SM4th and uBlockith: VOLE-based Post-Quantum Signature Schemes from Chinese Block Ciphers

Weihan Li[1], Yuchen Wang[1], Zhelei Zhou[1], Cheng Hong[1⋆], and Tao Wei[1]

Ant Group, Beijing, 100081, China
vince.hc@antgroup.com

Abstract. FAEST is a family of post-quantum signature schemes based on VOLE-in-the-Head, and is one of the nine candidates advanced to the third round of the NIST Additional Digital Signature process. FAEST relies only on symmetric cryptographic primitives, including block ciphers and hash functions, and does not require structured number-theoretic assumptions.

We propose two families of signature schemes, SM4th and uBlockith, targeting 128-bit and 256-bit classical security, respectively. SM4th and uBlockith follow the FAEST framework but instantiate it with Chinese-designed block ciphers, including SM4, uBlock, and Ballet. We further design constraint systems tailored to these block ciphers and provide instruction-set-aware optimized implementations.

Our evaluation on two Intel platforms and a Hygon platform shows that the end-to-end performance of the proposed schemes is strongly platform dependent. On an Intel platform with native SM4 support, the SM4th variants achieve performance comparable to the corresponding FAEST-128 variants, with a gap of less than 1×. On the Hygon platform with native CIS-SM4 support, the SM4th variants are within approximately 3× of FAEST-128. The SM4th-EM-s variant has a combined public-key and signature size of 3 850 bytes, compared with 3 938 bytes for FAEST-EM-128s. The uBlockith variants remain approximately 3–4× slower than FAEST-256. These results demonstrate the feasibility and costs of instantiating VOLE-based signatures with the selected Chinese block ciphers.

Keywords: Post-Quantum Signature · VOLE-in-the-Head · FAEST · SM4 · uBlock

1 Introduction

Post-quantum digital signature schemes have become a central topic in the global transition toward quantum-resistant cryptography. After announcing that three signature schemes CRYSTALS-Dilithium [6], Falcon [24], SPHINCS$^+$ [5] were selected for standardization in July 2022, the National Institute of Standards and

⋆ Corresponding author

Technology (NIST) of the United States initiated an additional signature standardization process two months later [35], with a particular interest in schemes based on non-lattice assumptions. Forty candidate schemes entered the first round. On May 14, 2026, NIST announced that nine candidates had advanced to the third round, including FAEST [9]. One main advantage of FAEST is its good trade-off between security and performance. On the one hand, FAEST is the only scheme of the 9 candidates which relies solely on symmetric primitives (block ciphers and hash functions) without any structured assumptions. This is similar to the standardized SPHINCS$^+$ [5] (*i.e.*, SLH-DSA [34]) built from only hash functions. On the other hand, FAEST achieves good general performance. For example, FAEST-128 features 100× faster signing times, 6× faster verifying times, and 3× smaller signature sizes than SLH-DSA-128 according to our benchmarks. The smallest combined size of public key and signature is 3 938 bytes among FAEST-128 variants, which is very close to the 3 732 bytes of the standardized CRYSTALS-Dilithium [6] (*i.e.*, ML-DSA [33]).

Besides the United States, several other countries have also investigated post-quantum signature schemes through national standardization activities or public competitions. In South Korea, the KpqC (Korean Post-Quantum Cryptography) competition [30] included a dedicated digital-signature-scheme track. In China, the Institute of Commercial Cryptography Standards (ICCS) has launched the Next-Generation Commercial Cryptographic Algorithms (NGGC) program [27].

FAEST demonstrates that VOLE-in-the-Head signatures can be built from symmetric primitives, but it uses AES and Rijndael. For deployments that require specified domestic primitives, replacing these components is not a black-box substitution: the PRG dominates a substantial part of the running time, while the algebraic representation of the block cipher directly determines the witness length, the degree of the constraints, and hence the signature size.

This work studies two such instantiations, SM4th and uBlockith, based on the Chinese national standard SM4 and the Chinese-designed block ciphers uBlock and Ballet. In the absence of a national standard for 256-bit block ciphers, we select uBlock and Ballet, as they are the only two first-prize winners in the block-cipher category of the Chinese Cryptographic Algorithm Design Competition[1] and their security has received growing attention in recent years [37,13]. We present concrete VOLEitH instantiations, including corresponding one-way functions (OWFs), pseudo-random generators (PRGs), constraint systems, and parameter sets, and evaluate the resulting signatures on x86-64 processors.

Contributions. The contributions of this paper are as follows.

- We construct the signature family SM4th targeting the 128-bit classical security, with a degree-3 non-EM proof system and a degree-2 Even–Mansour (EM) proof system, where SM4 serves as both the OWF and PRG. The latter uses the EM construction of SM4 as OWF and commits to a shorter witness, yielding smaller signature sizes.
- We construct the 256-bit family uBlockith from uBlock and Ballet. Specifically, the ZKP-friendly uBlock serves as the OWF and the software-friendly

[1] https://sfjs.cacrnet.org.cn/site/content/404.html

Ballet serves as the PRG. To represent the cubic Boolean S-box constraints in uBlock, we use a meet-in-the-middle approach so that only every *second* encryption-round boundary is committed.
- We give instruction-set-architecture-aware (ISA) implementations of the required CTR-style PRGs: SM4-NI/AES-NI based SM4 emulation on Intel processors, native SM4 instructions on Hygon and newer Intel processors, and four-key batching for Ballet.
- We provide concrete parameter sets and a comprehensive experimental comparison with FAEST across two Intel platforms and a Hygon platform. On the Intel platform with emulated SM4 backends, SM4th requires approximately 10× the runtime of the corresponding FAEST-128 variants. On the Intel platform with native SM4 support, SM4th achieves performance comparable to FAEST-128, within a factor of 1.8. On the Hygon platform with native CIS-SM4 support, SM4th requires approximately 2–3× the runtime of FAEST-128. Moreover, the combined public-key and signature sizes of SM4th range from 19% larger to 3% smaller than the corresponding FAEST-128 profiles. For uBlockith, the runtime is approximately 3–4× that of FAEST-256.

In summary, our results suggest that Chinese-designed block ciphers can serve as efficient building blocks for VOLE-based post-quantum signatures. A version of our schemes was submitted to the NGCC program.

Related works and comparisons. Our signature schemes are similar to other post-quantum signature schemes via the "in-the-Head" techniques. The original work is PICNIC [19], which is constructed from MPC-in-the-Head (MPCith) [28] and using LowMC as OWF [4]. The MPCith framework is later improved by threshold-computation-in-the-head (TCitH) [22]. Subsequent works combine the MPCith/TCitH framework with various OWFs to build signatures with different trade-offs, such as Banquet [12] from AES, Mirath [3] from the Min-Rank problem, MQOM [14] from the multivariate quadratic problem, PERK [15] from the permuted kernel problem, and RYDE [17] or SDitH-v1 [32] from syndrome decoding. Recently, VOLE-in-the-Head (VOLEitH) [11] is proposed, which can lead to smaller signatures sizes due to the more efficient repetition method to achieve target soundness, with schemes including FAEST [9] from AES, and SDitH-v2 [32] and ReSolveD [20] from syndrome decoding.

There also exist other post-quantum signatures based on different assumptions and techniques, such as lattice-based ML-DSA, Falcon, and HAWK [18], code-based CROSS [7] and LESS [8], multivariate-based MAYO [16], isogeny-based SQIsign [2], and solely hash-based SLH-DSA. Our signatures achieve different trade-offs in security and performance. Specifically, unlike most of the above schemes (except FAEST and SLH-DSA), our signature schemes do not rely on any other number-theoretic or structured hardness assumptions. Our schemes also have smaller signature sizes than the standardized SLH-DSA with faster signing and verifying times. These advantages, however, come at the cost of efficiency as suggested in FAEST and SLH-DSA: the resulting performance is expected to be less competitive than that of schemes based on these hardness assumptions such as lattice.

Outline of this paper. This paper is structured as follows. Section 2 introduces the high-level construction of VOLE-based signature schemes and FAEST. Section 3 gives a technical overview of our schemes, particularly focusing on the differences against FAEST. Section 4 and Section 5 present the constructions of SM4th and uBlockith, respectively. Section 6 gives the implementation details and reports the evaluation results. Finally, Section 7 concludes the paper.

2 Preliminaries

This section introduces the necessary background to describe the constraint systems and the resulting signatures. We use the VOLEitH framework and the QuickSilver [39] protocol as building blocks. This paper focuses on the latter part, and defers the concrete VOLE-correlation generation, VOLE parameters, and signature construction to Appendix A and Appendix B. The formal cryptographic assumptions used by the security theorems are stated in Assumption 1 in Appendix C.1.

Notation and relations. For integers $0 \le a \le b$, let $[a, b) = \{a, a+1, \ldots, b-1\}$. All numeric indices in this paper start from zero. For a word sequence $\boldsymbol{z}$, $\boldsymbol{z}[i]$ denotes its i-th word. If $\boldsymbol{z}[i]$ is a 32-bit SM4 word, then $\boldsymbol{z}[i][j]$ denotes its j-th byte, where $j \in [0, 4)$. Let $\mathbb{F}_{2^\lambda} = \mathbb{F}_2[Z]/(p_\lambda(Z))$ be the proof field.

Let $E : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}^\lambda$ denote the block cipher used by a parameter set. The non-EM and EM relations considered in this paper are $\mathcal{R}_{\mathsf{non\text{-}EM}} = \{((\boldsymbol{x}, \boldsymbol{y}), \boldsymbol{k}) : y = E_{\boldsymbol{k}}(\boldsymbol{x})\}$, and $\mathcal{R}_{\mathsf{EM}} = \{((\boldsymbol{x}, \boldsymbol{y}), \boldsymbol{k}) : y = E_{\boldsymbol{x}}(\boldsymbol{k}) \oplus \boldsymbol{k}\}$. For both relations, the public key is $\mathsf{pk} = (\boldsymbol{x}, \boldsymbol{y})$ and the secret key is $\mathsf{sk} = (\boldsymbol{x}, \boldsymbol{k})$, which contains the corresponding witness $\boldsymbol{k}$.

Constraint systems. For a public key pk, the signer derives an extended witness $\boldsymbol{w} \in \{0,1\}^\ell$. The witness contains the secret input together with the intermediate values required by the selected constraint representation. A parameter set specifies C functions $f_j : \{0,1\}^\ell \longrightarrow \mathbb{F}_{2^\lambda}$ for $j \in [0, C)$, each of degree at most d. The witness is valid if $f_j(\boldsymbol{w}) = 0$ for $j \in [0, C)$. The concrete witness layouts and constraint derivations for SM4th and uBlockith are given in Sections 4 and 5, respectively. We denote the common deterministic derivation interface by

$$\langle \boldsymbol{o} \rangle \leftarrow \mathsf{DeriveConstraints}_{\mathsf{param}}(\langle \boldsymbol{w} \rangle, \mathsf{pk}),$$

where $\langle \boldsymbol{w} \rangle$ is a degree-1 commitment to the extended witness and $\langle \boldsymbol{o} \rangle$ is the resulting length-C, degree-d constraint vector. The signer and verifier apply the same public derivation to their respective VOLE views. The procedures ProveConstraints and VerifyConstraints in Appendix B invoke this interface before applying ZKHash to batch the entries of $\langle \boldsymbol{o} \rangle$.

VOLEitH interface. For a witness $\boldsymbol{w} \in \{0,1\}^\ell$, the VOLEitH layer produces vectors $\boldsymbol{u} \in \mathbb{F}_2^{\hat{\ell}}, \boldsymbol{v}, \boldsymbol{q} \in \mathbb{F}_{2^\lambda}^{\hat{\ell}}$, and a global correlation value $\Delta \in \mathbb{F}_{2^\lambda}$ satisfying $\boldsymbol{q}[i] = \boldsymbol{u}[i]\Delta + \boldsymbol{v}[i]$ for every $i \in [0, \hat{\ell})$. The signer masks the witness with $d = \boldsymbol{w} + \boldsymbol{u}[0 \,..\, \ell - 1]$, which allows the verifier to obtain commitments to the witness without learning $\boldsymbol{w}$. We write $\langle \boldsymbol{w} \rangle^e$ for a degree-e VOLE commitment to a vector.

A degree-e commitment $\langle a \rangle^e$ to $a \in \{0,1\}$ can be represented as the prover $\mathcal{P}$ holding a polynomial $\rho_a(X) = a_0 + a_1 X + \cdots + a_{e-1} X^{e-1} + a X^e$, whereas the verifier $\mathcal{V}$ holds only $\gamma_a = \rho_a(\Delta)$. In what follows, an operation on angle-bracket notation denotes the following paired local computations:

$$\langle a \rangle^{e_1} + \langle b \rangle^{e_2} \longmapsto \langle z \rangle^{e=\max(e_1,e_2)} : \quad \begin{cases} \rho_z(X) = X^{e-e_1}\rho_a(X) + X^{e-e_2}\rho_b(X), & \mathcal{P}, \\ \gamma_z = \Delta^{e-e_1}\gamma_a + \Delta^{e-e_2}\gamma_b, & \mathcal{V}; \end{cases}$$

$$\langle a \rangle^{e} + c \longmapsto \langle z \rangle^{e} : \quad \begin{cases} \rho_z(X) = \rho_a(X) + cX^e, & \mathcal{P}, \\ \gamma_z = \gamma_a + c\Delta^e, & \mathcal{V}; \end{cases}$$

$$\langle a \rangle^{e_1} \cdot \langle b \rangle^{e_2} \longmapsto \langle z \rangle^{e_1+e_2} : \quad \begin{cases} \rho_z(X) = \rho_a(X)\rho_b(X), & \mathcal{P}, \\ \gamma_z = \gamma_a\gamma_b, & \mathcal{V}. \end{cases}$$

Multiplication by a public constant and every public linear map are applied component-wise under the same convention. Thus, both parties evaluate the same angle-bracket expressions without communication, and only a multiplication between commitments raises the degree.

The VOLE commitments are generated via the batch all-but-one vector commitment (BAVC) of Baum *et al.* [10]. It involves three interfaces: BAVC.Commit, BAVC.Open, and BAVC.Reconstruct. Specifically, the signer first produces a commitment to the extended witness via BAVC.Commit, derives Fiat–Shamir challenges, and then returns a compressed proof together with a BAVC opening using BAVC.Open. The verifier reconstructs the challenge-dependent commitment view, checks the BAVC opening via BAVC.Reconstruct, and verifies the compressed constraint equation. Their input and output formats, as well as the parameters $(\tau_0, \tau_1, k, L, T_{\mathsf{open}})$, are specified in Appendix A.

Fiat–Shamir transformation. The interactive proof is made non-interactive by applying the Fiat–Shamir transformation [23]. The message is bound to the proof through the first-round transcript to generate the first challenge by hashes. Subsequent transcript hashes derive the commitment challenge, the constraint challenge, and the BAVC opening challenge.

For the completeness, we give high-level signing and verification algorithms in Appendix B.

3 Overview of SM4th and uBlockith

This section introduces the technical overview of our schemes, which follow the VOLEitH framework of FAEST, but differ from it in several important aspects.

Using Chinese-designed OWFs and PRGs, with new ISA-aware PRG implementations. We replace the OWF and PRG instantiations. For SM4th, SM4 is used as the OWF and PRG. For uBlockith, uBlock is used as the OWF, while Ballet is used as a software-friendly PRG.

We design new ISA-aware optimizations for different PRGs. A major part of the signing and verifying time is spent in PRGs, making ISAs crucial. We

build platform-and-scheme-specific optimized implementations inspired by the following two observations. First, different from FAEST, we can not use the AES-specific ISA AES-NI available on most Intel or AMD CPUs due to using Ballet, and especially SM4. However, due to the similar construction of AES and SM4 S-boxes, it is feasible to simulate S-boxes of SM4 using AES-NI [36]. In addition, it is possible to leverage SM4-specific ISA available on certain processors such as Hygon and newer Intel platforms.

Second, PRGs in our schemes are invoked in several specific modes, which inspires us to make scheme-specific optimizations. For example, on Intel CPUs without native SM4 support, we design a 4-block key-expansion-routine backend and a 4-block different-key encryption-routine backend for SM4 based on the open-sourced 4-block same-key encryption-routine backend based on SM4-NI, which is helpful for the VOLE binary-tree-style commitment. We also design two output regimes of the SM4-CTR-style PRGs, 276 bytes and 162 bytes, as motivated by the scheme parameters. In addition, observing that many PRGs used in the signatures share a common plaintext, we also propose a fixed-plaintext batched PRG core, which may be useful for other VOLEitH-based protocols.

Designing ZKP constraint systems for OWFs. We design zero-knowledge constraint systems for SM4 and uBlock. For the former, we exploit the patched inversion representation of the S-box and give two proof strategies for optimized transmission costs based on the OWF construction. As for the non-EM variants, we modify the norm technique in FAEST to make it compatible with SM4, which has an iterative structure instead of the sequential one in AES. For the EM variants, we directly commit the outputs of all S-boxes, which lead to a smaller signature size. For uBlock using S-boxes based on boolean circuits, we use a "meet-in-the-middle" approach to avoid trivially committing all outputs of S-boxes. Our techniques may be helpful if using other block ciphers as OWFs for VOLE-based ZKPs in the future, as many block ciphers use S-boxes belonging to these two categories.

Using PRG-based leaf commitments. After generating the GGM tree, the prover needs to commit each leaf. We adopt the PRG-based leaf commitments introduced in FAEST-EM and use them in both our EM and non-EM variants. This reduces the cost of leaf commitments and improves performance in both communication and running times, but requiring an assumption that the PRG output is sufficiently binding as a commitment to its input seed. In the ideal cipher model, this assumption is expected to hold [9], but it should still be stated separately from standard PRG assumptions. If a more strict security requirement is desired, it is feasible to use the universal-hash-based leaf commitments in the non-EM variants of FAEST, which leads to a slightly larger signature size.

Overall, SM4th and uBlockith keep the modular VOLEitH architecture of FAEST, but replace and retune the concrete OWF, PRG, leaf commitment mechanisms, and ZKP constraint systems to match the selected block ciphers.

4 Construction of SM4th

This section presents the construction of SM4th, which uses two proving methods for OWFs in the non-EM and EM variants, respectively. The EM version commits all outputs of S-boxes using degree-2 polynomial constraints, while the non-EM one uses norm-based technique to avoid committing all S-box outputs but with degree-3 polynomial constraints.

4.1 The SM4 Block Cipher

The SM4 algorithm [1] is a Chinese standard block cipher with a 128-bit block size and a 128-bit key size, running in $R = 32$ rounds for both the key expansion and encryption/decryption routines. In SM4th, we ignore the decryption routine and use the encryption routine (along with the key expansion) as the OWF and PRG. SM4 uses the following building blocks.

The SM4 S-box. S-box is a non-linear function mapping 8 bits to 8 bits. Its algebraic representation is analyzed in [31] with details below.Given a $\boldsymbol{x}$ of 8 bits, the S-box function $S(\cdot)$ is defined as $S(\boldsymbol{x}) = \boldsymbol{A} \cdot I(\boldsymbol{A} \cdot \boldsymbol{x} + \boldsymbol{c}) + \boldsymbol{c}$. Here, $I(\boldsymbol{x})$ is the patched inversion over $\mathbb{F}_{2^8}$, and $\boldsymbol{A} \in \mathbb{F}_2^{8\times 8}$ and $\boldsymbol{c} \in \mathbb{F}_2^8$ are:

$$\boldsymbol{A} = \begin{pmatrix} 1&1&1&0&0&1&0&1 \\ 1&1&1&1&0&0&1&0 \\ 0&1&1&1&1&0&0&1 \\ 1&0&1&1&1&1&0&0 \\ 0&1&0&1&1&1&1&0 \\ 0&0&1&0&1&1&1&1 \\ 1&0&0&1&0&1&1&1 \\ 1&1&0&0&1&0&1&1 \end{pmatrix}, \qquad \boldsymbol{c} = (1,1,0,0,1,0,1,1).$$

The irreducible polynomial is

$$m_{\mathsf{SM4}}(T) = T^8 + T^7 + T^6 + T^5 + T^4 + T^2 + 1 \in \mathbb{F}_2[T]. \tag{1}$$

The $T(\cdot)$ function. $T(\cdot) : \mathbb{F}_2^{32} \to \mathbb{F}_2^{32}$ is a composite substitution function composed of S-boxes and linear transformation. Two linear transformations, with slight differences, denoted as $L'(\cdot)$ and $L(\cdot)$, are used in the key expansion and encryption routines, respectively. $T(\cdot)$ includes two phases below.

