

Practical Null-Branch Witness Attacks on In-the-Head Signatures

Guoxiao Liu¹, Zhuoqing Peng¹, Guowei Liu², Keting Jia¹, Xiaoyun Wang¹,
Zongyue Wang³, and Junfeng Fan³

¹ Tsinghua University

lgx22@mails.tsinghua.edu.cn, pzq24@mails.tsinghua.edu.cn

ktjia@tsinghua.edu.cn, xiaoyunwang@tsinghua.edu.cn

² Shandong University

guoweiliu@mail.sdu.edu.cn

³ Open Security Research

zongyue.wang@osr-tech.com, fan@osr-tech.com

Abstract. In-the-head signatures provide a route to post-quantum authentication based on symmetric primitives. Their algebraic instantiations rely on constraint relations that faithfully encode the underlying one-way function (OWF). We give a classical, public-key-only forgery attack on AIM2-based AIMer v2.0; AIMer was selected in Korea’s KpqC competition. For every honestly generated public key of every v2.0 parameter set, the attack forges signatures on arbitrary messages with probability one, without signing queries.

We introduce a *null-branch witness attack* exploiting zero factors that leave intermediate values unconstrained. We complete these local assignments into witnesses satisfying the full relation, including public-output binding, using only public data and without solving an OWF inversion problem. Prover completeness ensures that the original message-bound prover can use these witnesses to generate signatures accepted by the unmodified verifier. Proof-system soundness applies to the encoded relation, which the constructed witnesses satisfy.

Tests with reference implementations confirm accepted forgeries from satisfying witnesses whose decoded values are not valid OWF preimages. The median AIMer-256f forgery API time is approximately 11.6 ms in our reference benchmark. We identify a gap in AIMer’s security reduction: a satisfying relation witness need not decode to an OWF preimage. A complementary application to vulnerable Lynxer variants extends the analysis to VOLE-in-the-Head. We give quadratic encodings that are sound and complete for the full AIM2 and Lynx computations, including zero inputs. These results highlight relation encoding as a distinct security obligation between OWF hardness and proof-system soundness.

Keywords: Post-quantum signatures · MPC-in-the-head · VOLE-in-the-head
· AIMer · Null-branch witness attacks

1 Introduction

In-the-head signatures provide post-quantum public-key authentication from symmetric primitives and zero-knowledge proofs. They combine proofs of knowledge of a one-way function (OWF) preimage with the Fiat–Shamir transformation to bind proofs to messages [15]. Ishai et al. established the MPC-in-the-Head (MPCitH) approach through simulated multiparty computations [17], and Chase et al. developed practical post-quantum signatures along this route [9]. At CRYPTO 2023, Baum et al. introduced VOLE-in-the-Head (VOLEitH), which converts designated-verifier protocols based on vector oblivious linear evaluation into publicly verifiable proofs [5]. In the constructions considered here, in-the-head proofs enforce algebraic constraints encoding computations built from symmetric primitives.

Signature schemes built on these proof frameworks have gained substantial attention in post-quantum standardization efforts. The AES-based FAEST has advanced to Round 3 of NIST’s additional-signature process [3,1], while the LowMC-based Picnic reached the alternate-candidate stage of the original competition [8,2]. In Korea, the KpqC competition selected the AIM2-based AIMer [21,23]; an earlier version also entered the first round of NIST’s additional-signature process [28]. Other representative designs include the RAIN-based Rainier, GALAS, and the Lynx-based Lynxer [13,27,18]. These developments motivate careful analysis of both the underlying primitives and the proof systems used to establish their correct evaluation.

Two complementary lines of research are particularly relevant to our analysis. At the protocol level, security analyses address soundness and extraction, commitment and challenge binding, the Fiat–Shamir transformation, and concrete security bounds. Kales and Zaverucha’s BN++ provides efficient MPCitH proofs for arithmetic circuits [19], while Yang et al.’s QuickSilver supports designated-verifier proofs of algebraic relations [35]. For VOLEitH, Baum et al. analyzed the Fiat–Shamir transformation using round-by-round soundness [5]. Their subsequent *Shorter*, *Tighter*, *FAESTer* improves FAEST’s efficiency and concrete security analysis in the classical and quantum random-oracle models [4]. These analyses establish guarantees relative to the relation supplied to the proof system.

At the primitive level, algebraic cryptanalysis combines representations of the OWF computation with techniques for solving the resulting equations. Equivalent representations underlie Liu et al.’s attacks on RAIN and AIM [26], while Zhang et al. exploit power-map structure [36]. For algebraic modeling and solution, Liu et al. develop overdefined quadratic systems [25], and Yang et al. use polynomial decomposition [34]. Biryukov et al.’s *Magic Pot* introduces a new elimination framework and applies it to full AIM2 [7]. These attacks seek inputs satisfying the semantic OWF computation and assess the hardness assumption underlying the signature construction.

Connecting these analyses requires an explicit audit of the *relation-to-OWF reduction interface*. To reduce proving costs, designers express nonlinear OWF computations through low-degree constraints on intermediate and auxiliary val-

ues. These encodings connect the semantic OWF computation to the relation enforced by the proof system. They must preserve that computation on all admitted witness assignments, including exceptional zero cases. A reduction that obtains an OWF preimage by decoding an extracted witness requires the decoded value to satisfy the semantic OWF relation. We call the requirement that every satisfying relation witness decode to a valid preimage *relation-encoding soundness* (Definition 1). For decision-soundness-based reductions, the corresponding encoding requirement is weaker: semantically false statements must remain false under the encoded relation on the reduction’s statement domain. A discrepancy between the semantic and encoded relations does not by itself establish an efficient forgery attack. Exploiting it through witness construction requires efficient public computation of a complete satisfying witness that the signature’s proving procedure can use.

Our *null-branch witness attacks* target the AIM2-based AIMer specification of 30 January 2026 (v2.0) [10,24] and vulnerable Lynxer variants [18]. In the vulnerable encodings, a zero factor makes a local equation cease to enforce an intended semantic restriction. Using only public data, our constructors complete this local freedom into witnesses satisfying the full relation for the supplied public key, without solving an OWF inversion problem. The original proving procedure then uses these witnesses to produce signatures accepted by unmodified verification. The attack satisfies the equations enforced by BN++ and QuickSilver [19,35] and is consistent with their soundness guarantees. The AIM3-based AIMer v3.0 relation excludes the zero branches used here (Section 6.4).

1.1 Contributions

Our main result is a classical, public-key-only forgery attack on AIMer v2.0. AIMer’s selection in Korea’s KpqC competition makes this result directly relevant to ongoing post-quantum signature standardization [23]. For every honestly generated public key of every v2.0 parameter set, the attack forges signatures on arbitrary messages with probability one, without signing queries (Theorem 1). The Lynxer application extends the analysis to VOLEitH [18] and exposes a distinct gap in a decision-soundness-based reduction. Three contributions support these results.

- *Public-key-only witness construction.* We develop a polynomial-time method for completing local null branches into witnesses satisfying the entire signing relation. For AIMer v2.0, it satisfies all branch and public-output constraints using only publicly derived values. Each witness can be reused to sign arbitrary messages through AIMer’s original message-bound proving procedure. The Lynxer application shows how output binding determines whether witness completion is universal or conditional on the public key.
- *Reduction analysis and encoding requirements.* We identify a flaw in AIMer’s constraint encoding and show that its security reduction implicitly cancels a hidden value that may be zero. We formalize relation-encoding soundness and define *equivalent witnesses* as satisfying witnesses whose decoded inputs are not OWF preimages (Definition 2). A generic bound relates public

witness construction and prover completeness to forgery success, separating relation satisfaction from semantic preimage validity. Lynxer provides a complementary case involving failure of the language inclusion required by a decision-soundness reduction.

- *Implementation validation and semantics-preserving encodings.* We validate forgeries in the AIMer-128f, AIMer-192f, and AIMer-256f reference configurations; additional Lynxer tests bring the total to seven configurations. The tested witnesses satisfy all checked equations, decode to invalid OWF preimages, and produce signatures accepted by unmodified verifiers. The median AIMer-256f forgery API time is approximately 11.6 ms in our reference timing setup. We construct zero-safe quadratic encodings and prove their soundness and completeness for the full AIM2 and Lynx computations, including zero inputs, with explicit witness and constraint overheads.

Table 1 summarizes the proved key coverage and tested variants.

Table 1. Proved attack coverage and tested variants.

Scheme and variants	Proved key coverage	Tested variants
AIMer v2.0 128/192/256 (s, f)	All parser-accepted keys with defined IV derivation	128f, 192f, 256f
Lynxer 256/384/512 (s, f)	All parser-accepted keys; arbitrary public linear maps	256f, 384f, 512f
Lynxer 192 (s, f)	Keys satisfying the public linear consistency test	192f

Tests. Sections 4.4 and 5.4 document the reference tests and their configurations. Lynx-128’s inverse checks block these zero branches.

1.2 Related Work

Relation witnesses and primitive inputs. The primitive attacks discussed above, including Biryukov et al.’s analysis of AIM2 [7], seek inputs satisfying the semantic computation. Our constructors instead solve the encoded signing relation directly from a public key. The resulting witnesses suffice for signature generation; their decoded inputs are checked separately to establish failures of relation-encoding soundness.

Computation and constraint consistency. Beyond primitive cryptanalysis, the closest conceptual connection is to underconstrained circuits and discrepancies between intended computations and accepted witnesses. Pailoor et al. study output determinacy [29], while Wen et al. use static analysis to detect circuit vulnerabilities [33]. Stephens, Pailoor, and Dillig’s Zequal compares the outputs of accepted valuations and generated witnesses [31]. Related consistency analyses include Takahashi et al.’s zkFuzz and Kolozyan et al.’s domain-consistency study [32,22]. Our work connects such discrepancies to signature security through

public witness construction, analysis of the relation-to-OWF reduction, and accepted message-bound forgeries.

2 Preliminaries

We introduce the field arithmetic, the semantic and encoded relations, and the algebraic checks used by the two proof backends. These conventions let us compare a submitted witness with an OWF computation. Appendix A gives the transcript mechanics.

2.1 Notation and Field Arithmetic

Let λ denote the security parameter and n the extension-field degree. Both targets use

$$\mathbb{K} = \mathbb{F}_2, \quad \mathbb{E} = \mathbb{F}_{2^n}.$$

We index the submitted parameter families by their field dimension, $\lambda = n$. Fix the $\mathbb{K}$ -basis $(\beta_0, \dots, \beta_{n-1})$ used by the implementation. The coordinate vector $(a_0, \dots, a_{n-1}) \in \mathbb{K}^n$ represents the field element $a = \sum_{i=0}^{n-1} a_i \beta_i \in \mathbb{E}$. We write witnesses either as tuples of field elements or as their binary encodings. Addition and subtraction are coordinatewise XORs; multiplication is the extension-field operation.

AIMer’s MPCitH backend additively shares the field elements. Lynxer’s VOLEitH backend authenticates the witness bits and combines them into authenticated field elements using the public basis coefficients. Public $\mathbb{K}$ -linear maps are evaluated locally on the additive shares in MPCitH and on the authenticated bits before packing in VOLEitH. Appendix A details these witness interfaces.

Throughout, “linear” means $\mathbb{K}$ -linear, including Frobenius maps $a \mapsto a^{2^j}$. For public $a \in \mathbb{E}$, write $M_a : b \mapsto ab$, identifying this map with its binary matrix when taking ranks. The totalized inverse is $\text{Inv}_0(a) = a^{-1}$ for $a \neq 0$, with $\text{Inv}_0(0) = 0$.

We use m for the encoded message, including a fixed admissible context when the scheme has a context input. Public parameters keep their notation when witness coordinates are primed to denote an attack assignment.

Constraint degree is measured in $\mathbb{K}$ -coordinates. A product of two hidden $\mathbb{E}$ -valued linear expressions is therefore quadratic, even with large Frobenius exponents. For example, $x^2y = x$ is quadratic: squaring is linear and is followed by one hidden multiplication. The backends and repairs use this convention.

Equation tags distinguish semantics (Sem), checked branches and output (Br_j, Out), and repairs (Rep), with primitive names omitted in local references.

2.2 OWF Relations and Constraint Encodings

An in-the-head signature uses a proof of knowledge of an OWF preimage. To relate that claim to the equations checked by the proof, we distinguish the candidate input x from the complete witness W supplied to the prover.

For an IV-dependent OWF family $F = \{F_{iv} : \mathcal{X}_\lambda \rightarrow \mathcal{Y}_\lambda\}_{iv \in \mathcal{I}_\lambda}$ and statement $\text{pk} = (iv, y)$, the semantic preimage relation is

$$R_F((iv, y), x) = 1 \iff F_{iv}(x) = y.$$

Thus R_F describes the intended input–output computation. Here $\mathcal{X}_\lambda$ is the ambient input space, which is $\mathbb{E}$ in our targets; restricted-domain variants are specified below.

