anonymous, their identity can be revoked by the manager, Mesh Server or coalition of members. Although transactions are anonymous, they are still recognizable as originating from a known group, preserving local anonymity of group members. Mesh utilizes forward secrecy techniques to protect data confidentiality. Forward secrecy, as described in [61], allows any entity to decrypt and view the product's history up to the point of its ownership transfer. However, access to subsequent records is restricted, preventing entities from viewing future transactions or exchanges after the point of transfer.

DECOUPLES [129] is an SCM solution utilizing the blockchain presented in [174]. This blockchain features a customized block validation mechanism specifically designed for DECOUPLES, rendering it incompatible with established blockchain like Bitcoin [136] or Ethereum [54]. In their approach, certificates are issued and signed by a certificate organization (CO) for each participant in the supply chain. These certificates are then sent to the participant through certificate transactions on the blockchain. Certificate holders can subsequently use Schnorr signatures [158] to authenticate certificate ownership in their transactions, which include product information. Within the framework, Maouchi et al. proposed the product-specific stealth addresses (PASTA) protocol. This protocol employs Stealth addresses [68], to ensure that multiple payments made to the same payee are unlinkable, thereby preserving the transaction receiver's anonymity. Furthermore, to anonymize the sender, they utilized multi-layered linkable spontaneous anonymous group (MLSAG) ring signatures [142]. Additionally, confidential transaction information is encrypted using the elliptic curve integrated encryption scheme (ECIES) [52], with the public key of the receiving party. The DECOUPLE framework does not account for complex supply chain operations, such as the dividing or merging of products. Additionally, the security of a blockchain is largely contingent on its decentralization, typically achieved through a diverse and distributed network of validators. However, the framework relies on their customized blockchain might impact its decentralization.

zkLedger [137] is a transaction ledger designed to offer both privacy and auditability, making it ideal for banks seeking to encrypt their transactions while granting regulators access to their assets and deals. It allows auditors to request and receive verifiable confirmation from the banks, ensuring the accuracy of their responses. zkLedger utilizes a columnar ledger structure for secure and private transaction management. In this design, each column represents a different bank, while each row details a transaction where assets are transferred between banks. To execute a transaction, the sending bank makes two commitments using Pedersen commitments [145]: one for the assets deducted from its account and another for the assets credited to the recipient's account. These commitments are then placed in their respective columns. To enhance unlinkability, the sending bank also commits to zero for all other banks, effectively anonymizing the transaction by obscuring the involvement of banks not participating in the transaction. This approach ensures that the details of the transaction remain private, except to the parties involved and authorized auditors. zkLedger incorporates three crucial types of cryptographic proofs to maintain integrity and privacy: (1) *Proof of Balance* (π^B) ensures that the total assets transferred in a transaction equate to zero, indicating no creation or destruction of assets, without disclosing the sender. (2) *Proof of Assets* (π^A) verifies that the spending bank has enough assets for the transfer. (3) *Proof of Consistency* (π^C) guarantees the integrity of the ledger by preventing the inclusion of invalid data that could compromise the ledger's verifiability or an auditor's ability to validate transactions. Pedersen commitments enable banks to perform secure statistical analyses—like calculating sums, averages, and variances—on their assets for auditors without revealing specific transaction details. This ensures both transparency and privacy, allowing banks to verify their financial health while maintaining the confidentiality of transactions. zkLedger's design mandates that all transactions be recorded directly on the ledger. This requirement poses challenges for SCM applications that necessitate detailed documentation of product histories. Deploying zkLedger on public blockchains, such as Ethereum, could incur prohibitive costs due to the extensive information storage required. Additionally, utilizing zkLedger as an Ethereum smart contract would inadvertently reveal the addresses of entities uploading data records, potentially compromising their privacy. Furthermore, zkLedger requires unanimous consent among all participating banks for adding or removing an entity (column) from the ledger. This consensus mechanism, combined with the increased complexity and time demands for transaction processing, broadcasting, and verification with more participants, significantly decelerates the process. These constraints impede zkLedger's scalability and adaptability in SCM scenarios, essentially preventing smaller entities from easily integrating into the network. Lastly, zkLedger recommends that auditors maintain commitment caches to expedite the verification of column value sums. Without these caches, auditors are compelled to process the entire ledger for transaction confirmation, regardless of their relevance to the audited entity or asset. This necessity places a considerable burden on end customers, particularly those who do not consistently monitor the ledger

Chapter 4

Accelerating Post-quantum Secure zkSNARKs through Optimizing Additive FFT

Declaration of Contributions

This chapter is based on an unpublished paper. I have co-authored the paper and my main contributions are as follows:

- Deriving the exact number of additions and multiplications in the Cantor additive fast Fourier transform (FFT) [56].
- The C++ implementation of all the Cantor additive FFT algorithm with the minimum number of additions.
- The dasign and implementation of precomputations in the presented additive FFTs.
- The C++ implementation of modified versions of the Gao-Mateer additive FFT [83].
- The FFT Complexity analysis of the Aurora zero-knowledge succinct non-interactive arguments of knowledge (zkSNARK) and modified the Aurora C++ to employ the Cantor additive FFT instead of the Gao-Mateer additive FFT.

4.1 Introduction

The FFT over additive groups can be utilized in various zkSNARK protocols operating over binary extension fields. Examples of such protocols include Ligero [9], STARK [27], Aurora [35], Fractal [66], and Polaris [81]. Among these zkSNARKs, Fractal features the fastest verifier algorithm, Ligero boasts the fastest prover algorithm, and Aurora achieves the smallest proof size. While STARK employs algebraic intermediate representation (AIR), which is designed for converting the execution trace of a program into algebraic representations (e.g., polynomials), the others rely on rank-1 constraint system (R1CS) [92], which is suited for arithmetic circuit and is preferred in cryptography related applications, as many privacy-preserving solutions rely on zero-knowledge proofs (ZKPs) of knowledge of the preimage of a leaf in a Merkle hash tree constructed by the circuit of a hash algorithm (e.g., SHA-256). Furthermore, zkSNARK-based post-quantum digital signature schemes typically involve proving knowledge of the secret key for a symmetric-key encryption algorithm (e.g., AES) represented as an arithmetic circuit.

For this study, we select Aurora [35] to demonstrate the performance improvements achieved by optimizing the FFT algorithm using the Cantor special basis [56]. While our optimization is applicable to all the aforementioned zkSNARKs, Aurora was chosen due to its small proof size, which makes it a strong candidate for post-quantum secure digital signature schemes. Accordingly, Aurora serves as the foundation of Preon [62], a post-quantum digital signature scheme that was a first-round candidate in NIST's post-quantum cryptography standardization process [138].

Contributions Our main contributions are summarized as follows:

- In the Cantor FFT algorithm, we present a theoretical analysis of the vanishing polynomials, providing a precise count of their terms based on Hamming-weight. This approach enables an accurate determination of the number of additions required. To the best of our knowledge, prior works have only reported upper bounds. We also efficiently computed the vanishing polynomials and multiplication factors, improving the efficiency even without precomputation.

- We propose building blocks for the Cantor FFT algorithm and demonstrate its significant performance improvements compared to the Gao-Mateer algorithm while using our Cantor FFT algorithm in the current Aurora implementation [34]. Table 4.1 shows how our FFT algorithm optimizations using the Cantor special basis can accelerate the prover and verifier algorithms in Aurora.

Table 4.1: Prover and Verifier of the Aurora zkSNARK [35] over $\mathbb{F}_{2^{256}}$ based on the number of constraints N and the size of the Reed-Solomon codeword domain $|L|$, using Gao-Mateer and Cantor FFTs, on an AMD Ryzen 9 9950x @ 5.7 GHz (sec).

		Aurora Prover		Aurora Verifier	
$\log_2(N)$	$\log_2(\lvert L\rvert)$	GM FFT* [34]	Cantor FFT (this work)	GM FFT* [34]	Cantor FFT (this work)
10	17	0.879	0.654	0.047	0.046
11	18	1.825	1.338	0.063	0.062
12	19	3.755	2.713	0.094	0.093
13	20	7.888	5.727	0.153	0.151
14	21	18.014	11.708	0.269	0.264
15	22	39.153	25.085	0.495	0.485
16	23	82.302	49.981	0.946	0.926
17	24	171.490	102.191	1.885	1.792
18	25	363.369	212.064	3.597	3.506
19	26	753.485	435.800	7.100	6.909

* Gao-Mateer FFT using standard basis.

- For the Gao-Mateer algorithm, we also provide a detailed breakdown of its two core components: the Expand and Aggregate modules. We incorporate the Cantor special basis into the Gao-Mateer algorithm to enhance the computational and space efficiency. We also introduce the precomputation techniques that substantially reduce overhead in the Cantor algorithm and two levels of precomputation in the Gao-Mateer algorithm.

- We propose an analysis of the FFT call complexity in the Aurora zkSNARK, evaluating the number and size of each FFT/IFFT call based on the number of constraints and variables in a given R1CS, and the target security parameter. Additionally, we demonstrate how the careful selection of the shift element in the affine subspaces of Aurora enables a significant reduction in the space complexity of the precomputation in the Cantor FFT by leveraging the unique properties of the Cantor special basis.

- We provide a C++ implementations of the Cantor algorithm, the Gao-Mateer algorithm which uses the Cantor special basis and, our precomputation techniques. We then provide a comprehensive comparison of these algorithms, along with their respective precomputations in Figure 4.4.

The chapter is structured as follows. Section 4.2 presents optimizations for the Cantor algorithm, while Section 4.3 details the Gao-Mateer algorithm and its precomputations.

Section 4.4 analyzes FFT call complexity in Aurora, and Section 4.5 compares the FFT algorithms. Finally, Section 4.6 summarizes findings.

4.2 Cantor Algorithm Building Blocks

In our implementation of the Cantor algorithm, we opted for iteration rather than recursion. The recursive approach, as presented in Algorithm 2.1, introduces extra overhead due to function calls and the increased memory needed to track these calls. Therefore, our Cantor FFT of length $n = 2^m$ consists of m iterative rounds to evaluate a polynomial $f(x) \in \mathbb{F}_{2^k}[x]$ of degree $< 2^m$ over the affine subspace $\theta + W_m$, where $\theta \in \mathbb{F}_{2^k}$ and W_m must be generated by the Cantor special basis as described in Section 2.5.1. Also, for the Cantor special basis, we know that $\mathbb{Z}_{W_i}(x) = S^i(x)$ for $i = 0, 1, \ldots, m$. Thus, the coefficients of the vanishing polynomials $\mathbb{Z}_{W_i}(x)$ are in $\mathbb{F}_2$. Additionally, we have $S^i(\beta_i) = 1$. We can derive the following result regarding th existance of the Cantor special basis.

Theorem 4.1. *If the polynomial $S^i(x)$ has no solution $\beta_i \in \mathbb{F}_{2^k}$ such that $S^i(\beta_i) = 1$, then for any $\ell > i$, the polynomial $S^\ell(x)$ also has no solution for $S^\ell(x) = 1$ in $\mathbb{F}_{2^k}$. In other words, there does not exist a set of Cantor special basis $\beta_0 = 1, \beta_1, \ldots, \beta_\ell$ such that $S(\beta_j) = \beta_{j-1}$ for $j = 1, \ldots, \ell$.*

Let $0 \leq r \leq m - 1$ be the round number. In each round r, the algorithm processes 2^r polynomials of degree $< 2^{m-r}$, resulting in 2^{r+1} polynomials of degree $< 2^{m-r-1}$ at the end of the round. $f(x)$ is represented by $\mathbf{f}$, a vector of size 2^m that stores the coefficients as described in Section 2.1.5. We use the same vector to store all intermediate polynomials during each round. For example, in round r, $\mathbf{f}$ stores the concatenation of the 2^{r+1} polynomials. Finally, in round $r = m - 1$, the algorithm outputs 2^m constant values stored in the vector $\mathbf{f}$, representing the evaluations of $f(x)$ over $\theta + W_m$.

Before presenting the details of our Cantor algorithm implementation, we first explain the selection process of the Cantor basis of length m, denoted as $\{\beta_0, \beta_1, \ldots, \beta_{m-1}\}$. From [83, Appendix], we know that $W_m = \langle \beta_0, \beta_1, \ldots, \beta_{m-1} \rangle$ must be a subspace of the field (or subfield) $\mathbb{F}_{2^{2^\ell}}$. To determine the Cantor special basis for $\mathbb{F}_{2^{2^\ell}}$, we begin by defining a basis $\{\beta_0, \beta_1, \ldots, \beta_{2^\ell-1}\}$ such that $\mathrm{Tr}_{\mathbb{F}_{2^{2^\ell}}/\mathbb{F}_2}(\beta_{2^\ell-1}) = 1$. Then, we recursively determine the remaining basis elements by $\beta_{j-1} = \beta_j^2 + \beta_j$ for $1 \leq j \leq 2^\ell - 1$. We then select the first m elements from this basis to construct our Cantor special basis of dimension m. In our implementation, we randomly try different values of $\beta_{2^\ell-1}$ and compute its trace. With a probability of 0.5, this value will have a trace of 1.

4.2.1 Vanishing Polynomials

In the Cantor additive FFT of length 2^m, the vanishing polynomials $\mathbb{Z}_{W_0}, \mathbb{Z}_{W_1}, \ldots, \mathbb{Z}_{W_{m-1}}$ must be computed to perform the division algorithm. The coefficients in each $\mathbb{Z}_{W_i}(x)$ are derived from

$$\mathbb{Z}_{W_i}(x) = \sum_{j=0}^{i} \left[\binom{i}{j} \mod 2 \right] x^{2^j}.$$

Employing Lucas's theorem [126], we efficiently compute $\binom{i}{j} \equiv \prod_{k=0}^{t-1} \binom{i_k}{j_k} \bmod 2$, where

$$\begin{aligned} i &= i_0 + i_1 2 + i_2 2^2 + \ldots + i_{t-1} 2^{t-1} \quad (i_k \in \mathbb{F}_2), \\ j &= j_0 + j_1 2 + j_2 2^2 + \ldots + j_{t-1} 2^{t-1} \quad (j_k \in \mathbb{F}_2). \end{aligned}$$

Theorem 4.2. *[78, Theorem 2] The number of integers j not exceeding i for which $\binom{i}{j} \not\equiv 0 \pmod 2$ is $\prod_{k=0}^{t}(i_k + 1)$.*

Thus, by the above theorem we conclude that the number of non-zero coefficients in $\mathbb{Z}_{W_i}(x)$ equals to $2^{\mathrm{wt}(i)}$, where $\mathrm{wt}(i)$ denotes the Hamming weight of i, i.e., the number of bits equal to 1. Since the number of non-zero coefficients in most vanishing polynomials is considerably less than its degree (i.e., 2^i), we avoid using vectorial representation of univariate polynomials described in Section 2.1.5 for these polynomials. Otherwise the polynomial division algorithm would require excessive addition operation on zero coefficients.

To efficiently represent the vanishing polynomials, we store the index number of the coefficients in the reverse order. Specifically, let $\mathbf{z}_i$ represent $\mathbb{Z}_{W_i}(x) = \sum_{j=0}^{i} c_j x^{2^j}$ in our implementation. We define

$$\mathbf{z}_i = (\zeta_0, \zeta_1, \ldots, \zeta_{2^{\mathrm{wt}(i)}-1}) = (2^i - 2^j | c_j = 1 \text{ in } \mathbb{Z}_{W_i}(x)),$$

where Algorithm 4.1 describes how $\mathbf{z}_i$ is computed.

Algorithm 4.1 also computes the evaluation of $\mathbb{Z}_{W_i}(\theta)$ which is used later in Section 4.2.3. Reversing the order (i.e., measuring the distance from 2^i rather than from 0) eliminates the need for degree shifting in $\mathbb{Z}_{W_i}(x)$ during the division rounds. Additionally, since $\binom{i}{i} = 1$, c_i is always one, and $\zeta_{2^{\mathrm{wt}(i)}-1}$ is zero which can be omitted from $\mathbf{z}_i$. This omission excludes this coefficient from the polynomial division algorithm, saving one addition per division round while keeping the quotient in the same vector as the dividend.

Algorithm 4.1 Vanishing Polynomial (i, θ)

```
Require: $i \in \mathbb{N}$ and $\theta \in \mathbb{F}_{2^k}$
Ensure: $\mathbf{z}_i$, eval.
 1: $\ell \leftarrow 0$
 2: eval $\leftarrow \theta$
 3: for $j = 0$ to $i - 1$ do
 4:     $t \leftarrow \lceil \log_2(j+1) \rceil$
 5:     $k \leftarrow 0$
 6:     while $(i_k \vee \neg j_k) \wedge (k < t)$ do
 7:         $k \leftarrow k + 1$
 8:     end while
 9:     if $k = t$ then
10:         $\zeta_\ell \leftarrow 2^i - 2^j$
11:         eval $\leftarrow$ eval $+ \theta^{2^j}$
12:         $\ell \leftarrow \ell + 1$
13:     end if
14: end for
15: eval $\leftarrow$ eval $+ \theta^{2^i}$                ▷ As the highest degree term is not in $\mathbf{z}_i$
16: assert $\ell = 2^{\mathrm{wt}(i)} - 2$
17: return $\mathbf{z}_i \leftarrow (\zeta_0, \zeta_1, \ldots, \zeta_{2^{\mathrm{wt}(i)}-2})$, eval.
```

4.2.2 Polynomial Division

In round r of the Cantor algorithm, the 2^r polynomials $f_{i,r}(x)$, each of degree $< 2^p$, are divided by $\mathbb{Z}_{Wp-1}(x + \theta_{i,r})$ and $\mathbb{Z}_{Wp-1}(x + \theta_{i,r} + \beta_{p-1})$, where $p = m - r$ and $0 \leq i \leq 2^r - 1$. The corresponding remainders of these divisions for each $f_{i,r}(x)$ are outputted to be processed in the next round. Since the vanishing polynomials are linearized polynomials, we have

$$\mathbb{Z}_{Wp-1}(x + \theta_{i,r}) = \mathbb{Z}_{Wp-1}(x) + \mathbb{Z}_{Wp-1}(\theta_{i,r}), \text{ and}$$
$$\mathbb{Z}_{Wp-1}(x + \theta_{i,r} + \beta_{p-1}) = \mathbb{Z}_{Wp-1}(x) + \mathbb{Z}_{Wp-1}(\theta_{i,r}) + \mathbb{Z}_{Wp-1}(\beta_{p-1}),$$

where $\mathbb{Z}_{Wp-1}(\beta_{p-1}) = 1$. Since $\deg(f_i(x)) < 2\deg(\mathbb{Z}_{Wp-1}(x))$, we reduce the two divisions required for each $f_i(x)$ to a single division by $\mathbb{Z}_{Wp-1}(x)$, and then compute the remainders of the original divisions accordingly.

Proposition 4.1. *Let $f(x)$ and $g(x)$ be polynomials such that $\deg(f) < 2^m$ and $\deg(g) = 2^{m-1}$. Suppose that dividing $f(x)$ by $g(x)$ yields the quotient $q(x)$ and the remainder $r(x)$, so* *that $f(x) = g(x)q(x)+r(x)$. Let c be a constant, and let $q'(x)$ and $r'(x)$ be the quotient and remainder when dividing $f(x)$ by $g(x)+c$. Then, $q'(x) = q(x)$, and $r'(x) = r(x)-c\,q(x)$.*

Proof. We prove this in two steps:

1. According to polynomial division algorithm, $\deg(q) = \deg(q') = deg(f) - deg(g) < 2^{m-1}$,
2. Let $f(x) = (g(x) + c)q'(x) + r'(x)$, then

$$\begin{aligned}&(g(x) + c)q'(x) + r'(x) = g(x)q(x) + r(x)\\ \Longrightarrow\ & g(x)(q'(x) - q(x)) = r(x) - r'(x) - cq'(x).\end{aligned}$$

Now, assume the opposite, that is, $q'(x) \neq q(x)$. This implies $q'(x) - q(x) \neq 0$, meaning $\deg(q'(x)-q(x)) \geq 0$. Since $g(x)$ divides $g(x)(q'(x)-q(x))$, the degree of the right-hand side $r(x)-r'(x)-cq'(x)$ must be at least $\deg(g)$. However, by the polynomial division algorithm, $\deg(r) < \deg(g)$ and $\deg(r') < \deg(g)$, and according to step 1, $\deg(q') < \deg(g) = 2^{m-1}$. Therefore, $\deg(r(x)-r'(x)-cq'(x)) < \deg(g)$, which is a contradiction. Hence, $q'(x) = q(x)$, and consequently, $r'(x) = r(x) - c\,q(x)$. □

Since the coefficients in $\mathbb{Z}_{W_{p-1}}(x)$ are in $\mathbb{F}_2$, division by $\mathbb{Z}_{W_{p-1}}(x)$ requires only additions. In the division algorithm, the dividend polynomial, denoted as $f_{i,r}(x)$, is added to scaled degree-shifts of $\mathbb{Z}_{W_{p-1}}(x)$. However, we eliminate the need for degree shifts by using the reversed index order in $\mathbf{z}_i$, which represents the distance of each non-zero coefficient from the highest degree (i.e., 2^p).

Dividing $f_{i,r}(x)$, a polynomial of degree $< 2^p$, by $\mathbb{Z}_{W_{p-1}}(x)$ yields the quotient $q_{i,r}(x)$ and the remainder $r_{i,r}(x)$, each with degree $< 2^{p-1}$. Let $\mathbf{f}_{i,r}$ represent the $(i+1)$-th sub-vector of 2^p elements in the vector $\mathbf{f}$ at the beginning of round r. This sub-vector stores the coefficients of $f_{i,r}(x)$, ordered from the constant term to the highest degree term. Our polynomial division algorithm begins with the coefficient of the highest degree term in $\mathbf{f}_{i,r}$ and subtracts that coefficient from the lower-degree coefficients, spaced by distances determined by $\mathbf{z}_i$. Since $\mathbf{z}_i$ does not include zero, the coefficient of the highest degree term remains unaffected. In the next round of the polynomial division, the algorithm repeats this process with the second highest degree term. After 2^{p-1} rounds, the higher-degree (right) half of $\mathbf{f}_{i,r}$ stores the coefficients of $q_{i,r}(x)$, while the lower-degree (left) half stores the coefficients of $r_{i,r}(x)$. Then, given Proposition 4.1, the algorithm processes the outputs of the polynomial division, which serve as inputs to the next round, as follows:

$$\begin{aligned}f_{2i,r+1}(x) &= r_{i,r}(x) - \mathbb{Z}_{W_{p-1}}(\theta_{i,r})\,q_{i,r}(x), \text{ and}\\ f_{2i+1,r+1}(x) &= f_{2i,r+1}(x) - q_{i,r}(x),\end{aligned}$$

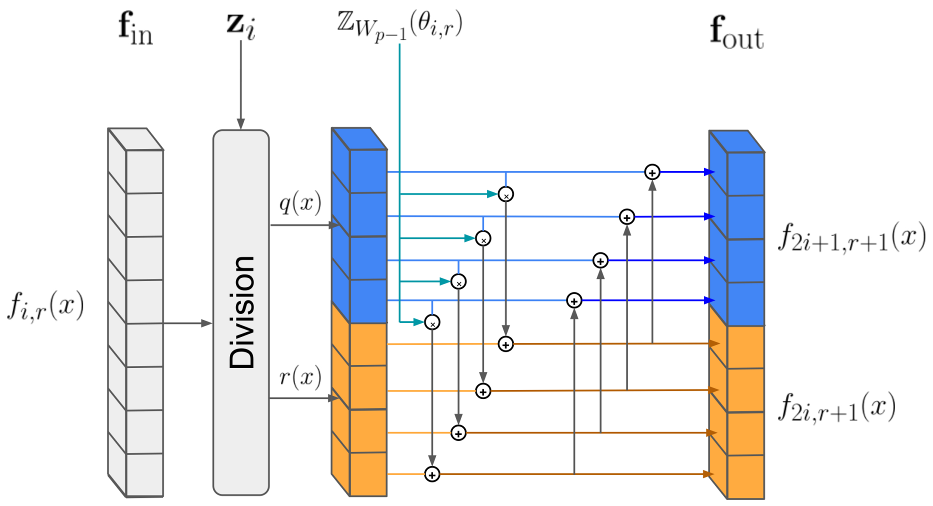


Figure 4.1: Computing the polynomials for the next round from the outputs of dividing $f_{i,r}(x)$ (of degree < 8) by $\mathbb{Z}_{W_{p-1}}$, represented by $\mathbf{z}_{p-1}$.

where $\mathbb{Z}_{W_{p-1}}(\theta_{i,r})$ denotes the evaluation of the vanishing polynomial at $\theta_{i,r}$. Figure 4.1 illustrates the computation of the polynomials for the next round from the quotient and remainder in each round. Our polynomial division algorithm implementation integrates the computation of the quotient, remainder, and the polynomials for the next round.

The polynomial division algorithm requires $2^{p-1} \times \left(2^{\text{wt}(p-1)} + 1\right)$ additions and 2^{p-1} multiplications. The peak space complexity is $O(1)$ since the algorithm does not require any auxiliary vectors for storing intermediate values.

The Canopy module, described in the next section, is responsible for providing all of the inputs of the polynomial division algorithm.

4.2.3 Canopy Module

Cantor algorithm is implemented by Canopy modules of varying input sizes 2^p and indices i, denoted as $\mathsf{Canopy}_{p,i}$. The index i indicates the module number in each round and determines the offset from the start of the $\mathbf{f}$ vector, where the coefficients of the input polynomial begin with $\mathsf{offset} = i2^p$. Figure 4.2 illustrates our iterative approach of using Canopy modules to implementing the Cantor algorithm.

The $\mathsf{Canopy}_{p,i}$ determines $\theta_{i,r}$ and then evaluate $\mathbb{Z}_{W_{p-1}}(\theta_{i,r})$, which is necessary for running Algorithm 4.2. Let Cantor algorithm evaluate $f(x)$ of degree $< 2^m$ over $\theta + \langle \beta_0 =$

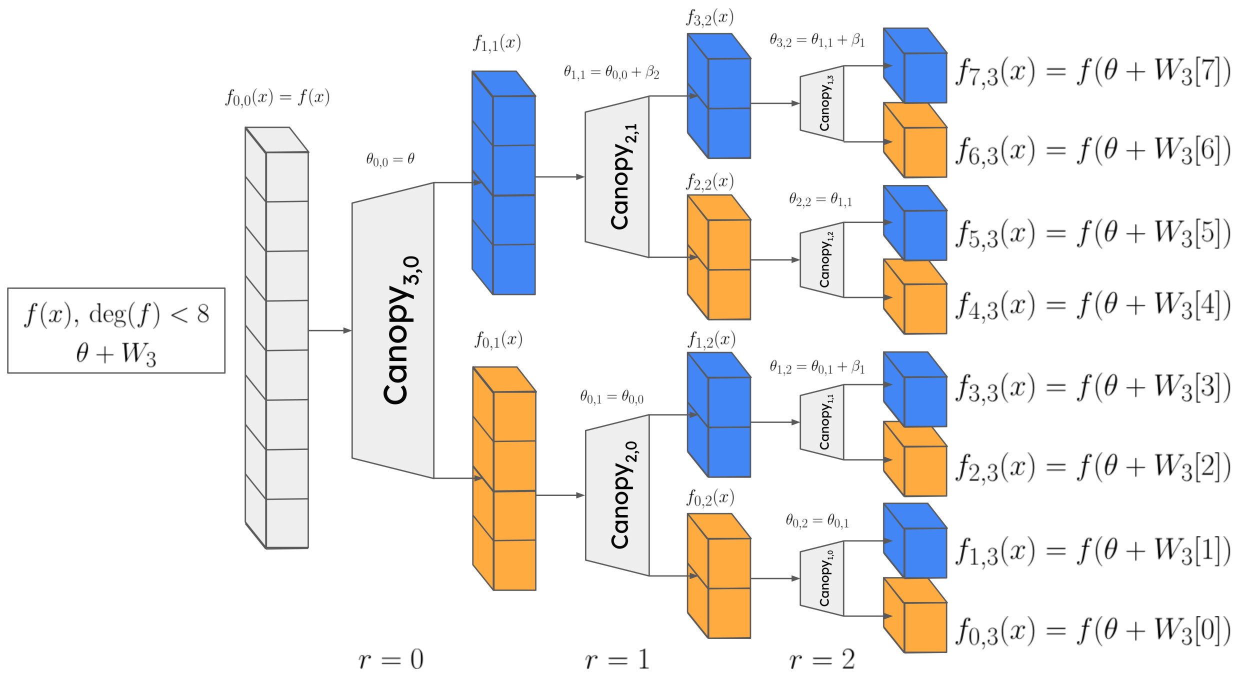

$f(x)$, $\deg(f) < 8$
$\theta + W_3$
$f_{0,0}(x) = f(x)$
$\theta_{0,0} = \theta$
$\text{Canopy}_{3,0}$
$f_{1,1}(x)$
$f_{0,1}(x)$
$\theta_{1,1} = \theta_{0,0} + \beta_2$
$\theta_{0,1} = \theta_{0,0}$
$\text{Canopy}_{2,1}$
$\text{Canopy}_{2,0}$
$f_{3,2}(x)$
$f_{2,2}(x)$
$f_{1,2}(x)$
$f_{0,2}(x)$
$\theta_{3,2} = \theta_{1,1} + \beta_1$
$\theta_{2,2} = \theta_{1,1}$
$\theta_{1,2} = \theta_{0,1} + \beta_1$
$\theta_{0,2} = \theta_{0,1}$
$\text{Canopy}_{1,3}$
$\text{Canopy}_{1,2}$
$\text{Canopy}_{1,1}$
$\text{Canopy}_{1,0}$
$f_{7,3}(x) = f(\theta + W_3[7])$
$f_{6,3}(x) = f(\theta + W_3[6])$
$f_{5,3}(x) = f(\theta + W_3[5])$
$f_{4,3}(x) = f(\theta + W_3[4])$
$f_{3,3}(x) = f(\theta + W_3[3])$
$f_{2,3}(x) = f(\theta + W_3[2])$
$f_{1,3}(x) = f(\theta + W_3[1])$
$f_{0,3}(x) = f(\theta + W_3[0])$
$r = 0$
$r = 1$
$r = 2$

Figure 4.2: Cantor FFT algorithm of length $n = 2^3$ which evaluates $f(x) \in \mathbb{F}[x]$ of degree < 8 over $\theta + W_3$, where $\theta \in \mathbb{F}_{2^k}$ and W_3 is the Cantor special basis.

Algorithm 4.2 Polynomial Division ($\mathbf{f}_{\text{in}}$, $\mathbf{z}_{p-1}$, $\mathbb{Z}_{W_{p-1}}(\theta_{i,r})$, p, i)

```
Require: $\mathbf{f}_{\text{in}} = (c_0, c_1, \ldots, c_{n-1})$, $\mathbf{z}_{p-1} = (\zeta_0, \zeta_1, \ldots, \zeta_{2^{\text{wt}(p-1)}-2})$, $\mathbb{Z}_{W_{p-1}}(\theta_{i,r}) \in \mathbb{F}_{2^k}$, $p = m - r$, and $i$ determines the polynomial $f_{i,r}(x)$, of degree $< 2^p$, in $\mathbf{f}_{\text{in}}$.
Ensure: $\mathbf{f}_{\text{out}}$
 1: offset $\leftarrow i \times 2^p$        ▷ The offset at which the coefficients of $f_{i,r}(x)$ are in $\mathbf{f}_{\text{in}}$
 2: for $k = 2^p + \text{offset} - 1$ to $2^{p-1} + \text{offset}$ do        ▷ Iterates over the higher-degree half in decreasing order
 3:     for $\ell = 0$ to $2^{\text{wt}(p-1)} - 2$ do
 4:         $c_{(k-\zeta_\ell)} \leftarrow c_{(k-\zeta_\ell)} + c_k$
 5:     end for
 6:     $c_{(k-2^{p-1})} \leftarrow c_{(k-2^{p-1})} + c_k \times \mathbb{Z}_{W_{p-1}}(\theta_{i,r})$        ▷ Computes $f_{2i,r+1}(x)$
 7: end for
 8: for $k = 2^{p-1} + \text{offset}$ to $2^p + \text{offset} - 1$ do
 9:     $c_k \leftarrow c_k + c_{(k-2^{p-1})}$        ▷ Computes $f_{2i+1,r+1}(x)$
10: end for
11: return $\mathbf{f}_{\text{out}} \leftarrow (c_0, c_1, \ldots, c_{n-1})$
```

$1, \beta_1, \ldots, \beta_{m-1}\rangle$. Let $\theta_{0,0} = \theta$, and for $1 \le r \le m-1$ and $0 \le i \le 2^r - 1$, $\theta_{i,r}$ is determined recursively according to the following rules:

$$\theta_{i,r} = \begin{cases} \theta_{i/2,r-1} & \text{if } i \bmod 2 = 0, \\ \theta_{(i-1)/2,r-1} + \beta_p & \text{if } i \bmod 2 = 1, \end{cases}$$

where $p = m - r$. This equation can be simplified to $\theta_{i,r} = \theta_{\lfloor i/2 \rfloor, r-1} + (i \bmod 2)\beta_p$, and can be written as,

$$\theta_{i,r} = \theta + \sum_{j=0}^{r-1} \left(\lfloor \frac{i}{2^j} \rfloor \bmod 2 \right) \beta_{p+j} = \theta + \sum_{j=0}^{r-1} i_j \, \beta_{p+j},$$

where $i = i_0 + i_1 2 + i_2 2^2 + \ldots + i_{r-1} 2^{r-1}$ ($i_j \in \mathbb{F}_2$), denotes the binary representation of i. Then, to evaluate $\mathbb{Z}_{W_{p-1}}(\theta_{i,r})$, we employ the rule $S^i(\beta_{i+\ell}) = \beta_\ell$ provided earlier in this section to simplify the computation. Specifically, we write $\mathbb{Z}_{W_{p-1}}(\beta_{p-1+j}) = \beta_j$. Therefore $\mathbb{Z}_{W_{p-1}}(\theta_{i,r})$ can be evaluated as

$$\mathbb{Z}_{W_{p-1}}(\theta_{i,r}) = \mathbb{Z}_{W_{p-1}}(\theta) + \sum_{j=0}^{r-1} i_j \, \beta_{j+1},$$

where $\mathbb{Z}_{W_{p-1}}(\theta)$ is evaluated at each round while constructing $\mathbf{z}_{p-1}$ from $\mathbb{Z}_{W_{p-1}}(x)$ during Algorithm 4.1. The computation of $\mathbb{Z}_{W_{p-1}}(\theta)$ is shared across all the Canopy modules in each row since the vanishing polynomial remains consistent.

Algorithm 4.3 Canopy$_{p,i}$ ($\mathbf{f}_{\text{in}}$, $\{\beta_0, \beta_1, \ldots, \beta_{m-1}\}$, $\mathbf{z}_{p-1}$, $\mathbb{Z}_{W_{p-1}}(\theta)$, p, i)

```
Require: $\mathbf{f}_{\text{in}}$, $\{\beta_0, \beta_1, \ldots, \beta_{m-1}\}$ is the Cantor special basis, $\mathbf{z}_{p-1} = (\zeta_0, \zeta_1, \ldots, \zeta_{2^{\text{wt}(i)}-2})$,
    $\mathbb{Z}_{W_{p-1}}(\theta) \in \mathbb{F}_{2^k}$, $p = m - r$, and $i = (i_0, i_1, i_2, \ldots, i_r)$.
Ensure: $\mathbf{f}_{\text{out}}$
1: $\psi_{i,r} \leftarrow \mathbb{Z}_{W_{p-1}}(\theta)$
2: for $j = 0$ to $r - 1$ do
3:     $\psi_{i,r} \leftarrow \psi_{i,r} + i_j \times \beta_{j+1}$
4: end for
5: $\mathbf{f}_{\text{out}} \leftarrow$ POLYNOMIAL DIVISION($\mathbf{f}_{\text{in}}$, $\mathbf{z}_{p-1}$, $\psi_{i,r}$, $p$, $i$)    ▷ Algorithm 4.2, where
    $\mathbb{Z}_{W_{p-1}}(\theta_{i,r}) = \psi_{i,r}$
6: return $\mathbf{f}_{\text{out}}$
```

Algorithm 4.4 Cantor Algorithm ($\mathbf{f}_{\text{in}}$, θ, $\{\beta_0, \beta_1, \ldots, \beta_{m-1}\}$)

```
Require: $\mathbf{f}_{\text{in}}$ is a vector of size $2^m$ representing the coefficients in $f(x)$ (where $\deg f < 2^m$),
    $\theta \in \mathbb{F}_{2^k}$ is the affine shift, and $\{\beta_0, \beta_1, \ldots, \beta_{m-1}\}$ is the Cantor special basis.
Ensure: $\mathbf{f}_{\text{out}}$ is the vector of evaluations of $f(x)$ over $\theta + \langle \beta_0, \beta_1, \ldots, \beta_{m-1} \rangle$.
1: for $r = 0$ to $m - 1$ do
2:     $p \leftarrow m - r$
3:     $\mathbf{z}_{p-1}$, eval $\leftarrow$ VANISHING POLYNOMIAL($p - 1$, $\theta$)    ▷ Algorithm 4.1
4:     for $i = 0$ to $2^r - 1$ do
5:         Canopy$_{p,i}$($\mathbf{f}_{\text{in}}$, $\{\beta_0, \beta_1, \ldots, \beta_{m-1}\}$, $\mathbf{z}_{p-1}$, eval, $p$, $i$)    ▷ Algorithm 4.3
6:     end for
7: end for
8: return $\mathbf{f}_{\text{out}}$
```

Algorithm 4.3 details the steps within the Canopy$_{p,i}$ module, and Algorithm 4.4 describes the implementation of the Cantor algorithm based on those modules.

4.2.4 Detailed Cost Analysis

At the rth iteration of the algorithm, we perform 2^r divisions by the polynomial $\mathbb{Z}_{W_{m-r-1}}(x)$. Consequently, the total number of additions resulting from polynomial division in the Cantor additive FFT is given by

$$\sum_{r=0}^{m-1} 2^r \cdot 2^{m-r-1}\left(2^{\mathrm{wt}(m-r-1)} - 1\right) = 2^{m-1}\sum_{r=0}^{m-1} 2^{\mathrm{wt}(r)} - m2^{m-1}$$

From Steps 6 and 9 of Algorithm 4.2, we know that for the input polynomial in the $(r+1)$th iteration, we require 2×2^r polynomial additions, each of degree less than 2^{m-r-1}. This leads to a total of $\sum_{r=0}^{m-1} 2 \cdot 2^r \cdot 2^{m-r-1} = 2^m m$ additions. Therefore, the total number of additions in the Cantor additive FFT is given by

$$2^m m + 2^{m-1}\sum_{r=0}^{m-1} 2^{\mathrm{wt}(r)} - m2^{m-1} = \frac{1}{2}n\log_2 n + \frac{1}{2}n \sum_{r=0}^{\log_2(n)-1} 2^{\mathrm{wt}(r)}.$$

This provides an exact count of the additions required in the Cantor additive FFT, whereas previous works, to the best of our knowledge, have only established upper bounds.

On the other hand, from Step 6 of Algorithm 4.2, we know that for the input polynomial in the $(r+1)$th iteration, we require $2^r \times 2^{m-r-1} = 2^{m-1}$ multiplications. Thus, the number of multiplications in the Cantor additive FFT is given by $\sum_{r=0}^{m-1} 2^{m-1} = \frac{1}{2}n\log_2 n$.

If the Cantor additive FFT is performed over a subspace W_m, due to Step 6 of the Algorithm 4.2, we must account for a reduction of $\sum_{r=0}^{m-1} 2^{m-r-1} = 2^m - 1 = n - 1$ in both additions and multiplications. Thus, the costs for additions and multiplications are changed to $\frac{1}{2}n\log_2 n + \frac{1}{2}n\sum_{r=0}^{\log_2(n)-1} 2^{\mathrm{wt}(r)} - n + 1$ and $\frac{1}{2}n\log_2 n - n + 1$, respectively.

4.2.5 Precomputation

The computations of $\mathbb{Z}_{W_{p-1}}(\theta_{i,r})$ for each $\mathsf{Canopy}_{p,i}$, denoted as $\psi_{i,r}$ in Line 3 of Algorithm 4.3, and the computation of the vanishing polynomials in Line 3 of Algorithm 4.4, do not depend on the input polynomial. The storage required to store the precomputed values for the Cantor algorithm of length $n = 2^m$ is $\sum_{i=0}^{m-1}\left(2^{\mathrm{wt}(i)} - 1\right)$ integers to store $\mathbf{Z} = (\mathbf{z}_0, \mathbf{z}_1, \ldots, \mathbf{z}_{m-1})$ and $2^m - 1$ field elements to store $\boldsymbol{\Psi} = (\boldsymbol{\psi}_0, \boldsymbol{\psi}_1, \ldots, \boldsymbol{\psi}_{m-1})$, where $\boldsymbol{\psi}_r = (\psi_{0,r}, \psi_{1,r}, \ldots, \psi_{2^r-1,r})$. Algorithm 4.5 describes the precomputation algorithm.

For a special case where the affine shift θ is an element of the Cantor special basis, we do not have to pre-compute all $\mathbb{Z}_{W_{p-1}}(\theta_{i,r})$, since these values are only combinations of the cantor basis. In this case we can precompute the lookup table where each store the combinations of subset of Cantor basis elements.

