Shorter VOLE-in-the-Head-based Signatures from Vector Semi-Commitment

Seongkwang Kim*
Korea University
Seoul, Korea
seongkwang-kim@korea.ac.kr

Byeonghak Lee
Samsung SDS
Seoul, Korea
byghak.lee@samsung.com

Mincheol Son†
Korea Advanced Institute of Science and Technology
Daejeon, Korea
encrypted.def@kaist.ac.kr

Abstract

The VOLE-in-the-Head (VOLEitH) paradigm compiles VOLE-based zero-knowledge proofs into publicly verifiable post-quantum signature schemes. A central component of VOLEitH is a GGM-style tree used to realize a vector commitment, whose size dominates the signature.

In this work, we adapt vector semi-commitment (VSC)—previously developed for MPC-in-the-Head (MPCitH) by Kim et al. (*Eurocrypt 2025*)—to the VOLEitH paradigm and prove its security. While prior MPCitH analyses relied on assumption-specific, fragmented security proofs, we give a *unified* treatment that characterizes which abstract property the underlying tree must satisfy, and instantiate it under the random oracle, Davies–Meyer, counter (CTR), and pseudorandom generator (PRG) models. Building on this, we construct reduced VOLE-in-the-Head (rVOLEitH) and apply it to FAEST, one of the NIST additional post-quantum signature candidates, obtaining signature sizes comparable to those of FAEST v2.0, ranging from 94.9% to 102.8% of the corresponding FAEST v2.0 sizes, while attaining 27.7–124.2% faster signing and 28.3–115.3% faster verification for 128-bit parameters.

Finally, we describe a forgery attack on a recently proposed VOLEitH variant of the subfield bilinear collision (SBC) signature scheme by Huth and Joux (*Asiacrypt 2025*) which replaces leaf commitments with a single global commitment, demonstrating the necessity of the commitment structure retained by VSC.

CCS Concepts

• **Security and privacy → Digital signatures**.

Keywords

VOLE-in-the-Head, Vector Semi-Commitment, Zero-Knowledge Proofs, Post-Quantum Signatures

ACM Reference Format:
Seongkwang Kim, Byeonghak Lee, and Mincheol Son. 2026. Shorter VOLE-in-the-Head-based Signatures from Vector Semi-Commitment. In *Proceedings of the 2026 ACM SIGSAC Conference on Computer and Communications Security (CCS '26), November 15–19, 2026, The Hague, Netherlands.* ACM, New York, NY, USA, 23 pages. https://doi.org/10.1145/3830454.3846805

1 Introduction

Vector oblivious linear evaluation (VOLE) is a two-party cryptographic protocol that generalizes oblivious transfer by enabling one party (the sender) to hold a secret vector $\mathbf{q} \in \mathbb{F}^n$ and a scalar multiplier $\Delta \in \mathbb{F}$, while the other party (the receiver) selects an input vector $\mathbf{u} \in \mathbb{F}^n$ and learns the evaluation

$$\mathbf{v} = \Delta \cdot \mathbf{u} - \mathbf{q}, \tag{1}$$

without learning any additional information about $\mathbf{q}$ or Δ. Meanwhile, the sender remains oblivious to $\mathbf{u}$. This linear relationship enables efficient generation of correlated randomness and serves as a building block for various secure computation protocols [8, 26].

One of the applications is VOLE-based zero-knowledge (VOLE-ZK) protocols, which employ the VOLE primitive to design prover-efficient and linear-sized zero-knowledge proofs [6, 28, 29]. In these protocols, the prover and verifier preprocess random VOLE correlations to commit to all wire values in the circuit and subsequently prove multiplicative relations among these values using information-theoretic authentication techniques. However, a critical limitation of these protocols is that the verifier must be designated before the proof, as Δ in VOLE must remain secret and is chosen during the preprocessing stage.

To overcome this limitation, Baum et al. introduced the VOLE-in-the-Head (VOLEitH) paradigm [5], a transformation of VOLE-ZK into a publicly verifiable proof using SoftSpoken OT [27]. This approach allows the verifier to delay choosing Δ in (1), thereby enabling public verifiability. The VOLE-in-the-Head paradigm has been used to build post-quantum signature schemes relying on trapdoor-less primitives, such as one-way functions [13, 21, 30], the unstructured multivariate quadratic problem [9], or the unstructured syndrome decoding problem [12]. Compared to the MPC-in-the-Head (MPCitH) [19] paradigm, which shares similar properties, VOLEitH-based signature schemes offer improved efficiency.

To efficiently generate randomness, both VOLEitH and MPCitH employed a GGM tree structure [15]. Since the GGM tree contributes a substantial portion of the signature size, and computing

*The initial draft of this work was prepared while Seongkwang Kim was at Samsung SDS, and the final revision was completed after joining Korea University. Seongkwang Kim was supported by Korea Internet & Security Agency (KISA) grant funded by the Korea government (PIPC) (No.2780000036, Masters and Doctoral Program for Advanced Talent Development in Personal Data Protection and Responsible Use), and in part by a Korea University Grant.
†Mincheol Son was supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. RS-2024-00399491, Development of Privacy-Preserving Multiparty Computation Techniques for Secure Multiparty Data Integration).

CCS '26, The Hague, Netherlands
© 2026 Copyright held by the owner/author(s).
This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record was published in *Proceedings of the 2026 ACM SIGSAC Conference on Computer and Communications Security (CCS '26), November 15–19, 2026, The Hague, Netherlands*, https://doi.org/10.1145/3830454.3846805.

it accounts for a substantial fraction of signing/verification time, there has been substantial prior work on optimizing its performance. Bui et al. proposed a fast, salted GGM tree that achieves full security in the ideal cipher model [10]. This salted tree maintains the evaluation speed of the best unsalted version while preventing multi-target attacks. Cui et al. applied the correlated GGM tree [16] to the VOLEitH-based signature scheme ReSolveD, optimizing the computation of GGM-based random vector commitments [12]. In parallel, Bui et al. replaced the random oracle calls in the correlated GGM with efficient primitives, such as fixed-key AES, and integrated this optimization into VOLEitH-based signature schemes [11]. Huth and Joux fixed the root of each cGGM tree to part of the secret key, exploiting the sum-preserving property to remove the correction term [17]. Kim et al. introduced a vector semi-commitment that relaxes the binding property of traditional vector commitments and applied it to an MPCitH-based signature scheme [22]. Although the vector semi-commitment allows an adversary to find multiple preimages for a given commitment, this relaxation of the binding property does not directly lead to a signature forgery.

1.1 Our Contribution

At a high level, we investigate security and efficiency of applying VSC to VOLEitH-based signature schemes, which results in the *reduced VOLE-in-the-Head (rVOLEitH)* paradigm. In addition, we demonstrate the vulnerability by analyzing the VOLEitH-SBC scheme [18], in which leaf commitments are removed rather than shortened as in VSC. The details are organized as follows.

Application of VSC to VOLEitH. In this paper, we apply vector semi-commitment (VSC) [22] to VOLEitH-based signature schemes. To apply VSC to an MPCitH-based signature scheme without increasing τ, the complexity of the Kales–Zaverucha attack [20] must be much larger than 2^λ. This is because multiple candidates for legitimate partial decommitment decrease the complexity of the Kales–Zaverucha attack. If the complexity is close to 2^λ, applying VSC may increase τ, and eventually the signature size. For VOLEitH-based signature schemes, τ is already set to be tight. There is no similar slack to absorb the cost of multiple candidates of partial decommitments.

We resolve this problem by integrating VoleCom with VSC, and VHash – which checks consistency of VOLE pairs across the GGM trees – into VoleComChk, and prove the extractable binding property rather than extractable semi-binding. VoleCom is a commitment to a VOLE pair generated from vector (semi-)commitments. If VSC is applied to VoleCom, then as in the VSC case itself, there may be a few VOLE-pair candidates which are compatible with a commitment. However, we prove that the VOLE pair compatible with a given $(\mathsf{com}, \widetilde{\mathbf{u}})$ is unique if the output size of VHash becomes slightly larger (B increases from 16 to 40 or 48).

Additionally, the extractable binding property of VoleComChk allows a more modular security proof. In [22], EUF-CMA security is not proved in a round-by-round security manner; since VSC is not binding by itself, the structure of the multiplication check should be considered in the EUF-CMA security proof. This implies that any change to the multiplication-checking protocol forces the entire EUF-CMA proof to be redone. However, in our case, since VoleComChk is binding by itself, EUF-CMA security is well-preserved even if the zero-knowledge protocol (e.g., QuickSilver [29]) is replaced by another sound ZK protocol.

Unification of Proof Model. The security analysis for the VSC technique was carried out separately for each idealized primitive: an ideal-cipher-based proof for the AES-style instantiation and a random-oracle-based proof for the hash-style instantiation. Such assumption-specific arguments do not transfer to all VOLEitH-based signature schemes of interest. In particular, FAEST [2] realizes its leaf-derivation function with AES in counter mode, which is modeled as a pseudorandom generator (PRG) rather than as an ideal cipher; modeling AES itself as an ideal cipher is problematic because the FAEST one-way function is AES, so the same primitive would simultaneously play the role of an idealized object and of a hardness assumption.

To address this, we abstract the GGM-style tree underlying the VSC into three keyed functions $\mathsf{F}_{\mathsf{tree}}$, $\mathsf{F}_{\mathsf{com}}$, $\mathsf{F}_{\mathsf{tape}}$, and identify a small set of generic properties that are sufficient to establish the extractable semi-binding and multi-instance hiding of the resulting VSC. We then verify these properties for four different sets of F-functions: VSC using random oracles, the Davies–Meyer construction over an ideal cipher, counter mode over an ideal cipher, and a generic PRG (under a non-standard computational assumption). This yields a single, modular security proof that simultaneously covers the VOLEitH instantiations with multiple sets of GGM-related functions, instead of a collection of model-specific arguments.

Implementation and Performance. We instantiate rVOLEitH with FAEST v2.0, obtaining rFAEST, and evaluate it against the corresponding FAEST parameter sets. The resulting signatures remain comparable in size to FAEST v2.0, ranging from 94.9% to 102.8% of the corresponding signature sizes across the evaluated 128-bit parameter sets. The main saving comes from the hidden-commitment term: each leaf commitment is shortened from 3λ to λ in FAEST and from 2λ to λ in FAEST-EM. This saving compensates for the increased VOLE-related and node-opening terms in all evaluated 128-bit parameter sets except EM-short, where the signature size slightly increases (by 108 bytes).

Beyond signature size, rFAEST also improves signing and verification time. The shorter leaf commitments reduce the input length of the SHAKE call that hashes leaf commitments, the leaf generation step uses only one PRG call per leaf, and the removal of the T_{open} condition reduces the cost of recomputing the final challenge. As a result, rFAEST achieves 27.7–124.2% faster signing and 28.3–115.3% faster verification compared with the corresponding FAEST parameters. The gains are larger for the short parameter sets than for the fast parameter sets, and for FAEST-based parameter sets than for FAEST-EM-based parameter sets.

Attack on VOLEitH-based Signature Scheme with MinMax Commitment. What if the leaf commitments are not reduced but removed? This is a quite natural question from our work. We explore this by analyzing the recently proposed VOLEitH-based signature scheme from [18]. The security of this scheme relies on the subfield bilinear collision (SBC) problem, introduced by Huth and Joux [17]. The same work also proposes an MPCitH-based signature scheme where the one-way function is based on SBC. However, one of the variants, called $\mathrm{SBC}^{M}_{\mathrm{VOLE}}$, deviates from the original structure of

VOLEitH by compressing individual seed commitments. This commitment is called the *MinMax commitment*. We show that this type of commitment does not provide sufficient binding and describe a forgery attack requiring no signature queries. Its estimated cost ranges from $2^{95.2}$ to $2^{121.2}$ for all the parameter sets.

2 Preliminaries

2.1 Notation

In this paper, we mainly follow the notation from three papers [4, 18, 22]. In Section 3, we follow the notations from [22] but modify them to fit into FAEST notations [4]. Section 4 follows the notations of FAEST [4]. Section 5 follows the notations of [18]. For consistency across sections, we start array indices with 0, not 1, while arrays in [22] usually start with index 1.

We denote $[n] = \{0, \dots, n-1\}$ and $[n]^* = \{1, \dots, n-1\}$. When this notation appears in an array, e.g., $(a_i)_{i\in[n]}$, we mean the array $(a_0, \dots, a_{n-1})$. For a set S, we write $a \leftarrow_\$ S$ to denote that a is chosen uniformly at random from S.

For two vectors or strings a and b, their concatenation is denoted by $a \| b$. For a vector $\mathbf{x}$, $\mathbf{x}[a:b]$ denotes a slice of the vector $\mathbf{x}$ from index a (inclusive) to b (exclusive). $\mathbf{x}[:b]$ and $\mathbf{x}[a:]$ denote the same as $\mathbf{x}[0:b]$ and $\mathbf{x}[a:\ell]$, respectively, where ℓ refers to the length of the vector $\mathbf{x}$. For a matrix $\mathbf{A}$, $\mathbf{A}[a:b]$ denotes a submatrix consisting of columns from index a (inclusive) to b (exclusive). For integers a and b, we denote the bitwise XOR of a and b by $a \oplus b$.

Unless stated otherwise, all logarithms are to the base 2. Throughout the paper, the security parameter is denoted by λ. The finite field of q elements is denoted by $\mathbb{F}_q$.

2.2 VOLE-in-the-Head

VOLE-in-the-Head [5] is a technique that extends VOLE-ZK to achieve public verifiability. Apart from the random challenges between the steps, this technique consists of the following four steps: (1) random VOLE commitment, (2) VOLE consistency check, (3) VOLE-ZK proof, (4) VOLE key selection.

As the details of each step are key to reading this paper, we will explain each step in the following. As we focus on the signature scheme based on the VOLE-in-the-Head paradigm, we will concentrate on the non-interactive version of VOLE-in-the-Head. We basically follow the notation of [4], with slightly different names.

FAEST. FAEST [2] is a non-interactive VOLEitH signature scheme for an AES-based one-way function (OWF). To sign, the prover generates random VOLE correlations from a GGM-based vector commitment, checks them with VHash, and shifts them to the OWF witness before applying QuickSilver to verify the AES relation. The Fiat–Shamir challenges determine which leaves remain hidden. Grinding and the one-tree BAVC opening rule reduce the size of the resulting opening. The rFAEST construction in Section 4 retains this OWF proof and Fiat–Shamir flow but replaces the vector commitment with a VSC-based GGM forest. Each process is explained in the following.

Random VOLE Commitment. The prover simulates a random VOLE protocol instead of actually invoking one with the verifier. By using the SoftSpoken VOLE protocol [27], the prover generates τ small-field VOLE correlations $(\mathbf{u}_i, \mathbf{v}_i)$ from GGM trees such that $\mathbf{q}_i = \Delta_i \cdot \mathbf{u}_i - \mathbf{v}_i$ for $i \in [\tau]$. Specifically, from each of the 2^k leaf nodes, vectors $\mathbf{r}_{i,j}$ are derived using a pseudorandom generator (PRG) and the prover obtains

$$\mathbf{u}_i = \sum_{j\in\mathbb{F}_{2^k}} \mathbf{r}_{i,j}, \qquad \mathbf{v}_i = \sum_{j\in\mathbb{F}_{2^k}} j \cdot \mathbf{r}_{i,j}.$$

Once the verifier learns all of the seeds except seed_{Δ_i} for some index Δ_i, it can compute

$$\mathbf{q}_i = \sum_{j\in\mathbb{F}_{2^k}} (\Delta_i - j)\cdot \mathbf{r}_{i,j} = \Delta_i \cdot \sum_{j\in\mathbb{F}_{2^k}} \mathbf{r}_{i,j} - \sum_{j\in\mathbb{F}_{2^k}} j \cdot \mathbf{r}_{i,j} = \Delta_i \cdot \mathbf{u}_i - \mathbf{v}_i,$$

where the verifier will know $(\Delta_i, \mathbf{q}_i)$. The small-field VOLE correlations are concatenated to form a large-field VOLE relation. Since each $\mathbf{u}_i$ is randomly generated, the prover generates the corrections $\mathbf{c}_i = \mathbf{u}_0 - \mathbf{u}_i$ for $i \in [\tau]^*$. Then, the prover can obtain $\mathbf{u}$ and $\mathbf{V}$ such that $\mathbf{Q} = \Delta \cdot \mathbf{u} - \mathbf{V}$ where $\mathbf{Q} = [\mathbf{q}_0 \| \cdots \| \mathbf{q}_{\tau-1}]$, $\mathbf{V} = [\mathbf{v}_0 \| \cdots \| \mathbf{v}_{\tau-1}]$, and $\Delta = [\Delta_0 \| \cdots \| \Delta_{\tau-1}]$ even though Δ and $\mathbf{Q}$ are not yet chosen. To prevent a manipulation of the GGM trees, the prover commits to the leaf nodes, which will be called *leaf commitments*. Although the leaf commitment does not work as a commitment scheme when we deal with vector semi-commitment, we retain the same terminology if the context is clear.

This sequence of algorithms is called VOLE commitment (VoleCom). It generates random VOLE correlations while delaying the choice of the sender's part and commits to the leaf nodes. The VSC version of VoleCom can be found in Algorithm 1. The corresponding verifier's function is called VOLE reconstruction (VoleRecon). This function will be explained in the "VOLE key selection" part.

VOLE Consistency Check. As aforementioned, to concatenate the small-field VOLE relations, they should be consistent (i.e., $\mathbf{u}_i + \mathbf{c}_i = \mathbf{u}_0$ for $i \in [\tau]^*$). The consistency is checked using a universal hash function (VHash). For a linear universal hash function H, if $\mathbf{Q} = \Delta \cdot \mathbf{u} - \mathbf{V}$ is a valid VOLE correlation, then it should satisfy that $H(\mathbf{Q}) = \Delta \cdot H(\mathbf{u}) - H(\mathbf{V})$, where H is applied in the column-major order. By putting $\widetilde{\mathbf{u}} = H(\mathbf{u})$ into the signature, the verifier can check the VOLE consistency.

After the VOLE consistency check, the prover generates a correction value $\mathbf{d} = \mathbf{w} - \mathbf{u}$ to the witness $\mathbf{w}$. By using this commitment, the verifier can update the random VOLE correlations to chosen VOLE correlations by

$$\mathbf{Q} \leftarrow \mathbf{Q} + \Delta \cdot \mathbf{d} = \Delta \cdot (\mathbf{u} + \mathbf{d}) - \mathbf{V} = \Delta \cdot \mathbf{w} - \mathbf{V}.$$

VOLE-ZK Proof. To prove knowledge of a preimage under a one-way function, we use the QuickSilver protocol [29], an interactive constant-round zero-knowledge proof system widely used in the VOLE-in-the-Head paradigm. To prove that a Boolean circuit C of t multiplication gates satisfies $C(\mathbf{w}) = 1$ for a witness $\mathbf{w} \in \mathbb{F}_2^n$, the prover and verifier invoke the standard subfield VOLE functionality $n + t$ times — once for each of the n input wires and the t multiplication-gate output wires — while addition gates incur zero communication. Concretely, for each such wire value w_i, the prover has a chosen VOLE correlation pair $(w_i, v_i) \in \mathbb{F}_2 \times \mathbb{F}_{2^k}$.

To verify the correctness of a multiplication gate indexed by (α, β, γ), the verifier computes a value $b_i = q_\alpha \cdot q_\beta - q_\gamma \cdot \Delta$, which expands to

$$b_i = v_\alpha \cdot v_\beta + (w_\alpha \cdot v_\beta + w_\beta \cdot v_\alpha - v_\gamma) \cdot \Delta + (w_\alpha \cdot w_\beta - w_\gamma) \cdot \Delta^2. \quad (2)$$

Here, b_i is linear in Δ if the prover computes w_γ correctly. To perform the verification, the prover provides $v_\alpha \cdot v_\beta$ and $(w_\alpha \cdot v_\beta + w_\beta \cdot v_\alpha - v_\gamma)$ to the verifier in a universal-hashed form (ZHash). The verifier then substitutes these values into Equation (2) and verifies that it holds.

For completeness, we write VOLE-ZK functions in OWFProve (Algorithm 6) and OWFVerify (Algorithm 7). These are essentially the degree-2 QuickSilver protocol; a higher degree version is compatible with our VSC constructions.

VOLE Key Selection. After all the commitments and proof are generated, the prover randomly selects the VOLE key Δ by hashing the transcript. Given the chosen Δ, a hidden seed in each GGM tree is selected. To send all seeds except the single hidden seed per GGM tree, the prover provides the partial decommitment pdecom_i which contains the copath and leaf commitment to the hidden seed.

From a given pdecom_i, the verifier can reconstruct the random VOLE correlation $(\mathbf{Q}, \Delta)$ by using VoleRecon. By computing $\mathbf{Q} \leftarrow \mathbf{Q} + \Delta \cdot \mathbf{d}$, the verifier can also reconstruct the chosen VOLE correlations. The VSC version can be found in Algorithm 2.

Universality of VHash and ZHash. Let $\delta > 0$, and let $H : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ be a keyed function for three non-empty sets $\mathcal{K}$, $\mathcal{X}$, and $\mathcal{Y}$. A linear function H is said to be *δ-almost universal (AU)* if for any nonzero $X \in \mathcal{X}$, $\Pr[K \leftarrow_\$ \mathcal{K} : H_K(X) = 0] \le \delta$. VHash and ZHash[1] are almost universal linear hash functions, and the almost universality of VHash and ZHash is shown in [4]; we denote their bounds by ε_{v} and $\varepsilon_{\mathrm{zk}}$, respectively.

3 Vector Semi-Commitment Revisited

Vector semi-commitment (VSC) was proposed to relax the concept of vector commitment (VC) in the context of MPCitH-based signature schemes [22]. A VSC scheme, similarly to VC schemes, abstracts the process of generating views and corresponding commitments. While VSC does not provide the full binding property, it reduces signature size without security degradation for some parameter choices.

Although Kim et al. [22] proposed VSC schemes based on both random oracles (RO-VSC) and an ideal cipher (IC-VSC), they do not cover the GGM-tree-based VCs currently in use, such as counter-mode constructions or generic PRG-based constructions. Instead, we analyze the properties that each primitive must possess to ensure the security of VSCs for a generic construction, thereby enabling the method to cover a variety of constructions and ideal primitive models. The generic construction of vector semi-commitment, on top of three functions $\mathsf{F}_{\mathsf{tree}}$, $\mathsf{F}_{\mathsf{com}}$, $\mathsf{F}_{\mathsf{tape}}$, is described in Figure 1.

3.1 Extractable Semi-binding and Multi-Instance Hiding Properties of VSC

The key difference between VSC and VC lies in the binding property. Informally, binding means it's hard to find multiple messages matching the same commitment. VSC relaxes this with extractable semi-binding: an adversary might find a few matching messages, but finding many is still hard.

[1]Original names of these functions in [4] are VOLEHash and ZKHash, respectively.