The prover’s witness also contains the intermediate and auxiliary values used by the proof. In our targets, we write $W = (x, \tau) \in \mathcal{W}_\lambda$ and use a decoder Dec that selects x . The proof system checks a finite list of residuals, each obtained by subtracting the right-hand side of a constraint from its left-hand side:

$$C_{iv,y}(W) = (f_1((iv, y), W), \dots, f_r((iv, y), W)).$$

The encoded relation and decoding map are

$$R_C((iv, y), W) = 1 \iff C_{iv,y}(W) = 0, \quad \text{Dec} : \mathcal{W}_\lambda \rightarrow \mathcal{X}_\lambda.$$

A satisfying witness makes every residual vanish, including consistency and output constraints. Whereas R_F tests whether x is an OWF preimage, R_C tests whether the submitted coordinates satisfy these equations.

For an honestly generated key, witness extension evaluates the OWF on its secret input x and records $W_h \leftarrow \text{Extend}_F(\text{pk}, x)$. The signer supplies this witness to the message-bound prover. The encoded relation must enforce the dependencies established by that computation, since an adversary can supply different intermediate values. A complete trace may also include values derived locally from other coordinates, as in AIMer’s affine layer; these remain fixed by the corresponding dependencies. Section 3.3 formalizes the prover interface used by the attacks.

Input and public-key domains. If the declared OWF domain is restricted, R_F additionally requires a public membership predicate $\text{Dom}_{\text{pk}}(x) = 1$. A restriction used only during KeyGen does not constrain arbitrary submitted witnesses. Our repairs preserve the total computation, including zero inputs; a restricted-domain instantiation also enforces its membership predicate. Appendix B formalizes these choices.

Separately, each encoding claim names a public-key domain $\mathcal{D} = \{\mathcal{D}_\lambda\}_\lambda$. The honest KeyGen support $\mathcal{P}_\lambda^{\text{hon}}$ is contained in the parser-accepted domain $\mathcal{P}_\lambda^{\text{syn}}$. A claim may cover either domain or a specified intermediate domain, such as parser-accepted keys with defined public IV derivation. The input-domain condition specifies which preimages count; $\mathcal{D}$ specifies the public keys for which the claim holds.

For a witness set $\mathcal{A}$, write $\text{Dec}(\mathcal{A}) = \{\text{Dec}(W) : W \in \mathcal{A}\}$. The satisfying sets, with elements in their respective witness spaces, are

$$\begin{aligned} \mathcal{S}_C(\text{pk}) &= \{W : R_C(\text{pk}, W) = 1\}, \\ \mathcal{S}_F(\text{pk}) &= \{x : R_F(\text{pk}, x) = 1\}. \end{aligned}$$

The central question is whether every satisfying encoded witness decodes to a semantic preimage: does $\text{Dec}(\mathcal{S}_C(\text{pk})) \subseteq \mathcal{S}_F(\text{pk})$ hold on the declared public-key domain? Section 3.1 formalizes this requirement as relation-encoding soundness.

2.3 In-the-Head Proof Backends

MPCitH and BN++. For AIMer’s MPCitH instantiation, $w \in \mathbb{E}$ is additively shared among N virtual parties as $w = \sum_i w^{(i)}$; the prover commits to simulated views and opens a challenged subset [17]. AIMer opens $N - 1$ views per repetition, hiding the complete witness, and batches hidden multiplications with BN++ [10, 19].

For submitted triples (x_j, y_j, z_j) , let $a_j \in \mathbb{E}$ be random masks and set $b_j = y_j$ and $c = \sum_j a_j b_j$. After these values and the triples are fixed, random coefficients ϵ_j define the opened values $\alpha_j = \epsilon_j x_j + a_j$ and the check

$$v = \sum_j \epsilon_j z_j - \sum_j \alpha_j b_j + c = \sum_j \epsilon_j (z_j - x_j y_j). \quad (\text{MPC})$$

Valid products make $v = 0$ for every batch. For any fixed nonzero residual vector, independent uniform $\epsilon_j \in \mathbb{E}$ make this single batch check pass with probability $1/|\mathbb{E}|$. AIMer’s repeated-multiplier optimization preserves this residual interpretation; view opening and repetition supply the remaining protocol checks.

VOLEitH and QuickSilver. VOLE supplies authenticated hidden coordinates. A degree-one information-theoretic MAC (IT-MAC) of $a \in \mathbb{E}$ has prover polynomial $p_a(X)$ and verifier evaluation K_a satisfying

$$p_a(X) = \rho_a + aX, \quad K_a = p_a(\Delta) = \rho_a + \Delta a. \quad (\text{MAC})$$

The evaluation point Δ is unpredictable when the prover polynomials are fixed. Public $\mathbb{E}$ -linear combinations preserve this invariant; multiplying polynomials and evaluations adds formal degrees [35]. General $\mathbb{K}$ -linear maps, including Frobenius maps, act on authenticated bits before packing into $\mathbb{E}$. Witness corrections fix random authenticated bits to any prover-chosen witness. VOLEitH realizes the correlations through seed commitments and all-but-one openings [5].

QuickSilver checks residuals through their highest polynomial coefficients. For the multiplication residual $ab - c$, it forms

$$g(X) = p_a(X)p_b(X) - Xp_c(X), \quad g(\Delta) = K_a K_b - \Delta K_c.$$

The coefficient of X^2 is $ab - c$. QuickSilver batches these residuals and masks the lower coefficients with a formal degree-two IT-MAC of zero: its X^2 coefficient is zero, while its lower coefficients are random. A satisfying witness makes every residual’s highest coefficient zero; the lower coefficients need not vanish. Thus the check authenticates the residual value without exposing the witness coordinates.

Both verifiers receive only (pk, m, σ) and check commitments, message-bound challenges, openings, and relation residuals. These operations bind a common hidden witness to the submitted relation.

3 A Framework for Null-Branch Witness Attacks

We introduce a *null-branch witness attack* that exploits the freedom left by vanishing factors in algebraic constraints. The central task is to complete these local assignments into a full relation witness, including public-output binding, usable by the original proving computation. We separate this construction from the semantic check that identifies a relation-encoding violation, then derive the signature-forgery bound from prover completeness.

3.1 Relation-Encoding Soundness and Equivalent Witnesses

A proof system establishes satisfaction of its declared relation. Interpreting the decoded value of an extracted witness as an OWF preimage requires an additional relation-encoding property.

Definition 1 (Relation-encoding soundness). *We call R_C sound with respect to R_F under Dec on $\mathcal{D}$ if, for every λ , every $\text{pk} \in \mathcal{D}_\lambda$, and every $W \in \mathcal{W}_\lambda$,*

$$R_C(\text{pk}, W) = 1 \implies R_F(\text{pk}, \text{Dec}(W)) = 1. \quad (\text{RES})$$

Equivalently, $\text{Dec}(\mathcal{S}_C(\text{pk})) \subseteq \mathcal{S}_F(\text{pk})$ for every public key in the declared domain.

A violation of this property is witnessed by a satisfying relation witness whose decoded value is not a semantic OWF preimage.

Definition 2 (Equivalent witness). *For a public key $\text{pk} \in \mathcal{D}_\lambda$ and the declared relations R_C, R_F and decoder Dec , an equivalent witness is a complete witness $W^* \in \mathcal{W}_\lambda$ such that*

$$R_C(\text{pk}, W^*) = 1 \quad \text{and} \quad R_F(\text{pk}, \text{Dec}(W^*)) = 0.$$

Here *equivalent* is a paper-specific term for a semantically invalid relation witness usable by the public prover under its completeness guarantee. It asserts neither OWF equivalence nor indistinguishability of proof transcripts.

Relation-encoding soundness is exactly the absence of equivalent witnesses on the declared domain. It is a structural guarantee for direct witness decoding, not a necessary condition for EUF-CMA security. Its failure alone does not establish efficient forgery (Appendix B.3).

Knowledge soundness or extractability recovers a witness for R_C ; relation-encoding soundness makes its decoded value an OWF preimage, as illustrated in Figure 1. Lynxer’s decision-soundness reduction instead needs the weaker language inclusion

$$\mathcal{S}_C(\text{pk}) \neq \emptyset \implies \mathcal{S}_F(\text{pk}) \neq \emptyset. \quad (\text{LI})$$

RES implies (LI), but (LI) does not require $\text{Dec}(W)$ to be a preimage. Thus, an equivalent witness disproves RES, whereas disproving (LI) requires a satisfiable constraint relation for a public key with no semantic OWF preimage. For honest keys, (LI) holds because a semantic preimage exists, even if other relation witnesses violate RES. Lynxer’s uniform-key game hop also requires (LI) outside the honest-key support, under the OWF game’s declared semantics. Section 5 examines outputs outside the semantic image where this inclusion fails.

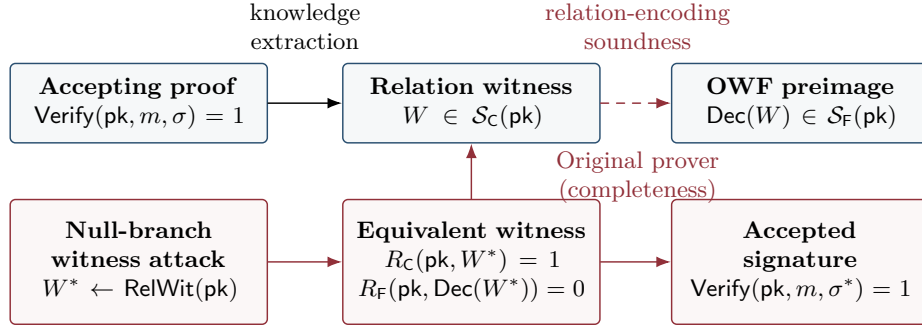


Fig. 1. The relation-to-OWF gap. Above, knowledge extraction yields a relation witness; relation-encoding soundness connects its decoded input to an OWF preimage. Below, the tested null-branch witness attacks supply equivalent witnesses to the original prover. It generates a signature on any chosen message, subject to the completeness guarantee in Section 3.3. The two witness conditions are those of Definition 2.

3.2 Null-Branch Witness Construction

Null branches. A *null branch* occurs when a vanishing factor removes an intended semantic restriction: replacing $H(a, z) = 0$ by $aH(a, z) = 0$, for example, leaves it unchecked at $a = 0$. Other constraints may restore that restriction or exclude the zero assignment. Exploiting a null branch therefore requires completing the local assignment into a witness satisfying the full relation.

- *Mersenne checks.* For an integer $e \geq 1$, the equation $t^{2^e} = ut$ determines $u = t^{2^e-1}$ when $t \neq 0$, but at $t = 0$ it leaves u unrestricted instead of requiring $u = 0$. AIMER uses a hidden first-layer output as this cancelling factor.
- *Cleared denominators.* The equation $p = wz$ encodes $w = p/z$ for $z \neq 0$. At $p = z = 0$, it leaves w free, allowing the public-output dependence carried by w to disappear in large Lynx.
- *Kasami checks.* For $d_i = 2^{2i} - 2^i + 1$, the compressed equation $x^{2^i}y = x^{2^{2i}}x$ determines $y = x^{d_i}$ for $x \neq 0$. At $x = 0$, it omits the semantic condition $y = 0$.

How output binding determines coverage. A later affine layer or output check may fix a coordinate freed by a local equation. After identifying a cancelled factor, we substitute the selected zero branch into the *whole* relation and solve every remaining residual. The supplied public output determines whether completion is possible.

Fix the public IV, derived parameters, and a branch choice with preprocessing independent of the public output. The following linear analysis applies when the full residual relation is equivalent to an affine system over the chosen field $\mathbb{K}$:

$$M\mathbf{v} = b + N\mathbf{y},$$

where $\mathbf{v}$ contains the remaining hidden coordinates and $\mathbf{y}$ encodes the public OWF output. The matrices M, N and vector b are fixed by the preceding choices and independent of $\mathbf{y}$. All remaining constraints must be included; solving only a linear subset does not certify a satisfying relation witness. Under these conditions, completion on this branch is possible exactly when

$$\text{rank}(M) = \text{rank}[M \mid b + N\mathbf{y}].$$

Gaussian elimination constructs the remaining coordinates. Completion for every $\mathbf{y}$ in the full coordinate space is possible exactly when $b \in \text{im}(M)$ and $\text{im}(N) \subseteq \text{im}(M)$; full row rank is sufficient. Coverage of all outputs for the fixed IV can be stronger than coverage of its honestly generated outputs. This criterion characterizes completion on a fixed branch; it does not provide an efficient procedure for finding that branch. Failure on the selected branch does not exclude other constructions.

Semantic evaluation separately determines whether a constructed relation witness also violates relation-encoding soundness.

3.3 From Witness Construction to Signature Forgery

Honest signing extends an OWF input to $W_h \leftarrow \text{Extend}_F(\text{pk}, x)$ and runs the public message-bound prover $\text{Prove}_{R_c}(\text{pk}, W_h; m)$ [15,10,18]. A forger supplies a complete satisfying witness W^* to the same computation. Relation satisfaction suffices for this forgery step, whether or not the resulting witness is equivalent in the sense of Definition 2.