Algorithm 4.5 Cantor Precomputation (θ, $\{\beta_0, \beta_1, \ldots, \beta_{m-1}\}$)

```
Require: $\theta \in \mathbb{F}_{2^k}$ is the affine shift, and $\{\beta_0, \beta_1, \ldots, \beta_{m-1}\}$ is the Cantor special basis.
Ensure: $\mathbf{Z} = (\mathbf{z}_0, \mathbf{z}_1, \ldots, \mathbf{z}_{m-1})$ is a vector of vanishing polynomials, $\mathbf{\Psi} = (\boldsymbol{\psi}_0, \boldsymbol{\psi}_1, \ldots, \boldsymbol{\psi}_{m-1})$, where $\boldsymbol{\psi}_r = (\psi_{0,r}, \psi_{1,r}, \ldots, \psi_{2^r-1,r})$
 1: for $r = 0$ to $m-1$ do
 2:     $\mathbf{z}_i, \mathsf{eval} \leftarrow$ VANISHING POLYNOMIAL$(m-r-1,\ \theta)$
 3:     for $i = 0$ to $2^r - 1$ do
 4:         $\psi_{i,r} \leftarrow \mathsf{eval}$
 5:         for $j = 0$ to $r-1$ do
 6:             $\psi_{i,r} \leftarrow \psi_{i,r} + i_j \times \beta_{j+1}$
 7:         end for
 8:     end for
 9:     $\boldsymbol{\psi}_r \leftarrow (\psi_{0,r}, \psi_{1,r}, \ldots, \psi_{2^r-1,r})$
10: end for
11: return $\mathbf{Z} \leftarrow (\mathbf{z}_0, \mathbf{z}_1, \ldots, \mathbf{z}_{m-1})$, $\mathbf{\Psi} \leftarrow (\boldsymbol{\psi}_0, \boldsymbol{\psi}_1, \ldots, \boldsymbol{\psi}_{m-1})$
```

4.3 Gao-Mateer Algorithm Building Blocks

The Gao-Mateer FFT implementation of length $n = 2^m$ consists of $2m$ iterative rounds to evaluate a polynomial $f(x) \in \mathbb{F}_{2^k}[x]$ of degree $< 2^m$ over the affine subspace $\theta + W_m$, where $\theta \in \mathbb{F}_{2^k}$ and $W_m = \langle \beta_0, \beta_1, \ldots, \beta_{m-1} \rangle$.

We describe the Gao-Mateer algorithm through two primary modules: the Expand module and the Aggregate module. Expand is an r-round algorithm where in each round $0 \le r \le m-1$, it expands 2^r polynomials of degree $< 2^{m-r}$ into 2^{r+1} smaller polynomials of degree $< 2^{m-r-1}$. Similar to our Cantor algorithm implementation, only one vector of length 2^m denoted as $\mathbf{f}$ is required to store all the polynomials in each round. Aggregate is an r-round algorithm that takes the output of Expand, and iteratively folds them over r rounds, ultimately producing the evaluations of $f(x)$ on $\theta + W_m$. In the following, we start by presenting the Taylor expansion algorithm, which serves as the core component of Expand.

4.3.1 Taylor Expansion

Let $f(x) = \sum_{i=0}^{m-1} c_i x^i, c_i \in \mathbb{F}_q$, $\delta(x) = x^2 + x$. We denote $y = \delta(x)$. Then we may write f as

$$f(x) = f_0(y) + x f_1(y) \tag{4.1}$$

Here, f_i, $i \in \{0,1\}$, are polynomials in $\mathbb{F}_{2^k}$ with degree $< 2^{m-1}$. The representation of f by (4.1) is referred to as the Taylor expansion in [83]. The polynomials f_i can be obtained iteratively as follows. Let $\mathbf{f} = (c_0, \ldots, c_{n-1})$, referred to as the coefficient vector of $f(x)$, where $n = 2^m$, denoted as $f(x) \leftrightarrow \mathbf{f}$. Let $t = m - 2$, we define a *quadrant concatenation* of $\mathbf{f}$ as $\mathbf{f} = (\mathbf{c}_0, \mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_3)$ where $c_i = (c_{i\cdot 2^t}, c_{i\cdot 2^t+1}, \ldots, c_{(i+1)\cdot 2^t-1})$. In the following, we model the Taylor expansion of f in terms of the concept of T_n module. The T_n module operation on f is defined as $\mathbf{b} = (\mathbf{c}_0, \mathbf{c}_1 + \mathbf{h}, \mathbf{h}, \mathbf{c}_3)$, where $\mathbf{h} = \mathbf{c}_2 + \mathbf{c}_3$. From $\mathbf{b} = (\mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3)$, we have the following result.

Lemma 4.1. *With the above notation, we have*

$$f(x) = g_0(x) + g_1(x)y^{2^t}, \text{ where } y = \delta(x) \tag{4.2}$$

where $g_0(x) \leftrightarrow (\mathbf{b}_0, \mathbf{b}_1)$ *and* $g_1(x) \leftrightarrow (\mathbf{b}_2, \mathbf{b}_3)$.

Now, to compute the Taylor expansion of $f(x) \in \mathbb{F}[x]$ of degree $< n = 2^m$, the Taylor expansion of $f(x)$ at $\delta(x) = x^2 + x$, denoted as $\mathsf{TE}(f, n, 2)$ can be computed through the Taylor expansions of two polynomials:

$$\mathsf{TE}(f, n, 2) = (\mathsf{TE}(g_0, n, 2), \mathsf{TE}(g_1, n, 2)) \tag{4.3}$$

where g_0 and g_1 are computed in Lemma 4.1 by T_n module.

Theorem 4.3. *Let* $\mathbf{u} = (u_0, u_1, \ldots, u_{n-1})$ *be the output at the recursion* t *in (4.3), then the Taylor expansion of* f*, defined by (4.1), is given by*

$$\begin{aligned} f_0 &\leftrightarrow (u_0, u_2, \ldots, u_{2i}, \ldots, u_{2\lfloor n/2 \rfloor}) \\ f_1 &\leftrightarrow (u_1, u_3, \ldots, u_{2i+1}, \ldots, u_{2\lfloor n/2 \rfloor+1}), \end{aligned}$$

where f_0 *and* f_1*, are polynomials each of degree* $< \lfloor n/2 \rfloor = 2^{m-1}$.

As described in the above theorem, the polynomials f_0 and f_1 are constructed by performing an even-odd rearrangement on $\mathbf{u}$. Subsequently, as described in Algorithm 2.2, the evaluations $\mathsf{TE}(f_0, n/2, 2)$ and $\mathsf{TE}(f_1, n/2, 2)$ also need to be computed and so on their outputs as well. Alternatively, as implemented in [34], instead of performing the even-odd rearrangements, the positions of the coefficients of these two polynomials can be retained within $\mathbf{u}$. This allows us to compute a single $\mathsf{TE}(u, n/2, 2)$ on the entire polynomial $u(x)$, represented by $\mathbf{u}$, in $t-1$ recursive steps (one fewer recursion than required for f). Finally, to account for the skipped even-odd rearrangements, in the last round the resulting vector is rearranged in bit-reversal order defined as follows:

Define a recursive process that splits the sequence of indices $\{0, 1, \ldots, 2^n - 1\}$ into two groups based on the least significant bit (LSB): the even-index group, containing indices where the LSB is 0, and the odd-index group, containing indices where the LSB is 1. Recursively apply the same splitting process to each group based on the next least significant bit until all groups contain exactly two indices.

Proposition 4.2. *The final arrangement of indices, after applying the recursive splitting process, represents the bit-reversal of the original sequence of indices* $\{0, 1, \ldots, 2^n - 1\}$.

Proof. We will prove this statement using mathematical induction on n.

For $n = 2$, we have the set of indices $\{0, 1, 2, 3\}$. In binary, the indices are represented as $\{00_2, 01_2, 10_2, 11_2\}$. The recursive splitting based on the least significant bit (LSB) yields:

- Even-index group: $\{0, 2\} = \{00_2, 10_2\}$ (LSB $= 0$)
- Odd-index group: $\{1, 3\} = \{01_2, 11_2\}$ (LSB $= 1$)

Thus, after splitting, the order is $\{0, 2, 1, 3\}$, which corresponds to the bit-reversal of the binary representations of the original sequence:

$$\{00_2, 01_2, 10_2, 11_2\} \quad \text{becomes} \quad \{00_2, 10_2, 01_2, 11_2\}.$$

Hence, the base case holds for $n = 2$.

Now, assume that for some $n = k \geq 2$, the recursive process results in a bit-reversal permutation for the sequence of indices $\left\{0, 1, \ldots, 2^k - 1\right\}$. That is, after the recursive splitting, the sequence of indices corresponds to the bit-reversal of their binary representations.

We need to show that the result holds for $n = k + 1$.

Consider the sequence $\left\{0, 1, \ldots, 2^{k+1} - 1\right\}$ of 2^{k+1} elements. This sequence can be split into two groups based on the least significant bit (LSB):

- Even-index group E: Indices where the LSB is 0, i.e.,

$$E = \left\{0, 2, 4, \ldots, 2^{k+1} - 2\right\}.$$

- Odd-index group O: Indices where the LSB is 1, i.e.,

$$O = \left\{1, 3, 5, \ldots, 2^{k+1} - 1\right\}.$$

Note that each group E and O contains 2^k indices. By the inductive hypothesis, the recursive splitting within each group E and O will result in the bit-reversal of the indices within that group.

After recursively applying the bit-reversal process to both groups, the final sequence of indices is formed by first placing the indices from the even group E (which are already in bit-reversed order) followed by the indices from the odd group O (also in bit-reversed order).

- Indices in E correspond to even indices from the original sequence $\{0, 1, \ldots, 2^{k+1} - 1\}$.
- Indices in O correspond to odd indices from the original sequence $\{0, 1, \ldots, 2^{k+1} - 1\}$.

Note that the binary representations of the indices in E all end in 0, and those in O all end in 1. When these groups are combined back together, the resulting sequence is formed by first appending the bit-reversed even-index group E, followed by the bit-reversed odd-index group O.

Since the even-index group E and odd-index group O are already in bit-reversed order, combining them results in the bit-reversal of the entire sequence $\{0, 1, \ldots, 2^{k+1} - 1\}$.

Hence, by the principle of mathematical induction, the result holds for all n.

□

Figure 4.3 compares the two approaches for computing the Taylor expansion in each round of the Expand module, as described in the next section. We present the alternative Taylor Expansion algorithm using an iterative approach in Algorithm 4.6, where the input parameter r specifies how many iterations are skipped from the end of the algorithm. Specifically, the algorithm runs for $m-r-1$ iterations for the input length of 2^m. Algorithm 4.6, on input of size $n = 2^m$ and parameter r, involves $2^{m-1}(m-r-1)$ finite field additions. In the next section, we present our Expand module where we employed an alternative approach, as implemented in [34].

4.3.2 Expand Module

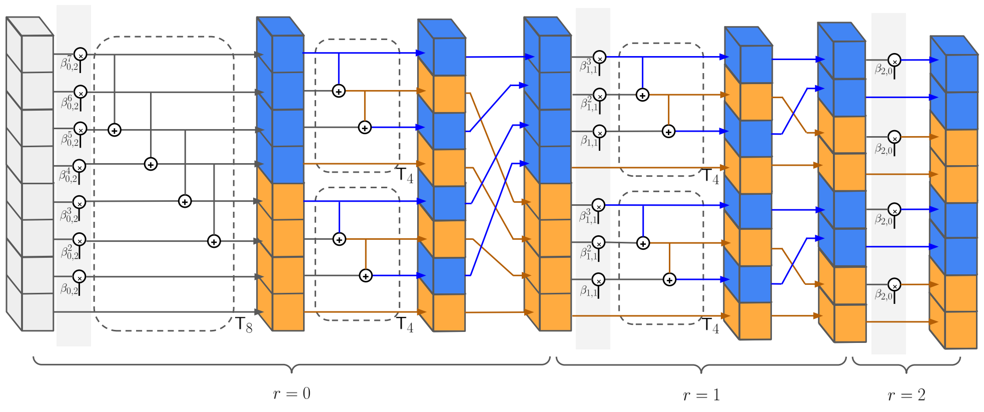
The Expand module involves multiple invocations of Algorithm 4.6, polynomial scaling (e.g., $f(\beta_m x)$), and computing basis vectors and shifts for the Aggregate module. As discussed in Section 4.3.1, we omit even-odd rearrangements, so the coefficients of terms with the same

Algorithm 4.6 Taylor Expansion ($\mathbf{f}_{\text{in}}, r$)

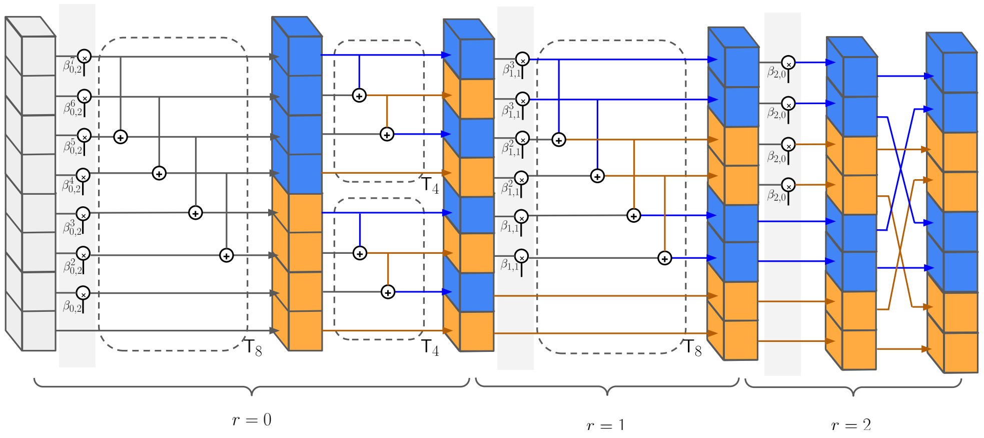
```
Require: $\mathbf{f}_{\text{in}} = (c_0, c_1, \dots, c_{n-1})$ and $r$ denotes the number of rounds reduction.
Ensure: $\mathbf{f}_{\text{out}}$
 1: $k \leftarrow m - 2$
 2: while $k \geq r$ do
 3:     $j \leftarrow 0$
 4:     while $j \leq n - 4 \cdot 2^k$ do        ▷ The following for-loop implements $\mathsf{T}_{4 \cdot 2^k}$
 5:         for $i = 0$ to $2^k - 1$ do
 6:             $c_{2 \cdot 2^k + i + j} \leftarrow c_{2 \cdot 2^k + i + j} + c_{3 \cdot 2^k + i + j}$
 7:             $c_{2^k + i + j} \leftarrow c_{2^k + i + j} + c_{2 \cdot 2^k + i + j}$
 8:         end for
 9:         $j \leftarrow j + 4 \cdot 2^k$        ▷ Sets the offset for the starting indices of $\mathsf{T}_{4 \cdot 2^k}$ modules
10:     end while
11:     $k \leftarrow k - 1$
12: end while
13: return $\mathbf{f}_{\text{out}} \leftarrow (c_0, c_1, \dots, c_{n-1})$
```

degree are placed next to each other. This allows Expand to multiply consecutive elements by the same scaling factor.

In the last round of the Expand module, $\mathbf{f}$ is rearranged in bit-reversal order. Let the bit-reversed index of j be denoted as j-rev. During the bit-reversal rearrangement process, the elements c_j where $j = j\text{-rev}$ are not swapped. These elements are referred to as fixed elements. For an input of length $n = 2^m$, the number of fixed elements is $2^{\lceil m/2 \rceil}$. Consequently, the algorithm requires $6 \times \frac{2^m - 2^{\lceil m/2 \rceil}}{2}$ memory accesses, as each swap involves 3 memory reads and 3 memory writes. Algorithm 4.8 details the Expand module. The Bit-Reversal Rearrangement algorithm is described in Algorithm 4.7.



(a) Original Approach: Groups even and odd indices after each Taylor expansion



(b) Alternative Approach: Rearranges the vector in bit-reversal order in the final round

Figure 4.3: The Expand module in the Gao FFT algorithm of length $n = 2^3$, which evaluates a polynomial $f(x) \in \mathbb{F}[x]$ of degree < 8 over $\theta + W_3$, where $W_3 = \langle \beta_0, \beta_1, \beta_2 \rangle$. In Figure (a), the Expand module groups even and odd indices after each Taylor expansion. In contrast, Figure (b) skips these rearrangements by using one T_8 module instead of two T_4 modules in round $r = 1$ then performs the bit-reversal in the final round. Additionally, in this figure, $\beta_{0,2} = \beta_2$, and $\beta_{1,1} = (\beta_1\beta_2^{-1})^2 - (\beta_1\beta_2^{-1})$. Given that $\beta_{1,0} = (\beta_0\beta_1^{-1})^2 - (\beta_0\beta_1^{-1})$, we can compute $\beta_{2,0} = (\beta_{1,0}\beta_{1,1}^{-1})^2 - (\beta_{1,0}\beta_{1,1}^{-1})$.

Algorithm 4.7 Bit Reverse Rearrangement ($\mathbf{f}_{\text{in}}$)

```
Require: $\mathbf{f}_{\text{in}} = (c_0, c_1, \ldots, c_j, \ldots, c_{n-1})$, where $n = 2^m$, and each index $j$ is represented as
    an $m$-bit integer.
Ensure: $\mathbf{f}_{\text{out}}$
 1: for $j = 0$ to $n - 1$ do
 2:     $j\text{-rev} \leftarrow 0$
 3:     $j_{\text{tmp}} \leftarrow j$
 4:     for $k = 0$ to $m - 1$ do
 5:         $j\text{-rev} \leftarrow (j\text{-rev} \ll 1)$ OR $(j_{\text{tmp}}$ AND $1)$
 6:         $j_{\text{tmp}} \leftarrow j_{\text{tmp}} \gg 1$
 7:     end for
 8:     if $j < j\text{-rev}$ then
 9:         Swap $c_j$ and $c_{j\text{-rev}}$
10:     end if
11: end for
12: return $\mathbf{f}_{\text{out}} \leftarrow (c_0, c_1, \ldots, c_{n-1})$
```

Algorithm 4.8 Expand $(\mathbf{f}_{\text{in}}, \theta, \{\beta_{0,0}, \ldots, \beta_{0,m-1}\})$

```
Require: $\mathbf{f}_{\text{in}} = (c_0, c_1, \ldots, c_{n-1})$, which represents $f(x) \in \mathbb{F}[x]$ of degree $< n = 2^m$, $\theta \in \mathbb{F}_{2^k}$ is the affine shift, and $\beta_{0,i} \in \mathbb{F}_{2^k}$ are the basis of $W_m$.
Ensure: $\mathbf{f}_{\text{out}}$, $\boldsymbol{\theta} = (\theta_0, \ldots, \theta_{m-1})$, $\boldsymbol{\Gamma} = (\mathbf{G}_0 = \emptyset, \mathbf{G}_1, \ldots, \mathbf{G}_{m-1})$, where $\theta_r$ and $\mathbf{G}_r = \{\gamma_{r,0}, \ldots, \gamma_{r,r-1}\}$ denote the affine shift and basis corresponding to round $r$ of the Aggregate module.
1: for $r = 0$ to $m - 1$ do                                   ▷ Scaling polynomials:
2:     $\psi \leftarrow 1$                                   ▷ $\psi$ denotes the scaling factor of each term
3:     offset $\leftarrow 2^r$
4:     while offset $\leq 2^m - 1$ do
5:         for $i = 0$ to $2^r - 1$ do
6:             $c_{\text{offset}+i} \leftarrow c_{\text{offset}+i} \times \psi$
7:         end for
8:         $\psi \leftarrow \psi \times \beta_{r,m-r-1}$
9:         offset $\leftarrow$ offset $+ 2^r$
10:    end while
11:    $(c_0, \ldots, c_{n-1}) \leftarrow$ TAYLOR EXPANSION$((c_0, \ldots, c_{n-1}), r)$        ▷ Algorithm 4.6
12:    for $i = 0$ to $m - r - 2$ do
13:        $\gamma_{m-r-1,i} \leftarrow \beta_{r,i} \times \beta_{r,m-r-1}^{-1}$
14:        $\beta_{r+1,i} \leftarrow \gamma_{m-r-1,i}^2 + \gamma_{m-r-1,i}$
15:    end for
16:    $\mathbf{G_{m-r-1}} \leftarrow (\gamma_{m-r-1,0}, \ldots, \gamma_{m-r-1,m-r-2})$
17:    $\theta_{m-r-1} \leftarrow \theta \times \beta_{r,m-r-1}^{-1}$
18:    $\theta \leftarrow \theta_{m-r-1}^2 + \theta_{m-r-1}$
19: end for
20: $(c_0, \ldots, c_{n-1}) \leftarrow$ BIT REVERSE REARRANGEMENT$((c_0, \ldots, c_{n-1}))$
21: return $\mathbf{f}_{\text{out}} \leftarrow (c_0, c_1, \ldots, c_{n-1})$
```

4.3.3 Aggregate Module

The Aggregate module combines 2^r adjacent elements in $\mathbf{f}$ during round r, where $0 \leq r \leq m-1$. Let $\{\eta_0, \eta_1, \ldots, \eta_{2^r-1}\}$ denote the elements in the affine subspace $\theta_r + \langle \gamma_{r,0}, \ldots, \gamma_{r,r-1} \rangle$ provided in the Expand module. At each round r, it computes those elements by spanning the mentioned affine subspace, which requires a total of $\sum_{i=0}^{r-1} 2^i = 2^r - 1$ finite field additions and $2^r + r$ memory accesses: r memory reads for retrieving the γ_is and 2^r memory writes for storing the η_ks. The algorithm for computing η_ks is described in Algorithm 4.9.

Algorithm 4.9 Span ($\mathbf{G} = \{\gamma_0, \ldots, \gamma_{r-1}\}, \theta$)

Require: $\mathbf{G} = \{\gamma_0, \ldots, \gamma_{r-1}\}$ is a linearly independent set over $\mathbb{F}_2$, and $\theta \in \mathbb{F}_{2^k}$ is an affine shift.

Ensure: $\{\eta_0, \eta_1, \ldots, \eta_{2^r-1}\} = \theta + \langle \gamma_0, \ldots, \gamma_{r-1} \rangle$

```
1: $\eta_0 \leftarrow \theta$
2: if $\mathbf{G} = \emptyset$ then
3:     return $\{\eta_0\}$
4: end if
5: for $i = 0$ to $r-1$ do
6:     for $k = 0$ to $2^i - 1$ do
7:         $\eta_{k+2^i} \leftarrow \eta_k + \gamma_i$
8:     end for
9: end for
10: return $\{\eta_0, \eta_1, \ldots, \eta_{2^r-1}\}$
```

4.3.4 Detailed Cost Analysis

We now compute the number of multiplications and additions required by the algorithm. From Algorithm 4.8, we know that at the rth iteration, we need to scale 2^r polynomials, each of degree 2^{m-r}. Thus, the multiplication for the scaling is given by $\sum_{r=0}^{m-1} 2^r \cdot (2^{m-r} - 1) = 2^m m - \sum_{r=0}^{m-1} 2^r = n \log_2 n - n + 1$.

From Algorithm 4.10, we know that at the rth iteration, the number of required multiplications is $2^r \cdot 2^{m-r-1} = 2^{m-1}$. Thus, the total multiplication cost in Algorithm 4.10 is $\sum_{r=0}^{m-1} 2^{m-1} = \frac{1}{2} n \log_2 n$. Therefore, the total number of multiplications in the Gao-Mateer algorithm is given by $n \log_2 n - n + 1 + \frac{1}{2} n \log_2 n = \frac{3}{2} n \log_2 n - n + 1$.

From Algorithm 4.6, we know that in the rth iteration, the number of additions due to the Taylor expansion is $2^{m-1}(m - r - 1)$. Thus, the total number of additions required for

Algorithm 4.10 Aggregate $(\mathbf{f}_{\text{in}}, \mathbf{\Gamma}, \boldsymbol{\theta})$

```
Require: $\mathbf{f}_{\text{in}} = (c_0, c_1, \ldots, c_{n-1})$ is a vector of length $n = 2^m$, $\boldsymbol{\theta} = (\theta_0, \ldots, \theta_{m-1})$, $\mathbf{\Gamma} = (\mathbf{G}_0 = \emptyset, \mathbf{G}_1, \ldots, \mathbf{G}_{m-1})$, where $\theta_r$ and $\mathbf{G}_r = \{\gamma_{r,0}, \ldots, \gamma_{r,r-1}\}$ denote the affine shift and basis corresponding to round $r$.
Ensure: $\mathbf{f}_{\text{out}}$ is the vector of evaluations of $f(x)$ over $\theta + W_m$.
1: for $r = 0$ to $m - 1$ do
2:     $\{\eta_0, \eta_1, \ldots, \eta_{2^r-1}\} \leftarrow \mathsf{Span}(\mathbf{G}_r, \theta_r)$
3:     for $j = 0$ to $2^{m-r-1} - 1$ do
4:         $d \leftarrow j \cdot 2^{r+1}$
5:         for $i = 0$ to $2^r - 1$ do
6:             $c_{d+i} \leftarrow c_{d+i} + c_{d+2^r+i} \times \eta_i$
7:             $c_{d+2^r+i} \leftarrow c_{d+2^r+i} + c_{d+i}$
8:         end for
9:     end for
10: end for
11: return $\mathbf{f}_{\text{out}} \leftarrow (c_0, c_1, \ldots, c_{n-1})$
```

the Taylor expansions is $\sum_{r=0}^{m-2} 2^{m-1}(m-r-1) = 2^{m-2}m(m-1)$. Also, in Algorithm 4.10, we know that at the rth iteration, the number of required additions is $2 \cdot 2^r \cdot 2^{m-r-1} = 2^m$. Therefore, the total addition cost for the algorithm is given by $2^{m-2}m(m-1) + m \cdot 2^m = \frac{1}{4}n(\log_2 n)^2 + \frac{3}{4}n \log_2 n$.

When performing the FFT over a subspace, at each rth iteration of Algorithm 4.10, we have $\eta_0 = 0$. Thus, for Steps 6 and 7, we need no multiplications are required, and only one addition is needed. Consequently, we must account for a reduction of $\sum_{r=0}^{m-1} 2^{m-r-1} = 2^m - 1 = n-1$ in both additions and multiplications. Therefore, the costs for multiplications and additions are changed to $\frac{3}{2}n\log_2 n - 2n + 2$ and $\frac{1}{4}n(\log_2 n)^2 + \frac{3}{4}n\log_2 n - n + 1$, respectively.

4.3.5 Optimization for Cantor Special Basis

If we have a Cantor special basis of dimension m, we can avoid the scaling in every iteration. We know that a Cantor special basis satisfy the following

$$\beta_0 = 1 \quad \text{and } S(\beta_i) = \beta_i^2 + \beta_i = \beta_{i-1} \quad \text{for } 1 \leq i \leq m-1,$$

where $S(x) = x^2 + x$. In addition, we know that $S^i(\beta_i) = \beta_0 = 1$ for $0 \leq i \leq m-1$ and $S^{i+\ell}(\beta_{i+\ell}) = \beta_\ell$ for any $i, \ell \geq 0$ with $i + \ell \leq m-1$. Now, consider the Cantor special basis in the reversed order, i.e.,

$$W_m = \langle \beta_{m-1}, \beta_{m-2}, \dots, \beta_1, 1\rangle = \langle \beta_{m-1}, S(\beta_{m-1}), \dots, S^{m-2}(\beta_{m-1}), 1\rangle.$$

Thus, we have $G = \langle \beta_{m-1}, S(\beta_{m-1}), \dots, S^{m-2}(\beta_{m-1}))$ and

$$D = \langle S(\beta_{m-1}), S^2(\beta_{m-1}), \dots, S^{m-1}(\beta_{m-1})) = \langle S(\beta_{m-1}), S^2(\beta_{m-1}), \dots, 1\rangle.$$

Thus, we do not need the scaling for the functions $f_0(x)$ and $f_1(x)$. Also, at the j-th iteration, G and D will be of the form

$$\begin{aligned} G^{(j)} &= \langle S^j(\beta_{m-1}), S^{j+1}(\beta_{m-1}), \dots, S^{m-2}(\beta_{m-1})\rangle \quad \text{and} \\ D^{(j)} &= \langle S^{j+1}(\beta_{m-1}), S^{j+2}(\beta_{m-1}), \dots, S^{m-1}(\beta_{m-1}) = 1\rangle. \end{aligned}$$

Therefore, at each iteration there is no need for scaling. Also, due to the chosen basis, computing the basis elements in $G^{(j)}$ and $D^{(j)}$ does not require any multiplications or additions. This can be done simply by selecting one fewer element from $G^{(j-1)}$ and $D^{(j-1)}$.

Detailed cost analysis of the optimized algorithm When using the Cantor basis in Algorithm 4.8, no scaling is required for the polynomials. All other steps in Algorithms 4.8 and 4.10 remain unchanged. Thus, the number of additions remains the same, as evaluated in Section 4.3.4. Consequently, the multiplication and addition costs are $\frac{1}{2}n\log_2 n$ and $\frac{1}{4}n(\log_2 n)^2 + \frac{3}{4}n\log_2 n$, respectively.

Furthermore, as discussed in Section 4.3.4, when performing the FFT over a subspace, the total multiplication and addition costs for the algorithm are $\frac{1}{2}n\log_2 n - n + 1$ and $\frac{1}{4}n(\log_2 n)^2 + \frac{3}{4}n\log_2 n - n + 1$, respectively.

Thus, by using Cantor special basis in the Gao-Mateer algorithm, we can efficiently compute the additive FFT of $f(x) \in \mathbb{F}_{2^k}[x]$ over $\theta + W_m$ where $\mathbb{F}_{2^k}$ contain a subfield $\mathbb{F}_{2^{2^\ell}}$ with $m \leq 2^\ell$.

4.3.6 Precomputation

In this section, we introduce two levels of precomputation. The first level precomputes the set $\boldsymbol{\beta} = \{\beta_{0,m-1}, \dots, \beta_{r,m-r-1}, \dots, \beta_{m-1,0}\}$, which is essential for determining the scaling factors. Additionally, it precomputes $\boldsymbol{\Gamma} = (\mathbf{G}_0 = \emptyset, \mathbf{G}_1, \dots, \mathbf{G}_{m-1})$ and $\boldsymbol{\theta} = (\theta_0, \dots, \theta_{m-1})$

required by the Aggregate module, initially provided by the Expand module. This precomputation requires m finite field elements for $\boldsymbol{\beta}$ and $\boldsymbol{\theta}$, along with $\sum_{i=0}^{m-1} i = m(m-1)/2$ finite field elements for $\boldsymbol{\Gamma}$. Consequently, this algorithm stores a total of $(m^2+3m)/2$ finite field elements. For the variant of the Gao-Mateer algorithm described in Section 4.3.5, the scaling operation is omitted, and each basis $\mathbf{G}_r$ is defined as $\{\beta_0, \beta_1, \ldots, \beta_{r-1}\}$. Therefore, this level of precomputation is not applicable for that.

Algorithm 4.11 Gao-Mateer Precomputation Level 1 (θ, $\{\beta_0, \beta_1, \ldots, \beta_{m-1}\}$)

```
Require: $\theta \in \mathbb{F}_{2^k}$ is the affine shift, and $\{\beta_0, \beta_1, \ldots, \beta_{m-1}\}$ is a linearly independent set
    over $\mathbb{F}_2$
Ensure: $\boldsymbol{\beta} = \{\beta_{0,m-1}, \ldots, \beta_{r,m-r-1}, \ldots, \beta_{m-1,0}\}$, $\boldsymbol{\theta} = (\theta_0, \ldots, \theta_{m-1})$, $\boldsymbol{\Gamma} = (\mathbf{G}_0 =$
    $\emptyset, \mathbf{G}_1, \ldots, \mathbf{G}_{m-1})$
 1: $\beta_{0,m-1} \leftarrow \beta_{m-1}$
 2: $\mathbf{G}_0 \leftarrow \emptyset$
 3: for $r = 0$ to $m-2$ do
 4:     for $i = 0$ to $m-r-2$ do
 5:         $\gamma_{m-r-1,i} \leftarrow \beta_{r,i} \times \beta_{r,m-r-1}^{-1}$
 6:         $\beta_{r+1,i} \leftarrow \gamma_{m-r-1,i}^2 + \gamma_{m-r-1,i}$
 7:     end for
 8:     $\mathbf{G}_{m-r-1} \leftarrow (\gamma_{m-r-1,0}, \ldots, \gamma_{m-r-1,m-r-2})$
 9:     $\theta_{m-r-1} \leftarrow \theta \times \beta_{r,m-r-1}^{-1}$
10:     $\theta \leftarrow \theta_{m-r-1}^2 + \theta_{m-r-1}$
11: end for
12: return $\boldsymbol{\beta} = \{\beta_{0,m-1}, \ldots, \beta_{r,m-r-1}, \ldots, \beta_{m-1,0}\}$, $\boldsymbol{\theta} = (\theta_0, \ldots, \theta_{m-1})$, $\boldsymbol{\Gamma} = (\mathbf{G}_0 =$
    $\emptyset, \mathbf{G}_1, \ldots, \mathbf{G}_{m-1})$
```

The second level of precomputation precomputes the powers of the actual scaling factors. Moreover, instead of storing the basis $\mathbf{G}_r$, it precomputes all elements in each affine subspace $\theta_r + \langle \gamma_{r,0}, \ldots, \gamma_{r,r-1} \rangle$. This requires storing $\sum_{r=0}^{m-1}(2^{m-r} - 1) = 2^{m+1} - m - 2$ finite field elements for all scaling factors stored in $\boldsymbol{\Psi} = (\boldsymbol{\psi}_0, \ldots, \boldsymbol{\psi}_{m-1})$ where $\boldsymbol{\psi}_r = (\psi_{r,1}, \ldots, \psi_{r,2^{m-r}-1})$ and $\psi_{r,d} = \beta_{r,m-r-1}^d$. Also, it stores $\sum_{r=0}^{m-1} 2^r = 2^m - 1$ finite field elements for all computed elements stored in $\mathbf{H} = (\boldsymbol{\eta}_0, \ldots, \boldsymbol{\eta}_{m-1})$, where $\boldsymbol{\eta}_r = \{\eta_0, \ldots, \eta_{2^r-1}\}$ are elements in $\theta_r + \langle \gamma_{r,0}, \ldots, \gamma_{r,r-1} \rangle$. Consequently, this algorithm stores a total of $3 \cdot 2^m - m - 3$ finite field elements. For the optimized variant described in Section 4.3.5, the scaling factor is not required, and we only need to obtain the elements in $\theta + \langle \beta_0, \beta_1, \ldots, \beta_{m-1} \rangle$, which equals 2^m finite field elements.

Algorithms 4.11 and 4.12 summarize those precomputations.

Algorithm 4.12 Gao-Mateer Precomputation Level 2 (θ, $\{\beta_0, \beta_1, \ldots, \beta_{m-1}\}$)

```
Require: $\theta \in \mathbb{F}_{2^k}$ is the affine shift, and $\{\beta_0, \beta_1, \ldots, \beta_{m-1}\}$ is a linearly independent set
    over $\mathbb{F}_2$
Ensure: $\boldsymbol{\Psi} = (\boldsymbol{\psi}_0, \ldots, \boldsymbol{\psi}_{m-1})$, $\mathbf{H} = (\boldsymbol{\eta}_0, \ldots, \boldsymbol{\eta}_{m-1})$, where each $\boldsymbol{\eta}_r = \{\eta_0, \ldots, \eta_{2^r-1}\}$ is the
    output of SPAN($\mathbf{G}_r, \theta_r$)
 1: $\boldsymbol{\beta}, \boldsymbol{\theta}, \boldsymbol{\Gamma} \leftarrow$ GAO PRECOMPUTATION LEVEL 1($\theta$, $\{\beta_0, \ldots, \beta_{m-1}\}$)
 2: for $r = 0$ to $m-1$ do
 3:     $\psi_{r,0} \leftarrow 1$                                   ▷ $\psi$ is the scaling factor of round $r$
 4:     for $j = 1$ to $2^{m-r} - 1$ do
 5:         $\psi_{r,j} \leftarrow \psi_{r,j-1} \times \beta_{r,m-r-1}$
 6:     end for
 7:     $\boldsymbol{\psi}_r \leftarrow (\psi_{r,1}, \ldots, \psi_{r,2^{m-r}-1})$                ▷ Note: $\psi_{r,0} = 1$ is skipped
 8:     $\boldsymbol{\eta}_r \leftarrow$ SPAN($\mathbf{G}_r, \theta_r$)
 9: end for
10: $\boldsymbol{\Psi} \leftarrow (\boldsymbol{\psi}_0, \ldots, \boldsymbol{\psi}_{m-1})$
11: $\mathbf{H} \leftarrow (\boldsymbol{\eta}_0, \ldots, \boldsymbol{\eta}_{m-1})$
12: return $\boldsymbol{\Psi}$, $\mathbf{H}$
```

4.4 Aurora FFT Complexity Analysis

Given a target security parameter $\boldsymbol{\lambda}$, and subsets H_1 and H_2, the size of the codeword domain (i.e., $|L|$) is determined. This size is important for the analysis of the FFT complexity of Aurora, as the input length of many FFT and IFFT instances is $|L|$. Before computing $|L|$, we need to review how the number of queries to the codeword is determined based on the target security parameter.

4.4.1 Soundness Errors: Queries and Repetition Analysis

Given $\boldsymbol{\lambda}$, $\boldsymbol{\epsilon_q}$ and $\boldsymbol{\epsilon_i}$ represent query and interactive soundness errors such that $\boldsymbol{\epsilon_q} + \boldsymbol{\epsilon_i} < 2^{-\boldsymbol{\lambda}}$, where $2^{-\boldsymbol{\lambda}-1}$ is allocated to each. According to [35, Theorem 4],

$$\boldsymbol{\epsilon_i} = \left(\frac{\eta+1}{|\mathbb{F}|}\right)^{\lambda_i} + \left(\frac{|L|}{|\mathbb{F}|}\right)^{\lambda_i'} + \epsilon_i^{\text{FRI}}, \text{ and } \boldsymbol{\epsilon_q} = \epsilon_q^{\text{FRI}},$$

where $\left(\frac{\eta+1}{|\mathbb{F}|}\right)^{\lambda_i}$ and $\left(\frac{|L|}{|\mathbb{F}|}\right)^{\lambda_i'}$ denote the lincheck and LDT soundness errors respectively. ϵ_i^{FRI} and ϵ_q^{FRI} denote the interactive and query soundness errors in FRI, respectively. Each term in $\boldsymbol{\epsilon_i}$ gets $2^{-\boldsymbol{\lambda}-3}$ and $\boldsymbol{\epsilon_q}$ gets $2^{-\boldsymbol{\lambda}-1}$ soundness error bound. The codeword is queried during FRI. Given, the target proximity parameter in (2.9), the number of query repetitions in FRI is:

$$\lambda_q^{\text{FRI}} = \frac{\log(\epsilon_q^{\text{FRI}})}{\log\left(1 - \min\left(\delta, \frac{1-3\rho-2^{\phi}/\sqrt{|L|}}{4}\right)\right)}. \tag{4.4}$$

Moreover, the required lincheck and LDT repetition parameters are determined as: $\lambda_i = \frac{-\boldsymbol{\lambda}-3}{\log\left(\frac{\eta+1}{|\mathbb{F}|}\right)}$, and $\lambda_i' = \frac{-\boldsymbol{\lambda}-3}{\log\left(\frac{|L|}{|\mathbb{F}|}\right)}$.

4.4.2 Codeword Size

The maximum degree of the polynomial in the codeword of size $|L|$ is $d = 2t + 2\mathtt{b}$ (according to Table 2.1), where $\mathtt{b}$ is determined by the expected number of queries to the corresponding codeword. The number of queries to the codeword is $\mathtt{b} = \lambda_q^{\text{FRI}} \cdot 2^{\phi}$, where according to (4.4), λ_q^{FRI} also relies on $|L|$. To solve this loop, we initialize $|L| = 4t/\rho$ and $\mathtt{b}$ as mentioned. Then, we check the following condition: Let $\mathtt{r} = \lfloor \log(\rho|L|)/\phi \rfloor$ denote the number of reductions in FRI, $d' = 2t + \mathtt{b} - 1$ (lincheck maximum degree) must be divisible by $2^{\mathtt{r}\phi}$, otherwise, d' must be increased to the nearest multiple of $2^{\mathtt{r}\phi}$. Then, we check if the new $d' \leq \rho|L|$, otherwise, increase the dimension of L by one. Again, we compute $\mathtt{b}$ and check if it is the same as it was; otherwise, we update that and again check the condition with the new L. This iteration is continued until the $\mathtt{b}$ remains unchanged, at which point $|L|$ is determined.

4.4.3 Construction of H_1, H_2, and L

Let $\{\beta_0, \beta_1, \ldots, \beta_{k-1}\}$ represent basis elements of $\mathbb{F}_{2^k}$. Then, we construct subsets $H_1 = \langle \beta_0, \beta_1, \cdots \beta_{\lceil \log \eta \rceil} \rangle$ and $H_2 = \langle \beta_0, \beta_1, \cdots \beta_{\lceil \log(\mu+1) \rceil} \rangle$ and the affine subspace $L = \beta_{\lceil \log(|L|) \rceil + 1} + \langle \beta_0, \beta_1, \cdots \beta_{\lceil \log(|L|) \rceil} \rangle$. In this way, $L \cap (H_1 \cup H_2) = \emptyset$. If the basis elements are Cantor special basis, the Cantor FFT can be used without precomputation due to this special construction, as detailed in Section 4.2.5.