- **Parameters**: integer $N = 2^k$, IV length $|\mathsf{iv}|$, commitment length $|\mathsf{com}|$, bit-length of message ℓ, keyed functions $\mathsf{F}_{\mathsf{tree}}$, $\mathsf{F}_{\mathsf{com}}$ and $\mathsf{F}_{\mathsf{tape}}$ with key space $\{0,1\}^\lambda$ where
$$\mathsf{F}_f : \{0,1\}^\lambda \times \{0,1\}^{|\mathsf{iv}|} \times [N] \to \begin{cases} \{0,1\}^{\lambda\cdot 2} & \text{if } f = \mathsf{tree}, \\ \{0,1\}^{|\mathsf{com}|} & \text{if } f = \mathsf{com}, \\ \{0,1\}^{\ell} & \text{if } f = \mathsf{tape} \end{cases}$$
- $\mathsf{Commit}(\mathsf{iv}, \mathsf{root}; \ell)$:
  1. Set $\mathsf{node}_1 \leftarrow \mathsf{root}$
  2. For $i \in \{1, \dots, N-1\}$, set
  $$(\mathsf{node}_{2i}, \mathsf{node}_{2i+1}) \leftarrow \mathsf{F}_{\mathsf{tree}}(\mathsf{node}_i, \mathsf{iv}, i)$$
  3. For $i \in [N]$, set
  $$\mathsf{com}_i \leftarrow \mathsf{F}_{\mathsf{com}}(\mathsf{node}_{i+N}, \mathsf{iv}, i)$$
  $$m_i \leftarrow \mathsf{F}_{\mathsf{tape}}(\mathsf{node}_{i+N}, \mathsf{iv}, i)$$
  4. Output $(\overline{\mathsf{com}}, \mathsf{decom}, (m_i)_{i\in[N]})$ where
  $$\overline{\mathsf{com}} = (\mathsf{com}_0, \dots, \mathsf{com}_{N-1})$$
  $$\mathsf{decom} = (\mathsf{node}_1, \dots, \mathsf{node}_{2N-1})$$
- $\mathsf{Open}(\mathsf{iv}, \mathsf{decom}, \Delta)$:
  1. For $e \in [k]$: set $i_e \leftarrow \lfloor (\Delta + N)/2^e \rfloor \oplus 1$
  2. Output $\mathsf{pdecom} = (\mathsf{path}_\Delta, \mathsf{com}_\Delta)$ where
  $$\mathsf{path}_\Delta = (\mathsf{node}_{i_0}, \dots, \mathsf{node}_{i_{k-1}})$$
- $\mathsf{Recon}(\mathsf{iv}, \mathsf{pdecom} = (\mathsf{path}_\Delta, \mathsf{com}_\Delta), \Delta; \ell)$:
  1. Similarly to Steps 2 and 3 in Commit, expand each node in path_Δ and get $(\mathsf{com}_j, m_j)_{j\in[N]\setminus\{\Delta\}}$
  2. Set $m_\Delta = \perp$
  3. Output $((m_i)_{i\in[N]}, \overline{\mathsf{com}} = (\mathsf{com}_0, \dots, \mathsf{com}_{N-1}))$

Figure 1: Generic construction of VSC from F-functions.

The formal definitions of the (semi-)binding and hiding properties of VSC are given below; all of them are adopted from the definitions in [22].

Definition 3.1 (Extractable Semi-binding). Let VSC be a vector semi-commitment scheme, which is built on top of ideal primitive $P \in \mathcal{P}$. The extractable semi-binding security of VSC, which is parameterized by positive integers Q_{com}, Q_P and u, is defined by

$$\mathbf{Adv}^{\mathsf{esb}}_{\mathsf{VSC}}(Q_{\mathsf{com}}, Q_P, u) \stackrel{\mathrm{def}}{=} \max_{\mathcal{A}} \Pr\left[\mathrm{G}^{\mathsf{esb}}_{\mathsf{VSC}}(\mathcal{A}, Q_{\mathsf{com}}, Q_P, u) \Rightarrow 1\right]$$

where $\mathrm{G}^{\mathsf{esb}}_{\mathsf{VSC}}$ is described in Figure 2.

Definition 3.2 (Multi-Instance Hiding). Let VSC be a vector semi-commitment scheme, which is built on top of ideal primitive $P \in \mathcal{P}$. The multi-instance hiding security of VSC, which is parameterized by positive integers Q_I and Q_P, is defined by

$$\mathbf{Adv}^{\mathsf{mih}}_{\mathsf{VSC}}(Q_I, Q_P) \stackrel{\mathrm{def}}{=} \max_{\mathcal{A}} \left| 2\Pr\left[\mathrm{G}^{\mathsf{mih}}_{\mathsf{VSC}}(\mathcal{A}, Q_I, Q_P) \Rightarrow 1\right] - 1 \right|$$

where $\mathrm{G}^{\mathsf{mih}}_{\mathsf{VSC}}$ is described in Figure 3.

Game $\mathrm{G}^{\mathrm{esb}}_{\mathrm{VSC}}(\mathcal{A}, Q_{\mathrm{com}}, Q_P, u)$:

$P \leftarrow_{\$} \mathcal{P}$; $q \leftarrow 0$; $p \leftarrow 0$; $\mathcal{L}_P \leftarrow \emptyset$

$(i, \mathrm{pdecom}, \Delta) \leftarrow \mathcal{A}^{\mathrm{Com,P}}(Q_{\mathrm{com}}, Q_P, u)$

If $\mathcal{E}_{i,\Delta} = \perp$: **return** 1

$(\mathbf{m}, \overline{\mathsf{com}}) \leftarrow \mathsf{VSC.Recon}^P(\mathsf{iv}_i, \mathrm{pdecom}, \Delta)$

If $\overline{\mathsf{com}}_i \neq \overline{\mathsf{com}}$: **return** 0 // Verifier rejects

return $\mathbf{m} \neq \perp \wedge \mathbf{m} \notin \mathcal{E}_{i,\Delta}$

Oracle $\mathrm{Com}(\mathsf{iv}, \overline{\mathsf{com}})$:

If $q \geq Q_{\mathrm{com}}$: **return** $\perp$

$\mathsf{iv}_q \leftarrow \mathsf{iv}$; $\overline{\mathsf{com}}_q \leftarrow \overline{\mathsf{com}}$

For $\Delta \in [N]$:

$\mathcal{E}_{q,\Delta} \leftarrow \mathsf{Ext}(\mathsf{iv}, \overline{\mathsf{com}}, \Delta, \mathcal{L}_P)$

$q \leftarrow q + 1$

Oracle $\mathrm{P}(x)$:

If $p \geq Q_P$: **return** $\perp$

$\mathcal{L}_P \leftarrow \mathcal{L}_P \cup \{(x, P(x))\}$

$p \leftarrow p + 1$

return $P(x)$

Procedure $\mathsf{Ext}(\mathsf{iv}, \overline{\mathsf{com}}, \Delta, \mathcal{L}_P)$:

Parse $\overline{\mathsf{com}} = (\mathsf{com}_0, \ldots, \mathsf{com}_{N-1})$

Compute $(\mathcal{S}^{(1)}_a)_{a\in[N]}$ where

$$\mathcal{S}^{(1)}_a = \left\{s \in \{0,1\}^\lambda : \mathsf{F}_{\mathsf{com}}(s, \mathsf{iv}, a) = \mathsf{com}_a\right\}$$

by brute-force search on $\mathsf{F}_{\mathsf{com}}$ using P

Compute $(\mathcal{S}^{(2)}_b)_{b\in[N/2]}$ where

$$\mathcal{S}^{(2)}_b = \left\{(y_0, y_1) : \begin{array}{c}\mathsf{F}_{\mathsf{tree}}(x, \mathsf{iv}, b + N/2) = (y_0, y_1) \\ \text{can be computed by } \mathcal{L}_P\end{array}\right\} \cap \left(\mathcal{S}^{(1)}_{2b} \times \mathcal{S}^{(1)}_{2b+1}\right)$$

If $|\mathcal{S}^{(1)}_a| > u$ for some $a \in [N]$: **Return** $\perp$

If $|\mathcal{S}^{(2)}_b| \geq 2$ for some $b \in [N/2]$: **Return** $\perp$

Compute the following:

$$\mathcal{S} \leftarrow \left\{(s_i)_{i\in[N]} : \begin{array}{c}s_\Delta = \perp,\ s_{\Delta\oplus 1} \in \mathcal{S}^{(1)}_{\Delta\oplus 1}, \\ (s_{2b}, s_{2b+1}) \in \mathcal{S}^{(2)}_b \text{ if } \Delta \notin \{2b, 2b+1\}\end{array}\right\}$$

$$\mathcal{E} \leftarrow \left\{(m_i)_{i\in[N]} : \begin{array}{c}m_\Delta = \perp \text{ and } \exists (s_i)_{i\in[N]} \in \mathcal{S} \text{ such that} \\ m_i = \mathsf{F}_{\mathsf{tape}}(s_i, \mathsf{iv}, i) \text{ if } i \neq \Delta\end{array}\right\}$$

Return $\mathcal{E}$ // $|\mathcal{E}| \leq u$.

Figure 2: Extractable semi-binding game $\mathrm{G}^{\mathrm{esb}}_{\mathrm{VSC}}(\mathcal{A}, Q_{\mathrm{com}}, Q_P, u)$ for a VSC based on an ideal primitive P.

3.2 Required Conditions for F-functions

In this section, we describe requirements for F to ensure the security of the VSC.

Definition 3.3. Let $Q_I, Q_P, u, k, |\mathsf{iv}|$ be positive integers and $N = 2^k$. Let $\mathsf{F} = (\mathsf{F}_{\mathsf{tree}}, \mathsf{F}_{\mathsf{com}}, \mathsf{F}_{\mathsf{tape}})$ be the keyed functions in VSC on top of ideal primitive P. Consider the following game with stateful adversary $\mathcal{A}$, which can make oracle queries to P at most Q_P times.

Game $\mathrm{G}^{\mathrm{mih}}_{\mathrm{VSC}}(\mathcal{A}, Q_I, Q_P)$:

$b^* \leftarrow_{\$} \{0,1\}$

$P \leftarrow_{\$} \mathcal{P}$; $q \leftarrow 0$; $p \leftarrow 0$

$b \leftarrow \mathcal{A}^{\mathrm{Get,P}}(Q_I, Q_P)$

return $b^* = b$

Oracle $\mathrm{P}(x)$:

If $p \geq Q_P$: **return** $\perp$

$p \leftarrow p + 1$

return $P(x)$

Oracle Get() :

If $q \geq Q_I$: **return** $\perp$

$\mathsf{iv} \leftarrow_{\$} \{0,1\}^{|\mathsf{iv}|}$; $q \leftarrow q + 1$

For $i \in [\tau]$:

$\mathsf{root}_i \leftarrow_{\$} \{0,1\}^\lambda$; $\Delta_i \leftarrow_{\$} [N]$

$(\overline{\mathsf{com}}_i, \mathsf{decom}_i, (m_{i,j})_{j\in[N]}) \leftarrow \mathsf{VSC.Commit}(\mathsf{iv}, \mathsf{root}_i)$

$\mathrm{pdecom}_i \leftarrow \mathsf{VSC.Open}(\mathsf{iv}, \mathsf{decom}_i, \Delta_i)$

If $b^* = 1$: $m_{i,\Delta_i} \leftarrow_{\$} \{0,1\}^\ell$

return $(\mathsf{iv}, ((m_{i,j})_{j\in[N]}, \mathrm{pdecom}_i, \Delta_i)_{i\in[\tau]})$

Figure 3: Multi-instance hiding game $\mathrm{G}^{\mathrm{mih}}_{\mathrm{VSC}}(\mathcal{A}, Q_I, Q_P)$ for a VSC based on an ideal primitive P.

(1) $(\mathsf{iv}_i, (z_{i,j})_{j\in[N]})_{i\in[Q_I]} \leftarrow \mathcal{A}^P(Q_I, Q_P, u)$

(2) Set $\mathsf{win}_{\mathsf{mcoll}} = \mathsf{true}$ if

$$\max_{i\in[Q_I], j\in[N]} \left|\left\{s \in \{0,1\}^\lambda : \mathsf{F}_{\mathsf{com}}(s, \mathsf{iv}_i, j) = z_{i,j}\right\}\right| > u.$$

(3) Set $\mathsf{win}_{\text{4-xor}} = \mathsf{true}$ if there exist $s_0, s_1, s'_0, s'_1 \in \{0,1\}^\lambda$, $i \in [Q_I]$ and $j_0, j_1 \in [N]$ such that $(s_0, s_1) \neq (s'_0, s'_1)$ and

$$\begin{cases}\bigwedge_{a\in[2]} \mathsf{F}_{\mathsf{com}}(s_a, \mathsf{iv}_i, j_a) = \mathsf{F}_{\mathsf{com}}(s'_a, \mathsf{iv}_i, j_a) = z_{i,j_a}, \\ \bigoplus_{a\in[2]} \mathsf{F}_{\mathsf{tape}}(s_a, \mathsf{iv}_i, j_a) = \bigoplus_{a\in[2]} \mathsf{F}_{\mathsf{tape}}(s'_a, \mathsf{iv}_i, j_a)\end{cases}$$

We consider another collision finding game in the same setting.

(1) $(\mathsf{iv}, x, x', i) \leftarrow \mathcal{A}^P(Q_P)$

(2) Set $\mathsf{win}_{\mathsf{coll2}} = \mathsf{true}$ if $x \neq x'$ and there exist $y_0, y_1, y'_0, y'_1 \in \{0,1\}^\lambda$ such that

$$\begin{cases}\mathsf{F}_{\mathsf{tree}}(x, \mathsf{iv}, i + N/2) = (y_0, y_1), \\ \mathsf{F}_{\mathsf{tree}}(x', \mathsf{iv}, i + N/2) = (y'_0, y'_1), \\ \mathsf{F}_{\mathsf{com}}(y_0, \mathsf{iv}, 2i) = \mathsf{F}_{\mathsf{com}}(y'_0, \mathsf{iv}, 2i), \\ \mathsf{F}_{\mathsf{com}}(y_1, \mathsf{iv}, 2i + 1) = \mathsf{F}_{\mathsf{com}}(y'_1, \mathsf{iv}, 2i + 1).\end{cases}$$

Next, we consider an IV-based PRG game in the same setting.

(1) $b \leftarrow_{\$} \{0,1\}$.

(2) $b' \leftarrow \mathcal{A}^{\mathrm{Get},P}(Q_I, Q_P)$

(3) Set $\mathsf{win}_{\mathsf{prg}} = \mathsf{true}$ if $b = b'$

where $\mathcal{A}$ can query Get at most Q_I times and $\mathrm{Get}(f, i)$ returns (iv, y) for

$$\mathsf{iv} \leftarrow_{\$} \{0,1\}^{|\mathsf{iv}|}, k \leftarrow_{\$} \{0,1\}^\lambda, y \leftarrow \begin{cases}\mathsf{F}_f(k, \mathsf{iv}, i) & \text{if } b = 0; \\ \leftarrow_{\$} \{0,1\}^{|\mathsf{F}_f(k,\mathsf{iv},i)|} & \text{otherwise}\end{cases}$$

From the above games, we define the following security definitions.

$$\mathbf{Adv}^{\mathsf{mcoll}}_{\mathsf{F}}(Q_I, Q_P, u) = \max_{\mathcal{A}} \Pr\left[\mathsf{win}_{\mathsf{mcoll}} = \mathsf{true}\right],$$
$$\mathbf{Adv}^{\mathsf{coll2}}_{\mathsf{F}}(Q_P) = \max_{\mathcal{A}} \Pr\left[\mathsf{win}_{\mathsf{coll2}} = \mathsf{true}\right],$$
$$\mathbf{Adv}^{\text{4-xor}}_{\mathsf{F}}(Q_I, Q_P) = \max_{\mathcal{A}} \Pr\left[\mathsf{win}_{\text{4-xor}} = \mathsf{true}\right],$$
$$\mathbf{Adv}^{\mathsf{prg}}_{\mathsf{F}}(Q_I, Q_P) = \max_{\mathcal{A}} 2\left|\Pr\left[\mathsf{win}_{\mathsf{prg}} = \mathsf{true}\right] - \frac{1}{2}\right|.$$

The security of VSC is described by the following lemmas. Since proofs are relatively simple, we give proof sketches.

Lemma 3.4. *Let q, p, u, k be positive integers and $N = 2^k$. Then, we have*

$$\mathbf{Adv}^{\mathsf{esb}}_{\mathsf{VSC}}(q, p, u) \le \mathbf{Adv}^{\mathsf{coll2}}_{\mathsf{F}}(p) + \mathbf{Adv}^{\mathsf{mcoll}}_{\mathsf{F}}(q, p, u).$$

Proof (Sketch). It is easy to see that if Ext outputs $\perp$ at any point, $\mathcal{A}$ can also break the mcoll or coll2 security of F. Now assume that Ext does not output $\perp$. Since Ext searches all possibilities of $\mathsf{F}_{\mathsf{com}}$ in a brute-force manner, $\mathcal{A}$ wins the game only if it breaks coll2 security of F after extraction. Therefore, we have

$$\mathbf{Adv}^{\mathsf{esb}}_{\mathsf{VSC}}(q, p, u) \le \mathbf{Adv}^{\mathsf{coll2}}_{\mathsf{F}}(p) + \mathbf{Adv}^{\mathsf{mcoll}}_{\mathsf{F}}(q, p, u)$$

which concludes the proof. □

Lemma 3.5. *Let p, q, k, τ be positive integers and $N = 2^k$. Then, we have*

$$\mathbf{Adv}^{\mathsf{mih}}_{\mathsf{VSC}}(q, p) \le \tau(k+2) \cdot \mathbf{Adv}^{\mathsf{prg}}_{\mathsf{F}}(q, p).$$

Proof (Sketch). We randomly replace nodes one by one along the path from the root of the tree to the hidden leaf, while computing VSC.Commit(iv, root). For each Get oracle, there are τ trees to be expanded, and for each tree, there are k nodes to be replaced. Also, since the leaf commitment and the random tape of the hidden leaf node are also randomly replaced, this replacement can be done at cost

$$\tau(k+2)\mathbf{Adv}^{\mathsf{prg}}_{\mathsf{F}}(q, p).$$

After replacement, the distribution of hidden message (m_{i,Δ_i} in Figure 3) is truly random, which concludes the proof. □

Remark. We can also consider VSC schemes that are not based on ideal primitives. In this case, the number of queries for the ideal primitive represented by Q_P is replaced by the time resources allocated to $\mathcal{A}$, and the security definitions (Definition 3.3) of F_f for $f \in \{\mathsf{tree}, \mathsf{com}, \mathsf{tape}\}$ become computational assumptions of those functions. For example, in the case of FAEST-EM, $\mathsf{F}_{\mathsf{com}}$ is AES-CTR and the following computational assumption is required:

$$\mathbf{Adv}^{\mathsf{mcoll}}_{\mathsf{F}_{\mathsf{com}}}(\mathcal{A}, Q_I, t, 1) \le \mathrm{negl}(\lambda)$$

for a polynomially large number of queries Q_I and time t.

3.3 Instantiations

The main difference between vector commitment and vector semi-commitment lies in the relaxed security requirements of F, particularly for $\mathsf{F}_{\mathsf{com}}$. Therefore, it is natural to consider reducing the commit size $|\mathsf{com}|$ in the existing VC. We present four instantiations of F with $|\mathsf{com}| = \lambda$. Let E be an ideal cipher and let $\langle i, a\rangle$ denote the λ-bit encoding of (i, a).

F^{ro}: $\mathsf{F}^{\mathsf{ro}}_{\mathsf{tree}}$, $\mathsf{F}^{\mathsf{ro}}_{\mathsf{com}}$, and $\mathsf{F}^{\mathsf{ro}}_{\mathsf{tape}}$ are independent random oracles.

$\mathsf{F}^{\mathsf{ctr}}$: set the seed as a secret key and domain separation as a cipher input. It requires $|\mathsf{iv}| = |\mathsf{com}|$, and $\mathsf{E} : \{0,1\}^\lambda \times \{0,1\}^{|\mathsf{iv}|} \to \{0,1\}^{|\mathsf{iv}|}$. Concretely,

$$\mathsf{F}^{\mathsf{ctr}}_{\mathsf{tree}}(s, \mathsf{iv}, i) = E_s(\mathsf{iv} \oplus \langle i, 0\rangle) \,\|\, E_s(\mathsf{iv} \oplus \langle i, 1\rangle)$$
$$\mathsf{F}^{\mathsf{ctr}}_{\mathsf{com}}(s, \mathsf{iv}, i) = E_s(\mathsf{iv} \oplus \langle i, 2\rangle)$$
$$\mathsf{F}^{\mathsf{ctr}}_{\mathsf{tape}}(s, \mathsf{iv}, i) = E_s(\mathsf{iv} \oplus \langle i, 3\rangle) \,\|\, \cdots \,\|\, E_s(\mathsf{iv} \oplus \langle i, \lceil \ell/\lambda \rceil + 2\rangle).$$

F^{dm}: set seed as a cipher input and domain separation as a secret key. It requires $|\mathsf{com}| = \lambda$ and $\mathsf{E} : \{0,1\}^{|\mathsf{iv}|} \times \{0,1\}^\lambda \to \{0,1\}^\lambda$ and

$$\mathsf{F}^{\mathsf{dm}}_{\mathsf{tree}}(s, \mathsf{iv}, i) = (E_{\mathsf{iv}\oplus\langle i,0\rangle}(s) \oplus s) \,\|\, (E_{\mathsf{iv}\oplus\langle i,1\rangle}(s) \oplus s)$$
$$\mathsf{F}^{\mathsf{dm}}_{\mathsf{com}}(s, \mathsf{iv}, i) = (E_{\mathsf{iv}\oplus\langle i,2\rangle}(s) \oplus s)$$
$$\mathsf{F}^{\mathsf{dm}}_{\mathsf{tape}}(s, \mathsf{iv}, i) = (E_{\mathsf{iv}\oplus\langle i,3\rangle}(s) \oplus s) \,\|\, \cdots \,\|\, (E_{\mathsf{iv}\oplus\langle i,\lceil \ell/\lambda \rceil + 2\rangle}(s) \oplus s).$$

$\mathsf{F}^{\mathsf{prg}}$: the oracles $\mathsf{F}^{\mathsf{prg}}_{\mathsf{tree}}$, $\mathsf{F}^{\mathsf{prg}}_{\mathsf{com}}$ and $\mathsf{F}^{\mathsf{prg}}_{\mathsf{tape}}$ are iv-based tweakable PRGs with proper domain separation.

For ideal-primitive-based constructions with $|\mathsf{com}| = |\mathsf{iv}| = \lambda$, the security of F is given by the following lemma while its proof is postponed to Appendix F.

Lemma 3.6. *We have the following for $\mathsf{F} \in \left\{\mathsf{F}^{\mathsf{ro}}, \mathsf{F}^{\mathsf{ctr}}, \mathsf{F}^{\mathsf{dm}}\right\}$,*

$$\mathbf{Adv}^{\mathsf{mcoll}}_{\mathsf{F}}(Q_I, Q_P, u) \le \frac{Q_I N + Q_P}{e^2} \cdot \left(\frac{2e}{u}\right)^u \cdot 2^\lambda$$
$$\mathbf{Adv}^{\mathsf{coll2}}_{\mathsf{F}}(Q_P) \le \frac{64N^2Q_P^2}{2^{2\lambda}}$$
$$\mathbf{Adv}^{\text{4-xor}}_{\mathsf{F}}(Q_I, Q_P) \le 64(Q_I + Q_P)N^2 \cdot \left(\frac{2}{2^\lambda}\right)^{\lfloor \ell/\lambda \rfloor - 2}$$
$$\mathbf{Adv}^{\mathsf{prg}}_{\mathsf{F}}(Q_I, Q_P) \le \frac{Q_I(\lceil \ell/\lambda \rceil + 3)^2}{2^\lambda} + \frac{Q_I^2(\lceil \ell/\lambda \rceil + 3)^2}{2^{2\lambda}} + \frac{2Q_I \cdot Q_P(\lceil \ell/\lambda \rceil + 3)}{2^{2\lambda}}$$

provided that $Q_I N + Q_P \le 2^{\lambda-1}$ and $|\mathsf{iv}| = |\mathsf{com}| = \lambda$.

Note that we only give non-tight results (especially for F^{ro}) for simplicity. For tight results, one can refer to the proof.

4 Application of VSC to VOLEitH

In this section, we introduce a VSC-based version of the VOLE-in-the-Head paradigm, dubbed *reduced VOLEitH (rVOLEitH)*. Essentially, the difference between VOLEitH and rVOLEitH is the replacement of VC by VSC. We present the major algorithms from VOLEitH in this section, while minor algorithms are written in Appendix E for completeness. In the description of rVOLEitH, we refer to the notations of FAEST [4] for readability, while some notations are different. We assume $k_0 = k_1$ and explain with a single k when addressing the specification and security proof for readability, while we use mixed tree heights $k_0 \neq k_1$ when evaluating performance.