- S-box phases. Assume the input is $\boldsymbol{a} = (\boldsymbol{a}[0], \boldsymbol{a}[1], \boldsymbol{a}[2], \boldsymbol{a}[3]) \in (\mathbb{F}_2^8)^4$, and the output is $\boldsymbol{b} = (\boldsymbol{b}[0], \boldsymbol{b}[1], \boldsymbol{b}[2], \boldsymbol{b}[3]) \in (\mathbb{F}_2^8)^4$. Then we have $(\boldsymbol{b}[0], \boldsymbol{b}[1], \boldsymbol{b}[2], \boldsymbol{b}[3]) = (S(\boldsymbol{a}[0]), S(\boldsymbol{a}[1]), S(\boldsymbol{a}[2]), S(\boldsymbol{a}[3]))$.
- Linear transformations $L(\cdot)$ and $L'(\cdot)$. Assume the input is $\boldsymbol{b} = (\boldsymbol{b}[0], \boldsymbol{b}[1], \boldsymbol{b}[2], \boldsymbol{b}[3]) \in (\mathbb{F}_2^8)^4$, and the output is $\boldsymbol{c} = (\boldsymbol{c}[0], \boldsymbol{c}[1], \boldsymbol{c}[2], \boldsymbol{c}[3]) \in (\mathbb{F}_2^8)^4$. In the encryption routine, we have $\boldsymbol{c} = L(\boldsymbol{b}) = \boldsymbol{b} \oplus (\boldsymbol{b} \lll 2) \oplus (\boldsymbol{b} \lll 10) \oplus (\boldsymbol{b} \lll 18) \oplus (\boldsymbol{b} \lll 24)$, where $\lll$ means cyclic left shift. In the key expansion routine, we have $\boldsymbol{c} = L'(\boldsymbol{b}) = \boldsymbol{b} \oplus (\boldsymbol{b} \lll 13) \oplus (\boldsymbol{b} \lll 23)$.

The key expansion and encryption routines. Given the initial key $\boldsymbol{k} = (\boldsymbol{k}[0], \boldsymbol{k}[1], \boldsymbol{k}[2], \boldsymbol{k}[3]) \in (\mathbb{F}_2^{32})^4$, suppose the expanded key $\bar{\boldsymbol{k}} \in (\mathbb{F}_2^{32})^{36}$, system parameters $\{FK[i]\}_{i\in[0,3]}$, and constant parameters $\{CK[i]\}_{i\in[0,31]}$. These parameters are fixed and public. Then, for key expansion, we have

$$(\bar{\boldsymbol{k}}[0],\bar{\boldsymbol{k}}[1], \bar{\boldsymbol{k}}[2], \bar{\boldsymbol{k}}[3]) = (\boldsymbol{k}[0] \oplus FK[0], \boldsymbol{k}[1] \oplus FK[1], \boldsymbol{k}[2] \oplus FK[2], \boldsymbol{k}[3] \oplus FK[3]),$$
$$RK[i] = \bar{\boldsymbol{k}}[i+4] = \bar{\boldsymbol{k}}[i] \oplus T'(\bar{\boldsymbol{k}}[i+1] \oplus \bar{\boldsymbol{k}}[i+2] \oplus \bar{\boldsymbol{k}}[i+3] \oplus CK[i]), \forall i \in [0, 31],$$

where $T'(\cdot)$ substitues the $L(\cdot)$ as $L'(\cdot)$ based on $T(\cdot)$.

Given the plaintext $\boldsymbol{z} = (\boldsymbol{z}[0], \boldsymbol{z}[1], \boldsymbol{z}[2], \boldsymbol{z}[3]) \in (\mathbb{F}_2^{32})^4$ and round keys $\{RK[i]\}_{i\in[0,31]}$, suppose the ciphertext is $\boldsymbol{y} = (\boldsymbol{y}[0], \boldsymbol{y}[1], \boldsymbol{y}[2], \boldsymbol{y}[3]) \in (\mathbb{F}_2^{32})^4$. Then, for the encryption routine, we have

$$\boldsymbol{z}[i+4] = \boldsymbol{z}[i] \oplus T(\boldsymbol{z}[i+1] \oplus \boldsymbol{z}[i+2] \oplus \boldsymbol{z}[i+3] \oplus RK[i]), \forall i \in [0, 31],$$
$$(\boldsymbol{y}[0], \boldsymbol{y}[1], \boldsymbol{y}[2], \boldsymbol{y}[3]) = (\boldsymbol{z}[35], \boldsymbol{z}[34], \boldsymbol{z}[33], \boldsymbol{z}[32]).$$

Throughout this section, square brackets index components of vector-valued sequences: $\boldsymbol{z}[i]$ denotes the i-th 32-bit word of the trace, and $\boldsymbol{z}[i][j]$ its j-th byte for $j \in [0, 4)$. The same convention applies to key words and to other word sequences.

4.2 The ZKP Constraint System of SM4th-EM

This section introduces the ZKP constraint system of SM4th-EM, which uses the EM construction $\mathrm{SM4}_{\boldsymbol{x}}(\boldsymbol{k}) \oplus \boldsymbol{k} = \boldsymbol{y}$ as an OWF. In this setting, the SM4 key $\boldsymbol{x}$ are public, and hence the round keys $\{RK[i]\}_{i\in[0,31]}$ can be generated explicitly by the prover and verifier via the key expansion routine.

Embedding. All SM4th parameter sets use a fixed $\mathbb{F}_2$-field embedding $\phi : \mathbb{F}_{2^8} \hookrightarrow \mathbb{F}_{2^{128}}$ compatible with the SM4 byte polynomial. It preserves addition, multiplication, and squaring: for all $a, b \in \mathbb{F}_{2^8}$, $\phi(a+b) = \phi(a)+\phi(b)$, $\phi(ab) = \phi(a)\phi(b)$, and $\phi(a^2) = \phi(a)^2$.

For a degree-1 commitment $\langle \boldsymbol{b} \rangle$ to the eight bits of a byte b, the procedure EmbedAndSquare locally derives commitments $\langle \phi(b) \rangle$ and $\langle \phi(b)^2 \rangle = \langle \phi(b^2) \rangle$. Both outputs are fixed public $\mathbb{F}_2$-linear combinations of the input-bit commitments; hence no commitment multiplication or degree increase is required. Appendix D specifies the embedding, its implementation constant and its correctness.

SM4th-EM.DeriveConstraints. Algorithm 1 describes the constraint deriving procedure for the prover or the verifier in SM4th-EM. Its inputs are the commitments to the extended witness $\boldsymbol{w} = \boldsymbol{k} \| \boldsymbol{z}[4] \| \cdots \| \boldsymbol{z}[31]$, where $\boldsymbol{z}[i]$ is the i-th intermediate SM4 state word in the encryption routine. The output is $\langle \boldsymbol{o} \rangle$, a degree-2 vector commitment to the constraints. It is the SM4th-EM instantiation of the $\mathsf{DeriveConstraints}_{\mathsf{param}}$ interface of Equation (2) in Section 2; Appendix B invokes it inside ProveConstraints and VerifyConstraints before the ZKHash step.

Algorithm 1 goes as follows. First, the round keys $\bar{\boldsymbol{x}}$ are computed and transformed into VOLE commitments. All commitments to the round keys $\bar{\boldsymbol{x}}$ are public constants because the SM4 key $\boldsymbol{x}$ is public. Second, VOLE commitments

for the inputs and outputs of all round functions, *i.e.*, $\{\boldsymbol{z}[i]\}_{i\in[0,35]}$ are dervied. Then, for each round, the two constraints for each of the four S-boxes are appended to the output vector in round-major order. By construction, every entry of this vector is zero if all computations are correct. Specifically, in round i, the algorithm derives the S-box input word $\boldsymbol{a}[i]$ from $\boldsymbol{z}[i+1], \boldsymbol{z}[i+2], \boldsymbol{z}[i+3]$ and the public expanded-key word, and recovers the claimed S-box output word $\boldsymbol{s}[i] = L^{-1}(\boldsymbol{z}[i] \oplus \boldsymbol{z}[i+4])$. For each byte $j \in [0,4)$, it applies the input affine map to $\boldsymbol{a}[i][j]$ and the inverse output-affine map to $\boldsymbol{s}[i][j]$, then invokes $\mathsf{EmbedAndSquare}$ on both results, obtaining $\langle \hat{x}_{i,j} \rangle$, $\langle \hat{x}_{i,j,\mathsf{sq}} \rangle$, $\langle \hat{y}_{i,j} \rangle$, and $\langle \hat{y}_{i,j\mathsf{sq}} \rangle$. Then, the parties compute the degree-2 constraint commitments

$$\langle o_0 \rangle^2 = \langle \hat{x}_{i,j,\mathsf{sq}} \rangle \cdot \langle \hat{y}_{i,j} \rangle + \langle \hat{x}_{i,j} \rangle, \qquad \langle o_1 \rangle^2 = \langle \hat{x}_{i,j} \rangle \cdot \langle \hat{y}_{i,j,\mathsf{sq}} \rangle + \langle \hat{y}_{i,j} \rangle,$$

which should have zero leading coefficients exactly when the patched inversion is correct. Finally, $\langle o_0 \rangle^2$ and $\langle o_1 \rangle^2$ are appended into $\langle \boldsymbol{o} \rangle$.

Algorithm 1: $\mathsf{SM4th\text{-}EM.DeriveConstraints}$ from VOLE commitments

Input: Degree-1 VOLE commitments $\langle \boldsymbol{w} \rangle$ for $\boldsymbol{w} = \boldsymbol{k} \| \boldsymbol{z}[4] \| \cdots \| \boldsymbol{z}[31]$, public key $(\boldsymbol{x}, \boldsymbol{y})$
Output: Degree-2 constraint vector $\langle \boldsymbol{o} \rangle$ of length 256
Initialize $\langle \boldsymbol{o} \rangle \leftarrow ()$;
Compute $\overline{\boldsymbol{x}} \leftarrow \mathsf{SM4.KeyExpansion}(\boldsymbol{x})$ and obtain $\langle \overline{x} \rangle$;
Set $\langle \boldsymbol{z}[0] \rangle, \ldots, \langle \boldsymbol{z}[3] \rangle \leftarrow \langle \boldsymbol{k} \rangle$ and $\langle \boldsymbol{z}[4] \rangle, \ldots, \langle \boldsymbol{z}[31] \rangle \leftarrow \langle \boldsymbol{w}_{\mathsf{mid}} \rangle$;
Set $\langle \boldsymbol{out} \rangle \leftarrow \langle \boldsymbol{k} \rangle + \boldsymbol{y}$ and $\langle \boldsymbol{z}[32+i] \rangle \leftarrow \langle \boldsymbol{out}[3-i] \rangle$ for $i \in [0,4)$;
for $i \in [0, R)$ **do**
  $\langle \boldsymbol{a}[i] \rangle \leftarrow \langle \boldsymbol{z}[i+1] \rangle + \langle \boldsymbol{z}[i+2] \rangle + \langle \boldsymbol{z}[i+3] \rangle + \langle \overline{\boldsymbol{x}}[i+4] \rangle$;
  $\langle \boldsymbol{s}[i] \rangle \leftarrow L^{-1}\big(\langle \boldsymbol{z}[i] \rangle + \langle \boldsymbol{z}[i+4] \rangle\big)$;
  for $j \in [0,4)$ **do**
    $\langle \boldsymbol{r}[i,j] \rangle \leftarrow \boldsymbol{A}\langle \boldsymbol{a}[i][j] \rangle + \boldsymbol{c}$ and $\langle \boldsymbol{t}[i,j] \rangle \leftarrow \boldsymbol{A}^{-1}\big(\langle \boldsymbol{s}[i][j] \rangle + \boldsymbol{c}\big)$;
    $(\langle \hat{x}_{i,j} \rangle, \langle \hat{x}_{i,j,\mathsf{sq}} \rangle) \leftarrow \mathsf{EmbedAndSquare}(\langle \boldsymbol{r}[i,j] \rangle)$;
    $(\langle \hat{y}_{i,j} \rangle, \langle \hat{y}_{i,j,\mathsf{sq}} \rangle) \leftarrow \mathsf{EmbedAndSquare}(\langle \boldsymbol{t}[i,j] \rangle)$;
    $(\langle o_0 \rangle^2, \langle o_1 \rangle^2) \leftarrow (\langle \hat{x}_{i,j,\mathsf{sq}} \rangle \cdot \langle \hat{y}_{i,j} \rangle + \langle \hat{x}_{i,j} \rangle, \langle \hat{x}_{i,j} \rangle \cdot \langle \hat{y}_{i,j,\mathsf{sq}} \rangle + \langle \hat{y}_{i,j} \rangle)$;
    Append $\langle o_0 \rangle^2$ and $\langle o_1 \rangle^2$ to $\langle \boldsymbol{o} \rangle$;
  end
end
return $\langle \boldsymbol{o} \rangle$;

Summary. The committed $\mathsf{SM4th\text{-}EM}$ witness has $128 + 32(R-4) = 1024$ bits: the secret input of 128 bits and the $R-4 = 28$ sets of private intermediate state words. There are $2 \cdot 4R = 256$ S-box constraints. All constraints have degree 2.

4.3 The ZKP Constraint System of SM4th

The non-EM verion uses the OWF $\mathrm{SM4}_{\boldsymbol{k}}(\boldsymbol{x}) = \boldsymbol{y}$. Consequently, its committed trace covers both the key expansion and the encryption routine, unlike $\mathsf{SM4th\text{-}EM}$, whose round keys are public.

Norm witnesses and anchor words. FAEST uses norm compression solely in its encryption trace by committing norm values in alternating odd and even rounds. This is feasible due to the sequential structure of AES. However, this technique cannot be directly applied to SM4th, as SM4 instead has a cyclic four-word recurrence: each newly generated word is used in the S-box inputs of the following three recursions. We represent one word by a norm witness, retain the next three words in full, and apply this norm-and-three-full-words pattern to both the key expansion and encryption routines.

We first explain the norm technique in more details. Consider a byte whose input to the patched inversion, after the public input-affine map, is $x \in \mathbb{F}_{2^8}$. At an anchor, the witness does not store the full S-box output byte. Instead, it stores a four-bit norm coordinate $\boldsymbol{n} = (n_0, n_1, n_2, n_3) \in \{0,1\}^4$ for $h = x^{-17} \in \mathbb{F}_{2^4}$, the norm of x^{-1} from $\mathbb{F}_{2^8}$ to its fixed subfield $\mathbb{F}_{2^4}$. With respect to the fixed public basis $(1, \beta, \beta^2, \beta^3)$ of that subfield, the four bits mean $h = n_0 + n_1\beta + n_2\beta^2 + n_3\beta^3$. That is to say, a norm coordinate is neither an ordinary byte nor an S-box output: it is a compact representation of the auxiliary subfield value h.

The four committed bits are expanded by public linear operations and the embedding $\phi : \mathbb{F}_{2^8} \hookrightarrow \mathbb{F}_{2^\lambda}$ to $\langle \hat{h} \rangle$ and its Frobenius conjugates, *i.e.*, $\langle \hat{h}^2 \rangle, \langle \hat{h}^4 \rangle$, and $\langle \hat{h}^8 \rangle$. With the embedded affine input $\langle \hat{x} \rangle = \langle \phi(x) \rangle$, they are checked by

$$\langle \hat{h} \rangle \cdot \langle \hat{x}^{16} \rangle \cdot \langle \hat{x}^2 \rangle + \langle \hat{x} \rangle = 0.$$

When this equality holds, the parties recover $\langle \hat{q} \rangle = \langle \hat{x}^{16} \rangle \cdot \langle \hat{h} \rangle$ for $\hat{q} = \hat{x}^{-1}$, then apply the public output-affine map and the appropriate word linear layer to reconstruct the virtual anchor word. The norm segments store these four-bit coordinates in anchor-index-major and byte-major order: $\boldsymbol{w}_{\mathsf{ke},N}$ contains $\boldsymbol{n}_{\mathsf{ke}}[i][j]$ for $i \in \mathcal{A}_{\mathsf{ke}}$, and $\boldsymbol{w}_{\mathsf{enc},N}$ contains $\boldsymbol{n}_{\mathsf{enc}}[a][j]$ for $a \in \mathcal{A}_{\mathsf{enc}}$, with $j \in [0,4)$.

Key expansion routine. We now present the key expansion pattern. For an anchor index i, we have

$$\bar{\boldsymbol{k}}[i] = \bar{\boldsymbol{k}}[i-4] \oplus T'(\boldsymbol{u}[i]), \qquad \boldsymbol{u}[i] = \bar{\boldsymbol{k}}[i-3] \oplus \bar{\boldsymbol{k}}[i-2] \oplus \bar{\boldsymbol{k}}[i-1] \oplus CK[i-4].$$

The three words entering $\boldsymbol{u}[i]$ are full, degree-1 commitments.

For each byte $\boldsymbol{u}[i][j]$, where $j \in [0,4)$, both parties first derive a degree-1 commitment $\langle \hat{x}_{i,j} \rangle^1$ to the embedded S-box input after the public input-affine map, together with its Frobenius powers such as $\langle \hat{x}_{i,j}^2 \rangle^1$ and $\langle \hat{x}_{i,j}^{16} \rangle^1$. These are derived commitments, rather than additional witness values. At an anchor, the coordinate $\langle \boldsymbol{n}_{\mathsf{ke}}[i][j] \rangle$ expands to a 4-bit degree-1 commitment $\langle \hat{h}_{i,j} \rangle^1$ to $h_{i,j} = x_{i,j}^{-17} \in \mathbb{F}_{2^4}$, instead of the S-box output byte. The parties append the constraint

$$\langle o_{i,j} \rangle^3 = \langle \hat{h}_{i,j} \rangle^1 \cdot \langle \hat{x}_{i,j}^{16} \rangle^1 \cdot \langle \hat{x}_{i,j}^2 \rangle^1 + \langle \hat{x}_{i,j} \rangle^1.$$

When this constraint vanishes, the committed norm value is consistent with the patched inverse of $x_{i,j}$. The parties can then reconstruct the inverse, beginning with $\langle \hat{q}_{i,j} \rangle^2 = \langle \hat{x}_{i,j}^{16} \rangle^1 \cdot \langle \hat{h}_{i,j} \rangle^1$, and likewise reconstruct its required Frobenius conjugates, *i.e.*, $\langle \hat{q}_{i,j}^{2^r} \rangle^2 = \langle \hat{x}_{i,j}^{2^{(r+4) \bmod 8}} \rangle^1 \cdot \langle \hat{h}_{i,j}^{2^{r \bmod 4}} \rangle^1$, where $r \in [0,8)$. These

are local derived values, and are used to evaluate the public output-affine map and L' in their linearized forms, yielding the locally maintained embedded tuple $\langle\widehat{\bar{\boldsymbol{k}}}[i]\rangle^2$ for the virtual anchor word; this tuple supplies the virtual predecessor when forming $\langle\hat{x}_{i+1,j}\rangle$.

The next word $\bar{\boldsymbol{k}}[i+1]$ is a non-anchor and is stored in full as a degree-1 commitment. It is

$$\bar{\boldsymbol{k}}[i+1] = \bar{\boldsymbol{k}}[i-3] \oplus T'(\boldsymbol{u}[i+1]), \qquad \boldsymbol{u}[i+1] = \bar{\boldsymbol{k}}[i-2] \oplus \bar{\boldsymbol{k}}[i-1] \oplus \bar{\boldsymbol{k}}[i] \oplus CK[i-3].$$

Since $\bar{\boldsymbol{k}}[i]$ is degree 2, the four derived S-box inputs $\langle\hat{x}_{i+1,j}\rangle^2$, for $j \in [0,4)$, are also degree 2. For every $j \in [0,4)$, rather than recomputing the S-box output in the forward direction, the parties use the claimed full word $\bar{\boldsymbol{k}}[i+1]$ to recover that output through the inverse linear layer:

$$\langle \boldsymbol{s}[i+1]\rangle^1 = L'^{-1}\big(\langle\bar{\boldsymbol{k}}[i-3]\rangle^1 + \langle\bar{\boldsymbol{k}}[i+1]\rangle^1\big).$$

Undoing the output-affine map gives degree-1 commitments $\langle\hat{q}_{i+1,j}\rangle^1$ and $\langle\hat{q}^2_{i+1,j}\rangle^1$ to the claimed patched-inversion output and its square. For every $j \in [0,4)$, the parties then append the direct S-box constraints

$$\langle o_{i+1,j,0}\rangle^3 = \langle\hat{x}^2_{i+1,j}\rangle^2 \cdot \langle\hat{q}_{i+1,j}\rangle^1 + \langle\hat{x}_{i+1,j}\rangle^2, \quad \langle o_{i+1,j,1}\rangle^3 = \langle\hat{x}_{i+1,j}\rangle^2 \cdot \langle\hat{q}^2_{i+1,j}\rangle^1 + \langle\hat{q}_{i+1,j}\rangle^1.$$

Thus, the full witness word $\bar{\boldsymbol{k}}[i+1]$ is accepted only if it is consistent with the S-box input determined by its three predecessors, including the reconstructed anchor $\bar{\boldsymbol{k}}[i]$.

The next two words $\bar{\boldsymbol{k}}[i+2]$ and $\bar{\boldsymbol{k}}[i+3]$ are also non-anchors and are retained in full. The earliest possible next anchor is therefore $\bar{\boldsymbol{k}}[i+4]$, which is $\bar{\boldsymbol{k}}[i+4] = \bar{\boldsymbol{k}}[i] \oplus T'\big(\bar{\boldsymbol{k}}[i+1] \oplus \bar{\boldsymbol{k}}[i+2] \oplus \bar{\boldsymbol{k}}[i+3] \oplus CK[i]\big)$.