Assume that, for every honest public key, every message, and every satisfying relation witness, the public PPT prover produces a signature accepted by Verify with probability at least $1 - \delta_{\text{comp}}(\lambda)$. This completeness guarantee covers all satisfying relation witnesses, not only those produced by honest witness extension. In the random-oracle model, the bound must hold conditional on the constructor's output and prior oracle transcript.

Proposition 1 (Witness-to-forgery conversion). *Let RelWit be a public-key-only PPT constructor with output in $\mathcal{W}_\lambda \cup \{\perp\}$. Suppose it returns a satisfying relation witness with probability $\varepsilon_{\text{RelWit}}(\lambda)$ over honest key generation and its own randomness. Under the completeness assumption above, a PPT adversary making no signing queries forges on any chosen message with probability at least*

$$\varepsilon_{\text{RelWit}}(\lambda)(1 - \delta_{\text{comp}}(\lambda)).$$

The bound averages over honest key generation and construction randomness, rather than asserting a uniform per-key success rate. If the constructor succeeds with probability one for every honestly generated public key, the forgery probability is at least $1 - \delta_{\text{comp}}$ for each such key. Non-negligible construction success and negligible completeness error contradict EUF-CMA security.

AIMer's BN++ prover has perfect completeness for every satisfying relation witness, so $\delta_{\text{comp}} = 0$. For Lynxer, δ_{comp} includes counter-grinding termination

failure. Appendix A describes witness injection and subsequent transcript generation through the original proving routines. Appendix B.2 gives the proposition’s proof, explicit probability conventions, and the grinding bound with its assumptions.

4 Application to AIMer

For every honestly generated public key of every parameter set in AIMer v2.0 [10], our null-branch witness attack produces signatures on arbitrary messages with probability one, without signing queries. The attack constructs a satisfying AIM2 relation witness from public data and supplies it to the BN++ prover. The resulting signatures are accepted by the unmodified verifier; Section 4.4 reports validation with the reference implementation.⁴

4.1 AIM2 Computation and Constraint Encoding

The public key is (iv, ct) , where $ct = AIM2_{iv}(pt)$, and the secret key contains pt . Over $\mathbb{E} = \mathbb{F}_{2^n}$, let $Mer[e](z) = z^{2^e - 1}$. Here, $Mer[e]^{-1}$ denotes the inverse permutation of this power map. The branch constants γ_j are fixed by the parameter set. The public IV determines the binary matrices $A_{iv,j}$ and a field element b_{iv} represented by an n -bit vector. On input pt , the AIM2 computation evaluates

$$\begin{aligned} t_j &= Mer[e_j]^{-1}(pt + \gamma_j) \quad (0 \leq j < \ell), \\ t_* &= b_{iv} + \sum_{j=0}^{\ell-1} A_{iv,j} t_j, \\ ct &= Mer[e_*](t_*) + pt. \end{aligned} \tag{AIM2-Sem}$$

Figure 2 shows the computation in our trace notation. The parallel layer has $\ell = 2$ branches at $n = 128, 192$ and $\ell = 3$ at $n = 256$.

We represent the complete honest computation trace as

$$W_h = (pt, t_0, \dots, t_{\ell-1}, t_*),$$

where t_* is determined by the affine layer. Only $(pt, t_0, \dots, t_{\ell-1})$ are independently supplied witness coordinates: the prover shares these values and computes t_* locally on the shares. Thus t_* is included in our trace notation but is not an independently adjustable input to BN++.

BN++ checks the first-layer branch constraints Br_j ($0 \leq j < \ell$) and the output constraint Out. Their low-degree encodings are

$$t_j^{2^{e_j}} + \gamma_j t_j = ptt_j, \tag{AIM2-Br}$$

$$t_*^{2^{e_*}} + ct t_* = ptt_*. \tag{AIM2-Out}$$

⁴ <https://github.com/samsungsds-opensource/AIMer>

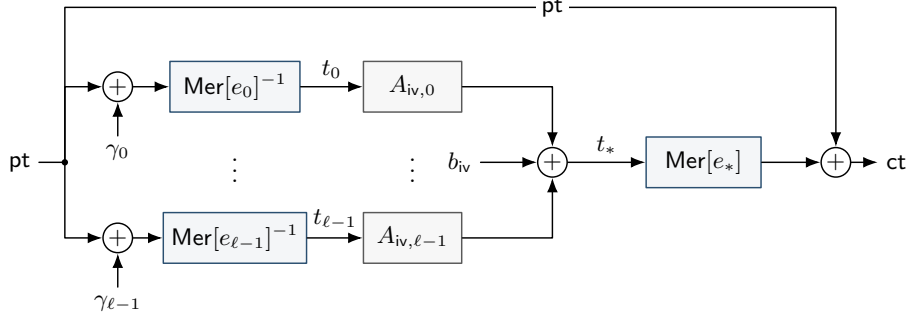


Fig. 2. AIM2 evaluation: inverse Mersenne branches feed an IV-derived affine layer, followed by a forward Mersenne map and input feed-forward. Here $\text{Mer}[e](a) = a^{2^e-1}$; the witness comprises $\text{pt}, t_0, \dots, t_{\ell-1}, t_*$.

The local affine computation is embedded in the reconstructed party views. Appendix A details witness injection and the corresponding BN++ checks.

For $t_j \neq 0$, cancelling the factor t_j in (AIM2-Br) recovers $t_j^{2^{e_j}-1} = \text{pt} + \gamma_j$. At $t_j = 0$, however, (AIM2-Br) reduces to $0 = 0$ for every value of $\text{pt} + \gamma_j$, leaving the semantic requirement $\text{pt} + \gamma_j = 0$ unenforced. The construction below completes this null-branch assignment by satisfying the affine and output equations.

4.2 Null-Branch Witness Construction

Setting every first-layer value to zero satisfies all branch constraints and fixes the affine value to $t'_* = b_{iv}$. The public choice $\text{pt}' = \text{ct} + b_{iv}^{2^{e_*}-1}$ then satisfies the output constraint. Algorithm 1 implements this closed-form construction without searching for an AIM2 preimage. The theorem below establishes relation satisfaction for every public key in its stated domain. Whether the result is also an equivalent witness is determined separately by testing $\text{AIM2}_{iv}(\text{pt}') \neq \text{ct}$, as in Section 4.4.

Algorithm 1 $\text{RelWit}_{\text{AIMer}}(iv, ct)$

1. Derive the public affine constant b_{iv} .
 2. Set $t'_j \leftarrow 0$ for every first-layer branch.
 3. Set $t'_* \leftarrow b_{iv}$.
 4. Set $\text{pt}' \leftarrow \text{ct} + b_{iv}^{2^{e_*}-1}$.
 5. Return $W^* = (\text{pt}', t'_0, \dots, t'_{\ell-1}, t'_*)$.
-

Theorem 1 (AIMer witness construction). *Fix any AIMer v2.0 parameter set specified in [10]. For every $\text{pk} = (iv, ct) \in \mathcal{P}_\lambda^{\text{syn}}$ on which the specified public IV derivation is defined, Algorithm 1 deterministically computes, in polynomial time, a witness $W^* \in \mathcal{W}_\lambda$ satisfying*

$$R_C(\text{pk}, W^*) = 1.$$

Consequently, there exists a public-key-only PPT adversary that, for every honestly generated public key and every chosen message, produces a signature accepted with probability one over its signing randomness, without making signing queries.

Proof. All values in the construction are public or publicly derived. For every first-layer branch, substituting $t'_j = 0$ into (AIM2-Br) gives $0 = 0$ independently of $\mathbf{pt}'$. The affine equality gives

$$t'_* = b_{iv} + \sum_j A_{iv,j} t'_j = b_{iv}.$$

Using $\mathbf{pt}' = \mathbf{ct} + b_{iv}^{2^{e^*}-1}$, equation (AIM2-Out) becomes

$$b_{iv}^{2^{e^*}} + \mathbf{ct} b_{iv} = (\mathbf{ct} + b_{iv}^{2^{e^*}-1}) b_{iv},$$

which is an identity, including at $b_{iv} = 0$. Thus every equation submitted to BN++ holds on W^* . The construction uses only public derivation and field operations and runs in polynomial time. It succeeds deterministically for each honestly generated public key. Applying the per-key conclusion of Proposition 1 with AIMer's perfect completeness gives forgery success probability one for each such key. $\square$

4.3 Gap in the AIMer Security Reduction

The reduction from AIMer forgery to AIM2 inversion relies on interpreting correct multiplication triples as a valid AIM2 computation. The equivalent witness in Section 4.4 invalidates the required implication

$$\text{all submitted AIM2-MPC triples are correct} \implies \text{AIM2}_{iv}(\mathbf{pt}) = \mathbf{ct}.$$

Every submitted identity holds, yet the decoded value satisfies $\text{AIM2}_{iv}(\mathbf{pt}') \neq \mathbf{ct}$. The null branches therefore separate relation satisfaction from semantic preimage validity.

The gap occurs in the specification's EUF-KO reduction (Theorem 1) [10]. Its AIM2 inverter $\mathcal{B}$ records plaintext candidates recovered from the forger's random-oracle queries. After the forger terminates, $\mathcal{B}$ outputs a recorded $\mathbf{pt}_k$ only if $\text{AIM2}_{iv}(\mathbf{pt}_k) = \mathbf{ct}$; otherwise it outputs $\perp$. Conditioned on this latter event, the proof bounds successful verification using the multiplication-check soundness bound in the specification's Lemma 1, which applies to an *incorrect* submitted multiplication triple. A candidate's failure to be an AIM2 preimage, however, does not imply that any submitted triple is incorrect.

BN++ correctly checks satisfaction of the encoded relation. The missing step is the additional implication from that relation to the AIM2 semantics; multiplication-check soundness alone does not establish it. Appendix A details the corresponding verifier operations.

4.4 Validation and Cost

The archived KAT records cover one key for each of AIMER-128f, AIMER-192f, and AIMER-256f. Each record contains an honest control signature, a distinct forgery on the original message m_0 , and a forgery on a fresh message m_1 , all accepted by the unmodified verifier. Table 2 summarizes these implementation checks; the all-key conclusion follows from Theorem 1.

Table 2. KAT validation of AIMER equivalent witnesses. Each example has one control signature and two forgeries, on the original and a fresh message. Signature lengths are in bytes.

Target	Matrices	Accepted	Bytes
AIMer-128f	IV-derived	3/3	5,888
AIMer-192f	IV-derived	3/3	13,056
AIMer-256f	IV-derived	3/3	25,120

Accepted reports accepted/checked signatures. Direct AIM2 evaluation at each forged trace’s decoded input differs from the public output. Appendix F.2 identifies the full records and rebuild procedure.

Table 3 gives the AIMER-256f example. Supplying Algorithm 1’s witness to the BN++ prover yields both forgeries. All branch and output residuals vanish, whereas direct semantic evaluation gives $\text{AIM2}_{\text{iv}}(\text{pt}') \neq \text{ct}$. Together, these checks establish an accepted forgery from an equivalent witness, beyond satisfaction of the displayed equations alone.

The attack inputs are the public key and the chosen message; the KAT seed is only reproduction metadata. The harness reuses the signing API’s secret-key buffer format, but its `pt` prefix is unnecessary for the attack. Setting that prefix to zero, while retaining the public-key fields, preserves both witness construction and verifier acceptance in the recorded check. Appendix F documents the source and verifier checks.

Attack cost. Constructing W^* requires the public affine constant, one Mersenne power, and one field addition. Each signature uses the resulting witness in an ordinary BN++ proof; no first-layer matrix–vector products are needed for the zero branch values.

For the AIMER-256f example, the reference timing baseline in Appendix F.3 reports a median of 0.296 ms for public witness reconstruction and 11.610 ms for the attack signer-API call. The latter includes BN++ proof generation, API-internal signing randomness, and repeated public reconstruction. These timings measure separate calls with shared setup work and should not be added.

Table 3. AImMer-256f KAT example.

Item	Value or result
<i>Public inputs</i>	
Original message m_0	0xd81c4d8d...bb556ac8 (33 bytes)
Fresh message m_1	Null-branch forgery: new message.
iv	0x8626ed79...872bfb8f
ct	0x121e2afd...cdb59224
<i>Equivalent witness and relation checks</i>	
t'_0, t'_1, t'_2	All zero
$t'_* = b_{iv}$	0xb2019053...7b85153e
pt'	0x9ee462ce...a9ed308e
Constraint residuals	All Br_j and Out residuals are zero
$AIM2_{iv}(pt')$	0x8b082447...180fcab9 $\neq$ ct
<i>Unmodified verifier</i>	
Honest σ_0 on m_0	Accept
Forged σ_0^* on m_0	Accept; $\sigma_0^* \neq \sigma_0$
Forged σ_1^* on m_1	Accept
Signature size	25,120 bytes

Note. The KAT seed 0x06155023...d2e1ffa1 is reproduction metadata, not an attack input. ... omits intermediate hexadecimal digits; the first and last eight are shown. Full values are in the anonymous repository’s KAT data (Appendix F).