4.1 From VSC to VOLE Commitment

In the context of VOLEitH-based signature schemes, messages generated from a vector commitment are used to generate a VOLE

pair. The VOLE pair will be used in the ZK protocol. VoleCom (Algorithm 1) commits to the VOLE pair using VSC, and VoleRecon (Algorithm 2) reconstructs a shifted VOLE pair from a partial decommitment. The main difference is the replacement of VC by VSC. Application of VSC reduces the size of the leaf commitment $\overline{\mathsf{com}}_{i,j}$ from 2λ (commitments from random oracle) or 3λ (commitments from PRG) bits to λ bits. Apart from this replacement, the two procedures follow those of FAEST. There is one minor interface change. In FAEST, ToVOLE takes leaf seeds and expands them internally. Here, VSC.Commit and VSC.Recon already return the messages obtained using $\mathsf{F}_{\mathsf{tape}}$, so ToVOLE takes these messages directly and consists only of XOR operations.

For a complete vector $(m_a)_{a\in[2^k]}$, ToVOLE computes

$$\mathbf{u} = \bigoplus_{a\in[2^k]} m_a \quad \text{and} \quad \mathbf{V}[t] = \bigoplus_{\substack{a\in[2^k] \\ \text{the } t\text{-th bit of } a \text{ is } 1}} m_a \quad \text{for } t \in [k].$$

During reconstruction, VSC.Recon sets $m'_{i,\Delta_i} = \bot$ at the hidden index Δ_i, which need not be zero. Before calling ToVOLE, VoleRecon reorders the messages as $m_{i,j} = m'_{i,j\oplus\Delta_i}$. This puts the hidden entry at index zero, since $m_{i,0} = m'_{i,\Delta_i} = \bot$. If $(\delta_{i,t})_{t\in[k]}$ is the bit representation of Δ_i, the reordering gives $\mathbf{q}_{i,t} = \mathbf{v}_{i,t} \oplus \delta_{i,t}\mathbf{u}_i$ for every $t \in [k]$. The proof of Lemma 4.1 uses these XOR expressions for $\mathbf{u}$ and $\mathbf{V}[t]$.

Algorithm 1. VoleCom$(\mathsf{iv}, (\mathsf{root}_i)_{i\in[\tau]}; \ell)$

for $i \in [\tau]$ **do**
  $(\overline{\mathsf{com}}_i, \mathsf{decom}_i, (m_{i,j})_{j\in[2^k]}) \leftarrow \mathsf{VSC.Commit}(\mathsf{iv}, \mathsf{root}_i)$
  $(\mathbf{u}_i, \mathbf{v}_{i,0}, \ldots, \mathbf{v}_{i,k-1}) \leftarrow \mathsf{ToVOLE}(m_{i,0}, \ldots, m_{i,2^k-1})$
  $\mathbf{V}_i \leftarrow [\mathbf{v}_{i,0} \ldots \mathbf{v}_{i,k-1}]$
$\mathbf{V} \leftarrow [\mathbf{V}_0 \ldots \mathbf{V}_{\tau-1}] \in \mathbb{F}_2^{\ell\times\lambda}$, $\mathbf{u} \leftarrow \mathbf{u}_0$
for $i \in \{1, \ldots, \tau-1\}$ **do**
  $\mathbf{c}_i \leftarrow \mathbf{u} \oplus \mathbf{u}_i$
return $((\overline{\mathsf{com}}_i)_{i\in[\tau]}, (\mathsf{decom}_i)_{i\in[\tau]}, \mathbf{c}_1, \ldots, \mathbf{c}_{\tau-1}, \mathbf{u}, \mathbf{V})$

Algorithm 2. VoleRecon$(\mathsf{iv}, \mathsf{chall}, (\mathsf{pdecom}_i)_{i\in[\tau]}; \ell)$

for $i \in [\tau]$ **do**
  $\Delta_i \leftarrow \mathsf{chall}[ki : k(i+1)]$
  $((m'_{i,j})_{j\in[2^k]}, \overline{\mathsf{com}}_i) \leftarrow \mathsf{VSC.Recon}(\mathsf{iv}, \mathsf{pdecom}_i, \Delta_i; \ell)$
for $i \in [\tau]$ **do**
  for $j \in [2^k]$ **do**
    $m_{i,j} \leftarrow m'_{i,j\oplus\Delta_i}$
  $(\mathbf{u}'_i, \mathbf{q}_{i,0}, \ldots, \mathbf{q}_{i,k-1}) \leftarrow \mathsf{ToVOLE}(\bot, m_{i,1}, \ldots, m_{i,2^k-1})$
  $\mathbf{Q}_i \leftarrow [\mathbf{q}_{i,0} \ldots \mathbf{q}_{i,k-1}]$
$\mathbf{Q} = [\mathbf{Q}_0 \ldots \mathbf{Q}_{\tau-1}]$
return $((\overline{\mathsf{com}}_i)_{i\in[\tau]}, \mathbf{Q})$

Unfortunately, VoleCom itself is not binding when using vector semi-commitments. Therefore, we define a useful helper function, dubbed VoleComChk, which is a composition of VoleCom with a VOLE consistency check using VHash. The main point of this function is that it is hard for an adversary to find a VOLE pair which matches $(\mathsf{chall}, \widetilde{\mathbf{u}}, \widetilde{\mathbf{V}})$ unless the adversary generated it legitimately from $(\mathbf{u}, \mathbf{V})$. Now we can define the extractable binding property of VoleComChk. This property ensures that for a given output including VHash outputs of VoleComChk, there is only one VOLE pair which is consistent with the output. This property will be used to prove EUF-KO security.

Algorithm 3. ToVOLE$(m_0, \ldots, m_{2^k-1})$

for $i \in [2^k]$ **do**
  if $m_i = \bot, \mathbf{r}_{0,i} \leftarrow 0^\ell$; **else** $\mathbf{r}_{0,i} \leftarrow m_i$
$\mathbf{v}_0, \ldots, \mathbf{v}_{k-1} \leftarrow 0^\ell$
for $j \in [k]$ **do**
  for $i \in [2^{k-j-1}]$ **do**
    $\mathbf{v}_j \leftarrow \mathbf{v}_j \oplus \mathbf{r}_{j,2i+1}$
    $\mathbf{r}_{j+1,i} \leftarrow \mathbf{r}_{j,2i} \oplus \mathbf{r}_{j,2i+1}$
$\mathbf{u} \leftarrow \mathbf{r}_{k,0}$; $\mathbf{V} \leftarrow [\mathbf{v}_0 \ldots \mathbf{v}_{k-1}]$
return $(\mathbf{u}, \mathbf{V})$

Lemma 4.1 (Extractable binding). *Let* VoleCom *be built on top of a vector semi-commitment scheme based on ideal primitive $P \in \mathcal{P}$. Let $\mathsf{H}_{\mathsf{chall}}$ be a random oracle and assume that the key length of* VHash *is at least 2λ. Then, for any arbitrary adversary $\mathcal{A}$, we have*

$$\begin{aligned}&\Pr\left[\mathrm{G}^{\mathsf{eb}}_{\mathsf{VoleComChk}}(\mathcal{A}, Q_{\mathsf{com}}, Q_{\mathsf{chall}}, Q_P) \Rightarrow 1\right] \\ &\le \frac{Q^2_{\mathsf{chall}}}{2^{2\lambda}} + \frac{Q_{\mathsf{chall}} N^2(\tau^2u^4 + 8\tau u^2)\varepsilon_{\mathsf{v}}}{8} + \mathbf{Adv}^{4\text{-xor}}_{\mathsf{F}}(Q_{\mathsf{com}}, Q_P) \\ &\quad + \mathbf{Adv}^{\mathsf{coll2}}_{\mathsf{F}}(Q_P) + \mathbf{Adv}^{\mathsf{mcoll}}_{\mathsf{F}}(Q_{\mathsf{com}}, Q_P, u)\end{aligned}$$

where $\mathrm{G}^{\mathsf{eb}}_{\mathsf{VoleComChk}}$ is described in Figure 4.

The next property of VoleCom is the multi-hiding property, which is semantically the same as the property having the same name in [2]. At a high level, the desired feature of both properties is the same; a VOLE pair generated from VoleCom is indistinguishable from the uniform distribution (up to VOLE correlation). The multi-hiding security of VoleCom is described by the following lemma, while the proof is omitted since it is a simple application of multi-instance hiding property of VSC.

Lemma 4.2 (Multi-hiding). *Let* VoleCom *be built on top of a vector semi-commitment scheme based on ideal primitive $P \in \mathcal{P}$. Then, for any arbitrary adversary $\mathcal{A}$, we have*

$$\left|2\Pr\left[\mathrm{G}^{\mathsf{mh}}_{\mathsf{VoleCom}}(\mathcal{A}, Q_I, Q_P) \Rightarrow 1\right] - 1\right| \le \mathbf{Adv}^{\mathsf{mih}}_{\mathsf{VSC}}(Q_I, Q_P)$$

where $\mathrm{G}^{\mathsf{mh}}_{\mathsf{VoleCom}}$ is described in Figure 5.

4.2 Specification of rVOLEitH

In this section, we explain which parts of rVOLEitH are changed from VOLEitH. As the Sign and Verify algorithms do not differ substantially from those of VOLEitH, we provide the full algorithms in Appendix E.1. To summarize, there are three major changes: VoleCom/VoleRecon, functions related to the one-way function (OWF), and the use of the grinding technique.

The VoleCom and VoleRecon algorithms are provided in Algorithm 1 and Algorithm 2, respectively. Apart from replacing VC by

Game $\mathrm{G}^{\mathrm{eb}}_{\mathrm{VoleComChk}}(\mathcal{A}, Q_{\mathsf{com}}, Q_{\mathsf{chall}}, Q_P)$:

$P \leftarrow_{\$} \mathcal{P};\ p \leftarrow 0;\ q \leftarrow 0;\ r \leftarrow 0;\ \mathcal{L}_{\mathsf{chall}} \leftarrow \emptyset;\ \mathcal{L}_P \leftarrow \emptyset$

$\begin{pmatrix} a, \Delta, \mathsf{pdecom}, \\ \mu, (\mathbf{c}_i)_{i\in[\tau]^*}, \mathsf{iv} \end{pmatrix} \leftarrow \mathcal{A}^{\mathrm{Com},\mathrm{H}_{\mathsf{chall}},\mathrm{P}}(Q_{\mathsf{com}}, Q_{\mathsf{chall}}, Q_P)$ // $a \in [q]$

$(\mathsf{com}, \mathbf{Q}) \leftarrow \mathsf{VoleRecon}(\mathsf{iv}, \Delta, \mathsf{pdecom})$

$\mathsf{chall} \leftarrow \mathrm{H}_{\mathsf{chall}}(\mu \,\|\, \mathsf{com} \,\|\, \mathbf{c}_1 \,\|\, \cdots \,\|\, \mathbf{c}_{\tau-1} \,\|\, \mathsf{iv})$

If $\mathsf{chall}_a \neq \mathsf{chall}$: **return** 0

For $i \in [\tau]^*, j \in [k]$: // Check VOLE consistency

$\quad \mathbf{Q}[ik+j] \leftarrow \mathbf{Q}[ik+j] \oplus \Delta[ik+j] \cdot \mathbf{c}_i$

If $\widetilde{\mathbf{V}}_a \oplus \Delta \cdot \widetilde{\mathbf{u}}_a \neq \mathsf{VHash}(\mathsf{chall}, \mathbf{Q})$: **return** 0

return $(\mathbf{u}^*_a, \mathbf{V}^*_a) = \bot \vee \mathbf{V}^*_a \oplus \Delta \cdot \mathbf{u}^*_a \neq \mathbf{Q}$ // Check extraction

Oracle $\mathrm{Com}(\mathsf{chall}, \widetilde{\mathbf{u}}, \widetilde{\mathbf{V}})$:

If $q \geq Q_{\mathsf{com}}$: **return** $\bot$

$(\mathbf{u}^*, \mathbf{V}^*)$ or $\bot \leftarrow \mathsf{Ext}(\mathsf{chall}, \widetilde{\mathbf{u}}, \widetilde{\mathbf{V}}, \mathcal{L}_{\mathsf{chall}}, \mathcal{L}_P)$

$(\mathsf{chall}_q, \widetilde{\mathbf{u}}_q, \widetilde{\mathbf{V}}_q, \mathbf{u}^*_q, \mathbf{V}^*_q) \leftarrow (\mathsf{chall}, \widetilde{\mathbf{u}}, \widetilde{\mathbf{V}}, \mathbf{u}^*, \mathbf{V}^*)$

$q \leftarrow q + 1$

Oracle $\mathrm{H}_{\mathsf{chall}}(x)$:

If $r \geq Q_{\mathsf{chall}}$: **return** $\bot$

$r \leftarrow r + 1;\ y \leftarrow \mathrm{H}_{\mathsf{chall}}(x)$

$\mathcal{L}_{\mathsf{chall}} \xleftarrow{\cup} \{(x, y)\}$

return y

Oracle $\mathrm{P}(x)$:

If $p \geq Q_P$: **return** $\bot$

$\mathcal{L}_P \leftarrow \mathcal{L}_P \cup \{(x, P(x))\}$

$p \leftarrow p + 1$

return $P(x)$

Procedure Ext: Defined in the proof of Lemma 4.1

Figure 4: Extractable binding game $\mathrm{G}^{\mathrm{eb}}_{\mathrm{VoleComChk}}$ based on random oracle $\mathrm{H}_{\mathsf{chall}}$ and ideal primitive P.

VSC and the resulting change to the ToVOLE interface explained in Section 4.1, their structure is unchanged from FAEST.

We generalize the OWF-proof functions from the AES-specific proof functions. The purpose of this change is to show that our VSC works regardless of the choice of OWF. These functions are not essential in this paper, and we use the original construction as a black box. To be clear, we abstractly describe functions OWFProve and OWFVerify in Appendix E.

The FAEST v2.0 signature scheme [2] uses the grinding technique, which requires the last w bits of the output of H^3_2 to be all zero. We incorporate the grinding technique in rVOLEitH.

For the security proof, we omit the hash compression of commitments or $\widetilde{\mathbf{V}}$, as it just makes proofs unnecessarily complicated. However, in the implementation, we evaluate all those functions for a fair comparison.

EUF-CMA Security of rVOLEitH. From the extractable binding property of VoleComChk, we can prove the EUF-KO security of rVOLEitH. As there is a unique extracted VOLE pair similarly to

Game $\mathrm{G}^{\mathrm{mh}}_{\mathrm{VoleCom}}(\mathcal{A}, Q_I, Q_P)$:

$b^* \leftarrow_{\$} \{0,1\}$

$P \leftarrow_{\$} \mathcal{P};\ q \leftarrow 0;\ p \leftarrow 0$

$b \leftarrow \mathcal{A}^{\mathrm{Get},\mathrm{P}}(Q_I, Q_P)$

return $b^* = b$

Oracle $\mathrm{P}(x)$:

If $p \geq Q_P$: **return** $\bot$

$p \leftarrow p + 1$

return $P(x)$

Oracle $\mathrm{Get}()$:

If $q \geq Q_I$: **return** $\bot$

$\mathsf{iv} \leftarrow_{\$} \{0,1\}^{|\mathsf{iv}|};\ \mathbf{r} \leftarrow_{\$} (\{0,1\}^{\lambda})^{\tau};\ (\Delta_i)_{i\in[\tau]} \leftarrow_{\$} [N]^{\tau}$

$(\mathsf{com}, (\mathsf{decom}_i)_{i\in[\tau]}, (\mathbf{c}_i)_{i\in[\tau]^*}, \mathbf{u}, \mathbf{V}) \leftarrow \mathsf{VoleCom}(\mathsf{iv}, \mathbf{r})$

For $i \in [\tau]$: $\mathsf{pdecom}_i \leftarrow \mathsf{VSC.Open}(\mathsf{iv}, \mathsf{decom}_i, \Delta_i)$

If $b^* = 1$: $(\mathbf{u}, \mathbf{c}_1, \ldots, \mathbf{c}_{\tau-1}) \leftarrow_{\$} (\mathbb{F}^{\ell}_2)^{\tau}$

$q \leftarrow q + 1$

return $(\mathsf{iv}, \mathbf{u}, (\mathbf{c}_i)_{i\in[\tau]^*}, (\mathsf{pdecom}_i, \Delta_i)_{i\in[\tau]})$

Figure 5: Multi-hiding game $\mathrm{G}^{\mathrm{mh}}_{\mathrm{VoleCom}}$ for a VoleCom.

FAEST, the remaining part – handling ZHash failure and QuickSilver check failure – of the EUF-KO security proof is similar to that of FAEST. The detailed theorem statement and security proof can be found in Appendix G.

The EUF-CMA security proof follows the game-hopping technique used for FAEST [4]. The full proof is provided in Appendix G.

Theorem 4.3 (EUF-CMA Security of rVOLEitH). *Assume that $\mathrm{H}_1 : \{0,1\}^* \to \{0,1\}^{2\lambda}$, $\mathrm{H}^1_2, \mathrm{H}^2_2$, and H^3_2 are modeled as random oracles. Let P be an ideal primitive associated with F-functions. For a PPT adversary $\mathcal{A}$ against the EUF-CMA security of rVOLEitH with a total of Q_{sig} signing oracle queries, Q random oracle queries, and Q_P ideal primitive queries, there exist a positive integer u, PPT adversaries $\mathcal{B}$ and $\mathcal{C}$ against the OWF security of* KeyGen *and PRF security of* H_3,[2] *such that*

$$\begin{aligned}\mathbf{Adv}^{\text{euf-cma}}_{\text{rVOLEitH}}(\mathcal{A}) \leq{}& \frac{(Q_{\mathrm{sig}}+Q)^2}{2^{2\lambda}} + \frac{QN^2(\tau^2u^4+8\tau u^2)\varepsilon_{\mathsf{v}}}{8} + Q\varepsilon_{\mathsf{zk}} \\ &+ \frac{2(Q_{\mathrm{sig}}+Q)}{2^{\lambda}} + \mathbf{Adv}^{\mathsf{owf}}_{\mathsf{KeyGen}}(\mathcal{B}) + Q_{\mathrm{sig}}\mathbf{Adv}^{\mathsf{prf}}_{\mathrm{H}_3}(C) \\ &+ \tau(k+2)\mathbf{Adv}^{\mathsf{prg}}_{\mathsf{F}}(Q_{\mathrm{sig}}, Q_P) + \mathbf{Adv}^{\mathsf{coll2}}_{\mathsf{F}}(Q_P) \\ &+ \mathbf{Adv}^{\mathsf{mcoll}}_{\mathsf{F}}(Q, Q_P, u) + \mathbf{Adv}^{\text{4-xor}}_{\mathsf{F}}(Q, Q_P).\end{aligned}$$

Concretely, Lemma 3.6 gives $\mathbf{Adv}^{\mathsf{coll2}}_{\mathsf{F}}(Q_P) \leq 64N^2Q_P^2/2^{2\lambda}$, so the bound already depends on the ratio $NQ_P/2^{\lambda}$. We set $u = \lambda$, for which the bound on the mcoll term is no larger than that on the coll2 term. We increase B from 16 in FAEST to 48 for the short and 40 for the fast rFAEST parameter sets; with these values, the term is $O(NQ/2^{\lambda})$. The additional loss due to VSC is then roughly $NQ/2^{\lambda} + \tau(k+2)\mathbf{Adv}^{\mathsf{prg}}_{\mathsf{F}}$, apart from the standard OWF, PRF, and ZK terms.

[2] H_3 itself is not a PRF, but it is used as a PRF with key-prepending. We use this notation for convenience.

Scheme	Scheme Parameters						Time (ms)		Sig. (B)
	τ	$(\tau_0, 2^{k_0})$	$(\tau_1, 2^{k_1})$	B	T_{open}	w	Sign	Verify	
FAEST-s	11	$(0, 2^{12})$	$(11, 2^{11})$	16	102	7	5.404	3.977	4506
rFAEST-s	11	$(8, 2^{11})$	$(3, 2^{10})$	48	118	10	2.410	1.847	4454
FAEST-EM-s	11	$(0, 2^{12})$	$(11, 2^{11})$	16	103	7	3.911	2.914	3906
rFAEST-EM-s	11	$(8, 2^{11})$	$(3, 2^{10})$	48	118	10	2.320	1.753	4014
FAEST-f	16	$(8, 2^{8})$	$(8, 2^{7})$	16	110	8	0.690	0.588	5924
rFAEST-f	16	$(8, 2^{8})$	$(8, 2^{7})$	40	120	8	0.468	0.373	5620
FAEST-EM-f	16	$(8, 2^{8})$	$(8, 2^{7})$	16	112	8	0.571	0.435	5060
rFAEST-EM-f	16	$(8, 2^{8})$	$(8, 2^{7})$	40	120	8	0.447	0.339	4980

Table 1: Performance comparison between 128-bit parameter sets of FAEST(-EM) and rFAEST(-EM). Times are averaged over 100 runs.

4.3 Application to Signature Schemes

We instantiate reduced VOLE-in-the-Head with FAEST v2.0, one of the candidates in the NIST additional post-quantum signature project, and compare the resulting variants with the original FAEST scheme at the 128-bit security level. All experiments were performed on an AMD Ryzen 7 2700X 3.7 GHz with 128 GB of RAM. The implementation and accompanying stage-wise profiling code are publicly available, as described in Appendix B. The results are summarized in Table 1. We also compare the corresponding 192- and 256-bit parameter sets. The results are reported in Tables 3 and 4, in Appendix D.1, respectively.

Changes in rFAEST. rFAEST differs from FAEST in four main aspects. The leaf-commitment length is reduced to λ bits in rFAEST(-EM), from 3λ bits in FAEST and 2λ bits in FAEST-EM, respectively, using an AES-CTR PRG. The one-tree BAVC structure of FAEST is replaced by a GGM-forest vector commitment, so the opening procedure always succeeds, and the rejection caused by the T_{open}-nodes condition is removed. For the rFAEST(-EM)-short parameters, the grinding parameter is increased from $w = 7$ to $w = 10$, while the rFAEST(-EM)-fast parameters keep $w = 8$. In the VOLE consistency check, the original choice $B = 16$ is replaced by $B = 48$ for the short parameters and $B = 40$ for the fast parameters.

Parameter Selection. The parameter τ denotes the number of repetitions of GGM tree evaluation. τ is chosen to be the same as that of the corresponding parameters in FAEST. 2^{k_0} and 2^{k_1} are the numbers of GGM tree leaf nodes. τ_i is the number of repetitions with a GGM tree of 2^{k_i} leaf nodes, where $\tau = \tau_0 + \tau_1$. Parameter B is the bit-length of additional VHash output to achieve the hiding property. The reduced versions require slightly larger B since there is more than one preimage copath to be VHashed. In rFAEST, T_{open} is not a rejection threshold; it is the total number of copath nodes opened across the GGM forest, namely $T_{\text{open}} = \tau_0 k_0 + \tau_1 k_1$.

w is the number of bits required to be zero in the grinding technique. Increasing the grinding parameter w increases the expected signing cost, as more candidates must be tested on average, but reduces both the verification cost and the signature size by permitting a smaller tree and therefore less associated computation. In our parameter selection, we set $w = 10$ for the short parameter sets and $w = 8$ for the fast parameter sets to trade off signing and verification speed.

We set the iv length to 16 bytes, as in FAEST.

Signature Size. Compared to FAEST, the signature size of rFAEST differs in three components: the VOLE-related term, the node-opening term, and the hidden-commitment term. The VOLE-related term increases in the reduced variants, since the proof size of the VOLE consistency check is $\lambda + B$ bits. The total increase in the VOLE-related term is $(B - 16)\tau$ bits per signature. By contrast, the hidden-commitment term decreases because rFAEST(-EM) reduces the leaf-hash length to λ bits (from 3λ bits in FAEST and 2λ bits in FAEST-EM). The remaining contribution is the node-opening term. For fixed τ, rFAEST opens more nodes than FAEST, since the GGM-forest opening rule has no threshold. For the same parameter set, this increases the node-opening term of rFAEST over that of FAEST by $\Delta T_{\text{open}} \cdot \lambda$ bits, where ΔT_{open} denotes the additional number of nodes opened by rFAEST compared with the corresponding FAEST.