Its S-box input contains only the three full degree-1 words $\bar{\boldsymbol{k}}[i+1], \bar{\boldsymbol{k}}[i+2], \bar{\boldsymbol{k}}[i+3]$; the previous degree-2 anchor $\bar{\boldsymbol{k}}[i]$ occurs only outside the S-box, in a linear XOR. Consequently, $\bar{\boldsymbol{k}}[i+4]$ can again be reconstructed as a degree-2 anchor. Intuitively, this should be the earliest anchor as if an anchor were placed earlier, its S-box input would already contain the degree-2 word $\bar{\boldsymbol{k}}[i]$, causing the norm check to exceed degree 3. Hence, three full words must separate consecutive anchors; choosing the period-four pattern gives the densest such compression:

$$\mathcal{A}_{\mathsf{ke}} = \{4, 8, 12, 16, 20, 24, 28, 32\}.$$

Encryption routine. The encryption trace uses the same period-four pattern, but with a one-word offset relative to the key expansion. In the first encryption recurrence, the S-box input for $\boldsymbol{z}[4]$ contains $RK[0] = \bar{\boldsymbol{k}}[4]$, which is a key anchor and hence a degree-2 commitment. If $\boldsymbol{z}[4]$ were also an anchor, its norm constraint would contain this degree-2 S-box input and exceed degree 3. Thus, $\boldsymbol{z}[4]$ is retained in full. For an encryption anchor a, the S-box input is

$$\boldsymbol{u}[a] = \boldsymbol{z}[a-3] \oplus \boldsymbol{z}[a-2] \oplus \boldsymbol{z}[a-1] \oplus \bar{\boldsymbol{k}}[a], \qquad \boldsymbol{z}[a] = \boldsymbol{z}[a-4] \oplus T(\boldsymbol{u}[a]).$$

For each byte $j \in [0, 4)$, the input-affine map gives the embedded value $\langle \hat{x}_{a,j} \rangle$ and its Frobenius powers. The coordinate $\langle \boldsymbol{n}_{\mathsf{enc}}[a][j] \rangle$ expands to $\langle \hat{h}_{a,j} \rangle$, and the norm branch appends

$$\langle o^N_{a,j} \rangle^3 = \langle \hat{h}_{a,j} \rangle^1 \cdot \langle \hat{x}^{16}_{a,j} \rangle^1 \cdot \langle \hat{x}^2_{a,j} \rangle^1 + \langle \hat{x}_{a,j} \rangle^1.$$

The corresponding tuple $(\langle \hat{q}_{a,j} \rangle^2, \langle \hat{q}^2_{a,j} \rangle^2, \ldots, \langle \hat{q}^{128}_{a,j} \rangle^2)$ is reconstructed as above and is used by the output-affine map and L to obtain the embedded representation of the virtual state word $\boldsymbol{z}[a]$. Choosing $a \equiv 1 \pmod 4$ ensures that $\bar{\boldsymbol{k}}[a] = RK[a-4]$ is a full, non-anchor round key, since every key-expansion anchor has index $0 \pmod 4$. The first possible such state anchor is therefore $\boldsymbol{z}[5]$. After an anchor $\boldsymbol{z}[a]$, the next three state words $\boldsymbol{z}[a+1], \boldsymbol{z}[a+2], \boldsymbol{z}[a+3]$ are retained in full; they form the S-box input of the next anchor $\boldsymbol{z}[a+4]$, while $\boldsymbol{z}[a]$ occurs only in that recurrence's outer linear XOR. Hence, the same degree argument requires, and permits, one anchor every four state words. This gives

$$\mathcal{A}_{\mathsf{enc}} = \{5, 9, 13, 17, 21, 25, 29\}.$$

SM4th.DeriveConstraints. Algorithm 2 describes the constraint deriving procedure for the prover or the verifier in SM4th. Its inputs are the degree-1 commitments to the extended witness, *i.e.*,

$$\boldsymbol{w} = \bar{\boldsymbol{k}}[0\,..\,4) \parallel \boldsymbol{w}_{\mathsf{ke},N} \parallel \boldsymbol{w}_{\mathsf{ke},F} \parallel \boldsymbol{w}_{\mathsf{enc},N} \parallel \boldsymbol{w}_{\mathsf{enc},F},$$

where ke and enc are the segments corresponding to the key expansion and the encryption routines, and N and F are the norm and full segments, respectively. Their lengths are 128, 128, 768, 112, and 672 bits, respectively. The output is $\langle \boldsymbol{o} \rangle$, a vector of 452 degree-3 constraint commitments. It is the SM4th instantiation of the $\mathsf{DeriveConstraints}_{\mathsf{param}}$ interface of Equation (2) in Section 2; Appendix B invokes it inside ProveConstraints and VerifyConstraints before the ZKHash step.

The algorithm uses four compact subprocedures. EmbedAndSquare is the procedure from Section 4.2: it maps an affine byte commitment to its embedded value and its square. Next maintains an implicit cursor in a full-word segment and returns the next unconsumed 32-bit degree-1 word commitment. It performs neither arithmetic nor a constraint check.

$\mathsf{NormSBox}(\langle \hat{x} \rangle, \langle \hat{x}_{\mathsf{sq}} \rangle, \langle \boldsymbol{n} \rangle)$ takes the embedded affine input and its square together with a four-bit norm coordinate. It expands the coordinate to $\hat{h}$ and its conjugates, checks the norm relation, and returns the reconstructed S-box-output byte together with one degree-3 norm constraint. $\mathsf{DirectSBox}(\langle \hat{x} \rangle, \langle \hat{x}_{\mathsf{sq}} \rangle, \langle \hat{y} \rangle, \langle \hat{y}_{\mathsf{sq}} \rangle)$ takes an embedded affine input/output pair and returns two degree-3 constraints.

Algorithm 2 first parses the initial four key words, the two norm-coordinate arrays, and the two full-word streams. It then processes the key expansion in increasing word index: an anchor is reconstructed by NormSBox, whereas a non-anchor is read by Next and checked by DirectSBox. Next, it initializes the encryption trace with the public input and output boundaries and processes its rounds in increasing order by the same two branches. The encryption branch uses L in place of L'; it reads an internal full state word with Next and uses

the public output boundary for the final state words. Every returned norm or direct constraint is appended in the order generated by these loops. Thus all 452 entries are zero when both traces are consistent with the committed witness.

Concretely, let $i \in \mathcal{A}_{\mathsf{ke}}$ be a key-expansion anchor. The algorithm forms $\boldsymbol{u}[i] = \bar{\boldsymbol{k}}[i-3] \oplus \bar{\boldsymbol{k}}[i-2] \oplus \bar{\boldsymbol{k}}[i-1] \oplus CK[i-4]$. For each $j \in [0,4)$, it embeds the affine image of $\boldsymbol{u}[i][j]$ and calls $\mathsf{NormSBox}$ with $\langle \boldsymbol{n}_{\mathsf{ke}}[i][j] \rangle$. This returns the S-box byte and the norm constraint

$$\langle o^N_{i,j} \rangle^3 = \langle \hat{h}_{i,j} \rangle \cdot \langle \hat{x}^{16}_{i,j} \rangle \cdot \langle \hat{x}^2_{i,j} \rangle + \langle \hat{x}_{i,j} \rangle.$$

Concatenating the four returned S-box bytes and applying L' reconstructs the virtual word $\langle \bar{\boldsymbol{k}}[i] \rangle$. At the following non-anchor $i+1$, Next reads the claimed full word $\langle \bar{\boldsymbol{k}}[i+1] \rangle$. The algorithm obtains its claimed S-box output as $L'^{-1}(\langle \bar{\boldsymbol{k}}[i-3] \rangle + \langle \bar{\boldsymbol{k}}[i+1] \rangle)$, embeds the affine input/output pair $(\langle \hat{x}_{i+1,j} \rangle, \langle \hat{x}_{i+1,j,\mathsf{sq}} \rangle)$ and $(\langle \hat{y}_{i+1,j} \rangle, \langle \hat{y}_{i+1,j,\mathsf{sq}} \rangle)$, and uses $\mathsf{DirectSBox}$ to append

$$\langle o^D_{i+1,j,0} \rangle^3 = \langle \hat{x}_{i+1,j,\mathsf{sq}} \rangle \cdot \langle \hat{y}_{i+1,j} \rangle + \langle \hat{x}_{i+1,j} \rangle,$$
$$\langle o^D_{i+1,j,1} \rangle^3 = \langle \hat{x}_{i+1,j} \rangle \cdot \langle \hat{y}_{i+1,j,\mathsf{sq}} \rangle + \langle \hat{y}_{i+1,j} \rangle.$$

Summary. The non-EM witness has 1808 committed bits. The $8 \cdot 4 + 7 \cdot 4 = 60$ anchor bytes contribute one norm constraint each, while the remaining $(128 - 32) + (128 - 28) = 196$ S-boxes contribute two direct constraints each. This gives $C_{\mathsf{non\text{-}EM}} = 60 + 2 \cdot 196 = 452$ constraints, all of degree 3.

4.4 Construction of Signature Schemes

We instantiate the FAEST VOLEitH signature framework with the two SM4 one-way functions studied above, leading to SM4th and SM4th-EM. The resulting ZKP parameters are compared with the corresponding FAEST instances in Table 1; the FAEST figures are from [9]. Note that the actual signature scheme requires one additional constraint due to the key-space reduction, which is described in Appendix B.

The extended-witness column $\hat{\ell}$ is the total length committed by VOLE, rather than the OWF witness length ℓ. As described in FAEST [9],

$$\hat{\ell} = \ell + (\lambda + B) + (d-1)\lambda = \ell + d\lambda + B.$$

The first $\lambda + B$ bits are masks for the linear value revealed in the VOLE consistency check, which is needed as a malicious prover may send inconsistent values for multiple GGM sub-trees. The remaining $(d-1)\lambda$ bits mask the $d-1$ nonconstant coefficient blocks disclosed by ZKHash, as the values $\tilde{a}_1, \ldots, \tilde{a}_{d-1}$, introduced in Section 2, may leak additional information. Table 1 uses $\lambda = 128$ and $B = 16$. Although SM4th-EM has a 64-bit longer OWF witness than FAEST-EM-128, its degree-2 constraint system has a smaller total extended-witness length (1 296 rather than 1 360 bits), so it is expected to yield a smaller signature size under corresponding parameter profiles.

Algorithm 2: SM4th.DeriveConstraints from VOLE commitments

INPUT: Degree-1 VOLE commitments $\langle \boldsymbol{w} \rangle$ to $\boldsymbol{w} = \bar{\boldsymbol{k}}[0\,..\,4) \| \boldsymbol{w}_{\mathsf{ke},N} \| \boldsymbol{w}_{\mathsf{ke},F} \| \boldsymbol{w}_{\mathsf{enc},N} \| \boldsymbol{w}_{\mathsf{enc},F}$, public input/output blocks $(\boldsymbol{x}, \boldsymbol{y})$
OUTPUT: Degree-3 constraint vector $\langle \boldsymbol{o} \rangle$ of length 452
Set $\mathcal{A}_{\mathsf{ke}} \leftarrow \{4, 8, 12, 16, 20, 24, 28, 32\}$ and $\mathcal{A}_{\mathsf{enc}} \leftarrow \{5, 9, 13, 17, 21, 25, 29\}$;
Initialize $\langle \boldsymbol{o} \rangle \leftarrow ()$;
Parse $\langle \boldsymbol{w} \rangle$ into $\langle \bar{\boldsymbol{k}}[0\,..\,4) \rangle$, $\langle \boldsymbol{w}_{\mathsf{ke},N} \rangle$, $\langle \boldsymbol{w}_{\mathsf{ke},F} \rangle$, $\langle \boldsymbol{w}_{\mathsf{enc},N} \rangle$, $\langle \boldsymbol{w}_{\mathsf{enc},F} \rangle$, and their norm-coordinate arrays $\langle \boldsymbol{n}_{\mathsf{ke}}[i][j] \rangle$ and $\langle \boldsymbol{n}_{\mathsf{enc}}[a][j] \rangle$;
for $i \in [4, R+4)$ **do**
  $\langle \boldsymbol{u} \rangle \leftarrow \langle \bar{\boldsymbol{k}}[i-3] \rangle + \langle \bar{\boldsymbol{k}}[i-2] \rangle + \langle \bar{\boldsymbol{k}}[i-1] \rangle + CK[i-4]$;
  if $i \in \mathcal{A}_{\mathsf{ke}}$ **then**
    for $j \in [0, 4)$ **do**
      $(\langle \hat{x}_{i,j} \rangle, \langle \hat{x}_{i,j,\mathsf{sq}} \rangle) \leftarrow \mathsf{EmbedAndSquare}(\boldsymbol{A}\langle \boldsymbol{u}[j] \rangle + \boldsymbol{c})$;
      $(\langle \boldsymbol{s}[j] \rangle, \langle o^N_{i,j} \rangle^3) \leftarrow \mathsf{NormSBox}(\langle \hat{x}_{i,j} \rangle, \langle \hat{x}_{i,j,\mathsf{sq}} \rangle, \langle \boldsymbol{n}_{\mathsf{ke}}[i][j] \rangle)$;
      Append $\langle o^N_{i,j} \rangle^3$ to $\langle \boldsymbol{o} \rangle$;
    end
    $\langle \bar{\boldsymbol{k}}[i] \rangle \leftarrow \langle \bar{\boldsymbol{k}}[i-4] \rangle + L'\big(\langle \boldsymbol{s}[0] \rangle \| \langle \boldsymbol{s}[1] \rangle \| \langle \boldsymbol{s}[2] \rangle \| \langle \boldsymbol{s}[3] \rangle\big)$;
  end
  else
    $\langle \bar{\boldsymbol{k}}[i] \rangle \leftarrow \mathsf{Next}(\langle \boldsymbol{w}_{\mathsf{ke},F} \rangle)$;
    $\langle \boldsymbol{v} \rangle \leftarrow L'^{-1}(\langle \bar{\boldsymbol{k}}[i-4] \rangle + \langle \bar{\boldsymbol{k}}[i] \rangle)$;
    for $j \in [0, 4)$ **do**
      $(\langle \hat{x}_{i,j} \rangle, \langle \hat{x}_{i,j,\mathsf{sq}} \rangle) \leftarrow \mathsf{EmbedAndSquare}(\boldsymbol{A}\langle \boldsymbol{u}[j] \rangle + \boldsymbol{c})$ and $(\langle \hat{y}_{i,j} \rangle, \langle \hat{y}_{i,j,\mathsf{sq}} \rangle) \leftarrow \mathsf{EmbedAndSquare}(\boldsymbol{A}^{-1}(\langle \boldsymbol{v}[j] \rangle + \boldsymbol{c}))$;
      $(\langle o^D_{i,j,0} \rangle^3, \langle o^D_{i,j,1} \rangle^3) \leftarrow \mathsf{DirectSBox}(\langle \hat{x}_{i,j} \rangle, \langle \hat{x}_{i,j,\mathsf{sq}} \rangle, \langle \hat{y}_{i,j} \rangle, \langle \hat{y}_{i,j,\mathsf{sq}} \rangle)$;
      Append both direct outputs to $\langle \boldsymbol{o} \rangle$;
    end
  end
end
Set $\langle \boldsymbol{z}[0\,..\,4) \rangle \leftarrow \langle \boldsymbol{x} \rangle$ and $\langle \boldsymbol{z}[R+i] \rangle \leftarrow \langle \boldsymbol{y}[3-i] \rangle$ for $i \in [0, 4)$;
for $i \in [0, R)$ **do**
  $a \leftarrow i+4$ and $\langle \boldsymbol{u} \rangle \leftarrow \langle \boldsymbol{z}[i+1] \rangle + \langle \boldsymbol{z}[i+2] \rangle + \langle \boldsymbol{z}[i+3] \rangle + \langle \bar{\boldsymbol{k}}[i+4] \rangle$;
  if $a \in \mathcal{A}_{\mathsf{enc}}$ **then**
    for $j \in [0, 4)$ **do**
      $(\langle \hat{x}_{a,j} \rangle, \langle \hat{x}_{a,j,\mathsf{sq}} \rangle) \leftarrow \mathsf{EmbedAndSquare}(\boldsymbol{A}\langle \boldsymbol{u}[j] \rangle + \boldsymbol{c})$;
      $(\langle \boldsymbol{s}[j] \rangle, \langle o^N_{a,j} \rangle^3) \leftarrow \mathsf{NormSBox}(\langle \hat{x}_{a,j} \rangle, \langle \hat{x}_{a,j,\mathsf{sq}} \rangle, \langle \boldsymbol{n}_{\mathsf{enc}}[a][j] \rangle)$;
      Append $\langle o^N_{a,j} \rangle^3$ to $\langle \boldsymbol{o} \rangle$;
    end
    $\langle \boldsymbol{z}[a] \rangle \leftarrow \langle \boldsymbol{z}[i] \rangle + L\big(\langle \boldsymbol{s}[0] \rangle \| \langle \boldsymbol{s}[1] \rangle \| \langle \boldsymbol{s}[2] \rangle \| \langle \boldsymbol{s}[3] \rangle\big)$;
  end
  else
    if $a < R$ **then**
      $\langle \boldsymbol{z}[a] \rangle \leftarrow \mathsf{Next}(\langle \boldsymbol{w}_{\mathsf{enc},F} \rangle)$;
    end
    $\langle \boldsymbol{v} \rangle \leftarrow L^{-1}(\langle \boldsymbol{z}[i] \rangle + \langle \boldsymbol{z}[a] \rangle)$;
    for $j \in [0, 4)$ **do**
      $(\langle \hat{x}_{a,j} \rangle, \langle \hat{x}_{a,j,\mathsf{sq}} \rangle) \leftarrow \mathsf{EmbedAndSquare}(\boldsymbol{A}\langle \boldsymbol{u}[j] \rangle + \boldsymbol{c})$ and $(\langle \hat{y}_{a,j} \rangle, \langle \hat{y}_{a,j,\mathsf{sq}} \rangle) \leftarrow \mathsf{EmbedAndSquare}(\boldsymbol{A}^{-1}(\langle \boldsymbol{v}[j] \rangle + \boldsymbol{c}))$;
      $(\langle o^D_{a,j,0} \rangle^3, \langle o^D_{a,j,1} \rangle^3) \leftarrow \mathsf{DirectSBox}(\langle \hat{x}_{a,j} \rangle, \langle \hat{x}_{a,j,\mathsf{sq}} \rangle, \langle \hat{y}_{a,j} \rangle, \langle \hat{y}_{a,j,\mathsf{sq}} \rangle)$;
      Append both direct outputs to $\langle \boldsymbol{o} \rangle$;
    end
  end
end
return $\langle \boldsymbol{o} \rangle$ of length 452;

Table 1. ZKP and extended-witness lengths for SM4th and FAEST-128

Scheme	ZKP witness ℓ (bits)	Degree d	Extended witness $\hat{\ell}$ (bits)	Constraints C
SM4th	1 808	3	$1\,808 + 3 \cdot 128 + 16 = 2\,208$	452
SM4th-EM	1 024	2	$1\,024 + 2 \cdot 128 + 16 = 1\,296$	256
FAEST-128	1 280	3	$1\,280 + 3 \cdot 128 + 16 = 1\,680$	321
FAEST-EM-128	960	3	$960 + 3 \cdot 128 + 16 = 1\,360$	241

Small and fast variants. Each construction has a small (s) and a fast (f) variant. The two variants have identical OWF, PRG, extended witness, ZKP, field, GGM-tree commitment, and hash/XOF instantiations; they differ only in the VOLEitH parameter profile. In particular, s involves less GGM sub-trees, leading to smaller signature sizes but slower running times. f is faster, but incurs more communication overhead. Apart from the ZKP parameters in Table 1 and the SM4-based PRG/OWF instantiation, we use the corresponding FAEST parameter profile unchanged. The complete FAEST construction and its parameter selection are given in [9].

Theorem 1 (Security of SM4th and SM4th-EM). *Under Assumption 1,* SM4th *and* SM4th-EM *are* EUF-CMA*-secure in the random-oracle model against probabilistic polynomial-time adversaries and in the quantum-accessible random-oracle model against quantum polynomial-time adversaries. The formal direct ROM and QROM bounds are stated in Appendix C.2.*

The signing-query bound should be less than 2^{64}, which is limited by the birthday-bound due to the 128-bit IV. The bounds in Appendix C.2 specialize the corresponding FAEST analysis [9].

5 Construction of uBlockith

This section instantiates the VOLEitH framework with the 256-bit version of uBlock. The resulting schemes uBlockith and uBlockith-EM target 256-bit classical security. Both use the 24-round uBlock cipher as their one-way-function primitive and the Ballet cipher in counter mode as their PRG; the two schemes differ only in whether the block-cipher key or input is secret.

5.1 The uBlock Block Cipher

The block cipher uBlock is designed by Wu, Zhang, and Zhang *et al.* [38]. Since proposed, it has received both implementation [25,26] and cryptanalytic [37] study. The selected uBlock variant has a 256-bit key, a 256-bit block, and $R = 24$ rounds [38]. A round key $\bar{\boldsymbol{k}}[i]$ is a 256-bit word, written as four 64-bit double words $(\boldsymbol{k}_0, \boldsymbol{k}_1, \boldsymbol{k}_2, \boldsymbol{k}_3)$. The S-box is a 4-bit permutation. Both its Boolean algebraic normal form and that of its inverse have degree 3. We denote their parallel

application to 16 and 32 nibbles by SubDWord and SubQWord, respectively. Thus, applying either map to degree-1 VOLE commitments produces degree-3 ones.

For a 128-bit branch $\boldsymbol{v}$, $\boldsymbol{v}[q][h]$ denotes bit $h \in [0,4)$ of nibble $q \in [0,32)$. Byte $j \in [0,16)$ consists of high nibble $\boldsymbol{v}[2j]$ and low nibble $\boldsymbol{v}[2j+1]$; 32-bit word $w \in [0,4)$ consists of bytes $4w, \ldots, 4w+3$, from most to least significant. The state at two-round boundary r is written $\boldsymbol{X}[r] = (\boldsymbol{X}[r][0], \boldsymbol{X}[r][1])$, where the second index selects one of its two 128-bit branches.