5 Application to Lynxer

Lynxer extends our analysis to VOLEitH signatures built from the Lynx OWF and QuickSilver [18]. For $n \in \{256, 384, 512\}$, a null branch admits a publicly computable relation witness for every public output. The output constraint loses its dependence on that output, so the same witness suffices for all outputs at a fixed IV. Lynx-192 requires an additional linear consistency condition, whereas Lynx-128 excludes the zero assignments used in these constructions.

5.1 Lynx Computation and Constraint Encoding

The public key is (iv, t) and the secret key contains k . The public maps L_0, L_1, L_2 are fixed or IV-derived, depending on the instantiation; L_3 is fixed independently of the IV. The constants C_0, C_1, C_2 are likewise public and obtained as specified by the chosen parameter set. Set

$$x = k + C_0, \quad u = L_0(k) + C_1, \quad z = k + C_2.$$

For a trace $W = (k, v_1, v_2)$, define the linear-layer outputs and the output offset by

$$A = L_1(v_1), \quad B = L_2(v_2), \quad h = L_3(k + v_1 + v_2), \quad w = t + h.$$

The semantic computation is

$$\begin{aligned} v_1 &= S_1(x), & v_2 &= S_2(u), \\ t &= \delta_n(AB, \text{Inv}_0(z)) + h, \end{aligned} \quad (\text{Lynx-Sem})$$

where $\delta_n(x, y) = x$ for $n \in \{128, 192\}$ and $\delta_n(x, y) = xy$ for $n \in \{256, 384, 512\}$. Here $\text{Inv}_0(0) = 0$, matching the specification's totalized inverse convention. Honest witness extension records $W_h = (k, v_1, v_2)$. Figure 3 shows the computation with constants renamed to match the reference code.

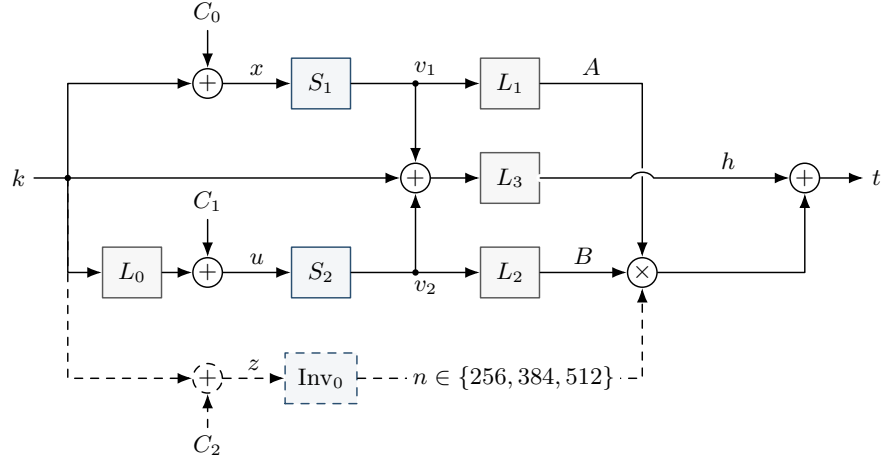


Fig. 3. Lynx evaluation: the two nonlinear branches supply the product factors A, B and the linear term h . The dashed path multiplies the product by $\text{Inv}_0(k + C_2)$ only at $n \in \{256, 384, 512\}$; at $n \in \{128, 192\}$ the output is $AB + h$.

Table 4 gives the nonlinear maps for each dimension. We use $\text{Mer}[e](a) = a^{2^e-1}$ and $\text{Kas}[i](a) = a^{2^{2^i}-2^i+1}$; a superscript -1 denotes the inverse permutation.

Table 4. Lynx nonlinear maps.

n	$S_1(x)$	$S_2(u)$	nonlinear output term
128	$\text{Inv}_0(x)$	$\text{Inv}_0(u)$	AB
192	$\text{Mer}[13]^{-1}(x)$	$\text{Mer}[59](u)$	AB
256	$\text{Mer}[5]^{-1}(x)$	$\text{Mer}[53](u)$	$AB \text{Inv}_0(z)$
384	$\text{Mer}[13]^{-1}(x)$	$\text{Kas}[30](u)$	$AB \text{Inv}_0(z)$
512	$\text{Kas}[6]^{-1}(x)$	$\text{Kas}[90](u)$	$AB \text{Inv}_0(z)$

Table 5 lists the checked branch and output equations and the resulting construction at each dimension.

The output constraint determines whether witness completion depends on the public output. Lynx-192 retains $AB = w$, which leaves a linear consistency condition after the branch assignment in Section 5.5. For $n \in \{256, 384, 512\}$, clearing the denominator $z = k + C_2$ yields the checked equation $AB = wz$. Setting $z = 0$ removes the public output from this equation; the remaining constraints must still be satisfied.

Table 5. Lynx constraints and witness constructions.

n	Br_1	Br_2	Out	Construction
128	$v_1(k + C_0) = 1$	$v_2u = 1$	$AB = w$	null branches excluded
192	$v_1(k + C_0) = v_1^{2^{13}}$	$u^{2^{59}} = uv_2$	$AB = w$	conditional linear solve
256	$v_1(k + C_0) = v_1^{2^5}$	$u^{2^{53}}u = u^2v_2$	$AB = wz$	closed form
384	$v_1(k + C_0) = v_1^{2^{13}}$	$u^{2^{60}}u = u^{2^{30}}v_2$	$AB = wz$	closed form
512	$v_1^{2^6}(k + C_0) = v_1^{2^{12}}v_1$	$u^{2^{180}}u = u^{2^{90}}v_2$	$AB = wz$	closed form

Note. The last column refers to the null branches considered here. For $n = 256$, the specification gives $u^{2^{53}} = uv_2$; the analyzed implementation multiplies both sides by u . Our construction satisfies both forms by evaluating the second branch semantically.

5.2 Null-Branch Witness Construction

For $n \in \{256, 384, 512\}$, the output part of (Lynx-Sem) is

$$w = AB \text{Inv}_0(z),$$

and therefore requires $w = 0$ at $z = 0$. The corresponding output constraint is

$$AB = wz. \quad (\text{Lynx-Out})$$

At $z = 0$, setting $A = 0$ satisfies the checked output equation for every t . Algorithm 2 achieves these assignments through $k' = C_2$ and $v'_1 = 0$, then evaluates the second branch semantically to complete the relation witness. Direct evaluation separately identifies an equivalent witness when $\text{Lynx}_{iv}(k') \neq t$, as in Section 5.4.

Algorithm 2 $\text{RelWit}_{\text{LynxLarge}}(\text{pk} = (iv, t))$

1. Obtain the public maps and constants for iv and the chosen parameter set, loading the fixed map L_3 .
 2. Set $k' \leftarrow C_2$ and $v'_1 \leftarrow 0$.
 3. Compute $v'_2 \leftarrow S_2(L_0(k') + C_1)$.
 4. Return $W^* = (k', v'_1, v'_2)$.
-

Theorem 2 (Lynx witness construction). Fix $n \in \{256, 384, 512\}$ and a corresponding speed or size variant of Lynxer [18] with the constraints in Table 5. For every $\mathbf{pk} = (\text{iv}, t) \in \mathcal{P}_\lambda^{\text{syn}}$, Algorithm 2 deterministically computes, in polynomial time, a witness $W^* \in \mathcal{W}_\lambda$ satisfying

$$R_C(\mathbf{pk}, W^*) = 1.$$

This holds for any public linear maps, including fixed and IV-derived matrices. Under the completeness assumption of Section 3.3, there exists a public-key-only PPT adversary making no signing queries with

$$\Pr[\text{Verify}(\mathbf{pk}, m, \sigma^*) = 1] \geq 1 - \delta_{\text{comp}}(\lambda),$$

for each honestly generated $\mathbf{pk} \in \mathcal{P}_\lambda^{\text{hon}}$ and each chosen message m . The probability includes fresh proving randomness and random-oracle randomness as specified in Appendix B.2; $\mathbf{pk}$ and m are fixed. Here σ^* denotes the adversary's output, with exhaustion of its grinding budget counted as failure.

Proof. The construction uses only public maps and constants. For Lynx-256 and Lynx-384, $v'_1 = 0$ turns Br_1 into $0 = 0$. For Lynx-512, both sides of its inverse-Kasami Br_1 equation likewise vanish. The value $v'_2 = S_2(L_0(k') + C_1)$ is a semantic second-branch output, so it satisfies Br_2 , including both Lynx-256 forms.

Finally, $k' = C_2$ gives $z' = 0$, while linearity gives $A' = L_1(v'_1) = 0$. Consequently, the output constraint (Lynx-Out) becomes

$$A'B' = w'z' = 0$$

for every public output t , independently of the ranks or distribution of the public maps. All three residuals therefore vanish. The constructor uses public field operations, linear maps, and one evaluation of S_2 , so it is deterministic and polynomial-time. Construction succeeds for each honestly generated key. The per-key conclusion of Proposition 1 then gives the forgery bound, including the grinding failure term in δ_{comp} . $\square$

5.3 Gap in the Lynxer Security Reduction

Lynxer's decision-soundness reduction requires language inclusion (LI): every satisfiable constraint instance must admit a semantic OWF preimage. A witness that decodes to an invalid input alone does not refute this property; the public output must have no preimage in the declared domain.

For fixed $\mathbf{pk}$, let $f_{\text{Br}_1}(W)$, $f_{\text{Br}_2}(W)$, and $f_{\text{Out}}(W)$ denote the left-hand side minus the right-hand side of the corresponding equations in Table 5. The instantiated QuickSilver relation is the encoded relation R_C , namely

$$R_{\text{QS}}(\mathbf{pk}, W) = 1 \iff f_{\text{Br}_1}(W) = f_{\text{Br}_2}(W) = f_{\text{Out}}(W) = 0.$$

The required inclusion is

$$(\exists W : R_{\text{QS}}((iv, t), W) = 1) \implies (\exists k : R_{\text{F}}((iv, t), k) = 1).$$

Here R_{F} uses the OWF game’s declared input domain. For $n \in \{256, 384, 512\}$, Theorem 2 supplies a satisfying QuickSilver witness for every syntactically valid output. Each restricted input domain discussed in Appendix B.5 has fewer elements than $\mathbb{E}$, so its semantic image omits some outputs. At those outputs, the constraint relation remains satisfiable although no semantic preimage exists. Thus language inclusion fails under either declared restricted-domain interpretation.

This failure affects the application of FAESTv2 Theorems 9.24 and 9.36 in Lynxer’s EUF-CMA theorem (Theorem 1) [18,3]. FAEST’s OWF instance (x, y) corresponds to (iv, t) , and its secret input to k . The $\text{Adv}^{\text{PRG-OWF}}$ game distinguishes an honestly generated output from a uniform field element. In the uniform-key game, Theorem 9.24 bounds acceptance outside the semantic image by decision soundness. For the instantiated Lynxer relation, however, such outputs are satisfiable instances, so this soundness bound does not apply. Theorem 9.36 subsequently upgrades EUF-KO to EUF-CMA by simulating signing queries; it does not establish the missing inclusion.

Verifier boundary. Witness corrections bind the supplied trace through IT-MACs to all three QuickSilver equations, whose residuals vanish. The encoded output relation imposes no condition $w' = 0$ when $A' = z' = 0$. Upon successful counter grinding, the public prover produces a transcript accepted by the unmodified verifier. Appendix A gives the transcript details.

5.4 Validation and Cost

The archived KAT records cover one fixed-matrix key for each of Lynxer-192f, Lynxer-256f, Lynxer-384f, and Lynxer-512f in the C reference implementation.⁵ Each record contains an honest control, a distinct forgery on the original message, and a forgery on a fresh message, all accepted by the unmodified verifier. These checks cover the fixed-matrix speed variants; Theorem 2 establishes the broader algebraic scope at dimensions 256, 384, and 512.

For the Lynxer-256f example, all three constraint residuals vanish, while direct evaluation gives $\text{Lynx}_{iv}(k') \neq t$. The accepted signatures demonstrate that this equivalent witness can be used by the original proving computation. Table 8 in Appendix F.2 preserves the detailed values. Its KAT seed is reproduction metadata, not an attack input.

Table 6 summarizes the four Lynxer examples and their signature lengths. Direct OWF evaluation separately confirms semantic invalidity at each constructed witness’s decoded input. The Lynxer-192f example uses a selected key satisfying the consistency conditions of Section 5.5. Appendix F documents the records, drivers, and verifier checks.

⁵ Lynxer reference snapshot, commit 3fd635a.

Table 6. KAT validation of Lynxer equivalent witnesses. Each example has one control signature and two forgeries, on the original and a fresh message. Signature lengths are in bytes.

Target	Matrices	Accepted	Bytes
Lynxer-192f	fixed	3/3	8,348
Lynxer-256f	fixed	3/3	14,772
Lynxer-384f	fixed	3/3	32,228
Lynxer-512f	fixed	3/3	57,364

Accepted reports accepted/checked signatures. Direct Lynx evaluation at each forged trace’s decoded input differs from the public output. The Lynxer-192f row uses a selected key satisfying both public systems. Appendix F.2 identifies the full records and rebuild procedure.