As a result, rFAEST reduces the signature size by 52, 304, and 80 bytes for short, fast, and EM-fast, respectively; for EM-short, the node-opening overhead dominates the hidden-commitment saving, increasing the size by 108 bytes.

Improvement in Signing and Verification Time. Compared with the corresponding FAEST(-EM) baselines, rFAEST reduces both signing and verification time across all evaluated parameter sets.

Table 2 presents the signing-time breakdown for the 128-bit FAEST instances. The remaining profiling results, covering the 192- and 256-bit FAEST instances and the 128-, 192-, and 256-bit FAEST-EM instances, are provided in Appendix D.2.

Since the profiling code introduces measurement overhead and the grinding loop makes signing time probabilistic, the totals may differ slightly from Table 1. Exact explanations for each stage in Table 2 can be found in Appendix D.2.

In rFAEST, each leaf commitment has length λ, whereas the corresponding commitment length is 3λ in FAEST and 2λ in FAEST-EM. As a result, the input length to the SHAKE call used to compress the leaf commitments is reduced proportionally. Thus, the cost of compressing the leaf commitments drops by roughly factors of three and two compared to FAEST and FAEST-EM, respectively.

In addition, the leaf generation time is reduced because the number of PRG calls per leaf decreases from four in FAEST and two in FAEST-EM to only one in rFAEST.

Moreover, the *Open & Grinding* cost is reduced because forest opening in rFAEST has no rejects. For instance, for the 128-bit parameter sets, FAEST-s/f require about $2^{12.4}/2^{8.5}$ challenge candidates, whereas FAEST-EM-s/f require about $2^{11.8}/2^{8.3}$ to satisfy both the T_{open} and grinding conditions. In contrast, rFAEST(-EM) needs about $2^w = 2^{10}/2^8$ candidates for the short/fast variants.

Combining all these effects, rFAEST achieves 27.7–124.2% faster signing and 28.3–115.3% faster verification compared with the corresponding FAEST parameters.

4.4 Proof of Lemma 4.1

In this section, we will show that an input to VoleComChk is bound to a single VOLE pair and that it can be extracted using the extractor. Since we use VSC rather than VC, there may be multiple VOLE pairs extracted from the VOLE commitment; however, because an adversary cannot fully control those VOLE pairs and the VOLE

	short		fast	
Stage	FAEST	rFAEST	FAEST	rFAEST
Expand tree	355.5	299.8	43.4	35.4
Gen. & Commit leaf	593.7	214.1	84.4	34.6
Compress leaf com.	2328.8	647.0	234.1	80.0
VOLE & QS	897.7	856.8	220.2	206.3
Open & Grinding	1516.1	313.0	106.3	65.5
Other	40.7	46.0	44.3	44.2
Total	5732.4	2376.8	732.6	466.0

Table 2: Signing time breakdown for FAEST and rFAEST at the 128-bit security level. Times are averaged over 100 runs and reported in microseconds.

consistency check challenge (i.e., the hash key of VHash) is generated randomly, it will be difficult (when appropriate parameters are used) to pass multiple VOLE pairs through the VOLE consistency check simultaneously.

To briefly summarize the proof of Lemma 4.1, we examine the conditions under which multiple extracted VOLE pairs could pass through the VOLE consistency check simultaneously, demonstrate that the probability of this occurring is negligible, and finally show that, with high probability, only a single VOLE pair can be extracted.

We will construct a VoleComChk extractor Ext on top of the VSC extractor $\mathsf{Ext}_{\mathsf{VSC}}$ from Figure 2. At a high level, the VoleComChk extractor will try to extract a VOLE pair which also passes the VOLE consistency check from the adversarial queries. If there is no such VOLE pair or there are multiple VOLE pairs that pass the VOLE consistency check, then the extractor returns $\perp$. When the extractor returns $\perp$, we will classify $\perp$ into $\perp_{\mathsf{abort}}$ and $\perp_{\mathsf{non}}$ for readability; $\perp_{\mathsf{abort}}$ means that it will be an abort condition in the extractable binding proof, and $\perp_{\mathsf{non}}$ means otherwise. Algorithmically, both $\perp_{\mathsf{abort}}$ and $\perp_{\mathsf{non}}$ return nothing.

Fix an arbitrary $\mathcal{A}$ and let $\mathcal{L}_{\mathsf{chall}}$ be the set of query-response pairs made by $\mathcal{A}$ to $\mathsf{H}_{\mathsf{chall}}$. Then, given an input $(\mathsf{chall}, \widetilde{\mathbf{u}}, \widetilde{\mathbf{V}}, \mathcal{L}_{\mathsf{chall}}, \mathcal{L}_P)$, the VoleComChk extractor Ext does the following.

(1) Find a preimage $\mu \,\|\, \mathsf{com} \,\|\, \mathbf{c}_1 \,\|\, \cdots \,\|\, \mathbf{c}_{\tau-1} \,\|\, \mathsf{iv}$ of chall under $\mathsf{H}_{\mathsf{chall}}$ from $\mathcal{L}_{\mathsf{chall}}$. If there is no unique preimage, Ext immediately returns $\perp_{\mathsf{non}}$, otherwise, let $\mathbf{c}_0 = 0$ and parse com as $(\overline{\mathsf{com}}_i)_{i\in[\tau]}$.

(2) For each $i \in [\tau]$ and $\Delta \in [N]$, extract $\mathcal{E}_{i,\Delta}$ using VSC extractor $\mathsf{Ext}_{\mathsf{VSC}}$ where

$$\mathcal{E}_{i,\Delta} = \mathsf{Ext}_{\mathsf{VSC}}(\mathsf{iv}, \overline{\mathsf{com}}_i, \Delta, \mathcal{L}_P).$$

If $\mathsf{Ext}_{\mathsf{VSC}}$ returns $\perp$, Ext immediately returns $\perp_{\mathsf{abort}}$. If there exists $i \in [\tau]$ such that $|\mathcal{E}_{i,\Delta}| = 0$ for all $\Delta \in [N]$, Ext immediately returns $\perp_{\mathsf{non}}$.

(3) For $i \in [\tau]$ and $j \in [N]$, let $m^*_{i,j} = \perp$. For each $i \in [\tau], \Delta \in [N]$ and $(m_a)_{a\in[N]} \in \mathcal{E}_{i,\Delta}$, do the following.
   (a) Compute $(\mathbf{u}, \mathbf{V}) = \mathsf{ToVOLE}((m_a)_{a\in[N]})$. If

$$\mathsf{VHash}(\mathsf{chall}, \Delta \cdot (\mathbf{u} \oplus \mathbf{c}_i) \oplus \mathbf{V}) \neq \Delta \cdot \widetilde{\mathbf{u}} \oplus \widetilde{\mathbf{V}},$$

   continue to the next iteration.
   (b) If there exists $j \in [N] \setminus \{\Delta\}$ such that $\perp \neq m^*_{i,j} \neq m_j$, Ext immediately returns $\perp_{\mathsf{abort}}$.
   (c) For $j \in [N] \setminus \{\Delta\}$, update $m^*_{i,j} \leftarrow m_j$.

   Ext immediately returns $\perp_{\mathsf{non}}$ if there exists $i \in [\tau]$ and distinct $j, j' \in [N]$ such that $m^*_{i,j} = m^*_{i,j'} = \perp$.

(4) For each $i \in [\tau]$, if $\exists! j \in [N]$ such that $m^*_{i,j} = \perp$, let $\Delta^*_i = j$, otherwise, let $\Delta^*_i = \perp$. Also, let

$$(\mathbf{u}^*_i, \mathbf{V}^*_i) = \mathsf{ToVOLE}((m^*_{i,j})_{j\in[N]}).$$

   If there exist distinct $i, j \in [\tau]$, such that $\Delta^*_i = \Delta^*_j = \perp$ and $\mathbf{u}^*_i \neq \mathbf{u}^*_j$, Ext immediately returns $\perp_{\mathsf{abort}}$. Then, all $\mathbf{u}^*_i$'s with $\Delta^*_i = \perp$ are identical, and let $\mathbf{u}^*$ be that value (if there is no such $\mathbf{u}^*_i$, set $\mathbf{u}^*$ as arbitrary value). Also, compute $\mathbf{V}^*$ as follows.

$$\begin{cases} \mathbf{V}^*[ik+j] = \mathbf{V}^*_i[j] & \text{if } \Delta^*_i = \perp \\ \mathbf{V}^*[ik+j] = \mathbf{V}^*_i[j] \oplus \Delta^*_i[j] \cdot (\mathbf{u}^*_i \oplus \mathbf{u}^*) & \text{otherwise} \end{cases}$$

   Finally, Ext returns $(\mathbf{u}^*, \mathbf{V}^*)$.

From Step 1 to 2, Ext extracts committed messages. In Step 3, it extracts (unique) small-field VOLE for each GGM tree that passes the VOLE consistency check. If such small-field VOLE is not unique, it aborts. In Step 4, Ext aggregates small-field VOLEs to create a large-field VOLE and outputs it. If the small-field VOLEs are inconsistent with each other, it aborts.

We argue via game hops. Let G_0 be the original game. Intuitively, if Ext cannot extract multiple VOLE pairs, the extractable binding property is satisfied. G_1 and G_2 proactively prevent some bad events for the ease of probability computation in the following games. G_3 and G_4 abort if Ext can extract multiple VOLE pairs that pass the VOLE consistency check (i.e., Ext returns $\perp_{\mathsf{abort}}$).

G_1: This game aborts if there exists a collision or preimage-finding event on $\mathsf{H}_{\mathsf{chall}}$. Then,

$$\Pr[\mathrm{G}_1 \text{ aborts}] \leq \frac{Q^2_{\mathsf{chall}}}{2^{2\lambda}}$$

since the key length of VHash is at least 2λ.

G_2: This game aborts if there exists $i \in [\tau], \Delta, \Delta' \in [N], (m_a)_{a\in[N]} \in \mathcal{E}_{i,\Delta}$ and $(m'_a)_{a\in[N]} \in \mathcal{E}_{i,\Delta'}$ such that

$$m_{\Delta\oplus 1} \oplus m_{\Delta'\oplus 1} \oplus m'_{\Delta\oplus 1} \oplus m'_{\Delta'\oplus 1} = 0.$$

Then, we have

$$\Pr[\mathrm{G}_2 \text{ aborts}] \leq \mathbf{Adv}^{4\text{-xor}}_{\mathsf{F}}(Q_{\mathsf{com}}, Q_P).$$

G_3: In this game, we will capture the failure condition that Ext returns $\perp_{\mathsf{abort}}$ in Step 3. Precisely, G_3 aborts if there exists $i \in [\tau]$, $\Delta, \Delta' \in [N]$, $(m_a)_{a\in[N]} \in \mathcal{E}_{i,\Delta}$ and $(m'_a)_{a\in[N]} \in \mathcal{E}_{i,\Delta'}$ such that $(m_a)_{a\in[N]} \neq (m'_a)_{a\in[N]}$, $\Delta \oplus \Delta' \neq 1$ and

$$\begin{cases} (\mathbf{u}, \mathbf{V}) \coloneqq \mathsf{ToVOLE}(m_0, \ldots, m_{N-1}), \\ (\mathbf{u}', \mathbf{V}') \coloneqq \mathsf{ToVOLE}(m'_0, \ldots, m'_{N-1}), \\ \mathsf{VHash}(\mathsf{chall}, \Delta \cdot (\mathbf{u} \oplus \mathbf{c}_i) \oplus \mathbf{V}) = \Delta \cdot \widetilde{\mathbf{u}} \oplus \widetilde{\mathbf{V}} \\ \mathsf{VHash}(\mathsf{chall}, \Delta' \cdot (\mathbf{u}' \oplus \mathbf{c}_i) \oplus \mathbf{V}') = \Delta' \cdot \widetilde{\mathbf{u}} \oplus \widetilde{\mathbf{V}} \end{cases} \tag{3}$$

Let $(\delta_a)_{a\in[k]}$ and $(\delta'_a)_{a\in[k]}$ be the bit-representation of Δ and Δ', respectively. Then, given $\Delta \oplus \Delta' \neq 1$, the following is a necessary condition of (3):
– if $\delta_0 = \delta'_0$, $\mathsf{VHash}(\mathsf{chall}, \delta_0(\mathbf{u} \oplus \mathbf{u}') \oplus \mathbf{V}[0] \oplus \mathbf{V}'[0]) = 0$;

– if $\delta_0 \oplus \delta'_0 = 1$, for $j = \min\left\{a \in \{1,\dots,k-1\} : \delta_a \neq \delta'_a\right\}$,

$$\mathsf{VHash}(\mathsf{chall}, (\delta_0 \oplus \delta_j)\mathbf{u} \oplus \mathbf{V}[0] \oplus \mathbf{V}[j]) = \mathsf{VHash}(\mathsf{chall}, (\delta'_0 \oplus \delta'_j)\mathbf{u}' \oplus \mathbf{V}'[0] \oplus \mathbf{V}'[j]).$$

Recall that $m_a = m'_a$ for $a \notin \{\Delta, \Delta \oplus 1, \Delta', \Delta' \oplus 1\}$, since otherwise $\mathsf{Ext}_{\mathsf{VSC}}$ should return $\perp$ so Ext also returns $\perp_{\mathsf{abort}}$ at Step 2. Also, coefficients of m_Δ and $m_{\Delta\oplus 1}$ are both 0 in $\delta_j\mathbf{u} \oplus \mathbf{V}[j]$ for $j > 0$, while those are 0 and 1 in $\delta_0\mathbf{u} \oplus \mathbf{V}[0]$, respectively. Now assume $\delta_0 \oplus \delta'_0 = \delta_j \oplus \delta'_j = 1$. Then we have

$$(\delta_0 \oplus \delta_j)(\mathbf{u} \oplus \mathbf{u}') \oplus \mathbf{V}[0] \oplus \mathbf{V}[j] \oplus \mathbf{V}'[0] \oplus \mathbf{V}'[j] = 0 \Leftrightarrow m_{\Delta\oplus 1} \oplus m_{\Delta'\oplus 1} \oplus m'_{\Delta\oplus 1} \oplus m'_{\Delta'\oplus 1} = 0$$

which contradicts the modification in G_2. Therefore, by the universality of VHash, $\Pr[(3)\text{ holds}] \le \varepsilon_{\mathsf{v}}$ as chall is chosen at random. Also, if $\delta_0 = \delta'_0$, we have

$$\delta_0\mathbf{u} \oplus \mathbf{V}[0] \oplus \delta'_0\mathbf{u}' \oplus \mathbf{V}'[0] = 0 \Leftrightarrow m_{\Delta\oplus 1} \oplus m'_{\Delta\oplus 1} = \begin{cases} m_{\Delta'\oplus 1} \oplus m'_{\Delta'\oplus 1} & \text{if } \Delta \neq \Delta' \\ 0^\ell & \text{otherwise.} \end{cases}$$

Therefore, $\Pr[(3)\text{ holds}] \le \varepsilon_{\mathsf{v}}$ also for this case and we have

$$\Pr[\mathrm{G}_3 \text{ aborts}] \le Q_{\mathsf{chall}} \cdot \tau \cdot N^2 \cdot u^2 \cdot \varepsilon_{\mathsf{v}}$$

Now, if G_3 does not abort, two extracted messages $(m_a)_{a\in[N]} \in \mathcal{E}_{i,\Delta}$ and $(m'_a)_{a\in[N]} \in \mathcal{E}_{i,\Delta'}$ can both pass the consistency check only if $\Delta \oplus \Delta' = 1$. Therefore, Ext does not return $\perp_{\mathsf{abort}}$ in Step 3.

G_4: In this game, we will capture the failure condition that Ext returns $\perp_{\mathsf{abort}}$ in Step 4. G_4 checks whether there exists a pair of small-field VOLEs from different trees that cause a bad consistency check for a given challenge chall and corrections, and the game aborts if that happens. For $i \in [\tau]$, let $\mathcal{E}^*_i$ be the collection of $(m^*_a)_{a\in[N]}$ such that there exists $\Delta \in [N]$ such that

$$\begin{cases} \exists (m_a)_{a\in[N]} \in \mathcal{E}_{i,\Delta}, (m^*_a)_{a\neq\Delta} = (m_a)_{a\neq\Delta} \\ \exists (m'_a)_{a\in[N]} \in \mathcal{E}_{i,\Delta\oplus 1}, m^*_\Delta = m'_\Delta \end{cases}$$

Next, let

$$\mathcal{E}_i = \left\{(\mathbf{u},\mathbf{V}) = \mathsf{ToVOLE}((m_a)_{a\in[N]}) : (m_a)_{a\in[N]} \in \mathcal{E}^*_i\right\}.$$

G_4 aborts if there exist distinct $i, j \in [\tau]$, $(\mathbf{u}_i, \mathbf{V}_i) \in \mathcal{E}_i$, and $(\mathbf{u}_j, \mathbf{V}_j) \in \mathcal{E}_j$ such that

$$\begin{cases} \mathbf{u}_i \oplus \mathbf{c}_i \neq \mathbf{u}_j \oplus \mathbf{c}_j & (\text{with } \mathbf{c}_0 \coloneqq 0) \\ \mathsf{VHash}(\mathsf{chall}, \mathbf{u}_i \oplus \mathbf{c}_i) = \mathsf{VHash}(\mathsf{chall}, \mathbf{u}_j \oplus \mathbf{c}_j). \end{cases}$$

By the universality of VHash, we have

$$\Pr[\mathrm{G}_4 \text{ aborts}] \le Q_{\mathsf{chall}} \cdot \sum_{i\neq j} |\mathcal{E}_i| \cdot |\mathcal{E}_j| \cdot \varepsilon_{\mathsf{v}} \le Q_{\mathsf{chall}} \cdot \binom{\tau}{2} \cdot \frac{N^2u^4}{4} \cdot \varepsilon_{\mathsf{v}} \le \frac{Q_{\mathsf{chall}} \cdot \tau^2 \cdot N^2 \cdot u^4 \cdot \varepsilon_{\mathsf{v}}}{8}$$

where the last inequality comes from $|\mathcal{E}_i| \le \frac{Nu^2}{2}$ since otherwise $\mathsf{Ext}_{\mathsf{VSC}}$ should return $\perp$ so Ext also returns $\perp_{\mathsf{abort}}$ at Step 2.

By the modifications in G_i for $i \in \{1,2,3,4\}$, Ext returns $\perp_{\mathsf{abort}}$ only if $\mathsf{Ext}_{\mathsf{VSC}}$ returns $\perp$. Also, if Ext returns $\perp_{\mathsf{non}}$ or $(\mathbf{u}^*, \mathbf{V}^*)$, $\mathcal{A}$ can never win because Ext searches the extraction in a brute-force manner. Therefore, $\mathcal{A}$ wins in G_4 only when extractable semi-binding security of VSC is broken, i.e.,

$$\Pr[\mathcal{A} \text{ wins in } \mathrm{G}_4] \le \mathbf{Adv}^{\mathsf{esb}}_{\mathsf{VSC}}(Q_{\mathsf{com}}, Q_P, u).$$

Together with Lemma 3.4, we can conclude the proof. □

4.5 Future Work

Application to One-Tree Technique. Our security proof of rVOLEitH heavily depends on the shape of pdecom_i. We use the fact that a copath contains only one leaf node so that any set of commitments $(\overline{\mathsf{com}}_{i,j})_j$ does not have too many preimage copaths. As nodes with height ≥ 1 can collide in commitments with negligible probability, the number of preimage copaths depends on the number of leaf nodes contained in pdecom_i.

However, the one-tree technique proposed by Baum et al. [3] joined all the τ trees to one tree so that the shape of a copath – the structure here is no longer a copath in the strict sense, but we will retain the term – is not fixed; a copath may have too many leaf nodes. At this moment, we do not know whether the one-tree technique can be securely combined with a vector semi-commitment, and we leave it as future work.

Secret-Key-Root cGGM. A complementary line of work fixes the root of the (correlated) GGM tree to a part of the secret key, rather than to a uniformly random string, in order to reduce $\tau\lambda$ bits of the offset into the secret-key-derived correlation and thereby shrink the signature. This *secret-key-root cGGM* (scGGM) technique was first used by SBC [17] and later by MQOM [7] and rBN++ [22], and was recently extended to a forest of scGGM trees by $\mathrm{SBC}_{\mathrm{VOLE}}$ [18]. However, Kosuge and Xagawa pointed out that the standard hiding argument for vector commitments does not hold with scGGM [23]. In concurrent works, Feneuil and Rivain [14], and Kosuge and Xagawa [24] provide the first formal security analyses of scGGM-based MPCitH signatures under a degraded one-wayness assumption.

VSC is orthogonal to the scGGM technique; VSC abstracts the GGM-style tree into the F-functions and only requires a small set of generic properties on them, none of which restricts the root from being secret-key-derived. Although the properties themselves do not syntactically forbid secret-key-derived roots, the hiding proof must be redone under a degraded one-wayness assumption.

An ordinary cGGM tree without a secret-key-derived root can be used for $\mathsf{F}_{\mathsf{tree}}$ without changing the main proof; only the concrete advantage bounds in Lemma 3.6 may change slightly. We omit this instantiation to isolate the effect of VSC on FAEST.

Post-Quantum (QROM) Security. Except for statistically bounded VHash failures and the stated F bad events, VoleComChk yields a unique extracted VOLE pair, which makes a QROM proof plausible. The current ROM proof does not lift directly. We therefore leave QROM security for future work.

5 Attack on VOLEitH with MinMax Commitment

In this section, we overview the VOLEitH-SBC signature scheme and describe a forgery attack on its MinMax variant. To make the attack more readable, we follow the original notations from [18],

which differ significantly from those used in the previous sections in this paper.

5.1 VOLEitH-SBC

VOLEitH-SBC is a signature scheme proposed by Huth and Joux [18]. It is a translation of the MPC-in-the-Head (MPCitH) signature scheme based on subfield bilinear collision (SBC) [17] into the VOLE-in-the-Head (VOLEitH) paradigm. VOLEitH-SBC basically follows VOLEitH-style and proves the knowledge of an SBC solution, but it differs from the original VOLEitH framework [5] in details such as the use of a pre-tree and the hypercube technique [1].

In the VOLEitH-SBC paper, the authors proposed two signature schemes: $\mathrm{SBC}_{\mathrm{VOLE}}$ and $\mathrm{SBC}^{\mathcal{M}}_{\mathrm{VOLE}}$. The difference between the two schemes is the leaf commitment; the former one uses the usual leaf commitment (2λ-bit hash output), and the latter one uses the MinMax commitment. As the attack utilizes the vulnerability of MinMax commitment in $\mathrm{SBC}^{\mathcal{M}}_{\mathrm{VOLE}}$ rather than the SBC problem itself, we provide the overview of $\mathrm{SBC}^{\mathcal{M}}_{\mathrm{VOLE}}$ in Appendix H.

5.2 Overview of the Attack

In this section, we introduce the motivation for our attack on $\mathrm{SBC}^{\mathcal{M}}_{\mathrm{VOLE}}$.[3] In the next subsection, we will introduce the attack in detail. This attack is a forgery attack by manipulating the GGM tree while the leaf commitment remains the same. It does not require any signature query.

In the VOLEitH paradigm, leaf commitment is a commitment to each leaf node of GGM trees, and it is usually instantiated using a hash function or a pseudorandom generator (PRG). It prevents an illegitimate signer from changing the hidden leaf node after selecting which leaf node will be hidden. As leaf commitments are hashed to the first challenge, if a signer tries to change partial decommitment (GGM copath and commitment to the hidden node), then the second and the third challenge will also change. It implies that the illegitimate signer cannot match the last challenge hash.

On the other hand, MinMax commitment is a kind of batch commitment to all the leaf nodes. The commitment value is the global maximum and global minimum value of the original leaf commitments. In fact, it does not bind the leaf nodes at all. If we find inputs whose hash values are sufficiently large or small (e.g., $\mathcal{H}(x_{\max}) = \texttt{0xffffff}\ldots$ and $\mathcal{H}(x_{\min}) = \texttt{0x000000}\ldots$), we fix those two inputs as leaf nodes. Then, we can change other puncturable pseudorandom function (PPRF) keys even after the last challenge is fixed.