Key expansion. To distinguish 64-bit key-schedule words from 128-bit encryption subkeys, write

$$\bar{\boldsymbol{k}}[i] = (\boldsymbol{d}[i][0], \boldsymbol{d}[i][1], \boldsymbol{d}[i][2], \boldsymbol{d}[i][3]), \quad \boldsymbol{K}[i][b] = \boldsymbol{d}[i][2b] \| \boldsymbol{d}[i][2b+1] \text{for } b \in \{0,1\}.$$

Here every $\boldsymbol{d}[i][j]$ is a 64-bit double word. In the key-schedule equations below, the temporary symbols $\boldsymbol{k}_0, \ldots, \boldsymbol{k}_3$ denote such double words; the encryption routine uses the 128-bit values $\boldsymbol{K}[i][0]$ and $\boldsymbol{K}[i][1]$. Let $\mathsf{PK}(\boldsymbol{k}_0, \boldsymbol{k}_1) = (\mathsf{PK}_0(\boldsymbol{k}_0, \boldsymbol{k}_1), \mathsf{PK}_1(\boldsymbol{k}_0, \boldsymbol{k}_1))$ denote the two 64-bit outputs of the public nibble permutation on the first two double words, and let T be the public linear map that multiplies each nibble of a 64-bit double word by 2 in $\mathbb{F}_{2^4}$ with modulus $t^4 + t + 1$. For $i \in [0, R)$, $\mathsf{RC}[i]$ is the $(i+1)$-st specified uBlock round constant. Starting from $\bar{\boldsymbol{k}}[0] = \boldsymbol{k}$, each round uses $\mathsf{RC}[i]$ and performs

$$(\boldsymbol{t}_0, \boldsymbol{t}_1) \leftarrow \mathsf{PK}(\boldsymbol{k}_0, \boldsymbol{k}_1), \qquad \boldsymbol{t}_2 \leftarrow \boldsymbol{k}_2 \oplus \mathsf{SubDWord}(\boldsymbol{t}_0 \oplus \mathsf{RC}[i]),$$
$$\boldsymbol{t}_3 \leftarrow \boldsymbol{k}_3 \oplus \mathsf{T}(\boldsymbol{t}_1), \quad \bar{\boldsymbol{k}}[i+1] \leftarrow (\boldsymbol{t}_2, \boldsymbol{t}_3, \boldsymbol{t}_1, \boldsymbol{t}_0).$$

Only the SubDWord is non-linear. Other operations, *i.e.*, PK and T, are linear.

Encryption. Write a 256-bit state as two 128-bit branches $\boldsymbol{Z} = (\boldsymbol{Z}[0], \boldsymbol{Z}[1])$. In round i, the two halves of $\bar{\boldsymbol{k}}[i]$, namely $\boldsymbol{K}[i][0]$ and $\boldsymbol{K}[i][1]$, are first xored into the branches, and SubQWord is applied to both. The resulting pair is processed by an invertible public linear network $\mathcal{L}_{\mathsf{u}}$: six ordered XOR updates, four of which use word rotations by $4, 8, 8, 20$ bits, followed by the public byte permutations PL and PR. Writing this complete linear layer as $\mathcal{L}_{\mathsf{u}}$, a round is

$$\boldsymbol{Z} \leftarrow \mathcal{L}_{\mathsf{u}}\big(\mathsf{SubQWord}(\boldsymbol{Z}[0] \oplus \boldsymbol{K}[i][0]), \mathsf{SubQWord}(\boldsymbol{Z}[1] \oplus \boldsymbol{K}[i][1])\big).$$

After the last round, the cipher xors $\bar{\boldsymbol{k}}[R]$ into the final state. Hence, for plaintext $\boldsymbol{x}$ and ciphertext $\boldsymbol{y}$, $\boldsymbol{y} = \bar{\boldsymbol{k}}[R] \oplus \boldsymbol{Z}$ at the output boundary. Appendix E gives the constructions of S-box ANFs, the detailed linear-layer PK, T, PL, PR, the complete definitions of $\mathcal{L}_{\mathsf{u}}$, its inverse, and the associated Fwd and Bwd mappings used by the two-round meet-in-the-middle constraints.

5.2 The Ballet Block Cipher

The block cipher Ballet is a software-friendly ARX construction designed by Cui, Wang, and Fan [21]. In uBlockith, we instantiate the CTR-style PRG with the 74-round Ballet with a 256-bit key and block size. The concrete construction of Ballet, including its key schedule and round function, is specified in the original design paper [21]. Recent cryptanalytic study for Ballet can be found in [13].

5.3 The ZKP Constraint System of uBlockith

The uBlockith and uBlockith-EM constraint systems are evaluated over $\mathbb{F}_{2^{256}}$ and have degree $d = 3$. Their witnesses are arranged so that every non-linear computation receives degree-1 inputs. For uBlockith, the witness contains the initial 256-bit secret key, the first 64 bits of each of the 24 subsequent round keys, and the 256-bit state after every second encryption round except the final boundary. Its length is therefore $\ell = 256 + 24 \cdot 64 + (24/2 - 1) \cdot 256 = 4\,608$ bits. For uBlockith-EM, the key expansion is public, so the witness instead begins with the secret input block and has length $\ell = 256 + (24/2 - 1) \cdot 256 = 3\,072$ bits. Here and below, $R = 24$ remains the number of physical cipher rounds. The index $r \in [0, R/2)$ instead labels a block of two consecutive rounds; thus the encryption trace has $R/2 = 12$ such blocks and only $R/2 - 1 = 11$ internal block-boundary states are committed.

Key-expansion constraints. For each key-expansion round i, write the committed round key as $\langle \bar{\boldsymbol{k}}[i] \rangle = (\langle \boldsymbol{k}_0^{(i)} \rangle, \langle \boldsymbol{k}_1^{(i)} \rangle, \langle \boldsymbol{k}_2^{(i)} \rangle, \langle \boldsymbol{k}_3^{(i)} \rangle)$. The witness contains only the first 64-bit double word $\langle \boldsymbol{k}_0^{(i+1)} \rangle$ of the next round key. Indeed, for two consecutive rounds i and $i + 1$, the first round derives

$$\begin{aligned} (\langle \boldsymbol{t}_0 \rangle, \langle \boldsymbol{t}_1 \rangle) &\leftarrow \mathsf{PK}(\langle \boldsymbol{k}_0^{(i)} \rangle, \langle \boldsymbol{k}_1^{(i)} \rangle), \\ \langle \boldsymbol{t}_2 \rangle^3 &\leftarrow \langle \boldsymbol{k}_2^{(i)} \rangle + \mathsf{SubDWord}(\langle \boldsymbol{t}_0 \rangle + \mathsf{RC}[i]), \\ \langle \boldsymbol{t}_3 \rangle &\leftarrow \langle \boldsymbol{k}_3^{(i)} \rangle + \mathsf{T}(\langle \boldsymbol{t}_1 \rangle). \end{aligned}$$

It appends the 64-bit degree-3 equality constraint $\langle \boldsymbol{c}_{\mathsf{ke}}[i] \rangle^3 = \langle \boldsymbol{k}_0^{(i+1)} \rangle + \langle \boldsymbol{t}_2 \rangle^3$. If this vector is zero, then the witness-provided word is exactly the non-linear output $\boldsymbol{t}_2$, and the complete next key is recovered as $\langle \bar{\boldsymbol{k}}[i+1] \rangle = (\langle \boldsymbol{k}_0^{(i+1)} \rangle, \langle \boldsymbol{t}_3 \rangle, \langle \boldsymbol{t}_1 \rangle, \langle \boldsymbol{t}_0 \rangle)$. The second round repeats the same computation from this reconstructed degree-1 key: it obtains

$$\langle \boldsymbol{u}_2 \rangle^3 \leftarrow \langle \boldsymbol{k}_2^{(i+1)} \rangle + \mathsf{SubDWord}\big(\mathsf{PK}_0(\langle \boldsymbol{k}_0^{(i+1)} \rangle, \langle \boldsymbol{k}_1^{(i+1)} \rangle) + \mathsf{RC}[i+1]\big)$$

where PK_0 denotes the first 64-bit output of PK, and appends $\langle \boldsymbol{k}_0^{(i+2)} \rangle + \langle \boldsymbol{u}_2 \rangle^3$. Hence, only the first 64 bits of every new key need be committed: the other three double words are public linear functions of the preceding complete key. The 24 rounds yield $24 \cdot 64 = 1536$ degree-3 constraints.

Meet-in-the-middle encryption constraints. Rather than commit to every round state, the construction treats two consecutive physical rounds as one constraint block. More precisely, for $r \in [0, R/2)$, let $i = 2r$: block r checks physical rounds i and $i + 1$. Write $\langle \boldsymbol{X}[r] \rangle = (\langle \boldsymbol{X}[r][0] \rangle, \langle \boldsymbol{X}[r][1] \rangle)$ for the degree-1 committed state before physical round $2r$, and write $\langle \boldsymbol{X}[r+1] \rangle = (\langle \boldsymbol{X}[r+1][0] \rangle, \langle \boldsymbol{X}[r+1][1] \rangle)$ for the claimed state after physical round $2r+1$. Thus, $R = 24$ yields twelve two-round blocks, rather than twelve cipher rounds. Using the mappings defined in Appendix E, a forward computation through round i gives $(\langle \boldsymbol{a}_0 \rangle^3, \langle \boldsymbol{a}_1 \rangle^3) \leftarrow \mathsf{Fwd}_i\big(\langle \boldsymbol{X}[r] \rangle, \langle \bar{\boldsymbol{k}}[i] \rangle, \langle \bar{\boldsymbol{k}}[i+1] \rangle\big)$, where the two 128-bit vectors

are the claimed inputs to the two SubQWord calls in round $i+1$. Independently, the backward computation inverts the public linear layer of round $i+1$ from $\langle \boldsymbol{X}[r+1] \rangle$ and applies the inverse S-box: $(\langle \boldsymbol{b}_0 \rangle^3, \langle \boldsymbol{b}_1 \rangle^3) \leftarrow \mathsf{Bwd}_{i+1}\big(\langle \boldsymbol{X}[r+1] \rangle\big)$. It obtains a second degree-3 representation of those same two S-box inputs. The two constraints for this two-round example are therefore

$$\langle \boldsymbol{c}_{\mathsf{enc},0}[r] \rangle^3 = \langle \boldsymbol{a}_0 \rangle^3 + \langle \boldsymbol{b}_0 \rangle^3, \qquad \langle \boldsymbol{c}_{\mathsf{enc},1}[r] \rangle^3 = \langle \boldsymbol{a}_1 \rangle^3 + \langle \boldsymbol{b}_1 \rangle^3.$$

They contain $2 \cdot 128 = 256$ bit constraints and vanish exactly when the forward and backward computations meet. The witness stores only $\langle \boldsymbol{X}[1] \rangle, \ldots, \langle \boldsymbol{X}[R/2-1] \rangle$, namely the eleven internal block boundaries. Its endpoints $\langle \boldsymbol{X}[0] \rangle$ and $\langle \boldsymbol{X}[R/2] \rangle$ are recovered from the public input and output (and, in the EM case, the committed secret input). For the last pair, the post-round state is recovered from the public output by xoring $\bar{\boldsymbol{k}}[R]$, so no final state is stored in the witness.

uBlockith.DeriveConstraints. Algorithm 3 is the common derivation procedure for the prover and verifier. Its input is the degree-1 VOLE commitment to the extended witness together with the public key $(\boldsymbol{x}, \boldsymbol{y})$. In the non-EM case, it parses $\langle \boldsymbol{w} \rangle = \langle \bar{\boldsymbol{k}}[0] \rangle \| \langle \boldsymbol{k}_0^{(1)} \rangle \| \cdots \| \langle \boldsymbol{k}_0^{(R)} \rangle \| \langle \boldsymbol{X}[1] \rangle \| \cdots \| \langle \boldsymbol{X}[R/2-1] \rangle$; in the EM case, it parses $\langle \boldsymbol{w} \rangle = \langle \boldsymbol{X}[0] \rangle \| \langle \boldsymbol{X}[1] \rangle \| \cdots \| \langle \boldsymbol{X}[R/2-1] \rangle$. The output is a degree-3 vector of 4608 constraints for uBlockith or 3072 constraints for uBlockith-EM. Its angle-bracket operations denote the respective computations on the two views of the same VOLE commitment, as in Section 2. This algorithm is the uBlockith/uBlockith-EM instantiation of the $\mathsf{DeriveConstraints}_{\mathsf{param}}$ interface of Equation (2) in Section 2; Appendix B invokes it inside ProveConstraints and VerifyConstraints before the ZKHash step. Every call to Fwd and Bwd below uses the mappings of Appendix E.

The algorithm first reconstructs all round keys: it checks each witness-provided 64-bit word against the cubic SubDWord output and then obtains the remaining words by linear operations. It next fixes the two encryption boundaries. For uBlockith, the public input and output give $\langle \boldsymbol{X}[0] \rangle = \langle \boldsymbol{x} \rangle, \langle \boldsymbol{X}[R/2] \rangle = \langle \boldsymbol{y} \rangle + \langle \bar{\boldsymbol{k}}[R] \rangle$. For uBlockith-EM, $\boldsymbol{x}$ is the public block-cipher key, while the first witness block $\langle \boldsymbol{X}[0] \rangle$ is the secret encryption input. Since the public value is $\boldsymbol{y} = \mathrm{uBlock}_{\boldsymbol{x}}(\boldsymbol{X}[0]) \oplus \boldsymbol{X}[0]$, the final post-round state is $\langle \boldsymbol{X}[R/2] \rangle = \langle \boldsymbol{X}[0] \rangle + \boldsymbol{y} + \langle \bar{\boldsymbol{k}}[R] \rangle$. The eleven internal block-boundary states are then read from the witness. For each $r \in [0, R/2)$, the procedure sets $i = 2r$ and appends the two meet-in-the-middle constraints for physical rounds i and $i+1$. Here, Fwd_i takes a state before round i and returns the two committed inputs of the S-boxes in round $i+1$, whereas Bwd_{i+1} reverses the following linear layer and inverse S-box from the claimed state after round $i+1$; their complete definitions are given in Appendix E. Thus, the returned vector has $24 \cdot 64 + 12 \cdot 2 \cdot 128 = 4608$ entries for uBlockith and $(R/2) \cdot 2 \cdot 128 = 12 \cdot 256 = 3072$ entries for uBlockith-EM.

5.4 Construction of Signature Schemes

In the non-EM scheme, the public key is $(\boldsymbol{x}, \boldsymbol{y})$, the signer knows the secret key $\boldsymbol{k}$, and the proved relation is $\boldsymbol{y} = \mathrm{uBlock}_{\boldsymbol{k}}(\boldsymbol{x})$. In the EM scheme, $\boldsymbol{x}$ is the

Algorithm 3: uBlockith.DeriveConstraints from VOLE commitments

Input: Degree-1 commitments $\langle \boldsymbol{w} \rangle$ to the extended witness, public key $(\boldsymbol{x}, \boldsymbol{y})$, and $\mathsf{mode} \in \{\mathsf{non\text{-}EM}, \mathsf{EM}\}$
Output: Degree-3 constraint vector $\langle \boldsymbol{o} \rangle$ of length 4 608 (non-EM) or 3 072 (EM)
Initialize $\langle \boldsymbol{o} \rangle \leftarrow ()$;
if $\mathsf{mode} = \mathsf{non\text{-}EM}$ **then**
  Parse $\langle \bar{\boldsymbol{k}}[0] \rangle$ and the 24 successive 64-bit words $\langle \boldsymbol{k}_0^{(i+1)} \rangle$ from $\langle \boldsymbol{w} \rangle$;
  for $i \leftarrow 0$ **to** $R-1$ **do**
    $(\langle \boldsymbol{t}_0 \rangle, \langle \boldsymbol{t}_1 \rangle) \leftarrow \mathsf{PK}(\langle \boldsymbol{k}_0^{(i)} \rangle, \langle \boldsymbol{k}_1^{(i)} \rangle)$;
    $\langle \boldsymbol{t}_2 \rangle^3 \leftarrow \langle \boldsymbol{k}_2^{(i)} \rangle + \mathsf{SubDWord}(\langle \boldsymbol{t}_0 \rangle + \mathsf{RC}[i])$;
    $\langle \boldsymbol{t}_3 \rangle \leftarrow \langle \boldsymbol{k}_3^{(i)} \rangle + \mathsf{T}(\langle \boldsymbol{t}_1 \rangle)$;
    Append $\langle \boldsymbol{k}_0^{(i+1)} \rangle + \langle \boldsymbol{t}_2 \rangle^3$ to $\langle \boldsymbol{o} \rangle$;
    $\langle \bar{\boldsymbol{k}}[i+1] \rangle \leftarrow (\langle \boldsymbol{k}_0^{(i+1)} \rangle, \langle \boldsymbol{t}_3 \rangle, \langle \boldsymbol{t}_1 \rangle, \langle \boldsymbol{t}_0 \rangle)$;
  end
  $\langle \boldsymbol{X}[0] \rangle \leftarrow \langle \boldsymbol{x} \rangle$;
  $\langle \boldsymbol{X}[R/2] \rangle \leftarrow \langle \boldsymbol{y} \rangle + \langle \bar{\boldsymbol{k}}[R] \rangle$;
else
  Compute the public expanded keys $\langle \bar{\boldsymbol{k}}[0] \rangle, \ldots, \langle \bar{\boldsymbol{k}}[R] \rangle$ from $\boldsymbol{x}$;
  $\langle \boldsymbol{X}[0] \rangle \leftarrow$ the initial 256 committed bits of $\langle \boldsymbol{w} \rangle$;
  $\langle \boldsymbol{X}[R/2] \rangle \leftarrow \langle \boldsymbol{X}[0] \rangle + \boldsymbol{y} + \langle \bar{\boldsymbol{k}}[R] \rangle$;
end
Parse the eleven internal 256-bit block-boundary states as $\langle \boldsymbol{X}[1] \rangle, \ldots, \langle \boldsymbol{X}[R/2-1] \rangle$;
for *two-round block* $r \leftarrow 0$ **to** $R/2-1$ **do**
  $i \leftarrow 2r$;
  $(\langle \boldsymbol{a}_0 \rangle^3, \langle \boldsymbol{a}_1 \rangle^3) \leftarrow \mathsf{Fwd}_i(\langle \boldsymbol{X}[r] \rangle, \langle \bar{\boldsymbol{k}}[i] \rangle, \langle \bar{\boldsymbol{k}}[i+1] \rangle)$;
  $(\langle \boldsymbol{b}_0 \rangle^3, \langle \boldsymbol{b}_1 \rangle^3) \leftarrow \mathsf{Bwd}_{i+1}(\langle \boldsymbol{X}[r+1] \rangle)$;
  Append $\langle \boldsymbol{a}_0 \rangle^3 + \langle \boldsymbol{b}_0 \rangle^3$ and $\langle \boldsymbol{a}_1 \rangle^3 + \langle \boldsymbol{b}_1 \rangle^3$ to $\langle \boldsymbol{o} \rangle$;
end
return $\langle \boldsymbol{o} \rangle$;

public block-cipher key, $\boldsymbol{k}$ is the secret input block, and the proved relation is $\boldsymbol{y} = \mathrm{uBlock}_{\boldsymbol{x}}(\boldsymbol{k}) \oplus \boldsymbol{k}$. The derivation in Algorithm 3 proves the corresponding relation under VOLE commitments. All other procedures follow the FAEST construction summarized in Section 2.

The ZKP parameters are compared with the corresponding FAEST variants [9] in Table 2. Both uBlockith and uBlockith-EM use degree 3, a proof field $\mathbb{F}_{2^{256}}$, and $B = 16$. Their total extended witness lengths are respectively $4\,608 + 3 \cdot 256 + 16 = 5\,392$ bits and $3\,072 + 3 \cdot 256 + 16 = 3\,856$ bits. As in FAEST-256, each has a small (s) and a fast (f) profile, following the parameters of FAEST. We use the Ballet-CTR PRG as introduced in Section 3; all remaining construction details follow FAEST [9]. Note that the actual signature scheme requires one additional constraint due to the key-space reduction, which is described in Appendix B.

Table 2. ZKP and extended-witness lengths for uBlockith and FAEST-256

Scheme	ZKP witness ℓ (bits)	Degree d	Extended witness $\hat{\ell}$ (bits)	Constraints C
uBlockith	4 608	3	$4\,608 + 3 \cdot 256 + 16 = 5\,392$	4 608
uBlockith-EM	3 072	3	$3\,072 + 3 \cdot 256 + 16 = 3\,856$	3 072
FAEST-256	3 104	3	$3\,104 + 3 \cdot 256 + 16 = 3\,888$	777
FAEST-EM-256	2 688	2	$2\,688 + 2 \cdot 256 + 16 = 3\,216$	673

Theorem 2 (Security of uBlockith and uBlockith-EM). *Under Assumption 1,* uBlockith *and* uBlockith-EM *are* EUF-CMA*-secure in the ROM against probabilistic polynomial-time adversaries and in the quantum-accessible ROM against quantum polynomial-time adversaries. The formal direct ROM and QROM bounds are stated in Appendix C.2.*

6 Implementation and Evaluation

This section presents the implementation details and reports the evaluation results of our signature schemes. Our codes are built upon the FAEST library[2] and will be open-source soon.