Attack cost. For $n \in \{256, 384, 512\}$, witness construction uses one S_2 evaluation in addition to public-map derivation and application. The Lynx-192 cost includes the linear systems described in Section 5.5. A successfully constructed witness can be reused across messages; each signature invokes the ordinary VOLEitH/QuickSilver prover, including counter grinding.

The reference timing baseline in Appendix F.3 reports a median of 0.399 ms for fixed-matrix Lynxer-256f witness reconstruction on the example key. The corresponding `faest_sign` call has median 14.172 ms, with ρ generated before the timed interval. This timing excludes randomness generation, unlike the AIMer signer-API measurement, so the two values do not represent matched end-to-end costs.

5.5 The Lynx-192 Attack and the Lynx-128 Boundary

Lynx-192. The constructor first selects a canonical solution of one public linear system, then tests whether a second system can match the public output.

At dimension 192, the checked equation Br_2 is

$$u^{2^{59}} = uv_2, \quad u = L_0(k) + C_1.$$

Setting $u = 0$ frees v_2 , but **Out** remains the direct equation

$$L_1(v_1)L_2(v_2) + L_3(k + v_1 + v_2) = t.$$

If $L_0(k) = C_1$ is inconsistent, the constructor fails. Otherwise, it selects the canonical solution returned by Gaussian elimination and computes $v_1 = S_1(k + C_0)$. Multiplication by $L_1(v_1)$ is then a known $\mathbb{K}$ -linear map. Expanding the L_3 term gives the completion system $T_k(v_2) = t + L_3(k + v_1)$. For this selected k , success is exactly the rank condition

$$\text{rank}(T_k) = \text{rank}[T_k \mid t + L_3(k + v_1)], \quad T_k = \mathbf{M}_{L_1(v_1)} \circ L_2 + L_3.$$

This is the completion system of Section 3.2: M is the binary matrix of T_k , N is the $n \times n$ identity, and b is the coordinate vector of $L_3(k + v_1)$. The vectors $\mathbf{v}$

and $\mathbf{y}$ encode v_2 and t , respectively. For the selected k , the attainable outputs therefore form the affine set $L_3(k + v_1) + \text{im}(T_k)$.

For the archived fixed-matrix variant, let p_{192}^{fixed} denote the canonical-solution constructor's success probability over honest key generation, as defined in Appendix D. Proposition 1 gives forgery probability at least $p_{192}^{\text{fixed}}(1 - \delta_{\text{comp}})$. This bound averages over KeyGen; it does not assert success for every public key. Gaussian elimination solves the two public $n \times n$ systems in $O(n^3)$ bit operations each. Failure for the selected k does not exclude successful completion for another solution; Appendix D distinguishes this algorithm from exhaustive search.

The fixed-matrix Lynx-192f example demonstrates feasibility on a key satisfying both systems. It does not estimate p_{192}^{fixed} , which remains unmeasured.

Lynx-128. The checked branch equations are

$$v_1(k + C_0) = 1, \quad v_2(L_0(k) + C_1) = 1.$$

They force both S-box inputs and outputs to be nonzero in every satisfying trace. In particular, they reject both $u = 0$, used at dimension 192, and $v'_1 = 0$, used at the larger dimensions. This excludes the zero assignments used by our constructions, without establishing security against other attacks.

6 Boundary Cases of Null-Branch Analysis

The attacks in Sections 4 and 5 exploit zero branches that admit semantically invalid relation witnesses. We now examine encodings in which auxiliary freedom preserves the reconstructed output or the constraints exclude zero inputs. The norm-compression template used by FAEST v2 and GALAS illustrates the first case [3,27]. AIMer v3.0 illustrates the second by excluding the zero branch exploited in our AIM2 attack.

6.1 The Compressed Inversion Relation

Let $m \geq 1$, set $q = 2^m$, and view $\mathbb{F}_{q^2}$ as a quadratic extension of $\mathbb{F}_q$. For $x \neq 0$, the norm of its inverse is

$$c = N_{\mathbb{F}_{q^2}/\mathbb{F}_q}(x^{-1}) = x^{-(q+1)} \in \mathbb{F}_q.$$

Representing c requires m witness bits, compared with $2m$ bits for a full inverse. The encoding checks a consistency equation and reconstructs the inverse y as

$$cx^2x^q = x, \quad y = x^qc. \quad (\text{Norm})$$

Both $x \mapsto x^2$ and $x \mapsto x^q$ are $\mathbb{F}_2$ -linear. The consistency check therefore has degree at most three in the binary witness coordinates, while the reconstruction has degree at most two.

For $x \neq 0$, dividing the consistency equation by x gives $cx^{q+1} = 1$, hence $c = x^{-(q+1)}$ and $y = x^{-1}$. At $x = 0$, the consistency equation admits every $c \in \mathbb{F}_q$, while the reconstruction still forces $y = 0$. Conversely, every totalized inverse pair has such an encoding: use the inverse norm at nonzero inputs and any c at zero. Thus

$$(\exists c \in \mathbb{F}_q : cx^2x^q = x \wedge y = x^qc) \iff y = \text{Inv}_0(x).$$

The encoding is sound and complete for the totalized inverse relation, although its auxiliary witness is not unique at zero. This freedom does not produce a semantically invalid input-output pair. Thus, non-uniqueness of auxiliary values alone does not imply a violation of relation-encoding soundness.

6.2 Norm Compression in FAEST v2 and GALAS

FAEST v2. FAEST uses $\mathbb{F}_{2^8}$ with subfield $\mathbb{F}_{2^4}$, so $q = 16$ and $c = x^{-17}$ for nonzero S-box inputs. Selected AES/Rijndael layers commit to four inverse-norm bits and reconstruct the inverse and its Frobenius conjugates. The specified `InvNorm` routine returns zero at input zero, whereas `InvNormConstraints` checks $cx^2x^{16} = x$ [3, Figs. 6.5 and 6.19]. At $x = 0$, all 16 subfield values satisfy this check, and every reconstructed conjugate $(x^{16}c)^{2^j}$ remains zero. The subsequent S-box affine transformation therefore receives the same inverse value [3, Fig. 6.20].

GALAS. GALAS applies this template over $\mathbb{F}_{2^n}$ for $n \in \{128, 192, 256\}$ [27]. Its three first-layer inversions use $n/2$ -bit witnesses in $\mathbb{F}_{2^{n/2}}$ and reconstruct their outputs as x^qc , where $q = 2^{n/2}$. The analyzed manuscript derives this template for nonzero inputs and uses key-generation rejection sampling to exclude zero S-box inputs [27]. Evaluating the local equations at zero still yields $y = 0$ for every subfield value of c . This observation concerns the local totalized-inverse relation; it does not establish enforcement of GALAS’s nonzero-input domain. As in Section 2, a KeyGen restriction must also be enforced by the proved relation when it forms part of the declared OWF domain.

6.3 Non-Unique Witness Encodings in FAEST v2

For FAEST v2, auxiliary-witness non-uniqueness can yield distinct accepted signatures under fixed signing randomness. This construction requires the secret key and illustrates representation non-uniqueness, not an EUF-CMA or SUF-CMA forgery. Suppose the key’s OWF computation contains a zero input to a norm-compressed S-box. The signer may replace its canonical auxiliary value by another subfield element. Recomputing the transcript yields a distinct accepted signature for each such choice whose grinding loop terminates successfully. Appendix C.2 gives the construction and the signature-distinctness argument.

The modified witnesses retain the same decoded OWF preimage, so they are not equivalent witnesses in the sense of Definition 2. The prescribed `Sign`

algorithm remains deterministic for fixed randomness; the alternative signatures result from changing its witness extension. The example therefore separates semantic correctness from uniqueness of the accepted representation.

6.4 Excluding Zero Inputs in AIMer v3.0

AIMer version 3.0 replaces AIM2 with AIM3, which uses enhanced Mersenne S-boxes [11]. Over $\mathbb{E}$, the S-box is $S[e](x) = x^{2^e-1} + x^{2^n-2}$. Its specified branch relation is

$$xy = x^{2^e} + 1. \quad (\text{AIM3-Br})$$

At $x = 0$, the relation would require $0 = 1$, so every satisfying pair has $x \neq 0$. Dividing by x gives $y = x^{2^e-1} + x^{-1} = S[e](x)$; conversely, every nonzero input with its S-box output satisfies the equation. The branch relation therefore exactly encodes the enhanced S-box on its declared nonzero input domain.

Key generation invokes `AIM3_NoZeroInpSB` and resamples whenever an S-box input is zero; the MPC computation uses the same branch relation [11, Algs. 5–6]. Crucially, the equation itself enforces nonzero inputs, independently of key-generation rejection sampling. This excludes the zero branch exploited in our AIM2 attack, without constituting a comprehensive security analysis of AIMer v3.0.

AIMer v3.0 changes the primitive and restricts its input domain. Section 7 instead repairs the AIM2 and Lynx encodings while preserving their totalized semantics, including zero inputs.

7 Semantics-Preserving Relation Repairs

We construct low-degree relation encodings that preserve the full AIM2 and Lynx computations, including zero inputs. The encodings exclude semantically invalid witnesses while retaining every semantic preimage.

This requires relation-encoding soundness and the complementary requirement of relation-encoding completeness, both on the same declared domain.

Definition 3 (Relation-encoding completeness). *The relation R_C is complete with respect to R_F under Dec on $\mathcal{D}$ if, for every λ , every $\mathbf{pk} \in \mathcal{D}_\lambda$, and every $x \in \mathcal{X}_\lambda$,*

$$R_F(\mathbf{pk}, x) = 1 \implies \exists W \in \mathcal{W}_\lambda : R_C(\mathbf{pk}, W) = 1 \wedge \text{Dec}(W) = x.$$

Together, the two properties give $\text{Dec}(\mathcal{S}_C(\mathbf{pk})) = \mathcal{S}_F(\mathbf{pk})$, excluding equivalent witnesses as defined in Section 3.1. For each repaired relation, we extend the decoder to ignore the new auxiliary tuple η , writing $\text{Dec}(W, \eta) = \text{Dec}(W)$. Relation-encoding completeness is an existence property; our constructions additionally provide efficient witness extension with explicit auxiliary assignments at zero inputs.

7.1 Zero-Safe Graph Encodings and Composition

We use the totalized-inverse convention from Section 2. Each encoding enforces the intended zero-case output while preserving the nonzero computation. For example, $t^{2^e} = ut$ leaves u free at $t = 0$. Adding $u = tr$ forces $u = 0$ there and admits $r = u/t$ otherwise.

Proposition 2 (Zero-safe power graph). *For any field $\mathbb{F}$, elements $t, u \in \mathbb{F}$, and integer $e \geq 1$,*

$$\exists r \in \mathbb{F}: \quad t^{2^e} = ut, \quad u = tr \quad \Longleftrightarrow \quad u = t^{2^e-1}.$$

Proposition 3 (Zero-safe inverse graph). *For x, y in a field,*

$$x^2y = x, \quad xy^2 = y \quad \Longleftrightarrow \quad y = \text{Inv}_0(x).$$

For the Kasami exponent $d_i = 2^{2i} - 2^i + 1$, a compressed equation also needs a companion at zero.

Proposition 4 (Zero-safe Kasami graph). *For integer $i \geq 1$, over $\mathbb{E}$ the equations*

$$x^{2^i}y = x^{2^{2i}}x, \quad y^{2^i}y = x^{2^{3i}}x$$

hold if and only if $y = x^{d_i}$.

The large-Lynx output needs a zero-safe encoding of a product followed by totalized division. The next repair remains quadratic; the auxiliary value p avoids a cubic product.

Proposition 5 (Zero-safe product–division graph). *For A, B, z, w in a binary field,*

$$w = AB \text{Inv}_0(z)$$

if and only if there exist p, r satisfying

$$p = AB, \quad zp = z^2w, \quad w = zr. \quad (\text{Div-Rep})$$

The squaring map in the middle equation is $\mathbb{F}_2$ -linear, so all three equations are quadratic authenticated-field equations.

Proof. If $z \neq 0$, the first two equations give $p = AB$ and $p = zw$, hence $w = AB/z$. If $z = 0$, the last equation forces $w = 0$. The middle equation then imposes no additional restriction on p , while the first still enforces $p = AB$. Conversely, given a semantic evaluation, set $p = AB$ and choose $r = w/z$ at $z \neq 0$, or $r = 0$ at $z = 0$. All three equations hold.

Why the output equation must be replaced. Adding constraints can remove spurious witnesses, but cannot restore semantic assignments already excluded by the original equation. The totalized computation contains the point $(A, B, z, w) = (1, 1, 0, 0)$, which violates $AB = wz$. Appending $w = zr$ would enforce $w = 0$ at zero but retain the incompatible condition $AB = 0$. Replacing the cleared equation with (Div-Rep) preserves every semantic numerator at $z = 0$.