4.4.4 FFT/IFFT Calls

Table 4.2 summarizes the FFT and IFFT calls for computing the codewords in Table 2.1. Virtual oracles are used for consistency checks, and are not included in the random combination of codewords input to the FRI protocol.

Table 4.2: The FFT and IFFT calls in the Aurora zkSNARK protocol

FFT Calls	**Description**
$\lambda_i\times$ FFT of len. $\lvert L\rvert$	Compute the masking oracle $\hat{\mathbf{r}}_\ell$: Evaluate the polynomial r_ℓ of degree $< 2t+b-1$ over L, where $\ell \in [1,\lambda_i]$.
(1) IFFT of len. $\kappa+1$ (2) FFT of len. $\mu+1$ (3) IFFT of len. $\mu+1$ (4) FFT of len $\lvert L\rvert$	Compute the oracle $\hat{\mathbf{f}}_{\mathbf{w}}$: (1) Interpolate $\mathbf{v}$ to get $f_{(1,\mathbf{v})}$ of degree $< \kappa+1$. (2) Evaluate $f_{(1,\mathbf{v})}$ over H_2. Then, computing $\mathbf{f}'_{\mathbf{w}} = \mathbf{w}_{i-\kappa-1}[0:\mu-\kappa-1] - \mathbf{f}_{(1,\mathbf{v})}[\kappa+1:\mu]$ (3) Interpolate $\mathbf{f}'_{\mathbf{w}}$ over H_2 to get $f'_{\mathbf{w}}$ of degree $< \mu+1$. Then, divide $f'_{\mathbf{w}}$ by $\mathbb{Z}_{\{h_0,\ldots,h_\kappa\}}$ to get $f_{\mathbf{w}}$. (4) Evaluate $f^*_{\mathbf{w}}$ over L.
(1) $3\times$ IFFT of len. η (2) $3\times$ FFT of len. $\lvert L\rvert$	Compute the oracles $\hat{\mathbf{f}}_{\mathbf{Az}}$, $\hat{\mathbf{f}}_{\mathbf{Bz}}$, and $\hat{\mathbf{f}}_{\mathbf{Cz}}$: (1) Interpolate $\mathbf{Az}$, $\mathbf{Bz}$, and $\mathbf{Cz}$ to get $f_{\mathbf{Az}}$, $f_{\mathbf{Bz}}$, $f_{\mathbf{Cz}}$ of degree $< \eta$. (2) Evaluate $f^*_{\mathbf{Az}}$, $f^*_{\mathbf{Bz}}$, $f^*_{\mathbf{Cz}}$ over L.
$\lambda'_i\times$ FFT of len. $\lvert L\rvert$	Compute the masking oracle $\hat{\mathbf{r}}'_\ell$: Evaluate the polynomial r'_ℓ of degree $< 2t+2b$ over L, where $\ell \in [1,\lambda'_i]$.
$\lambda_i\times$ IFFT of len. t	Interpolate the polynomial p_{α_ℓ} in (2.7), where $\ell \in [1,\lambda_i]$.
$\lambda_i\times$ IFFT of len. t	Interpolate the polynomial $p^{ABC}_{\alpha_\ell}$ in (2.7), where $\ell \in [1,\lambda_i]$.
FFT of len. $\lvert L\rvert$	Compute the virtual oracle $\hat{\mathbf{f}}_{\mathbf{z}} := \hat{\mathbf{f}}_{\mathbf{w}} \cdot \mathbb{Z}_{\{h_0,\ldots,h_\kappa\}} + \hat{\mathbf{f}}_{(1,\mathbf{v})}(h_i)$ which requires Evaluating $f_{(1,\mathbf{v})}$ over L.
$\lambda_i\times 2\times$FFT of len. $\lvert L\rvert$	Compute virtual oracles $\hat{\mathbf{q}}^M_\ell := \hat{\mathbf{f}}_{\mathbf{Mz}} \cdot \hat{\mathbf{p}}_{\alpha_\ell} - \hat{\mathbf{f}}_{\mathbf{z}} \cdot \hat{\mathbf{p}}^{ABC}_{\alpha_\ell}$, where $\ell \in [1,\lambda_i]$ and $M \in \{A,B,C\}$, by evaluating p_{α_ℓ} and $p^{ABC}_{\alpha_\ell}$ over L.
(1) $\lambda_i\times$IFFT of len. d where $d = 2^{\lceil \log(t+b)\rceil}$ (2) $\lambda_i\times$FFT of len. $\lvert L\rvert$	Compute the oracle $\hat{\mathbf{h}}_\ell$: (1) Interpolate $\sum_{M\in\{A,B,C\}} s^M_\ell \hat{\mathbf{q}}^M_\ell$ to get a polynomial of degree less than the minimum power of two greater than $t+\mathtt{b}$.Then, compute the polynomial h according to Table 2.1 (2) Evaluate h over L

4.5 Complexity Analysis and Benchmarking

In this section, we provide a comprehensive comparison of the Cantor and the Gao-Mateer algorithms, along with their respective variants. The short names for these algorithms are summarized in Table 4.3, and will be referenced throughout this section for clarity.

Table 4.3: Short names for the algorithms used in the comparison.

Name	**Description**
Cantor	Cantor Algorithm [Section 4.2]
Cantor PC	Cantor Algorithm with precomputation [Section 4.2.5]
GM	Gao-Mateer algorithm (implemented libiop [34])
GM PCL1	Gao-Mateer algorithm with level 1 precomputation [Section 4.3.6]
GM PCL2	Gao-Mateer algorithm with level 2 precomputation [Section 4.3.6]
GM CO	Gao-Mateer algorithm using Cantor Basis (Cantor Optimized)
GM CO PCL2	Gao-Mateer algorithm using Cantor Basis with level 2 precomputation

Table 4.4 presents the number of additions and multiplications for the precomputed variant of the FFT algorithms, while Table 4.5 compares the space requirements of each precomputation method. Figure 4.5 illustrates the contribution of each sub-algorithm in the Gao-Mateer FFT implementation from [34]. The basis computations, which are precomputed in GM PCL1, require fewer resources as m increases. In contrast, the computational cost of bit-reversal rearrangement and Taylor expansion grows more rapidly for larger m, significantly impacting the overall FFT runtime.

Table 4.4: Comparison of the number of additions and multiplications

Name	**# Additions**	**# Multiplications**
Cantor PC	$\frac{1}{2}n\log_2 n + \frac{1}{2}n\sum_{r=0}^{\log_2(n)-1} 2^{\mathrm{wt}(r)}$	$\frac{1}{2}n\log_2 n$
GM PCL2	$\frac{1}{4}n(\log_2 n)^2 + \frac{3}{4}n\log_2$	$\frac{3}{2}n\log_2 n - n + 1$
GM CO PCL2	$\frac{1}{4}n(\log_2 n)^2 + \frac{3}{4}n\log_2$	$\frac{1}{2}n\log_2 n$

We implemented the all variants of the Cantor and Gao-Mateer algorithms in C++ except Gao-Mateer which has been implemented in the libiop library [34]. We used the same library, libff [37], for finite field operations as in libiop. We selected the field $\mathbb{F}_{2^{256}}$ for our

Table 4.5: The storage required for precomputation of each algorithm

Method	**Storage**
Cantor PC	$2^m - 1$ elements in $\mathbb{F}_{2^k}$ and $\sum_{i=0}^{m-1}\left(2^{\mathrm{wt}(i)} - 1\right)$ integers [Section 4.2.5]
GM PCL1	$(m^2 + 3m)/2$ elements in $\mathbb{F}_{2^k}$ [Section 4.3.6]
GM PCL2	$3 \cdot 2^m - m - 3$ elements in $\mathbb{F}_{2^k}$ [Section 4.3.6]
GM CO PCL2	2^m elements in $\mathbb{F}_{2^k}$ [Section 4.3.6]

experiments. Each field element is represented using four 64-bit limbs. Field multiplication is performed using 13 carry-less multiplication (`CLMUL`) instructions, commonly available on many x86 processors [97], following the Karatsuba multiplication [109] method for multiplying 4-limb values [34]. This approach is provided in Appendix B.

The benchmarks were conducted on a system featuring an AMD Ryzen 9 9950x @ 5.7 CPU, equipped with 64 GB of DDR5 RAM. The CPU governor was set to *Performance*, to ensure the CPU operates at its maximum clock speed throughout testing, thus minimizes fluctuations caused by power-saving mechanisms. We executed each algorithm for 1000 different random polynomials $f(x) \in \mathbb{F}_{2^{256}}[x]$ of degree less than $n = 2^m$, where $4 \leq m \leq 28$.

Figure 4.4 compares the runtime of the FFT algorithms. It shows that the time savings in Cantor PC are consistent across different input lengths. However, for Gao PCL1, Gao PCL2, and Gao CO PCL2, the savings become insignificant for larger m. The reason is that GM requires extensive memory access for Taylor expansion, the Aggregate module, and bit-reversal rearrangements, which suppresses the savings gained from the precomputations. Figure 4.5 depicts the fraction of each sub-algorithm in GM.

4.6 Summary

This work demonstrates the effectiveness of leveraging the Cantor special basis to enable the use of the Cantor additive FFT algorithm in post-quantum secure zkSNARKs, with a focus on Aurora [35]. Our implementation shows that replacing the Gao-Mateer FFT with the Cantor FFT results in a significant reduction in computation time. Furthermore, we present a detailed theoretical analysis of the computational costs of the Cantor FFT, including exact counts of additions and multiplications, and an evaluation of FFT call complexity within the encoding of the R1CS in Aurora, based on the number of constraints, variables, and the target security parameter. Additionally, we introduce optimized building blocks for the Cantor FFT implementation and propose precomputation techniques that

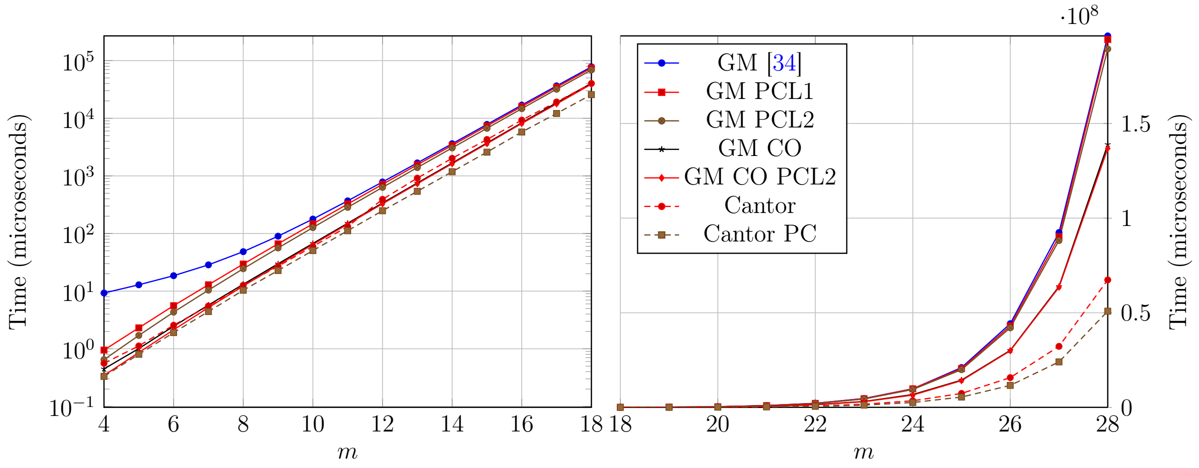


Figure 4.4: Performance comparison of variances of Gao-Mateer and Cantor additive FFT algorithms of length $n = 2^m$ for $4 \leq m \leq 28$. The left plot is in a logarithmic scale.

reduce overhead in both the Cantor and Gao-Mateer FFT algorithms when the basis of the affine subspaces is predetermined.

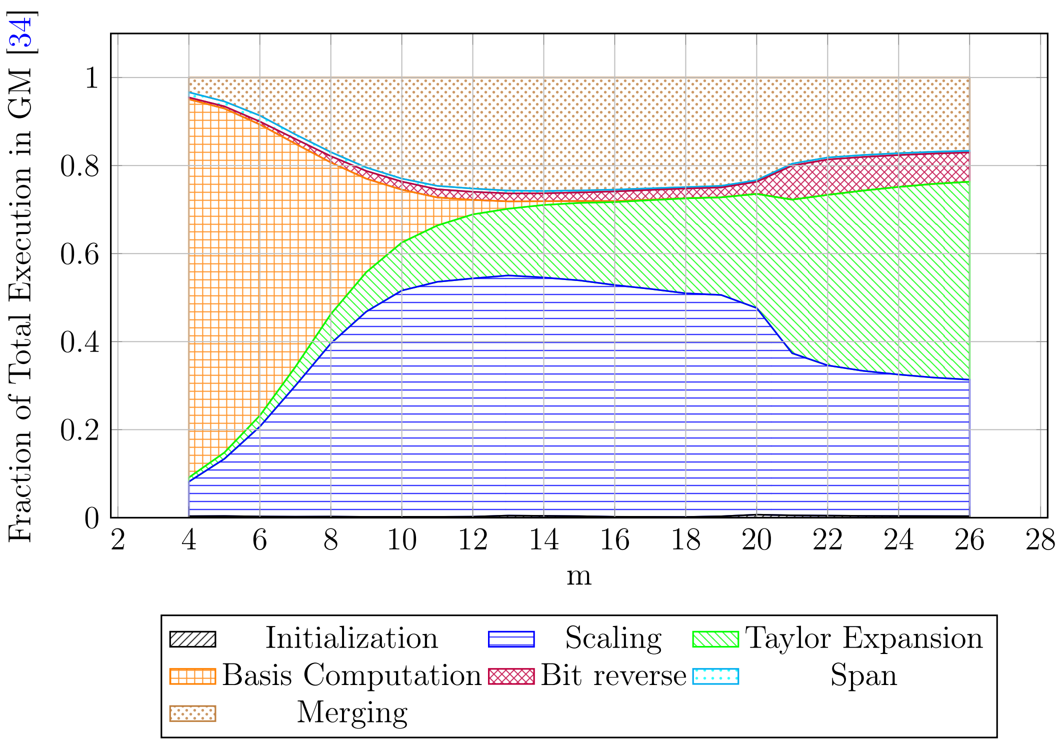


Figure 4.5: The fraction of execution time for each sub-algorithm in the Gao-Mateer FFT algorithm over $\mathbb{F}_{2^k}$ implemented in [34]. The execution times of the Taylor expansion and bit-reversal sub-algorithms increase more rapidly with the input size. *Initialization* is a one-time computing for copying the input polynomial to a new vector. *Span* and *Merging* are the Span function and the nested for loop in Aggregate Module

Chapter 5

Polaris zkSNARK Optimization

Declaration of Contributions

This chapter is based on [19] I have co-authored the paper and my main contributions are as follows:

- Design of the GKR circuit and wrote Section 5.3.

5.1 Introduction

The efficient implementation of the post-quantum secure zkSNARK protocols is crucial for enabling practical deployment in real-world applications. To address concerns regarding efficiency, this chapter proposes the fast implementation of Polaris by emphasizing on the optimization of GKR and FRI protocols. In this chapter, we present an instantiation of the FRI protocol. This instantiation eliminates the field inversion operations in both the Commit phase and Query phase, expecting to show better efficiency. Also, we present an instantiation of the GKR circuit tailored for the Polaris implementation. By designing the circuit as a satisfiability circuit, we ensure the verifiable computation of values essential for the Polaris protocol while minimizing the number of gates. This would reduce the communication overhead, and the verifier and the prover complexities.

5.2 FRI Instantiation

In the FRI protocol, let r be the number of rounds, let $\{\beta_0 = 1, \beta_1, \beta_2, \cdots, \beta_{191}\}$ be one of the $\mathbb{F}_2$-basis of $\mathbb{F}_{2^{192}}$ defined as:

$$\mathbb{F}_{2^{192}} := \mathbb{F}_{2^{64}}[Y]/(Y^3 + Y + 1), \tag{5.1}$$

where

$$\mathbb{F}_{2^{64}} := \mathbb{F}_2[X]/(X^{64} + X^4 + X^3 + X + 1). \tag{5.2}$$

Evaluation Domains. We assume that the verifier and prover have agreed upon the evaluation domains L_k $(0 \le k \le \mathsf{r})$, whose sizes are the powers of 2, and $m = \log_2(|L_0|)$. Those affine subspaces are adopted in Preon [?]. The evaluation domains are recursively defined as follows. First,

$$L_0 =< \beta_0, \beta_1, \cdots, \beta_{m-1} > +\beta_m. \tag{5.3}$$

For an integer i $(0 \le i \le 2^m - 1)$, its binary expression is

$$i = (i_{m-1} i_{m-2} \cdots i_1 i_0)_2 = i_0 + i_1 \cdot 2 + \cdots + i_{m-1} \cdot 2^{m-1}, i_j \in \{0, 1\}.$$

Then we can define the i-th element in L_0 as

$$L_0[i] = (i_0 \cdot \beta_0 + i_1 \cdot \beta_1 + \cdots + i_{m-1} \cdot \beta_{m-1}) + \beta_m,\ 0 \le i < 2^m. \tag{5.4}$$

Let us define the polynomial

$$q_0(X) = X(X - \beta_0) \tag{5.5}$$

and

$$\beta_j^{(1)} = q_0(\beta_{j+1}),\ 0 \le j \le m - 1,$$

then L_1 is defined as

$$L_1 = q_0(L_0) =< \beta_0^{(1)}, \beta_1^{(1)}, \cdots, \beta_{m-2}^{(1)} > +\beta_{m-1}^{(1)}, \tag{5.6}$$

and the i-th element in L_1 is

$$L_1[i] = (i_0 \cdot \beta_0^{(1)} + i_1 \cdot \beta_1^{(1)} + \cdots + i_{m-2} \cdot \beta_{m-2}^{(1)}) + \beta_{m-1}^{(1)},\ 0 \le i < 2^{m-1}.$$

We can thus recursively define that, for $1 \le k \le m-1$,

$$\begin{aligned} q_k(X) &= X(X-\beta_0^{(k)}), \\ \beta_j^{(k+1)} &= q_k(\beta_{j+1}^{(k)}), \text{ for } 0 \le j \le m-k-1, \\ L_{k+1} &= q_k(L_k) =< \beta_0^{(k+1)}, \beta_1^{(k+1)}, \cdots, \beta_{m-k-2}^{(k+1)} > + \beta_{m-k-1}^{(k+1)}, \end{aligned} \tag{5.7}$$

and for $0 \le i < 2^{m-k-1}$, the i-th element of L_{k+1} is defined as

$$L_{k+1}[i] = (i_0 \cdot \beta_0^{(k+1)} + i_1 \cdot \beta_1^{(k+1)} + \cdots + i_{m-1} \cdot \beta_{m-k-2}^{(k+1)}) + \beta_{m-k-1}^{(k+1)}.$$

FRI Commit Phase.

Prover's input: $f^{(0)} : L_0 \to \mathbb{F}$, a purported Reed Solomn codeword corresponding to polynomial $f_0(X)$, with rate ρ.

Loop for $0 \le k \le \mathsf{r}-1$:

1. Prover commits to all codewords.
   - $f^{(k)} : L_k \to \mathbb{F}$ is recursively defined in Step 3.
   - Prover computes a Merkle commitment to $f^{(k)}$ and sends out the Merkle root.
2. Verifier sends a uniformly random $\alpha^{(k)} \in \mathbb{F}$.
3. Prover defines the codeword $f^{(k+1)}$ with domain L_{k+1}, such that for each $0 \le i < |L_{k+1}|$,
   - It is easy to check that

     $$q_k(L_k[2i]) = q_k(L_k[2i+1]) = L_{k+1}[i].$$

   - The values in codeword $f^{(k+1)}$ is derived from the previous codeword,

     $$f_{k+1}(L_{k+1}[i]) = \frac{f_k(L_k[2i]) - f_k(L_k[2i+1])}{L_k[2i] - L_k[2i+1]}(\alpha^{(k)} - L_k[2i]) + f_k(L_k[2i]).$$

   Here the denominator $L_k[2i] - L_k[2i+1] = \beta_0^{(k)}$, whose inverse can be precomputed.

For $k = \mathsf{r}$:

- $f^{(\mathsf{r})} : L_{\mathsf{r}} \to \mathbb{F}$ is defined in Step 2.
- prover sends out the last codeword $f^{(\mathsf{r})}$.

FRI Query Phase.

1. Verifier extracts all Merkle roots, all challenges $\alpha^{(0)}, \alpha^{(1)}, \cdots, \alpha^{(\mathsf{r}-1)}$, and the last codeword $f^{(\mathsf{r})}$. Verifier has oracle access to $f^{(0)}, f^{(1)}, \cdots, f^{(\mathsf{r}-1)}$.
2. Verifier computes the interpolant $f_{\mathsf{r}}(X)$ from points $f^{(\mathsf{r})}$, then check if the degree of $f_{\mathsf{r}}(X)$ is no more than $\rho \cdot |L^{(\mathsf{r})}| - 1$. If not, reject.
3. Verifier does the consistency check between two neighboring codewords.

   Repeat ℓ times:

   - Sample random index $s^{(0)} = i$ from $0 \le i < |L_0|$, and for $0 \le k \le \mathsf{r} - 1$, compute $s^{(k+1)} = \lfloor s^{(k)}/2 \rfloor$.
   - If $s^{(\mathsf{r})}$ is repeated, resample $s^{(0)}$.
   - **Round consistency check**:

     Denote
     $$x_1^{(k)} = L_k[2s^{(k+1)}],\ x_2^{(k)} = L_k[2s^{(k+1)} + 1],$$
     the verifier first queries
     $$f_{k+1}(L_{k+1}[s^{(k+1)}]), f_k(x_1^{(k)}), f_k(x_2^{(k)}),$$
     and checks the Merkle commit paths for those three points, then checks that for every $k \in \{0, 1, 2, \cdots, \mathsf{r} - 1\}$,
     $$f_{k+1}(L_{k+1}[s^{(k+1)}]) = \frac{f_k(x_1^{(k)}) - f_k(x_2^{(k)})}{x_1^{(k)} - x_2^{(k)}}(\alpha^{(k)} - x_1^{(k)}) + f_k(x_1^{(k)}). \tag{5.8}$$
     If any one equation of the consistency check fails, reject.

     Notice that $x_1^{(k)} - x_2^{(k)} = \beta_0^{(k)}$, whose inverse can be precomputed.
4. Accept if all checks pass. This implies that the degree of the original polynomial $f_0(X)$ is no more than $\rho \cdot |L_0| - 1$.

5.3 GKR Circuit

In this section we are going to present the circuit $\mathfrak{C}$ to provide a verifiable computation for $C_M(r_x, r_y)$. By considering that the arithmetic circuit can only include addition and multiplication operations, computing the multiplicative inverse is expensive. Therefore, we turn the straightline computation, in which we should compute the multiplicative inverse, into an *satisfiability* circuit instance [169]. Therefore, the circuit $\mathfrak{C}$ receives a set of inputs that includes $\overline{\mathsf{c}^{(d)}} = \{\mathsf{c}^{(d)}(i) \mid i \in [n]\}$, alongside r_x, r_y, $\overline{\mathsf{row}} = \{\mathsf{row}(i) \mid i \in [n]\}$, $\overline{\mathsf{col}} = \{\mathsf{col}(i) \mid i \in [n]\}$, and $\overline{\mathsf{val}} = \{\mathsf{val}(i) \mid i \in [n]\}$, where d denotes the depth of the circuit and $\mathsf{c}^{(\ell)}(i)$ is defined as follows:

$$\mathsf{c}^{(\ell)}(i) := \begin{cases} \frac{\mathsf{val}(i)}{(r_x - \mathsf{row}(i))(r_y - \mathsf{col}(i))}, & \ell = d,\ i \in [n] \\ \mathsf{c}^{(\ell+1)}(2i-1) + \mathsf{c}^{(\ell+1)}(2i), & \ell \in [0, d), i \in [2^{\ell-d} n] \\ 0 & \text{otherwise.} \end{cases} \tag{5.9}$$

This definition specifies that $\mathfrak{C}$ encompasses all terms included in the summation outlined in Equation (2.19), utilizing these terms as inputs (where $\ell = d$). To enable verifiable computation of this summation, $\mathfrak{C}$ is equipped with a *summation* component structured as a binary tree. Within this structure, for any level $\ell < d$, the function $\mathsf{c}^{(\ell)}(i)$ calculates the sum of two preceding terms from the immediately lower layer (i.e., layer $\ell + 1$), using addition gates. In Equation (5.9), we simplify our notation by assuming a balanced binary tree structure for ease of definition. This assumption entails that the input layer, denoted by $\mathsf{c}^{(d)}(i)$, comprises a power of two elements. Consequently, the number of layers are $d = \log n$. The output layer, serving as the tree's root, is designated by $\mathsf{c}^{(0)}(1) = C_M(r_x, r_y)$. However, this binary structure is not a strict requirement for the actual implementation. In practice, given that each addition gate necessitates two inputs, layers featuring an odd number of elements incorporate a zero as the supplemental input for the subsequent layer. This zero is consistently available at every layer of the circuit.

For $\mathfrak{C}$ to qualify as a satisfiability circuit, it must verify that each asserted term $\mathsf{c}^{(d)}(i)$ aligns with the parameters r_x, r_y, $\mathsf{row}(i)$, $\mathsf{col}(i)$, and $\mathsf{val}(i)$. Consequently, we need to design a *consistency* component embedded to the circuit. The number of layers $d = \log n$ is determined by the summation component presented above.

Layer $\ell = d - 1$: (The immediate layer beyond the input layer) $\alpha(i) = r_x - \mathsf{row}(i)$, $\beta(i) = r_y - \mathsf{col}(i)$, where $\overline{\alpha} = \{\alpha(i) \mid i \in [n]\}$ and $\overline{\beta} = \{\beta(i) \mid i \in [n]\}$ are realized by addition gates.

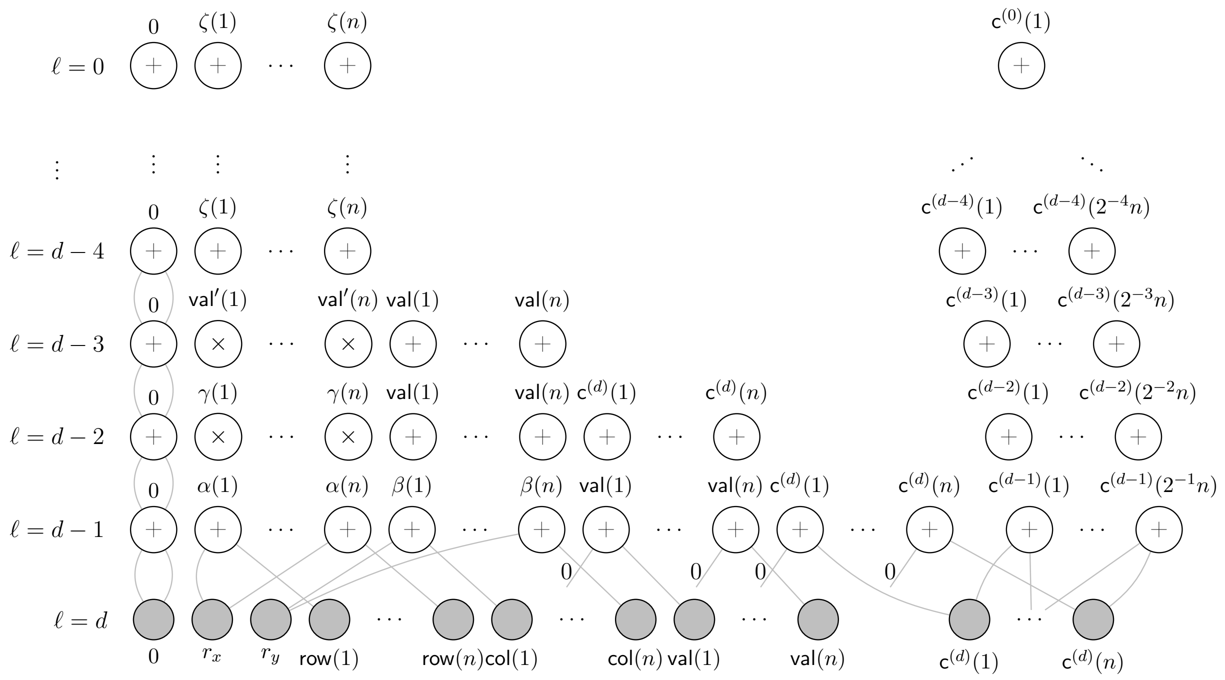


Figure 5.1: The GKR satisfiability circuit for calculating $C_M(r_x, r_y)$ (Equation (2.19)). Sample connections are shown for clarity. $\forall i, \zeta(i) = 0$, and $\mathsf{c}^{(0)}(1) = C_M(r_x, r_y)$.

Layer $\ell = d-2$: $\gamma(i) = \alpha(i)\beta(i)$ for $i \in [n]$, where $\overline{\gamma} = \{\gamma(i) \mid i \in [n]\}$ is realized by multiplication gates

Layer $\ell = d-3$: $\mathsf{val}'(i) = \gamma(i)\mathsf{c}^{(d)}(i)$, where $\mathsf{c}^{(d)}(i)$ is copied to this layer from the input layer by being added to zero in previous two layers. This zero is consistently available at every layer of the circuit. $\overline{\mathsf{val}'} = \{\mathsf{val}'(i) \mid i \in [n]\}$ is realized by multiplication gates.

Layer $\ell = d-4$: $\zeta(i) = \mathsf{val}'(i) + \mathsf{val}(i)$, where $\mathsf{val}(i)$ is also copied from the input layer to this layer by being added to zero. This layer performs an addition of $\mathsf{val}'(i)$ to $\mathsf{val}(i)$, such that, when $\mathsf{val}'(i)$ equals $\mathsf{val}(i)$, $\zeta(i) = 0$, since the addition is equivalent to the XOR operation in this field. This property ensures $\mathsf{val}'(i)$ and $\mathsf{val}(i)$ are equal. $\overline{\zeta} = \{\zeta(i) \mid i \in [n]\}$ is realized by addition gates.

Layers $\ell < d-4$: $\overline{\zeta}$ is copied to the upper layer till the output layer.

Figure 5.1 illustrates the GKR circuit used in Polaris. For clarity, only connections from the input layer are shown, as the other connections follow a similar pattern and have been omitted from this visual representation.

The summation component of the presented GKR circuit consists of $2n - 1$ gates, under the assumption that it forms a balanced binary tree (i.e., n is a power of two). If this condition is not met, the circuit will have a number of gates that is close to this figure. The consistency component of the circuit has $(n + 1)\log n + 7n + 2$ gates. Consequently the size of the circuit is $S = (n + 1)\log n + 9n + 1$ ($2n$ of them are multiplication and the rest are addition gates). The prover only needs to send $\mathsf{c}^{(0)}(1)$ to the verifier as the output of the circuit because the verifier assumes that the other $n + 1$ gate values should be zero; therefore, $S_0 = 1$ (S_0 is the size of the output layer). Note that the depth of the circuit is $d = \log(n)$. According to Section 2.4.1, the communication cost is $O(S_0 + d\log(S)) = O(\log(n) \cdot \log((n+1)\log n + 9n))$ which simplifies to $O(\log(n) \cdot \log(n \log n)) = O((\log(n))^2)$.

The input size to the circuit is $\nu = 4n + 3$. According to Section 2.4.1, the verifier's computation cost should be $O(\nu + d\log(S)) = O(n + \log(n) \cdot (n\log(n))) = O(n(\log(n))^2)$. However, in the implementation of the GKR protocol, this computation can be delegated to the prover via verifiable polynomial delegation (VPD) schemes such as Virgo [185]. Finally, the prover's computation is $O(S^3) = O((n\log(n))^3)$.

5.4 Conclusion

In this chapter, we introduced the Polaris protocol and its construction components. We presented an instantiation of the GKR circuit to minimize the number of gates, with the aim to reduce communication overhead, as well as the verifier and prover's complexities. We also explained an instantiation of the FRI protocol that eliminates the field inversion operations in both the Commit phase and Query phase, while would help to achieve better efficiency. The complete performance benchmark and the expected improvements will be demonstrated in the upcoming full implementation of Polaris.

Chapter 6

Zupply Framework Design

Declaration of Contributions

This chapter is based on [17]. I am the sole author of this chapter.

6.1 Introduction

The Zupply framework facilitates the authenticated and privacy-preserving, decentralized maintenance of directed acyclic graphs (DAGs). DAGs, which are pivotal in representing data flows, find utility in diverse applications. One notable application is in supply chain management (SCM), where a DAG is instrumental in tracking a product's progression. In this context, each DAG node represents a unique stage in the product's lifecycle, facilitating an understanding of the sequence and interdependencies of these stages. DAG data structure has the flexibility to either extend a sequence, divide it into two, or merge two sequences into one. Beyond SCM, DAG structured data (or simply DAG) is also employed in various other domains, including version control systems (VCS) (e.g., Git [86] and Mercurial [133]). Proposing a practical, fully decentralized, and trustless scheme for maintaining DAG that simultaneously preserves the privacy of entities creating data records and ensures the authentication of these records addresses significant challenges in current SCM solutions.

SCM brings together numerous entities, from suppliers to consumers, requiring the secure flow of items, information, and funds [182]. Ensuring product quality and origin compliance necessitates transparent traceability [167], but protecting private information and trade secrets is equally vital, especially for smaller enterprises [181]. Existing systems often rely on centralized authorities or permissioned blockchains, which are neither fully trustless nor decentralized. Permissionless (public) blockchains create a trustless environment, but their full transparency and high on-chain storage costs pose challenges [186]. Although solutions based on interplanetary file system (IPFS) [38] can address storage issues, they lack privacy-preserving authentication [113, 135].

We are proposing a universal SCM framework that meets various requirements, such as transparency, agility in collaboration, and affordability for micro, small and medium enterprises (MSMEs). To achieve this, we suggest designing an SCM system that uses public blockchains. This system aims to provide cost-efficient solutions by using off-chain storage for product histories while ensuring data integrity and authenticity. This way, all supply chain participants, including end consumers, can verify a product's history. Since products often comprise materials from various sources, we recommend using DAG structured data to manage multi-sourced products, enhancing traceability and efficiency.

The trustless nature of public blockchains underpins our framework, where the authorization of entities is managed via smart contracts. This setup facilitates a fully decentralized structure, allowing for the autonomous execution of business agreements. Consequently, entities, regardless of their prior reputations, can establish trust within this immutable blockchain environment. Moreover, this system streamlines the inception of new collaborations, enabling partners to engage effortlessly and swiftly through mutual smart contract adherence. Nevertheless, the inherent transparency of public blockchains presents a challenge to privacy. To address these challenges, Zupply introduces anonymous authentication token (AAT), AAT ownership transfer (AATOT), and leverages off-chain storage for anonymous, affordable, transparent, and decentralized product histories while guaranteeing data authenticity and preserving the unlinkability of collaborators. Zupply incorporates AAT and AATOT to ensure anonymity and unlinkability among entities. By employing DAG structures, it efficiently manages multi-sourced products and eliminates the need for centralized intermediaries. Zupply is the first framework enabling fully decentralized anonymous authentication for off-chain DAG data, Zupply has extensive potential beyond SCM, including VCS and general-purpose anonymous authentication.

Our Contributions:

1. **Zupply Framework with Enhanced Security Properties**: We utilize zero-knowledge proof (ZKP) to design a novel AAT scheme on smart contract-enabled public blockchains. Zupply includes a set of algorithms and protocols that execute on-chain

unlinkable AATOT and off-chain anonymous data authentication, satisfying four main properties: (1) Anonymity: The anonymity of data uploaders and AAT owners is preserved, except for AATs initiating a DAG. (2) Unlinkability: The link between entities collaborating in maintaining a DAG, which represents supply chain data records, is hidden. (3) Integrity: Each data record is authenticated and unaltered, allowing the auditor to verify that the data record is created solely by the authorized entity for uploading at that stage. (4) Trustlessness: Zupply operates without trusted parties during protocol execution and achieves high decentralization by relying on public blockchains.

2. **Off-chain Authenticated Storage**: Zupply separates storage from blockchain while maintaining a concealed link to the AATs. This allows entities to anonymously upload data. The off-chain anonymous authentication capability decreases blockchain storage costs, enhances anonymity, and allows entities to chose their storage approach based on their specific needs for decentralization and cost. Our framework decreases blockchain storage costs mores by minimizing proof sizes and introducing a novel algorithm for Merkle hash trees [134].

6.2 Zupply Framework

Table 6.1: The main symbols used in the Zupply framework

Symbol	Description
τ	Time
$e_i, \mathbf{E}_\tau$	The i-th entity, the set of all es at τ
$T_i, \mathbf{T}_\tau$	The i-th AAT, the set of all Ts at τ
$\texttt{cm}_i, \mathbf{C}_\tau$	Commitment to the i-th AAT, the set of all $\texttt{cm}$s at τ
$\texttt{eol}_i, \mathbf{X}_\tau$	End-of-life of the i-th AAT, the set of all $\texttt{eol}$s at τ
$\texttt{tx}_i, \mathbf{TX}_\tau$	The i-th transaction, the set of all $\texttt{tx}$s at τ
MHT, $\texttt{rt}_\tau$	Merkle Hash Tree, the MHT's root at τ
$d_n, \mathbf{D}_\tau$	The n-th data record, the set of all ds at τ
$\textsc{cid}_n, \mathbf{CID}_\tau$	CID of the n-th data record, the set of all $\textsc{cid}$s at τ
$\texttt{Tag}_i$	Data ownership transfer tag associated with T_i
$\mathbf{L}_\tau$	The shared ledger at τ

Zupply is a portmanteau of 'Z' from ZKPs [90] and 'Supply' from SCM. Zupply presents a novel AAT scheme, using ZKP algorithms that are efficiently managed on smart contract-compatible public blockchains (e.g., Ethereum [54]), including an anonymous, unlinkable

AATOT. This enables the creation of authenticated records of DAG, thereby maintaining a verifiable history of products in the supply chain. At the same time, the anonymity of each entity and the unlinkability among them are preserved. Namely, any auditor can verify that a data record is authenticated (Definition 6.2) while the data creator remains anonymous. Additionally, no adversary may learn the collaboration between different entities and the supply chain each entity is involved with.

The main symbols used in this chapter are listed in Table 6.1. The Zupply framework comprises three primary components, as illustrated in Figure 6.1. The following descriptions provide an overview of each component:

1. *Entities*: Data uploaders or auditors in the supply chain, ranging from producers to consumers, form the entity set in the Zupply Framework. Entities need to run Zupply Node (Z-Node).

2. *Blockchain Platform (BP)*: Zupply operates on a smart contract-enabled, permissionless blockchain, essential for managing the Zupply smart contract ($\mathcal{C}_Z$) that enables the creation and transfer of AATs.

3. *Decentralized Cloud Storage (DCS)*: The DCS utilized by Zupply is a secure, managerless system based on IPFS [38], storing data records (d) addressed by content identifiers (cid).

In the Zupply's DAG structure, data records are organized in sequences that can either split or merge. Except for the initial record, each one links back to its predecessor's cid. When sequences merge, they incorporate the cids from previous records. Consequently, given any data record d_n, an auditor can trace back and retrieve all preceding records of d_n. This system allows any entity, such as a customer, to access a product's entire history up to that point by obtaining the most recent data record linked to the product, which could be uploaded by the last retailer in the chain. Given that the cid for each data record is derived from its multihash[1] [38], any modification to a record will invalidate the entire data sequence. Every data record comes with a ZKP, showing ownership of an AAT on the blockchain without revealing the owner's identity. Additionally, each record is digitally signed with a secret key, and the corresponding public key is a part of the AAT.

In the initial stage of a supply chain (represented as a source vertex in a DAG), an AAT is created using the Init algorithm, which is executed by the first entity. This token can

[1]Multihash is a self-identifying hash. It is a protocol designed to distinguish between the outputs of various established cryptographic hash functions [39].