6.1 Hardware and Software

We evaluate the optimized implementations on two Intel platforms and one Hygon platform, and pin each measurement to one CPU core.

We use two Intel platforms to report the effect of the availability of native SM4 ISA extensions. The first Intel platform is an Intel Xeon 6982P-C processor with up to 3.90 GHz with 14 GiB memory, running Linux 6.12.57 and GCC 14.2.0. This platform does not provide the native SM4 ISA extensions. The second Intel platform is an Intel Xeon 6992E+C processor with up to 3.20 GHz clock frequency and 377 GiB memory, running CentOS Stream 10 with Linux kernel `6.12.0-243.el10.x86_64` and GCC 14.4.1. It provides native SM4 ISAs.

We also use a dual-socket Hygon C86-4G (OPN:7490) server at about 3.00 GHz frequency with 2.2 TiB memory, running Alibaba Cloud Linux 3 with the kernel `6.6.47-004.ant8.x86_64` and GCC 10.2.1. The Hygon platform also offers native SM4 ISA extensions.

Although both backends implement the standard SM4 key schedule and encryption routines, they use different vendor-specific x86 instruction interfaces. Intel uses the VEX-encoded `VSM4KEY4` and `VSM4RNDS4` instructions, each performing four SM4 rounds on independent XMM/YMM lanes, which enables packed multi-key and multi-counter kernels. In contrast, Hygon uses the SSE-style CIS-SM4 `SM4RK` and `SM4RND` interface on XMM states with immediate-selected rounds.

[2] https://github.com/faest-sign

6.2 Acceleration Techniques

The main implementation costs of our schemes arise from PRG evaluation, hashing and Fiat–Shamir challenge generation, binary-field arithmetic, and the opening of the batch all-but-one vector commitment. We make corresponding optimizations for acceleration.

Reusing work for specific PRG modes in the VOLEitH framework. PRGs in our signature schemes are used in a CTR style: they are produced by encrypting the combination of IV, tweak, and successive public counter blocks under a secret seed key. Further, they are invoked in three modes: the seed expansion from a λ-bit seed to a 2λ-bit seed, the leaf commitment from a λ-bit seed to a 2λ-bit commitment output, and the VOLE conversion from a λ-bit seed to a $\hat{\ell}$-bit extended witness.

We make the following two observations. First, the leaf commitment and VOLE conversion invoke the PRG with the same seed keys. We therefore retain a seed's expanded round keys after the first use instead of deriving them again. Second, when the protocol processes one fixed sub-GGM tree in the batch all-but-one vector commitment, many PRG calls have different seed keys but the same public IV, tree label (tweak), and sequence of counter blocks, *i.e.*, the same plaintext. The implementation then builds this shared counter sequence once, then encrypts it under several keys in parallel. This fixed-plaintext batching is especially useful for the VOLE conversion.

SM4-CTR PRG without native ISAs. We design SM4-CTR-style PRGs that combine existing instruction sets and scheme properties on earlier Intel CPUs without native SM4 ISAs. In particular, we use SM4-NI, which emulates the S-boxes of SM4 via AES-NI.

For 2λ-bit outputs, the open-source SM4-NI backend[3] only provides a 4-block same-key encryption-routine implementation, which does not match our targets. To reduce the overhead, we make the following two observations. Any 4 sets of 2-block key expansion routine can be seen as 2 independent sets of 4-block key expansion routine. Further, the key expansion and encryption routines of SM4 are similar. We then modify the open-source 4-block same-key encryption-routine backend to implement a 4-block key-expansion-routine backend, as well as a 4-block different-key encryption-routine backend.

For $\hat{\ell}$-bit outputs, we exploit the open-source 16-block same-key SM4-NI kernel whenever profitable. For $\hat{\ell} = 276$ (17 full blocks plus 4 bytes), we first precompute the fixed-tweak counter sequence, then evaluate the first 16 counters with one 16-block call, generate the remaining counters with one extra 4-block call, and finally keep the first 20 bytes of the tail buffer. For $\hat{\ell} = 162$ (10 full blocks plus 2 bytes), experiments show that forcing a 16-block call would reduce efficiency. We therefore reuse the 4-block different-key encryption-routine backend and obtain the final outputs by truncation.

[3] https://git.fh-aachen.de/tb3838s/linux/-/blob/c291c9cfd76a8fb92ef3d66567e507009236ce90/arch/x86/crypto/sm4-aesni-avx2-asm_64.S

SM4-CTR PRG with native ISAs. The native Intel instructions operate on one XMM lane or two independent 128-bit lanes in a YMM register. They do not directly provide the complete multi-key PRG operations required by the protocol. We therefore build software scheduling kernels for a single key and one block, a single key and two counter blocks, two keys and one common counter, four keys and one common counter, four keys and two common counters, and four keys and six common counters. They are compositions of native instructions.

For four-key processing, the key schedule directly produces a transposed, pair-packed YMM representation of the round keys. The round keys for keys zero and one form one YMM value, while those for keys two and three form another. This layout is consumed directly by the encryption kernels and avoids first writing four scalar round-key arrays and subsequently repacking them. For a 2λ-bit output, four keys and two public counter blocks are processed together using four YMM state registers. For long outputs, the four-key–six-counter kernel keeps twelve YMM states live and applies each group of four round keys to all independent states before proceeding to the next round group. This interleaving exposes sufficient independent work to hide much of the latency of the native SM4 round instruction.

The non-EM instances require 276 output bytes, corresponding to 18 encrypted counter blocks followed by truncation to 276 bytes. We generate these blocks using three four-key–six-counter calls. The EM instances require 162 bytes; they use two such calls and retain the first 162 bytes of the resulting twelve-block stream. The same packed round-key schedule is also reused when the protocol requests both a 2λ-bit leaf output and a long VOLE-conversion output for the same four seed keys.

The Hygon backend follows the same protocol-level batching principle: counter blocks shared by several keys are constructed once and independent keys are processed together.

Ballet-CTR PRG for uBlockith. For uBlockith, Ballet serves as the CTR PRG. We process four independent Ballet seed keys together: their round keys are stored in a lane-packed layout and are consumed directly by a four-key encryption routine. The long PRG calls used by VOLE conversion use the same shared-counter batching strategy described above.

Field arithmetic and SM4th constraints. The proof system also performs many multiplications in binary finite fields; these are part of checking the ZKP constraints rather than part of ordinary block-cipher encryption. On x86, `PCLMULQDQ` directly accelerates the carry-less polynomial multiplications from which these field products are formed. For uBlockith, a 256-bit field multiplication is split into smaller pieces using the two-way Karatsuba method [29].

For SM4th, recall that an SM4 byte over $\mathbb{F}_{2^8}$ must first be embedded into the larger proof field $\mathbb{F}_{2^\lambda}$ via the mapping $\phi(\cdot)$ before a constraint can be checked. Observing that the embedding constants, their Frobenius conjugates (successive squarings), and the constants used by the norm construction are all fixed by the scheme, we precompute them once. Consequently, reconstructing a byte, obtaining its square, and expanding a norm coordinate become fixed linear com-

binations rather than repeated finite-field setup computations. When several such products occur together, they are accumulated before a common field embedding, further reducing the workload.

6.3 Evaluation

Tables 3, 4, and 5 report single-core signing and verification latencies averaged over 1 000 runs, in milliseconds and million CPU cycles. The size columns give the public-key size, signature size, and their combined size. Within each scheme family, the small (s) and fast (f) variants differ only in their VOLE parameters: the fast profile reduces latency at the cost of a larger signature. The EM variants have a shorter witness and fewer encryption S-box constraints, which generally reduces both signature size and signing time. Our schemes are compared with ML-DSA and SLH-DSA at OpenSSL[4] and FAEST[5]. For the latter, we modify the implementation to make the non-EM variants use PRG-based leaf commitments, leading to smaller signature sizes than the previously reported figures.

Table 3. Comparison of SM4th with other schemes on two different Intel platforms with or without native SM4 ISAs, targeting NIST's L1 security

Scheme	Native SM4 ISAs?	Runtimes in ms		Runtimes in Mcyc.		Sizes in B		
		Sign	Verify	Sign	Verify	pk	Sig	pk + Sig
SM4th-s	No	20.158	19.678	76.133	74.332	32	5 056	5 088
	Yes	5.297	4.208	16.896	13.425			
SM4th-EM-s	No	19.995	19.731	75.545	74.556	32	3 818	3 850
	Yes	3.948	3.254	12.589	10.380			
SM4th-f	No	2.965	2.844	11.205	10.744	32	6 724	6 756
	Yes	0.884	0.693	2.818	2.212			
SM4th-EM-f	No	2.723	2.707	10.299	10.236	32	4 932	4 964
	Yes	0.529	0.454	1.688	1.449			
FAEST-128s (PRG-based com.)	No	2.555	2.218	8.696	7.297	32	4 330	4 362
	Yes	4.114	3.142	12.457	9.211			
FAEST-EM-128s	No	2.327	2.125	7.839	6.943	32	3 906	3 938
	Yes	3.631	3.002	10.917	8.771			
FAEST-128f (PRG-based com.)	No	0.308	0.261	1.164	0.987	32	5 668	5 700
	Yes	0.487	0.388	1.553	1.238			
FAEST-EM-128f	No	0.285	0.242	1.078	0.916	32	5 060	5 092
	Yes	0.448	0.362	1.429	1.154			
ML-DSA-44	No	0.448	0.090	1.696	0.341	1 312	2 420	3 732
	Yes	0.532	0.106	1.696	0.339			
SLH-DSA-SHAKE-128s	No	685.558	0.626	2594.984	2.369	32	7 856	7 888
	Yes	749.425	0.724	2388.819	2.307			
SLH-DSA-SHAKE-128f	No	32.746	1.904	123.952	7.208	32	17 088	17 120
	Yes	36.085	2.266	115.018	7.223			

[4] https://github.com/openssl/openssl
[5] https://github.com/faest-sign/faest-arch-opt

Table 4. Comparison of SM4th with other schemes on Hygon CPUs, targeting the NIST's L1 security

Scheme	Runtimes in ms		Runtimes in Mcyc.		Sizes in B		
	Sign	Verify	Sign	Verify	pk	Sig	pk + Sig
SM4th-s	11.546	8.960	34.615	26.866	32	5 056	5 088
SM4th-EM-s	10.139	8.345	30.402	25.025	32	3 818	3 850
SM4th-f	1.990	1.626	5.966	4.877	32	6 724	6 756
SM4th-EM-f	1.339	1.184	4.015	3.550	32	4 932	4 964
FAEST-128s (PRG-based leaf com)	5.046	4.146	14.245	11.410	32	4 330	4 362
FAEST-EM-128s	4.692	3.989	13.192	10.940	32	3 906	3 938
FAEST-128f (PRG-based leaf com)	0.699	0.554	2.096	1.661	32	5 668	5 700
FAEST-EM-128f	0.633	0.526	1.897	1.576	32	5 060	5 092
ML-DSA-44	0.857	0.163	2.572	0.488	1 312	2 420	3 732
SLH-DSA-SHAKE-128s	1150.957	1.078	3452.872	3.235	32	7 856	7 888
SLH-DSA-SHAKE-128f	55.417	3.422	166.250	10.266	32	17 088	17 120

For the two Intel tables, the *No* rows are measured on the Intel Xeon 6982P-C platform without native SM4 ISA extensions, whereas the *Yes* rows are measured on the Intel Xeon 6992E+C platform with the native ISA backends enabled.

Comparison with standardized schemes at NIST L1. SM4th-EM-s has a combined public-key and signature size of 3 850 B, close to the 3 732 B of ML-DSA-44. Its signing and verification times are, however, about 50× (resp., 5.3×) and 200× (resp., 25.6×) those of ML-DSA-44 on Intel platforms without (resp., with) native SM4 ISAs, respectively. This comparison reflects a different design trade-off: ML-DSA relies on a structured module-lattice assumption, whereas SM4th relies only on symmetric primitives and does not use a structured algebraic assumption.

SM4th and SLH-DSA both have conservative symmetric-primitive-based design goals. Relative to SLH-DSA-SHAKE-128f, SM4th-f, on Intel platforms without native SM4 ISAs, signs about 10× faster (3.948 ms versus 32.746 ms) and has a 2.5× smaller signature (6 724 B versus 17 088 B). Its verification is slower, but by less than a factor of two (2.844 ms versus 1.904 ms). On the native-ISA Intel platform, SM4th-f signs about 40× faster than SLH-DSA-SHAKE-128f (0.884 ms versus 36.085 ms), while its verification is about 3.3× faster (0.693 ms versus 2.266 ms). The EM fast variant further decreases the signature to 4 932 B and signs in 0.529 ms.

Comparison with FAEST on different platforms. On the Intel Xeon 6982P-C platform without native SM4 extensions, all SM4th variants are roughly 10× slower than FAEST in both signing and verification. On Intel Xeon 6992E+C, the matching SM4th and FAEST variants are within a factor of 1.8 in times. In particular, SM4th-EM-s signs in 3.948 ms, compared with 3.631 ms for FAEST-EM-128s, while their combined sizes are 3 850 B and 3 938 B, respectively.

Table 4 shows the evaluation results on Hygon. With the Hygon CIS-SM4 backend, the SM4th–FAEST gap narrows to about 2–3×. On this platform, SM4th-f also outperforms SLH-DSA-SHAKE-128f in both operations: its sign-

ing time and verification time are about 42× and 2× smaller, respectively. For the small variants, the signing-time gap between SM4th-EM-s and ML-DSA-44 narrows to about 12×, although the verification-time gap remains about 50× in the reported measurements.

NIST L5 results. Table 5 reports the corresponding NIST L5 results. The uBlockith-EM-s variant has a 2.6× larger combined public-key and signature size than ML-DSA-87 (19 120 B versus 7 219 B). Its signing and verification times are about 40× and 150× those of ML-DSA-87, respectively.

Table 5. Comparison of signatures on Intel Xeon 6982P-C, targeting NIST's L5 security

Scheme	Runtimes in ms		Runtimes in Mcyc.		Sizes in B		
	Sign	Verify	Sign	Verify	pk	Sig	pk + Sig
uBlockith-s	42.165	41.193	154.175	154.794	64	24 144	24 208
uBlockith-EM-s	35.176	33.643	128.175	126.525	64	19 056	19 120
uBlockith-f	5.529	5.277	19.971	19.920	64	31 556	31 620
uBlockith-EM-f	4.550	4.314	16.441	16.284	64	25 028	25 092
FAEST-256s (PRG-based com.)	12.628	12.538	42.884	42.394	48	19 992	20 040
FAEST-EM-256s	12.659	12.193	42.984	41.098	64	17 984	18 048
FAEST-256f (PRG-based com.)	1.989	1.851	6.941	6.291	48	25 524	25 572
FAEST-EM-256f	1.897	1.818	6.566	6.147	64	23 476	23 540
ML-DSA-87	0.833	0.218	3.153	0.825	2 592	4 627	7 219
SLH-DSA-SHAKE-256s	1033.268	1.463	3910.813	5.539	64	29 792	29 856
SLH-DSA-SHAKE-256f	108.873	2.886	412.095	10.924	64	49 856	49 920

Relative to FAEST-256, the uBlockith variants are about 3–4× slower in both signing and verification. Their combined sizes are 6%–24% larger for the matching small or fast profiles. For example, uBlockith-EM-s signs in 35.176 ms and has a combined size of 19 120 B, compared with 12.659 ms and 18 048 B for FAEST-EM-256s. This remaining overhead is due to the longer witness and the degree-3 uBlock constraint system, rather than a different VOLE parameter set.

The uBlockith-f variants trade faster signing for slower verification relative to SLH-DSA-SHAKE-256f. They provide a 20× signing-time advantage, whereas verification is about 2–3× slower. They also have smaller combined public-key and signature sizes: 31 620 B and 25 092 B, respectively, versus 49 920 B for SLH-DSA-SHAKE-256f.

7 Conclusion and Future Work

We present SM4th and uBlockith, two families of VOLE-based post-quantum signatures using selected Chinese block ciphers. The constraint systems are adapted to the four-word recurrence of SM4 and to the cubic Boolean S-box of uBlock.

We also develope ISA-aware implementations, including native SM4 support on recent Intel processors and the dedicated CIS-SM4 backend for Hygon ones.

Our experiments reveal substantial platform dependence in the end-to-end cost of these instantiations. On the Intel platform using software SM4 paths, SM4th requires approximately 10× the runtime of the corresponding FAEST-128 profiles; on another Intel platform with native SM4 support, it is within a factor of 1.8 of FAEST-128. On the Hygon platform with native CIS-SM4 support, SM4th requires approximately 2–3× the runtime of FAEST-128. For uBlockith, the corresponding overhead is approximately 3–4× of FAEST-256. The schemes also provide competitive communication costs in selected parameter sets. In particular, SM4th-EM-s has a combined public-key and signature size of 3 850 bytes, compared with 3 938 bytes for FAEST-EM-128s and the 3 732 bytes for ML-DSA-44.

Overall, our results establish the feasibility and efficiency of instantiating VOLE-in-the-Head signatures with the selected Chinese primitives. Future work may include designing ZKP-friendly OWFs and software-friendly PRGs, and evaluating them on additional processor architectures.

References

1. Information security technology—SM4 block cipher algorithm (2016)
2. Aardal, M.A., Adj, G., Aranha, D.F., Basso, A., Martínez, I.A.C., Chávez-Saab, J., Santos, M.C.R., Dartois, P., Feo, L.D., Duparc, M., Eriksen, J.K., Fouotsa, T.B., Filho, D.L.G., Hess, B., Kohel, D., Leroux, A., Longa, P., Maino, L., Meyer, M., Nakagawa, K., Onuki, H., Panny, L., Patranabis, S., Petit, C., Pope, G., Reijnders, K., Robert, D., Henríquez, F.R., Schaeffler, S., Wesolowski, B.: SQIsign: Algorithm specifications and supporting documentation, version 2.0.1, https://sqisign.org/spec/sqisign-20250707.pdf
3. Adj, G., Aragon, N., Barbero, S., Bardet, M., Bellini, E., Bidoux, L., Chi-Domínguez, J.J., Dyseryn, V., Esser, A., Feneuil, T., Gaborit, P., Neveu, R., Rivain, M., Rivera-Zamarripa, L., Sanna, C., Tillich, J.P., Verbel, J., Zweydinger, F.: Mirath signature scheme: Algorithm specifications and supporting documentation, version 2.0, https://pqc-mirath.org/assets/downloads/mirath_v2.0.1.pdf
4. Albrecht, M.R., Rechberger, C., Schneider, T., Tiessen, T., Zohner, M.: Ciphers for MPC and FHE. In: EUROCRYPT (2015)
5. Aumasson, J.P., Bernstein, D.J., Beullens, W., Dobraunig, C., Eichlseder, M., Fluhrer, S., Gazdag, S.L., Hülsing, A., Kampanakis, P., Kölbl, S., Lange, T., Lauridsen, M.M., Mendel, F., Niederhagen, R., Rechberger, C., Rijneveld, J., Schwabe, P., Westerbaan, B.: SPHINCS+: Submission to the NIST post-quantum project, v.3.1, https://sphincs.org/data/sphincs+-r3.1-specification.pdf
6. Bai, S., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schwabe, P., Seiler, G., Stehlé, D.: CRYSTALS-Dilithium algorithm specifications and supporting documentation (version 3.1), https://pq-crystals.org/dilithium/data/dilithium-specification-round3-20210208.pdf
7. Baldi, M., Barenghi, A., Battagliola, M., Bitzer, S., Gianvecchio, M., Karl, P., Manganiello, F., Pavoni, A., Pelosi, G., Pintore, F., Santini, P., Schupp, J., Signorini, E., Slaughter, F., Wachter-Zeh, A., Weger, V.: CROSS: Codes and restricted

objects signature scheme, algorithm specifications and supporting documentation, version 2.2, https://www.cross-crypto.com/CROSS_Specification_v2.2.pdf
8. Baldi, M., Barenghi, A., Beckwith, L., Biasse, J.F., Chou, T., Esser, A., Gaj, K., Karl, P., Mohajerani, K., Pelosi, G., Persichetti, E., Saarinen, M.J.O., Santini, P., Wallace, R., Zweydinger, F.: LESS: Linear equivalence signature scheme (2025-02-07), https://www.less-project.com/LESS-2025-02-07.pdf
9. Baum, C., Beullens, W., Braun, L., de Saint Guilhem, C.D., Klooß, M., Majenz, C., Mukherjee, S., Orsini, E., Ramacher, S., Rechberger, C., Roy, L., Scholl, P.: FAEST v2: Algorithm specifications version 2.0, https://faest.info/faest-spec-v2.0.pdf
10. Baum, C., Beullens, W., Mukherjee, S., Orsini, E., Ramacher, S., Rechberger, C., Roy, L., Scholl, P.: One tree to rule them all: Optimizing GGM trees and owfs for post-quantum signatures. In: International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT). pp. 463–493. Springer, Berlin (2024)
11. Baum, C., Braun, L., de Saint Guilhem, C.D., Klooß, M., Orsini, E., Roy, L., Scholl, P.: Publicly verifiable zero-knowledge and post-quantum signatures from VOLE-in-the-head. In: Annual Cryptology Conference (CRYPTO). pp. 581–615. Springer, Berlin (2023)
12. Baum, C., de Saint Guilhem, C.D., Kales, D., Orsini, E., Scholl, P., Zaverucha, G.: Banquet: Short and fast signatures from AES. In: PKC (2021)
13. Bellini, E., Bellini, G., Piccoli, A.D., Gallone, M., Gerault, D., Huang, Y.J., Huynh, P., Onger, M., Pelizzola, S., Visconti, A.: More brisés in Ballet: Extending differential and linear cryptanalysis (2026), https://eprint.iacr.org/2026/501
14. Benadjila, R., Feneuil, T., Rivain, M.: MQ on my mind: Post-quantum signatures from the non-structured multivariate quadratic problem. In: EuroS&P (2024)
15. Bettaieb, S., Bidoux, L., Dyseryn, V., Esser, A., Gaborit, P., Kulkarni, M., Palumbi, M.: PERK: compact signature scheme based on a new variant of the permuted kernel problem. Design, Codes and Cryptography **92**(8), 2131–2157 (2024)
16. Beullens, W., Campos, F., Celi, S., Hess, B., Kannwischer, M.J.: MAYO: Round 2 version, https://pqmayo.org/assets/specs/mayo-round2.pdf
17. Bidoux, L., Chi-Domínguez, J., Feneuil, T., Gaborit, P., Joux, A., Rivain, M., Vinçotte, A.: RYDE: A digital signature scheme based on rank-syndrome-decoding problem with mpcith paradigm (2023)
18. Bos, J.W., Bronchain, O., Ducas, L., Fehr, S., Huang, Y.H., Pornin, T., Postlethwaite, E.W., Prest, T., Pulles, L.N., van Woerden, W.: HAWK: Version 1.1, https://hawk-sign.info/hawk-spec.pdf
19. Chase, M., Derler, D., Goldfeder, S., Orlandi, C., Ramacher, S., Rechberger, C., Slamanig, D., Zaverucha, G.: Post-quantum zero-knowledge and signatures from symmetric-key primitives. In: CCS (2017)
20. Cui, H., Liu, H., Yan, D., Yang, K., Yu, Y., Zhang, K.: ReSolveD: Shorter signatures from regular syndrome decoding and VOLE-in-the-head. In: PKC (2024)
21. Cui, T., Wang, M., Fan, Y., Hu, K., Fu, Y., Huang, L.: The design of Ballet block cipher (in Chinese), https://sfjs.cacrnet.org.cn/site/term/list_76_1.html
22. Feneuil, T., Rivain, M.: Threshold computation in the head: Improved framework for post-quantum signatures and zero-knowledge arguments. Journal of Cryptology **38**(3), 28 (2025)
23. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Annual International Cryptology Conference (CRYPTO). pp. 186–194. Springer, Berlin (1986)