An attacker may try to fix all the variables in the signature, except PPRF keys. The goal of the forgery is to generate PPRF keys that make h'_1 (generated in Verify) equal to h_1 (fixed in Sign). If the MinMax commitment is fixed, finding PPRF keys satisfying the SBC relation ($\Phi(\Delta^*) = A + B\Delta^*$) will yield a valid forgery. The MinMax commitment can be fixed with high probability by including $x_{\max}$ and $x_{\min}$ in the PPRF keys. However, since $\Phi(\Delta^*)$ is a complex quadratic polynomial in random tapes generated by a PRG fed with seeds, it seems quite complicated to match $\Phi(\Delta^*) = A + B\Delta^*$. To overcome this complexity, we use a meet-in-the-middle approach.

[3] We note that this version has no security proof in the original paper.

Now, we overview our attack using the following three steps: (1) collecting candidates of $\Phi(\Delta^*)$, (2) fixing $\Phi(\Delta^*)$, (3) finding PPRF keys for the selected $\Phi(\Delta^*)$.

In the first step, an attacker collects plausible candidates of $\vec{F}_x(\Delta^*)$ and $\vec{F}_y(\Delta^*)$ where the corresponding $\Phi(\Delta^*)$ satisfies $\Phi(\Delta^*) = A + B\Delta^*$ for some Δ^*. Let γ be the number of such candidates. At this step, $\vec{F}_x(\Delta^*)$ and $\vec{F}_y(\Delta^*)$ are not honestly computed, but selected arbitrarily. We denote such candidates of $(\vec{F}_x(\Delta^*), \vec{F}_y(\Delta^*))$ as $(\vec{s}, \vec{w})$.

At the second step, the attacker generates Δ^* by fixing inputs of hash function $\mathcal{H}_1$. Although Δ^* itself is $k \approx 2\lambda$-bit long, it is represented as $\Delta^* = \sum_{i\in[\tau], j\in[D]} b_{j,i}\alpha_{iD+j}$. It implies that Δ^* has only $\tau D \approx \lambda$ bit entropy if α_i are all fixed. As there are γ candidates, the probability that the generated Δ^* satisfies $\Phi(\Delta^*) = A + B\Delta^*$ is expected to be $\gamma/2^\lambda$. We denote $(\vec{s}, \vec{w})$ corresponding to such Δ^* as $(\vec{s}^*, \vec{w}^*)$.

At the last step, the attacker uses Wagner's generalized birthday algorithm to seek PPRF keys that make $(\vec{s}^*, \vec{w}^*)$ the honestly computed values $(\vec{F}_x(\Delta^*), \vec{F}_y(\Delta^*))$. $\vec{F}_x(\Delta^*)$ and $\vec{F}_y(\Delta^*)$ are sums of pseudorandom strings generated from PPRF keys. Since the indices of hidden seeds are chosen before this step, the copath nodes serve as sources of strings for the algorithm. We assume that the computed values are random for the application of generalized birthday algorithm.

While generating h_1, the attacker should restrict the hidden-node indices in the first and second trees to 1. At those two 0th positions, $x_{\max}$ and $x_{\min}$ will be located. At this moment, the VOLE pair and the choice satisfy the MinMax commitment with high probability while most of the cGGM tree remains undetermined. The attacker can freely manipulate partial decommitment (copaths and hidden commitments in the cGGM trees), so the attacker can generate VOLE pairs regardless of the choice.

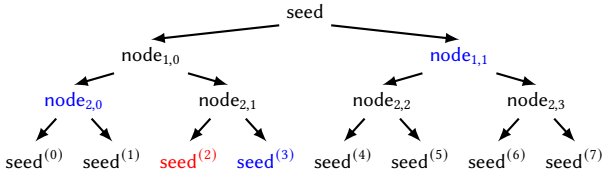


Figure 6: Diagram of puncturable pseudorandom function (PPRF). In this example, $\mathrm{seed}^{(2)}$ is the unopened node, and blue nodes are the PPRF key.

5.3 Attack on $\mathrm{SBC}^{\mathcal{M}}_{\mathrm{VOLE}}$

We provide the detailed attack algorithm in the following. As this section heavily relies on specific notations in [18], we refer to the original paper or an overview in Appendix H. Notation with a prime ($'$) refers to the notation that appears in the verification algorithm.

For the cost analysis, we take $q = 2$ and model the final matching problem in Step 7 as a uniform $2n$-bit generalized birthday problem. We also assume that the lists produced in Step 6 satisfy the random-list conditions needed by Wagner's algorithm. Define

$$a_D = 1 + \lfloor \log_2(D-2) \rfloor, \qquad b_\tau = 1 + \lfloor \log_2 \tau \rfloor,$$

and let

$$M = 2^{2n/b_\tau}, \qquad Q = 2^{(k+\log_2 M)/a_D}. \tag{4}$$

Here, M is the number of valid PPRF key candidates per main tree, and Q is the input list size per free copath position. List sizes are rounded up to integers when needed.

(1) Fix salt and find $x_{\min}$ and $x_{\max}$ such that

$$\mathcal{H}_2(\mathsf{salt}\|0\|(D,0)\|x_{\min}) \le 2^{2\lambda-u}$$
$$\mathcal{H}_2(\mathsf{salt}\|1\|(D,0)\|x_{\max}) \ge 2^{2\lambda} - 2^{2\lambda-u},$$

which means that 0-th seed of 0-th main tree has a sufficiently small hash value, and 0-th seed of 1-st main tree has a sufficiently large hash value. The value u will be chosen later. The choice of $(j,i) = (0,0)$ and $(1,0)$ is only for convenience; when $i_0^* = i_1^* = 1$, each 0-th seed occupies the leaf-level position in the corresponding copath and can be chosen independently without affecting any other leaf.

(2) Fix transcripts for the first challenge and some for the second challenge: msg, $(\delta_{\vec{y}_j}, \delta_{z_j})_{j\in[\tau]}, A, B$, and $(a^{(j)})_{j\in[\tau]}$. Variable b is not fixed yet. It fixes the hash outputs h_0, challenges $\vec{\mu}$, and $(\alpha_m)_{m\in[\tau D]}$.

(3) Find γ pairs $(\vec{s},\vec{w})$ in $(\mathbb{F}_q)^{n-2}$ such that

$$\widetilde{\Phi}(\vec{s},\vec{w}) \coloneqq \left(\vec{u}\cdot\begin{pmatrix}\vec{s}\\1\\0\end{pmatrix}\right)\left(\vec{v}\cdot\begin{pmatrix}\vec{w}\\0\\1\end{pmatrix}\right) - \left(\vec{u}\cdot\begin{pmatrix}\vec{w}\\0\\1\end{pmatrix}\right)\left(\vec{v}\cdot\begin{pmatrix}\vec{s}\\1\\0\end{pmatrix}\right)$$
$$\in \mathsf{Span}(B\alpha_0,\dots,B\alpha_{\tau D-1}) + A$$

where $\mathsf{Span}(B\alpha_0,\dots,B\alpha_{\tau D-1}) + A$ means the $\mathbb{F}_2$-linear span of $(B\alpha_0,\dots,B\alpha_{\tau D-1})$ shifted by A. Make a list L of triples $(\vec{s},\vec{w},\widetilde{\Phi}(\vec{s},\vec{w}))$. One pair can be found by solving a linear equation

$$\widetilde{\Phi}(\vec{s}_0,\vec{w}) = A + \sum_{i\in[\tau D]} c_i(B\alpha_i)$$

with respect to $\vec{w}$ and $c_i \in \{0,1\}$ after fixing random $\vec{s} = \vec{s}_0$ with a significant probability. Gaussian elimination requires $O(n^3)$ arithmetic operations for each pair. $\gamma = |L|$ will be chosen later.

(4) Fix variable b. It fixes the hash outputs $h_1, (i_j^*)_{j\in[\tau]}, (\Delta_j^*)_{j\in[\tau]}$, and i_p^*. Since $\Delta^* = \sum_{j\in[\tau]}\Delta_j^*$, it also fixes Δ^*. Check whether Δ^* satisfies $A + B\Delta^* = \widetilde{\Phi}(\vec{s},\vec{w})$ for some $(\vec{s},\vec{w},\widetilde{\Phi}(\vec{s},\vec{w})) \in L$. Check also whether $i_0^* = i_1^* = 1$. If either condition is not satisfied, do this step again with another b. If those two conditions are satisfied, we will denote such a pair $(\vec{s},\vec{w})$ as $(\vec{s}^*,\vec{w}^*)$. This step will expectedly end in $2^{(\tau+2)D}/\gamma$ repetitions.

(5) Generate a pre-tree (with 2^τ leaves) using a random root, and fix it. This fixes one of the two highest nodes in each main tree ($\mathsf{node}_{1,1}$ in Figure 6), which is in the copath of i_j^*.

(6) For $j \in [\tau]$, do the following:

(a) To make $h_1 = h_1'$, the attacker needs to find PPRF keys such that $A = A'$ and $(a^{(j)})_{j\in[\tau]} = (a^{(j)'})_{j\in[\tau]}$. To find PPRF keys passing the verification, evaluate tree hash functions at each PPRF key position. Since $(i_j^*)_{j\in[\tau]}$ is fixed, the positions for PPRF keys are fixed. For example, in Figure 6, $i_j^* = 2$ and the PPRF key positions are $\mathsf{node}_{1,1}$, $\mathsf{node}_{2,0}$ and $\mathsf{seed}^{(3)}$. Since $\mathsf{node}_{1,1}$ is fixed by the pre-tree fixation at the previous step, evaluating hash functions at positions $\mathsf{node}_{2,0}$ and $\mathsf{seed}^{(3)}$ is sufficient. Get seeds and shares of (y,z) values by random evaluation at the PPRF key positions. Record $((j,h,l),\mathsf{node}_{h,l},\mathsf{psum}_{j,h,l})$ where

$$\mathsf{psum}_{j,h,l} = \sum_{i\in[D]}\alpha_{jD+i}\Bigg(\vec{\mu}\cdot\sum_{m\in[2^{D-h}]}\left(B_i(p_{h,l,m}) \stackrel{?}{=} b_{i,j}\right)\left(\vec{y}_j^{[\![p_{h,l,m}]\!]} + (p_{h,l,m} \stackrel{?}{=} 2^D-1)\delta_{\vec{y}_j}\right)$$
$$+\sum_{m\in[2^{D-h}]}\left(B_i(p_{h,l,m}) \stackrel{?}{=} b_{i,j}\right)\left(z_j^{[\![p_{h,l,m}]\!]} + (p_{h,l,m} \stackrel{?}{=} 2^D-1)\delta_{z_j}\right)\Bigg),$$

and $p_{h,l,m} = 2^{D-h}\cdot l + m$ are seed positions generated from $\mathsf{node}_{h,l}$. The notation $(x \stackrel{?}{=} y)$ means that it outputs 1 if $x = y$, and 0 otherwise. $\mathsf{psum}_{j,h,l}$ is a partial sum of $a^{(j)} \oplus b\Delta_j^*$, which is generated from $\mathsf{node}_{h,l}$ position. Therefore, summing $\mathsf{psum}_{j,h,l}$ over PPRF key position (h,l) gives $a^{(j)} \oplus b\Delta_j^*$. Both $\mathsf{psum}_{j,h,l}$ and its target lie in $\mathbb{F}_{2^k}$, so this per-tree matching condition has k bits. For each free PPRF key position, evaluate the subtree Q times. Overall, it requires $O(\tau 2^D Q)$ hash computations.

(b) Find $M = 2^{2n/b_\tau}$ PPRF key candidates for this tree such that

$$a^{(j)} = Z^{(j)}(\Delta_j^*) \oplus \vec{F}_y^{(j)}(\Delta_j^*)\cdot\vec{\mu} \oplus b\Delta_j^*.$$

Apply Wagner's generalized birthday algorithm to the lists of psum at $D-2$ free copath positions.[4] Producing M outputs requires input list size

$$Q = 2^{k/a_D} M^{1/a_D} = 2^{k/a_D + 2n/(a_D b_\tau)}.$$

The per-tree cost is $O(DQ)$, and the cost over all trees is $O(\tau DQ)$.

(7) Among the candidates of a τ-tuple of PPRF keys, find one such that

$$\vec{F}_x(\Delta^*) = \vec{s}^* \wedge \vec{F}_y(\Delta^*) = \vec{w}^*.$$

There are $M^\tau = 2^{2n\tau/b_\tau}$ possible τ-tuples. Wagner's generalized birthday algorithm on $\vec{F}_x(\Delta^*)\|\vec{F}_y(\Delta^*)$ requires $O(\tau M)$ time, without enumerating all these tuples. If $\mathcal{H}_2(\mathsf{salt}\|0\|(D,0)\|x_{\min})$ and $\mathcal{H}_2(\mathsf{salt}\|1\|(D,0)\|x_{\max})$ remain the minimum and maximum leaf commitments, respectively, the forged signature passes verification. Assuming the other leaf commitments in the selected tuple have uniform hash values, a union bound gives a probability at most $\tau 2^{D+1-u}$ that one falls outside the fixed interval. This is a failure probability for preserving the MinMax commitment. Setting $u = D+\tau+10$ makes this bound $\tau 2^{-\tau-9}$; it does not by itself establish the success probability of the candidate searches.

Under the above model, the total time cost is

$$O\Big(2^{u+1} + \gamma\cdot n^3 + 2^{(\tau+2)D}/\gamma + \tau 2^D Q + \tau DQ + \tau M\Big).$$

[4] The pre-tree fixes one copath node in every main tree. A second node is fixed by $x_{\min}$ or $x_{\max}$ in the first two trees; in each remaining tree, we fix one additional copath node to use a uniform bound of $D-2$ free positions.

Set $u = D + \tau + 10$ and $\gamma = 2^{(\tau/2+1)D}/n^{3/2}$, with Q and M as in (4). Since $2^D + D \le 2^{D+1}$, the resulting cost is $O(T)$, where

$$T = 2^{D+\tau+11} + n^{3/2}2^{(\tau/2+1)D+1} + \tau 2^{D+1+k/a_D+2n/(a_D b_\tau)} + \tau 2^{2n/b_\tau}. \tag{5}$$

Evaluating this expression for $n = 130$, $k = 257$, and the grid $D = 9, \ldots, 15$, $\tau = \lceil 128/D \rceil$, the resulting estimates range from $2^{95.2}$ to $2^{121.2}$, up to constant factors.

References

[1] Carlos Aguilar-Melchor, Nicolas Gama, James Howe, Andreas Hülsing, David Joseph, and Dongze Yue. 2023. The return of the SDitH. In *EUROCRYPT 2023*. Springer, 564–596.

[2] Carsten Baum, Ward Beullens, Lennart Braun, Cyprien Delpech de Saint Guilhem, Michael Klooß, Christian Majenz, Shibam Mukherjee, Emmanuela Orsini, Sebastian Ramacher, Christian Rechberger, et al. 2025. FAEST v2: Algorithm Specifications. (2025).

[3] Carsten Baum, Ward Beullens, Shibam Mukherjee, Emmanuela Orsini, Sebastian Ramacher, Christian Rechberger, Lawrence Roy, and Peter Scholl. 2024. One Tree to Rule Them All: Optimizing GGM Trees and OWFs for Post-Quantum Signatures. Cryptology ePrint Archive, Paper 2024/490. https://eprint.iacr.org/2024/490

[4] Carsten Baum, Lennart Braun, Cyprien Delpech de Saint Guilhem, Michael Klooß, Christian Majenz, Shibam Mukherjee, Emmanuela Orsini, Sebastian Ramacher, Christian Rechberger, Lawrence Roy, and Peter Scholl. 2023. FAEST. Technical report, National Institute of Standards and Technology, 2023. available at https://csrc.nist.gov/Projects/pqc-dig-sig/round-1-additional-signatures.

[5] Carsten Baum, Lennart Braun, Cyprien Delpech de Saint Guilhem, Michael Klooß, Emmanuela Orsini, Lawrence Roy, and Peter Scholl. 2023. Publicly Verifiable Zero-Knowledge and Post-Quantum Signatures from VOLE-in-the-Head. In *CRYPTO 2023*, Helena Handschuh and Anna Lysyanskaya (Eds.). Springer Nature Switzerland, Cham, 581–615.

[6] Carsten Baum, Lennart Braun, Alexander Munch-Hansen, and Peter Scholl. 2022. Moz$\mathbb{Z}_{2^k}$arella: Efficient Vector-OLE and Zero-Knowledge Proofs Over $\mathbb{Z}_{2^k}$. In *Advances in Cryptology – CRYPTO 2022*, Yevgeniy Dodis and Thomas Shrimpton (Eds.). Springer Nature Switzerland, Cham, 329–358.

[7] Ryad Benadjila, Charles Bouillaguet, Thibauld Feneuil, and Matthieu Rivain. 2024. MQOM: MQ on My Mind. NIST PQC Standardization for Additional Signature Round-2.

[8] Alexander Bienstock, Sarvar Patel, Joon Young Seo, and Kevin Yeo. 2023. Near-Optimal Oblivious Key-Value Stores for Efficient PSI, PSU and Volume-Hiding Multi-Maps. In *32nd USENIX Security Symposium (USENIX Security 23)*. USENIX Association, Anaheim, CA, 301–318. https://www.usenix.org/conference/usenixsecurity23/presentation/bienstock

[9] Dung Bui. 2024. Shorter VOLEitH Signature from Multivariate Quadratic. *Cryptology ePrint Archive* (2024).

[10] Dung Bui, Eliana Carozza, Geoffroy Couteau, Dahmun Goudarzi, and Antoine Joux. 2024. Short Signatures from Regular Syndrome Decoding, Revisited. Cryptology ePrint Archive, Paper 2024/252. https://eprint.iacr.org/2024/252

[11] Dung Bui, Kelong Cong, and Cyprien Delpech de Saint Guilhem. 2024. Improved All-but-One Vector Commitment with Applications to Post-Quantum Signatures. Cryptology ePrint Archive, Paper 2024/097. https://eprint.iacr.org/2024/097

[12] Hongrui Cui, Hanlin Liu, Di Yan, Kang Yang, Yu Yu, and Kaiyi Zhang. 2024. ResolveD: Shorter signatures from regular syndrome decoding and vole-in-the-head. In *PKC 2024*. Springer, 229–258.

[13] Christoph Dobraunig, Daniel Kales, Christian Rechberger, Markus Schofnegger, and Greg Zaverucha. 2022. Shorter Signatures Based on Tailor-Made Minimalist Symmetric-Key Crypto. In *ACM CCS 2022*. Association of Computing Machinery, 843–857. https://www.microsoft.com/en-us/research/publication/shorter-signatures-based-on-tailor-made-minimalist-symmetric-key-crypto/

[14] Thibauld Feneuil and Matthieu Rivain. 2026. On the Security of MPC-in-the-Head Signatures with Correlated GGM Trees. Cryptology ePrint Archive, Paper 2026/615.

[15] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. 1986. How to Construct Random Functions. *J. ACM* 33, 4 (aug 1986), 792–807. doi:10.1145/6490.6503

[16] Xiaojie Guo, Kang Yang, Xiao Wang, Wenhao Zhang, Xiang Xie, Jiang Zhang, and Zheli Liu. 2023. Half-tree: Halving the cost of tree expansion in cot and dpf. In *EUROCRYPT 2023*. Springer, 330–362.

[17] Janik Huth and Antoine Joux. 2024. MPC in the Head Using the Subfield Bilinear Collision Problem. In *Advances in Cryptology – CRYPTO 2024*, Leonid Reyzin and Douglas Stebila (Eds.). Springer Nature Switzerland, Cham, 39–70.

[18] Janik Huth and Antoine Joux. 2026. VOLE-in-the-Head Signatures from Subfield Bilinear Collisions. In *Advances in Cryptology – ASIACRYPT 2025*, Goichiro Hanaoka and Bo-Yin Yang (Eds.). Springer Nature Singapore, Singapore, 3–34.

[19] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. 2007. Zero-knowledge from Secure Multiparty Computation. In *ACM STOC 2007*. 21–30.

[20] Daniel Kales and Greg Zaverucha. 2020. An Attack on Some Signature Schemes Constructed from Five-Pass Identification Schemes. In *Cryptology and Network Security*, Stephan Krenn, Haya Shulman, and Serge Vaudenay (Eds.). Springer International Publishing, Cham, 3–22.

[21] Seongkwang Kim, Jincheol Ha, Mincheol Son, Byeonghak Lee, Dukjae Moon, Joohee Lee, Sangyub Lee, Jihoon Kwon, Jihoon Cho, Hyojin Yoon, and Jooyoung Lee. 2023. AIM: Symmetric Primitive for Shorter Signatures with Stronger Security. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security* (Copenhagen, Denmark) *(CCS '23)*. Association for Computing Machinery, New York, NY, USA, 401–415. doi:10.1145/3576915.3616579

[22] Seongkwang Kim, Byeonghak Lee, and Mincheol Son. 2024. Relaxed Vector Commitment for Shorter Signatures. Cryptology ePrint Archive, Paper 2024/1004. https://eprint.iacr.org/2024/1004

[23] Haruhisa Kosuge and Keita Xagawa. 2025. New Security Proofs of MPC-in-the-Head Signatures in the Quantum Random Oracle Model. Cryptology ePrint Archive, Paper 2025/1999. https://eprint.iacr.org/2025/1999

[24] Haruhisa Kosuge and Keita Xagawa. 2026. Towards Formal Security Proofs of MQOM. Cryptology ePrint Archive, Paper 2026/629.

[25] Jacques Patarin. 2008. The coefficients H technique. In *International Workshop on Selected Areas in Cryptography*. Springer, 328–345.

[26] Peter Rindal and Phillipp Schoppmann. 2021. VOLE-PSI: Fast OPRF and Circuit-PSI from Vector-OLE. In *EUROCRYPT 2021*. Springer, Cham, 901–930.

[27] Lawrence Roy. 2022. SoftSpokenOT: Communication–Computation Tradeoffs in OT Extension. *Cryptology ePrint Archive* (2022).

[28] Chenkai Weng, Kang Yang, Jonathan Katz, and Xiao Wang. 2021. Wolverine: Fast, Scalable, and Communication-Efficient Zero-Knowledge Proofs for Boolean and Arithmetic Circuits. In *2021 IEEE Symposium on Security and Privacy (S&P)*. 1074–1091. doi:10.1109/SP40001.2021.00056

[29] Kang Yang, Pratik Sarkar, Chenkai Weng, and Xiao Wang. 2021. QuickSilver: Efficient and Affordable Zero-Knowledge Proofs for Circuits and Polynomials over Any Field. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security* (Virtual Event, Republic of Korea) *(CCS '21)*. Association for Computing Machinery, New York, NY, USA, 2986–3001. doi:10.1145/3460120.3484556

[30] Greg Zaverucha, Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, Jonathan Katz, Xiao Wang, Vladmir Kolesnikov, and Daniel Kales. 2022. Picnic. Technical report, National Institute of Standards and Technology, 2020.

A Generative AI Usage

Parts of the implementation described in Section 4.3 were developed with the assistance of Codex, including implementation-level suggestions. The authors manually reviewed, modified, and tested all AI-assisted code. The final implementation was validated by independently running the experimental pipeline and confirming that the reported experimental results were reproducible.

Parts of the manuscript were edited with the assistance of generative AI tools (ChatGPT, Gemini, Claude Code, and Codex) for language polishing, including phrasing, grammar, typo correction, and clarity. All AI-assisted suggestions were reviewed and revised by the authors, who take full responsibility for the correctness and integrity of the manuscript.

B Open Science

The rFAEST implementation and the accompanying stage-wise profiling code used to obtain the experimental results in Section 4.3 are publicly available at https://github.com/samsungsds-research-papers/rFAEST.

C Ethical Considerations

This work does not involve human participants or human data, nor does it study or disclose vulnerabilities in real-world deployed systems. We therefore identify no specific ethical concerns arising from this research.