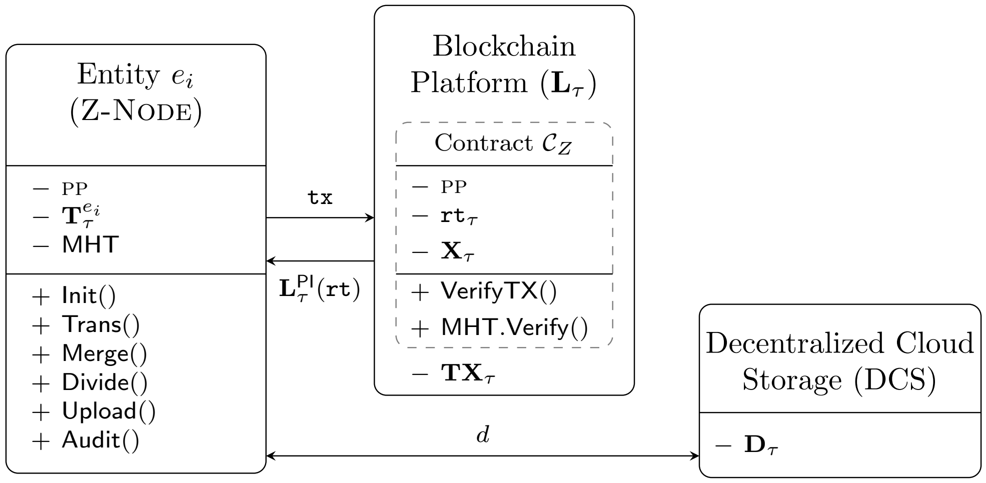


Figure 6.1: Primary components of the Zupply framework. In the figure, τ denotes time; PP represents public parameters; $\mathbf{T}_\tau^{e_i}$ denotes the set of AATs associated with e_i; $\mathtt{rt}_\tau$ is the root of the MHT; $\mathbf{X}_\tau$ is the set of AATs that have reached their end-of-life (preventing re-transfer); $\mathtt{tx}$ denotes a transaction; $\mathbf{L}_\tau^{\mathsf{PI}}(\mathtt{rt})$ is the proof-of-inclusion (PI) of $\mathtt{rt}_\tau$ in the blockchain; d denotes a data record; and $\mathbf{D}_\tau$ is the set of all such data records.

then be passed along the supply chain to subsequent entities through AATOT, including three algorithms: Trans for transferring the token, Merge for combining two tokens before transferring, and Divide for splitting a token into two tokens, which are then transferred. These processes, which employ ZKPs, not only ensure the token's continuous and secure progression through the supply chain but also preserve the unlinkability between entities. Entities within a supply chain can embed essential certificates into their data records to verify the product's originality to auditors. For instance, the initial data record could include a digital certificate of origin for a diamond, ensuring transparency and trust from the outset. However, subsequent entities responsible for transferring or storing the product can provide anonymous reports on conditions such as temperature, among others. These entities receive authorization to upload data from their predecessors, ensuring a secure and transparent chain of custody.

The Zupply Framework ensures that any entity uploading a data record has been authorized by its preceding entity. Therefore, it might be necessary for the first entity in a DAG to upload a certificate as a data record, allowing auditors to verify the source. This requirement remains indispensable in all SCM platforms, as customers check certificates issued by trusted certificate authorities, such as governments at the source or various steps of the supply chain. These certificates may or may not disclose the identity of the entities holding the certificates. In the initial stage of a DAG, the first AAT must be committed by an entity for its own use during the Init algorithm. While this process compromises the AAT's anonymity, the authenticated data record will not reveal any information about the AAT itself. In contrast, the anonymity of owners for subsequently transferred tokens is maintained. This is because each AAT is committed by its preceding entity, ensuring the owner's anonymity.

6.2.1 Cryptographic Building Blocks

Zupply utilizes the following cryptographic building blocks:

1. *Statistically-hiding commitment* scheme (Definition 2.7) allows an entity to commit to their AAT T by sending $\mathtt{cm} := \mathsf{COMM}_\rho(T)$ to the blockchain.
2. *Strongly-unforgeable digital signature* scheme (Definition 2.8) is used for signing data records stored on DCS.
3. *Symmetric-key encryption* scheme (Definition 2.9) is used for encrypting the private part of data records.
4. *Zero-knowledge succinct non-interactive argument of knowledge* scheme (Definition 2.16) is used to ensure the authenticity of data records while preserving the anonymity of entities and unlinkability of AAT commitments.

6.2.2 Threat Model

The adversary is driven by three primary motivations: the forgery or alteration of data records, the deanonymization of data uploaders, and the linking of collaborating entities.

This adversary possesses the capability to access the blockchain, all data records stored on the DCS, and observe all entities and their interactions with $\mathcal{C}_Z$. Zupply operates under the assumption that the adversary lacks possession of the product at each step of its lifecycle within the supply chain. This implies that they never hold an authentication token transferred from an authenticated entity. Therefore, any data reported by authorized entities, who are legitimately involved in the supply chain, is considered accurate.

6.2.3 Security Goals

The Zupply framework is designed as a trustless system for recording product histories, aiming to satisfy the following conditions:

1. The authenticity of data records,
2. The integrity of data records,
3. The anonymity of data record uploaders and owners of transferred tokens,
4. The unlinkability among all entities collaborating within a supply chain.

6.2.4 Terms and Concepts in the Framework

This section introduces the notations and symbols for data structures employed by Zupply. The symbol τ represents time and is also used to denote the state of each data structure. The superscript is for attribution, and the subscript is for indexing. For instance, $T_i^{e_a}$ is the ith AAT which is attributed to or known by entity e_a, while $\texttt{tx}^{e_a}$ signifies that the transaction $\texttt{tx}$ is published by e_a on the blockchain. Furthermore, to emphasize which entity runs a specific algorithm, the algorithm's name is presented as a superscript of the entity. For instance, if entity e_a executes the algorithm Init, it is denoted as e_a^{Init}.

Entities

The set of all entities in Zupply is $\mathbf{E}_\tau = \{e_1, \ldots, e_{|\mathbf{E}_\tau|}\}$. Each entity e_i, possesses a blockchain addresses denoted as BPAddr^{e_i}. This scheme operates without a central manager, placing all entities at an equal level of authority.

Anonymous Authentication Token (AAT)

$\mathbf{T}_\tau = \{T_1, \ldots, T_{|\mathbf{T}_\tau|}\}$ denotes the time-ordered set of AATs, where each is represented as $T_i := (\tilde{T}_i, \mathrm{SKsig}_i)$. Here $\tilde{T}_i := (q_i, \mathrm{PKsig}_i, \rho_i)$ denotes a partial AAT. Let $q_i \in \{0,1\}^{N_q}$ keep track of the quantity of products. It should be kept unchanged unless two supply chains are merged or one supply chain is divided into two sub supply chains. SKsig_i corresponds to PKsig_i. The owner of the AAT T_i uses SKsig_i to sign the data records uploaded by them. It is important not to confuse PKsig_i, which represents the public key associated with the

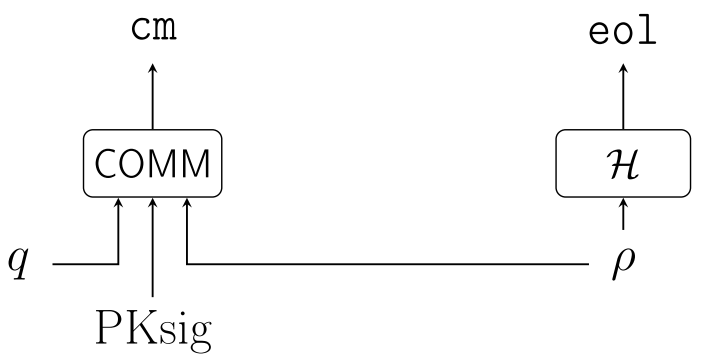


Figure 6.2: The process of committing to an AAT and its corresponding EoL.

token T_i, with the public keys related to the entity's BPAddr. Moreover, $\rho_i \in_R \{0,1\}^{N_\rho}$ plays a crucial role in preventing the token's owner from re-transferring the AAT after transferring its ownership to the new owner(s). This is achieved by publishing the end-of-life (EoL) of the AAT, denoted as $\texttt{eol}_i = \mathcal{H}(\rho_i)$, on the blockchain.

End of Life (EoL)

In an AAT, $\rho_i \in_R \{0,1\}^{N_\rho}$ prevents the owner of the AAT from re-transferring it after it has already been transferred by publishing the end-of-life (EoL), denoted as $\texttt{eol}_i = \mathcal{H}(\rho_i)$, on the blockchain. $\mathbf{X}_\tau = \{\texttt{eol}_1, \ldots, \texttt{eol}_{|\mathbf{X}_\tau|}\}$ is the time-ordered set of published EoL of the obsoleted tokens.

Commitment to AAT

$\mathbf{C}_\tau = \{\texttt{cm}_1, \ldots, \texttt{cm}_{|\mathbf{C}_\tau|}\}$ is the time-ordered set of commitments to $\mathbf{T}_\tau$. So, the corresponding commitment to T_i is denoted as $\texttt{cm}_i := \mathsf{COMM}_{\rho_i}(\tilde{T}_i)$. Figure 6.2 illustrates the diagram of committing to an AAT.

Merkle Hash Tree

Commitments to AATs are stored on an L-layer Merkle hash tree ($|\mathbf{T}_\tau| < 2^{L-1}$) denoted as MHT whose root is $\texttt{rt}_\tau$. Each leaf of MHT has an index number $\mathsf{ind} \in [2^{L-1}]$ and $\mathsf{path}_{\mathsf{ind}}$ represents the Merkle proof corresponding to ind. MHT is a data structure that keeps the whole Merkle hash tree nodes and encompasses three algorithms:

1. $\mathsf{MHT.Add}(\mathtt{cm}) \rightarrow (\mathtt{rt}^{\text{new}}, \mathsf{ind}, \mathsf{path}_{\mathsf{ind}})$. Given a new $\mathtt{cm}$, the algorithm adds $\mathtt{cm}$ to the MHT and outputs the new Merkle root $\mathtt{rt}^{\text{new}}$, the index of the new commitment (ind), and $\mathsf{path}_{\mathsf{ind}}$.

2. $\mathsf{MHT.Verify}(\mathtt{rt}_{\tau-1}, \mathtt{rt}_{\tau}, \mathsf{cm}, \mathsf{ind}, \mathsf{path}_{\mathsf{ind}}) \rightarrow \{0,1\}$. Given the previous and the new root of MHT before and after assigning $\mathtt{cm}$ to the tree at ind, with corresponding $\mathsf{path}_{\mathsf{ind}}$, this algorithms verifies whether $\mathtt{rt}_{\tau}$ is computed correctly.

3. $\mathsf{MHT.Search}(\mathtt{cm}_i) \rightarrow (\mathsf{ind}_i, \mathsf{path}_{\mathsf{ind}_i})$. Given $\mathtt{cm}_i$, $\mathsf{MHT.Search}$ returns the index of $\mathtt{cm}_i$ (ind_i), and the corresponding $\mathsf{path}_{\mathsf{ind}_i}$.

$\mathsf{MHT.Add}()$ and $\mathsf{MHT.Search}()$ are stateful, having direct access to the MHT data structure. In contrast, $\mathsf{MHT.Verify}()$ does not. This algorithm is implemented within the smart contract, which does not store the MHT.

Blockchain Platform Ledger

We can model the blockchain as an immutable, public, decentralized ledger, denoted as $\mathbf{L}_{\tau}$, on which the Zupply smart contract, $\mathcal{C}_Z$, is deployed. $\mathcal{C}_Z$ maintains $\mathtt{rt}_{\tau}$, and $\mathbf{X}_{\tau}$. The commitments in $\mathbf{C}_{\tau}$ are uploaded to the blockchain via transactions that invoke $\mathcal{C}_Z$. Consequently, $\mathbf{L}_{\tau}$ records all transactions associated with the Zupply framework (discussed below), along with other unrelated transactions on the blockchain platform.

Wherever data records are concerned, the protocol employs the *proof-of-inclusion (PI)* mechanism provided by the blockchain to prove the correctness of the value $\mathtt{rt}_{\tau}$ used in off-chain proofs. The PI of $\mathtt{rt}_{\tau}$ in ledger $\mathbf{L}_{\tau}$ at time τ is represented as $\mathbf{L}_{\tau}^{\mathsf{PI}}(\mathtt{rt})$. This proof ensures that the auditors can verify $\mathtt{rt}$ employed for off-chain authentication.

Transactions

Each transaction $\mathtt{tx} \in \{\mathtt{tx}_{\mathsf{Init}}, \mathtt{tx}_{\mathsf{Trans}}, \mathtt{tx}_{\mathsf{Merge}}, \mathtt{tx}_{\mathsf{Div}}\}$ invokes a specific function in $\mathcal{C}_Z$. Each $\mathtt{tx}$ contains one or two $\mathtt{cm}$s, and whenever it finalizes on the blockchain, its corresponding $\mathtt{cm}$s are added to MHT locally and updates $\mathtt{rt}_{\tau}$ on the blockchain. For a transactions that transfers the ownership of AAT(s), $\mathtt{tx} \in \{\mathtt{tx}_{\mathsf{Trans}}, \mathtt{tx}_{\mathsf{Merge}}, \mathtt{tx}_{\mathsf{Div}}\}$, the transaction contains the EoL of expired AAT(s) whose commitment(s) are stored on MHT, and a ZKP of owning them. These proofs are denoted as $\pi_{\mathtt{x}} \in \{\pi_{\mathsf{Trans}}, \pi_{\mathsf{Merge}}, \pi_{\mathsf{Div}}\}$. The time-ordered set of transactions on blockchain is denoted as $\mathbf{TX}_{\tau} = \{\mathtt{tx}_1, \ldots, \mathtt{tx}_{|\mathbf{TX}_{\tau}|}\}$.

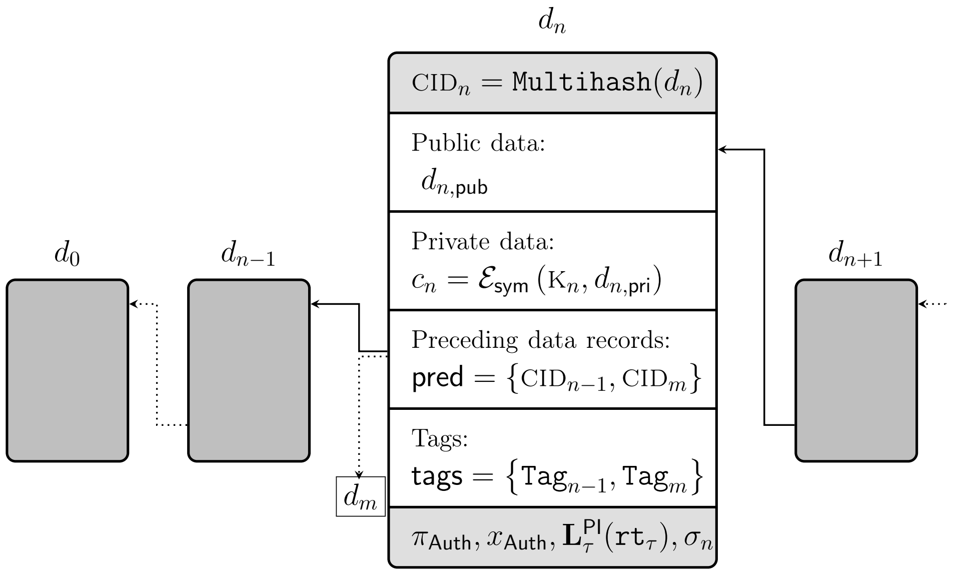

d_n
$\mathrm{CID}_n = \texttt{Multihash}(d_n)$
Public data:
$d_{n,\mathsf{pub}}$
Private data:
$c_n = \mathcal{E}_{\mathsf{sym}}(\mathrm{K}_n, d_{n,\mathsf{pri}})$
Preceding data records:
$\mathsf{pred} = \{\mathrm{CID}_{n-1}, \mathrm{CID}_m\}$
Tags:
$\mathsf{tags} = \{\texttt{Tag}_{n-1}, \texttt{Tag}_m\}$
$\pi_{\mathsf{Auth}}, x_{\mathsf{Auth}}, \mathbf{L}^{\mathsf{PI}}_\tau(\texttt{rt}_\tau), \sigma_n$
d_0
d_{n-1}
d_{n+1}
d_m

Figure 6.3: Data records create a DAG structured data. d_n represents the data record after two data sequences merges in the DAG.

Data Records

The time-ordered set of data records is denoted as $\mathbf{D}_\tau = \{d_1, \ldots, d_{|\mathbf{D}_\tau|}\}$. Data records are stored on a DCS. The CID of a data record d_n in the DCS is denoted as $\textsc{cid}_n$. The set of all CIDs is $\mathbf{CID}_\tau = \{\textsc{cid}_1, \ldots, \textsc{cid}_{|\mathbf{CID}_\tau|}\}$. $\mathsf{DCS}(\textsc{cid}_n) \rightarrow d_n$ denotes a query to the DCS to get the data record corresponded $\textsc{cid}$. Each data record d_n contains references to its predecessor, d_p and d_q, where $p, q < n$, in the array $d_n.\mathsf{pred}$ if applicable. Specifically, $d_n.\mathsf{pred}[b] \in \mathbf{CID}_\tau$ for $b = 0, 1$; although one or both entries of the array may be $\emptyset$. A data record d_n contains public data $d_{n,\mathsf{pub}}$ which is stored as plain-text, and private data $d_{n,\mathsf{pri}}$ which is stored as cipher-text $c_n = \mathcal{E}_{\mathsf{sym}}(\mathrm{K}_n, d_{n,\mathsf{pri}})$. Zupply may incorporate the forward secrecy scheme outlined in Mesh [8] to determine K_n (Appendix 6.5.3). d_n also includes a ZKP π_{Auth} of owning T_i such that its corresponding $\texttt{cm}_i$ is in MHT. The public input to this proof is denoted as $x_{\texttt{Auth}}$. Each T_i, can be used more than once to prove authenticity. Also, the data record d_n includes $\sigma_n = \mathcal{S}_{\mathsf{sig}}(\mathrm{SKsig}_i, d_n)$, where SKsig_i is the secret key associated with PKsig_i in T_i. Consequently, d_n is denoted as the following tuple and illustrated in Figure 6.3:

$$\begin{aligned} d_n := \big(& d_{n,\mathsf{pub}}, c_n, \mathsf{pred} = \{\mathrm{CID}_p, \mathrm{CID}_q\}, \mathsf{tags} = \{\texttt{Tag}_p, \texttt{Tag}_q\}, \\ & \pi_{\mathsf{Auth}}, x_{\mathsf{Auth}} = \{\texttt{rt}_\tau, \mathrm{PKsig}_i\}, \mathbf{L}^{\mathsf{PI}}_\tau(\texttt{rt}_\tau)\big)\,, \end{aligned}$$

where the digital signature σ_n is uploaded together with the data d_n and the data ownership transfer tags (tags) are explained in the following.

Data Ownership Transfer Tag

When two consecutive data records d_{n-1}, d_n in a data sequence uses different secret keys for signing the data records. Namely, $\sigma_{n-1} = \mathcal{S}_{\mathsf{sig}}(\text{SKsig}_i, d_{n-1})$ and $\sigma_n = \mathcal{S}_{\mathsf{sig}}(\text{SKsig}_j, d_n)$. In this scenario, the later data record (d_n) has to include the $\texttt{Tag}_i = \mathcal{S}_{\mathsf{sig}}(\text{SKsig}_i, \text{PKsig}_j)$ created by the owner of the former data record (d_{n-1}). Data ownership transfer tags are created and used in AATOT presented in Section 6.3.2.

Directed Acyclic Graph

Given that each data record can reference its predecessor, they collectively form a DAG. Consequently, we define three key concepts integral to our DAG data structure:

1. *Init data* ($d_{n,\mathsf{Init}}$): Any data record d_n which does not have any predecessor, i.e., $d_n.\mathsf{pred}[b] = \emptyset$ for both $b \in \{0, 1\}$.
2. *Data Sequence* (S_k): S_k is a sequence of data records where each data record has one and only one predecessor. The first data record in S_k can have zero, one, or two predecessors.
3. *Progressive Data Sequence* ($\hat{S}_k$): is a data sequence denoted as $\hat{S}_k = \{d_{i_1}, \ldots, d_{i_m}\}$, where data record d_{i_m} is the most recent, and it's ensured that no subsequent data record references d_{i_m}.

Remark 6.1 (AAT Public Key). In the π_{Auth}, PKsig is used as a public input. π_{Auth} is then published off-chain on data records. However, for proofs directly published on the blockchain (π_{Trans}, π_{Merge}, and π_{Div}), PKsig is treated as a private input. Also, the PKsig is never published on blockchain.

Definition 6.1 (Valid AAT)**.** T_{i_2} is a valid AAT for authenticating a data record d_{n_2} if and only if the $\texttt{cm}_{i_2} := \mathsf{COMM}_{\rho_{i_2}}(T_{i_2})$ is included in MHT.

Definition 6.2 (Authenticated Data)**.** A data record d_n is an authenticated data if and only if d_n contains

1. a valid zero-knowledge proof of ($\pi_{\texttt{Auth}}$) of owning a *valid AAT* T_i (Definition 6.1).

2. a valid signature σ is created using the secret key SKsig_i, ensuring that both it and its corresponding public key PKsig_i are included in T_i.

3. valid tag(s) if and only if the predecessor data record is signed by different key(s) and $d_n.\text{pred} \neq \emptyset$

Definition 6.3 (Valid Transaction)**.** A transaction $\mathtt{tx} \in \{\mathtt{tx}_{\mathsf{Init}}, \mathtt{tx}_{\mathsf{Trans}}, \mathtt{tx}_{\mathsf{Merge}}, \mathtt{tx}_{\mathsf{Div}}\}$ is a valid transaction if and only if

1. $\mathsf{MHT.Verify}(\mathtt{rt}_\tau, \mathtt{rt}^{\text{new}}, \mathtt{cm}, \mathsf{ind}, \mathsf{path}_{\mathsf{ind}}) = 1$, where The root of MHT was $\mathtt{rt}_\tau$ before $\mathtt{tx}$. For Div transaction (i.e., $\mathtt{tx} = \mathtt{tx}_{\mathsf{Div}}$) the transition of the root is examined in two rounds of running $\mathsf{MHT.Verify}$ algorithm: $\mathtt{rt}_\tau \rightarrow \mathtt{rt}_1^{\text{new}} \rightarrow \mathtt{rt}_2^{\text{new}}$.

2. $\mathtt{eol} \notin \mathbf{X}_\tau$. Where $\mathtt{tx}_{\mathsf{Trans}}$ and $\mathtt{tx}_{\mathsf{Div}}$ contain one and $\mathtt{tx}_{\mathsf{Merge}}$ contains two $\mathtt{eol}$s. This value represents the AAT that is expired during the owner transferring algorithms.

3. $\mathsf{Verify}(\text{vk}_{\mathbb{x}}, x_{\mathbb{x}}, \pi_{\mathbb{x}}) = 1$ for $\mathbb{x} \in \{\mathsf{Trans}, \mathsf{Merge}, \mathsf{Div}\}$

6.2.5 NP Statements

This section presents the NP-statements related to ZKPs employed in Zupply.

Authenticity proof (π_{Auth})

Whenever an entity wants to upload a data record on DCS, the entity (prover) proves they own a commitment $\mathtt{cm}$ in the MHT. The NP problems associated with this proof are explained as follows:

- $\mathtt{cm}_i = \mathsf{COMM}_{\rho_i}(\tilde{T}_i)$
- $\mathtt{path}_i$ is a valid path from leaf $\mathtt{cm}_i$ to root $\mathtt{rt}_\tau$.

Where $\tilde{T}_i = (q_i, \text{PKsig}_i, \rho_i)$. To prove that the used $\mathtt{rt}$ is a valid value stored on $\mathcal{C}_Z$, the entity provides the PI $\mathbf{L}_\tau^{\mathsf{PI}}(\mathtt{rt})$.

Transferability proof (π_{Trans})

When an entity wishes to transfer the ownership of an AAT (T_{i_1}), it (as the prover) proves that the newly minted $\tilde{T}_{i_2}$ maintains the same product quantity as T_{i_1} (i.e., $q_{i_1} = q_{i_2}$), without disclosing details about T_{i_1} and $\tilde{T}_{i_2}$. Moreover, $\texttt{cm}_{i_1}$ is kept confidential. In addition, the entity proves that the published $\texttt{eol}_{i_1}$ is equal to $\mathcal{H}(\rho_{i_1})$, while keeping ρ_{i_1} secret. The NP problems associated with this proof are explained as follows:

- $\texttt{eol}_{i_1} = \mathcal{H}(\rho_{i_1})$
- $\texttt{cm}_{i_1} = \mathsf{COMM}_{\rho_{i_1}}(\tilde{T}_{i_1})$
- $\texttt{path}_{i_1}$ is a valid path from leaf $\texttt{cm}_{\texttt{cm}_{i_1}}$ to root $\texttt{rt}_\tau$
- $\texttt{cm}_{i_2} = \mathsf{COMM}_{\rho_{i_2}}(\tilde{T}_{i_2})$
- $q_{i_1} = q_{i_2}$

Where $\tilde{T}_{i_1} = (q_{i_1}, \text{PKsig}_{i_1}, \rho_{i_1})$, $\tilde{T}_{i_2} = (q_{i_2}, \text{PKsig}_{i_2}, \rho_{i_2})$, and $x_{\mathsf{Trans}} = (\texttt{eol}_{i_1}, \texttt{cm}_{i_2}, \texttt{rt}_\tau)$ denotes the public inputs.

Mergeability proof (π_{Merge})

An entity may merge two AATs T_{i_1} and T_{i_2} to create T_{i_3}. The entity proves that the quantity in T_{i_3} is the same as the summation of quantities in T_{i_1} and T_{i_2} (i.e., $q_{i_1} + q_{i_2} = q_{i_3}$). The ZKP π_{Merge} follows the same approach as π_{Trans}. The NP problems associated with this proof are explained as follows:

- $\texttt{eol}_{i_1} = \mathcal{H}(\rho_{i_1})$ and $\texttt{eol}_{i_2} = \mathcal{H}(\rho_{i_2})$
- $\texttt{cm}_{i_1} = \mathsf{COMM}_{\rho_{i_1}}(\tilde{T}_{i_1})$, $\texttt{cm}_{i_2} = \mathsf{COMM}_{\rho_{i_2}}(\tilde{T}_{i_2})$
- $\texttt{path}_{i_1}$ and $\texttt{path}_{i_2}$ are valid paths respectively from leaf $\texttt{cm}_{\texttt{cm}_{i_1}}$ and $\texttt{cm}_{\texttt{cm}_{i_2}}$ to root $\texttt{rt}_\tau$.
- $\texttt{cm}_{i_3} = \mathsf{COMM}_{\rho_{i_3}}(\tilde{T}_{i_3})$
- $q_{i_1} + q_{i_2} = q_{i_3}$

Where $\tilde{T}_{i_1} = (q_{i_1}, \text{PKsig}_{i_1}, \rho_{i_1})$, $\tilde{T}_{i_2} = (q_{i_2}, \text{PKsig}_{i_2}, \rho_{i_2})$ and $\tilde{T}_{i_3} = (q_{i_3}, \text{PKsig}_{i_3}, \rho_{i_3})$. The public input to the proof is denoted as $x_{\mathsf{Merge}} = (\texttt{eol}_{i_{1,2}}, \texttt{cm}_{i_3}, \texttt{rt}_\tau)$.

Divisibility proof (π_{Div})

An entity can divide an AAT T_{i_1} into two new T_{i_2} and T_{i_3}. The approach of creating ZKP π_{Div} follows the same approaches in π_{Trans} and π_{Merge}. The NP problems associated with this proof are explained as follows:

- $\mathtt{eol}_{i_1} = \mathcal{H}(\rho_{i_1})$
- $\mathtt{cm}_{i_1} = \mathsf{COMM}_{\rho_{i_1}}(\tilde{T}_{i_1})$
- $\mathtt{path}_{i_1}$ is a valid path from leaf $\mathtt{cm}_{\mathtt{cm}_{i_1}}$ to root $\mathtt{rt}_{\tau}$
- $\mathtt{cm}_{i_2} = \mathsf{COMM}_{\rho_{i_2}}(\tilde{T}_{i_2})$ and $\mathtt{cm}_{i_3} = \mathsf{COMM}_{\rho_{i_3}}(\tilde{T}_{i_3})$
- $q_{i_1} = q_{i_2} + q_{i_3}$

Where $\tilde{T}_{i_1} = (q_{i_1}, \text{PKsig}_{i_1}, \rho_{i_1})$, $\tilde{T}_{i_2} = (q_{i_2}, \text{PKsig}_{i_2}, \rho_{i_2})$ and $\tilde{T}_{i_3} = (q_{i_3}, \text{PKsig}_{i_3}, \rho_{i_3})$. The public input is denoted as $x_{\mathsf{Div}} = \big(\mathtt{eol}_{i_1}, \mathtt{cm}_{i_{2,3}}, \mathtt{rt}_{\tau}\big)$.

6.3 Zupply Algorithms and Protocols

The Zupply framework comprises a set of algorithms and protocols detailed in Sections 6.3.1 and 6.3.2, respectively.

Definition 6.4 (Zupply Framework)**.** Zupply framework Π Consists of a tuple of polynomial-time algorithms and protocols $\Pi =$ (Setup, Init, Trans, Merge, Divide, Upload, VerifyTX, Audit; OT-Protocol, MHT-Protocol).

6.3.1 Algorithms

All algorithms have read-only access to $\mathbf{TX}\mathcal{T}$, $\mathbf{C}\mathcal{T}$, MHT, $\mathbf{X}\mathcal{T}$, and $\mathbf{D}\mathcal{T}$ (accessible via DCS queries). The smart contract is responsible for updating $\mathbf{TX}\mathcal{T}$, $\mathbf{C}\mathcal{T}$, $\mathbf{X}_{\tau}$, and the root of MHT when the VerifyTX algorithm outputs 1.

Setup $(1^\lambda) \rightarrow$ PP

Given a security parameter λ, Setup generates a set of global public parameters, denoted as PP. The specific implementation of Setup depends on the zkSNARK algorithm in use: it may require a trusted party, or it may rely on a transparent (public) setup. The resulting PP is made publicly available. The pseudocode for Setup is provided in Algorithm 6.1.

Algorithm 6.1 Setup $(1^\lambda) \rightarrow$ PP

```
$(\mathrm{pk}_{\texttt{Auth}}, \mathrm{vk}_{\texttt{Auth}}) \leftarrow \mathsf{KeyGen}(1^\lambda, \mathsf{Auth})$
$(\mathrm{pk}_{\mathsf{Trans}}, \mathrm{vk}_{\mathsf{Trans}}) \leftarrow \mathsf{KeyGen}(1^\lambda, \mathsf{Trans})$
$(\mathrm{pk}_{\mathsf{Merge}}, \mathrm{vk}_{\mathsf{Merge}}) \leftarrow \mathsf{KeyGen}(1^\lambda, \mathsf{Merge})$
$(\mathrm{pk}_{\mathsf{Div}}, \mathrm{vk}_{\mathsf{Div}}) \leftarrow \mathsf{KeyGen}(1^\lambda, \mathsf{Divide})$
$\mathrm{PP}_{\mathsf{sig}} \leftarrow \mathcal{G}_{\mathsf{sig}}(1^\lambda)$
$\mathrm{PP} \leftarrow (\mathrm{pk}_{\texttt{Auth}}, \mathrm{vk}_{\texttt{Auth}}, \mathrm{pk}_{\mathsf{Trans}}, \mathrm{vk}_{\mathsf{Trans}}, \mathrm{pk}_{\mathsf{Merge}}, \mathrm{vk}_{\mathsf{Merge}}, \mathrm{pk}_{\texttt{Div}}, \mathrm{vk}_{\texttt{Div}}, \mathrm{PP}_{\mathsf{sig}})$
return PP
```

Init $(\mathrm{PP}, q_{i_1}) \rightarrow (\mathtt{tx}_{\mathsf{Init}}, T_{i_1})$

Initiates a DAG. The algorithm gets q_{i_1}, which will be maintained in the DAG. It creates a new token T_{i_1}, commits to it, and sends the commitment via

$$\mathtt{tx}_{\mathsf{Init}} := (\mathtt{cm}_{i_1}, \mathtt{rt}^{\mathrm{new}}, \mathsf{ind}_{i_1}, \mathsf{path}_{\mathsf{ind}_{i_1}})$$

to the blockchain. The pseudocode for Init is provided in Algorithm C.1 is provided in Algorithm C.2 in Appendix C.

Trans $(\mathrm{PP}, T_{i_1}, \mathbf{PKsig}_{i_2}) \rightarrow (\mathtt{tx}_{\mathsf{Trans}}, \tilde{T}_{i_2}, \mathtt{Tag})$

This algorithm takes an existing AAT T_{i_1} and renders it obsolete by publishing its associated EoL on the blockchain. Specifically, it computes $\mathtt{eol}_{i_1} := \mathcal{H}(\rho_{i_1})$, where ρ_{i_1} is a secret contained in T_{i_1}, and includes this value in the resulting transaction $\mathtt{tx}_{\mathsf{Trans}}$. Subsequently, the algorithm constructs a new partial AAT, denoted $\tilde{T}_{i_2}$, by sampling a fresh random value ρ_{i_2} and assigning it the same quantity as in T_{i_1}. The public key PKsig_{i_2} is used to bind the new token to its intended recipient, in accordance with the OT-Protocol described in Section 6.3.2. The algorithm commits to $\tilde{T}_{i_2}$ and includes the resulting commitment in the transaction. The transaction also includes a ZKP demonstrating: (1) knowledge of T_{i_1}

whose commitment is included in the MHT, without revealing the commitment itself, (2) consistency of this committment with the published $\texttt{eol}_{i_1}$, (3) knowledge of the preimage of the new commitment ($\texttt{cm}_{i_2}$) to $\tilde{T}_{i_2}$, and (4) equality of the quantity values in T_{i_1} and $\tilde{T}_{i_2}$. Accordingly, the contents of the transaction $\texttt{tx}_{\mathsf{Trans}}$ are defined as:

$$\texttt{tx}_{\mathsf{Trans}} := \left(\pi_{\mathsf{Trans}}, x_{\mathsf{Trans}}, \texttt{eol}_{i_1}, \texttt{cm}_{i_2}, \texttt{rt}^{\text{new}}, \mathsf{ind}_{i_2}, \mathsf{path}_{\mathsf{ind}_{i_2}}\right)$$

where

$$x_{\mathsf{Trans}} := \left(\texttt{eol}_{i_1}, \texttt{cm}_{i_2}, \texttt{rt}_{\tau}\right)$$

denotes the public inputs to the ZKP, $\texttt{rt}^{\text{new}}$ is the updated root of the MHT after inserting the new commitment $\texttt{cm}_{i_2}$, and ind_{i_2} with $\mathsf{path}_{\mathsf{ind}i_2}$ represent the index and Merkle proof of the new commitment. Finally, the algorithm outputs a $\texttt{Tag}$, which enables the new AAT owner to continue the data sequence, ensuring continuity with the most recent data record generated using T_{i_1}. The pseudocode for Trans is provided in Algorithm C.2 in Appendix C.

Merge ($\textbf{PP}$, T_{i_1}, T_{i_2}, $\textbf{PKsig}_{i_3}$) $\rightarrow$ ($\texttt{tx}_{\mathsf{Merge}}$, $\tilde{T}_{i_3}$, $\texttt{Tag}_{i_1}$, $\texttt{Tag}_{i_2}$)

The Merge algorithm follows a similar approach to the Trans algorithm. In this procedure, two AATs, T_{i_1} and T_{i_2}, are rendered obsolete, and their corresponding EoLs, $\texttt{eol}_{i_1}$ and $\texttt{eol}_{i_2}$, are published to the blockchain via $\texttt{tx}_{\mathsf{Merge}}$. A new partial token, $\tilde{T}_{i_3}$, is created using the input public key PKsig_{i_3}, sampling a random value ρ_{i_2}, and a quantity value set to the sum of the quantity values in T_{i_1} and T_{i_2}. The transaction also includes a commitment to the new AAT, denoted by $\texttt{cm}_{i_3}$, along with a ZKP that proves the following statements without revealing the underlying values: (1) knowledge of T_{i_1} and T_{i_2} whose commitments are included in the MHT, (2) consistency of these commitments with the published $\texttt{eol}_{i_1}$ and $\texttt{eol}_{i_2}$, (3) the knowledge of $\tilde{T}i_3$ in the new commitment $\texttt{cm}_{i_3}$ to $\tilde{T}_{i_3}$, and (4) that the quantity in $\tilde{T}_{i_3}$ equals the sum of the quantities in T_{i_1} and T_{i_2}. Accordingly, the transaction $\texttt{tx}_{\mathsf{Merge}}$ is defined as:

$$\texttt{tx}_{\mathsf{Merge}} := (\pi_{\mathsf{Merge}}, x_{\mathsf{Merge}}, \texttt{eol}_{i_1}, \texttt{eol}_{i_2}, \texttt{cm}_{i_3}, \texttt{rt}^{\text{new}}, \mathsf{ind}_{i_3}, \mathsf{path}_{\mathsf{ind}_{i_3}}),$$

where

$$x_{\mathsf{Merge}} = (\texttt{eol}_{i_1}, \texttt{eol}_{i_2}\texttt{cm}_{i_3}, \texttt{rt}_{\tau})$$

denotes the public inputs to the ZKP. Here, $\texttt{rt}_1^{\text{new}}$ is the updated root of the MHT after inserting the first new commitment $\texttt{cm}_{i_2}$, and $\texttt{rt}_2^{\text{new}}$ is the root after inserting the second new commitment $\texttt{cm}_{i_3}$. Accordingly, ind_{i_2} with its corresponding Merkle path $\mathsf{path}_{\mathsf{ind}_{i_2}}$, and ind_{i_3} with $\mathsf{path}_{\mathsf{ind}_{i_3}}$, represent the indices and Merkle proofs for the first and second new commitments, respectively. The Merge algorithm also outputs two data ownership transfer tags, $\texttt{Tag}_{i_1}$ and $\texttt{Tag}_{i_2}$, which are used together in a new data record to ensure continuity with the two most recent data records generated by the AATs T_{i_1} and T_{i_2} and merging those sequences. The pseudocode for the Merge algorithm is provided in Algorithm C.3 in Appendix C.

Divide ($\mathbf{PP}$, T_{i_1}, $\mathbf{PKsig}_{i_2}$, $\mathbf{PKsig}_{i_3}$, q_{i_2}, q_{i_3}) $\to$ ($\texttt{tx}_{\mathsf{Div}}$, $\tilde{T}_{i_2}$, $\tilde{T}_{i_3}$, $\texttt{Tag}_{i_2}$, $\texttt{Tag}_{i_3}$)

The Divide algorithm is the inverse of the Merge algorithm. It takes a single AAT, T_{i_1}, as input and creates two new partial AATs, denoted $\tilde{T}_{i_2}$ and $\tilde{T}_{i_3}$. These are generated using the provided public keys PKsig_{i_2} and PKsig_{i_3}, sampled random values ρ_{i_2} and ρ_{i_3}, and quantity values q_{i_2} and q_{i_3}, such that the sum $q_{i_2} + q_{i_3}$ equals the quantity in T_{i_1}. This algorithm renders T_{i_1} obsolete, and its associated EoL, denoted $\texttt{eol}_{i_1}$, is included in the transaction sent to the blockchain. The transaction also includes the commitments to the newly created partial tokens, $\texttt{cm}_{i_2}$ and $\texttt{cm}_{i_3}$, as well as a ZKP proving the following statements: (1) knowledge of T_{i_1}, whose commitment is included in the MHT, (2) consistency between the commitment to T_{i_1} and the published $\texttt{eol}_{i_1}$, (3) knowledge of $\tilde{T}_{i_2}$ and $\tilde{T}_{i_3}$ which are the preimages of the commitments $\texttt{cm}i_2$ and $\texttt{cm}i_3$, and (4) that the quantity in T_{i_1} equals the sum of the quantities in $\tilde{T}_{i_2}$ and $\tilde{T}_{i_3}$. Accordingly, the transaction $\texttt{tx}_{\mathsf{Div}}$ is defined as:

$$\texttt{tx}_{\mathsf{Div}} := (\pi_{\mathsf{Div}}, x_{\mathsf{Div}}, \texttt{eol}_{i_1}, \texttt{cm}_{i_2}, \texttt{cm}_{i_3}, \texttt{rt}_1^{\text{new}}, , \texttt{rt}_2^{\text{new}}, \mathsf{ind}_{i_2}, \mathsf{ind}_{i_3}, \mathsf{path}_{\mathsf{ind}_{i_2}}, \mathsf{path}_{\mathsf{ind}_{i_3}})$$

where

$$x_{\mathsf{Div}} = \left(\texttt{eol}_{i_1}, \texttt{cm}_{i_2}, \texttt{cm}_{i_3}, \texttt{rt}_{\tau}\right)$$

denotes the public inputs to the ZKP. The algorithm also outputs two data ownership transfer tags, $\texttt{Tag}_{i_2}$ and $\texttt{Tag}_{i_3}$, which are used in two separate data records whose common predecessor is the most recent data record generated by T_{i_1}. Each resulting data record initiates a new data sequence. The pseudocode for the Divide algorithm is provided in Algorithm C.4 in Appendix C.

VerifyTX ($\mathbf{PP}$, $\texttt{tx}_{\mathtt{x}}$) $\to b \in \{0, 1\}$

The VerifyTX algorithm is executed within the Zupply smart contract $\mathcal{C}_Z$. It takes as input a transaction $\texttt{tx}$ and returns one if and only if $\texttt{tx}$ is valid, as defined in Definition 6.3. Below, we present the algorithm in more detail.