Composition. The local encodings are connected through a shared semantic trace. A gate output and its subsequent uses share the same witness coordinate; only the new auxiliary variables are fresh. Fixing the decoded input therefore fixes the semantic values successively, and the final output check binds this computation to the public output. Auxiliary values need not be unique, provided every satisfying assignment determines the same semantic computation for the decoded input.

Conversely, each semantic execution extends to a satisfying trace using the local auxiliary assignments, whose freshness ensures compatible choices across gates. Appendix E formalizes both directions in Theorem 6 and gives the remaining local proofs. This composition argument concerns satisfaction of the full residual system; randomized batching is covered separately by the proof system’s soundness analysis.

7.2 A Sound and Complete Relation Encoding for AIMer

For AIMer v2.0, define R_{AIMer}^0 by retaining the AIM2 branch and output constraints and the affine dependency from Section 4. Add fresh auxiliary values $r_0, \dots, r_{\ell-1}, r_*$ satisfying

$$\mathbf{pt} + \gamma_j = t_j r_j \quad (0 \leq j < \ell), \quad \mathbf{ct} + \mathbf{pt} = t_* r_*. \quad (\text{AIM2-Rep})$$

The value t_* remains derived from the original witness coordinates through the affine layer; it is not an additional independent witness input.

Theorem 3 (AIMer repair correctness). *Fix an AIMer v2.0 parameter set and a syntactically valid public key $\mathbf{pk} = (\mathbf{iv}, \mathbf{ct})$ with defined IV derivation. For any candidate trace $W = (\mathbf{pt}, t_0, \dots, t_{\ell-1}, t_*)$, write $\eta = (r_0, \dots, r_{\ell-1}, r_*)$. Then*

$$\exists \eta \in \mathbb{E}^{\ell+1} : \quad R_{\text{AIMer}}^0(\mathbf{pk}, (W, \eta)) = 1$$

if and only if W is a semantic AIM2 trace with output $\mathbf{ct}$. Consequently, R_{AIMer}^0 is a sound and complete encoding of the AIM2 relation.

Proof. The original first-layer equation is $t_j^{2^{e_j}} = (\mathbf{pt} + \gamma_j)t_j$. Together with (AIM2-Rep), Proposition 2 gives $\mathbf{pt} + \gamma_j = t_j^{2^{e_j}-1}$ for every branch. The same argument applied to the final equation gives $\mathbf{ct} + \mathbf{pt} = t_*^{2^{e_*}-1}$. The checked affine equality fixes $t_* = b_{\mathbf{iv}} + \sum_j A_{\mathbf{iv},j} t_j$. The submitted Mersenne maps are permutations at each parameter set (Appendix E.1), so the displayed equalities are exactly the AIM2 trace equations. The reverse direction follows from Proposition 2, choosing each quotient witness as in its proof. Theorem 6 completes the claim.

This exact total-graph repair adds $\ell + 1$ hidden field elements and $\ell + 1$ field-valued quadratic equations: three of each for the 128- and 192-bit variants, and four of each for the 256-bit variants.

7.3 A Sound and Complete Relation Encoding for Lynx

We use the trace values x, u, z, A, B, h, w defined in Section 5. For Lynx-128, replace each strict inverse equation by the corresponding pair in Proposition 3. This restores completeness for the totalized-inverse semantics; it does not address a forgery established in this paper. For the other parameters, retain the branch equations in Table 5 and add the following graph completions:

n	additional branch equations	
192	$x = v_1 r_1, \quad v_2 = ur_2$	(Lynx-Rep)
256	$x = v_1 r_1, \quad v_2 = ur_2$	
384	$x = v_1 r_1, \quad v_2^{2^{30}} v_2 = u^{2^{90}} u$	
512	$x^{2^6} x = v_1^{2^{18}} v_1, \quad v_2^{2^{90}} v_2 = u^{2^{270}} u.$	

For $n = 128$ and $n = 192$, retain the direct output equation $AB = w$. For the three larger dimensions, replace the cleared equation $AB = wz$ by the three equations in (Div-Rep).

Write $R_{\text{Lynx},n}^{0,\nu}$ for the resulting relation, where $\nu \in \{\text{spec}, \text{snap}\}$ identifies the base equations. At $n = 256$, the retained second-branch equation is $u^{2^{53}} = uv_2$ in the specification and $u^{2^{53}} u = u^2 v_2$ in the snapshot. Both receive the companion $v_2 = ur_2$; at other dimensions, the two labels denote the same base relation. The power maps in Table 4 are permutations on their respective fields, as verified in Appendix E.1.

Theorem 4 (Lynx repair correctness). *Fix $n \in \{128, 192, 256, 384, 512\}$, $\nu \in \{\text{spec}, \text{snap}\}$, and public maps and constants defining both the repaired relation and Lynx_{iv} . For every $k, v_1, v_2, t \in \mathbb{E} = \mathbb{F}_{2^n}$, with η ranging over the auxiliary tuples of that relation,*

$$\exists \eta : R_{\text{Lynx},n}^{0,\nu}((\text{iv}, t), (k, v_1, v_2, \eta)) = 1 \iff \begin{cases} v_1 = S_1(k + C_0), \\ v_2 = S_2(L_0(k) + C_1), \\ t = \text{Lynx}_{\text{iv}}(k). \end{cases}$$

Hence the repaired relation is a sound and complete encoding of the total Lynx graph.

Proof. Propositions 2–4 make every repaired branch equivalent to its semantic map; Appendix E.1 justifies every inverse-permutation use. This includes both Lynx-256 versions, where $v_2 = ur_2$ supplies the snapshot equation’s missing zero case. Proposition 5 gives the exact large-parameter output graph, while $AB = w$ is already exact at $n \in \{128, 192\}$. Substitution and Theorem 6 give both directions. The complete parameter-by-parameter derivation appears in Appendix E.

For $n = 128, 192, 256, 384, 512$, the respective algebraic increments $(\Delta W, \Delta E)$ are $(0, 2), (2, 2), (4, 4), (3, 4), (2, 4)$, relative to the base relations in Table 5. Here ΔW counts additional hidden field elements and ΔE counts the net increase in field-valued quadratic equations. Appendix E gives the accounting details.

7.4 Requirements for Proof-System Integration

Witness extension computes the semantic trace and fresh auxiliaries, using the explicit zero-case assignments whenever a quotient would be undefined. The auxiliary values remain private witness coordinates, shared or authenticated alongside the original witness. Applying the proof system to the enlarged relation requires updating witness dimensions, randomness allocation, and the constraint checks used by prover and verifier. Its completeness, soundness, and zero-knowledge guarantees must apply to this relation with the updated parameters.

All residuals remain quadratic in the base-field coordinates, matching the multiplication-check machinery of Section 2. The reported algebraic increments do not directly measure signature size or runtime. Proof costs also depend on hidden products, witness corrections, batching, and repetitions; for example, $zp = z^2w$ contributes one residual with two hidden multiplications.

An instantiation with a restricted input domain must also enforce membership in that domain. The reported algebraic overhead excludes these additional constraints. Appendix E gives the conditions for retaining the original conditional equations in such a design.

These results establish soundness and completeness of the repaired relation encodings. Signature security additionally depends on the underlying OWF assumption and the security of the instantiated proof system and Fiat–Shamir transformation.

8 Conclusion

Our null-branch witness attack yields public-key-only forgery against AIMer v2.0 across all parameter sets. For every honestly generated public key, it forges signatures on arbitrary messages with probability one, without signing queries or OWF inversion. The attack uses the original proving procedure and unmodified verifier. Complementary Lynxer results extend the analysis to the vulnerable VOLEitH variants. Reference-implementation tests confirm accepted forgeries whose decoded witness inputs fail the semantic OWF check.

These attacks expose a gap between satisfaction of the encoded relation and validity of the semantic OWF computation. Proof-system soundness must be complemented by the relation-to-OWF property required by the reduction. Our quadratic encodings establish relation-encoding soundness and completeness for AIM2 and Lynx, including zero inputs, with efficient witness extension. These relation-level guarantees do not alone establish signature security, which also depends on OWF hardness and the security of the proof system and Fiat–Shamir transformation. Relation encodings therefore require explicit auditing on the reduction’s declared domain.

Acknowledgments

We thank the AIMer team for constructive discussions following our disclosure of the attack on AIM2-based AIMer. The team confirmed the validity of the attack. The subsequent AIM3 design excludes the null branch exploited here, as discussed in Section 6.4.

We also thank the Lynxer team for confirming the validity of our attacks on the affected Lynxer variants following our disclosure.

We used ChatGPT to assist with grammar checking and language polishing during the preparation of this manuscript.

References

1. Alagic, G., Bros, M., Ciadoux, P., Dang, Q., Dang, T.H., Kelsey, J., Lichtinger, J., Liu, Y.K., Miller, C., Moody, D., Peralta, R., Perlner, R., Robinson, A., Silberg, H., Smith-Tone, D., Waller, N.: Status report on the second round of the additional digital signature schemes for the NIST post-quantum cryptography standardization process. NIST Internal Report 8610, National Institute of Standards and Technology (2026). <https://doi.org/10.6028/NIST.IR.8610>, published May 14, 2026
2. Albrecht, M.R., Rechberger, C., Schneider, T., Tiessen, T., Zohner, M.: Ciphers for MPC and FHE. In: Oswald, E., Fischlin, M. (eds.) *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I. Lecture Notes in Computer Science, vol. 9056, pp. 430–454. Springer (2015). https://doi.org/10.1007/978-3-662-46800-5_17
3. 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. NIST Post-Quantum Cryptography Additional Digital Signature Round 2 submission (2025), <https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/round-2/spec-files/faest-spec-round2-web.pdf>
4. 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.: Shorter, tighter, FAESTer: Optimizations and improved (QROM) analysis for VOLE-in-the-head signatures. In: Tauman Kalai, Y., Kamara, S.F. (eds.) *Advances in Cryptology - CRYPTO 2025 - 45th Annual International Cryptology Conference*, Santa Barbara, CA, USA, August 17–21, 2025, Proceedings, Part VI. Lecture Notes in Computer Science, vol. 16005, pp. 124–156. Springer (2025). https://doi.org/10.1007/978-3-032-01887-8_5
5. 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: Handschuh, H., Lysyanskaya, A. (eds.) *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference*, CRYPTO 2023, Santa Barbara, CA, USA, August 20–24, 2023, Proceedings, Part V. Lecture Notes in Computer Science, vol. 14085, pp. 581–615. Springer (2023). https://doi.org/10.1007/978-3-031-38554-4_19
6. Baum, C., Nof, A.: Concretely-efficient zero-knowledge arguments for arithmetic circuits and their application to lattice-based cryptography. In: Kiayias, A., Kohlweiss, M., Wallden, P., Zikas, V. (eds.) *Public-Key Cryptography - PKC 2020 - 23rd IACR International Conference on Practice and Theory of Public-Key Cryptography*, Edinburgh, UK, May 4–7, 2020, Proceedings, Part I. Lecture Notes in Computer Science, vol. 12110, pp. 495–526. Springer (2020). https://doi.org/10.1007/978-3-030-45374-9_17
7. Biryukov, A., García Fernández, P., Udovenko, A.: Magic Pot: Cryptanalysis of full AIM2 in the standard and related-/reused-key settings using new elimination framework. In: Daemen, J., Thomé, E. (eds.) *Advances in Cryptology - EUROCRYPT 2026 - 45th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Rome, Italy, May 10–14, 2026, Proceedings, Part VI. Lecture Notes in Computer Science, vol. 16546, pp. 63–90. Springer (2026). https://doi.org/10.1007/978-3-032-25333-0_3