24. Fouque, P.A., Lyubashevsky, V., Hoffstein, J., Pornin, T., Kirchner, P., Prest, T., Seiler, G., Whyte, W., Ricosset, T., Zhang, Z.: Falcon: Fast-Fourier lattice-based compact signatures over NTRU, https://falcon-sign.info/falcon.pdf
25. Gao, Y., Wang, L., Tian, L., Hu, Y., Zhang, Y., Yan, Y., Wu, Q.: Fast software implementation of the block cipher uBlock algorithm. Journal of National University of Defense Technology **46**(6), 96–106 (2024)
26. Gong, Z., Guo, H., Chen, C., Zhang, Y., Chen, J., Guan, Z.: FBS-uBlock: Flexible bit slicing optimization method of uBlock algorithm. Journal of Software **36**(10), 4827–4845 (2025). https://doi.org/10.13328/j.cnki.jos.007316
27. Institute of Commercial Cryptography Standards: Call for Proposals for the Next-generation Public-Key Cryptographic Algorithms, https://www.niccs.org.cn/niccs/Notice/pc/content/content_1975896137741635584.html
28. Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Zero-knowledge from secure multiparty computation. In: STOC (2007)
29. Karatsuba, A., Ofman, Y.: Multiplication of many-digital numbers by automatic computers. Proceedings of the USSR Academy of Sciences **145**, 293–294 (1962), translated in Physics-Doklady, 7 (1963), 595–596
30. KpqC Research Group: KpqC Algorithms, https://www.kpqc.or.kr/contents/03_exhibit/sub_03.html
31. Liu, F., Ji, W., Hu, L., Ding, J., Lv, S., Pyshkin, A., Weinmann, R.: Analysis of the SMS4 block cipher. In: Information Security and Privacy - Australasian Conference (ACISP). pp. 472–492. Springer, Berlin (2007)
32. Melchor, C.A., Bettaieb, S., Bidoux, L., Feneuil, T., Gaborit, P., Gama, N., Gueron, S., Howe, J., Hülsing, A., Joseph, D., Joux, A., Kulkarni, M., Persichetti, E., Randrianarisoa, T.H., Rivain, M., Yue, D.: The syndrome decoding in the head (SD-in-the-head) signature scheme: Algorithm specifications and supporting documentation, version 2.0, https://sdith.org/docs/sdith-v2.0.pdf
33. National Institute of Standards and Technology: Module-lattice-based digital signature standard (2024), https://doi.org/10.6028/NIST.FIPS.204
34. National Institute of Standards and Technology: Stateless hash-based digital signature standard (2024), https://doi.org/10.6028/NIST.FIPS.205
35. National Institute of Standards and Technology (NIST): Post-quantum cryptography: Digital signature schemes, https://csrc.nist.gov/projects/pqc-dig-sig
36. Saarinen, M.J.O.: SM4NI, https://github.com/mjosaarinen/sm4ni
37. Wang, C., Cui, J., Li, M., Wang, M.: Improved integral cryptanalysis on block cipher uBlock. Journal of Electronics & Information Technology **46**(5), 2149–2158 (2024)
38. Wu, W., Zhang, L., Zheng, Y., Li, C.: The design of uBlock block cipher (in Chinese), https://sfjs.cacrnet.org.cn/site/term/list_76_1.html
39. Yang, K., Sarkar, P., Weng, C., Wang, X.: Quicksilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field. In: ACM SIGSAC Conference on Computer and Communications Security (CCS). pp. 2986–3001. ACM, New York (2021)

A Parameters and VOLE Generation

This appendix specifies the parameters used by the proposed signature schemes and summarizes how the batch all-but-one vector commitment (BAVC) produces the VOLE correlations used in Section 2. The parameter sets in this appendix are part of the scheme specification.

A.1 Main Parameters

Each parameter set contains the main parameters

$$\mathsf{param} = (\lambda, \tau, w_{\mathsf{grind}}, T_{\mathsf{open}}, B, \mathsf{OWF}, d, \ell).$$

Their meanings are as follows.

- λ is both the security parameter and the extension degree of the proof field $\mathbb{F}_{2^\lambda}$.
- τ is the number of small VOLE instances, equivalently the number of BAVC sub-trees.
- w_{grind} is the grinding parameter. The signer accepts an opening challenge derived by the Fiat–Shamir challenge only when its final w_{grind} bits are zero.
- T_{open} is the maximum permitted number of internal seeds in a BAVC opening. The signer retries the final Fiat–Shamir challenge when the selected opening exceeds this threshold.
- B is the statistical-security padding used by the VOLE consistency check, which is used to unify the VOLE correlations of multiple GGM sub-trees. All parameter sets in this paper use $B = 16$.
- OWF is the one-way function whose preimage relation is proved by the signature.
- d is the maximum algebraic degree of the constraint system.
- ℓ is the bit length of the OWF witness before the additional masking, which is determined by the OWF and proving method.

The concrete values of the main parameters are given in Table 6.

Table 6. Main parameters of the proposed signature schemes

Scheme	OWF	λ	ℓ	τ	w_{grind}	T_{open}	B	d
SM4th-s	$\mathsf{SM4}_{\boldsymbol{k}}(\boldsymbol{x})$	128	1808	11	7	102	16	3
SM4th-f	$\mathsf{SM4}_{\boldsymbol{k}}(\boldsymbol{x})$	128	1808	16	8	110	16	3
SM4th-EM-s	$\mathsf{SM4}_{\boldsymbol{x}}(\boldsymbol{k}) \oplus \boldsymbol{k}$	128	1024	11	7	103	16	2
SM4th-EM-f	$\mathsf{SM4}_{\boldsymbol{x}}(\boldsymbol{k}) \oplus \boldsymbol{k}$	128	1024	16	8	112	16	2
uBlockith-s	$\mathsf{uBlock}_{\boldsymbol{k}}(\boldsymbol{x})$	256	4608	22	6	245	16	3
uBlockith-f	$\mathsf{uBlock}_{\boldsymbol{k}}(\boldsymbol{x})$	256	4608	32	8	246	16	3
uBlockith-EM-s	$\mathsf{uBlock}_{\boldsymbol{x}}(\boldsymbol{k}) \oplus \boldsymbol{k}$	256	3072	22	6	218	16	3
uBlockith-EM-f	$\mathsf{uBlock}_{\boldsymbol{x}}(\boldsymbol{k}) \oplus \boldsymbol{k}$	256	3072	32	8	234	16	3

A.2 Hash Functions and Output Lengths

This subsection specifies the logical hash/XOF interfaces used by every parameter set. Inputs are concatenated in the fixed, length-delimited order specified by

Table 7. Cryptographic hash interfaces and output lengths

Function	Domain		Range and use
H_{msg}	$\{0,1\}^*$		$\{0,1\}^{2\lambda}$; binds the public key and message
H_{seed}	$\boldsymbol{k}\|\mu\|\rho$ $\in \{0,1\}^{4\lambda}$		$\{0,1\}^{\lambda+\lambda_{\mathsf{iv}}}$, parsed as $(r, \mathsf{iv}_{\mathsf{pre}}) \in \{0,1\}^{\lambda} \times \{0,1\}^{\lambda_{\mathsf{iv}}}$
H_{iv}	$\{0,1\}^{\lambda_{\mathsf{iv}}}$		$\{0,1\}^{\lambda_{\mathsf{iv}}}$; derives the PRG IV
H_1	$\{0,1\}^*$		$\{0,1\}^{2\lambda}$; aggregates BAVC leaf commitments and sub-tree commitments into com
$H_2^{(1)}$	$\{0,1\}^*$		$\{0,1\}^{5\lambda+64}$; samples the VOLE-consistency-check seed
$H_2^{(2)}$	$\{0,1\}^*$		$\{0,1\}^{3\lambda+64}$; samples the constraint-batching seed
$H_2^{(3)}$	$\{0,1\}^*$		$\{0,1\}^{\lambda}$; derives the final Fiat–Shamir challenge

param, and the domain labels msg, seed, iv, bavc, and $1, 2, 3$ identify the seven interfaces in Table 7.

All parameter sets have $\lambda_{\mathsf{iv}} = \lambda$. The unkeyed interfaces in Table 7 are modeled as random oracles in the ROM and quantum-accessible random oracles in the QROM. In contrast, H_{seed} is a keyed derivation: $\boldsymbol{k}$ is its PRF key, and the function

$$(\mu, \rho) \longmapsto H_{\mathsf{seed}}(\boldsymbol{k}\|\mu\|\rho)$$

is modeled as a PRF with range $\{0,1\}^{\lambda+\lambda_{\mathsf{iv}}}$. The procedure VOLEHash is a linear universal hash used in the VOLE consistency check, rather than a cryptographic hash/XOF interface.

A.3 VOLE and BAVC Parameters

The VOLE parameters, derived from the main parameters, are $\mathsf{param}_{\mathsf{VOLE}} = (\tau_1, \tau_0, k, L, T_{\mathsf{open}})$. The BAVC creates τ_1 small VOLE instances over $\mathbb{F}_{2^k}$ and τ_0 small VOLE instances over $\mathbb{F}_{2^{k-1}}$. Thus, we have $\tau = \tau_0 + \tau_1$. The first type contributes k challenge bits per instance and the second type contributes $k-1$ challenge bits per instance. Since the final w_{grind} bits of the opening challenge are fixed by grinding, the parameters satisfy $\lambda - w_{\mathsf{grind}} = \tau_1 k + \tau_0(k-1)$. The total number of BAVC leaves is $L = \tau_1 2^k + \tau_0 2^{k-1}$. Here, T_{open} is shared with the main parameter set: it bounds the number of internal seeds released in an accepted BAVC opening. The concrete values of the VOLE parameters are listed in Table 8.

A.4 VOLE Generation and Extended Witnesses

For each small instance $h \in [0, \tau)$, the BAVC commits to a seed tree and leaves one challenge-selected leaf unopened. The unopened leaf determines a sub-challenge Δ_h in either $\mathbb{F}_{2^k}$ or $\mathbb{F}_{2^{k-1}}$. Before the small instances are combined,

Table 8. VOLE and BAVC parameters of the proposed signature schemes

Scheme	τ_1	τ_0	k	L	T_{open}
SM4th-s	0	11	12	22528	102
SM4th-f	8	8	8	3072	110
SM4th-EM-s	0	11	12	22528	103
SM4th-EM-f	8	8	8	3072	112
uBlockith-s	8	14	12	61440	245
uBlockith-f	24	8	8	7168	246
uBlockith-EM-s	8	14	12	61440	218
uBlockith-EM-f	24	8	8	7168	234

the h-th tree produces its own mask $\boldsymbol{u}_h \in \{0,1\}^{\hat{\ell}}$. The procedure $\mathsf{VOLECommit}$ sets $\boldsymbol{u} = \boldsymbol{u}_0$ and includes the correction vectors

$$\mathsf{aux}_{\mathsf{com}} = (\boldsymbol{e}_1, \ldots, \boldsymbol{e}_{\tau-1}), \qquad \boldsymbol{e}_h = \boldsymbol{u} \oplus \boldsymbol{u}_h \quad \text{for } h \in [1, \tau).$$

Here, $\mathsf{aux}_{\mathsf{com}} \in \{0,1\}^{(\tau-1)\hat{\ell}}$ is the transmitted data that reconciles the independently generated sub-tree masks to the same vector $\boldsymbol{u}$. After the verifier applies these corrections, each small VOLE correlation satisfies $\boldsymbol{q}_h[i] = \boldsymbol{u}[i]\Delta_h + \boldsymbol{v}_h[i]$ for every $i \in [0, \hat{\ell})$. The bit representations of the Δ_h values are concatenated to form the first $\lambda - w_{\mathsf{grind}}$ bits of the final opening challenge; the remaining w_{grind} bits are required to be zero. Likewise, the corrected small-field verifier values and prover masks are concatenated and viewed as elements of $\mathbb{F}_{2^\lambda}$, yielding the large-field relation used in Section 2, *i.e.*, $\boldsymbol{q}[i] = \boldsymbol{u}[i]\Delta + \boldsymbol{v}[i]$.

More precisely, on input a root seed $r \in \{0,1\}^\lambda$, an IV $\mathsf{iv} \in \{0,1\}^{\lambda_{\mathsf{iv}}}$, and $\hat{\ell}$, $\mathsf{VOLECommit}$ returns $(\mathsf{com}, \mathsf{st}, \mathsf{aux}_{\mathsf{com}}, \boldsymbol{u}, \boldsymbol{V})$, where $\mathsf{com} \in \{0,1\}^{2\lambda}$ is the BAVC commitment, st is private opening state that is not serialized, $\mathsf{aux}_{\mathsf{com}} \in \{0,1\}^{(\tau-1)\hat{\ell}}$ is the correction-vector string defined above, and $\boldsymbol{V} \in \{0,1\}^{\hat{\ell}\times\lambda}$ is the concatenation of the VOLE verifier sub-values, including the w_{grind} zero columns.

The VOLE length is larger than the OWF witness length. In particular, the consistency check releases a linear hash of $\boldsymbol{u}$ of length $\lambda + B$, and the degree-d ZKHash check releases $d-1$ additional field elements. The corresponding mask coordinates are included in the VOLE correlation, giving

$$\hat{\ell} = \ell + (\lambda + B) + (d-1)\lambda = \ell + d\lambda + B.$$

The first ℓ mask bits hide the OWF witness through $\boldsymbol{d} = \boldsymbol{w} \oplus \boldsymbol{u}[0, \ell)$. The next $\lambda + B$ coordinates mask the value disclosed by the VOLE consistency check, and the final $(d-1)\lambda$ coordinates mask the non-constant coefficients disclosed by ZKHash.

The BAVC uses a CTR-style PRG to expand tree seeds and leaf seeds. Given a seed s, an IV iv, a 32-bit tweak twk, and a requested output length m, the

PRG first forms its j-th block-cipher input as

$$\mathsf{CtrBlock}(\mathsf{iv}, \mathsf{twk}, j) = \mathsf{AddLower}(\mathsf{AddUpper}(\mathsf{iv}, \mathsf{twk}), j), \qquad j = 0, 1, 2, \ldots.$$

The function AddUpper adds twk modulo 2^{32} to the designated domain-separation word of iv. The function AddLower adds j modulo 2^{32} to a distinct counter word. The PRG output is then

$$\mathsf{PRG}(s, \mathsf{iv}, \mathsf{twk}; m) = \mathsf{Trunc}_m\big(E_s(\mathsf{CtrBlock}(\mathsf{iv}, \mathsf{twk}, 0)) \| E_s(\mathsf{CtrBlock}(\mathsf{iv}, \mathsf{twk}, 1)) \| \cdots\big).$$

Consequently, the same $(s, \mathsf{iv}, \mathsf{twk})$ produces a deterministic stream, the tweak separates PRG calls for different BAVC roles, and j is the internal CTR block counter. In all proposed parameter sets, $\mathsf{iv}_{\mathsf{pre}} = \lambda_{\mathsf{iv}} = \lambda$. The signer transmits $\mathsf{iv}_{\mathsf{pre}} \in \{0,1\}^{\lambda_{\mathsf{iv}}}$ and derives $\mathsf{iv} = H_{\mathsf{iv}}(\mathsf{iv}_{\mathsf{pre}})$.

For the signature format in Appendix B, the correction vectors in $\mathsf{aux}_{\mathsf{com}}$ contribute $(\tau - 1)\hat{\ell}$ bits. The remaining proof values contribute $|\widetilde{\boldsymbol{u}}| + |\boldsymbol{d}| + \sum_{j=1}^{d-1} |\widetilde{a}_j| = (\lambda + B) + \ell + (d-1)\lambda = \hat{\ell}$ bits. Moreover, BAVC.Open returns $\mathsf{decom}_I \in \{0,1\}^{(2\tau + T_{\mathsf{open}})\lambda}$ or $\perp$. Consequently, an accepted signature has length

$$\underbrace{\tau\hat{\ell}}_{\mathsf{aux}_{\mathsf{com}}, \widetilde{\boldsymbol{u}}, \boldsymbol{d}, \widetilde{a}_1, \ldots, \widetilde{a}_{d-1}} + \underbrace{(2\tau + T_{\mathsf{open}})\lambda}_{\mathsf{decom}_I} + \underbrace{\lambda}_{c_3} + \underbrace{\lambda_{\mathsf{iv}}}_{\mathsf{iv}_{\mathsf{pre}}} + \underbrace{32}_{\mathsf{ctr}}.$$

The final 32 bits encode the grinding counter, not the internal PRG block counter.

B High-Level Signature Specification

This appendix gives a common high-level description of the three algorithms of every proposed parameter set: KeyGen generates an OWF key pair, Sign produces a Fiat–Shamir signature for a message, and Verify checks such a signature. The public description param fixes the OWF E, the mode $\mathsf{mode} \in \{\mathsf{non\text{-}EM}, \mathsf{EM}\}$, all parameter values in Appendix A, and hash/XOF functions used below. All inputs to a hash function are concatenated in the fixed, length-delimited order specified by param.

B.1 Key Generation

Algorithm 4 describes the key-generation procedure. It takes the main parameter param, the mode flag, and the block cipher E as inputs, and outputs the public key and secret key.

For both modes, $\boldsymbol{k}$ is the secret value and $\boldsymbol{x}$ is the public input to the OWF. The public input is sampled uniformly from $\{0,1\}^{\lambda}$. The secret value is sampled uniformly from the restricted key space $\mathcal{K}_\lambda = \boldsymbol{k} \in \{0,1\}^{\lambda} : \boldsymbol{k}[0] \cdot \boldsymbol{k}[1] = 0\}$, namely, $\boldsymbol{k}$ is rejected when $\boldsymbol{k}[0] \wedge \boldsymbol{k}[1] = 1$. Then, the public output $\boldsymbol{y}$ is computed according to the mode. In the non-EM mode, $\boldsymbol{k}$ is the secret block-cipher key

Algorithm 4: KeyGen(param)

```
Input: A parameter set param with mode mode ∈ {non-EM, EM} and block
  cipher E
Output: Secret key sk and public key pk
Sample x ← {0,1}^λ;
repeat
  | Sample k ← {0,1}^λ;
until k[0] · k[1] = 0;
if mode = non-EM then
  | y ← E_k(x);
else
  | y ← E_x(k) ⊕ k;
end
sk ← (x, k) and pk ← (x, y);
return (sk, pk);
```

and $\boldsymbol{x}$ is the public plaintext. In the EM mode, $\boldsymbol{x}$ is the public block-cipher key and $\boldsymbol{k}$ is the secret plaintext. The public key is $\mathsf{pk} = (\boldsymbol{x}, \boldsymbol{y})$ and the secret key is $\mathsf{sk} = (\boldsymbol{x}, \boldsymbol{k})$.