The verification steps depend on the type of transaction, denoted by $\mathtt{x}$, which can be one of Trans, Merge, or Div. Regardless of the type, the algorithm ensures that the EoL in the transaction is not included in the set $\mathbf{X}_\tau$. It also calls the MHT.Verify algorithm (described in Algorithm 6.3) to check whether the proposed updated Merkle root $\mathtt{rt}^{\mathsf{new}}$ has been computed correctly, following the MHT-Protocol, which is introduced later. Note that the smart contract does not store the MHT itself. Furthermore, the ZKPs corresponding to each transaction type, π_{Trans}, π_{Merge}, and π_{Div}, are verified using their respective verification keys, namely $\mathsf{vk}_{\mathsf{Trans}}$, $\mathsf{vk}_{\mathsf{Merge}}$, and $\mathsf{vk}_{\mathsf{Div}}$, along with the public inputs embedded in the transaction.

In addition to the shared verification logic, tx of type Merge includes one extra EoL element that must also be checked for exclusion from $\mathbf{X}_\tau$. On the other hand, a Div transaction includes two new commitments, requiring an additional verification step. After verifying the first new Merkle root as described, the algorithm verifies the second root against the second commitment.

If any of these verifications fail, the algorithm returns zero. Otherwise, it returns one. The pseudocode for VerifyTX is provided in Algorithm C.4 in Appendix C.

Upload $(\textsc{pp}, [\mathsf{pred}], T_i, d_{\mathsf{pub}}, d_{\mathsf{pri}}, \mathbf{K}_{n_3}, [\mathsf{tags}]) \rightarrow d_{n_3}$

The Upload algorithm processes components of d_{n_3} along with the CID of the predecessor data records and the corresponding data ownership transfer tags, if applicable.

The algorithm commits to the AAT T_i provided as input and searches for the corresponding commitment in the MHT. Once the commitment is located, it generates a ZKP, denoted as π_{Auth}, that proves knowledge of a pre-image to a commitment in the MHT. Unlike the ZKPs generated by other algorithms, this proof includes PKsig_i, which is embedded in T_i, as a public input to the proof. This inclusion is essential because, once all components of the new data record d_{n_3} are generated, the record is signed using the SKsig_i found in T_i. The presence of PKsig_i as a public input allows the verifier to validate the signature and ensures that the same T_i (which includes PKsig_i) was used throughout the sequence. If a different AAT were substituted in the sequence, the Audit algorithm, described later, would reject the data record, as it detects a mismatch in the PKsig relative to previous records. An exception to this rule occurs only when a data ownership transfer tag is included in the first data record that uses a different AAT. Such a tag is generated by one of the Trans, Merge, or Divide algorithms.

The public input to the aforementioned ZKP includes the Merkle root at the time the proof is generated, denoted as $\mathtt{rt}_\tau$. Since auditors are lightweight clients and do not store the blockchain, they cannot directly verify whether the claimed $\mathtt{rt}_\tau$ is indeed the Merkle root of the MHT. To address this, the prover includes $\mathbf{L}^{\mathsf{PI}}_\tau(\mathtt{rt}_\tau)$, a proof that attests to the correctness of $\texttt{rt}_\tau$ on the blockchain. This proof is constructed in a manner that allows verification by lightweight clients.

Moreover, given the secret key κ_{n_3} and the private portion of the data record, denoted as d_{pri}, the Upload algorithm encrypts d_{pri} using κ_{n_3}. The pseudocode for Upload is provided in Algorithm C.6 in Appendix C.

Audit $(\textbf{PP}, d_n) \to b \in \{0, 1\}$

The Audit algorithm takes a data record d_n and returns one if and only if it is an authenticated data record, as specified in Definition 6.2. More precisely, it verifies the ZKP included in d_n, and checks the correctness of $\texttt{rt}_\tau$ using $\mathbf{L}_\tau^{\mathsf{PI}}(\texttt{rt}_\tau)$.

If the set of predecessor data records pred is non-empty, the algorithm retrieves each predecessor and checks whether they share the same PKsig. Otherwise, d_n must include a data ownership transfer tag, which consists of the PKsig in d_n signed by the key that signed the predecessor data record. The Audit algorithm then verifies the signature in d_n as well as the data ownership transfer tags, if present. The pseudocode for Audit is provided in Algorithm C.7 in Appendix C.

6.3.2 Protocols

Ownership Transfer (OT-Protocol)

Let A and B be two consecutive entities within a supply chain. A (the transferor) wants to transfer T_1^A to B (the transferee). They communicate via an off-chain secure channel, which can be realized in multiple ways, such as through their in-person interaction during the product transfer. First, A requests B for a new PKsig$_2^B$. Then, passes T_1^A and PKsig$_2^B$ to the Trans algorithm to generate $\tilde{T}_2^B$ and $\texttt{Tag}_1$. Then, gives them to B. Subsequently, B samples new pairs of PKsig$_3^B$ and SKsig$_3^B$ and runs the Trans again to obsolete the received AAT ($\tilde{T}_2^B$) and create $\tilde{T}_3^B$. This preserves the privacy of the lifecycle of the transferred AAT from A, who can determine when the AAT T_2^B is transferred since A knows ρ_2^B. However, A cannot discern if, when and to whom T_3^B is transferred. Also, entities not involved cannot recognize a link between the corresponding AAT commitments, i.e., $\texttt{cm}_1^A$, $\texttt{cm}_2^B$, $\texttt{cm}_3^B$. Additionally, for merging and dividing, A runs Merge or Divide locally, then transfers the outputted token(s) using the explained protocol.Figure 6.4 describes the protocol.

Protocol: Anonymous Authentication Token Ownership Transfer (AATOT)

Entity **A**		Entity **B**
$T_1^{\mathrm{A}} = \big((q_1^{\mathrm{A}}, \mathrm{PKsig}_1^{\mathrm{A}}, \rho_1^{\mathrm{A}}), \mathrm{SKsig}_1^{\mathrm{A}}\big)$	$\xrightarrow{\text{“Req PKsig}_2\text{”}}$	
	$\xleftarrow{\mathrm{PKsig}_2^{\mathrm{B}}}$	$(\mathrm{PKsig}_2^{\mathrm{B}}, \mathrm{SKsig}_2^{\mathrm{B}}) \leftarrow B^{\mathcal{K}_{\mathsf{sig}}}(\mathrm{PP}_{\mathsf{sig}})$
$(\mathtt{tx}^{A}_{\mathsf{Trans}_1}, \tilde{T}_2^{\mathrm{B}}, \mathtt{Tag}_1) \leftarrow A^{\mathsf{Trans}}(\mathrm{PP}, T_1^{\mathrm{A}}, \mathrm{PKsig}_2^{\mathrm{B}})$ Sends $\mathtt{tx}^{A}_{\mathsf{Trans}_1}$ to $\mathcal{C}_Z$	$\xrightarrow{\tilde{T}_2^{\mathrm{B}};\ \mathtt{Tag}_1}$	$T_2^{\mathrm{B}} \leftarrow (\tilde{T}_2^{\mathrm{B}}, \mathrm{SKsig}_2^{\mathrm{B}})$ $(\mathrm{PKsig}_3^{\mathrm{B}}, \mathrm{SKsig}_3^{\mathrm{B}}) \leftarrow B^{\mathcal{K}_{\mathsf{sig}}}(\mathrm{PP}_{\mathsf{sig}})$ $(\mathtt{tx}^{B}_{\mathsf{Trans}_2}, \tilde{T}_3^{\mathrm{B}}, \mathtt{Tag}_2) \leftarrow B^{\mathsf{Trans}}(\mathrm{PP}, T_2^{\mathrm{B}}, \mathrm{PKsig}_3^{\mathrm{B}})$ Sends $\mathtt{tx}^{B}_{\mathsf{Trans}_2}$ to $\mathcal{C}_Z$

Figure 6.4: OT-Protocol: A and B are two entities where A transfers the ownership of an AAT to B.

Merkle Hash Tree (MHT-Protocol)

Storing all $2^L - 1$ nodes of an L-layer MHT on the blockchain is costly. However, in $\mathcal{C}_Z$, it suffices to maintain $\mathtt{rt}_\tau$. Therefore, MHT populating follows a mechanism to let $\mathcal{C}_Z$ verify consistency among the current $\mathtt{rt}_\tau$, the new $\mathtt{cm}$, and the new $\mathtt{rt}^{\text{new}}$. Such that, MHT leaves are initialized with their $\mathsf{ind} \in [2^{L-1}]$. MHT.Add adds the new $\mathtt{cm}$ to the smallest unassigned ind, which is tracked in $\mathcal{C}_Z$. MHT.Verify first investigates $\mathsf{path}_{\mathsf{ind}}$ to ensure it is consistent with $\mathtt{rt}_\tau$ for the leaf that holds the value ind. Then, it passes $\mathtt{cm}$ to $\mathsf{path}_{\mathsf{ind}}$ and expects $\mathtt{rt}^{\text{new}}$. If both conditions passes, the root updates from $\mathtt{rt}_\tau$ to $\mathtt{rt}^{\text{new}}$ in $\mathcal{C}_Z$. The pseudocodes for MHT.Add and MHT.Verify are presented in Algorithms 6.2 and 6.3, respectively.

Algorithm 6.2 $\mathsf{MHT.Add}(\mathtt{cm}) \rightarrow (\mathtt{rt}^{\text{new}}, \mathsf{ind}, \mathsf{path}_{\mathsf{ind}})$

```
1: $\mathsf{ind} \leftarrow$ The smallest unassigned index maintained in BP
2: $\mathsf{path}_{\mathsf{ind}} \leftarrow$ The Merkle proof corresponding to the index ind.
3: Assign $\mathtt{cm}$ to the index ind of MHT
4: $\mathtt{rt}^{\text{new}} \leftarrow$ Updates MHT root after adding $\mathtt{cm}$
5: return $(\mathtt{rt}^{\text{new}}, \mathsf{ind}, \mathsf{path}_{\mathsf{ind}})$
```

Property 6.1. *MHT.Verify Algorithm guarantees that $\mathtt{rt}^{new}$ is the new root after a new cm is appropriately assigned to ind.*

To prove Property 6.1, we first need to present and prove Lemmas 6.1 and 6.2.

Algorithm 6.3 $\mathsf{MHT.Verify}(\mathtt{rt}_\tau, \mathtt{rt}^{\text{new}}, \mathtt{cm}, \mathsf{ind}, \mathsf{path}_{\mathsf{ind}}) \rightarrow b \in \{0, 1\}$

```
1: if not VerifyPath($\mathtt{rt}_\tau$, , $\mathsf{path}_{\mathsf{ind}}$, $\mathsf{ind}$) then
2:     return 0
3: end if
4: if not VerifyPath($\mathtt{rt}^{\text{new}}$, $\mathsf{path}_{\mathsf{ind}}$, $\mathtt{cm}$) then
5:     return 0
6: end if
7: return 1
```

Lemma 6.1. *The Merkle Hash Tree provides collision resistance, meaning that it is computationally infeasible to find two different sets of leaves that produce the same Merkle root.*

Proof. we can make use of the properties of hash functions and the structure of the Merkle Tree to justify this lemma. □

Lemma 6.2. *A valid Merkle proof ($Path_i$) and Merkle root ($\mathtt{rt}$) for a leaf with index i uniquely determine the leaf at index i. In other words, it is computationally infeasible to generate the same valid Merkle proof and root combination for two different leaves in a Merkle tree.*

Proof. Due to the cryptographic properties of hash functions and the construction of Merkle trees, it is computationally impractical for an adversary to generate the same valid Merkle proof and root combination for two different leaves. □

Proof of Property 6.1. The algorithm initially executes $\mathsf{VerifyPath}$ to ascertain if the provided Merkle proof ($\mathsf{path}_{\mathsf{ind}}$) corresponds to the leaf at index ind, which also contains the value ind. This step also ensures that the leaf has not previously held any commitment.

Then, the algorithm executes $\mathsf{VerifyPath}$ again. However, this time it changes the value of ind to $\mathtt{cm}$. While the Merkle proof $\mathsf{path}_{\mathsf{ind}}$ remains the same, the root is updated to $\mathtt{rt}^{\text{new}}$.

If both calls to $\mathsf{VerifyPath}$ return 1, indicating success, it can be concluded that $\mathtt{cm}$ has been added to the index ind, and $\mathtt{rt}^{\text{new}}$ represents the new root after the addition of the new authentication token commitment.

If an adversary attempts to overwrite an existing commitment $\mathtt{cm}^*$ at index ind^*, the first $\mathsf{VerifyPath}$ execution will return 0, as the value of the leaf at that index does not equal ind^*. Moreover, according to Lemma 6.2, it is computationally infeasible to generate a valid

Merkle proof and root combination for different leaf indices in a Merkle tree. Additionally, Lemma 6.1 states that there is a negligible probability of two Merkle trees having the same Merkle root, where one holds the value $\texttt{cm}^*$ at index ind^* and the other holds ind^* at the same index. Therefore, the algorithm can ascertain that the new commitment is added to the index ind, which has not previously held any commitment.

If the adversary aims to corrupt the root by passing $\texttt{rt}^*$ as an argument to the second $\mathsf{VerifyPath}$ execution, where $\texttt{rt}^*$ is not computed correctly from adding the new $\texttt{cm}$ to MHT, $\mathsf{VerifyPath}$ will return 0. This is because $\mathsf{path}_{\mathsf{ind}}$ is not consistent with $\texttt{rt}^*$ and the commitment. □

6.4 Security Analysis

6.4.1 Adversary Assumptions

In the Zupply framework Π, The adversary $\mathcal{A}$ is probabilistic polynomial-time (PPT) and has access to $\mathbf{TX}_\tau$, $\mathbf{E}_\tau$, $\mathbf{C}_\tau$, $\mathbf{X}_\tau$, MHT and $\mathbf{D}_\tau$ and is capable of executing either the exact or modified versions of the algorithms defined in Π, except the Setup algorithm, when a trusted setup zkSNARK is employed. The capabilities of the adversary can be enhanced by assuming it has oracle access to the following two functionalities:

1. *AAT commitment attribution oracle*: The adversary $\mathcal{A}$ can query an oracle, denoted as $\mathcal{O}^{\mathsf{Attr}} : \mathbf{C}_\tau \mapsto \mathbf{E}_\tau$. This oracle acts as a mapping function, $\mathcal{O}^{\mathsf{Attr}}(\texttt{cm}_i) = e_j$, where it assigns each commitment $\texttt{cm}_i$ to the entity e_j that created it.

2. *AAT commitment classification oracle*: The adversary $\mathcal{A}$ has the capability to categorize commitments based on the type of transaction they're contained in. The Adversary's oracle access for classifying AAT commitments is:
$\mathcal{O}^{\mathsf{Class}} : \mathbf{C}_\tau \mapsto \{\mathsf{Init}, \mathsf{Trans}, \mathsf{Merge}, \mathsf{Div}\}$.

If we assume that blockchain public addresses reveal the identities of their owners, the first assumption becomes realistic. Many existing on-chain privacy-preserving protocols [8, 48, 137, 163] do not adequately address this assumption. We formally define this assumption later in Section 6.4.2. Under this assumption, measuring the anonymity and unlinkability of entities within the Zupply framework requires evaluating the probability that each entity sends a specific transaction type to the blockchain within a given time interval (e.g., during block generation). The second assumption states that an adversary cannot distinguish between transactions of different types, as their contents are indistinguishable. This assumption is adopted without further elaboration.

Potentially, in the Zupply framework, the adversary $\mathcal{A}$ may attempt to

1. Link a data record, d_n, to its corresponding AAT commitment, $\texttt{cm}_i$ (*Data de-anonymizing attack*).
2. Determine whether two AAT commitments, $\texttt{cm}_i$ and $\texttt{cm}_j$, which are created by different entities are consecutive authentication tokens, with one having been transferred to the other (*Linking attack*).
3. Forge π^*_{Auth} and σ^* such that they are consistent to an AAT that the attacker does not have access to (*Data forging attack*).
4. Generate transactions $\texttt{tx}_{\mathsf{Trans}}$, $\texttt{tx}_{\mathsf{Merge}}$, $\texttt{tx}_{\mathsf{Div}}$ to transfer, merge, or divideAATs that the attacker does not have access to (*AAT ownership spoofing attack*).
5. Alter any existing $d_n \in \mathbf{D}_\tau$ (*Data tampering attack*).
6. Re-transfers a token that is already transferred, merged, or divided (*AAT double transfer attack*).

6.4.2 Indistinguishability Experiment

In this section, we do not provide security proofs, deferring them to future work. Instead, we discuss the formalization of the privacy property within the Zupply framework. Informally, consider an adversary $\mathcal{A}$ that constructs two ledgers and two corresponding DAGs by inducing honest entities, ensuring consistency between the public information of ledgers and DAGs. If the adversary is then provided with both ledgers but only one DAG, it should not be able to distinguish the ledgers. Thus, the adversary remains uncertain about the association between the given DAG and the respective ledgers.

To formalize this, we present the experimental setup for evaluating ledger indistinguishability. To characterize the ledger indistinguishability of the Zupply framework Pi, we design an $\mathsf{L\text{-}IND}$ experiment following the similarly named experiment in Zerocash [31]. Recall that a ledger L includes a set of transactions $\mathbf{TX}$. From $\mathbf{TX}$, the ordered set $\mathbf{C}$, which stores commitments to AATs ($\texttt{cm}$s), and the set of EoLs, denoted as $\mathbf{X}$ can be computed. Furthermore, the MHT can be constructed from $\mathbf{C}$. Each $\texttt{cm}$ in $\mathbf{C}$ can be replaced by a tuple $(\texttt{cm}, \text{TXtype}, e)$, where $\texttt{cm}$ is the commitment added to the ledger by an entity e

through a transaction of type TXtype. Here, for simplicity, the transaction type is either Trans or Init. We denote this new set as $\mathbf{C}_G$ as the generalized form of $\mathbf{C}$. Notably, if TXtype $= \mathsf{Trans}$, the entity e does not *own* the corresponding AAT; rather, it only adds it to the ledger. The adversary's capabilities determine whether they can extract the entity e associated with each commitment in the transactions ($\mathcal{A}$'s access to $\mathcal{O}^{\mathsf{Attr}}$).

Let $\mathcal{C}$ denote the challenger in the L-IND experiment, which receives queries from the adversary $\mathcal{A}$, performs sanity checks, and forwards them to the Zupply framework oracle, denoted as $\mathcal{O}^{\Pi}$. In the L-IND experiment, we instantiate two oracles, $\mathcal{O}^{\Pi}_0$ and $\mathcal{O}^{\Pi}_1$. Each oracle $\mathcal{O}^{\Pi}$ internally stores:

1. The set $\mathbf{C}_G$ (generalized form).
2. A set of entities $\mathbf{E}$.
3. A set of EoLs, denoted as $\mathbf{X}$.
4. Sets of AATs *owned* by each entity $e \in \mathbf{E}$, denoted as $\mathbf{T}^e$.
5. A set of data records, denoted as $\mathbf{D}$.

The oracle $\mathcal{O}^{\Pi}$ is initialized by public parameters PP and can receive and process the following queries:

- $Q(\mathbf{Init}, q, e)$
  1. Compute $\mathsf{Init}(\mathrm{PP}, q) \rightarrow (\mathtt{tx}_{\mathsf{Init}}, T)$, where $\mathtt{tx}_{\mathsf{Init}} = \left(\mathtt{cm}, \mathtt{rt}^{\mathrm{new}}, \mathsf{ind}, \mathsf{path}_{\mathsf{ind}}\right)$,
  2. Add $(\mathtt{cm}, \mathsf{Init}, e)$ to $\mathbf{C}_G$.
  3. Add T to $\mathbf{T}^e$
- $Q(\mathbf{Trans}, \mathtt{cm}^{\mathrm{old}}, e^{\mathrm{new}})$
  1. Assert $\mathtt{cm}^{\mathrm{old}}$ is in $\mathbf{C}$, otherwise output $\perp$.
  2. Let $T^{e^{\mathrm{old}}} \in \mathbf{T}^{e^{\mathrm{old}}}$ be the AAT corresponding to $\mathtt{cm}^{\mathrm{old}}$.
  3. Sample $(\mathrm{PKsig}^{\mathrm{new}}, \mathrm{SKsig}^{\mathrm{new}}) \leftarrow \mathcal{K}_{\mathsf{sig}}(\mathrm{PP})$
  4. Compute $\mathsf{Trans}\left(\mathrm{PP}, T^{e^{\mathrm{old}}}, \mathrm{PKsig}^{\mathrm{new}}\right) \rightarrow \left(\mathtt{tx}_{\mathsf{Trans}}, \tilde{T}^{e^{\mathrm{new}}}, \mathsf{Tag}\right)$, where $\mathtt{tx}_{\mathsf{Trans}} = \left(\pi_{\mathsf{Trans}}, x_{\mathsf{Trans}}, \mathtt{eol}, \mathtt{cm}^{\mathrm{new}}, \mathtt{rt}^{\mathrm{new}}, \mathsf{ind}, \mathsf{path}_{\mathsf{ind}}\right)$ and $x_{\mathsf{Trans}} = \left(\mathtt{eol}, \mathtt{cm}^{\mathrm{new}}, \mathtt{rt}^{\mathrm{old}}\right)$
  5. Add $(\mathtt{cm}, \mathsf{Trans}, e^{\mathrm{old}})$ to $\mathbf{C}_G$.

6. Add $T^{e^{\text{new}}} = (\tilde{T}^{e^{\text{new}}}, \text{PKsig}^{\text{new}})$ to $\mathbf{T}^{e^{\text{new}}}$

- $Q(\textbf{Upload}, \text{IsTransferred}, d^{\text{old}}, d^{\text{new}}_{\text{pub,pri}}, e^{\text{new}})$, where $d^{\text{new}}_{\text{pub,pri}}$ represents the public and private parts of the new data to be added to D, succeeding d^{old}, which is already in D. The Boolean variable IsTransferred indicates whether the new data has a different PKsig than d^{old}; it is set to `True` if PKsig differs and `False` otherwise.[2]

1. Assert d^{old} is either Null or is in D and it is the last data in a progressive data sequence, otherwise output $\perp$.
2. If $d^{\text{old}} \neq \text{Null}$, find the T^{old} with $\text{PKsig}^{\text{old}}$ in all $\mathbf{T}^{e}$s and from that determine $\text{SKsig}^{\text{old}}$ otherwise do nothing.
3. If IsTransferred $=$ `True`, then commit to T^{old} to compute $\texttt{cm}^{\text{old}}$, and execute $Q(\textbf{Trans}, \texttt{cm}^{\text{old}}, e^{\text{new}})$ to produce $T^{e^{\text{new}}}$. Otherwise, $T^{e^{\text{new}}} = T^{\text{old}}$.
4. Sample a random κ and Compute $\mathsf{Upload}\left(\text{PP}, d^{\text{old}}, T^{e^{\text{new}}}, d^{\text{new}}_{\textsf{pub,pri}}, \kappa, [\textsf{tags}]\right) \rightarrow d^{\text{new}}$.
5. Add d^{new} to D.

- $Q(\textbf{InsertTX}, \texttt{tx})$, where $\texttt{tx}$ is either the type of Trans or Init.

1. If $\mathsf{VerifyTX}\left(\text{PP}, \texttt{tx}\right)$ returns 0, return $\perp$.
2. Add the $(\texttt{cm}, \text{TXtype}, e^*)$ to $\mathbf{C}$, where e^* denote the type of entities where the oracle does not hold their T^{e^*}.
3. If $\text{TXtype} = \mathsf{Trans}$ and the receiver of the AAT is not of type e^*, add Add $T^{e^{\text{new}}}$ to $\mathbf{T}^{e^{\text{new}}}$.

- $Q(\textbf{InsertData}, d)$,

1. If $\mathsf{Audit}\left(\text{PP}, d\right)$ returns 0, return $\perp$.
2. Add d to $\mathbf{D}$.

In the $\mathsf{L\text{-}IND}$ experiment, the challenger $\mathcal{C}$ instantiates two oracles, denoted as $\mathcal{O}^{\Pi}_0$ and $\mathcal{O}^{\Pi}_1$. Then, $\mathcal{C}$ randomly selects $b \in \{0,1\}$ and provides the adversary $\mathcal{A}$ with two ledgers: $L_{\text{Left}} = L_b$ and $L_{\text{Right}} = L_{1-b}$. The adversary then submits two queries, Q and Q', to $\mathcal{C}$, where both queries are of the same type and are publicly consistent, meaning all public values in these queries are identical. Let D_b and D_{1-b} denote the data sets corresponding

[2]The query $Q(\textbf{Upload}, \text{False}, \text{Null}, d^{\text{new}}_{\text{pub,pri}}, e^{\text{new}})$ initiates a DAG.

to each ledger. If the queries are of type **InsertTX** or **InsertData**, the challenger $\mathcal{C}$ sends Q to (L_b, D_b) and Q' to (L_{1-b}, D_{1-b}). Otherwise, $\mathcal{C}$ sends Q to (L_0, D_0) and Q' to (L_1, D_1).

Notably, here we do not assume that the adversary has access to the $\mathcal{O}^{\mathsf{Attr}}$ oracle. As a result, the adversary is restricted to constructing $\mathbf{C}$ from each ledger and cannot derive $\mathbf{C}_G$ from them. Consequently, selecting different entities in queries Q and Q' does not violate the public consistency of the queries. Furthermore, selecting different entities does not provide $\mathcal{A}$ with any advantage in distinguishing the ledgers.

The adversary's advantage in this experiment, denoted as $\mathsf{Adv}_{\Pi}^{\mathsf{L\text{-}IND}}$, is the probability that the adversary can correctly guess b given L_b, L_{1-b}, and D_0. The core idea of the experiment is that each random-looking value is either randomly sampled or simulated, rather than being generated according to the actual algorithm. Later, in the proof, it must be shown that the simulated results are indistinguishable from those produced by the actual algorithm execution. This can be achieved by leveraging the security properties of the underlying building blocks.

Here, we provide an overview of the $\mathsf{L\text{-}IND}$ experiment without delving into the details. In this experiment, when $\mathbf{C}$ receives an **Init** query, it modifies the Init algorithm for each Q and Q', constructing the commitments `cm` by committing to random values instead of AATs. When $\mathbf{C}$ receives a **Trans** query, it modifies the Trans algorithm such that the `eol` value included in the created transaction is randomly sampled. If e^{new} is not one of the entities whose secrets the adversary knows (i.e., $e^{\text{new}} \neq e^*$), $\mathbf{C}$ randomly samples $T^{e^{\text{new}}}$ and `cm`; otherwise, it follows the original Trans algorithm. Finally, the ZKPs in **Trans** and **Upload** queries are simulated based on their public inputs. We leave the detailed description of the experiment and its proof for future work.

Now, suppose the adversary has access to $\mathcal{O}^{\mathsf{Attr}}$. In this case, $\mathcal{C}$ should not enforce $\mathcal{A}$ to use identical entities in both Q and Q' queries to satisfy the public consistency requirement. Consequently, the adversary would be able to distinguish between the two ledgers (i.e., guess b) with a significant probability. To address this scenario, additional assumptions are required, such as assigning a probability to each entity in an anonymity set for sending a specific transaction type within a given time interval. The formalization of this scenario and the measurement of anonymity based on the aforementioned probability will be addressed in future research.

6.5 Discussion

In this section, we discuss the applicability of the Zupply framework in real-world scenarios.

6.5.1 Product Lifecycle

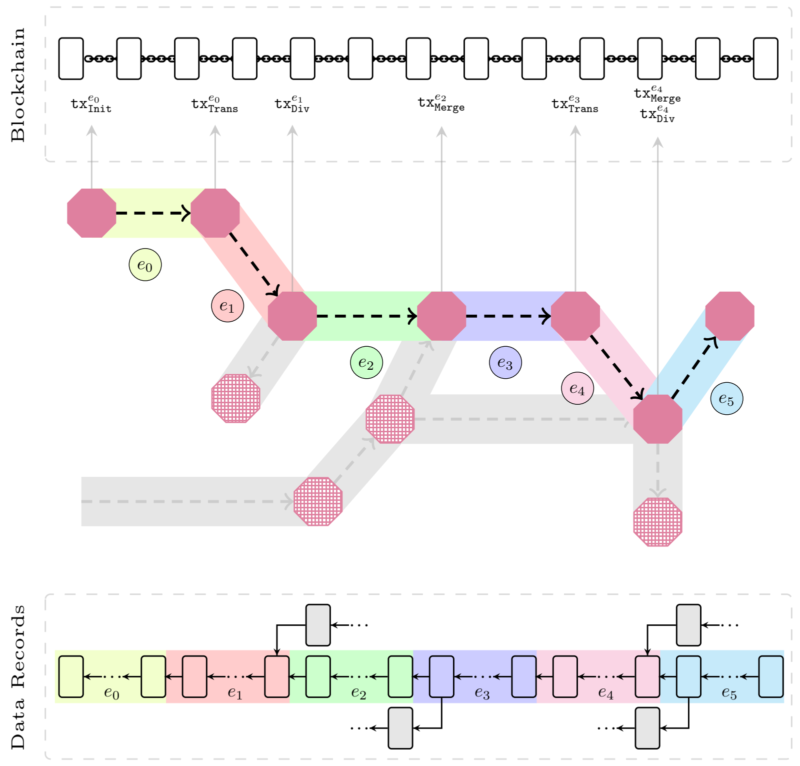

Blockchain
$\mathtt{tx}^{e_0}_{\mathtt{Init}}$
$\mathtt{tx}^{e_0}_{\mathtt{Trans}}$
$\mathtt{tx}^{e_1}_{\mathtt{Div}}$
$\mathtt{tx}^{e_2}_{\mathtt{Merge}}$
$\mathtt{tx}^{e_3}_{\mathtt{Trans}}$
$\mathtt{tx}^{e_4}_{\mathtt{Merge}}$
$\mathtt{tx}^{e_4}_{\mathtt{Div}}$
e_0
e_1
e_2
e_3
e_4
e_5
Data Records
e_0
e_1
e_2
e_3
e_4
e_5

Figure 6.5: The lifecycle of a Product in the Zupply framework

The Zupply framework is designed to ensure the integrity and authenticity of a product's historical data, represented as a DAG.

This section describes how the Zupply framework can be employed in its main application i.e., SCM. Entities in our framework refer to participants in a supply chain — including producers, distributors, and retailers — who upload data records related to products. These entities have three primary objectives in this ecosystem: verifying the authenticity of previously uploaded data records, preserving the integrity of their data records over time, and maintaining their anonymity. Figure 6.5 illustrates all three components of the Zupply framework in an example supply chain.

The initial entity in the supply chain, referred to as e_0 (e.g., a supplier of raw materials), creates an AAT, denoted as $T^{e_0}_{j_1}$. To accomplish this, e_0 executes the Init function, passing the product quantity (q_{j_1}) as an argument. Consequently, a new token, $T^{e_0}_{j_1}$, is generated, and its commitment, defined as $\mathtt{cm}^{e_0}_{j_1} := \mathsf{COMM}_{\rho_{j_1}}(T^{e_0}_{j_1})$, is transmitted to the blockchain through a transaction termed $\mathtt{tx}^{e_0}_{\mathsf{Init}}$. Then updates its localized version of the MHT, and also updates the root $\mathtt{rt}_\tau$ on the smart contract ($\mathcal{C}_Z$). The smart contract verifies the correctness of the new root accordingly (as explained in Section 6.3.2). Following this, entity e_0 can begin uploading data records to the DCS by utilizing the Upload function. This function employs a public parameters, referred to as PP, to establish a ZKP, proving the ownership of an authentication token in MHT. Moreover, each data record is signed using a private key, denoted as $\text{SKsig}^{e_0}_{j_1}$. SKsig_i and its corresponding PKsig_i are included in $T^{e_0}_{j_1}$ (see Section 6.2.4). This ensures the authenticity and integrity of the data.

In supply chains, a product is transferred among various entities, and each entity needs to transfer the ownership of the products they possess. If the entity e_0 wants to give a batch of products to the next entity e_1 and the corresponding AAT to the batch is $T^{e_0}_{j_1}$, the entity executes Trans algorithm, passing the AAT as one of its arguments. The algorithm generates a new $\tilde{T}^{e_1}_{j_2}$ for the batch and passes it to e_1 in a secure off chain channel and submits transaction $\mathtt{tx}^{e_0}_{\mathsf{Trans}}$ to the blockchain (the AAT ownership transfer protocol, i.e., OT-Protocol, is presented in Section 6.3.2). According to the OT-Protocol, e_0 cannot use $\tilde{T}^{e_1}_{j_2}$ to upload data records since it does not know the corresponding $\text{SKsig}^{e_1}_{j_2}$. The transaction $\mathtt{tx}^{e_0}_{\mathsf{Trans}}$ contains a ZKP of four things: (1) knowing the pre-image of the $\mathtt{cm}^{e_0}_{j_1}$ which is in the MHT without revealing $\mathtt{cm}^{e_0}_{j_1}$, (2) the included $\mathtt{eol}$ in the transaction is related to $\mathtt{cm}^{e_0}_{j_1}$, (3) the proof of knowledge of $\tilde{T}^{e_1}_{j_2}$ related to $\mathtt{cm}^{e_1}_{j_2} := \mathsf{COMM}_\rho(T^{e_{i_2}}_{j_2})$ included in the transaction (4) the quantity $q_{j_2} = q_{j_1}$ does not change.

In the Zupply framework, entities can split a single authentication token into two using the Div algorithm. For instance, e_1 receives a batch from e_0 and decides to divide it into two smaller batches. One batch is sent to entity e_2, while the other is transferred to a different supply chain. Importantly, both supply chains should be able to access the product's history prior to this division. Additionally, entities have the ability to merge two of their authentication tokens into one using the Merge algorithm. For example, e_2 receives some products from a different supply chain and seeks to batch them together before passing them to e_3. Furthermore, auditors should have the capability to access and review the data records pertaining to both merged supply chains. These functionalities allow for the merging and dividing of data sequences associated with the AATs, thereby enabling the maintenance of data in a DAG structure within the framework.

6.5.2 Business Contracts

Employing smart contract-enabled blockchain platforms for managing AAT in the Zupply framework can enable business smart contracts that employ Zupply smart contract $\mathcal{C}_Z$ as a lower layer of their operations. Business smart contracts can provide financial security for entities so they will not rely on a trusted third party for their payments. One example of a business contract is transferring funds upon delivering a product (i.e., transferring an AAT). Here, we are going to present an example of a business contract $\mathcal{C}_B$ that checks whether a specific $\texttt{cm}$ is uploaded to $\mathcal{C}_Z$ or not. Then, it transfers the funds that were locked in $\mathcal{C}_B$.

In this example, e_1 wants to send a product to e_3 via e_2. e_2 uploads data collected from the product till it transfers that to e_3. Later on, e_3 is going to continue uploading data records related to the product. e_2 wants a guaranteed payment by the time the product is delivered to e_3. Therefore, any entity who is responsible for the payment (e_1 in this example) can lock some funds in $\mathcal{C}_B$. The funds will be transferred to e_2 if the entity uploads an AAT commitment $\texttt{cm}$, which has been determined at the time of locking the funds to $\mathcal{C}_B$.

The followings are the steps of the the protocol:

1. $(\text{PKsig}_1^{e_3}, \text{SKsig}_1^{e_3}) \leftarrow {e_3}^{\mathcal{K}_{\mathsf{sig}}}(\text{PP})$
2. e_3 passes $\text{PKsig}_1^{e_3}$ to e_1.
3. e_2: $\rho \overset{R}{\leftarrow} \{0,1\}^{O}(\lambda)$
4. e_2 passes ρ to e_1 upon receiving the product from e_1.
5. e_1 creates $\tilde{T} = (q, \text{PKsig}_1^{e_3}, \rho)$ $\texttt{cm} \leftarrow \mathsf{COMM}_\rho(\tilde{T})$
6. e_1 locks some funds in $\mathcal{C}_B$. These funds will go to e_2 whenever $\texttt{cm}$ is uploaded to $\mathcal{C}_Z$ in a $\texttt{tx}_{\mathsf{Trans}}$ transactions.
7. Upon receiving the product by e_3, both e_2 and e_3 will engage in the $\mathsf{OT\text{-}Protocol}$ presented in Section 6.3.2. e_3 will send $\text{PKsig}_1^{e_3}$ to e_2 after receiving "Public key request" from e_2 in the protocol.
8. e_2 uses ρ received from e_1 at the time of taking the product and $\text{PKsig}_1^{e_3}$ received from e_3 at the time of giving product. Then outputs a new authentication token commitment $\texttt{cm}$ via Trans algorithm.

9. In the previous step $\mathtt{cm}$ is uploaded to the $\mathcal{C}_Z$ contract; therefore, e_2 can take the funds from $\mathcal{C}_B$.

6.5.3 Data Encryption

In the Zupply framework, various key management schemes can be employed to encrypt the private parts of the data records, adapting them according to different applications. In this thesis, we suggest following the Mesh forward secrecy approach for generating keys [8]. Such that the secret keys κ are generated by the radio frequency identification (RFID) tag's Pseudo-random bit generator (PRBG). This approach ensures that, in supply chain applications, a new product owner must have access to all previous encryption keys to trace the product's history effectively. However, it should not be able to decrypt the future data records after the point of transfer to the following entity. Ultimately, the final entity in the supply chain can view the complete history of the associated data progressive chain.

6.6 Summary

Transferring applications such as SCM to permissionless (public) blockchains eliminates the need for trust in a manager and offers a range of benefits, which we will discuss later in this paper. However, by the nature of permissionless blockchain, miners, monitors, auditors and validators see the plain transactions on the blockchain. This could reveal the privacy of the data providers and actors in ownership transfer of the data from one to another. However, those will expose inventory information of different business sectors. So, for blockchain based SCM, the privacy of data uploader (anonymous problem), and ownership transfer of data (unlinkable problem) is a crucial factor to deploy this technology.

This chapter proposed a novel anonymous authentication token (AAT) framework that supports unlinkable AAT ownership transfer (AATOT) including unlinkable merging and dividing of AATs, realized by $\mathsf{OT\text{-}Protocol}$ to ensure unlinkability during ownership transfer. The construction leverages zero-knowledge proof (ZKP) protocols to guarantee anonymity, unlinkability, and authentication. Building on this foundation, we introduce Zupply, a decentralized framework for maintaining directed acyclic graphs (DAGs) of authentic data records. Designed on a permissionless, smart contract-enabled blockchain, Zupply provides a trustless framework that preserves anonymity and unlinkability among participants while ensuring the integrity and authenticity of data records throughout the supply chain. Its efficiency is realized by minimizing the number of public inputs to the

ZKPs, and the MHT-Protocol which updates the root of the MHT on the $\mathcal{C}_Z$ and eliminate the need for storing the entire tree on-chain.

Zupply adopted a stricter adversary model that previous methods, in which the underlying blockchain platform does not preserve the anonymity of the transaction issuer (adversary's access to $\mathcal{O}^{\mathsf{Attr}}$). Moreover, transactions scale independently of participant count, Zupply requires no centralized servers or layer-1 changes, and let auditors focus only on relevant records. In contrast, zkLedger [137]has to involve all entities per transaction, Mesh [8] lacks unlinkability and needs a central server, and DECOUPLES [129] modifies the base blockchain.

Chapter 7

Zupply Framework Implementation

Declaration of Contributions

This chapter is based on [17] and [18]. I am the sole author of this chapter.

7.1 Introduction

In the previous two chapters, we introduced the design and security analysis of the Zupply framework. This chapter focuses on its implementation, detailing the instantiation of core building blocks, and the implementation of the zero-knowledge proving and verification algorithms. Specifically, we describe the arithmetic circuits for the NP problems associated with Auth, Trans, Merge, and Div. These circuits generate R1CS instances in a manner that enables succinct zero-knowledge proofs (ZKPs) while preserving confidentiality. To optimize on-chain costs (i.e., smart contract execution) we minimize the public inputs to these circuits. We provide two implementation variants of Zupply based on its ZKP algorithms. The first employs Groth16 [95] zkSNARK over BN254 [23] and BLS12-381 [22] elliptic curves, accommodating different security levels. The second leverages Aurora [35], a zkSNARK designed for a transparent setup and post-quantum security against quantum-capable malicious provers.

Zupply is implemented in C++ and Solidity, demonstrating both computational and cost efficiency. This development validates the framework's practicality for real-world applications. While directed acyclic graphs (DAGs) are widely used for maintaining product histories in supply chains, Zupply's utility extends beyond this domain. We show that Zupply provides a practical, cost-efficient, and privacy-preserving solution for decentralized supply chain management (SCM) systems.

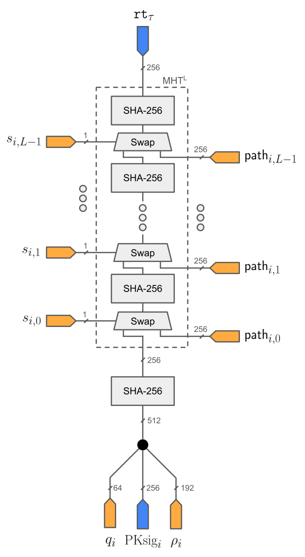
7.2 Zupply Arithmetic Circuits

The Auth, Trans, Merge, and Divide algorithms in the Zupply framework generate ZKPs to demonstrate knowledge of the pre-image for one or two commitments to anonymous authentication tokens (AATs) in the Merkle hash tree (MHT), with the root at time τ is denoted by $\texttt{rt}_\tau$. These ZKPs are used in Zupply algorithms to prove ownership of an AAT for authenticating a data record or to transfer ownership of the AATs. In all cases, the prover avoids revealing the AAT or its commitment stored in the MHT. Figure 7.1 shows a schematic view of the arithmetic circuits associated with each of these ZKPs.