8. Chase, M., Derler, D., Goldfeder, S., Kales, D., Katz, J., Kolesnikov, V., Orlandi, C., Ramacher, S., Rechberger, C., Slamanig, D., Wang, X., Zaverucha, G.: The Picnic signature algorithm specification. Version 3.0 (2020), <https://github.com/microsoft/Picnic/blob/master/spec/spec-v3.0.pdf>
9. 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: Thuraisingham, B., Evans, D., Malkin, T., Xu, D. (eds.) Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017. pp. 1825–1842. ACM (2017). <https://doi.org/10.1145/3133956.3133997>
10. Cho, J., Ha, J., Kim, S., Kwon, J., Lee, B., Lee, J., Lee, J., Lee, S., Moon, D., Son, M., Yoon, H.: The AIMer signature scheme. Specification, version 2.0 (2026), https://aimer-signature.org/docs/AIMer_Specification_v260130.pdf, dated January 30, 2026
11. Cho, J., Ha, J., Kim, S., Kwon, J., Lee, B., Lee, J., Lee, J., Lee, S., Moon, D., Son, M., Yoon, H.: The AIMer signature scheme. Specification, version 3.0 (2026), [https://aimer-signature.org/docs/AIMer-specification-v3%20\(Eng\).pdf](https://aimer-signature.org/docs/AIMer-specification-v3%20(Eng).pdf), dated July 31, 2026
12. Coglio, A., McCarthy, E., Smith, E., Chin, C., Gaddamadugu, P., Dellepère, M.: Compositional formal verification of zero-knowledge circuits. Cryptology ePrint Archive, Paper 2023/1278 (2023), <https://eprint.iacr.org/2023/1278>
13. Dobraunig, C., Kales, D., Rechberger, C., Schofnegger, M., Zaverucha, G.: Shorter signatures based on tailor-made minimalist symmetric-key crypto. In: Yin, H., Stavrou, A., Cremers, C., Shi, E. (eds.) Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7–11, 2022. pp. 843–857. ACM (2022). <https://doi.org/10.1145/3548606.3559353>
14. Feneuil, T., Rivain, M.: Threshold computation in the head: Improved framework for post-quantum signatures and zero-knowledge arguments. J. Cryptol. **38**(3), 28 (2025). <https://doi.org/10.1007/S00145-025-09543-8>
15. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings. Lecture Notes in Computer Science, vol. 263, pp. 186–194. Springer (1987). https://doi.org/10.1007/3-540-47721-7_12
16. Giacomelli, I., Madsen, J., Orlandi, C.: ZKBoo: Faster zero-knowledge for Boolean circuits. In: Holz, T., Savage, S. (eds.) 25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10–12, 2016. pp. 1069–1083. USENIX Association (2016), <https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/giacomelli>
17. Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Zero-knowledge proofs from secure multiparty computation. SIAM J. Comput. **39**(3), 1121–1152 (2009). <https://doi.org/10.1137/080725398>
18. Jiao, L., Yang, H., Cui, H., He, Y., Hao, Y., Guo, X., Zheng, Q., Zhang, J., Yu, Y., Yang, K.: Lynx: Symmetric primitive for shorter and faster VOLE-in-the-head signatures. Cryptology ePrint Archive, Paper 2026/1099 (2026), <https://eprint.iacr.org/2026/1099>
19. Kales, D., Zaverucha, G.: Efficient lifting for shorter zero-knowledge proofs and post-quantum signatures. Cryptology ePrint Archive, Paper 2022/588 (2022), <https://eprint.iacr.org/2022/588>

20. Katz, J., Kolesnikov, V., Wang, X.: Improved non-interactive zero knowledge with applications to post-quantum signatures. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018. pp. 525–537. ACM (2018). <https://doi.org/10.1145/3243734.3243805>
21. Kim, S., Ha, J., Son, M., Lee, B., Moon, D., Lee, J., Lee, S., Kwon, J., Cho, J., Yoon, H., Lee, J.: AIM: Symmetric primitive for shorter signatures with stronger security. In: Meng, W., Jensen, C.D., Cremers, C., Kirda, E. (eds.) Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, CCS 2023, Copenhagen, Denmark, November 26-30, 2023. pp. 401–415. ACM (2023). <https://doi.org/10.1145/3576915.3616579>
22. Kolozyan, A., Vandenbogaerde, B., Swalens, J., Hoste, L., Chaliasos, S., De Roover, C.: Language-agnostic detection of computation-constraint inconsistencies in ZKP programs via value inference. In: 47th IEEE Symposium on Security and Privacy, S&P 2026. pp. 3091–3110. IEEE (2026). <https://doi.org/10.1109/SP63933.2026.00207>
23. Korean Post-Quantum Cryptography Research Group: Final results of round 2 of the KpqC competition (Jan 2025), https://www.kpqc.or.kr/contents/03_exhibit/board.html?board_id=board_competition&mode=view&no=9, announcement dated January 16, 2025; in Korean
24. Korean Post-Quantum Cryptography Research Group: KpqC algorithms: Final specification documents (2026), https://www.kpqc.or.kr/contents/03_exhibit/sub_03.html, published January 30, 2026
25. Liu, F., Mahzoun, M., Meier, W.: Modelling ciphers with overdefined systems of quadratic equations: Application to Friday, Vision, RAIN and Biscuit. In: Chung, K.M., Sasaki, Y. (eds.) Advances in Cryptology - ASIACRYPT 2024 - 30th International Conference on the Theory and Application of Cryptology and Information Security, Kolkata, India, December 9-13, 2024, Proceedings, Part VII. Lecture Notes in Computer Science, vol. 15490, pp. 424–456. Springer (2025). https://doi.org/10.1007/978-981-96-0941-3_14
26. Liu, F., Mahzoun, M., Øygarden, M., Meier, W.: Algebraic attacks on RAIN and AIM using equivalent representations. IACR Trans. Symmetric Cryptol. **2023**(4), 166–186 (2023). <https://doi.org/10.46586/TOSC.V2023.I4.166-186>
27. Liu, G., Sun, Y., Peng, Z., Ren, C., Liu, G., Lin, J., Jia, K., Wang, M.: GALAS: Compact VOLE-in-the-head signatures from subfield compression. In: Proceedings of the 2026 ACM SIGSAC Conference on Computer and Communications Security. ACM (2026), <https://www.sigsac.org/ccs/CCS2026/program/accepted-papers.html>, accepted in Cycle 1; to appear
28. National Institute of Standards and Technology: Round 1 additional digital signature candidates (2023), <https://csrc.nist.gov/projects/pqc-dig-sig/round-1-additional-signatures>
29. Pailoor, S., Chen, Y., Wang, F., Rodríguez, C., Van Geffen, J., Morton, J., Chu, M., Gu, B., Feng, Y., Dillig, I.: Automated detection of under-constrained circuits in zero-knowledge proofs. Proc. ACM Program. Lang. **7**(PLDI), 1510–1532 (2023). <https://doi.org/10.1145/3591282>
30. Roy, L.: SoftSpokenOT: Quieter OT extension from small-field silent VOLE in the Minicrypt model. In: Dodis, Y., Shrimpton, T. (eds.) Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15-18, 2022, Proceedings, Part I. Lecture

- Notes in Computer Science, vol. 13507, pp. 657–687. Springer (2022). https://doi.org/10.1007/978-3-031-15802-5_23
31. Stephens, J., Pailoor, S., Dillig, I.: Automated verification of consistency in zero-knowledge proof circuits. In: Piskac, R., Rakamarić, Z. (eds.) Computer Aided Verification, CAV 2025, Part I. Lecture Notes in Computer Science, vol. 15931, pp. 315–338. Springer (2025). https://doi.org/10.1007/978-3-031-98668-0_16
 32. Takahashi, H., Kim, J., Jana, S., Yang, J.: zkFuzz: Foundation and framework for effective fuzzing of zero-knowledge circuits. In: 47th IEEE Symposium on Security and Privacy, S&P 2026. pp. 3055–3074. IEEE (2026). <https://doi.org/10.1109/SP63933.2026.00049>
 33. Wen, H., Stephens, J., Chen, Y., Ferles, K., Pailoor, S., Charbonnet, K., Dillig, I., Feng, Y.: Practical security analysis of zero-knowledge proof circuits. In: 33rd USENIX Security Symposium, USENIX Security 2024. pp. 1471–1487. USENIX Association (2024), <https://www.usenix.org/conference/usenixsecurity24/presentation/wen>
 34. Yang, H.S., Zheng, Q.X., Yang, J.: Algebraic cryptanalysis of AO primitives based on polynomial decomposition - applications to Rain and full AIM-I/III/V. In: Hanaoka, G., Yang, B.Y. (eds.) Advances in Cryptology - ASIACRYPT 2025 - 31st International Conference on the Theory and Application of Cryptology and Information Security, Melbourne, VIC, Australia, December 8-12, 2025, Proceedings, Part I. Lecture Notes in Computer Science, vol. 16245, pp. 383–410. Springer (2026). https://doi.org/10.1007/978-981-95-5018-0_13
 35. Yang, K., Sarkar, P., Weng, C., Wang, X.: QuickSilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field. In: Kim, Y., Kim, J., Vigna, G., Shi, E. (eds.) CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021. pp. 2986–3001. ACM (2021). <https://doi.org/10.1145/3460120.3484556>
 36. Zhang, K., Wang, Q., Yu, Y., Guo, C., Cui, H.: Algebraic attacks on round-reduced Rain and full AIM-III. In: Guo, J., Steinfeld, R. (eds.) Advances in Cryptology - ASIACRYPT 2023 - 29th International Conference on the Theory and Application of Cryptology and Information Security, Guangzhou, China, December 4-8, 2023, Proceedings, Part III. Lecture Notes in Computer Science, vol. 14440, pp. 285–310. Springer (2023). https://doi.org/10.1007/978-981-99-8727-6_10

Supplementary Material

A Concrete Proof-System Instantiations

Section 2 gives the proof-system semantics needed to analyze relation-encoding soundness. This appendix records the transcript mechanics of the two target proof systems and where an external witness enters each prover.

A.1 AIMer: from a field trace to BN++ views

ZKBoo and subsequent work made MPCitH practical for Boolean circuits and signatures [16,9,20]. Arithmetic and threshold-based variants improved efficiency and expanded the range of applications [6,14]. AIMer uses the arithmetic BN++ backend described below.

Injecting the witness by share offsets. Let $U = (\mathbf{pt}, t_0, \dots, t_{\ell-1})$ collect the independently supplied AIMer witness coordinates. For every BN++ repetition e , party seeds first determine random additive shares $U_e^{(1)}, \dots, U_e^{(N)}$. The prover publishes the correction

$$D_e = U - \sum_{i=1}^N U_e^{(i)}$$

and adds it to the designated party’s share. The affine value t_* and the multiplication outputs are then computed locally from these corrected shares and the public data; they are not independently adjustable offsets. The auxiliary BN++ check values have their own share corrections.

For honest signing, U contains the plaintext and its semantic first-layer S-box outputs. The forger instead supplies $U^* = (\mathbf{pt}', 0, \dots, 0)$, after which the local affine computation gives $t'_* = b_{iv}$. This produces the complete trace W^* of Algorithm 1 while preserving affine consistency. The downstream BN++ computation checks this trace without rerunning semantic witness extension.

In the reference code, `aim2_mpc` constructs the multiplication residuals for (AIM2-Br) and (AIM2-Out), and `run_phase_2_and_3` batches these residuals through BN++.

AIMer’s repeated-multiplier specialization. The generic BN++ check is Equation (MPC). AIMer’s multiplication identities share a common hidden multiplier, so its repeated-multiplier optimization combines several α_j values into one random linear combination. This reduces the signature size but does not change the algebraic statement: the opened value is still a random combination of the submitted errors $z_j - x_j y_j$. When $t'_j = 0$ makes a submitted AIMer product identity equal to $0 = 0$, that gate contributes zero to every such combination.

Fiat-Shamir and view opening. The prover commits to the party seeds and simulated views, hashes the statement, message, and commitments to obtain the multiplication-check challenges, commits to the check messages, and hashes again to select the unopened party in each repetition. The signature reveals the seed-tree information for all other parties together with the share offsets and aggregate-check data.

The verifier reconstructs the opened random tapes and shares, reapplies the offsets, recomputes all public-linear operations and BN++ check shares, and recomputes both hashes. It verifies consistency of the opened execution and the aggregate product check. It never reconstructs the complete aggregate values $\mathbf{pt}, t_j, t_*$, so a zero hidden S-box output is represented only through valid shares and equations, not by an explicit public zero flag.

A.2 Lynxer: from witness corrections to QuickSilver

VOLeith builds on efficient correlation generation, including SoftSpokenOT [30], and algebraic checking through QuickSilver [35]. Subsequent work improves the resulting signatures and security analysis [4]. We detail the witness and transcript interfaces used by Lynxer’s instantiation.

Generalized IT-MACs. For a formal degree- d IT-MAC $\llbracket a \rrbracket^d$ over $\mathbb{E}$, the prover holds

$$p_a(X) = aX^d + \sum_{h=0}^{d-1} \rho_{a,h} X^h,$$

while the verifier holds an evaluation point Δ and $K_a = p_a(\Delta)$, following the generalized IT-MAC formulation of QuickSilver [35]. Opening an authenticated value means sending $p_a(X)$ and checking it at Δ . If the prover changes the X^d coefficient, the difference between the two openings is a nonzero polynomial of degree at most d that must vanish at Δ , giving error at most $d/|\mathbb{E}|$ when the pair is fixed before uniform $\Delta \in \mathbb{E}$ is sampled. For equal formal degrees, addition and public scaling act componentwise on the prover polynomials and verifier evaluations. Multiplication uses

$$p_{ab}(X) = p_a(X)p_b(X), \quad K_{ab} = K_a K_b,$$

and raises the formal degree from (d_a, d_b) to $d_a + d_b$. The degree-one instance is Equation (MAC).