The restriction is enforced by appending the single degree-2 Boolean constraint $\boldsymbol{k}[0] \cdot \boldsymbol{k}[1] = 0$ to the concrete constraint system. For readability, the main text omits this auxiliary constraint from the cipher-specific derivations and displayed constraint counts.

This restriction is important for the QROM security proof. The excluded one-quarter of the key space provides the key-domain slack required by the FAEST-style reduction to embed a false statement, thereby converting the knowledge-soundness reduction into an ordinary soundness reduction. Accordingly, the 4ρ factor in the ROM and QROM bounds applies under this restricted key-generation procedure. Since $|\mathcal{K}_\lambda| = 3 \cdot 2^{\lambda-2}$, the restriction loses only $\log_2(4/3) \approx 0.415$ bits of key-space entropy, namely less than 0.5 bit of security.

B.2 Signing

Algorithm 5 describes the signing procedure. It takes a secret key, a public key, a message, and a parameter set, and returns a signature. It first samples fresh signing randomness ρ and computes $\mu = H_{\mathsf{msg}}(\mathsf{pk} \| m)$. The function H_{msg} binds the proof to the ordered pair (pk, m). The hash/XOF interfaces used in this algorithm are listed in Table 7. Next, $H_{\mathsf{seed}}(\boldsymbol{k} \| \mu \| \rho)$ derives a root seed $r \in \{0,1\}^\lambda$ and a pre-IV $\mathsf{iv}_{\mathsf{pre}} \in \{0,1\}^{\lambda_{\mathsf{iv}}}$. The former seeds the BAVC, while the latter is the public value from which H_{iv} deterministically derives the IV iv used by its PRG.

The signer then sets $\hat{\ell} = \ell + d\lambda + B$ and invokes

$$(\mathsf{com}, \mathsf{st}, \mathsf{aux}_{\mathsf{com}}, \boldsymbol{u}, \boldsymbol{V}) \leftarrow \mathsf{VOLECommit}(r, \mathsf{iv}, \hat{\ell}; \mathsf{param}).$$

The inputs are the BAVC root seed, IV, target VOLE length, and public parameters. Its outputs $\mathsf{com} \in \{0,1\}^{2\lambda}$, $\mathsf{aux}_{\mathsf{com}} \in \{0,1\}^{(\tau-1)\hat{\ell}}$, $\boldsymbol{u} \in \{0,1\}^{\hat{\ell}}$, and

$\boldsymbol{V} \in \{0,1\}^{\hat{\ell}\times\lambda}$; the private state st is retained only to later generate an opening. In particular, $\mathsf{aux}_{\mathsf{com}}$ contains the correction vectors that reconcile the masks produced by the τ BAVC sub-trees to the common mask $\boldsymbol{u}$. The hash

$$c_1 = H_2^{(1)}(\mu \| \mathsf{com} \| \mathsf{aux}_{\mathsf{com}} \| \mathsf{iv})$$

binds this commitment phase to the message and is used as the public seed of the linear map $\mathsf{VOLEHash}$. On input c_1 and either $\boldsymbol{u}$ or $\boldsymbol{V}$, $\mathsf{VOLEHash}$ outputs, respectively, $\widetilde{\boldsymbol{u}} \in \{0,1\}^{\lambda+B}$ or $\widetilde{\boldsymbol{V}} \in \{0,1\}^{(\lambda+B)\times\lambda}$. This compact linear check is needed because the VOLE correlation is assembled from multiple BAVC sub-trees: the correction vectors align their masks, and $\mathsf{VOLEHash}$ checks their consistency using only $\lambda + B$ rows instead of communicating the full $\hat{\ell}$-row relation.

Next, $\mathsf{ExtendWitness}(\mathsf{sk}, \mathsf{pk}; \mathsf{param})$ expands the secret-key preimage into the ℓ-bit OWF witness $\boldsymbol{w}$. Its output contains precisely the cipher-specific intermediate values required by the constraint system, including norm witnesses when the concrete instantiation uses them. The signer masks this witness as $\boldsymbol{d} = \boldsymbol{w} \oplus \boldsymbol{u}[0,\ell)$ and computes

$$c_2 = H_2^{(2)}(c_1 \| \widetilde{\boldsymbol{u}} \| \widetilde{\boldsymbol{V}} \| \boldsymbol{d}).$$

Thus, c_2 binds both the compressed VOLE-consistency check and the masked witness before the constraint proof is formed. The procedure $\mathsf{ProveConstraints}(\boldsymbol{w}, \boldsymbol{u}, \boldsymbol{V}, \mathsf{pk}, c_2; \mathsf{param})$ first selects the prescribed mask coordinates from $(\boldsymbol{u}, \boldsymbol{V})$ to form its degree-1 VOLE view of the witness. It then invokes the concrete $\mathsf{DeriveConstraints}_{\mathsf{param}}$ interface of Equation (2) in Section 2, instantiated by Sections 4 and 5, to obtain the degree-d constraint vector $\langle \boldsymbol{o} \rangle$. Finally, it invokes ZKHash with c_2 to batch all entries of $\langle \boldsymbol{o} \rangle$ into one degree-d polynomial. This batching avoids transmitting one response per constraint: it returns only the d field elements $(\widetilde{a}_0, \ldots, \widetilde{a}_{d-1})$, of which the signer sends $\widetilde{a}_1, \ldots, \widetilde{a}_{d-1}$ and the verifier reconstructs $\widetilde{a}_0$.

Finally, the signer sets the 32-bit grinding counter ctr to zero and computes

$$c_3 = H_2^{(3)}(c_2 \| \widetilde{a}_0 \| \cdots \| \widetilde{a}_{d-1} \| \mathsf{ctr}).$$

The counter is incremented until the final w_{grind} bits of c_3 are zero and the opening is short enough. The non-grinding prefix of c_3 is input to $\mathsf{DecodeChallenge}$, which returns the tuple I containing one leaf index for each BAVC sub-tree. On input $(\mathsf{st}, I; \mathsf{param})$, $\mathsf{BAVC.Open}$ returns the fixed-length opening $\mathsf{decom}_I \in \{0,1\}^{(2\tau+T_{\mathsf{open}})\lambda}$, or $\perp$ if more than T_{open} auxiliary seeds would be needed. The accepted output tuple in Algorithm 5 is the signature.

B.3 Verification

Algorithm 6 describes the verification procedure. It takes a public key, a message, a purported signature, and a parameter set. It first parses the signature as

$$\sigma = (\mathsf{aux}_{\mathsf{com}}, \widetilde{\boldsymbol{u}}, \boldsymbol{d}, \widetilde{a}_1, \ldots, \widetilde{a}_{d-1}, \mathsf{decom}_I, c_3, \mathsf{iv}_{\mathsf{pre}}, \mathsf{ctr})$$

Algorithm 5: $\mathsf{Sign}(\mathsf{sk}, \mathsf{pk}, m; \mathsf{param})$

INPUT: Secret key $\mathsf{sk} = (\boldsymbol{x}, \boldsymbol{k})$, public key pk, message m, and parameter set param
OUTPUT: Signature σ
$\mu \leftarrow H_{\mathsf{msg}}(\mathsf{pk} \| m)$ and sample signing randomness ρ;
$(r, \mathsf{iv}_{\mathsf{pre}}) \leftarrow H_{\mathsf{seed}}(\boldsymbol{k} \| \mu \| \rho)$ and $\mathsf{iv} \leftarrow H_{\mathsf{iv}}(\mathsf{iv}_{\mathsf{pre}})$;
$\hat{\ell} \leftarrow \ell + d\lambda + B$;
$(\mathsf{com}, \mathsf{st}, \mathsf{aux}_{\mathsf{com}}, \boldsymbol{u}, \boldsymbol{V}) \leftarrow \mathsf{VOLECommit}(r, \mathsf{iv}, \hat{\ell}; \mathsf{param})$;
$c_1 \leftarrow H_2^{(1)}(\mu \| \mathsf{com} \| \mathsf{aux}_{\mathsf{com}} \| \mathsf{iv})$;
$\widetilde{\boldsymbol{u}} \leftarrow \mathsf{VOLEHash}(c_1, \boldsymbol{u})$ and $\widetilde{\boldsymbol{V}} \leftarrow \mathsf{VOLEHash}(c_1, \boldsymbol{V})$;
$\boldsymbol{w} \leftarrow \mathsf{ExtendWitness}(\mathsf{sk}, \mathsf{pk}; \mathsf{param})$ and $\boldsymbol{d} \leftarrow \boldsymbol{w} \oplus \boldsymbol{u}[0, \ell)$;
$c_2 \leftarrow H_2^{(2)}(c_1 \| \widetilde{\boldsymbol{u}} \| \widetilde{\boldsymbol{V}} \| \boldsymbol{d})$;
$(\widetilde{a}_0, \ldots, \widetilde{a}_{d-1}) \leftarrow \mathsf{ProveConstraints}(\boldsymbol{w}, \boldsymbol{u}, \boldsymbol{V}, \mathsf{pk}, c_2; \mathsf{param})$;
$\mathsf{ctr} \leftarrow 0$;
$\mathsf{decom}_I \leftarrow \perp$;
repeat
  $c_3 \leftarrow H_2^{(3)}(c_2 \| \widetilde{a}_0 \| \cdots \| \widetilde{a}_{d-1} \| \mathsf{ctr})$;
  if $c_3[\lambda - w_{\mathsf{grind}}, \lambda) = 0^{w_{\mathsf{grind}}}$ **then**
    $I \leftarrow \mathsf{DecodeChallenge}(c_3[0, \lambda - w_{\mathsf{grind}}); \mathsf{param})$;
    $\mathsf{decom}_I \leftarrow \mathsf{BAVC.Open}(\mathsf{st}, I; \mathsf{param})$;
    if $\mathsf{decom}_I = \perp$ **then**
      $\mathsf{ctr} \leftarrow \mathsf{ctr} + 1$;
    end
  else
    $\mathsf{ctr} \leftarrow \mathsf{ctr} + 1$;
  end
until $\mathsf{decom}_I \neq \perp$;
return $\sigma = (\mathsf{aux}_{\mathsf{com}}, \widetilde{\boldsymbol{u}}, \boldsymbol{d}, \widetilde{a}_1, \ldots, \widetilde{a}_{d-1}, \mathsf{decom}_I, c_3, \mathsf{iv}_{\mathsf{pre}}, \mathsf{ctr})$;

and rejects unless the component lengths are, respectively, $(\tau - 1)\hat{\ell}$, $\lambda + B$, ℓ, $(d-1)\lambda$, $(2\tau + T_{\mathsf{open}})\lambda$, λ, λ_{iv}, and 32 bits. It recomputes $\mu = H_{\mathsf{msg}}(\mathsf{pk} \| m)$ and $\mathsf{iv} = H_{\mathsf{iv}}(\mathsf{iv}_{\mathsf{pre}})$, and rejects unless the final w_{grind} bits of c_3 are zero. It then invokes $\mathsf{DecodeChallenge}$ on the remaining bits of c_3. This procedure returns the tuple I of challenge-selected BAVC leaves, or $\perp$ if the bit string does not describe a valid tuple.

On input $(\mathsf{decom}_I, I, \mathsf{aux}_{\mathsf{com}}, \mathsf{iv}; \mathsf{param})$, $\mathsf{BAVC.Reconstruct}$ checks the BAVC opening, regenerates the BAVC commitment $\mathsf{com} \in \{0,1\}^{2\lambda}$, and outputs the verifier's VOLE view $\boldsymbol{Q} \in \{0,1\}^{\hat{\ell} \times \lambda}$. During reconstruction, $\mathsf{aux}_{\mathsf{com}}$ supplies the correction vectors that convert the individual sub-tree masks to the common mask represented by the signer's $\boldsymbol{u}$. The verifier therefore obtains the same combined VOLE relation without receiving $\boldsymbol{u}$ or $\boldsymbol{V}$ in full. It recomputes

$$c_1 = H_2^{(1)}(\mu \| \mathsf{com} \| \mathsf{aux}_{\mathsf{com}} \| \mathsf{iv}) \quad \text{and} \quad \widetilde{\boldsymbol{Q}} = \mathsf{VOLEHash}(c_1, \boldsymbol{Q}).$$

Algorithm 6: $\mathsf{Verify}(\mathsf{pk}, m, \sigma; \mathsf{param})$

Input: Public key pk, message m, signature σ, and parameter set param
Output: true or false
Parse σ as $(\mathsf{aux}_{\mathsf{com}}, \widetilde{\boldsymbol{u}}, \boldsymbol{d}, \widetilde{a}_1, \ldots, \widetilde{a}_{d-1}, \mathsf{decom}_I, c_3, \mathsf{iv}_{\mathsf{pre}}, \mathsf{ctr})$; reject a malformed signature;
$\mu \leftarrow H_{\mathsf{msg}}(\mathsf{pk} \| m)$ and $\mathsf{iv} \leftarrow H_{\mathsf{iv}}(\mathsf{iv}_{\mathsf{pre}})$;
if $c_3[\lambda - w_{\mathsf{grind}}, \lambda) \neq 0^{w_{\mathsf{grind}}}$ **then**
  return false;
end
$I \leftarrow \mathsf{DecodeChallenge}(c_3[0, \lambda - w_{\mathsf{grind}}); \mathsf{param})$;
$(\mathsf{com}, \boldsymbol{Q}) \leftarrow \mathsf{BAVC.Reconstruct}(\mathsf{decom}_I, I, \mathsf{aux}_{\mathsf{com}}, \mathsf{iv}; \mathsf{param})$;
if $(\mathsf{com}, \boldsymbol{Q}) = \perp$ **then**
  return false;
end
$c_1 \leftarrow H_2^{(1)}(\mu \| \mathsf{com} \| \mathsf{aux}_{\mathsf{com}} \| \mathsf{iv})$ and $\widetilde{\boldsymbol{Q}} \leftarrow \mathsf{VOLEHash}(c_1, \boldsymbol{Q})$;
Parse $c_3 = (\delta_0, \ldots, \delta_{\lambda-1})$ and compute $\widetilde{\boldsymbol{V}}' \leftarrow \widetilde{\boldsymbol{Q}} \oplus (\delta_0 \widetilde{\boldsymbol{u}} \| \cdots \| \delta_{\lambda-1} \widetilde{\boldsymbol{u}})$;
$c_2 \leftarrow H_2^{(2)}(c_1 \| \widetilde{\boldsymbol{u}} \| \widetilde{\boldsymbol{V}}' \| \boldsymbol{d})$;
$\widetilde{a}_0 \leftarrow \mathsf{VerifyConstraints}(\boldsymbol{d}, \boldsymbol{Q}, \mathsf{pk}, c_2, c_3, \widetilde{a}_1, \ldots, \widetilde{a}_{d-1}; \mathsf{param})$;
if $\widetilde{a}_0 = \perp$ **then**
  return false;
end
$c_3' \leftarrow H_2^{(3)}(c_2 \| \widetilde{a}_0 \| \cdots \| \widetilde{a}_{d-1} \| \mathsf{ctr})$;
return $(c_3' = c_3)$;

The latter is the $(\lambda + B) \times \lambda$ compressed verifier value corresponding to the signer's $\widetilde{\boldsymbol{V}}$. Writing $c_3 = (\delta_0, \ldots, \delta_{\lambda-1})$, the verifier uses the same challenge bits and the transmitted $\widetilde{\boldsymbol{u}}$ to compute

$$\widetilde{\boldsymbol{V}}' = \widetilde{\boldsymbol{Q}} \oplus (\delta_0 \widetilde{\boldsymbol{u}} \| \cdots \| \delta_{\lambda-1} \widetilde{\boldsymbol{u}}).$$

This is the verifier-side reconstruction of the compressed value used by the signer; the linearity of $\mathsf{VOLEHash}$ makes it possible to perform the consistency check without opening every BAVC leaf. The verifier then derives the same constraint challenge $c_2 = H_2^{(2)}(c_1 \| \widetilde{\boldsymbol{u}} \| \widetilde{\boldsymbol{V}}' \| \boldsymbol{d})$.

Finally, $\mathsf{VerifyConstraints}(\boldsymbol{d}, \boldsymbol{Q}, \mathsf{pk}, c_2, c_3, \widetilde{a}_1, \ldots, \widetilde{a}_{d-1}; \mathsf{param})$ evaluates the concrete constraint system using the masked witness and the reconstructed VOLE view. It first forms the verifier's degree-1 VOLE view and invokes the same $\mathsf{DeriveConstraints}_{\mathsf{param}}$ interface from Equation (2) in Section 2, with the concrete instantiation selected in Section 4 or Section 5. It then performs the verifier side of the same ZKHash batch check as $\mathsf{ProveConstraints}$ and returns the omitted coefficient $\widetilde{a}_0 \in \mathbb{F}_{2^\lambda}$, or $\perp$ if the constraints, format, or VOLE relation are inconsistent. Thus the verifier checks all OWF constraints through a constant-size polynomial response rather than one communication item per constraint. It computes $c_3' = H_2^{(3)}(c_2 \| \widetilde{a}_0 \| \cdots \| \widetilde{a}_{d-1} \| \mathsf{ctr})$ and accepts exactly when $c_3' = c_3$.

C Security

C.1 Security Assumptions

This appendix states the cryptographic assumptions used by the security theorems for all parameter sets in Appendix A. The assumption separates properties of the OWF, PRG, leaf commitment, and hash functions, so that a change to any concrete instantiation can be checked against the corresponding security condition.

Assumption 1 (Security assumptions for SM4th and uBlockith). *Let Π be any parameter set in Appendix A. Let E be its block cipher, let* PRG *be its IV- and tweak-based pseudorandom generator, and let $\mathcal{H}$ denote the unkeyed hash/XOF interfaces used for BAVC aggregation and the Fiat–Shamir transcript. We assume the following holds.*

1. ***OWF security.*** *The relation proved by Π is one-way: for non-EM parameter sets, $F_x(k) = E_k(x)$ is one-way; for EM parameter sets, $F_x(k) = E_x(k) \oplus k$ is one-way. The block cipher E also satisfies the PRP security required by the corresponding reduction.*
2. ***PRG security.*** *For every permitted output length and every admissible IV and tweak, $\mathsf{PRG}(s, \mathsf{iv}, \mathsf{twk}; m)$ is pseudorandom when s is uniformly sampled from the seed space.*
3. ***Almost-injective PRG leaf commitments.*** *The leaf commitment maps a seed $s \in \{0,1\}^\lambda$ to $\mathsf{PRG}(s, \mathsf{iv}, \mathsf{twk}; 2\lambda)$. For a fixed IV, an adversary outputs Q_{out} pairs (twk_j, y_j). Its almost-injectivity advantage is the probability that, for some j, there are distinct $s, s' \in \{0,1\}^\lambda$ s.t.*

$$\mathsf{PRG}(s, \mathsf{iv}, \mathsf{twk}_j; 2\lambda) = y_j = \mathsf{PRG}(s', \mathsf{iv}, \mathsf{twk}_j; 2\lambda).$$

   We denote this probability by $\mathsf{AdvInj}_{\mathcal{B}}^{\mathsf{PRG}_{2\lambda}}[Q_{\mathsf{out}}]$. This is not a standard consequence of PRG or PRP security and is therefore stated separately. We follow the ideal-cipher heuristic used in FAEST [9], which gives the estimate

$$\mathsf{AdvInj}_{\mathcal{B}}^{\mathsf{PRG}_{2\lambda}}[Q_{\mathsf{out}}] \lesssim \frac{Q_{\mathsf{C}}(Q_{\mathsf{C}} - Q_{\mathsf{out}})}{2^{2\lambda}} + \frac{Q_{\mathsf{out}}}{2^{2\lambda}}, \tag{2}$$

   where Q_{C} bounds the ideal-cipher queries. Hence, this loss becomes material only when the ideal-cipher query budget approaches 2^λ. If this heuristic is not desired, the PRG-based leaf commitment can instead be replaced by the universal-linear-hash leaf commitment used by the non-EM FAEST variants; the proof then uses the standard assumption related to universal hash.
4. ***Hash-function model.*** *The members of $\mathcal{H}$ are modeled as random oracles in the ROM and as quantum-accessible random oracles in the QROM. The keyed derivation used to simulate signing randomness, i.e., H_{seed} in Algorithm 5, satisfies the PRF property.*

C.2 Direct EUF-CMA Bounds in the ROM and QROM

This subsection gives the direct bounds used by the security theorems in the main text. It follows the ROM and QROM analyses of FAEST [9], but uses the notation of this paper. Let $Q_{2,\Sigma} = Q_{\mathsf{msg}} + Q_{2,1} + Q_{2,2} + Q_{2,3}$, where Q_{msg} counts the queries to H_{msg}, and the three terms $Q_{2,1}, Q_{2,2}, Q_{2,3}$ count queries to $H_2^{(1)}, H_2^{(2)}, H_2^{(3)}$, respectively. Let Q_1 denote the query bound for the BAVC aggregation hash H_1 in Table 7, and let Q_4 denote the query bound for the IV-derivation hash role; Q_{C} bounds the ideal-cipher queries. The keyed seed derivation H_{seed} is treated as a PRF. In both theorems, ϱ is the FAEST simulation factor; for every parameter set in Appendix A, $\varrho \leq 2$.

For clarity, the advantage terms in the two bounds have the following meanings. All reduction adversaries are classical in the ROM theorem and quantum in the QROM theorem.