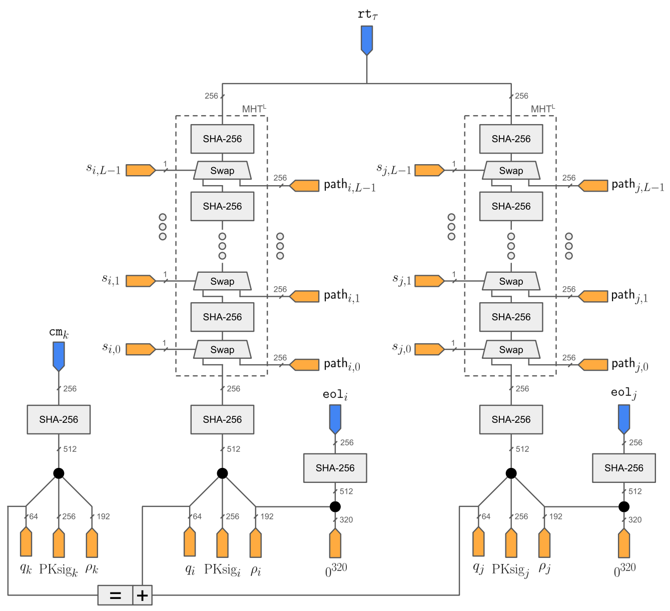
These circuits represent the NP problem related to each ZKP and are converted into R1CS, which are satisfied only when the prover provides consistent inputs to the constraint system. For example, in a circuit designed to prove knowledge of the pre-image of a leaf in a Merkle hash tree with root $\texttt{rt}$, without revealing the specific leaf, the root of the Merkle hash tree is given as a public input. The private inputs include the corresponding leaf pre-image and the Merkle proof. These private inputs must align with the root and satisfy the circuit constraints. As a result, the prover must provide valid private inputs, thereby demonstrating knowledge of the pre-image of a leaf in the Merkle hash tree. In the Zupply framework, the root value is determined by the shared MHT, which is maintained within the framework and serves as a public input to the circuits. The prover must demonstrate knowledge of the pre-image of a leaf in the fully decentralized MHT. Importantly, this process does not reveal any information about the AAT or its commitment (the leaf itself).

Each arithmetic circuit in the Zupply framework includes a proof of knowledge for one or two leaves within the MHT. These circuits are constructed using the arithmetic circuits for the hash function and the Merkle hash tree. The R1CS generated by the hash function arithmetic circuit is satisfied if and only if, for a given input value x assigned to the circuit, the output value y satisfies $y = \mathcal{H}(x)$, where $\mathcal{H}$ is the hash function. The arithmetic circuit of the Merkle hash tree implements the Merkle proving algorithm. At each level, a hash function output is concatenated to either the left or right of its adjacent hash. The concatenated value is then input to the hash function of the subsequent layer, continuing until the root is reached. Formally, this chain of hash functions is expressed as:

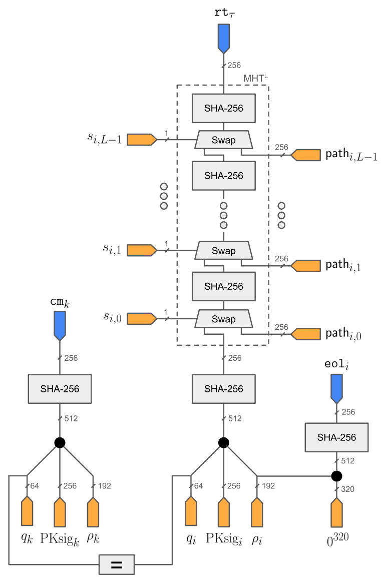
$$h_{i,j+1} = \mathcal{H}\left(s_{i,j} \cdot (h_{i,j} \mid \mathsf{path}_{i,j}) + (1 - s_{i,j}) \cdot (\mathsf{path}_{i,j} \mid h_{i,j})\right),$$



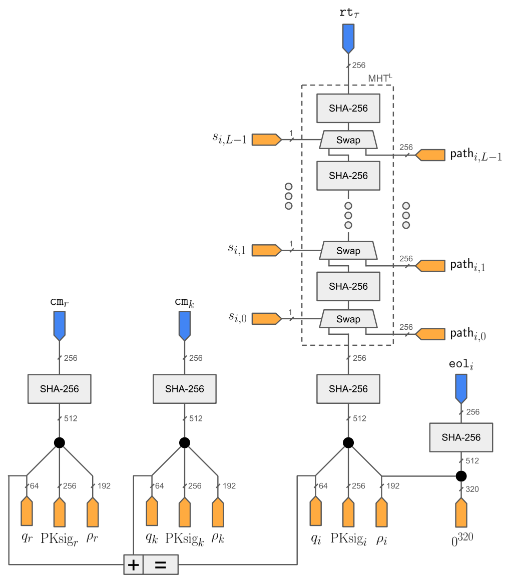
(a) Auth circuit



(b) Merge circuit



(c) Trans circuit



(d) Div circuit

Figure 7.1: Schematic representation of the arithmetic circuits for ZKPs in the Zupply framework. The Merkle hash tree has a depth of L, with public inputs highlighted in **blue**. Inputs highlighted in **orange**, along with the intermediate values within the arithmetic circuits, are private (auxiliary) inputs. The hash function used in the implementation is SHA-256.

where at layer j, $h_{i,j}$ is the output from the previous hash in the chain. Let $h_{i,0}$ denote the i-th leaf value and $h_{i,L}$ is equal to the root $\texttt{rt}$. $\mathsf{path}_{i,j}$ represents the adjacent hash at layer j in the Merkle proof related to index i, and $s_{i,j} \in \{0, 1\}$ determines whether $h_{i,j}$ is concatenated to the left or right of its adjacent hash. The arithmetic circuit responsible for managing this order selection is called the "Swap" block. The R1CS generated by the arithmetic circuit of a Merkle hash tree with L layers is satisfied if and only if the leaf h_0 exists in a Merkle hash tree with root $\texttt{rt}$, and the Merkle proof, comprising $\{\mathsf{path}_{i,0}, \ldots, \mathsf{path}_{i,L-1}\}$ and $\{s_{i,0}, \ldots, s_{i,L-1}\}$, is consistent with both the leaf and the root.

In the following, we delve into the details of each circuit.

7.2.1 Auth Circuit

The Auth circuit, illustrated in Figure 7.1a, generates the ZKP π_{Auth}, which is embedded in data records to prove that the data creator possesses an AAT within a Merkle hash tree. The root $\texttt{rt}_\tau$ serves as a public input to this proof to allow the auditor to verify that the commitment to the claimed AAT is stored in the MHT with root $\texttt{rt}_\tau$. Specifically, the prover asserts, "she knows a preimage of one of the commitments to an AAT in the MHT with root $\texttt{rt}_\tau$," without revealing which commitment. Additionally, the PKsig associated with the AAT is also provided as a public input. The data creator signs the data record using the corresponding SKsig, thereby proving that the data has been signed by the token owner and remains unaltered.

7.2.2 Trans Circuit

The Trans circuit generates the ZKP π_{Trans}, which is submitted to the blockchain platform when an entity transfers ownership of an AAT during OT-Protocol. Specifically, it obsoletes an AAT owned by the entity and creates a new AAT that is consistent with the obsoleted AAT. The circuit used for generating this proof is depicted in Figure 7.1c. In the Trans circuit, the pre-images of the commitments to both the new and obsoleted AATs are private inputs. The circuit ensures that the q value, representing the quantity of the product, remains unchanged. The commitment to the new AAT serves as a public input to the circuit, which will be added to the MHT. Additionally, the end of life ($\texttt{eol}$) value of the obsoleted AAT is a public input to the circuit and is published on the blockchain platform. This ensures that an entity cannot reuse an AAT included in the MHT but transferred before. In this proof, the prover asserts: "She knows an AAT in the MHT with root $\texttt{rt}_\tau$, where the end-of-life value of the claimed token is $\texttt{eol}$, and she has created a new AAT with a commitment `cm`, such that the quantity in the new token matches the quantity in the originally claimed token."

7.2.3 Merge Circuit

The Merge circuit is similar to the Trans circuit, but it merges two existing commitments to two known AATs in MHT with root `rt`. Hence, this circuit generates the ZKP π_{Merge} which proves the knowledge of two AATs, where their corresponding `eol`s are public inputs to this circuit, allowing the verification algorithm to check whether those AATs have been transferred before. This circuit also guarantees that the quantity of the product in the new commitment is equivalent to the sum of the product quantities in the two preceding AATs. The new commitment to the new AAT is a public input to this circuit.

7.2.4 Div Circuit

The Div circuit is the opposite of the Merge circuit. It generates the ZKP π_{Div}, which proves that a known existing AAT in MHT with root `rt` divides into two new AATs. The `eol` of the divided token and the commitments to the two new AATs are public inputs to this circuit. This circuit ensures consistency in the product quantity before and after division, so that the sum of the q values in the new tokens equals the q value in the original AAT.

7.3 Cryptographic Primitives and Parameters

In this section, we present the instantiation of each building block of the Zupply framework, considering a 128-bit security level.

7.3.1 Blockchain platform

Ethereum provides the highest level of security among all ethereum virtual machine (EVM) compatible blockchains, such as Arbitrum and Avalanche [139, 108, 151]. Therefore, we have selected Ethereum as the blockchain platform upon which Zupply is built. If Zupply remains practical with respect to the costs incurred when calling the smart contract $\mathcal{C}_Z$, then its feasibility extends to other EVM-compatible blockchains as well. To realize the proof-of-inclusion ($\mathbf{L}_{\tau}^{\mathsf{PI}}(\texttt{rt})$), Ethereum employes Merkle Patricia tree (trie) [75, 54] and provides `eth_getProof` API which is elaborated in EIP-1186 [106].

7.3.2 Decentralized Cloud Storage (DCS)

Zupply utilizes the InterPlanetary File System (IPFS) network [38] for file storage, offering flexibility in data management based on the preferences of the supply chain entities. For instance, entities can opt to share their data records directly from their computers via a Tor hidden service [125] to obscure their IP address, or they can entrust storage to major stakeholders within the supply chain. These stakeholders then anonymously publish the data records to the IPFS network, ensuring availability. Furthermore, Zupply guarantees the authenticity and integrity of data records, which are independently verified by auditors, thus eliminating reliance on DCS for this verification.

7.3.3 Collision-resistance Hash Function

The collision resistance hash function $\mathcal{H}$ is a 512-bit input to a 256-bit output mapping; namely, $\mathcal{H} : \{0,1\}^{512} \rightarrow \{0,1\}^{256}$. It can be implemented by either SHA-256 or SHA3-256. For SHA-256, it has 512 bit input and 256 output, so there is no padding bits needed. For SHA3-256, the input size is 1088 bits and outputs 256 bits. So, in this case, padding is needed. In our implementation, we selected SHA-256 for its simplicity and wide adoption.

7.3.4 Statistically-hiding Commitment

We instantiate the commitment scheme COMM via $\mathcal{H}$. Such that $\texttt{cm} := \mathsf{COMM}_{\rho}(\tilde{T}) = \mathcal{H}(q||\text{PKsig}||[\rho]_{224})$ Where, $q \in \{0,1\}^{32}$, $\text{PKsig} \in \{0,1\}^{256}$, and $\rho \in \{0,1\}^{512}$. Moreover, $[\rho]_{224}$ denotes that the 512-bit ρ is truncated to 224 bits by dropping most significant bits. Both PKsig and ρ can serve as randomness for the commitment.

7.3.5 Merkle Hash Tree

MHT is constructed based on the collision resistance hash function $\mathcal{H}$ (i.e., SHA-256). Let the number of layers L be equal to 20, then MHT has $2^{20} - 1$ nodes, and each node has 256 bits. Consequently, the tree requires 32 MB of storage. Recall that the tree is stored off-chain on entities' Z-NODE software, and only the root $\texttt{rt}$ is maintained on the smart contract $\mathcal{C}_Z$.

7.3.6 Strongly-unforgeable Digital Signature

The signature scheme algorithm Sig used in the framework is Elliptic Curve Schnorr signatures (EC-Schnorr) [158] over the elliptic curve secp256k1 [161] where the size of both PKsig and SKsig are 256 bits and signature has 512 bits. The algorithm uses SHA-256 as the hash function required for the signatures. This algorithm satisfies the *SUF-CMA* security property.

7.3.7 Symmetric-key Encryption

For SymEnc, we use 128-bit advanced encryption standard (AES-128) algorithm in cipher block chaining (CBC) mode which satisfies *IND-CPA* security property [152, 160].

7.4 zkSNARK Instantiations

The choice of zkSNARK protocol in the Zupply framework enables a trade-off between computational efficiency and enhanced security properties, such as post-quantum security and a transparent setup.

7.4.1 Groth16

In the first implementation variant of the Zupply framework, we employ Groth16 [95], a zkSNARK known for generating succinct proofs and enabling fast verification. This protocol is a pairing-based zkSNARK for R1CS. The scheme requires a trusted setup phase that can be done via a secure multi-party computation (SMPC) [33, 140]. We evaluate the implementation using two elliptic curves: BN254 [23], offering the fastest algorithms and the smallest proof sizes and approximately 100-bit security [20], and BLS12-381 [22], designed for 128-bit security.

By using the Groth16 zkSNARK in the Zupply framework, the verification keys ($\mathsf{vk}_{\mathbb{x}}$), stored in $\mathcal{C}_Z$, remain small when the number of public inputs ($x_{\mathbb{x}}$) is minimal. Additionally, the verification algorithm (Verify) of Groth16 is less complex compared to other zkSNARKs. Due to the high cost of on-chain storage, this efficiency makes Groth16 particularly suitable for public blockchains where execution and storage costs are high. In Groth16, the proof size remains constant regardless of the number of constraints in the associated R1CS. The proof is denoted as $\pi \in \mathbb{G}_1^2 \times \mathbb{G}_2$, where $\mathbb{G}_1$ consists of points on the elliptic curve, and $\mathbb{G}_2$ comprises points on a twist of the curve.

The size of the verification key vk depends on the number of public inputs in each R1CS instance within the Zupply framework. The public input of the instances in the Zupply framework are denoted as x_{Auth}, x_{Trans}, x_{Merge}, and x_{Div}. Let $|x_{\mathbb{x}}|$ represent the number of public inputs for an instance $\mathbb{x}$. As described in [95], vk consists of $|x_{\mathbb{x}}| + 2$ elements in $\mathbb{G}_1$ and three elements in $\mathbb{G}_2$. It may also include a precomputed element in $\mathbb{G}_T$ via the bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ applied to two specific elements in vk, since the verification process requires only their pairing, not those elements themselves. Consequently, one element from each of $\mathbb{G}_1$ and $\mathbb{G}_2$ can be omitted from vk and replaced by an element in $\mathbb{G}_T$.

In the Zupply framework, every public input is represented as either a 256-bit hash or a 256-bit public key. However, like other scalar elements in R1CS instances, these inputs must belong to the scalar prime field of the underlying elliptic curve. Since the scalar field elements of BN254 and BLS12-381 have bit-lengths of 254 and 255 bits, respectively, each 256-bit value is split into two 128-bit public inputs to minimize the public input size. In contrast, the original implementation of the SHA-256 arithmetic circuit in [32] treats each 256-bit hash as 256 distinct public inputs, significantly increasing the size of the verification keys. Based on our construction, the number of public inputs represented as $n(x_{\mathbb{x}})$ in Table 7.1 are double the count of public inputs as per the arithmetic circuits presented in Section 7.2. Moreover, the size of the proving key pk is determined by the number of constraints in each instance and the proving keys in the Zupply framework are stored off-chain on the devices of entities. The number of constraints is determined by the size of the circuit implementing NP statements in Section 6.2.5.

Over BN254 Curve

We initially implement the Groth16 zkSNARK over the BN254 curve. The curve was generally assumed to offer around 128 bits of security [23]; However, research by Kim et al. [115] suggests that its actual security level might be lower. In Groth16, the proof size is constant, denoted as $\pi \in \mathbb{G}_1^2 \times \mathbb{G}_2$, where $\mathbb{G}_1$ consists of points on the BN254 elliptic curve, and $\mathbb{G}_2$ comprises points on a twist of the BN254. Consequently, the proof size is 128 bytes, regardless of the size of the number of constraints in the R1CS related to the proof. Since Ethereum only provides pre-compiled contracts for the BN254 curve [55, 74], we selected this pairing-friendly curve for the on-chain Groth16 verification algorithms implementation.

Table 7.1: Comparison of verification and proving key sizes of Groth16 zkSNARK for NP statements in the Zupply framework for $L = 20$. $\nu = n(x_{\mathbb{x}})$ is the number of public inputs, N is the number of constraints, $|\mathsf{vk}_{\mathbb{x}}|$ is the size of the verification key in Bytes, and $|\mathsf{pk}_{\mathbb{x}}|$ is the size of the proving key in MegaBytes.

$\mathbb{x}$	ν	N	$\|\mathsf{vk}_{\mathbb{x}}\|$ (B)		$\|\mathsf{pk}_{\mathbb{x}}\|$ (MB)	
			BN254	BLS12-381	BN254	BLS12-381
Complexity	-	$O(L)$	$O(\nu)$		$O(N)$	
Auth	4	588,248	768	1,152	182.5	456.3
Trans	6	642,876	896	1,344	200	246.75
Merge	8	1,258,337	1,024	1,536	393.8	447.93
Div	8	670,091	1,024	1,536	210	262.5

Over BLS12-381 Curve

To strengthen the security of our framework, we replace the BN254 curve with BLS12-381, which offers 128-bit security. However, as of the time of writing, Ethereum does not provide precompiled support for operations over this curve [176]. Consequently, we did not implement our smart contract $C_{\mathcal{Z}}$ using BLS12-381, leaving this for future work. Adopting this curve increases the proof size to 192 bytes regardless of the size of the number of constraints in R1CS. Table 7.1 compares the verification and proving key sizes for each instance in Zupply using both the BN254 and BLS12-381 curves.

7.4.2 Aurora

In the first implementation variant of the Zupply framework, we employ Aurora [35]. It is a zkSNARK protocol that offers transparent-setup, which eliminates the necessity for a trusted parties in the setup phase. Aurora is built upon particularly the Interactive Oracle Proofs (IOPs) [36] for R1CS obtained by the Fast Reed-Solomon Interactive Oracle Proof of Proximity (FRI) protocol for low degree testing [26]. Aurora notably avoids dependence on hard mathematical problems such as discrete logarithms or integer factorization, which makes Aurora plausibly post-quantum secure. Aurora relies on minimal assumptions as it requires only a collision-resistant hash function. This makes it a suitable choice for transitioning the Zupply framework toward enhanced post-quantum security.

Since the R1CS constraints generated for the Zupply framework in the previous implementation have their elements in the scalar field associated with the chosen elliptic curve, we used the Aurora implementation over the same finite fields and on multiplicative groups. But, Aurora can also be used over binary extension fields and additive groups, which can make its computations faster due to simpler arithmetic operations and reduced computational overhead. We chose the former configuration for compatibility with the first implementation.

7.5 Software Implementation

In this section, we provide a detailed overview of the C++ software implementation[1] for each variant discussed above. Additionally, we introduce a baseline model for comparison, which does not incorporate privacy preservation.

7.5.1 Baseline Model

To assess the cost overhead introduced by privacy preservation using ZKPs in the Zupply framework for a supply chain management (SCM) application, we propose a non-privacy-preserving baseline model that maintains the same functionality; specifically, (non-anonymous) authentication tokens for off-chain data uploading. In the baseline model's smart contract, a token owned by an entity e_i holds e_i's address (BPAddr^{e_i}) and is only transferable by e_i. Furthermore, each token holds a quantity, enabling the merging and dividing of tokens. The token owner e_i can use the private key corresponding to BPAddr^{e_i} to sign off-chain data records, as explained in ERC-4361 [58], or transfer the token.

7.5.2 Groth16 Based Implementation

For the Groth16 zkSNARK instantiation, we employed libsnark [32], a C++ library developed by SCIPR lab [1], which offers implementations of the Groth16 zkSNARK scheme for provers and verifiers. This library was instrumental in generating R1CS constraints for NP problems within our framework, achieved by implementing them as arithmetic circuits. Additionally, to integrate the Groth16 verification algorithm into $\mathcal{C}_Z$, we utilized the ZoKrates [73] toolbox. As previously discussed, we only provide the Zupply smart contract implementation using Groth16 over BN254.

[1]The implementation is available at `https://github.com/mtbadakhshan/zupply-zkp`

Table 7.2 presents the costs associated with executing transactions in both the Zupply framework and the baseline model on the Ethereum blockchain. At the time of deploying $\mathcal{C}_Z$, some one-time costs should be paid as it uploads the code, a 256-bit $\texttt{rt}$, and all verification keys except $\mathsf{vk}_{\mathsf{Auth}}$ to blockchain. vk sizes are listed in Table 7.1. Transactions $\texttt{tx}_{\mathsf{Trans}}$, $\texttt{tx}_{\mathsf{Merge}}$, and $\texttt{tx}_{\mathsf{Div}}$ execute $\mathsf{VerifyTX}$ on $\mathcal{C}_Z$ and contain a 256 bytes π, one or two 20×256 bits path (for $L = 20$), new 256-bit $\texttt{rt}$s, 256-bit $\texttt{cm}$s, and 256-bit $\texttt{eol}$s. While the deployment cost is a one-time expense, entities incur transaction fees each time they transfer products to the next entity.

We also implemented the zero-knowledge proving and verification algorithms of the Zupply framework using Groth16 over the BLS12-381 curve. Consequently, the elements in the R1CS constraint system are in the scalar prime field associated with the BLS12-381 curve.

7.5.3 Aurora Based Implementation

The implementation of the Aurora zkSNARK is provided in libiop [34], which utilizes the libff [37] library for finite field arithmetic. Similarly, libsnark employs an older version of libff for operations involving both finite fields and elliptic curves. To enable the simultaneous execution of prover and verifier algorithms in both zkSNARKs (i.e., Aurora and Groth16) for the R1CS constraint systems generated by the arithmetic circuits, we updated several lines of code in the libsnark library to ensure compatibility with the newer version of the libff library[2].

The current implementation of the Aurora zkSNARK in libiop [34] requires that the number of constraints in the R1CS be a power of two, and that the number of variables and public inputs be one less than a power of two. To meet these requirements, constraints and variables are padded with zeros as necessary. Since the prover time, proof size, and verifier time in the Aurora zkSNARK asymptotically depend on the number of constraints, the computational complexity and proof size for any arithmetic circuit encoded into R1CS effectively correspond to those of an R1CS with a number of constraints equal to the next power of two.

[2]The new version of the libsnark library is available at: https://github.com/mtbadakhshan/libsnark

Table 7.2: Comparison of the transaction costs for the baseline model and the $\mathcal{C}_Z$ smart contract based on Groth16 over BN254 curve and $L = 20$. The costs are in Gas and USD, based on ETH's value of USD 2,030.01 and gas price of 32 Gwei at the time of writing.

	Zupply		Baseline model	
Transaction	Gas	USD	Gas	USD
Deployment	3,088,611	\$200.63	902,355	\$58.62
$\texttt{tx}_{\mathsf{Init}}$	133,415	\$8.67	96,166	\$6.25
$\texttt{tx}_{\mathsf{Trans}}$	448,013	\$29.10	29,649	\$1.93
$\texttt{tx}_{\mathsf{Merge}}$	455,534	\$29.59	92,382	\$6.00
$\texttt{tx}_{\mathsf{Div}}$	518,701	\$33.69	168,740	\$10.96

7.6 Benchmark

The benchmarks were conducted on a system featuring an Intel® Core™ i7-4790 CPU at 3.60 GHz, using 4 physical cores and 8 threads. The system was equipped with 32 GB of DDR3–1333 RAM and ran Ubuntu 20.04 LTS. The CPU governor was set to *Performance* to ensure the CPU operated at its maximum clock speed throughout testing, thus minimizing fluctuations caused by power-saving mechanisms. We used the Google Benchmark library [93] to measure the average execution time for each MHT depth L and prover and verifier algorithms for each circuit.

We executed the prover and verifier algorithms in Aurora and Groth16 for each circuit in the Zupply framework, using varying numbers of layers in the MHT. Figure 7.2 shows the number of constraints generated for each circuit. As previously noted, the Aurora zkSNARK implementation in libiop [34] requires the number of constraints to be padded to the next power of two. Consequently, as shown in the figure, the number of constraints for each L in each circuit is increased to the next power of two. This padding introduces significant computational overhead for the prover and verifier algorithms and leads to an increase in proof size.

For each L in each circuit, the computation times of the prover algorithms were averaged over multiple executions using different sets of inputs (i.e., public inputs and auxiliary inputs) to generate proofs. These inputs were randomly generated through separate functions simulating the circuit, ensuring consistency with the R1CS of each circuit. Similarly, the computation times of the verifier algorithms were averaged over executions using different valid proofs generated by the prover algorithm given the aforementioned randomly generated inputs. The experiments for the prover and verifier algorithms in Groth16 were

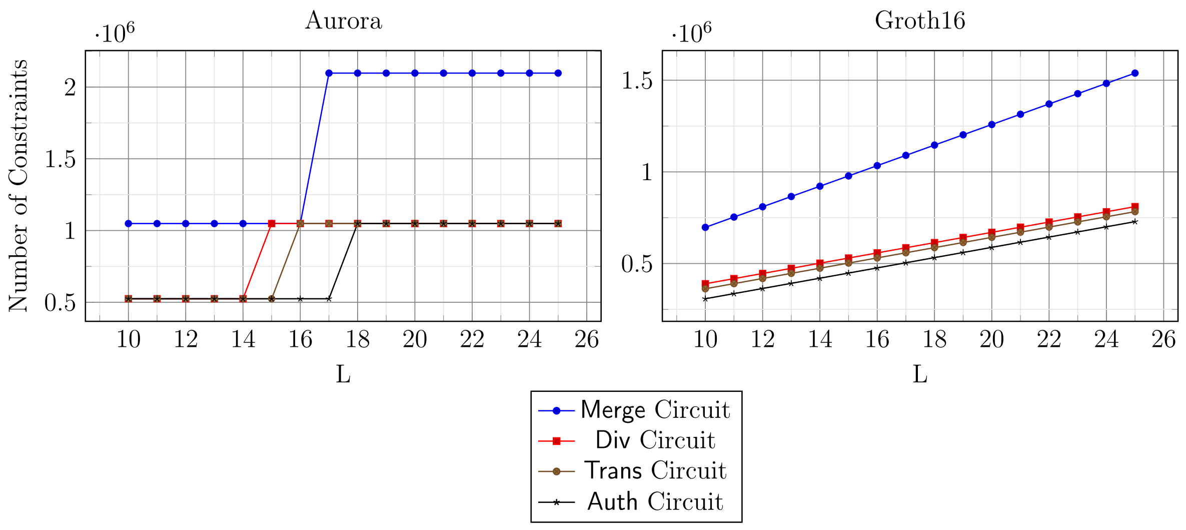


Figure 7.2: The number of constraints in the R1CS instances generated by arithmetic circuits in the Zupply framework for both Aurora [35] and Groth16 [95] zkSNARKs, as the Merkle hash tree depth L varies from 10 to 25.

conducted using both BN254 and BLS12-381 curves to evaluate the computational overhead introduced by the BLS12-381 curve, which achieves 128 bits of security at the cost of increased execution time for the algorithms. Likewise, the experiments for the prover and verifier algorithms in Aurora were executed over the scalar prime fields associated with the BN254 and BLS12-381 curves, with sizes of 254 and 255 bits, respectively. Figure 7.3 shows the execution times of the prover and verifier algorithms for both Groth16 and Aurora zkSNARKs.

Figure 7.4 illustrates the proof sizes for Aurora. The results are provided for each circuit in the Zupply framework, with the number of MHT layers ranging from 10 to 25. Note that the proof size for Groth16 is independent of the circuit; it is 128 bytes for the BN254 curve and 192 bytes for the BLS12-381 curve. As discussed in Section 7.5.3, the Aurora implementation in Libiop [34] increases the prover and verifier computation times and proof sizes to match those of a circuit where the number of constraints, public inputs, and variables are equal to the smallest power of two that exceeds the actual values in the circuit.

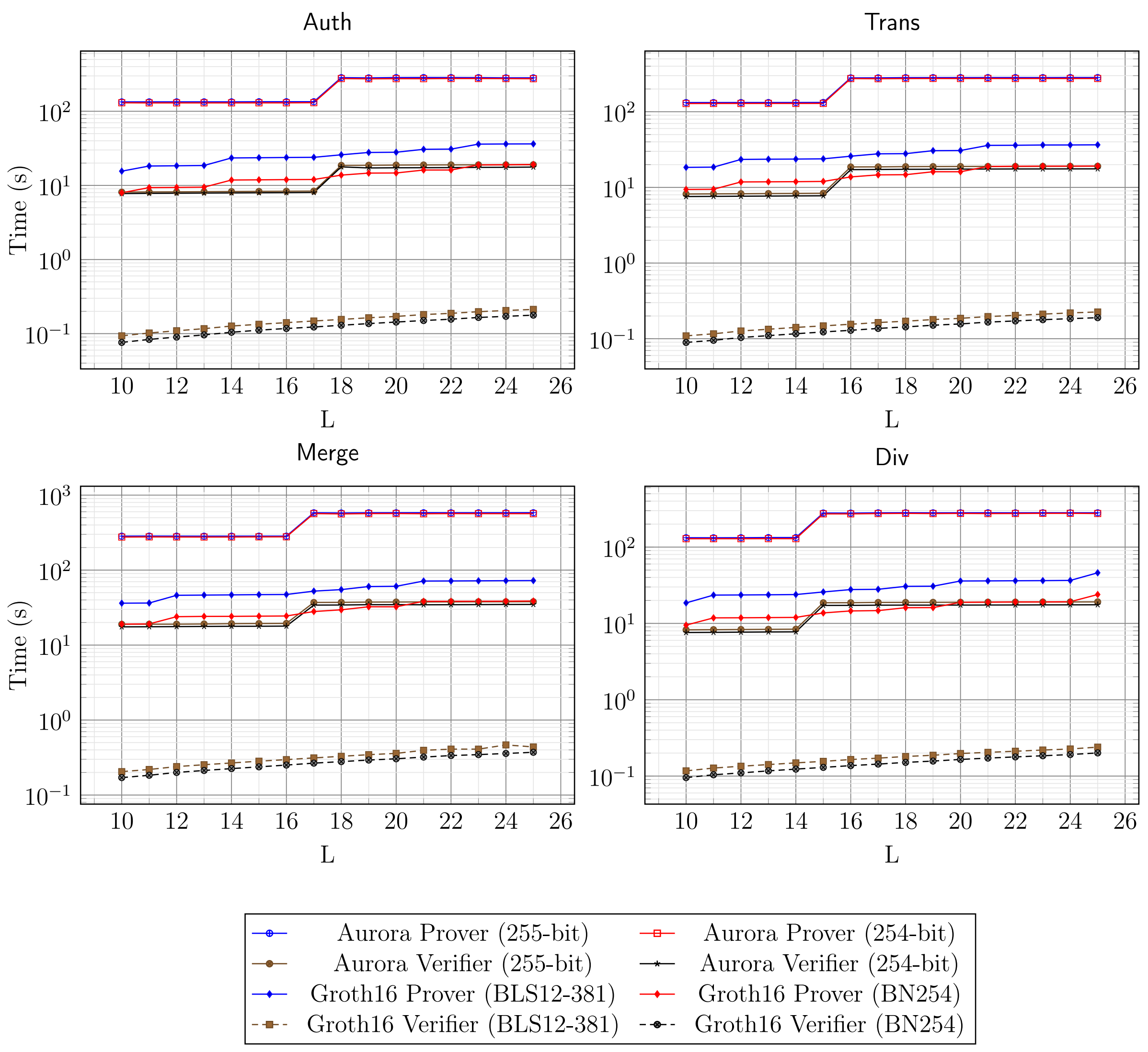


Figure 7.3: Prover and Verifier for Groth16 and Aurora in Zupply. Groth16 is evaluated on BN254 and BLS12-381, while Aurora uses their 254-bit and 255-bit scalar fields. L denotes MHT layers.

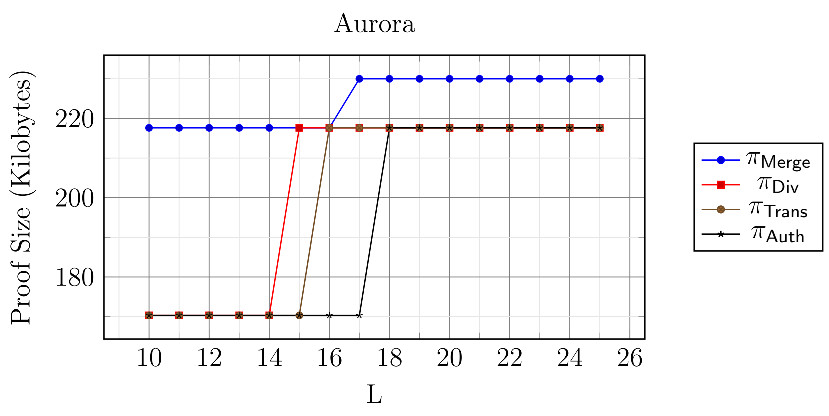


Figure 7.4: Proof sizes of Aurora. The proof sizes for 255-bit and 254-bit prime fields are identical, as both require 4×64-bit limbs on a 64-bit architecture. L denotes MHT layers. Groth16 proof sizes are 128 bytes (BN254) and 192 bytes (BLS12-381).

7.7 Performance Analysis

Figure 7.3 demonstrates that operating over BLS12-381 significantly increases the execution times of Groth16 algorithms. This disparity arises because Groth16 algorithms rely heavily on bilinear pairings over elliptic curves. Pairing operations on the BLS12-381 curve are more computationally intensive than those on BN254, primarily due to BLS12-381's larger base field size of 381 bits compared to BN254's 254 bits. However, the scalar field sizes of the two curves are nearly identical, with BN254 and BLS12-381 having scalar fields of 254 and 255 bits, respectively.

The results clearly show that adopting Aurora into the Zupply framework adds a significant overhead to the runtime of the algorithms. This illustrates the importance of both theoretical and implementation optimizations for current post-quantum transparent setup zkSNARKs and the arithmetic circuits of Zupply. Aurora and many other post-quantum transparent setup zkSNARKs, such as STARK [27] and Fractal [66], heavily rely on the fast Fourier transform (FFT). Although there has been a great deal of research aimed at optimizing these algorithms, using special basis elements (e.g., the Cantor special basis) to construct evaluation domains in these zkSNARKs can still significantly reduce the runtime of additive FFT algorithms. On the other hand, the arithmetic circuits can be optimized by replacing the standard cryptographic hash function we used (i.e., SHA-256) with algebraic hash functions such as Poseidon [94], which require fewer R1CS constraints.

Entities handling product histories must store all proving keys ($\mathsf{pk}_{\mathbb{x}}$, see Table 7.1) and a MHT of 32 MB for $L = 20$. This requires significant storage due to numerous constraints, but it is manageable even on smartphones. In contrast, auditors, like final customers, only need to store $\mathsf{vk}_{\mathsf{Auth}}$ (less than 1 KB) and audit only the relevant records.

7.8 Evaluation

Zupply employs off-chain data storage while anonymously authenticating the data, which is a significant advantage considering the high costs associated with on-chain storage[3]. None of the studied related works could provide anonymous authenticated off-chain storage. On-chain storage forces all full nodes in the blockchain platform to maintain a replication of each data record, which is why it is very expensive.

Moreover, the separation of storage and blockchain enhances anonymity, even when an adversary's access to $\mathcal{O}^{\mathsf{Attr}}$, is assumed. In other words, Zupply employs a more restrictive adversary model in which the underlying blockchain platform (e.g., Ethereum) does not preserve the anonymity of the transaction issuer. Then, if a data record is uploaded to the blockchain platform, the identity of the data uploader will be linked to the data record. However, this adversary's assumption is not considered in any of the previous works. For example, Mesh [8] and zkLedger [137], which are deployed over public blockchains, are prone to this attack.

To achive unlinkability among entities involved in a token transfer, we proposed an anonymous authentication token ownership transfer algorithm (AATOT), presented in Section 6.3.2. zkLedger requires involving the entire set of entities (including non-participating ones) in each transactions, which has a negative impact on its scalability. zkLedger commits to a value of zero for non-participating entities to obscure the link between the entities involved in the transaction. This increases the size of transactions in zkLedger by increasing the number of participants. In contrast, the size of transactions in Zupply is independent of the number of entities. Mesh requires the first entity in a supply chain to send a list of the accounts of the involved entities in the supply chain to a smart contract, while the list is sorted by the order in which the products are going to be owned. Therefore, it cannot provide unlinkability. Moreover, since adding a new entity to the supply chain list in the Mesh framework requires a new setup, Mesh cannot provide a good scalability.

DECOUPLES [129] proposes a layer-1 blockchain where multilayered linkable spontaneous anonymous group (MSLAG) ring signatures and stealth addresses are used to provide

[3]For example, storing a 1 MB data record on the Ethereum blockchain currently costs over USD 38,000 based on ETH's value at the time of writing.

Table 7.3: Comparison of security properties and costs among Zupply, related works, and baseline model. ●, ◖, and ○ indicate that the framework satisfies, partially satisfies, and does not satisfy a security property, respectively. ◆, ▸, and ◇ indicate that the framework has high, medium, and low cost, respectively.

Framework	Security Properties						Costs				
							Storage				
	Anonymity	AATOT*	Unlinkability	Integrity	Decentralization	No Central Server	On-chain	Entity Software	Transfer (on-chain)	Audit (off-chain)	Scalability
Mesh [8]	◖	N/A	○	●	◖	○	◆	◇	N/A	N/A	◆
DECOUPLES [129]	●	N/A	●	●	◖	●	◆	◇	◇	N/A	◇
zkLedger [137]	◖	N/A	●	●	●	●	◆	◇	▸	◆	◆
Zupply	●	●	●	●	●	●	◇	▸	◇	◇	◇
Baseline model	○	N/A	○	●	●	●	◇	◇	◇	◇	◇

*Anonymous Authentication Token Ownership Transfer

full anonymity and unlinkability. However, DECOUPLES is not fully decentralized (trustless) because it requires the mass adoption of the protocol among a large number of miners. However, Zupply does not require any changes to the layer-1 blockchain platform and is fully applicable to existing public blockchains. This enables it to facilitate the complete decentralization of current public blockchains. Additionally, although Mesh also operates over decentralized public blockchains, it requires a centralized Mesh server, which makes the protocol not fully trustless.

In Zupply, entities who are actively involved in a supply chain (i.e., those who may possess a product and upload product histories) should retain all proving keys (i.e., pk_x) in their local storage, with sizes as reported in Table 7.1. Additionally, they must maintain MHT, which occupies 32 MB if it contains 20 layers. Accordingly, entities in Zupply need extensive storage due to the R1CS relations related to the NP statements of Zupply, which have a large number of constraints. However, providing the required storage is feasible even on smartphones. In contrast, entities auditing the history of data records, such as final customers, only need to maintain $\mathsf{vk}_{\mathsf{Auth}}$, less than a kilobyte in size, and audit data records relevant to their product. However, within the zkLedger framework, a customer (likely acting as an offline auditor) must verify all transactions, both related and unrelated to the product, to be able to audit the product history. A transaction in zkLedger is analogous to a data record in the Zupply ecosystem. Finally, Table 7.3 summarizes the comparison between the frameworks studied in this research.

7.9 Summary

In summary, this chapter presented the designs of four arithmetic circuits, Auth, Trans, Merge, and Div, employed in the Zupply framework. It also introduced two implementations of this framework and compared their performance. The first implementation utilizes Groth16 [95] over two elliptic curves, the BN254 curve [23] that is computationally efficient but limited to a security level of 100 bits [20] and the BLS12-381 curve [22] that achieves a 128-bit security level, as recommended by NIST even for post-2030 applications in the absence of quantum computers [21]. Figure 7.3 shows the computational overhead enforced by using the BLS12-381 curve. The reported costs in Table 7.2 underscore the practicality of the Zupply framework using Groth16 over BN254.

Our framework extends beyond Groth16, accommodating other zkSNARK instantiations. Notably, Aurora [35] is transparent setup and offers potential post-quantum security. The second implementation replaced the Groth16 zkSNARK protocol with Aurora, which not only is secure against quantum-enabled malicious provers attempting to forge invalid proofs but also enables the framework to operate without a trusted setup, whether at initialization or during updates. Updates may include increasing the number of MHT layers or modifying the structure of anonymous authentication tokens. Figures 7.3 compares the computation times of the prover and verifier algorithms of Groth16 and Aurora, given the arithmetic circuits, and Figure 7.4 shows shows the proof sizes in Aurora. As the results indicate, the Aurora zkSNARK introduces a significant overhead in the runtime of the algorithms within the Zupply framework. This highlights the importance of optimizing algorithms for post-quantum secure zkSNARKs, such as Aurora. Therefore, the next two chapters focus on accelerating post-quantum secure zkSNARKs.