Scheme	Scheme Parameters						Time (ms)		Sig. (B)
	τ	$(\tau_0, 2^{k_0})$	$(\tau_1, 2^{k_1})$	B	T_{open}	w	Sign	Verify	
FAEST-s	16	$(4, 2^{12})$	$(12, 2^{11})$	16	162	12	18.853	12.630	11260
rFAEST-s	16	$(4, 2^{12})$	$(12, 2^{11})$	48	180	12	8.538	7.223	10988
FAEST-EM-s	16	$(8, 2^{12})$	$(8, 2^{11})$	16	162	8	12.055	10.613	9340
rFAEST-EM-s	16	$(8, 2^{12})$	$(8, 2^{11})$	48	184	8	8.192	8.109	9548
FAEST-f	24	$(16, 2^{8})$	$(8, 2^{7})$	16	163	8	2.259	1.939	14948
rFAEST-f	24	$(16, 2^{8})$	$(8, 2^{7})$	40	184	8	1.457	1.257	14372
FAEST-EM-f	24	$(16, 2^{8})$	$(8, 2^{7})$	16	176	8	1.534	1.373	12380
rFAEST-EM-f	24	$(16, 2^{8})$	$(8, 2^{7})$	40	184	8	1.280	1.110	12068

Table 3: Performance comparison between 192-bit parameter sets of FAEST(-EM) and rFAEST(-EM). Times are averaged over 100 runs.

Scheme	Scheme Parameters						Time (ms)		Sig. (B)
	τ	$(\tau_0, 2^{k_0})$	$(\tau_1, 2^{k_1})$	B	T_{open}	w	Sign	Verify	
FAEST-s	22	$(8, 2^{12})$	$(14, 2^{11})$	16	245	6	25.052	25.060	20696
rFAEST-s	22	$(8, 2^{12})$	$(14, 2^{11})$	48	250	6	13.952	13.920	19536
FAEST-EM-s	22	$(8, 2^{12})$	$(14, 2^{11})$	16	218	6	19.554	18.705	17984
rFAEST-EM-s	22	$(8, 2^{12})$	$(14, 2^{11})$	48	250	6	13.746	13.418	18392
FAEST-f	32	$(24, 2^{8})$	$(8, 2^{7})$	16	246	8	3.495	3.267	26548
rFAEST-f	32	$(24, 2^{8})$	$(8, 2^{7})$	40	248	8	2.336	2.070	24660
FAEST-EM-f	32	$(24, 2^{8})$	$(8, 2^{7})$	16	234	8	2.763	2.527	23476
rFAEST-EM-f	32	$(24, 2^{8})$	$(8, 2^{7})$	40	248	8	2.255	2.025	22996

Table 4: Performance comparison between 256-bit parameter sets of FAEST(-EM) and rFAEST(-EM). Times are averaged over 100 runs.

D Additional Performance Results

D.1 Performance Comparison at the 192- and 256-bit Security Levels

This section presents performance comparisons at the 192- and 256-bit security levels. For each security level, we set w to the value used in the corresponding FAEST parameter set. As in the 128-bit setting, we replace the original choice $B = 16$ in the VOLE consistency check with $B = 48$ for the short parameter sets and $B = 40$ for the fast parameter sets.

D.2 Detailed Signing Time Breakdown Results

The stage-breakdown benchmarks partition the signing time into six components. The *Expand tree* stage measures the cost of expanding the GGM trees from the root seeds. The *Gen. & Commit leaf* stage measures the subsequent leaf derivation step that converts these final tree nodes into the leaf values and commitments. The *Compress leaf com.* stage isolates the hash computation over the leaf commitments. The *VOLE & QS* stage includes the computation of the small-VOLE sender correlation data from leaf values and the generation of the QuickSilver proof. The *Open & Grinding* stage measures the final challenge-search and opening phase. Finally, the *Other* stage collects the remaining measured work, such as secret-key unpacking, transcript hashing, seed/IV derivation, and signature packing.

	short		fast	
Stage	FAEST	rFAEST	FAEST	rFAEST
Expand tree	1210.4	1160.9	144.5	140.7
Gen. & Commit leaf	2814.0	1063.5	360.1	132.4
Compress leaf com.	5334.1	1795.3	677.2	232.5
VOLE & QS	3477.5	3489.5	800.7	792.8
Open & Grinding	6270.9	1119.0	145.3	63.3
Other	90.3	94.0	104.7	106.5
Total	19197.3	8722.1	2232.5	1468.1

Table 5: Signing time breakdown for FAEST and rFAEST at the 192-bit security level. Times are averaged over 100 runs and reported in microseconds.

	short		fast	
Stage	FAEST	rFAEST	FAEST	rFAEST
Expand tree	2005.0	2146.3	243.0	245.2
Gen. & Commit leaf	4912.2	1543.5	535.9	174.9
Compress leaf com.	11685.2	3866.8	1258.7	430.4
VOLE & QS	6388.1	6267.5	1234.0	1207.6
Open & Grinding	155.9	23.8	84.5	67.1
Other	151.6	155.2	171.1	175.7
Total	25298.0	14003.1	3527.2	2300.9

Table 6: Signing time breakdown for FAEST and rFAEST at the 256-bit security level. Times are averaged over 100 runs and reported in microseconds.

	short		fast	
Stage	FAEST-EM	rFAEST-EM	FAEST-EM	rFAEST-EM
Expand tree	253.9	213.7	33.5	33.7
Gen. & Commit leaf	272.0	211.9	38.9	34.3
Compress leaf com.	1462.2	560.0	150.6	78.5
VOLE & QS	726.8	640.5	179.8	178.7
Open & Grinding	1072.9	265.2	85.7	63.2
Other	36.8	38.6	39.7	41.0
Total	3824.5	1929.9	528.3	429.5

Table 7: Signing time breakdown for FAEST-EM and rFAEST-EM at the 128-bit security level. Times are averaged over 100 runs and reported in microseconds.

E Complete Set of Algorithms in rVOLEitH

E.1 Sign and Verify

The signing and verification algorithms for rVOLEitH are presented in Algorithm 4 and Algorithm 5, respectively. In the Sign algorithm, we do not explicitly define the function ExtendWitness, which extracts the witness of the one-way function. At a high level, starting from the secret key sk, the output of each multiplication gate within the one-way function is treated as a witness. To aid understanding, the function ExtendWitness for AES is formally specified as FAEST.AES.ExtendWitness in FAEST [4].

	short		fast	
Stage	FAEST-EM	rFAEST-EM	FAEST-EM	rFAEST-EM
Expand tree	1402.0	1364.0	144.1	139.2
Gen. & Commit leaf	1500.2	1272.3	155.4	132.3
Compress leaf com.	4262.7	2133.5	453.9	231.8
VOLE & QS	3421.6	3062.5	641.3	642.4
Open & Grinding	1355.9	65.2	84.7	53.0
Other	77.9	81.7	87.6	90.3
Total	12020.4	7979.2	1567.1	1289.0

Table 8: Signing time breakdown for FAEST-EM and rFAEST-EM at the 192-bit security level. Times are averaged over 100 runs and reported in microseconds.

	short		fast	
Stage	FAEST-EM	rFAEST-EM	FAEST-EM	rFAEST-EM
Expand tree	2039.4	2122.3	239.6	245.8
Gen. & Commit leaf	2609.4	1541.7	269.2	175.6
Compress leaf com.	7772.1	3868.5	846.2	432.5
VOLE & QS	5950.1	5904.7	1184.9	1204.2
Open & Grinding	1205.3	21.3	92.4	66.5
Other	141.0	146.1	158.0	163.5
Total	19717.2	13604.6	2790.3	2288.2

Table 9: Signing time breakdown for FAEST-EM and rFAEST-EM at the 256-bit security level. Times are averaged over 100 runs and reported in microseconds.

E.2 Unchanged Algorithms

The functions OWFProve and OWFVerify extend FAEST.AES.AESProve and FAEST.AES.AESVerify in FAEST to support arbitrary circuits. Apart from this generalization, the computation remains the same. For an $m \times n$-matrix $\mathbf{A}$, we denote by $\mathbf{A}|_{[i]}$ the i-th row, and we denote by $\mathbf{A}|_{[i:j]}$ the row slice from the i-th row (inclusive) to the j-th row (exclusive).

The universal hash functions VHash and ZHash are the same as the universal hash functions VOLEHash and ZKHash in FAEST, respectively. As we do not need exact algorithms of them, we omit the pseudocode of them. Instead, we specify input-output sizes for a compatibility purpose as follows.

- $\mathsf{VHash} : \{0,1\}^{5\lambda+64} \times \{0,1\}^{\ell+2\lambda+B} \to \{0,1\}^{\lambda+B}$
- $\mathsf{ZHash} : \{0,1\}^{3\lambda+64} \times \{0,1\}^{\ell+\lambda} \to \{0,1\}^{\lambda}$

F Proof of Lemma 3.6

Before we provide the proofs, we provide a useful fact. This fact was introduced in [22], but we slightly modified some numbers and adapted the proof.

Chernoff bound. Let X be a random variable following the binomial distribution $B(Q,p)$. Then, the probability of the tail of X is upper bounded by a Chernoff bound

$$\Pr[X > (1+\delta)\mu] < \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{\mu} = \frac{1}{e^{\mu}} \cdot \left(\frac{e\mu}{(1+\delta)\mu}\right)^{(1+\delta)\mu}$$

Algorithm 4. Sign(msg, sk, pk) - rVOLEitH, signing algorithm.

```
// Randomness generation
ρ ←$ {0,1}^λ (ρ ← 0^λ for deterministic signature)
μ ← H_1(pk‖msg)
(iv, root) ∈ {0,1}^{|iv|} × ({0,1}^λ)^τ ← H_3(sk‖μ‖ρ)
// Commitment and VOLE derandomization
w ← ExtendWitness(sk, pk)
(com, decom_0, ..., decom_{τ-1}, c_1, ..., c_{τ-1}, u, V) ←
  VoleCom(iv, root; ℓ + 2λ + B)
chall_1 ← H_2^1(μ‖com‖c_1‖ ... ‖c_{τ-1}‖iv; 5λ + 64)
// VOLE consistency check and witness commitment
ũ ← VHash(chall_1, u)
Ṽ ← VHash(chall_1, V)
h_V ← H_1(Ṽ)
d ← w ⊕ u[: ℓ]
chall_2 ← H_2^2(chall_1‖ũ‖h_V‖d; 3λ + 64)
// Proof of knowledge of OWF witness
u ← u[: ℓ + λ]
V ← V|_{[:ℓ+λ]}
(ã, b̃) ← OWFProve(w, u, V, pk, chall_2)
ctr ← 0, reject ← 1
while reject do
    chall_3 ← H_2^3(chall_2‖ã‖b̃‖ctr; λ)
    if chall_3[λ − w : λ] ≠ 0^w then
        ctr ← ctr + 1
    else
        reject ← 0
for i ∈ [τ] do
    s_i ← chall_3[ki : k(i + 1)]
    pdecom_i ← VSC.Open(iv, decom_i, s_i)
return σ := ((c_i)_{i∈[1,τ]}, ũ, d, ã, (pdecom_i)_{i∈[τ]}, chall_3, iv, ctr)
```

for any $\delta > 0$, where e is Euler's number and $\mu = Q \cdot p$ is the expectation of X.

H-coefficient Technique. The H-coefficient technique [25] is a method which provides a way to bound the distinguishing advantage of an adversary interacting with an idealized cryptographic system and a real algorithm. The core idea of the H-coefficient technique is to partition the set of possible transcripts (i.e., sequences of queries and responses) into two subsets: "good" and "bad" transcripts. The probability of bad transcripts can be shown to be negligible, while the probability of distinguishing between the distributions of good transcripts can be tightly bounded. The following lemma formally states the H-coefficient technique.

Lemma F.1 (H-Coefficient Lemma). *Let $\mathcal{A}$ be an algorithm that interacts with either an ideal world or a real world and tries to distinguish two worlds. Let T_{id} and T_{re} be the distribution of transcript in the ideal world and the real world, respectively, and let $\mathcal{T}$ denote the set of all attainable transcripts in the ideal world. Suppose there exists a partition $\mathcal{T}_{\mathrm{Good}}$ (good transcripts) and $\mathcal{T}_{\mathrm{Bad}}$ (bad transcripts)*

Algorithm 5. Verify(msg, σ, pk) - rVOLEitH, verification algorithm.

// Parsing signature
Parse $\sigma = ((\mathbf{c}_i)_{i\in[1,\tau]}, \widetilde{\mathbf{u}}, \mathbf{d}, \widetilde{a}, (\mathsf{pdecom}_i)_{i\in[\tau]}, \mathsf{chall}_3, \mathsf{iv}, \mathsf{ctr})$
$\mu \leftarrow \mathsf{H}_1(\mathsf{pk}\|\mathsf{msg})$
// Check grinding
if $\mathsf{chall}_3[\lambda - w : \lambda] \neq 0^w$ **then**
  return REJECT
// Reconstruct VOLEs and check commitment
$(\mathsf{com}, \mathbf{Q}'_0, \ldots, \mathbf{Q}'_{\tau-1}) \leftarrow$ $\mathsf{VoleRecon}(\mathsf{iv}, \mathsf{chall}_3, (\mathsf{pdecom}_i)_{i\in[\tau]}; \ell + 2\lambda + B)$
$\mathsf{chall}_1 \leftarrow \mathsf{H}_2^1(\mu\|\mathsf{com}\|\mathbf{c}_1\|\ldots\|\mathbf{c}_{\tau-1}\|\mathsf{iv}; 5\lambda + 64)$
// Apply the VOLE correction values and check consistency
$\mathbf{Q}_0 \leftarrow \mathbf{Q}'_0$
for $i \in [\tau]$ **do**
  $(\delta_0, \ldots, \delta_{k-1}) \leftarrow \mathsf{chall}_3[ki : k(i+1)]$
  $\widetilde{\mathbf{D}}_i \leftarrow [\delta_0 \cdot \widetilde{\mathbf{u}} \ldots \delta_{k-1} \cdot \widetilde{\mathbf{u}}] \in \{0,1\}^{(\lambda+B)\times k}$
  if $i \geq 1$ **then**
    $\mathbf{Q}_i \leftarrow \mathbf{Q}'_i \oplus [\delta_0 \cdot \mathbf{c}_i \ldots \delta_{k-1} \cdot \mathbf{c}_i]$
Denote $\mathbf{Q} = [\mathbf{Q}_0 \ldots \mathbf{Q}_{\tau-1}]$, and $\widetilde{\mathbf{D}} = [\widetilde{\mathbf{D}}_0 \ldots \widetilde{\mathbf{D}}_{\tau-1}]$
$\widetilde{\mathbf{Q}} \leftarrow \mathsf{VHash}(\mathsf{chall}_1, \mathbf{Q})$
$h_V \leftarrow \mathsf{H}_1(\widetilde{\mathbf{Q}} \oplus \widetilde{\mathbf{D}})$
$\mathsf{chall}_2 \leftarrow \mathsf{H}_2^2(\mathsf{chall}_1\|\widetilde{\mathbf{u}}\|h_V\|\mathbf{d}; 3\lambda + 64)$
// Compute OWF consistency values
$\widetilde{b} \leftarrow \mathsf{OWFVerify}(\mathbf{d}, \mathbf{Q}|_{[:\ell+\lambda]}, \mathsf{chall}_2, \mathsf{chall}_3, \widetilde{a}, \mathsf{pk})$
$\mathsf{chall}'_3 \leftarrow \mathsf{H}_2^3(\mathsf{chall}_2\|\widetilde{a}\|\widetilde{b}\|\mathsf{ctr}; \lambda)$
if $\mathsf{chall}_3 = \mathsf{chall}'_3$ **then**
  return ACCEPT
else
  return REJECT

Algorithm 6. $\mathsf{OWFProve}(\mathbf{w}, \mathbf{u}, \mathbf{V}, \mathsf{pk}, \mathsf{chall})$

Represent the OWF as a circuit C with C multiplication gates
for $i \in [\ell]$ **do**
  $w[i] \leftarrow \mathbf{w}[i]$
for $i \in [\ell + \lambda]$ **do**
  $v[i] \leftarrow \mathbf{V}|_{[i]}$
$j \leftarrow 0$
for *gate* $g \in C$ **do**
  Denote the input wires x and y
  Denote the output wire z
  if g *is* ADD **then**
    $w_z \leftarrow w_x + w_y$
    $v_z \leftarrow v_x + v_y$
  if g *is* MUL **then**
    $A_{1,j} \leftarrow v_x \cdot v_y$
    $A_{2,j} \leftarrow (w_x + v_x)(w_y + v_y) - w_z - v_z - A_{1,j}$
    $j \leftarrow j + 1$
$\mathbf{a}_1 \leftarrow (A_{1,0}, \ldots, A_{1,C-1})$
$\mathbf{a}_2 \leftarrow (A_{2,0}, \ldots, A_{2,C-1})$
for $i \in [\ell, \ell + \lambda]$ **do**
  $u[i - \ell] \leftarrow \mathbf{u}[i]$
$u^* \leftarrow \sum_{i\in[\lambda]} u[i]\alpha_\lambda^i$
$v^* \leftarrow \sum_{i\in[\lambda]} v[\ell + i]\alpha_\lambda^i$ // $\alpha_\lambda \in \mathbb{F}_{2^\lambda}$
$\widetilde{a} \leftarrow \mathsf{ZHash}(\mathsf{chall}, \mathbf{a}_2\|u^*)$
$\widetilde{b} \leftarrow \mathsf{ZHash}(\mathsf{chall}, \mathbf{a}_1\|v^*)$
return $(\widetilde{a}, \widetilde{b})$

of $\mathcal{T}$, and constants ϵ and δ such that for any $\gamma \in \mathcal{T}_{\mathrm{Good}}$,

$$\Pr[T_{\mathrm{id}} \in \mathcal{T}_{\mathrm{Bad}}] \leq \epsilon, \qquad \frac{\Pr[\gamma = T_{\mathrm{re}}]}{\Pr[\gamma = T_{\mathrm{id}}]} \geq 1 - \delta.$$

Then, the distinguishing advantage of $\mathcal{A}$ in distinguishing $\mathcal{I}$ from $\mathcal{R}$ is bounded by:

$$\mathbf{Adv}^{\mathrm{dist}}_{\mathcal{I},\mathcal{R}}(\mathcal{A}) \leq \epsilon + \delta.$$

Security of F^{ro}. Fix an adversary $\mathcal{A}$. We will first consider $\mathbf{Adv}^{\mathrm{mcoll}}_{\mathsf{F}}$. For each query $\mathsf{F}_{\mathsf{com}}(s, \mathsf{iv}, i)$, the output is chosen uniformly at random from $\{0,1\}^\lambda$. Let X be a random variable that follows binomial distribution $B(2^\lambda, 2^{-\lambda})$. Then, for each (iv, i) and $z \in \{0,1\}^\lambda$,

$$\Pr\left[\left|\left\{s \in \{0,1\}^\lambda : \mathsf{F}_{\mathsf{com}}(s, \mathsf{iv}, i) = z\right\}\right| > u\right] = \Pr[X > u] < \frac{1}{e}\cdot\left(\frac{e}{u}\right)^u$$

by a Chernoff bound. Since there are at most $Q_I N + Q_P$ of (iv, i), we have

$$\mathbf{Adv}^{\mathrm{mcoll}}_{\mathsf{F}^{\mathsf{ro}}}(Q_I, Q_P, u) \leq 2^\lambda \cdot \frac{Q_I N + Q_P}{e} \cdot \left(\frac{e}{u}\right)^u$$

Next, consider $\mathbf{Adv}^{\mathrm{coll2}}_{\mathsf{F}}$. Let $\mathsf{F}_{\mathsf{com}\circ\mathsf{tree}}(s, \mathsf{iv}, i) = \mathsf{F}_{\mathsf{com}}(y_0, \mathsf{iv}, 2i) \,\|\, \mathsf{F}_{\mathsf{com}}(y_1, \mathsf{iv}, 2i+1)$ for $(y_0, y_1) = \mathsf{F}_{\mathsf{tree}}(s, \mathsf{iv}, i)$. Then, $\mathsf{F}_{\mathsf{com}\circ\mathsf{tree}}(x) = \mathsf{F}_{\mathsf{com}\circ\mathsf{tree}}(x')$ occurs when there exists a collision in either $\mathsf{F}_{\mathsf{tree}}$ or $\mathsf{F}_{\mathsf{com}}$. Since there are at most Q_P^2 of such x and x', we have

$$\mathbf{Adv}^{\mathrm{coll2}}_{\mathsf{F}^{\mathsf{ro}}}(Q_P) \leq \frac{4Q_P^2}{2^{2\lambda}}$$

Next, consider $\mathbf{Adv}^{\text{4-xor}}_{\mathsf{F}}$. Fix $\mathsf{iv}, s_0, s_1, s'_0, s'_1, z_0, z_1 \in \{0,1\}^\lambda$, $j_0, j_1 \in [N]$. Then, we have

$$\Pr\left[\begin{cases}\bigwedge_{a\in[2]} \mathsf{F}_{\mathsf{com}}(s_a, \mathsf{iv}, j_a) = \mathsf{F}_{\mathsf{com}}(s'_a, \mathsf{iv}, j_a) = z_a, \\ \bigoplus_{a\in[2]} \mathsf{F}_{\mathsf{tape}}(s_a, \mathsf{iv}, j_a) = \bigoplus_{a\in[2]} \mathsf{F}_{\mathsf{tape}}(s'_a, \mathsf{iv}, j_a)\end{cases}\right] = \frac{1}{2^{\ell+4\lambda}}.$$

As there are at most $Q_I + Q_P$ candidates of iv, we have

$$\mathbf{Adv}^{\text{4-xor}}_{\mathsf{F}^{\mathsf{ro}}}(Q_I, Q_P) \leq \frac{(Q_I + Q_P)N^2 2^{6\lambda}}{2^{\ell+4\lambda}} \leq \frac{(Q_I + Q_P)N^2}{2^{\ell-2\lambda}}$$

Finally, consider $\mathbf{Adv}^{\mathrm{prg}}_{\mathsf{F}}$. Since two worlds, $b = 0$ and $b = 1$, differ only when there exists an iv-key collision, we have

$$\mathbf{Adv}^{\mathrm{prg}}_{\mathsf{F}}(Q_I, Q_P) \leq \frac{Q_I \cdot Q_P}{2^{2\lambda}}.$$

Security of $\mathsf{F}^{\mathsf{ctr}}$. Fix an adversary $\mathcal{A}$. We will first consider $\mathbf{Adv}^{\mathrm{mcoll}}_{\mathsf{F}}$. Since $\mathcal{A}$ can query at most Q_P times and $Q_I N + Q_P \leq 2^{\lambda-1}$, for each query (either by $\mathcal{A}$ or an additional query in the proof), $E(s, x)$

Algorithm 7. $\mathsf{OWFVerify}(\mathbf{d}, \mathbf{Q}, \mathsf{chall}_2, \mathsf{chall}_3, \widetilde{a}, \mathsf{pk})$

Represent the OWF as a circuit C with C multiplication gates
$\Delta \leftarrow \mathsf{chall}_3$
Write $\mathbf{Q} = [\mathbf{Q}_0, \dots, \mathbf{Q}_{\tau-1}]$
for $i \in [\tau]$ **do**
  $(\delta_0, \dots, \delta_{k-1}) \leftarrow \mathsf{chall}_3[ki : k(i+1)]$
  Write $\mathbf{Q}_i = [\mathbf{q}_{i,0}, \dots, \mathbf{q}_{i,k-1}]$
  for $j \in [k]$ **do**
    $\mathbf{q}_{i,j}|_{[:\ell]} \leftarrow \mathbf{q}_{i,j}|_{[:\ell]} \oplus \delta_j \cdot \mathbf{d}$
for $i \in [\ell + \lambda]$ **do**
  $q[i] \leftarrow \mathbf{Q}|_{[i]}$
$j \leftarrow 0$
for *gate* $g \in C$ **do**
  Denote the input wires x and y
  Denote the output wire z
  if g *is* ADD **then**
    $q_z \leftarrow q_x + q_y$
  if g *is* MUL **then**
    $B_j \leftarrow q_x q_y - q_z \cdot \Delta$
    $j \leftarrow j + 1$
$\mathbf{b} \leftarrow (B_0, \dots, B_{C-1})$
$q^* \leftarrow \sum_{i \in [\lambda]} q[\ell + i]\alpha_\lambda^i$ // $\alpha_\lambda \in \mathbb{F}_{2^\lambda}$
$\widetilde{q} \leftarrow \mathsf{ZHash}(\mathsf{chall}_2, \mathbf{b} \| q^*)$
return $\widetilde{q} - \widetilde{a} \cdot \Delta$

or $E^{-1}(s, y)$, the output is chosen uniformly at random from the set of size at least $2^{\lambda-1}$. Let X be a random variable that follows $B(2^\lambda, 2^{-\lambda+1})$. Then, for each $x, z \in \{0,1\}^\lambda$,

$$\Pr\left[\left|\left\{s \in \{0,1\}^\lambda : E(s,x) = z\right\}\right| > u\right] = \Pr[X > u] < \frac{1}{e^2} \cdot \left(\frac{2e}{u}\right)^u$$

by a Chernoff bound. Since there are at most $(Q_I N + Q_P)$ of $\mathsf{iv} \oplus \langle i, 2\rangle$, we have

$$\mathbf{Adv}^{\mathsf{mcoll}}_{\mathsf{F}^{\mathsf{ctr}}}(Q_I, Q_P, u) \le 2^\lambda \cdot \frac{Q_I N + Q_P}{e^2} \cdot \left(\frac{2e}{u}\right)^u$$

Next, consider $\mathbf{Adv}^{\mathsf{coll2}}_{\mathsf{F}}$. Let $\mathsf{F}_{\mathsf{com}\circ\mathsf{tree}}(s, \mathsf{iv}, i) = \mathsf{F}_{\mathsf{com}}(y_0, \mathsf{iv}, 2i) \,\|\, \mathsf{F}_{\mathsf{com}}(y_1, \mathsf{iv}, 2i+1)$ for $(y_0, y_1) = \mathsf{F}_{\mathsf{tree}}(s, \mathsf{iv}, i)$. Then, $\mathsf{F}_{\mathsf{com}\circ\mathsf{tree}}(x) = \mathsf{F}_{\mathsf{com}\circ\mathsf{tree}}(x')$ occurs when there exists a collision in either $\mathsf{F}_{\mathsf{tree}}$ or $\mathsf{F}_{\mathsf{com}}$.