- $\mathsf{AdvEUFCMA}^{\mathsf{Sig}}_{\mathcal{A}}[Q_{\mathsf{sig}}]$ is the probability that $\mathcal{A}$ outputs a valid signature on a message not submitted to its signing oracle, after at most Q_{sig} signing queries.
- $\mathsf{AdvOWF}^{\mathsf{OWF}}_{\mathcal{B}_{\mathsf{owf}}}$ is the one-wayness advantage of recovering a preimage k' from a public OWF instance (x, y) with $y = F_x(k)$, namely the probability that $F_x(k') = y$. Here $F_x(k) = E_k(x)$ in the non-EM variants and $F_x(k) = E_x(k) \oplus k$ in the EM variants.
- $\mathsf{AdvPRF}^{H_{\mathsf{seed}}}_{\mathcal{B}_{\mathsf{prf}}}$ is the PRF distinguishing advantage of the keyed seed derivation. More precisely, it is the distinguishing gap between $H_{\mathsf{seed}}(k\|\cdot)$ for a uniformly generated secret key k and a uniformly random function with the same domain and range.
- $\mathsf{AdvPRP}^{\mathsf{PRG}_{2\lambda}}_{\mathcal{B}_{\mathsf{prg}}}[1]$ is the one-instance PRP distinguishing advantage incurred when the block-cipher calls implementing a 2λ-bit PRG output are replaced by ideal-permutation calls. The notation [1] follows the FAEST one-instance PRP game.
- $\mathsf{AdvInj}^{\mathsf{PRG}_{2\lambda}}_{\mathcal{B}_{\mathsf{inj}}}[L]$ is the almost-injectivity advantage from Equation (2): it bounds the event that one of the L leaf commitments has two distinct seed preimages.

The remaining terms are explicit random-oracle bad-event and proof-soundness losses; d is the maximum constraint degree, L is the total BAVC leaf count, and τ is the number of BAVC sub-trees.

Theorem 3 (EUF-CMA in the ROM [9]). *Let all unkeyed hash interfaces except H_{seed} be random oracles. Let $\mathcal{A}$ be a PPT adversary making Q_{sig} signing queries, and let the Q values above bound its corresponding oracle and ideal-cipher queries. Suppose that $\hat{\ell} \leq 2^{13}$ and $2\log\tau \leq \min\{B, 50\}$. Then there are*

adversaries $\mathcal{B}_{\mathsf{owf}}$, $\mathcal{B}_{\mathsf{prf}}$, $\mathcal{B}_{\mathsf{inj}}$, *and* $\mathcal{B}_{\mathsf{prg}}$ *such that*

$$\begin{aligned}\mathsf{AdvEUFCMA}^{\mathsf{Sig}}_{\mathcal{A}}[Q_{\mathsf{sig}}] \le 4\varrho\left(\begin{array}{l}\dfrac{(Q_{2,\Sigma}+4)^2+(Q_1+1)^2}{2^{2\lambda}} + \mathsf{AdvInj}^{\mathsf{PRG}_{2\lambda}}_{\mathcal{B}_{\mathsf{inj}}}[L] \\ + \dfrac{(Q_{2,1}+Q_{2,2}+Q_{2,3}+9)d}{2^{\lambda}} + \mathsf{AdvOWF}^{\mathsf{OWF}}_{\mathcal{B}_{\mathsf{owf}}}\end{array}\right) \\ + \varrho Q_{\mathsf{sig}}\Big(\mathsf{AdvPRF}^{H_{\mathsf{seed}}}_{\mathcal{B}_{\mathsf{prf}}} + (\lceil \log L\rceil + 1)\mathsf{AdvPRP}^{\mathsf{PRG}_{2\lambda}}_{\mathcal{B}_{\mathsf{prg}}}[1]\Big) \\ + \varrho Q_{\mathsf{sig}}\left(\frac{2Q_1+Q_{2,1}}{2^{2\lambda}} + \frac{Q_{\mathsf{sig}}+Q_{2,2}}{2^{5\lambda+64}} + \frac{Q_{\mathsf{sig}}+Q_{2,3}}{2^{3\lambda+64}}\right).\end{aligned} \tag{3}$$

Theorem 4 (EUF-CMA in the QROM). *Let all unkeyed hash interfaces except* H_{seed} *be quantum-accessible random oracles, and let* H_{seed} *be a PRF. Let* $\mathcal{A}$ *be a quantum polynomial-time adversary making* Q_{sig} *signing queries. Define*

$$Q = Q_1 + Q_{\mathsf{msg}} + Q_{2,1} + Q_{2,2} + Q_{2,3} + Q_4 + 3\tau + 19.$$

Then there are adversaries $\mathcal{B}_{\mathsf{owf}}$, $\mathcal{B}_{\mathsf{inj}}$, $\mathcal{B}_{\mathsf{prf}}$, *and* $\mathcal{B}_{\mathsf{prg}}$ *such that*

$$\begin{aligned}\mathsf{AdvEUFCMA}^{\mathsf{Sig}}_{\mathcal{A}}[Q_{\mathsf{sig}}] \le 4\varrho\left(\frac{10(\tau+1)Q^3}{2^{2\lambda}} + \frac{10dQ^2}{2^{\lambda}} + \mathsf{AdvOWF}^{\mathsf{OWF}}_{\mathcal{B}_{\mathsf{owf}}} + \mathsf{AdvInj}^{\mathsf{PRG}_{2\lambda}}_{\mathcal{B}_{\mathsf{inj}}}[L]\right) \\ + \varrho Q_{\mathsf{sig}}\Big(\mathsf{AdvPRF}^{H_{\mathsf{seed}}}_{\mathcal{B}_{\mathsf{prf}}} + (\lceil \log L\rceil + 1)\mathsf{AdvPRP}^{\mathsf{PRG}_{2\lambda}}_{\mathcal{B}_{\mathsf{prg}}}[1]\Big) \\ + \varrho Q_{\mathsf{sig}}\left((3\sqrt{Q_1} + 2\sqrt{Q_{2,1}})2^{-\lambda} + Q_{2,1}2^{-2\lambda}\right. \\ \left. + \frac{3}{2}\sqrt{\frac{Q_{\mathsf{sig}}+Q_{2,2}}{2^{5\lambda+64}}} + \frac{3}{2}\sqrt{\frac{Q_{\mathsf{sig}}+Q_{2,3}}{2^{3\lambda+64}}}\right).\end{aligned} \tag{4}$$

D The SM4 Embedding

This appendix specifies the embedding used by SM4th. All SM4th variants use the proof field $\mathbb{F}_{2^{128}} = \mathbb{F}_2[Z]/\big(Z^{128}+Z^7+Z^2+Z+1\big)$. Since $8 \mid 128$, this field contains a unique subfield of size 2^8.

Definition 1 (SM4-Compatible Embedding Element). *An element* $\alpha_8 \in \mathbb{F}_{2^{128}}$ *is* SM4-compatible *if* $m_{\mathsf{SM4}}(\alpha_8) = 0$, *where* m_{SM4} *is the SM4 byte polynomial in Equation* (1). *Equivalently,* α_8 *has minimal polynomial* m_{SM4} *over* $\mathbb{F}_2$.

Writing field elements as 16 bytes in least-significant-byte-first order, we fix

$$\alpha_8 = \texttt{b8 4c 7b 2e 8b a1 14 84 b9 44 1f b3 b4 95 a5 51}.$$

For $b = \sum_{r=0}^{7} b_r T^r \in \mathbb{F}_2[T]/(m_{\mathsf{SM4}})$, define $\phi(b) = \sum_{r=0}^{7} b_r \alpha_8^r \in \mathbb{F}_{2^{128}}$.

Lemma 1 (SM4 Embedding Compatibility). *The map ϕ is an $\mathbb{F}_2$-field isomorphism from $\mathbb{F}_2[T]/(m_{\mathsf{SM4}})$ onto $\mathbb{F}_2[\alpha_8]$. In particular, for all $a, b \in \mathbb{F}_{2^8}$,*

$$\phi(a+b) = \phi(a) + \phi(b), \qquad \phi(ab) = \phi(a)\phi(b), \qquad \phi(a^2) = \phi(a)^2.$$

Proof. The polynomial m_{SM4} is irreducible of degree 8, and α_8 is a root of it. Therefore, evaluation at α_8 has kernel (m_{SM4}) and induces the stated isomorphism. The three identities follow immediately.

For a degree-1 commitment $\langle \boldsymbol{b} \rangle$ to the bits of a byte $b = \sum_{r=0}^{7} \boldsymbol{b}[r] T^r$, the public sub-procedure EmbedAndSquare returns

$$\langle \hat{b} \rangle = \sum_{r=0}^{7} \langle \boldsymbol{b}[r] \rangle \alpha_8^r, \qquad \langle \hat{b}_{\mathsf{sq}} \rangle = \sum_{r=0}^{7} \langle \boldsymbol{b}[r] \rangle \alpha_8^{2r} = \langle \phi(b^2) \rangle = \langle \phi(b)^2 \rangle.$$

Both outputs are fixed public $\mathbb{F}_2$-linear combinations of the eight input-bit commitments. Consequently, the parties derive them locally without a commitment multiplication and without increasing the commitment degree.

E Detailed uBlock Round Constants, S-box, and Linear-Layer Definitions

This section gives the detailed information of uBlock. All uBlockith variants use the proof field $\mathbb{F}_{2^{256}} = \mathbb{F}_2[Z]/\big(Z^{256} + Z^{10} + Z^5 + Z^2 + 1\big)$.

E.1 Round Constants

For each key-schedule round $i \in [0, 24)$, the round constant $\mathsf{RC}[i]$ is the corresponding 32-bit hexadecimal value in Table 9.

E.2 The uBlock S-box and Parallel Substitutions

Let $\mathsf{S} : \mathbb{F}_2^4 \to \mathbb{F}_2^4$ denote the uBlock S-box. For an input nibble (x_0, x_1, x_2, x_3) and an output nibble (y_0, y_1, y_2, y_3), the bit indexed by 0 is the least significant bit and the following Boolean algebraic normal form defines S:

$$\begin{aligned}
y_0 &= 1 \oplus x_0 \oplus x_2 x_3, \\
y_1 &= 1 \oplus x_0 \oplus x_1 \oplus x_0 x_1 \oplus x_0 x_2 \oplus x_0 x_1 x_2 \oplus x_0 x_3 \oplus x_2 x_3, \\
y_2 &= 1 \oplus x_1 \oplus x_0 x_1 \oplus x_2 \oplus x_1 x_2 x_3, \\
y_3 &= x_1 \oplus x_2 \oplus x_3 \oplus x_1 x_2.
\end{aligned}$$

For an input nibble (y_0, y_1, y_2, y_3), its inverse S^{-1} outputs (z_0, z_1, z_2, z_3) according to

$$\begin{aligned}
z_0 &= y_0 y_1 \oplus y_2 \oplus y_3 \oplus y_0 y_3 \oplus y_2 y_3 \oplus y_0 y_1 y_3, \\
z_1 &= y_0 \oplus y_1 \oplus y_3 \oplus y_0 y_3, \\
z_2 &= 1 \oplus y_0 \oplus y_2 \oplus y_0 y_1, \\
z_3 &= 1 \oplus y_2 \oplus y_3 \oplus y_0 y_2 \oplus y_1 y_2 \oplus y_2 y_3 \oplus y_0 y_2 y_3.
\end{aligned}$$

Table 9. uBlock key-schedule round constants $\{RC[i]\}_{i\in[0,24)}$ in 32-bit hexadecimal

i	0	1	2	3
RC[i]	0x988CC9DD	0xF0E4A1B5	0x21357064	0x8397D2C6
i	4	5	6	7
RC[i]	0xC7D39682	0x4F5B1E0A	0x5E4A0F1B	0x7C682D39
i	8	9	10	11
RC[i]	0x392D687C	0xB3A7E2F6	0xA7B3F6E2	0x8E9ADFCB
i	12	13	14	15
RC[i]	0xDCC88D99	0x786C293D	0x30246175	0xA1B5F0E4
i	16	17	18	19
RC[i]	0x8296D3C7	0xC5D19480	0x4A5E1B0F	0x55410410
i	20	21	22	23
RC[i]	0x6B7F3A2E	0x17034652	0xEFFBBEAA	0x1F0B4E5A

For a vector $\boldsymbol{a}$ of 16 nibbles and a vector $\boldsymbol{b}$ of 32 nibbles, the forward and inverse parallel substitutions are defined componentwise as

$$\mathsf{SubDWord}(\boldsymbol{a})[i] = \mathsf{S}(\boldsymbol{a}[i]), \quad \mathsf{SubDWord}^{-1}(\boldsymbol{a})[i] = \mathsf{S}^{-1}(\boldsymbol{a}[i]) \quad \text{for } i \in [0,16),$$
$$\mathsf{SubQWord}(\boldsymbol{b})[i] = \mathsf{S}(\boldsymbol{b}[i]), \quad \mathsf{SubQWord}^{-1}(\boldsymbol{b})[i] = \mathsf{S}^{-1}(\boldsymbol{b}[i]) \quad \text{for } i \in [0,32).$$

E.3 The Linear Layer and Its Inverse

A 128-bit branch is written as four 32-bit words. For a 128-bit bit string $\boldsymbol{v}$, the notation $\boldsymbol{v}(\lll s)_{32}$ denotes the independent cyclic left rotation by s bits of each 32-bit word. The byte permutations PL and PR use the input-to-output convention of Table 11.

For input branches $(\boldsymbol{a}_0, \boldsymbol{a}_1)$, define $(\boldsymbol{b}_0, \boldsymbol{b}_1) = \mathcal{L}_{\mathsf{u}}(\boldsymbol{a}_0, \boldsymbol{a}_1)$ by the following ordered assignments:

$$\begin{aligned}
&\boldsymbol{t}_1 \leftarrow \boldsymbol{a}_1 \oplus \boldsymbol{a}_0, && \boldsymbol{t}_0 \leftarrow \boldsymbol{a}_0 \oplus (\boldsymbol{t}_1(\lll 4)_{32}),\\
&\boldsymbol{t}_1 \leftarrow \boldsymbol{t}_1 \oplus (\boldsymbol{t}_0(\lll 8)_{32}), && \boldsymbol{t}_0 \leftarrow \boldsymbol{t}_0 \oplus (\boldsymbol{t}_1(\lll 8)_{32}),\\
&\boldsymbol{t}_1 \leftarrow \boldsymbol{t}_1 \oplus (\boldsymbol{t}_0(\lll 20)_{32}), && \boldsymbol{t}_0 \leftarrow \boldsymbol{t}_0 \oplus \boldsymbol{t}_1,\\
&\boldsymbol{b}_0 \leftarrow \mathsf{PL}(\boldsymbol{t}_0), && \boldsymbol{b}_1 \leftarrow \mathsf{PR}(\boldsymbol{t}_1).
\end{aligned}$$

The inverse mapping $(\boldsymbol{a}_0, \boldsymbol{a}_1) = \mathcal{L}_{\mathsf{u}}^{-1}(\boldsymbol{b}_0, \boldsymbol{b}_1)$ is obtained by reversing these assignments:

$$\begin{aligned}
&\boldsymbol{t}_0 \leftarrow \mathsf{PL}^{-1}(\boldsymbol{b}_0), && \boldsymbol{t}_1 \leftarrow \mathsf{PR}^{-1}(\boldsymbol{b}_1),\\
&\boldsymbol{t}_0 \leftarrow \boldsymbol{t}_0 \oplus \boldsymbol{t}_1, && \boldsymbol{t}_1 \leftarrow \boldsymbol{t}_1 \oplus (\boldsymbol{t}_0(\lll 20)_{32}),\\
&\boldsymbol{t}_0 \leftarrow \boldsymbol{t}_0 \oplus (\boldsymbol{t}_1(\lll 8)_{32}), && \boldsymbol{t}_1 \leftarrow \boldsymbol{t}_1 \oplus (\boldsymbol{t}_0(\lll 8)_{32}),\\
&\boldsymbol{t}_0 \leftarrow \boldsymbol{t}_0 \oplus (\boldsymbol{t}_1(\lll 4)_{32}), && \boldsymbol{t}_1 \leftarrow \boldsymbol{t}_1 \oplus \boldsymbol{t}_0,\\
&\boldsymbol{a}_0 \leftarrow \boldsymbol{t}_0, && \boldsymbol{a}_1 \leftarrow \boldsymbol{t}_1.
\end{aligned}$$

All assignments are $\mathbb{F}_2$-linear. Consequently, $\mathcal{L}_{\mathsf{u}}$ and $\mathcal{L}_{\mathsf{u}}^{-1}$ can be applied locally to VOLE commitments without increasing their degree.

Table 10. Input-to-output uBlock key-schedule permutation PK

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\mathsf{PK}(i)$	3	12	4	13	10	1	9	5	6	15	0	14	11	7	8	2
i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$\mathsf{PK}(i)$	24	25	26	27	28	29	30	31	16	17	18	19	20	21	22	23

Table 11. Input-to-output uBlock byte permutations PL and PR

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\mathsf{PL}(i)$	15	8	0	4	9	14	5	1	2	6	11	13	7	3	12	10
$\mathsf{PR}(i)$	9	2	6	13	5	15	0	8	14	4	11	1	3	10	12	7

E.4 Forward and Backward Two-Round Mappings

Let the state before physical round i be $\boldsymbol{Z} = (\boldsymbol{Z}[0], \boldsymbol{Z}[1])$. The two 128-bit encryption subkeys $\boldsymbol{K}[i][0]$ and $\boldsymbol{K}[i][1]$ are the two halves of $\bar{\boldsymbol{k}}[i]$ defined in Section 5.1. The forward mapping first evaluates physical round i and then returns the two inputs to the S-boxes in physical round $i+1$:

$$
\begin{aligned}
(\boldsymbol{s}_0, \boldsymbol{s}_1) &\leftarrow \big(\mathsf{SubQWord}(\boldsymbol{Z}[0] \oplus \boldsymbol{K}[i][0]), \mathsf{SubQWord}(\boldsymbol{Z}[1] \oplus \boldsymbol{K}[i][1])\big),\\
(\boldsymbol{r}_0, \boldsymbol{r}_1) &\leftarrow \mathcal{L}_{\mathsf{u}}(\boldsymbol{s}_0, \boldsymbol{s}_1),\\
\mathsf{Fwd}_i\big(\boldsymbol{Z}, \bar{\boldsymbol{k}}[i], \bar{\boldsymbol{k}}[i+1]\big) &= \big(\boldsymbol{r}_0 \oplus \boldsymbol{K}[i+1][0], \boldsymbol{r}_1 \oplus \boldsymbol{K}[i+1][1]\big).
\end{aligned}
$$

Conversely, let $\boldsymbol{Z}' = (\boldsymbol{Z}'[0], \boldsymbol{Z}'[1])$ be the claimed state after physical round $i+1$. The backward mapping removes the linear layer of that round and then applies the inverse parallel S-box:

$$
\begin{aligned}
(\boldsymbol{q}_0, \boldsymbol{q}_1) &\leftarrow \mathcal{L}_{\mathsf{u}}^{-1}(\boldsymbol{Z}'[0], \boldsymbol{Z}'[1]),\\
\mathsf{Bwd}_{i+1}(\boldsymbol{Z}') &= \big(\mathsf{SubQWord}^{-1}(\boldsymbol{q}_0), \mathsf{SubQWord}^{-1}(\boldsymbol{q}_1)\big).
\end{aligned}
$$

No key xor is applied in Bwd_{i+1}: its output is already the pre-S-box input of physical round $i+1$, and hence includes that round's key xor. On raw bit strings, $\mathsf{Fwd}_i(\boldsymbol{Z}, \bar{\boldsymbol{k}}[i], \bar{\boldsymbol{k}}[i+1]) = \mathsf{Bwd}_{i+1}(\boldsymbol{Z}')$ if and only if $\boldsymbol{Z}'$ is obtained by applying physical rounds i and $i+1$ to $\boldsymbol{Z}$ under the stated subkeys. For $b \in \{0,1\}$, write $\mathsf{Fwd}_i(\cdot)[b]$ and $\mathsf{Bwd}_{i+1}(\cdot)[b]$ for the two components. When these mappings are

applied to VOLE commitments, every public linear operation is evaluated locally, while the S-box and inverse-S-box ANFs above are evaluated coefficientwise. Thus, both outputs have degree at most 3. For a two-round block indexed by $r \in [0, R/2)$ with $i = 2r$, write $\boldsymbol{X}[r] = (\boldsymbol{X}[r][0], \boldsymbol{X}[r][1])$ for the boundary state as in Section 5.1. The raw-bit degree-3 constraint vectors are

$$\boldsymbol{c}_{\mathsf{enc},0}[r] = \mathsf{Fwd}_i\big(\boldsymbol{X}[r], \bar{\boldsymbol{k}}[i], \bar{\boldsymbol{k}}[i+1]\big)[0] \oplus \mathsf{Bwd}_{i+1}(\boldsymbol{X}[r+1])[0],$$
$$\boldsymbol{c}_{\mathsf{enc},1}[r] = \mathsf{Fwd}_i\big(\boldsymbol{X}[r], \bar{\boldsymbol{k}}[i], \bar{\boldsymbol{k}}[i+1]\big)[1] \oplus \mathsf{Bwd}_{i+1}(\boldsymbol{X}[r+1])[1].$$

When lifted from raw bits to VOLE commitments, each componentwise $\oplus$ in these two expressions is evaluated as addition in $\mathbb{F}_{2^{256}}$. Each vector has 128 bit coordinates. Their concatenation therefore contributes 256 degree-3 constraints per two-round block.