Chapter 8

Conclusions and Future Works

8.1 Concluding Remarks

This thesis conducted a comprehensive study on optimizing and enhancing the performance of post-quantum secure zero-knowledge succinct non-interactive arguments of knowledge (zkSNARKs). As zkSNARKs play an increasingly vital role in privacy-preserving applications, blockchain scalability, and cryptographic proofs of data integrity, improving their efficiency is paramount. A central focus was placed on additive Fast Fourier Transform (FFT) algorithms, which are foundational to post-quantum secure zkSNARK constructions. The thesis also introduced Zupply, a proof-of-concept framework that demonstrates the practical use of zkSNARK protocols for building privacy-preserving, decentralized applications on public blockchain platforms.

Chapters 1, 2, and 3 provided the introduction and background necessary to understand zero-knowledge proofs (ZKPs), zkSNARKs, and explored the design of additive FFT algorithms. As a specific application of zkSNARKs, Chapter 3 also examined existing challenges and current solutions in supply chain management (SCM). It highlighted the critical need for a privacy-preserving framework to support SCM systems deployed over permissionless blockchains.

Chapter 4 demonstrated the effectiveness of leveraging the Cantor special basis [56] to enable the use of the Cantor additive FFT algorithm in post-quantum secure zkSNARKs, with a particular focus on the Aurora protocol [35]. The implementation showed that replacing the Gao-Mateer FFT [83] with the Cantor FFT resulted in a substantial reduction in computation time. In addition, this Chapter presented a detailed theoretical analysis of the Cantor FFT's computational cost, including exact counts of additions and multiplications. It also evaluated the FFT call complexity arising during the encoding of the rank 1 constraint system (R1CS) in Aurora, with respect to the number of constraints, variables, and the chosen security parameter. Additionally, this chapter introduced optimized building blocks for the Cantor FFT implementation and proposed precomputation techniques that reduced overhead in both the Cantor and Gao-Mateer FFT algorithms when the basis of the affine subspaces was predetermined.

Chapter 5 presented an instantiation of the fast reed-solomon interactive oracle proofs of proximity (FRI) protocol [26] that eliminated field inversion operations in both the Commit and Query phases, contributing to improved efficiency. It also introduced a tailored instantiation of the GKR circuit designed to minimize the number of gates, with the goal of reducing communication overhead as well as the computational complexities of both the verifier and the prover.

Chapter 6 presented the Zupply framework, which offers an anonymous, unlinkable, trustless, fully decentralized, and efficient solution for managing off-chain, directed acyclic graph (DAG) structured data in supply chains. The framework introduced an anonymous authentication token (AAT) scheme, including the AAT ownership transfer AATOT protocol, realized through the OT-Protocol, to ensure unlinkability during ownership transfers. Its efficiency is achieved via the MHT-Protocol, which updates the Merkle hash tree (MHT) root on the contract $\mathcal{C}_Z$, eliminating the need to store the entire tree on-chain. By supporting off-chain data storage with anonymous authentication, the framework addresses the high cost of on-chain storage[1]. This flexibility enables customizable storage strategies while preserving user anonymity.

Chapter 7 presented the implementation of the Zupply framework. The framework was implemented in C++ and Solidity, initially using the Groth16 zkSNARK [95] over the BN254 curve [23]. This implementation demonstrated efficiency despite the high gas costs on Ethereum, although its security level is limited to approximately 100 bits [20]. To improve security, the BLS12-381 curve [22] was also used, offering 128-bit security. Furthermore, replacing the Groth16 zkSNARK with Aurora [35] enhanced the framework by providing plausible post-quantum security against quantum-enabled malicious provers attempting to forge invalid proofs. It also enabled the system to operate without requiring a trusted setup, either at initialization or during future updates. Such updates may include increasing the number of MHT layers or modifying the structure of AATs.

[1]For example, storing 1 MB of data on Ethereum exceeds USD 38,000 at current ETH prices.

8.2 Future Works

Looking ahead, several promising research directions emerge. Building on the results and optimizations presented in the Chapter 4, several promising directions for future research can be explored. One such direction is to investigate the LCH additive FFT [122], including its basis conversion in conjunction with the Cantor special basis, and explore optimizations for other components of Aurora, such as the FRI protocol [26]. Another avenue could involve applying tower field constructions, or using FFT algorithms to accelerate field multiplications, where the `CLMUL` instuction is not availabe on CPUs. Moreover extending the additive FFT optimizations to support parallelism on hardware accelerators (e.g., GPUs and FPGAs) could further boost performance. Additionally, the proposed optimizations may also be applied to other post-quantum secure zkSNARKs operating over binary extension fields, such as Fractal [66], STARK [27], Ligero [9], etc. Moreover, integrating the proposed optimizations into emerging zkVMs and rollup architectures, such as [162, 146, 187] will be key to scaling computation in future decentralized applications. Also, this can be used in post-quantum secure digital signature algorithms such as Preon [62].

In future work, a full implementation of the Polaris zkSNARK, accompanied by comprehensive performance benchmarks and an evaluation of the expected improvements, will be presented as part of the complete realization of the Polaris protocol. Future work can investigate alternative Reed–Solomon (RS) proximity testing that have less query complexity, such as STIR [12], or faster verification algorithm, such as WHIR [13].

The Zupply framework presented in Chapters 6 and 7, enabled the maintenance of authenticated DAG-structured data, with a focus on supply chain management (SCMs). However, DAGs are widely used in various domains where representing data flow, dependencies, and sequential relationships is essential. The DAG structure facilitates mapping the progression between different stages while preserving their ordering and interdependencies. A notable example is version control systems (VCSs), such as Git [86] and Mercurial [133], which leverage DAGs to represent the evolution history of a project through its commits. Extending the Zupply framework to support such applications remains a promising direction for future work.

Moreover, the arithmetic circuits used in the Zupply framework can be optimized by replacing the standard cryptographic hash function we used (i.e., SHA-256) with algebraic hash functions such as Poseidon [94], which require fewer R1CS constraints. Moreover, the arithmetic circuits can be instantiated over smaller fields while running the zkSNARK algorithms over larger fields by leveraging the methods proposed in [70] and [92]. This approach aims to achieve both computational efficiency and enhanced security with higher bit levels.

Future work can also explore alternative decentralized cloud storage (DCS) platforms, such as PriCloud [116], which provides enhanced user privacy and stronger unlinkability between users and their stored files. However, it is important to consider that different DCS platforms may adopt varying standards for content addressing. Additionally, the IPFS protocol [38] does not inherently guarantee data availability or persistence unless nodes are explicitly pinned and maintained. Therefore, investigating DCS alternatives that offer more robust and persistent storage solutions presents another promising direction for future research.

References

[1] Succinct computational integrity and privacy research (scipr) lab. URL: http://www.scipr-lab.org/.

[2] Tracr – De Beers Group. URL: https://www.debeersgroup.com/sustainability-and-ethics/leading-ethical-practices-across-the-industry/tracr.

[3] Pricing - IBM supply chain intelligence suite, 2023. URL: http://web.archive.org/web/20240614111258/https://www.ibm.com/products/supply-chain-intelligence-suite/pricing.

[4] Andrew Ackerman. U.S. appellate court faults SEC's 'conflict minerals' rule - WSJ, 2014. URL: https://www.wsj.com/articles/SB10001424052702303887804579501590809000188.

[5] Andrew Agapiou. Overcoming the legal barriers to the implementation of smart contracts in the construction industry: The emergence of a practice and research agenda. *Buildings*, 13(3), 2023. doi:10.3390/buildings13030594.

[6] Satyabrata Aich, Sabyasachi Chakraborty, Mangal Sain, Hye-in Lee, and Hee-Cheol Kim. A review on benefits of iot integrated blockchain based supply chain management implementations across different sectors with case study. In *2019 21st International Conference on Advanced Communication Technology (ICACT)*, pages 138–141, 2019. doi:10.23919/ICACT.2019.8701910.

[7] Basim Aljabhan and Muath A. Obaidat. Privacy-preserving blockchain framework for supply chain management: Perceptive craving game search optimization (PCGSO). *Sustainability*, 15(8), 2023. doi:10.3390/su15086905.

[8] Riham AlTawy and Guang Gong. Mesh: A supply chain solution with locally private blockchain transactions. *Proceedings on Privacy Enhancing Technologies*, 3:149–169, 2019. doi:10.2478/popets-2019-0041.

[9] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 2087–2104, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3133956.3134104.

[10] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, Srinivasan Muralidharan, Chet Murthy, Binh Nguyen, Manish Sethi, Gari Singh, Keith Smith, Alessandro Sorniotti, Chrysoula Stathakopoulou, Marko Vukolić, Sharon Weed Cocco, and Jason Yellick. Hyperledger Fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the Thirteenth EuroSys Conference*, EuroSys '18, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3190508.3190538.

[11] Richard J. Arend and Joel D. Wisner. Small business and supply chain management: is there a fit? *Journal of Business Venturing*, 20(3):403–436, 2005. doi:10.1016/j.jbusvent.2003.11.003.

[12] Gal Arnon, Alessandro Chiesa, Giacomo Fenzi, and Eylon Yogev. STIR: Reed-Solomon proximity testing with fewer queries. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology – CRYPTO 2024*, pages 380–413, Cham, 2024. Springer Nature Switzerland. doi:10.1007/978-3-031-68403-6_12.

[13] Gal Arnon, Alessandro Chiesa, Giacomo Fenzi, and Eylon Yogev. WHIR: Reed–Solomon proximity testing with super-fast verification. Cryptology ePrint Archive, Paper 2024/1586, 2024. URL: https://eprint.iacr.org/2024/1586.

[14] Arasu Arun, Srinath Setty, and Justin Thaler. Jolt: SNARKs for virtual machines via lookups. In *Advances in Cryptology – EUROCRYPT 2024: 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 26–30, 2024, Proceedings, Part VI*, page 3–33, Berlin, Heidelberg, 2024. Springer-Verlag. doi:10.1007/978-3-031-58751-1_1.

[15] László Babai. Trading group theory for randomness. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, STOC '85, page 421–429, New

York, NY, USA, 1985. Association for Computing Machinery. doi:10.1145/22145.22192.

[16] Mohammadtaghi Badakhshan and Guang Gong. Privacy-preserving ownership transfer: Challenges and an outlined solution based on zero-knowledge proofs. In *2023 IEEE 9th World Forum on Internet of Things (WF-IoT)*, pages 1–9, 2023. doi:10.1109/WF-IoT58464.2023.10539396.

[17] Mohammadtaghi Badakhshan and Guang Gong. Zupply: Anonymously maintained decentralized dag data record over public blockchains. June 2024. doi:10.36227/techrxiv.171821776.63453406/v1.

[18] Mohammadtaghi Badakhshan and Guang Gong. Evaluating the integration of Aurora zkSNARK in the Zupply framework. In *2025 IEEE/ACM 6th International Workshop on Engineering and Cybersecurity of Critical Systems (EnCyCriS)*, 2025.

[19] Mohammadtaghi Badakhshan, Guiwen Luo, Tanmayi Jandhyala, and Guang Gong. Ursa Minor: The implementation framework for Polaris. In Svetla Petkova-Nikova and Daniel Panario, editors, *Arithmetic of Finite Fields*, pages 258–273, Cham, 2025. Springer Nature Switzerland. doi:10.1007/978-3-031-81824-0_16.

[20] Razvan Barbulescu and Sylvain Duquesne. Updating key size estimations for pairings. *Journal of Cryptology*, 32(4):1298–1336, 2019. doi:10.1007/s00145-018-9280-5.

[21] Elaine Barker. Recommendation for Key Management: Part 1 – General (NIST Special Publication 800-57 Part 1 Revision 5). Technical Report SP 800-57 Part 1 Rev. 5, National Institute of Standards and Technology, May 2020. URL: https://csrc.nist.gov/pubs/sp/800/57/pt1/r5/final, doi:10.6028/NIST.SP.800-57pt1r5.

[22] Paulo S. L. M. Barreto, Ben Lynn, and Michael Scott. Constructing elliptic curves with prescribed embedding degrees. In Stelvio Cimato, Giuseppe Persiano, and Clemente Galdi, editors, *Security in Communication Networks*, pages 257–267, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. doi:10.1007/3-540-36413-7_19.

[23] Paulo S. L. M. Barreto and Michael Naehrig. Pairing-friendly elliptic curves of prime order. In Bart Preneel and Stafford Tavares, editors, *Selected Areas in Cryptography*, pages 319–331, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. doi:10.1007/11693383_22.

[24] Carsten Baum, Lennart Braun, Cyprien Delpech de Saint Guilhem, Michael Klooß, Emmanuela Orsini, Lawrence Roy, and Peter Scholl. Publicly verifiable zero-knowledge and post-quantum signatures from VOLE-in-the-head. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023*, pages 581–615, Cham, 2023. Springer Nature Switzerland. doi:10.1007/978-3-031-38554-4_19.

[25] Mihir Bellare and Oded Goldreich. On defining proofs of knowledge. In Ernest F. Brickell, editor, *Advances in Cryptology — CRYPTO' 92*, volume 740, pages 390–420, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg. doi:10.1007/3-540-48071-4_28.

[26] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Fast Reed-Solomon interactive oracle proofs of proximity. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*, volume 107 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 14:1–14:17, Dagstuhl, Germany, 2018. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2018.14.

[27] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Paper 2018/046, 2018. URL: https://eprint.iacr.org/2018/046.

[28] Eli Ben-Sasson, Dan Carmon, Swastik Kopparty, and David Levit. Scalable and transparent proofs over all large fields, via elliptic curves. In Eike Kiltz and Vinod Vaikuntanathan, editors, *Theory of Cryptography*, pages 467–496, Cham, 2022. Springer Nature Switzerland. doi:10.1007/978-3-031-22318-1_17.

[29] Eli Ben-Sasson, Dan Carmon, Swastik Kopparty, and David Levit. Elliptic curve fast Fourier transform (ECFFT) part i: Low-degree extension in time O(n log n) over all finite fields. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 700–737, 2023. doi:10.1137/1.9781611977554.ch30.

[30] Eli Ben-Sasson, Alessandro Chiesa, Ariel Gabizon, and Madars Virza. Quasi-linear size zero knowledge from linear-algebraic PCPs. In Eyal Kushilevitz and Tal Malkin, editors, *Theory of Cryptography*, pages 33–64, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg. doi:10.1007/978-3-662-49099-0_2.

[31] 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, 2014. doi:10.1109/SP.2014.36.

[32] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Shaul Kfir, Eran Tromer, Madars Virza, and Howard Wu. Github - scipr-lab/libsnark: C++ library for zksnarks, 2020. URL: https://github.com/scipr-lab/libsnark.

[33] 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, 2015. doi:10.1109/SP.2015.25.

[34] Eli Ben-Sasson, Alessandro Chiesa, Alex Kazorian, Dev Ojha, Aleksejs Popovs, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas Ward. libiop: A C++ library for zero knowledge proofs. https://github.com/scipr-lab/libiop.

[35] 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, *Advances in Cryptology – EUROCRYPT 2019*, pages 103–128, Cham, 2019. Springer International Publishing. doi:10.1007/978-3-030-17653-2_4.

[36] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. In Martin Hirt and Adam Smith, editors, *Theory of Cryptography*, pages 31–60, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg. doi:10.1007/978-3-662-53644-5_2.

[37] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, Madars Virza, and Howard Wu. Github - scipr-lab/libff: C++ library for finite fields and elliptic curves, 2021. URL: https://github.com/scipr-lab/libff.

[38] Juan Benet. IPFS - content addressed, versioned, P2P file system [white paper], 7 2014. arXiv:1407.3561.

[39] Juan Benet and Steven Allen. Multihash: Self describing hashes - for future proofing, 2023. URL: https://github.com/multiformats/multihash.

[40] Juan Benet, David Dalrymple, and Nicola Greco. Proof of replication. *Protocol Labs, July*, 27:20, 2017. Accessed: 2024-11-14. URL: https://filecoin.io/proof-of-replication.pdf.

[41] Daniel J. Bernstein and Tung Chou. Faster binary-field multiplication and faster binary-field MACs. In Antoine Joux and Amr Youssef, editors, *Selected Areas in Cryptography – SAC 2014*, pages 92–111, Cham, 2014. Springer International Publishing. doi:10.1007/978-3-319-13051-4_6.

[42] Daniel J. Bernstein, Tung Chou, and Peter Schwabe. McBits: Fast constant-time code-based cryptography. In Guido Bertoni and Jean-Sébastien Coron, editors, *Cryptographic Hardware and Embedded Systems - CHES 2013*, pages 250–272, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. doi:10.1007/978-3-642-40349-1_15.

[43] Ye Bi, Kai Fan, Kuan Zhang, Yuhan Bai, Hui Li, and Yingtang Yang. A secure and efficient two-party protocol enabling ownership transfer of RFID objects. *IEEE Internet of Things Journal*, 10(18):16225–16237, 2023. doi:10.1109/JIOT.2023.3267501.

[44] Mojtaba Bisheh-Niasar, Reza Azarderakhsh, and Mehran Mozaffari-Kermani. High-speed NTT-based polynomial multiplication accelerator for post-quantum cryptography. In *2021 IEEE 28th Symposium on Computer Arithmetic (ARITH)*, pages 94–101, 2021. doi:10.1109/ARITH51176.2021.00028.

[45] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, page 326–349, New York, NY, USA, 2012. Association for Computing Machinery. doi:10.1145/2090236.2090263.

[46] Leo I. Bluestein. A linear filtering approach to the computation of discrete Fourier transform. *IEEE Transactions on Audio and Electroacoustics*, 18(4):451–455, 1970. doi:10.1109/TAU.1970.1162132.

[47] Alin Bostan and Éric Schost. Polynomial evaluation and interpolation on special sets of points. *Journal of Complexity*, 21(4):420–446, 2005. Festschrift for the 70th Birthday of Arnold Schonhage. doi:10.1016/j.jco.2004.09.009.

[48] Sean Bowe, Alessandro Chiesa, Matthew Green, Ian Miers, Pratyush Mishra, and Howard Wu. ZEXE: Enabling decentralized private computation. In *2020 IEEE Sym-*

posium on Security and Privacy (SP), pages 947–964, 2020. doi:10.1109/SP40000.2020.00050.

[49] Gilles Brassard, David Chaum, and Claude Crépeau. Minimum disclosure proofs of knowledge. *Journal of Computer and System Sciences*, 37(2):156–189, 1988. doi:10.1016/0022-0000(88)90005-0.

[50] Gilles Brassard and Claude Crépeau. Zero-knowledge simulation of boolean circuits. In Andrew M. Odlyzko, editor, *Advances in Cryptology — CRYPTO' 86*, volume 263, pages 223–233, Berlin, Heidelberg, 1986. Springer Berlin Heidelberg. doi:10.1007/3-540-47721-7_16.

[51] Jacqueline Brendel, Cas Cremers, Dennis Jackson, and Mang Zhao. The provable security of Ed25519: Theory and practice. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1659–1676, 2021. doi:10.1109/SP40001.2021.00042.

[52] D Brown. Standards for efficient cryptography, sec 1: elliptic curve cryptography. *Released Standard Version*, 1, 2009. URL: https://www.secg.org/SEC1-Ver-1.0.pdf.

[53] Benedikt Bunz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 315–334, Los Alamitos, CA, USA, May 2018. IEEE Computer Society. doi:10.1109/SP.2018.00020.

[54] Vitalik Buterin. Ethereum: A next-generation smart contract and decentralized application platform [white paper], 2014. URL: https://ethereum.org/content/whitepaper/whitepaper-pdf/Ethereum_Whitepaper_-_Buterin_2014.pdf.

[55] Vitalik Buterin and Christian Reitwiessner. EIP-197: Precompiled contracts for optimal ate pairing check on the elliptic curve alt_bn128. *Ethereum Improvement Proposals*, (197), Feb 2017. URL: https://eips.ethereum.org/EIPS/eip-197.

[56] David G. Cantor. On arithmetical algorithms over finite fields. *Journal of Combinatorial Theory, Series A*, 50(2):285–300, 1989. doi:10.1016/0097-3165(89)90020-4.

[57] Stefanos Chaliasos, Itamar Reif, Adrià Torralba-Agell, Jens Ernstberger, Assimakis Kattis, and Benjamin Livshits. Analyzing and benchmarking zk-rollups. In Rainer Böhme and Lucianna Kiffer, editors, *6th Conference on Advances in Financial Technologies (AFT 2024)*, volume 316 of *Leibniz International Proceedings in Informatics*

(LIPIcs), pages 6:1–6:24, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.AFT.2024.6.

[58] Wayne Chang, Gregory Rocco, Brantly Millegan, Nick Johnson, and Oliver Terbu. Erc-4361. *Ethereum Improvement Proposals*, 10 2021. URL: https://eips.ethereum.org/EIPS/eip-4361.

[59] Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, and Greg Zaverucha. Post-quantum zero-knowledge and signatures from symmetric-key primitives. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 1825–1842, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3133956.3133997.

[60] Bing-Jyue Chen, Suppakit Waiwitlikhit, Ion Stoica, and Daniel Kang. ZKML: An optimizing system for ml inference in zero-knowledge proofs. In *Proceedings of the Nineteenth European Conference on Computer Systems*, EuroSys '24, page 560–574, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3627703.3650088.

[61] Lidong Chen and Guang Gong. *Communication System Security.* CRC press, 2012.

[62] Ming-Shing Chen, Yu-Shian Chen, Chen-Mou Cheng, Shiuan Fu, Wei-Chih Hong, Jen-Hsuan Hsiang, Sheng-Te Hu, Po-Chun Kuo, Wei-Bin Lee, Feng-Hao Liu, et al. Preon: zk-SNARK based signature scheme. *Technical report. NIST*, 2023. URL: https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/round-1/spec-files/Preon-spec-web.pdf.

[63] Ming-Shing Chen, Chen-Mou Cheng, Po-Chun Kuo, Wen-Ding Li, and Bo-Yin Yang. Faster multiplication for long binary polynomials. arXiv: 1708.09746, 2018. URL: https://arxiv.org/abs/1708.09746.

[64] Vanya Cherneva and Jerry L. Trahan. TP-OTP: Two-party, ownership transfer protocol for RFID tags based on quadratic residues. In *2021 IEEE Green Energy and Smart Systems Conference (IGESSC)*, pages 1–6, 2021. doi:10.1109/IGESSC53124.2021.9618683.

[65] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In *Advances in Cryptology – EUROCRYPT 2020: 39th Annual International*

Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part I, page 738–768, Berlin, Heidelberg, 2020. Springer-Verlag. doi:10.1007/978-3-030-45721-1_26.

[66] Alessandro Chiesa, Dev Ojha, and Nicholas Spooner. Fractal: Post-quantum and transparent recursive proofs from holography. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part I*, pages 769–793, Cham, 2020. Springer International Publishing. doi:10.1007/978-3-030-45721-1_27.

[67] James W. Cooley and John W. Tukey. An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation*, 19(90):297–301, 1965. doi:10.2307/2003354.

[68] Nicolas T. Courtois. and Rebekah Mercer. Stealth address and key management techniques in blockchain systems. In *Proceedings of the 3rd International Conference on Information Systems Security and Privacy - ICISSP*, pages 559–566. INSTICC, SciTePress, 2017. doi:10.5220/0006270005590566.

[69] Christopher W. Craighead, David J. Ketchen, and Jessica L. Darby. Pandemics and supply chain management research: Toward a theoretical toolbox. *Decision sciences: journal of innovative education*, 51:838–866, 8 2020. doi:10.1111/DECI.12468.

[70] Benjamin E. Diamond and Jim Posen. Succinct arguments over towers of binary fields. Cryptology ePrint Archive, Paper 2023/1784, 2023. URL: https://eprint.iacr.org/2023/1784.

[71] Jeffrey H. Dyer and Harbir Singh. The relational view: Cooperative strategy and sources of interorganizational competitive advantage. *Academy of Management Review*, 23(4):660–679, 1998. doi:10.5465/amr.1998.1255632.

[72] Stefan Dziembowski, Shahriar Ebrahimi, and Parisa Hassanizadeh. VIMz: Private proofs of image manipulation using folding-based zkSNARKs. In *Proceedings on Privacy Enhancing Technologies*, page 344–362, 2025. doi:10.56553/popets-2025-0065.

[73] Jacob Eberhardt and Stefan Tai. Zokrates - scalable privacy-preserving off-chain computations. In *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber,*

Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), pages 1084–1091, 2018. doi:10.1109/Cybermatics_2018.2018.00199.

[74] Youssef El Housni and Aurore Guillevic. Families of snark-friendly 2-chains of elliptic curves. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology – EUROCRYPT 2022*, pages 367–396, Cham, 2022. Springer International Publishing. doi:10.1007/978-3-031-07085-3_13.

[75] Ethereum Development Documentation. Patricia Merkle trees, 2023. URL: https://ethereum.org/en/developers/docs/data-structures-and-encoding/patricia-merkle-trie/.

[76] Hyperledger Fabric. Hyperledger fabric – hyperledger foundation, 2023. URL: https://www.hyperledger.org/use/fabric.

[77] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology — CRYPTO' 86*, pages 186–194, Berlin, Heidelberg, 1987. Springer Berlin Heidelberg. doi:10.1007/3-540-47721-7_12.

[78] N. J. Fine. Binomial Coefficients Modulo a Prime. *The American Mathematical Monthly*, 54(10):589–592, 1947. URL: http://www.jstor.org/stable/2304500, doi:10.2307/2304500.

[79] Hyperledger Foundation. DLT Labs™ & Walmart Canada transform freight invoice management with Hyperledger Fabric, 07 2023. URL: https://www.hyperledger.org/case-studies/dltlabs-case-study.

[80] Hyperledger Foundation. Walmart case study – hyperledger foundation, 2023. URL: https://www.hyperledger.org/learn/publications/walmart-case-study.

[81] Shihui Fu and Guang Gong. Polaris: Transparent succinct zero-knowledge arguments for R1CS with efficient verifier. *Proceedings on Privacy Enhancing Technologies*, 2022:544–564, 2022. doi:10.2478/POPETS-2022-0027.

[82] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Paper 2019/953, 2019. URL: https://eprint.iacr.org/2019/953.

[83] Shuhong Gao and Todd Mateer. Additive fast Fourier transforms over finite fields. *IEEE Transactions on Information Theory*, 56(12):6265–6272, 2010. doi:10.1109/TIT.2010.2079016.

[84] Joachim von zur Gathen and Jürgen Gerhard. Arithmetic and factorization of polynomial over $\mathbb{F}_2$ (extended abstract). In *Proceedings of the 1996 International Symposium on Symbolic and Algebraic Computation*, ISSAC '96, page 1–9, New York, NY, USA, 1996. Association for Computing Machinery. doi:10.1145/236869.236882.

[85] 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, *Advances in Cryptology – EUROCRYPT 2013*, pages 626–645, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. doi:10.1007/978-3-642-38348-9_37.

[86] Git. Git SCM, 2023. URL: https://git-scm.com/.

[87] Oded Goldreich. *Foundations of Cryptography: Volume 1*. Cambridge University Press, USA, 2001. doi:10.1017/CBO9780511546891.

[88] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity or all languages in np have zero-knowledge proof systems. *J. ACM*, 38(3):690–728, July 1991. doi:10.1145/116825.116852.

[89] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: interactive proofs for muggles. In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, STOC '08, page 113–122, New York, NY, USA, 2008. Association for Computing Machinery. doi:10.1145/1374376.1374396.

[90] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, STOC '85, page 291–304, New York, NY, USA, 1985. Association for Computing Machinery. doi:10.1145/22145.22178.

[91] Shafi Goldwasser and Michael Sipser. Private coins versus public coins in interactive proof systems. In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*, STOC '86, page 59–68, New York, NY, USA, 1986. Association for Computing Machinery. doi:10.1145/12130.12137.

[92] Guang Gong. Uni/multi variate polynomial embeddings for zkSNARKs. *Cryptography and Communications*, 16(6):1257–1288, 2024. doi:10.1007/s12095-024-00723-0.

[93] Google Inc. and contributors. Benchmark - a microbenchmark support library. https://github.com/google/benchmark.

[94] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A new hash function for Zero-Knowledge proof systems. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 519–535. USENIX Association, August 2021. URL: https://www.usenix.org/conference/usenixsecurity21/presentation/grassi.

[95] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016*, pages 305–26, Berlin, Heidelberg, 2016. Springer. doi:10.1007/978-3-662-49896-5_11.

[96] Core Specification Working Group. Bluetooth core specification 6.0. Accessed: 2025-03-23. URL: https://www.bluetooth.com/specifications/specs/core60-html/.

[97] Shay Gueron and Michael E Kounavis. Intel® carry-less multiplication instruction and its usage for computing the GCM mode. *White Paper*, 2014. URL: https://www.intel.com/content/dam/develop/external/us/en/documents/clmul-wp-rev-2-02-2014-04-20.pdf.

[98] Ruixue Guo, Hau L. Lee, and Robert Swinney. Responsible sourcing in supply chains. *Management Science*, 62(9):2722–2744, 2016. doi:10.1287/mnsc.2015.2256.

[99] Mathias Hall-Andersen, Mark Simkin, and Benedikt Wagner. FRIDA: Data availability sampling from FRI. In *Advances in Cryptology – CRYPTO 2024: 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2024, Proceedings, Part VI*, page 289–324, Berlin, Heidelberg, 2024. Springer-Verlag. doi:10.1007/978-3-031-68391-6_9.

[100] Houssein Hellani, Layth Sliman, Abed Ellatif Samhat, and Ernesto Exposito. On blockchain integration with supply chain: Overview on data transparency. *Logistics*, 5(3), 2021. doi:10.3390/logistics5030046.

[101] Petri Helo and Yuqiuge Hao. Blockchains in operations and supply chains: A model and reference implementation. *Computers & Industrial Engineering*, 136:242–251, 2019. doi:10.1016/j.cie.2019.07.023.

[102] Daira-Emma Hopwood, Sean Bowe, Jack Grigg, Kris Nuttycombe, Ying Tong Lai, and Steven Smith. Halo2 book. https://zcash.github.io/halo2/index.html, 2021.

[103] IBM. IBM supply chain intelligence suite - solution - Canada | IBM, 2023. URL: https://www.ibm.com/ca-en/products/supply-chain-intelligence-suite/food-trust.

[104] IFC. Definitions of targeted sectors. URL: https://www.ifc.org/en/what-we-do/sector-expertise/financial-institutions/definitions-of-targeted-sectors.

[105] Tanmayi Jandhyala. Air-FRI: Acceleration of the FRI protocol on the GPU for low-degree polynomial testing in zk-SNARK applications. Master's thesis, University of Waterloo, Canada, 2024. URL: https://uwspace.uwaterloo.ca/items/6cee69e5-c9cf-4ef2-ba59-ca3541b58f71.

[106] Simon Jentzsch and Christoph Jentzsch. Eip-1186. *Ethereum Improvement Proposals*, 6 2018. URL: https://eips.ethereum.org/EIPS/eip-1186.

[107] Zhuoran Ji, Zhiyuan Zhang, Jiming Xu, and Lei Ju. Accelerating multi-scalar multiplication for efficient zero knowledge proofs with multi-GPU systems. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, ASPLOS '24, page 57–70, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3620666.3651364.

[108] Harry Kalodner, Steven Goldfeder, Xiaoqi Chen, S. Matthew Weinberg, and Edward W. Felten. Arbitrum: Scalable, private smart contracts. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1353–1370, Baltimore, MD, August 2018. USENIX Association. URL: https://www.usenix.org/conference/usenixsecurity18/presentation/kalodner.

[109] Anatolii A. Karatsuba and Yuri P. Ofman. Multiplication of many-digital numbers by automatic computers. In *Doklady Akademii Nauk SSSR*, volume 145, pages 293–294. Russian Academy of Sciences, 1962. URL: https://www.mathnet.ru/eng/dan26729.

[110] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Masayuki Abe, editor, *Advances in Cryptology - ASIACRYPT 2010*, pages 177–194, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. doi:10.1007/978-3-642-17373-8_11.

[111] Jonathan Katz, Vladimir Kolesnikov, and Xiao Wang. Improved non-interactive zero knowledge with applications to post-quantum signatures. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 525–537, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3243734.3243805.

[112] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, Second Edition*. Chapman & Hall/CRC, 2nd edition, 2014.

[113] Jing Huey Khor, Michail Sidorov, and Seri Aathira Balqis Zulqarnain. Scalable lightweight protocol for interoperable public blockchain-based supply chain ownership management. *Sensors*, 23(7), 2023. doi:10.3390/s23073433.

[114] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing*, STOC '92, page 723–732, New York, NY, USA, 1992. Association for Computing Machinery. doi:10.1145/129712.129782.

[115] Taechan Kim and Razvan Barbulescu. Extended tower number field sieve: A new complexity for the medium prime case. In *Proceedings, Part I, of the 36th Annual International Cryptology Conference on Advances in Cryptology — CRYPTO 2016 - Volume 9814*, page 543–571, Berlin, Heidelberg, 2016. Springer-Verlag. doi:10.1007/978-3-662-53018-4_20.

[116] Henning Kopp, David Mödinger, Franz J. Hauck, and Frank Kargl. Cryptographic design of pricloud, a privacy-preserving decentralized storage with remuneration. *IEEE Transactions on Dependable and Secure Computing*, 18(4):1908–1919, 2021. doi:10.1109/TDSC.2019.2942300.

[117] Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 839–858, May 2016. doi:10.1109/SP.2016.55.

[118] Nir Kshetri. Blockchain systems and ethical sourcing in the mineral and metal industry: a multiple case study. *International Journal of Logistics Management*, 33:1–27, 2 2022. doi:10.1108/IJLM-02-2021-0108/FULL/XML.

[119] Adam Lechmere. Wine vault offers security in a digital age | wine-searcher news & features, 12 2016. URL: https://www.wine-searcher.com/m/2016/12/wine-vault-offers-security-in-a-digital-age.

[120] Junzheng Li, Zhenqi Wang, Shaopeng Guan, and Youliang Cao. ProChain: A privacy-preserving blockchain-based supply chain traceability system model. *Computers & Industrial Engineering*, 187:109831, 2024. doi:10.1016/j.cie.2023.109831.

[121] Wen-Ding Li, Ming-Shing Chen, Po-Chun Kuo, Chen-Mou Cheng, and Bo-Yin Yang. Frobenius additive fast Fourier transform. In *Proceedings of the 2018 ACM International Symposium on Symbolic and Algebraic Computation*, ISSAC '18, page 263–270, 2018. doi:10.1145/3208976.3208998.

[122] Sian-Jheng Lin, Tareq Y. Al-Naffouri, and Yunghsiang Sam Han. FFT algorithm for binary extension finite fields and its application to Reed–Solomon codes. *IEEE Transactions on Information Theory*, 62(10):5343–5358, 2016. doi:10.1109/TIT.2016.2600417.

[123] Sian-Jheng Lin, Tareq Y. Al-Naffouri, Yunghsiang Sam Han, and Wei-Ho Chung. Novel polynomial basis with fast Fourier transform and its application to Reed–Solomon erasure codes. *IEEE Transactions on Information Theory*, 62(11):6284–6299, 2016. doi:10.1109/TIT.2016.2608892.

[124] Sian-Jheng Lin, Wei-Ho Chung, and Yunghsiang S. Han. Novel polynomial basis and its application to Reed-Solomon erasure codes. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 316–325, 2014. doi:10.1109/FOCS.2014.41.

[125] Peter Loshin. *Practical anonymity: hiding in plain sight online.* Syngress, 2013. URL: https://www.oreilly.com/library/view/practical-anonymity/9780124104044/.

[126] Edouard Lucas. Théorie des fonctions numériques simplement périodiques. *American Journal of Mathematics*, 1(4):289–321, 1878. doi:10.2307/2369308.

[127] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *J. ACM*, 39(4):859–868, oct 1992. doi:10.1145/146585.146605.

[128] Guiwen Luo, Shihui Fu, and Guang Gong. Speeding up multi-scalar multiplication over fixed points towards efficient zkSNARKs. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2023(2):358–380, Mar. 2023. doi:10.46586/tches.v2023.i2.358-380.

[129] Mourad El Maouchi, Oğuzhan Ersoy, and Zekeriya Erkin. Decouples: A decentralized, unlinkable and privacy-preserving traceability system for the supply chain. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, SAC '19, page 364–373, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3297280.3297318.

[130] Todd Mateer. *Fast Fourier Transform Algorithms with Applications*. PhD thesis, Clemson University, 2008. URL: https://open.clemson.edu/all_dissertations/231/.

[131] Gregory Maxwell. Coinjoin: Bitcoin privacy for the real world, 2013. URL: https://bitcointalk.org/index.php?topic=279249.

[132] Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoy, Geoffrey M. Voelker, and Stefan Savage. A fistful of Bitcoins: Characterizing payments among men with no names. *Commun. ACM*, 59(4):86–93, Mar 2016. doi:10.1145/2896384.

[133] Mercurial. Mercurial SCM, 2023. URL: https://www.mercurial-scm.org/.

[134] Ralph C. Merkle. Protocols for public key cryptosystems. In *1980 IEEE Symposium on Security and Privacy*, pages 122–122, 1980. doi:10.1109/SP.1980.10006.

[135] Ahmad Musamih, Khaled Salah, Raja Jayaraman, Junaid Arshad, Mazin Debe, Yousof Al-Hammadi, and Samer Ellahham. A blockchain-based approach for drug traceability in healthcare supply chain. *IEEE Access*, 9:9728–9743, 2021. doi:10.1109/ACCESS.2021.3049920.

[136] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. Accessed: 2025-03-23. URL: https://bitcoin.org/bitcoin.pdf.

[137] Neha Narula, Willy Vasquez, and Madars Virza. zkLedger: Privacy-Preserving auditing for distributed ledgers. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 65–80, Renton, WA, April 2018. USENIX Association. URL: https://www.usenix.org/conference/nsdi18/presentation/narula.

[138] National Institute of Standards and Technology. Post-Quantum Cryptography: Standardization Process. https://csrc.nist.gov/projects/pqc-dig-sig/round-1-additional-signatures, 2023.

[139] Ray Neiheiser, Gustavo Inácio, Luciana Rech, Carlos Montez, Miguel Matos, and Luís Rodrigues. Practical limitations of ethereum's layer-2. *IEEE Access*, 11:8651–8662, 2023. doi:10.1109/ACCESS.2023.3237897.

[140] Valeria Nikolaenko, Sam Ragsdale, Joseph Bonneau, and Dan Boneh. Powers-of-tau to people: Decentralizing setup ceremonies. In *Applied Cryptography and Network Security: 22nd International Conference, ACNS 2024, Abu Dhabi, United Arab Emirates, March 5–8, 2024, Proceedings, Part III*, page 105–134, Berlin, Heidelberg, 2024. Springer-Verlag. doi:10.1007/978-3-031-54776-8_5.

[141] Anca Nitulescu. Lattice-based zero-knowledge snargs for arithmetic circuits. In Peter Schwabe and Nicolas Thériault, editors, *Progress in Cryptology – LATINCRYPT 2019*, pages 217–236, Cham, 2019. Springer International Publishing. doi:10.1007/978-3-030-30530-7_11.

[142] Shen Noether, Adam Mackenzie, et al. Ring confidential transactions. *Ledger*, 1:1–18, 2016. doi:10.5195/ledger.2016.34.

[143] Shenle Pan, Damien Trentesaux, Duncan McFarlane, Benoit Montreuil, Eric Ballot, and George Q. Huang. Digital interoperability in logistics and supply chain management: State-of-the-art and research avenues towards physical internet. *Computers in Industry*, 128:103435, 2021. doi:10.1016/j.compind.2021.103435.

[144] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. *Commun. ACM*, 59(2):103–112, jan 2016. doi:10.1145/2856449.

[145] Holger Petersen. How to convert any digital signature scheme into a group signature scheme. In *Proceedings of the 5th International Workshop on Security Protocols*, page 177–190, Berlin, Heidelberg, 1997. Springer-Verlag. doi:10.1007/BFb0028168.

[146] Polygon Labs. Polygon zkEVM: Polygon knowledge layer. https://docs.polygon.technology/zkEVM/, 2024. Accessed: 2024-11-14.

[147] Charles M. Rader. Discrete Fourier transforms when the number of data samples is prime. *Proceedings of the IEEE*, 56(6):1107–1108, 1968. doi:10.1109/PROC.1968.6477.

[148] Irving Stoy Reed and Gustave Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304, 1960. doi:10.1137/0108018.

[149] A. Regattieri, M. Gamberi, and R. Manzini. Traceability of food products: General framework and experimental evidence. *Journal of Food Engineering*, 81(2):347–356, 2007. doi:10.1016/j.jfoodeng.2006.10.032.

[150] Fergal Reid and Martin Harrigan. *An Analysis of Anonymity in the Bitcoin System*, pages 197–223. Springer New York, New York, NY, 2013. doi:10.1007/978-1-4614-4139-7_10.

[151] Team Rocket, Maofan Yin, Kevin Sekniqi, Robbert van Renesse, and Emin Gün Sirer. Scalable and probabilistic leaderless bft consensus through metastability. 2020. doi:10.48550/arXiv.1906.08936.

[152] Phillip Rogaway. Evaluation of some blockcipher modes of operation. *Cryptography Research and Evaluation Committees (CRYPTREC) for the Government of Japan*, 630, 2011. URL: https://www.cs.ucdavis.edu/~rogaway/papers/modes-cryptrec.pdf.

[153] Dorit Ron and Adi Shamir. Quantitative analysis of the full bitcoin transaction graph. In Ahmad-Reza Sadeghi, editor, *Financial Cryptography and Data Security*, pages 6–24, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. doi:10.1007/978-3-642-39884-1_2.