For $x \in \{0,1\}^\lambda$, there are at most $2N$ pairs of (iv, i) such that $\mathsf{iv} \oplus \langle i, 0\rangle = x$ or $\mathsf{iv} \oplus \langle i, 1\rangle = x$. Therefore, there are at most $2NQ_P$ of such (x, iv, i) such that $\mathsf{F}_{\mathsf{tree}}(x, \mathsf{iv}, i)$ is queried, and we get

$$\mathbf{Adv}^{\mathsf{coll2}}_{\mathsf{F}^{\mathsf{ctr}}}(Q_P) \le (2NQ_P)^2 \cdot \frac{16}{2^{2\lambda}}$$

Next, consider $\mathbf{Adv}^{\text{4-xor}}_{\mathsf{F}}$. Fix $\mathsf{iv}, s_0, s_1, s_0', s_1', z_0, z_1 \in \{0,1\}^\lambda$, $j_0, j_1 \in [N]$. Then, we have

$$\Pr\left[\begin{cases}\bigwedge_{a\in[2]} \mathsf{F}_{\mathsf{com}}(s_a, \mathsf{iv}, j_a) = \mathsf{F}_{\mathsf{com}}(s_a', \mathsf{iv}, j_a) = z_a, \\ \bigoplus_{a\in[2]} \mathsf{F}_{\mathsf{tape}}(s_a, \mathsf{iv}, j_a) = \bigoplus_{a\in[2]} \mathsf{F}_{\mathsf{tape}}(s_a', \mathsf{iv}, j_a)\end{cases}\right] = \left(\frac{2}{2^\lambda}\right)^{4+\lfloor \ell/\lambda \rfloor}.$$

As there are at most $(Q_I + Q_P)N$ candidates of (iv, j_0), we have

$$\mathbf{Adv}^{\text{4-xor}}_{\mathsf{F}^{\mathsf{ctr}}}(Q_I, Q_P) \le (Q_I + Q_P) N^2 2^{6\lambda} \cdot \left(\frac{2}{2^\lambda}\right)^{4+\lfloor \ell/\lambda \rfloor}$$

Finally, consider $\mathbf{Adv}^{\mathsf{prg}}_{\mathsf{F}}$. Assume that all keys used in Get queries are revealed at the end of the game. Let

$$\tau_p = (s_i, u_i, v_i = E_{s_i}(u_i))_{i \in [Q_P]},$$
$$\tau_g = (f_j, x_j = (\mathsf{iv}_j, k_j, a_j), y_j)_{j \in [Q_I]}$$

be the list of primitive queries and Get queries, respectively. Let $\tau = (\tau_p, \tau_g)$ be the transcript, and we say τ is bad if

- exists $j_1, j_2 \in [Q_I]$ such that $k_{j_1} = k_{j_2}$ and at least one of the blocks comprising y_{j_1} and y_{j_2} is the same;
- exists $i \in [Q_P]$, $j \in [Q_I]$, $c \in [\lceil \ell/\lambda \rceil + 3]$ such that $s_i = k_j$ and $u_i = \mathsf{iv}_j \oplus \langle a_j, c\rangle$ or v_i appears in y_j.

Let T_0 and T_1 be the distributions of τ when $b = 0$ and $b = 1$, and $\mathcal{T}_{\mathsf{Bad}}$ be the set of bad transcripts when $b = 1$. We have

$$\Pr[\mathsf{T}_1 \in \mathcal{T}_{\mathsf{Bad}}] \le \frac{Q_I(\lceil \ell/\lambda \rceil + 3)^2}{2^\lambda} + \frac{Q_I^2(\lceil \ell/\lambda \rceil + 3)^2}{2^{2\lambda}} + \frac{2Q_I \cdot Q_P(\lceil \ell/\lambda \rceil + 3)}{2^{2\lambda}}$$

Also, for $\tau \notin \mathcal{T}_{\mathsf{Bad}}$, we have

$$\frac{\Pr[\mathsf{T}_0 = \tau]}{\Pr[\mathsf{T}_1 = \tau]} \ge 1$$

Then, by Lemma F.1, we get

$$\mathbf{Adv}^{\mathsf{prg}}_{\mathsf{F}^{\mathsf{ctr}}}(Q_I, Q_P) \le \frac{Q_I(\lceil \ell/\lambda \rceil + 3)^2}{2^\lambda} + \frac{Q_I^2(\lceil \ell/\lambda \rceil + 3)^2}{2^{2\lambda}} + \frac{2Q_I \cdot Q_P(\lceil \ell/\lambda \rceil + 3)}{2^{2\lambda}}$$

Security of F^{dm}. Fix an adversary $\mathcal{A}$. We will first consider $\mathbf{Adv}^{\mathsf{mcoll}}_{\mathsf{F}}$. Since $\mathcal{A}$ can query at most Q_P times and $Q_I N + Q_P \le 2^{\lambda-1}$, for each query (either by $\mathcal{A}$ or an additional query in the proof), $E(s, x)$ or $E^{-1}(s, y)$, the output is chosen uniformly at random from the set of size at least $2^{\lambda-1}$. Let X be a random variable that follows $B(2^\lambda, 2^{-\lambda+1})$. Then, for each $x, z \in \{0,1\}^\lambda$,

$$\Pr\left[\left|\left\{s \in \{0,1\}^\lambda : E(s,x) = z\right\}\right| > u\right] = \Pr[X > u] < \frac{1}{e^2} \cdot \left(\frac{2e}{u}\right)^u$$

by a Chernoff bound. Since there are at most $(Q_I N + Q_P)$ of $\mathsf{iv} \oplus \langle i, 2\rangle$, we have

$$\mathbf{Adv}^{\mathsf{mcoll}}_{\mathsf{F}^{\mathsf{dm}}}(Q_I, Q_P, u) \le 2^\lambda \cdot \frac{Q_I N + Q_P}{e^2} \cdot \left(\frac{2e}{u}\right)^u$$

Next, consider $\mathbf{Adv}^{\mathsf{coll2}}_{\mathsf{F}}$. Let $\mathsf{F}_{\mathsf{com}\circ\mathsf{tree}}(s, \mathsf{iv}, i) = \mathsf{F}_{\mathsf{com}}(y_0, \mathsf{iv}, 2i) \,\|\, \mathsf{F}_{\mathsf{com}}(y_1, \mathsf{iv}, 2i+1)$ for $(y_0, y_1) = \mathsf{F}_{\mathsf{tree}}(s, \mathsf{iv}, i)$. Then, $\mathsf{F}_{\mathsf{com}\circ\mathsf{tree}}(x) =$

$\mathsf{F}_{\mathsf{com}\circ\mathsf{tree}}(x')$ occurs when there exists a collision in either $\mathsf{F}_{\mathsf{tree}}$ or $\mathsf{F}_{\mathsf{com}}$.

For $x \in \{0,1\}^\lambda$, there are at most $2N$ pairs of (iv, i) such that $\mathsf{iv} \oplus \langle i, 0\rangle = x$ or $\mathsf{iv} \oplus \langle i, 1\rangle = x$. Therefore, there are at most Q_P of such (x, iv, i) such that $\mathsf{F}_{\mathsf{tree}}(x, \mathsf{iv}, i)$ is queried, and we get

$$\mathbf{Adv}^{\mathsf{coll2}}_{\mathsf{F}^{\mathsf{dm}}}(Q_P) \le (2NQ_P)^2 \cdot \frac{16}{2^{2\lambda}}$$

Next, consider $\mathbf{Adv}^{\text{4-xor}}_{\mathsf{F}}$. Fix $\mathsf{iv}, s_0, s_1, s'_0, s'_1, z_0, z_1 \in \{0,1\}^\lambda$, $j_0, j_1 \in [N]$. Then, we have

$$\Pr\left[\begin{cases}\bigwedge_{a\in[2]} \mathsf{F}_{\mathsf{com}}(s_a, \mathsf{iv}, j_a) = \mathsf{F}_{\mathsf{com}}(s'_a, \mathsf{iv}, j_a) = z_a, \\ \bigoplus_{a\in[2]} \mathsf{F}_{\mathsf{tape}}(s_a, \mathsf{iv}, j_a) = \bigoplus_{a\in[2]} \mathsf{F}_{\mathsf{tape}}(s'_a, \mathsf{iv}, j_a)\end{cases}\right] = \left(\frac{2}{2^\lambda}\right)^{4+\lfloor \ell/\lambda\rfloor}.$$

As there are at most $Q_I + Q_P$ candidates of (iv, j_0), we have

$$\mathbf{Adv}^{\text{4-xor}}_{\mathsf{F}^{\mathsf{dm}}}(Q_I, Q_P) \le (Q_I + Q_P)N2^{6\lambda} \cdot \left(\frac{2}{2^\lambda}\right)^{4+\lfloor \ell/\lambda\rfloor}$$

Finally, consider $\mathbf{Adv}^{\mathsf{prg}}_{\mathsf{F}}$. Assume that all keys used in Get queries are revealed at the end of the game. Let

$$\tau_p = (s_i, u_i, v_i = E_{s_i}(u_i))_{i\in[Q_P]},$$
$$\tau_g = (f_j, x_j = (\mathsf{iv}_j, k_j, a_j), y_j)_{j\in[Q_I]}$$

be the list of primitive queries and Get queries, respectively. Let $\tau = (\tau_p, \tau_g)$ be the transcript, and we say τ is bad if

- exists $j_1, j_2 \in [Q_I]$ such that $(\mathsf{iv}_{j_1}, f_{j_1}) = (\mathsf{iv}_{j_2}, f_{j_2})$ and $k_{j_1} \oplus k_{j_2}$ appears in $y_{j_1} \oplus y_{j_2}$;
- exists $i \in [Q_P]$, $j \in [Q_I]$, $c \in [\lceil \ell/\lambda\rceil + 3]$ such that $s_i = \mathsf{iv}_j \oplus \langle a_j, c\rangle$ and $u_i = k_j$ or $u_i \oplus v_i$ is identical to the block of y_j corresponding to position c.

Let T_0 and T_1 be the distributions of τ when $b = 0$ and $b = 1$, and $\mathcal{T}_{\mathsf{Bad}}$ be the set of bad transcripts when $b = 1$. We have

$$\Pr[\mathsf{T}_1 \in \mathcal{T}_{\mathsf{Bad}}] \le \frac{Q_I(\lceil \ell/\lambda\rceil + 3)^2}{2^\lambda} + \frac{Q_I^2(\lceil \ell/\lambda\rceil + 3)^2}{2^{2\lambda}} + \frac{2Q_I \cdot Q_P}{2^{2\lambda}}$$

Also, for $\tau \notin \mathcal{T}_{\mathsf{Bad}}$, we have

$$\frac{\Pr[\mathsf{T}_0 = \tau]}{\Pr[\mathsf{T}_1 = \tau]} \ge 1$$

Then, by Lemma F.1, we get

$$\mathbf{Adv}^{\mathsf{prg}}_{\mathsf{F}^{\mathsf{dm}}}(Q_I, Q_P) \le \frac{Q_I(\lceil \ell/\lambda\rceil + 3)^2}{2^\lambda} + \frac{Q_I^2(\lceil \ell/\lambda\rceil + 3)^2}{2^{2\lambda}} + \frac{2Q_I \cdot Q_P}{2^{2\lambda}}$$

G Full Proof of the EUF-CMA Security

We provide the EUF-KO security proof of rVOLEitH. As we proved the extractable binding property of VoleComChk in Lemma 4.1, it remains to ensure that there is no unfortunate event on ZHash. The remaining part of the proof is semantically the same as the corresponding part in [4].

Lemma G.1 (EUF-KO Security of rVOLEitH). *Assume that H_1 and H_2^j for $j \in \{1,2,3\}$ are modeled as random oracles. Let $\mathcal{A}$ be an arbitrary adversary against the EUF-KO security of rVOLEitH that makes a total of Q random oracle queries and Q_P primitive queries. Then there exists a PPT adversary $\mathcal{B}$ against OWF security of* KeyGen *such that*

$$\mathbf{Adv}^{\text{euf-ko}}_{\text{rVOLEitH}}(\mathcal{A}) \le \frac{Q^2}{2^{2\lambda}} + \frac{QN^2(\tau^2u^4 + 8\tau u^2)\varepsilon_{\mathsf{v}}}{8} + Q\varepsilon_{\mathsf{zk}} + \frac{2Q}{2^\lambda} + \mathbf{Adv}^{\mathsf{owf}}_{\mathsf{KeyGen}}(\mathcal{B}) + \mathbf{Adv}^{\mathsf{coll2}}_{\mathsf{F}}(Q_P) + \mathbf{Adv}^{\mathsf{mcoll}}_{\mathsf{F}}(Q, Q_P, u) + \mathbf{Adv}^{\text{4-xor}}_{\mathsf{F}}(Q, Q_P).$$

Proof. Let $\mathcal{A}$ be an EUF-KO adversary against rVOLEitH for a given pk. Let $\mathcal{L}_P$, $\mathcal{L}_1$, $(\mathcal{L}_2^i)_{i=1,2,3}$, and $\mathcal{L}_3$ be the record of query-response pairs made by $\mathcal{A}$ to P, H_1, $(\mathsf{H}_2^i)_{i=1,2,3}$, and H_3, respectively. We argue by game hops. Let G_0 be the original EUF-KO game.

G_1: When $(\mathsf{chall}_1, \widetilde{\mathbf{u}}, h_V, \mathbf{d})$ is queried to H_2^2, the game runs the extractor Ext of VoleComChk in Lemma 4.1 to extract the VOLEs $(\mathbf{u}^*, \mathbf{V}^*)$ and remembers the result. If the adversary's forgery is valid, but was not extracted before, the game aborts. By Lemma 4.1, we have

$$\Pr[\mathsf{G}_1 \text{ aborts}] \le \frac{Q^2}{2^{2\lambda}} + \frac{QN^2(\tau^2u^4 + 8\tau u^2)\varepsilon_{\mathsf{v}}}{8} + \mathbf{Adv}^{\mathsf{coll2}}_{\mathsf{F}}(Q_P) + \mathbf{Adv}^{\mathsf{mcoll}}_{\mathsf{F}}(Q, Q_P, u) + \mathbf{Adv}^{\text{4-xor}}_{\mathsf{F}}(Q, Q_P)$$

Now $(\mathbf{u}^*, \mathbf{V}^*)$ is the unique VOLE pair which passes the consistency check because of the extractable binding property.

G_2: When $(\mathsf{chall}_1, \widetilde{\mathbf{u}}, h_V, \mathbf{d})$ is queried to H_2^2 by $\mathcal{A}$ and extraction succeeds, the game samples $\mathsf{chall}_2 \in \{0,1\}^{3\lambda+64}$ and aborts if zero-knowledge soundness check fails. More precisely, after setting $\mathbf{w} = \mathbf{d} \oplus \mathbf{u}^*[:\ell]$, if $\mathbf{w}$ does not satisfy the given multiplication constraints (from OWF) but ZHash corrects the error into a valid VOLE correlation, the game aborts. By the $\varepsilon_{\mathsf{zk}}$-universality of ZHash, we have

$$\Pr[\mathsf{G}_2 \text{ aborts}] \le \varepsilon_{\mathsf{zk}} Q_{2,2}.$$

Note that this game is identical to Game G_6 in [4, Theorem 1].

G_3: When $(\mathsf{chall}_2, \widetilde{a}, \widetilde{b})$ is queried to H_2^3 by $\mathcal{A}$, if QuickSilver check failed to detect the invalid $\mathbf{w}$ (computed in G_2) because Δ derived from chall_3 is a root of the quadratic polynomial in QuickSilver check, the game aborts. Since we consider only quadratic polynomials in QuickSilver check, we have

$$\Pr[\mathsf{G}_3 \text{ aborts}] \le \frac{2Q_{2,3}}{2^\lambda}.$$

Note that this game is identical to Game G_8 in [4, Theorem 1].

If $\mathcal{A}$ wins at G_3, $\mathcal{A}$ succeeds in finding the witness $\mathbf{w}$, which is consistent with the given multiplication constraints. Therefore,

there exists a PPT adversary $\mathcal{B}$ against OWF security such that

$$\begin{aligned}\mathbf{Adv}^{\text{euf-ko}}_{\text{rVOLEitH}}(\mathcal{A}) \le& \sum_{i=1}^{3} \Pr\left[\mathrm{G}_i \text{ aborts}\right] + \mathbf{Adv}^{\text{owf}}_{\text{KeyGen}}(\mathcal{B}) \\ \le& \frac{Q^2}{2^{2\lambda}} + \frac{QN^2(\tau^2u^4 + 8\tau u^2)\varepsilon_{\mathsf{v}}}{8} + Q\varepsilon_{\mathsf{zk}} \\ &+ \frac{2Q}{2^{\lambda}} + \mathbf{Adv}^{\text{owf}}_{\text{KeyGen}}(\mathcal{B}) + \mathbf{Adv}^{\text{coll2}}_{\mathsf{F}}(Q_P) \\ &+ \mathbf{Adv}^{\text{mcoll}}_{\mathsf{F}}(Q, Q_P, u) + \mathbf{Adv}^{\text{4-xor}}_{\mathsf{F}}(Q, Q_P)\end{aligned}$$

which concludes the proof. □

Now we provide the EUF-CMA security proof of rVOLEitH. Most of the games in the proof are a kind of routine random string replacement.

Theorem 4.3 (EUF-CMA Security of rVOLEitH). *Assume that* $\mathsf{H}_1 : \{0,1\}^* \to \{0,1\}^{2\lambda}$, $\mathsf{H}_2^1, \mathsf{H}_2^2$, *and* H_2^3 *are modeled as random oracles. Let* P *be an ideal primitive associated with* F*-functions. For a PPT adversary* $\mathcal{A}$ *against the EUF-CMA security of rVOLEitH with a total of* Q_{sig} *signing oracle queries,* Q *random oracle queries, and* Q_P *ideal primitive queries, there exist a positive integer* u*, PPT adversaries* $\mathcal{B}$ *and* C *against the OWF security of* KeyGen *and PRF security of* H_3,[5] *such that*

$$\begin{aligned}\mathbf{Adv}^{\text{euf-cma}}_{\text{rVOLEitH}}(\mathcal{A}) \le& \frac{(Q_{\text{sig}} + Q)^2}{2^{2\lambda}} + \frac{QN^2(\tau^2u^4 + 8\tau u^2)\varepsilon_{\mathsf{v}}}{8} + Q\varepsilon_{\mathsf{zk}} \\ &+ \frac{2(Q_{\text{sig}} + Q)}{2^{\lambda}} + \mathbf{Adv}^{\text{owf}}_{\text{KeyGen}}(\mathcal{B}) + Q_{\text{sig}}\mathbf{Adv}^{\text{prf}}_{\mathsf{H}_3}(C) \\ &+ \tau(k+2)\mathbf{Adv}^{\text{prg}}_{\mathsf{F}}(Q_{\text{sig}}, Q_P) + \mathbf{Adv}^{\text{coll2}}_{\mathsf{F}}(Q_P) \\ &+ \mathbf{Adv}^{\text{mcoll}}_{\mathsf{F}}(Q, Q_P, u) + \mathbf{Adv}^{\text{4-xor}}_{\mathsf{F}}(Q, Q_P).\end{aligned}$$

Proof. Let $\mathcal{A}$ be an EUF-CMA adversary against rVOLEitH for a given pk. Let G_0 be the original EUF-CMA game. Let $\mathcal{O}_{\text{sig}}$ be the signing oracle, let $Q_{\text{sig}}, Q_1, Q_{2,1}, Q_{2,2}$, and $Q_{2,3}$ be the number of queries made to $\mathcal{O}_{\text{sig}}, \mathsf{H}_1, \mathsf{H}_2^1, \mathsf{H}_2^2$, and H_2^3 by $\mathcal{A}$, respectively. We prove security for the deterministic version of rVOLEitH ($\rho \leftarrow 0^\lambda$), while the probabilistic version can be proved similarly. Without loss of generality, we assume that all messages in signing queries are distinct.