[154] Stanislav Safaric and Kresimir Malaric. ZigBee wireless standard. In *Proceedings ELMAR 2006*, pages 259–262, 2006. doi:10.1109/ELMAR.2006.329562.

[155] Khaled Salah, Nishara Nizamuddin, Raja Jayaraman, and Mohammad Omar. Blockchain-based soybean traceability in agricultural supply chain. *IEEE Access*, 7:73295–73305, 2019. doi:10.1109/ACCESS.2019.2918000.

[156] Susanta Samanta. *Design and analysis of MDS and Near-MDS Matrices and their application to lightweight cryptography.* PhD thesis, Indian Statistical Institute, Kolkata, 2023. URL: http://dspace.isical.ac.in:8080/jspui/handle/10263/7421.

[157] Joseph Sarkis. Supply chain sustainability: learning from the covid-19 pandemic. *International Journal of Operations and Production Management*, 41:63–73, 1 2021. doi:10.1108/IJOPM-08-2020-0568/FULL/XML.

[158] C. P. Schnorr. Efficient signature generation by smart cards. *J. Cryptol.*, 4(3):161–174, jan 1991. doi:10.1007/BF00196725.

[159] Hanan R. Shehata, Danielle Bourque, Dirk Steinke, Shu Chen, and Robert Hanner. Survey of mislabelling across finfish supply chain reveals mislabelling both outside and within Canada. *Food Research International*, 121:723–729, 2019. doi:10.1016/j.foodres.2018.12.047.

[160] Ferdinand Sibleyras. *Security of Modes of Operation and other provably secure cryptographic schemes.* Theses, Sorbonne Université, October 2020. URL: https://hal.science/tel-03058306.

[161] Standards for Efficient Cryptography Group. SEC 2: Recommended Elliptic Curve Domain Parameters, 2010. URL: https://www.secg.org/sec2-v2.pdf.

[162] StarkWare Industries. StarkNet: validity-rollup (ZK-rollup) on Ethereum. https://www.starknet.io/, 2024. Accessed: 2024-11-14.

[163] Samuel Steffen, Benjamin Bichsel, Roger Baumgartner, and Martin Vechev. ZeeStar: Private smart contracts by homomorphic encryption and zero-knowledge proofs. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 179–197, May 2022. doi:10.1109/SP46214.2022.9833732.

[164] Samuel Steffen, Benjamin Bichsel, and Martin Vechev. Zapper: Smart contracts with data and identity privacy. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, page 2735–2749, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3548606.3560622.

[165] Tapas Sudan and Rashi Taggar. Recovering supply chain disruptions in post-COVID-19 pandemic through transport intelligence and logistics systems: India's experiences and policy options. *Frontiers in Future Transportation*, 2, 2021. doi:10.3389/ffutr.2021.660116.

[166] Haochen Sun, Jason Li, and Hongyang Zhang. zkLLM: Zero knowledge proofs for large language models. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, CCS '24, page 4405–4419, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3658644.3670334.

[167] Shengnan Sun and Xinping Wang. Promoting traceability for food supply chain with certification. *Journal of Cleaner Production*, 217:658–665, 2019. doi:10.1016/j.jclepro.2019.01.296.

[168] Weng Chun Tan and Manjit Singh Sidhu. Review of RFID and IoT integration in supply chain management. *Operations Research Perspectives*, 9:100229, 2022. doi:10.1016/j.orp.2022.100229.

[169] Justin Thaler. Proofs, arguments, and zero-knowledge. *Foundations and Trends® in Privacy and Security*, 4(2–4):117–660, 2022. doi:10.1561/3300000030.

[170] Louis Tremblay Thibault, Tom Sarry, and Abdelhakim Senhaji Hafid. Blockchain scaling using rollups: A comprehensive survey. *IEEE Access*, 10:93039–93054, 2022. doi:10.1109/ACCESS.2022.3200051.

[171] Feng Tian. An agri-food supply chain traceability system for china based on RFID & blockchain technology. In *2016 13th International Conference on Service Systems and Service Management (ICSSSM)*, pages 1–6, 2016. doi:10.1109/ICSSSM.2016.7538424.

[172] Kentaroh Toyoda, P. Takis Mathiopoulos, Iwao Sasase, and Tomoaki Ohtsuki. A novel blockchain-based product ownership management system (POMS) for anti-counterfeits in the post supply chain. *IEEE Access*, 5:17465–17477, 2017. doi:10.1109/ACCESS.2017.2720760.

[173] UN Global Compact. SME engagement strategy, 2022. URL: https://unglobalcompact.org/library/6049.

[174] Daniel van Flymen. *Learn Blockchain by Building One: A Concise Path to Understanding Cryptocurrencies.* Apress Berkeley, CA, 1 2020. doi:10.1007/978-1-4842-5171-3/COVER.

[175] Friedhelm Victor. Address clustering heuristics for Ethereum. In Joseph Bonneau and Nadia Heninger, editors, *Financial Cryptography and Data Security*, pages 617–633, Cham, 2020. Springer International Publishing. doi:10.1007/978-3-030-51280-4_33.

[176] Alex Vlasov, Kelly Olson, Alex Stokes, and Antonio Sanso. EIP-2537: Perecompile for BLS12-381 curve operations [draft]. *Ethereum Improvement Proposals, no. 2537*, 2 2020. URL: https://eips.ethereum.org/EIPS/eip-2537.

[177] Yao Wang and Xuelong Zhu. A fast algorithm for the Fourier transform over finite fields and its VLSI implementation. *IEEE Journal on Selected Areas in Communications*, 6(3):572–577, 1988. doi:10.1109/49.1926.

[178] Chenkai Weng, Kang Yang, Xiang Xie, Jonathan Katz, and Xiao Wang. Mystique: Efficient conversions for Zero-Knowledge proofs with applications to machine learning. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 501–518. USENIX Association, August 2021. URL: https://www.usenix.org/conference/usenixsecurity21/presentation/weng.

[179] Erich Wenger and Michael Hutter. Exploring the design space of prime field vs. binary field ECC-hardware implementations. In Peeter Laud, editor, *Information Security Technology for Applications*, pages 256–271, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. doi:10.1007/978-3-642-29615-4_18.

[180] Zachary J. Williamson. The AZTEC protocol. *Aztec Protocol*, 2018. URL: https://github.com/AztecProtocol/aztec-v1/blob/develop/AZTEC.pdf.

[181] Matthias Winter, Silvia Dopler, Julian M. Müller, and Alexander Zeisler. Information sharing and multi-tier supply chain management of SMEs in the context of industry 4.0. *Procedia Computer Science*, 217:1378–1385, 2023. 4th International Conference on Industry 4.0 and Smart Manufacturing. doi:10.1016/j.procs.2022.12.336.

[182] Joel D Wisner, Keah-Choon Tan, Keong Leong, et al. *Principles of supply chain management: A balanced approach*. South-Western, Cengage Learning, 2021.

[183] Alex Luoyuan Xiong, Binyi Chen, Zhenfei Zhang, Benedikt Bünz, Ben Fisch, Fernando Krell, and Philippe Camacho. VERI-ZEXE: Decentralized private computation with universal setup. In *32st USENIX Security Symposium (USENIX Security 23)*, pages 4445–4462, Anaheim, CA, August 2023. USENIX Association. URL: https://www.usenix.org/conference/usenixsecurity23/presentation/xiong.

[184] Xingchun Yang, Chunxiang Xu, Chaorong Li, et al. A privacy model for RFID tag ownership transfer. *Security and Communication Networks*, 2017, 2017. doi:10.1155/2017/5084636.

[185] 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 (SP)*, pages 859–876, 2020. doi:10.1109/SP40000.2020.00052.

[186] Jiajun Zhou, Chenkai Hu, Jianlei Chi, Jiajing Wu, Meng Shen, and Qi Xuan. Behavior-aware account de-anonymization on ethereum interaction graph. *IEEE Transactions on Information Forensics and Security*, 17:3433–3448, 2022. doi:10.1109/TIFS.2022.3208471.

[187] ZKsync Community. ZKsync EVM: Overview. https://docs.zksync.io/zk-stack/components/zksync-evm, 2023. Accessed: 2024-11-14.

APPENDICES

Appendix A

An Example of Converting a Boolean Circuit to R1CS

In this appendix we adopt an example from [92] which shows the computation of rank-1 constraint system (R1CS) instance from encoding a Boolean circuit depicted in Figure A.1. To encode the circuit to an R1CS instance, we are going to find $\mathbf{A}, \mathbf{B}, \mathbf{C} \in \mathbb{F}_2^{d_1 \times (d_2+1)}$ matrices for the circuit, where $d_1 = 3$ denotes the number of `AND` gates (number of constraints) and $d_2 = 7$ denotes the number of variables. We have three `AND` gates, denoted as $\mathbf{gate}_i$. Let vector $\vec{z}^{\intercal} = (z_0, z_1, \ldots, z_7)$ where $z_0 = 1$ represents inputs and wire values, Table A.1 shows left and right inputs and output of each gate.

Table A.1: Left and right inputs, along with the output of each gate. `XOR` gates are merged into `AND` gates.

$\mathbf{gate}_i$	g_l		g_r		g_o	
1	z_1	$\wedge$	z_2	$\oplus$	z_5	$= 0$
2	z_2	$\wedge$	z_3	$\oplus$	z_6	$= 0$
3	$(z_5 \oplus z_6)$	$\wedge$	$(z_3 \oplus z_4)$	$\oplus$	z_7	$= 0$

Accordingly, we can make $\mathbf{A}, \mathbf{B}, \mathbf{C}$ matrices:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}, B = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}, C = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.1}$$

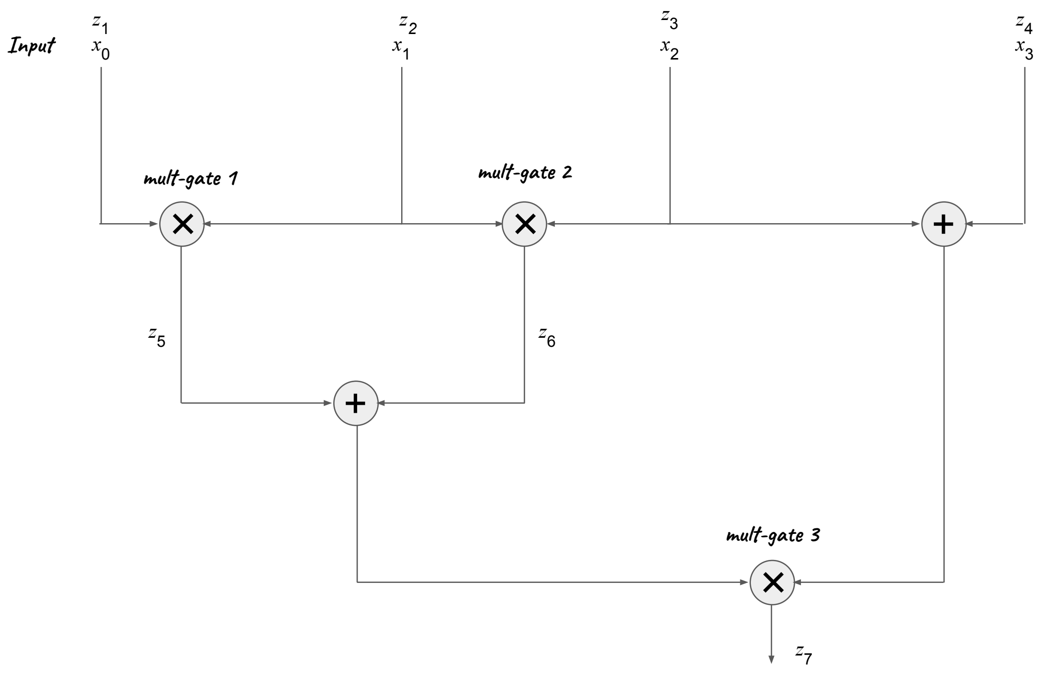


Figure A.1: An example of a Boolean circuit in which *mult-gates* are `AND` gates.

Consequently, setting $(z_1, z_2, z_3, z_4) = (1, 1, 0, 1)$ (inputs) results the remaining $(z_5, z_6, z_7) = (1, 0, 1)$ which are internal wires or the output. Hence, $\vec{z}^{\intercal} = (1, 1, 1, 0, 1, 1, 0, 1)$. The computed $\vec{z}$ satisfies $(A \cdot \vec{z}) \circ (B \cdot \vec{z}) - (C \cdot \vec{z}) = 0$.

Appendix B

Multiplication of Elements in Binary Extension Field $\mathbb{F}_{2^{256}}$

In this appendix we provide the approach existed in [37] for multiplying two elements in $a, b \in \mathbb{F}_{2^{256}}$. Let

$$\mathbb{F}_{2^{256}} := \mathbb{F}_2[x]/(x^{256} + x^{10} + x^5 + x^2 + 1) \tag{B.1}$$

define the field, where $p(x) := x^{256} + x^{10} + x^5 + x^2 + 1$ denotes a primitive irreducible polynomial over $\mathbb{F}_2$. Accordingly,

$$a(x) \cdot b(x) \bmod p(x)$$

denotes the multiplication of a and b in the field $\mathbb{F}_{2^{256}}$ as defined above, where $a(x)$ and $b(x)$ are the polynomial representations of a and b, respectively, as described below.

Partitioning Let $a, b \in \mathbb{F}_{2^{256}}$ be represented as binary polynomials of degree < 256

$$a(x) := a_0 + a_1x + a_2x^2 + a_3x^3 + \cdots + a_{255}x^{255} \in \mathbb{F}_2[x],$$

$$b(x) := b_0 + b_1x + b_2x^2 + b_3x^3 + \cdots + b_{255}x^{255} \in \mathbb{F}_2[x],$$

such that their polynomial product $c(x) = a(x) \cdot b(x)$ is represented as

$$c(x) := c_0 + c_1x + c_2x^2 + c_3x^3 + \cdots + c_{510}x^{510} \in \mathbb{F}_2[x].$$

Given a multiplication function $M : \{0,1\}^{64} \times \{0,1\}^{64} \to \{0,1\}^{128}$ that multiplies two 64-bit binary numbers and is instantiated by the CLMUL instruction on many x86 architecture CPUs [97], the Karatsuba multiplication algorithm [109] is applied to compute the product of $a(x)$ and $b(x)$. To do so, each polynomial is partitioed into four 64-bit limbs $a'_i(x), b'_i(x) \in \{0,1\}^{64}$ for $i = 0, 1, 2, 3$ defined as

$$\begin{aligned} a'_0(x) &:= a_0 + \cdots + a_{63}x^{63}, & a'_1(x) &:= a_{64} + \cdots + a_{127}x^{127}, \\ a'_2(x) &:= a_{128} + \cdots + a_{191}x^{191}, & a'_3(x) &:= a_{192} + \cdots + a_{255}x^{255}, \\ b'_0(x) &:= b_0 + \cdots + b_{63}x^{63}, & b'_1(x) &:= b_{64} + \cdots + b_{127}x^{127}, \\ b'_2(x) &:= b_{128} + \cdots + b_{191}x^{191}, & b'_3(x) &:= b_{192} + \cdots + b_{255}x^{255}. \end{aligned}$$

Accordingly, a and b will be represented in terms of $X = x^{64}$ as

$$\begin{aligned} a(X) &= a'_0 + a'_1 X + a'_2 X^2 + a'_3 X^3, \\ b(X) &= b'_0 + b'_1 X + b'_2 X^2 + b'_3 X^3. \end{aligned}$$

Accordingly, c will be

$$c(X) := c'_0 + c'_1 X + c'_2 X^2 + \cdots + c'_6 X^6, \tag{B.2}$$

where $c'_i \in \{0,1\}^{128}$ for $i = 0, \ldots, 6$ are overlapping 128-bit limbs in $c(X)$.

Polynomial Multiplication Algorithm Now, we present the multiplication algorithm used in [34]. Given the function M previously defined, let t and u, v intermediate auxilary 128-bit binary variables have the following values:

$$\begin{aligned} t &:= M(a'_1, b'_1), \\ u &:= M(a'_2, b'_2), \\ v &:= t \oplus^{128} u, \end{aligned}$$

where $\oplus^{128} : \{0,1\}^{128} \times \{0,1\}^{128} \to \{0,1\}^{128}$ denotes the 128-bit XOR opreator. Moreover, the following values are 64-bit itermediate auxilary variables:

$$\begin{aligned} w_0 &:= a'_0 \oplus^{64} a'_1, & y'_0 &:= b_0 \oplus^{64} b'_1, \\ w_1 &:= a'_0 \oplus^{64} a'_2, & y_1 &:= b'_0 \oplus^{64} b'_2, \\ w_2 &:= a'_2 \oplus^{64} a'_3, & y_2 &:= b'_2 \oplus^{64} b'_3, \\ w_3 &:= a'_1 \oplus^{64} a'_3, & y_3 &:= b'_1 \oplus^{64} b'_3, \\ w_4 &:= w_0 \oplus^{64} w_2, & y_4 &:= y_0 \oplus^{64} y_4, \end{aligned}$$

where $\oplus^{64} : \{0,1\}^{64} \times \{0,1\}^{64} \to \{0,1\}^{64}$ denotes the 64-bit XOR opreator. Using these, the valuse of c_i's are computed as follows:

$$\begin{aligned}
c_0' &:= M(a_0, b_0),\\
c_6' &:= M(a_3, b_3),\\
c_1' &:= M(w_0, y_0) \oplus^{128} c_0' \oplus^{128} t,\\
c_2' &:= M(w_1, y_1) \oplus^{128} c_0' \oplus^{128} v,\\
c_5' &:= M(w_2, y_2) \oplus^{128} c_6' \oplus^{128} u,\\
c_4' &:= M(w_3, y_3) \oplus^{128} c_6' \oplus^{128} v,\\
c_3' &:= M(w_4, y_4) \oplus^{128} c_0' \oplus^{128} c_1' \oplus^{128} c_2' \oplus^{128} c_4' \oplus^{128} c_5' \oplus^{128} c_6'.
\end{aligned}$$

Now, the overlapping c_is in (B.2) are merged to represent $c(X)$ as non-overlaping four 128-bit limbs $c_i'' \in \{0,1\}^{128}$ for $i = 0, \dots, 3$, such that

$$c(Y) := c_0'' + c_1''Y + c_2''Y^2 + c_3''Y^3, \tag{B.3}$$

where $Y = x^{128}$ and each c_i'' is computed as follows:

$$\begin{aligned}
c_0'' &:= c_0' \oplus^{128} (c_1' \ll 64),\\
c_1'' &:= c_2' \oplus^{128} (c_1' \gg 64) \oplus^{128} (c_3' \ll 64),\\
c_2'' &:= c_4' \oplus^{128} (c_3' \gg 64) \oplus^{128} (c_5' \ll 64),\\
c_3'' &:= c_6' \oplus^{128} (c_5' \gg 64),
\end{aligned}$$

where $\ll 64$ shifts the bits to the left by 64 positions, discarding the bits shifted out from the most significant end and inserting zeros into the least significant positions. Similarly, $\gg 64$ shifts the bits to the right by 64 positions, discarding the bits shifted out from the least significant end and inserting zeros into the most significant positions.

Reduction In this step, the polynomial in (B.3) is reduced modulo $p(x)$. Then the final result lies in $\mathbb{F}_{2^{256}}$, and is denoted as

$$d(x) := a(x) \cdot b(x) \bmod p(x),$$

where $d(x) = d_0 + d_1x + \cdots + d_{255}x^{255}$ is the polynomial representation of $d \in \mathbb{F}_{2^{256}}$, which can alternatively be expressed using four 64-bit limbs as

$$d(X) := d_0' + d_1'X + d_2'X^2 + d_3'X^3, \tag{B.4}$$

where $d'_i \in \{0,1\}^{64}$ for $i = 0,1,2,3$. Let us define the polynomial $p'(x) := x^{10} + x^5 + x^2 + 1$ from $p(x)$, corresponding to the 10-bit binary number $p' = (10000100101)_2$. To compute $d(X)$, we assume the following partitioning of terms in (B.3) into two 64-bit limbs:

$$c''_0 := c''_{00} + c''_{01}X, \qquad c''_1 := c''_{10} + c''_{11}X,$$
$$c''_2 := c''_{20} + c''_{21}X, \qquad c''_3 := c''_{30} + c''_{31}X,$$

where $c''_{i0}, c''_{i1} \in \{0,1\}^{64}$ for $i = 0,1,2,3$. Now, we define the following 128-bit auxilary intermediate values

$$z_0 := M(c''_{31}, p), \qquad z_1 := M(c''_{30}, p),$$
$$z_2 := M(c''_{21}, p), \qquad z_3 := M(c''_{20} \oplus^{64} z_{01}, p),$$

where they are partitioned into two 64-bit limbs as follows:

$$z_0 := z_{00} + z_{01}X, \qquad z_1 := z_{10} + z_{11}X,$$
$$z_2 := z_{20} + z_{21}X, \qquad z_3 := z_{30} + z_{31}X,$$

where $z_{i0}, z_{i1} \in \{0,1\}^{64}$ for $i = 0,1,2,3$.

Now, d'_0, d'_1, d'_2, d'_3 are computed as follows:

$$\begin{aligned} d'_0 &:= c''_{00} \oplus^{64} z_{30}, \\ d'_1 &:= c''_{01} \oplus^{64} z_{20} \oplus^{64} z_{31}, \\ d'_2 &:= c''_{10} \oplus^{64} z_{10} \oplus^{64} z_{21}, \\ d'_3 &:= c''_{11} \oplus^{64} z_{00} \oplus^{64} z_{11}. \end{aligned}$$

Consequently, $d(X)$ and then $d = a.b$ are derived.

Appendix C

The Pseudocodes of the Algorithms in the Zupply Framework

Algorithm C.1 $\mathsf{Init}\ (\mathrm{PP},\ q_{i_1}) \rightarrow (\mathtt{tx}_{\mathsf{Init}},\ T_{i_1})$

```
1: $\mathrm{PP}_{\mathsf{sig}} \leftarrow \mathrm{PP}$
2: $\rho_{i_1} \overset{R}{\leftarrow} \{0,1\}^{N_\rho}$
3: $(\mathrm{PKsig}_{i_1}, \mathrm{SKsig}_{i_1}) \leftarrow \mathcal{K}_{\mathsf{sig}}\big(\mathrm{PP}_{\mathsf{sig}}\big)$
4: $\tilde{T}_{i_1} \leftarrow \big(q_{i_1}, \mathrm{PKsig}_{i_1}, \rho_{i_1}\big)$, $T_{i_1} \leftarrow \big(\tilde{T}_{i_1}, \mathrm{SKsig}_{i_1}\big)$
5: $\mathtt{cm}_{i_1} \leftarrow \mathsf{COMM}_{\rho_{i_1}}(\tilde{T}_{i_1})$
6: $(\mathtt{rt}^{\mathrm{new}}, \mathsf{ind}_{i_1}, \mathsf{path}_{\mathsf{ind}_{i_1}}) \leftarrow \mathsf{MHT.Add}(\mathtt{cm}_{i_1})$
7: $\mathtt{tx}_{\mathsf{Init}} \leftarrow \big(\mathtt{cm}_{i_1}, \mathtt{rt}^{\mathrm{new}}, \mathsf{ind}_{i_1}, \mathsf{path}_{\mathsf{ind}_{i_1}}\big)$
8: **return** $\mathtt{tx}_{\mathsf{Init}}, T_{i_1}$
```

Algorithm C.2 Trans $\left(\mathrm{PP}, T_{i_1}, \mathrm{PKsig}_{i_2}\right) \rightarrow \left(\mathtt{tx}_{\mathsf{Trans}}, \tilde{T}_{i_2}, \mathtt{Tag}\right)$

```
1: $\mathtt{cm}_{i_1} \leftarrow \mathsf{COMM}_{\rho_{i_1}}(\tilde{T}_{i_1})$
2: if $\mathtt{cm}_{i_1} \notin \mathbf{C}_\tau$ then
3:     return $\perp$
4: end if
5: $(\mathsf{ind}_{i_1}, \mathsf{path}_{\mathsf{ind}_{i_1}}) \leftarrow \mathsf{MHT.Search}(\mathtt{cm}_{i_1})$
6: $\mathsf{pk}_{\mathsf{Trans}} \leftarrow \mathrm{PP}$
7: $\mathtt{eol}_{i_1} \leftarrow \mathcal{H}(\rho_{i_1})$
8: $q_{i_2} \leftarrow q_{i_1}$
9: $\rho_{i_2} \overset{R}{\leftarrow} \{0,1\}^{N_\rho}$
10: $\tilde{T}_{i_2} \leftarrow \left(q_{i_2}, \mathrm{PKsig}_{i_2}, \rho_{i_2}\right)$
11: $\mathtt{cm}_{i_2} \leftarrow \mathsf{COMM}_{\rho_{i_2}}(\tilde{T}_{i_2})$
12: $x_{\mathsf{Trans}} \leftarrow \left(\mathtt{eol}_{i_1}, \mathtt{cm}_{i_2}, \mathtt{rt}_\tau\right)$
13: $w_{\mathsf{Trans}} \leftarrow \left(\tilde{T}_{i_1}, \tilde{T}_{i_2}, \mathsf{path}_{\mathsf{ind}_{i_1}}\right)$
14: $\pi_{\mathsf{Trans}} \leftarrow \mathsf{Prove}\left(\mathsf{pk}_{\mathsf{Trans}}, x_{\mathsf{Trans}}, w_{\mathsf{Trans}}\right)$
15: $(\mathtt{rt}^{\mathrm{new}}, \mathsf{ind}_{i_2}, \mathsf{path}_{\mathsf{ind}_{i_2}}) \leftarrow \mathsf{MHT.Add}(\mathtt{cm}_{i_2})$
16: $\mathtt{tx}_{\mathsf{Trans}} \leftarrow \left(\pi_{\mathsf{Trans}}, x_{\mathsf{Trans}}, \mathtt{eol}_{i_1}, \mathtt{cm}_{i_2}, \mathtt{rt}^{\mathrm{new}}, \mathsf{ind}_{i_2}, \mathsf{path}_{\mathsf{ind}_{i_2}}\right)$
17: $\mathtt{Tag} \leftarrow \mathcal{S}_{\mathsf{sig}}(\mathrm{SKsig}_{i_1}, \mathrm{PKsig}_{i_2})$
18: return $\mathtt{tx}_{\mathsf{Trans}}, \tilde{T}_{i_2}$ , $\mathtt{Tag}$
```

Algorithm C.3 Merge $\left(\mathrm{PP}, T_{i_{1,2}}, \mathrm{PKsig}_{i_3}\right) \rightarrow \left(\mathtt{tx}_{\mathsf{Merge}}, \tilde{T}_{i_3}, \mathtt{Tag}_{i_{1,2}}\right)$

```
1:  $\mathtt{cm}_{i_1} \leftarrow \mathsf{COMM}_{\rho_{i_1}}(\tilde{T}_{i_1})$, $\mathtt{cm}_{i_2} \leftarrow \mathsf{COMM}_{\rho_{i_2}}(\tilde{T}_{i_2})$
2:  if $\mathtt{cm}_{i_1} \notin \mathbf{C}_\tau$ or $\mathtt{cm}_{i_2} \notin \mathbf{C}_\tau$ then
3:      return $\perp$
4:  end if
5:  $(\mathsf{ind}_{i_1}, \mathsf{path}_{\mathsf{ind}_{i_1}}) \leftarrow \mathsf{MHT.Search}(\mathtt{cm}_{i_1})$
6:  $(\mathsf{ind}_{i_2}, \mathsf{path}_{\mathsf{ind}_{i_2}}) \leftarrow \mathsf{MHT.Search}(\mathtt{cm}_{i_2})$
7:  $\mathsf{pk}_{\mathsf{Merge}} \leftarrow \mathrm{PP}$
8:  $\mathtt{eol}_{i_1} \leftarrow \mathcal{H}\left(\rho_{i_1}\right)$, $\mathtt{eol}_{i_2} \leftarrow \mathcal{H}\left(\rho_{i_2}\right)$
9:  $q_{i_3} \leftarrow q_{i_1} + q_{i_2}$
10: $\rho_{i_3} \overset{R}{\leftarrow} \{0,1\}^{N_\rho}$
11: $\tilde{T}_{i_3} \leftarrow \left(q_{i_3}, \mathrm{PKsig}_{i_3}, \rho_{i_3}\right)$
12: $\mathtt{cm}_{i_3} \leftarrow \mathsf{COMM}_{\rho_{i_3}}(\tilde{T}_{i_3})$
13: $x_{\mathsf{Merge}} \leftarrow \left(\mathtt{eol}_{i_{1,2}}, \mathtt{cm}_{i_3}, \mathtt{rt}_\tau\right)$
14: $w_{\mathsf{Merge}} \leftarrow \left(\tilde{T}_{i_{1,2}}, \tilde{T}_{i_3}, \mathsf{path}_{\mathsf{ind}_{i_{1,2}}}\right)$
15: $\pi_{\mathsf{Merge}} \leftarrow \mathsf{Prove}\left(\mathsf{pk}_{\mathsf{Merge}}, x_{\mathsf{Merge}}, w_{\mathsf{Merge}}\right)$
16: $(\mathtt{rt}^{\mathrm{new}}, \mathsf{ind}_{i_3}, \mathsf{path}_{\mathsf{ind}_{i_3}}) \leftarrow \mathsf{MHT.Add}(\mathtt{cm}_{i_3})$
17: $\mathtt{tx}_{\mathsf{Merge}} \leftarrow \left(\pi_{\mathsf{Merge}}, x_{\mathsf{Merge}}, \mathtt{eol}_{i_{1,2}}, \mathtt{cm}_{i_3}, \mathtt{rt}^{\mathrm{new}}, \mathsf{ind}_{i_3}, \mathsf{path}_{\mathsf{ind}_{i_3}}\right)$
18: $\mathtt{Tag}_{i_1} \leftarrow \mathcal{S}_{\mathsf{sig}}(\mathrm{SKsig}_{i_1}, \mathrm{PKsig}_{i_3})$
19: $\mathtt{Tag}_{i_2} \leftarrow \mathcal{S}_{\mathsf{sig}}(\mathrm{SKsig}_{i_2}, \mathrm{PKsig}_{i_3})$
20: return $\mathtt{tx}_{\mathsf{Merge}}, \tilde{T}_{i_3}$ , $\mathtt{Tag}_{i_{1,2}}$
```

Algorithm C.4 Divide $\big(\mathrm{PP}, T_{i_1}, \mathrm{PKsig}_{i_{2,3}}, q_{i_{2,3}}\big) \rightarrow \big(\mathtt{tx}_{\mathsf{Div}}, \tilde{T}_{i_{2,3}}, \mathtt{Tag}_{i_{2,3}}\big)$

```
1: $\mathtt{cm}_{i_1} \leftarrow \mathsf{COMM}_{\rho_{i_1}}(\tilde{T}_{i_1})$
2: if $\mathtt{cm}_{i_1} \notin \mathbf{C}_\tau$ or $q_{i_1} \neq q_{i_2} + q_{i_3}$ then
3:     return $\perp$
4: end if
5: $(\mathsf{ind}_{i_1}, \mathsf{path}_{\mathsf{ind}_{i_1}}) \leftarrow \mathsf{MHT.Search}(\mathtt{cm}_{i_1})$
6: $\mathsf{pk}_{\mathsf{Div}} \leftarrow \mathrm{PP}$
7: $\mathtt{eol}_{i_1} \leftarrow \mathcal{H}\big(\rho_{i_1}\big)$
8: $\rho_{i_2} \xleftarrow{R} \{0,1\}^{N_\rho}$, $\rho_{i_3} \xleftarrow{R} \{0,1\}^{N_\rho}$
9: $\tilde{T}_{i_2} \leftarrow \big(q_{i_2}, \mathrm{PKsig}_{i_2}, \rho_{i_2}\big)$, $\tilde{T}_{i_3} \leftarrow \big(q_{i_3}, \mathrm{PKsig}_{i_3}, \rho_{i_3}\big)$
10: $\mathtt{cm}_{i_2} \leftarrow \mathsf{COMM}_{\rho_{i_2}}(\tilde{T}_{i_2})$, $\mathtt{cm}_{i_3} \leftarrow \mathsf{COMM}_{\rho_{i_3}}(\tilde{T}_{i_3})$
11: $x_{\mathsf{Div}} \leftarrow \big(\mathtt{eol}_{i_1}, \mathtt{cm}_{i_{2,3}}, \mathtt{rt}_\tau\big)$
12: $w_{\mathsf{Div}} \leftarrow \big(\tilde{T}_{i_1}, \tilde{T}_{i_{2,3}}, \mathsf{path}_{\mathsf{ind}_{i_1}}\big)$
13: $\pi_{\mathsf{Div}} \leftarrow \mathsf{Prove}\big(\mathsf{pk}_{\mathsf{Div}}, x_{\mathsf{Div}}, w_{\mathsf{Div}}\big)$
14: $(\mathtt{rt}_1^{\mathrm{new}}, \mathsf{ind}_{i_2}, \mathsf{path}_{\mathsf{ind}_{i_2}}) \leftarrow \mathsf{MHT.Add}(\mathtt{cm}_{i_2})$
15: $(\mathtt{rt}_2^{\mathrm{new}}, \mathsf{ind}_{i_3}, \mathsf{path}_{\mathsf{ind}_{i_3}}) \leftarrow \mathsf{MHT.Add}(\mathtt{cm}_{i_3})$
16: $\mathtt{tx}_{\mathsf{Div}} \leftarrow \big(\pi_{\mathsf{Div}}, x_{\mathsf{Div}}, \mathtt{eol}_{i_1}, \mathtt{cm}_{i_{2,3}}, \mathtt{rt}_{1,2}^{\mathrm{new}}, \mathsf{ind}_{i_{2,3}}, \mathsf{path}_{\mathsf{ind}_{i_{2,3}}}\big)$
17: $\mathtt{Tag}_2 \leftarrow \mathcal{S}_{\mathsf{sig}}(\mathrm{SKsig}_{i_1}, \mathrm{PKsig}_{i_2})$
18: $\mathtt{Tag}_3 \leftarrow \mathcal{S}_{\mathsf{sig}}(\mathrm{SKsig}_{i_1}, \mathrm{PKsig}_{i_3})$
19: return $\mathtt{tx}_{\mathsf{Div}}$, $\tilde{T}_{i_{2,3}}$ , $\mathtt{Tag}_{i_{2,3}}$
```

Algorithm C.5 VerifyTX $\big(\mathrm{PP}, \mathtt{tx}_{\mathbb{x}}\big) \to b \in \{0,1\}$

```
1: $\mathbb{x} \leftarrow$ type of $\mathtt{tx}_{\mathbb{x}}$    ▷ $\mathbb{x} \in \{\mathsf{Trans}, \mathsf{Merge}, \mathsf{Div}\}$
2: $\mathsf{vk}_{\mathbb{x}} \leftarrow \mathrm{PP}$
3: $(x_{\mathbb{x}}, \pi_{\mathbb{x}}, \mathtt{eol}_1, \mathtt{cm}_1, \mathtt{rt}_1^{\mathrm{new}}, \mathsf{ind}_1, \mathsf{path}_{\mathsf{ind}_1}) \leftarrow \mathtt{tx}$
4: if $\mathtt{eol}_1 \in \mathbf{X}_\tau$ then
5:     return 0
6: end if
7: if $\mathbb{x} = \mathsf{Merge}$ then
8:     $\mathtt{eol}_2 \leftarrow \mathtt{tx}$
9:     if $\mathtt{eol}_2 \in \mathbf{X}_\tau$ then
10:        return 0
11:    end if
12: end if
13: if $\mathsf{Verify}\big(\mathsf{vk}_{\mathbb{x}}, x_{\mathbb{x}}, \pi_{\mathbb{x}}\big) \neq 1$ then
14:     return 0
15: end if
16: if $\mathsf{MHT.Verify}(\mathtt{rt}_\tau, \mathtt{rt}_1^{\mathrm{new}}, \mathtt{cm}_1, \mathsf{ind}_1, \mathsf{path}_{\mathsf{ind}_1}) \neq 1$ then
17:     return 0
18: end if
19: if $\mathbb{x} = \mathsf{Div}$ then
20:     $(\mathtt{cm}_2, \mathtt{rt}_2^{\mathrm{new}}, \mathsf{ind}_2, \mathsf{path}_{\mathsf{ind}_2}) \leftarrow \mathtt{tx}$
21:     if $\mathsf{MHT.Verify}(\mathtt{rt}_1^{\mathrm{new}}, \mathtt{rt}_2^{\mathrm{new}}, \mathtt{cm}_2, \mathsf{ind}_2, \mathsf{path}_{\mathsf{ind}_2}) \neq 1$ then
22:        return 0
23:     end if
24: end if
25: return 1
```

Algorithm C.6 Upload $\big(\textsc{pp}, [\mathsf{pred}], T_i, d_{\mathsf{pub}}, d_{\mathsf{pri}}, \mathrm{K}_{n_3}, [\mathsf{tags}]\big) \rightarrow d_{n_3}$

```
 1: $\mathsf{pk}_{\mathsf{Auth}} \leftarrow \textsc{pp}$
 2: $\mathtt{cm}_{i_1} \leftarrow \mathsf{COMM}_{\rho_{i_1}}(\tilde{T}_{i_1})$
 3: $(\mathsf{ind}_{i_1}, \mathsf{path}_{\mathsf{ind}_{i_1}}) \leftarrow \mathsf{MHT.Search}(\mathtt{cm}_{i_1})$
 4: $(\mathrm{PKsig}_i, \mathrm{SKsig}_i) \leftarrow T_i$
 5: $c_{n_3} \leftarrow \mathcal{E}_{\mathsf{sym}}(\mathrm{K}_{n_3}, d_{n_3,\mathsf{pri}})$
 6: $x_{\mathtt{Auth}} \leftarrow (\mathtt{rt}_\tau, \mathrm{PKsig}_i)$
 7: $w_{\mathtt{Auth}} \leftarrow (T_i, \mathsf{path}_{\mathsf{ind}_i})$
 8: $\pi_{\mathsf{Auth}} \leftarrow \mathsf{Prove}\big(\mathsf{pk}_{\mathsf{Auth}}, x_{\mathsf{Auth}}, w_{\mathsf{Auth}}\big)$
 9: $m \leftarrow ([\mathsf{pred}], d_{\mathsf{pub}}, c_{n_3}, [\mathsf{tags}], \pi_{\mathsf{Auth}}, x_{\mathtt{Auth}}, \mathbf{L}_\tau^{\mathsf{PI}}(\mathtt{rt}_\tau))$
10: $\sigma_{n_3} \leftarrow \mathcal{S}_{\mathsf{sig}}\big(\mathrm{SKsig}_i, m\big)$
11: $d_{n_3} \leftarrow \big([\mathsf{pred}], d_{\mathsf{pub}}, c_{n_3}, [\mathsf{tags}], \pi_{\mathtt{Auth}}, x_{\mathtt{Auth}}, \mathbf{L}_\tau^{\mathsf{PI}}(\mathtt{rt}_\tau), \sigma_{n_3}\big)$
12: **return** $d_{n_3}$
```

Algorithm C.7 Audit $\big(\textsc{pp}, d_n\big) \rightarrow b \in \{0,1\}$

```
1: $\mathsf{vk}_{\mathsf{Auth}} \leftarrow \textsc{pp}$
2: $\big([\mathsf{pred}], d_{\mathsf{pub}}, c_n, [\mathsf{tags}], \pi_{\mathtt{Auth}}, x_{\mathtt{Auth}}, \mathbf{L}^{\mathsf{PI}}_{\tau}(\mathtt{rt}_{\tau}), \sigma_n\big) \leftarrow d_n$
3: $(\mathtt{rt}_{\tau}, \mathrm{PKsig}_i) \leftarrow x_{\mathtt{Auth}}$
4: $m \leftarrow ([\mathsf{pred}], d_{\mathsf{pub}}, c_n, [\mathsf{tags}], \pi_{\mathsf{Auth}}, x_{\mathtt{Auth}})$
5: **if** $\mathbf{L}^{\mathsf{PI}}_{\tau}(\mathtt{rt}_{\tau})$ is not valid **then**
6:     **return** 0
7: **end if**
8: **if** $\mathcal{V}_{\mathsf{sig}}(\mathrm{PKsig}_i, m, \sigma_n) = 0$ **then**
9:     **return** 0
10: **end if**
11: **if** $\mathsf{Verify}(\mathsf{vk}_{\mathsf{Auth}}, x_{\mathtt{Auth}}, \pi_{\mathtt{Auth}}) = 0$ **then**
12:     **return** 0
13: **end if**
14: **while not** $\mathsf{pred.isEmpty}()$ **do**
15:     $\textsc{cid} \leftarrow \mathsf{pred.pop}()$
16:     $d_p \leftarrow \mathsf{DCS}(\textsc{cid})$
17:     $\mathrm{PKsig}_j \leftarrow d_p$
18:     **if** $\mathrm{PKsig}_i \neq \mathrm{PKsig}_j$ **then**
19:         $\mathsf{Tag} \leftarrow \mathsf{tags.pop}()$
20:         **if** $\mathcal{V}_{\mathsf{sig}}(\mathrm{PKsig}_j, \mathrm{PKsig}_i, \mathtt{Tag}) = 0$ **then**
21:             **return** 0
22:         **end if**
23:     **end if**
24: **end while**
25: **return** 1
```