G_1: $\mathcal{O}_{\text{sig}}$ samples $(\mathsf{iv}, (r_i)_{i\in[\tau]}) \in \{0,1\}^{|\mathsf{iv}|} \times (\{0,1\}^\lambda)^\tau$ at random instead of computing $\mathsf{H}_3(\mathsf{sk}\|\mu\|\rho)$. As μ is distinct for each query, by a hybrid argument, the difference between this game and the previous one reduces to the PRF security of H_3 with secret key sk. Therefore, there exists a PPT adversary C against the PRF security of H_3 such that

$$|\Pr[\mathcal{A} \text{ wins } \mathrm{G}_0] - \Pr[\mathcal{A} \text{ wins } \mathrm{G}_1]| \le Q_{\text{sig}} \cdot \mathbf{Adv}^{\text{prf}}_{\mathsf{H}_3}(C)$$

G_2: $\mathcal{O}_{\text{sig}}$ samples $\mathsf{chall}_1 \in \{0,1\}^{5\lambda+64}$ uniformly at random instead of computing

$$\mathsf{H}_2^1(\mu \parallel h_{\text{com}} \parallel \mathbf{c}_1 \parallel \cdots \parallel \mathbf{c}_{\tau-1} \parallel \mathsf{iv}),$$

and also programs the random oracle H_2^1 to output chall_1 for the respective query. The simulation is aborted if the queries to H_2^1 have been made previously. This game is indistinguishable from the previous game unless the simulation is aborted, as $\mu \in \{0,1\}^{2\lambda}$ and $\mathsf{iv} \in \{0,1\}^{|\mathsf{iv}|}$ are chosen uniformly at random,

$$\Pr\left[\mathrm{G}_2 \text{ aborts}\right] \le Q_{\text{sig}} \cdot \left(\frac{Q_{\text{sig}} + Q_{2,1}}{2^{2\lambda+|\mathsf{iv}|}}\right).$$

G_3: $\mathcal{O}_{\text{sig}}$ samples $\mathsf{chall}_2 \in \{0,1\}^{3\lambda+64}$ uniformly at random instead of computing

$$\mathsf{H}_2^2(\mathsf{chall}_1 \parallel \widetilde{\mathbf{u}} \parallel h_V \parallel \mathbf{d}),$$

and also programs the random oracle H_2^2 to output chall_2 for the respective query. The simulation is aborted if the queries to H_2^2 have been made previously. This game is indistinguishable from the previous game unless the simulation is aborted, as $\mathsf{chall}_1 \in \{0,1\}^{5\lambda+64}$ is chosen uniformly at random, the probability of abort is

$$\Pr\left[\mathrm{G}_3 \text{ aborts}\right] \le Q_{\text{sig}} \cdot \left(\frac{Q_{\text{sig}} + Q_{2,2}}{2^{5\lambda+64}}\right).$$

G_4: $\mathcal{O}_{\text{sig}}$ samples $\mathsf{chall}_3 \in \{0,1\}^{\lambda}$ uniformly at random instead of computing

$$\mathsf{H}_2^3(\mathsf{chall}_2 \parallel \widetilde{a} \parallel \widetilde{b}),$$

and also programs the random oracle H_2^3 to output chall_3 for the respective query. The simulation is aborted if the queries to H_2^3 have been made previously. This game is indistinguishable from the previous game unless the simulation is aborted, as $\mathsf{chall}_2 \in \{0,1\}^{3\lambda+64}$ is chosen uniformly at random, the probability of abort is

$$\Pr\left[\mathrm{G}_4 \text{ aborts}\right] \le Q_{\text{sig}} \cdot \left(\frac{Q_{\text{sig}} + Q_{2,3}}{2^{3\lambda+64}}\right).$$

G_5: $\mathcal{O}_{\text{sig}}$ replaces the call to VoleCom by sampling $\mathbf{u}$ and $(\mathbf{c}_i)_{i\in[\tau]^*}$ uniformly at random from $\{0,1\}^{\ell+2\lambda+B}$ and computing $\mathbf{V}$ to preserve the VOLE correlations.

The difference between G_4 and G_5 corresponds to the multi-instance hiding game of VoleCom given Q_{sig} instances. Therefore, we have

$$|\Pr[\mathcal{A} \text{ wins } \mathrm{G}_4] - \Pr[\mathcal{A} \text{ wins } \mathrm{G}_5]| \le \tau(k+2)\mathbf{Adv}^{\text{prg}}_{\mathsf{F}}(Q_{\text{sig}}, Q_P)$$

After this game, the distribution of $\widetilde{\mathbf{u}}$ and $\widetilde{a}$ becomes uniformly random because of the hiding property of VHash and ZHash. The distribution of the commitment to the witness $\mathbf{d} = \mathbf{w} \oplus \mathbf{u}[:\ell]$ also becomes uniformly random since $\mathbf{u}[:\ell]$ is chosen uniformly at random.

In G_5, we see that the distribution of σ produced by $\mathcal{O}_{\text{sig}}$ no longer depends on the secret key sk and the success probability of $\mathcal{A}$ in this final game can be reduced to the EUF-KO security of rVOLEitH. Together with Lemma G.1, one can conclude the proof. □

[5] H_3 itself is not a PRF, but it is used as a PRF with key-prepending. We use this notation for convenience.

H VOLEitH-SBC with MinMax Commitments

Subfield Bilinear Collision (SBC) Problem. Let q be a prime power, and let k, n be positive integers. Given two vectors $\vec{u}, \vec{v} \in (\mathbb{F}_{q^k})^n$ that are linearly independent over $\mathbb{F}_{q^k}$, the goal of the SBC problem is to find two vectors $\vec{x}, \vec{y} \in (\mathbb{F}_q)^n$, linearly independent over $\mathbb{F}_q$, satisfying the relation:

$$(\vec{u} \cdot \vec{x})(\vec{v} \cdot \vec{y}) = (\vec{u} \cdot \vec{y})(\vec{v} \cdot \vec{x}). \tag{6}$$

We denote the instance of the SBC problem as $\mathrm{SBC}[\vec{u}, \vec{v}]$, and if $\vec{x}, \vec{y}$ satisfy the equation above, we write $(\vec{x}, \vec{y}) \in \mathrm{SBC}[\vec{u}, \vec{v}]$.

In the signature scheme based on the SBC problem, a normalized version is used. Instead of arbitrary vectors, solutions are represented in the form $\vec{x}' = (\vec{x}, 1, 0)$ and $\vec{y}' = (\vec{y}, 0, 1)$ for $\vec{x}, \vec{y} \in (\mathbb{F}_q)^{n-2}$. This results in the normalized SBC (NSBC) problem, where given $\vec{u}, \vec{v} \in (\mathbb{F}_{q^k})^n$, the goal is to find $\vec{x}, \vec{y} \in (\mathbb{F}_q)^{n-2}$ such that $\vec{x}'$ and $\vec{y}'$ satisfy the same bilinear relation. We write $(\vec{x}', \vec{y}') \in \mathrm{NSBC}[\vec{u}, \vec{v}]$ for such a solution.

The hardness of the SBC problem depends on the relation between the parameters k and n. [17] claimed that $k \approx 2n$ is a good choice generally, but they only give the parameter set $n = 130$, $k = 257$, and $q = 2$ for 128-bit security. Our attack will also focus on this set of parameters.

Key Generation. To generate an instance of the NSBC problem with a solution, we start by randomly choosing two vectors $\vec{x}, \vec{y} \in (\mathbb{F}_q)^{n-2}$ and setting $\vec{x}' = (\vec{x}, 1, 0)$ and $\vec{y}' = (\vec{y}, 0, 1)$. Then, we randomly select $\vec{u} = (u_i)_{i\in[n]} \in (\mathbb{F}_{q^k})^n$ and the first $n-1$ components $(v_i)_{i\in[n-1]}$ of $\vec{v} = (v_i)_{i\in[n]} \in (\mathbb{F}_{q^k})^n$. The final component v_{n-1} is computed to satisfy the NSBC relation:

$$v_{n-1} = \frac{(\vec{u}\cdot\vec{y}')\sum_{i\in[n-1]} v_i x'_i - (\vec{u}\cdot\vec{x}')\sum_{i\in[n-1]} v_i y'_i}{y'_{n-1}(\vec{u}\cdot\vec{x}')}. \tag{7}$$

If the denominator happens to be zero, the selection process is restarted. The secret key is $(\vec{x}, \vec{y})$, and the public key is $(\vec{u}, \vec{v})$.

Signing Algorithm. We will briefly describe how the signing algorithm works. Figure 7 provides pseudo-code for the signing algorithm, which is almost the same as the pseudo-code in the original paper.

(1) (Commitment) Compute cGGM trees and offsets. Each cGGM tree for the VOLE correlation – which we will call it a *main tree* – is not generated independently, but generated from a single pre-tree. The pre-tree is also a cGGM tree of 2^τ leaves. By the hypercube technique, pre-tree outputs τ pairs of roots whose sum is always $\vec{x}$. We remark that each main tree is not generated from a single root, but generated from two nodes. Therefore, $\mathsf{cGGM}(x_0, x_1, 2^D)$ outputs a cGGM tree of 2^D leaf nodes whose highest two nodes are x_0 and x_1. The Folding function for the hypercube technique [1] is defined as follows.

$$\mathsf{Folding}\left(\vec{x}^{[\![0]\!]}, \ldots, \vec{x}^{[\![2^\tau-1]\!]}\right) \coloneqq \left(\sum_{B_j(i)=0} \vec{x}^{[\![i]\!]}, \sum_{B_j(i)=1} \vec{x}^{[\![i]\!]}\right)_{j\in[\tau]}$$

where $B_j(i)$ is the j-th bit of i.
For the generation of main trees, VOLE shares are generated by similar operations. Finally, the offsets $\delta_{\vec{y}_j}, \delta_{z_j}$ are computed. $(\vec{x}_j^{[\![0]\!]}, \ldots, \vec{x}_j^{[\![2^D-1]\!]})$ in Figure 7 play the role of individual seeds in the original VOLEitH, and there are no leaf commitments to those seeds. Instead, MinMax commitment replaces leaf commitment in a batched manner. The MinMax commitment is computed from the leaf commitments as follows.

$$\mathrm{com}_j^{[\![i]\!]} = \mathcal{H}(\vec{x}_j^{[\![i]\!]})$$
$$M^{\min} \leftarrow \min \bigcup_{j\in[\tau]} \{\mathrm{com}_j^{[\![0]\!]}, \ldots, \mathrm{com}_j^{[\![2^D-1]\!]}\}$$
$$M^{\max} \leftarrow \max \bigcup_{j\in[\tau]} \{\mathrm{com}_j^{[\![0]\!]}, \ldots, \mathrm{com}_j^{[\![2^D-1]\!]}\}$$

(2) (First Challenge) Hash the transcript and generate the challenges. The challenge includes $\vec{\mu} \in \mathbb{F}_{2^k}^{n-2}$ and $(\alpha_m)_{m\in[\tau D]} \in \mathbb{F}_{2^k}^{\tau D}$.

(3) (First Response) Compute the response values $(A, B, (a^{(j)})_{j\in[\tau]}, b)$. A and B are computed as follows. Denoting $\vec{A}^{10} = (\vec{A}, 1, 0)$ and $\vec{A}^{01} = (\vec{A}, 0, 1)$,

$$\begin{aligned} A \leftarrow{}& \left(\vec{u}\cdot\vec{A}^{10}_{F_x}\right)\left(\vec{v}\cdot\vec{A}^{01}_{F_y}\right) - \left(\vec{u}\cdot\vec{A}^{01}_{F_y}\right)\left(\vec{v}\cdot\vec{A}^{10}_{F_x}\right) \\ B \leftarrow{}& \left(\vec{v}\cdot\vec{A}^{01}_{F_y}\right)\left(\vec{u}\cdot\vec{x}^{10}\right) + \left(\vec{u}\cdot\vec{A}^{10}_{F_x}\right)\left(\vec{v}\cdot\vec{y}^{01}\right) \\ & - \left(\vec{u}\cdot\vec{A}^{01}_{F_y}\right)\left(\vec{v}\cdot\vec{x}^{10}\right) - \left(\vec{v}\cdot\vec{A}^{10}_{F_x}\right)\left(\vec{u}\cdot\vec{y}^{01}\right) \end{aligned} \tag{8}$$

where $\vec{A}_{F_x}$ and $\vec{A}_{F_y}$ are some linear combinations of $(\alpha_i)_{i\in[\tau D]}$ such that

$$\vec{A}_{F_x} = \sum_{j=0}^{\tau-1}\sum_{i=jD}^{(j+1)D-1} \vec{x}_j^{[\![0]\!]_i}\alpha_i, \qquad \vec{A}_{F_y} = \sum_{j=0}^{\tau-1}\sum_{i=jD}^{(j+1)D-1} \vec{y}_j^{[\![0]\!]_i}\alpha_i.$$

(4) (Second Challenge) Hash the transcript and generate the second challenge $(i^*_j)_{j\in[\tau]}$. Extract pre-challenge i^*_p for pre-tree from the second challenge.

(5) (Second Response) Open the puncturable pseudorandom function (PPRF) key. As the highest nodes of the main trees are generated from the pre-tree, the opened PPRF key should contain the PPRF key of both pre-tree and main trees.

Verification Algorithm. Before introducing the verification algorithm, we remark some notations which help in understanding the attack in the next section. We — and also the original paper [18] — use notation ($'$) to distinguish the same variables in signing and verification algorithms. For example, $a^{(j)}$ and $a^{(j)'}$, A and A' should be the same if the signature was valid. Otherwise, since they are computed in different ways, they may differ from each other.

We will also briefly describe how the verification algorithm works. Figure 8 is a pseudo-code for the verification algorithm, which is exactly the same as the pseudo-code in the original paper.

(1) (Reconstruct Trees) Parse the signature, reconstruct all the trees in a punctured state, and fold all the trees using the hypercube technique. Also reconstruct the MinMax commitments.

(2) (First Challenge) Generate the first challenge as in the signing algorithm.

(3) (Recompute First Response) Compute $(a^{(j)'})_{j\in[\tau]}$ and A' which should be the same as $(a^{(j)})_{j\in[\tau]}$ and A respectively for honest signature.

(4) (Compare Second Challenge) Hash the recomputed transcript and compare the hash output.

- **Inputs:** Secret key $\mathsf{sk} = (\vec{x}, \vec{y})$, public key $\mathsf{pk} = (\vec{u}, \vec{v})$ and a message msg.

(1) Sample $R_{\vec{x}} \leftarrow (\mathbb{F}_2)^{n-2}$, and salt $\leftarrow \{0,1\}^{2\lambda}$ for the cGGM tree derivation.

(2) Compute pre-tree: $\left(\vec{x}^{[\![0]\!]}, \dots, \vec{x}^{[\![2^\tau-1]\!]}\right) \leftarrow \mathsf{cGGM}_{\mathsf{salt}}(R_{\vec{x}}, R_{\vec{x}} \oplus \vec{x}, 2^\tau)$

(3) Fold pre-tree: $\left(\vec{x}^{[\![0]\!]_j}, \vec{x}^{[\![1]\!]_j}\right)_{j\in[\tau]} \leftarrow \mathsf{Folding}\left(\vec{x}^{[\![0]\!]}, \dots, \vec{x}^{[\![2^\tau-1]\!]}\right)$

(4) For $j \in [\tau]$:

(a) Get Tree T_j: $\left(\vec{x}_j^{[\![0]\!]}, \dots, \vec{x}_j^{[\![2^D-1]\!]}\right) \leftarrow \mathsf{cGGM}_{\mathsf{salt}}\left(\vec{x}^{[\![0]\!]_j}, \vec{x}^{[\![1]\!]_j}, 2^D\right)$

(b) Expand T_j and get $\left(\vec{y}_j^{[\![0]\!]}, \dots, \vec{y}_j^{[\![2^D-1]\!]}\right)$, $\left(z_j^{[\![0]\!]}, \dots, z_j^{[\![2^D-1]\!]}\right)$, and $\left(\mathrm{com}_j^{[\![0]\!]}, \dots, \mathrm{com}_j^{[\![2^D-1]\!]}\right)$

(c) Fold Tree T_j:

$\left(\vec{x}_j^{[\![0]\!]_i}, \vec{x}_j^{[\![1]\!]_i}\right)_{i\in[D]} \leftarrow \mathsf{Folding}\left(\vec{x}_j^{[\![0]\!]}, \dots, \vec{x}_j^{[\![2^D-1]\!]}\right)$

$\left(\vec{y}_j^{[\![0]\!]_i}, \vec{y}_j^{[\![1]\!]_i}\right)_{i\in[D]} \leftarrow \mathsf{Folding}\left(\vec{y}_j^{[\![0]\!]}, \dots, \vec{y}_j^{[\![2^D-1]\!]}\right)$

$\left(z_j^{[\![0]\!]_i}, z_j^{[\![1]\!]_i}\right)_{i\in[D]} \leftarrow \mathsf{Folding}\left(z_j^{[\![0]\!]}, \dots, z_j^{[\![2^D-1]\!]}\right)$

(d) Let $z \leftarrow z_0^{[\![0]\!]_0} \oplus z_0^{[\![1]\!]_0}$

(e) Compute offsets $\delta_{\vec{y}_j} \leftarrow \vec{y} \oplus \vec{y}_j^{[\![0]\!]_0} \oplus \vec{y}_j^{[\![1]\!]_0}$ and $\delta_{z_j} \leftarrow z \oplus z_j^{[\![0]\!]_0} \oplus z_j^{[\![1]\!]_0}$

(5) $M^{\min} \leftarrow \min \bigcup_{j\in[\tau]} \{\mathrm{com}_j^{[\![0]\!]}, \dots, \mathrm{com}_j^{[\![2^D-1]\!]}\}$

(6) $M^{\max} \leftarrow \max \bigcup_{j\in[\tau]} \{\mathrm{com}_j^{[\![0]\!]}, \dots, \mathrm{com}_j^{[\![2^D-1]\!]}\}$

(7) If $M^{\min}$ or $M^{\max}$ are not unique: ABORT

(8) $h_{\mathrm{com}} \leftarrow \mathcal{H}_3\left(\mathsf{salt}, M^{\min}, M^{\max}, \left(\delta_{\vec{y}_j}\right)_{j\in[\tau]}, \left(\delta_{z_j}\right)_{j\in[\tau]^*}\right)$

(9) $h_0 \leftarrow \mathcal{H}_0(\mathrm{msg}, h_{\mathrm{com}}, \mathsf{salt}, (\delta_{\vec{y}_j}, \delta_{z_j})_{j\in[\tau]})$

(10) $(\vec{\mu}, (\alpha_m)_{m\in[\tau\cdot D]}) \leftarrow \mathrm{PRG}_0(h_0)$

(11) Compute $\vec{A}_{F_x} \leftarrow \sum_{j=0}^{\tau-1} \sum_{i=jD}^{(j+1)D-1} \vec{x}_j^{[\![0]\!]_i} \alpha_i$

(12) Compute $\vec{A}_{F_y} \leftarrow \sum_{j=0}^{\tau-1} \sum_{i=jD}^{(j+1)D-1} \vec{y}_j^{[\![0]\!]_i} \alpha_i$

(13) Compute coefficients A and B as of Φ in Eq. 8

(14) Compute $b \leftarrow z + \vec{y} \cdot \vec{\mu}$

(15) For $j \in [\tau]$:

(a) Compute $\vec{A}_{F_y}^{(j)} \leftarrow \sum_{i=jD}^{(j+1)D-1} \vec{y}_j^{[\![0]\!]_i} \alpha_i$ and $\vec{A}_{F_z}^{(j)} \leftarrow \sum_{i=jD}^{(j+1)D-1} z_j^{[\![0]\!]_i} \alpha_i$

(b) Compute $a^{(j)} \leftarrow \vec{A}_{F_z}^{(j)} + \vec{A}_{F_y}^{(j)} \cdot \vec{\mu}$

(16) $h_1 \leftarrow \mathcal{H}_1(\mathrm{msg}, h_{\mathrm{com}}, \mathsf{salt}, (\delta_{\vec{y}_j}, \delta_{z_j})_{j\in[\tau]}, h_0, A, B, (a^{(j)})_{j\in[\tau]}, b)$

(17) $(i_j^*)_{j\in[\tau]} \leftarrow \mathrm{PRG}_1(h_1)$

(18) If there is an index $j \in [\tau]$ with $i_j^* = i^{\min}$ or $i_j^* = i^{\max}$: ABORT

(19) Compute the pre-challenge $i_p^* \leftarrow \sum_{j=0}^{\tau-1} \lfloor i_j^* / 2^{D-1} \rfloor \cdot 2^j$

(20) Compute the pre-tree PPRF key $K_{i_p^*}$ and the PPRF key $K_{i_0^*,\dots,i_{\tau-1}^*}$

(21) Output signature $\sigma \leftarrow (h_1, \mathsf{salt}, (\delta_{\vec{y}_j})_{j\in[\tau]}, (\delta_{z_j})_{j\in[\tau]^*}, B, b, K_{i_p^*}, K_{i_0^*,\dots,i_{\tau-1}^*})$

Figure 7: $\mathrm{SBC}^M_{\mathrm{VOLE}}$ **signature scheme - signing algorithm**

- **Inputs:** Public key $\mathsf{pk} = (\vec{u}, \vec{v})$, a message $\mathsf{msg} \in \{0,1\}^*$, and a signature $\sigma = (h_1, \mathsf{salt}, (\delta_{\vec{y}_j})_{j\in[\tau]}, (\delta_{z_j})_{j\in[\tau]^*}, B, b, K_{i_p^*}, K_{i_0^*,\ldots,i_{\tau-1}^*})$.

(1) $(i_j^*)_{j\in[\tau]} \leftarrow \mathrm{PRG}_1(h_1)$ where h_1 is parsed from σ

(2) Recompute and fold all trees (with one missing leaf each) using $K_{i_p^*}$, $K_{i_0^*,\ldots,i_{\tau-1}^*}$, and salt

(3) $M^{\min'} \leftarrow \min \bigcup_{j\in[\tau]} \left(\bigcup_{i\in[2^D]\setminus\{i_j^*\}} \left\{ \mathrm{com}_j^{[\![i]\!]} \right\} \right)$

(4) $M^{\max'} \leftarrow \max \bigcup_{j\in[\tau]} \left(\bigcup_{i\in[2^D]\setminus\{i_j^*\}} \left\{ \mathrm{com}_j^{[\![i]\!]} \right\} \right)$

(5) For $(i,j) \in [D] \times [\tau]$, let $b_{i,j}$ denote the known position of the respective hypercube binary sharings

(6) $h'_{\mathrm{com}} \leftarrow \mathcal{H}_3 \left(\mathsf{salt}, M^{\min'}, M^{\max'}, \left(\delta_{\vec{y}_j}\right)_{j\in[\tau]}, \left(\delta_{z_j}\right)_{j\in[\tau]^*} \right)$

(7) $\delta_{z_0} \leftarrow 0$

(8) $h'_0 \leftarrow \mathcal{H}_0(\mathsf{msg}, h'_{\mathrm{com}}, \mathsf{salt}, (\delta_{\vec{y}_j}, \delta_{z_j})_{j\in[\tau]})$

(9) $(\vec{\mu}, (\alpha_m)_{m\in[\tau\cdot D]}) \leftarrow \mathrm{PRG}_0(h'_0)$

(10) For $j \in [\tau]$:

- Compute $\Delta_j^* \leftarrow \sum_{i=jD}^{(j+1)D-1} b_{i,j}\alpha_i$
- Compute $\vec{F}_y^{(j)}(\Delta_j^*) \leftarrow \sum_{i=jD}^{(j+1)D-1} \alpha_i(\vec{y}_j^{[\![b_{i,j}]\!]_i} + b_{i,j}\delta_{\vec{y}_j})$
- Compute $Z^{(j)}(\Delta_j^*) \leftarrow \sum_{i=jD}^{(j+1)D-1} \alpha_i(z_j^{[\![b_{i,j}]\!]_i} + b_{i,j}\delta_{z_j})$
- $a^{(j)'} \leftarrow Z^{(j)}(\Delta_j^*) + \vec{F}_y^{(j)}(\Delta_j^*) \cdot \vec{\mu} + b\Delta_j^*$

(11) $\Delta^* \leftarrow \sum_{j=0}^{\tau-1} \Delta_j^*$

(12) $\vec{F}_x(\Delta^*) \leftarrow \sum_{m=0}^{\tau\cdot D-1} \vec{x}_m^{[\![b_m]\!]}\alpha_m$

(13) $\vec{F}_y(\Delta^*) \leftarrow \sum_{j=0}^{\tau-1} \vec{F}_y^{(j)}(\Delta_j^*)$

(14) Compute $\Phi(\Delta^*)$ as

$$\left(\vec{u} \cdot \begin{pmatrix} \vec{F}_x(\Delta^*) \\ 1 \\ 0 \end{pmatrix}\right)\left(\vec{v} \cdot \begin{pmatrix} \vec{F}_y(\Delta^*) \\ 0 \\ 1 \end{pmatrix}\right) - \left(\vec{u} \cdot \begin{pmatrix} \vec{F}_y(\Delta^*) \\ 0 \\ 1 \end{pmatrix}\right)\left(\vec{v} \cdot \begin{pmatrix} \vec{F}_x(\Delta^*) \\ 1 \\ 0 \end{pmatrix}\right)$$

(15) $A' \leftarrow \Phi(\Delta^*) - B\Delta^*$

(16) $h'_1 \leftarrow \mathcal{H}_1(\mathsf{msg}, h'_{\mathrm{com}}, \mathsf{salt}, (\delta_{\vec{y}_j}, \delta_{z_j})_{j\in[\tau]}, h'_0, A', B, (a^{(j)'})_{j\in[\tau]}, b)$

(17) If $h'_1 = h_1$ output `ACCEPT`, otherwise output `REJECT`

Figure 8: $\mathrm{SBC}^{M}_{\mathrm{VOLE}}$ **signature scheme - verification algorithm**