Authenticating the serialized witness. Lynxer’s field-valued trace is $W = (k, v_1, v_2)$. Its serialized buffer is

$$\xi = \text{bin}(k) \parallel \text{bin}(v_1) \parallel \text{bin}(v_2) = (\xi_1, \dots, \xi_{n_w}) \in \mathbb{K}^{n_w}, \quad n_w = 3n.$$

For each coordinate, subfield VOLE logically supplies a random degree-one IT-MAC

$$p_{r_i}(X) = \rho_i + r_i X, \quad K_{r_i} = p_{r_i}(\Delta) = \rho_i + \Delta r_i,$$

where $r_i \in \mathbb{K}$ is the random authenticated bit, $\rho_i \in \mathbb{E}$ is the prover-side randomizer, and $K_{r_i} \in \mathbb{E}$ is the verifier-side evaluation. To fix this random correlation to the prover-chosen bit ξ_i , the prover publishes

$$\delta_i = \xi_i - r_i \quad (= \xi_i + r_i \text{ over } \mathbb{F}_2).$$

The prover updates $p_{\xi_i}(X) = p_{r_i}(X) + \delta_i X$, while the verifier updates

$$K_{\xi_i} = K_{r_i} + \delta_i \Delta = \rho_i + \Delta \xi_i. \quad (\text{Fix})$$

Equation (Fix) shows that fixing preserves the IT-MAC invariant for every prover-chosen serialized witness. In the ideal subfield-VOLE correlation, the uniform r_i statistically hides ξ_i in δ_i ; the seed- and PRG-based realization provides the corresponding computational hiding.

The reference signing path computes W_h through Lynx witness extension before serializing and fixing it. Our attack instead supplies $W^* = (k', v'_1, v'_2)$ to the same serialization, fixing, VOLEitH, and QuickSilver path. Once the buffer is supplied, the remaining prover and verifier algorithms are identical in the honest and malicious executions.

VOLEitH transcript order. VOLEitH realizes the random correlations by committing to seed-generated candidate vectors and proving their consistency [5]. The prover first fixes the seed commitments, witness corrections, and QuickSilver response material. Only afterward is the all-but-one challenge derived, after which the prover opens every seed except the challenged one in each set. The verifier then reconstructs its evaluation point and authentication material, checks the VOLE correlations, and performs the final QuickSilver consistency check. Under Fiat-Shamir, these challenges are transcript-derived rather than permanently secret, but they remain unpredictable when the preceding commitments and responses are fixed. This order prevents the prover from choosing an alternative authenticated polynomial after learning the evaluation point.

QuickSilver coefficient check. Let $f_i(\xi) = 0$ be a submitted constraint of degree at most d , and decompose it as $f_i = \sum_{h=0}^d f_{i,h}$, where $f_{i,h}$ contains its homogeneous degree- h terms. QuickSilver constructs

$$g_i(X) = \sum_{h=0}^d f_{i,h}(p_{\xi_1}(X), \dots, p_{\xi_{n_w}}(X)) X^{d-h} = \sum_{h=0}^d A_{i,h} X^h.$$

Its highest coefficient and verifier-side evaluation are

$$A_{i,d} = f_i(\xi), \quad B_i = g_i(\Delta) = \sum_{h=0}^d f_{i,h}(K_{\xi_1}, \dots, K_{\xi_{n_w}}) \Delta^{d-h}.$$

The protocol derives batching coefficients χ_i and adds a random formal degree- d IT-MAC of zero

$$A^*(X) = \sum_{h=0}^{d-1} A_h^* X^h, \quad B^* = A^*(\Delta).$$

In the conceptual coefficient-opening view, the prover supplies the masked coefficients and the verifier checks

$$\begin{aligned} U_h &= A_h^* + \sum_i \chi_i A_{i,h} \quad (0 \leq h < d), \\ \sum_{h=0}^{d-1} U_h \Delta^h &= B^* + \sum_i \chi_i B_i. \end{aligned} \tag{QS}$$

If the batched constraint value $\sum_i \chi_i f_i(\xi)$ is nonzero, the right-hand side additionally contains its product with Δ^d , which is absent from the verifier's degree- $(d-1)$ reconstruction [35]. The zero IT-MAC masks the lower coefficients, while the random batch and evaluation point provide soundness. The response material must be fixed before the evaluation challenge becomes predictable; otherwise a lower coefficient could be chosen afterward to force the displayed equality.

The optimized degree-two Fiat–Shamir transcript compresses this conceptual opening. It explicitly carries only the nonconstant masked coefficient, while the verifier reconstructs the constant coefficient from the evaluation identity and binds the reconstruction into the Fiat–Shamir hash [18]. Thus the public transcript does not contain the witness coordinates or an unmasked constraint polynomial.

Evaluating the three submitted equations. Both the Lynxer prover and verifier parse the authenticated bits in the same order, pack them into field elements through public linear operations, and apply the linear and Frobenius maps from Table 5. The three parameter-specific constraints are evaluated as degree-two IT-MAC polynomials and batched through Equation (QS).

For the large-parameter attack, the selected witness has $z' = k' + C_2 = 0$ and $A' = L_1(v'_1) = 0$. The submitted output residual therefore satisfies

$$f_{\text{Out}}(W^*) = A'B' - w'z' = 0$$

independently of the public output contained in w' . QuickSilver correctly certifies this zero constraint value together with the two branch constraints. Equivalently, the X^2 coefficient of the associated IT-MAC polynomial is zero, although its lower-degree coefficients need not vanish. Because the constraint relation contains neither a polynomial encoding of $z' \neq 0$ nor the omitted residual condition $w' = 0$ on the null branch, neither IT-MAC authentication nor batching can recover that condition.

Visibility. A public Lynxer signature contains vector-commitment openings, witness corrections, authentication-check material, and the masked QuickSilver response. These objects bind one common hidden witness to the three submitted equations, but they reveal neither (k, v_1, v_2) nor the provenance of the serialized buffer. An additional hidden nonzero condition would be expressible, for example by authenticating q and checking $(k + C_2)q - 1 = 0$; the vulnerability is that no such polynomial is submitted.

B Relation Domains and Security Implications

B.1 Semantic and Public-Key Domains

The domain conventions behind Section 2 distinguish semantic inputs from the keys produced by honest sampling. For the OWF family defined there, write

$$\begin{aligned} R_F^{\text{tot}}((iv, y), x) = 1 &\iff F_{iv}(x) = y, \\ R_F^{\text{res}}((iv, y), x) = 1 &\iff \text{Dom}_{pk}(x) = 1 \wedge F_{iv}(x) = y, \end{aligned}$$

where Dom_{pk} is a public, efficiently decidable predicate explicitly declared by the construction.

Each claim uses the declared semantics $R_F \in \{R_F^{\text{tot}}, R_F^{\text{res}}\}$. A restriction used only to sample honest keys does not establish $\text{Dom}_{pk}(x) = 1$ for every accepted witness. For example, an OWF can totalize inversion at zero while KeyGen resamples zero S-box inputs. Our repairs preserve the total relation, including those inputs.

For Lynxer, syntactic validity requires a parsable parameter identifier and IV, derivable public maps and constants, and a field-element output. The honest-key domain is the support of the KeyGen distribution:

$$\mathcal{P}_\lambda^{\text{hon}} = \text{supp}\{\text{pk} : (\text{pk}, \text{sk}) \leftarrow \text{KeyGen}(1^\lambda)\}.$$

Correctly generated keys are accepted by the parser, so $\mathcal{P}_\lambda^{\text{hon}} \subseteq \mathcal{P}_\lambda^{\text{syn}}$. The inclusion may be strict. A correspondence claim covering honest keys can name any intermediate domain

$$\mathcal{P}_\lambda^{\text{hon}} \subseteq \mathcal{D}_\lambda \subseteq \mathcal{P}_\lambda^{\text{syn}}, \quad \mathcal{D} = \{\mathcal{D}_\lambda\}_\lambda.$$

Taking $\mathcal{D}_\lambda = \mathcal{P}_\lambda^{\text{hon}}$ gives the honest-key claim relevant to EUF-CMA; taking $\mathcal{D}_\lambda = \mathcal{P}_\lambda^{\text{syn}}$ extends it to every syntactically valid key.

B.2 From Witness Construction to Signature Forgery

We give the probability notation and proof for Proposition 1. Its completeness assumption states that, for every λ , every $\text{pk} \in \mathcal{P}_\lambda^{\text{hon}}$, every message m , and every $W \in \mathcal{S}_C(\text{pk})$,

$$\Pr[\text{Verify}(\text{pk}, m, \text{Prove}_{R_C}(\text{pk}, W; m)) = 1] \geq 1 - \delta_{\text{comp}}(\lambda),$$

over fresh proving and verification randomness. In the random-oracle model, unconditional probabilities also include sampling the shared random oracle. The completeness bound must hold conditional on the constructor's output and prior oracle transcript, averaging over fresh coins and previously unqueried oracle values. Define

$$\varepsilon_{\text{RelWit}}(\lambda) = \Pr_{\substack{(\text{pk}, \text{sk}) \leftarrow \text{KeyGen}(1^\lambda) \\ W \leftarrow \text{RelWit}(\text{pk})}}[W \neq \perp \wedge R_C(\text{pk}, W) = 1].$$

Proof. Run $W \leftarrow \text{RelWit}(\text{pk})$ and abort if $W = \perp$ or $R_C(\text{pk}, W) = 0$. Otherwise, choose a message m and output $\text{Prove}_{R_C}(\text{pk}, W; m)$. Construction succeeds with probability $\varepsilon_{\text{RelWit}}$. Conditional on success, the completeness assumption gives acceptance probability at least $1 - \delta_{\text{comp}}$. Multiplication yields the claimed bound. No signing queries are made, so the chosen message is fresh. $\square$

Completeness of the concrete provers. AIMer’s BN++ prover has perfect completeness for every satisfying relation witness, so $\delta_{\text{comp}} = 0$. For Lynxer, zero QuickSilver residuals verify deterministically once counter grinding succeeds. Appendix A describes how satisfying witnesses enter the original proving paths.

In the archived Lynxer reference code, `ref/faest_impl.c` hashes a fixed transcript prefix followed by a 32-bit counter. The signing loop has no explicit failure exit. It accepts a challenge only after the trailing-zero check, challenge decoding, and the opening-size check in `ref/bavc.c` succeed. These admissibility tests depend on the challenge and public parameters, not on the witness. Let p_{adm} denote their joint acceptance probability for a uniform challenge; it includes the opening-size restriction, not just the trailing-zero test.

For the PPT conversion, truncate execution after a polynomially bounded number $q(\lambda)$ of counter attempts and abort without a signature on exhaustion. For the reference counter, require $q \leq 2^{32}$ to avoid repeated inputs. This analytical truncation leaves successful transcripts and the verifier unchanged; the reported implementation runs use the original loop. Distinct counters yield independent uniform challenges in the random-oracle model only when their complete hash inputs have not been queried previously. Under this freshness condition, the grinding failure probability is $(1 - p_{\text{adm}})^q \leq \exp(-qp_{\text{adm}})$. If freshness can fail, its probability must also be included in δ_{comp} . A sufficient condition for asymptotically negligible error is a polynomial budget satisfying $q(\lambda)p_{\text{adm}}(\lambda) = \omega(\log \lambda)$ and negligible freshness failure. We retain δ_{comp} explicitly for Lynxer and do not infer a negligible bound from successful KAT runs.

B.3 A Computational Relaxation

Structural soundness versus computational hardness. Relation-encoding soundness is a strong structural guarantee, but it is not logically necessary for EUF-CMA security. The equivalent witnesses of Definition 2 form the relation gap

$$\text{Gap}(\text{pk}) = \{W \in \mathcal{W}_\lambda : R_C(\text{pk}, W) = 1 \wedge R_F(\text{pk}, \text{Dec}(W)) = 0\}.$$

Relation-encoding soundness holds on the declared domain if and only if this set is empty for every λ and $\text{pk} \in \mathcal{D}_\lambda$. For the honest-key distribution relevant to EUF-CMA, define

$$\text{Adv}_A^{\text{gap-hon}}(\lambda) = \Pr[(\text{pk}, \text{sk}) \leftarrow \text{KeyGen}(1^\lambda); W \leftarrow \mathcal{A}(\text{pk}) : W \in \text{Gap}(\text{pk})].$$

This definition measures gap hardness under honest key generation. Extending it to syntactically valid keys requires a specified key distribution or an adversary that outputs both pk and W . Structural relation-encoding soundness guarantees the extracted-witness conversion for every key in $\mathcal{D}_\lambda$.

B.4 Encoding Completeness and Honest Correctness

Completeness versus honest correctness. Relation-encoding completeness provides an encoding for every semantic preimage; proof-system completeness concerns acceptance given a satisfying witness. Honest signing requires an efficient witness-extension algorithm such that, for every honestly generated key and its semantic witness x ,

$$W_h \leftarrow \text{Extend}_F(\text{pk}, x), \quad \text{Dec}(W_h) = x, \quad R_C(\text{pk}, W_h) = 1.$$

Randomized extension can allow negligible failure probability. The concrete repairs give efficient extensions for every semantic preimage.

B.5 The Lynxer Key-Domain Prefactor

Key-distribution mismatch. A separate issue concerns the key domain used for FAEST’s prefactor

$$\left(1 - \frac{|\text{dom}(\text{OWF})|}{|\text{cod}(\text{OWF})|}\right)^{-1} \leq 4.$$

FAEST obtains this bound by excluding keys whose first two bits are both one. That restriction occurs in the archived Lynxer snapshot. The written Lynx KeyGen instead resamples when specified S-box inputs are zero, a public-IV-dependent domain, and does not by itself justify the same 3/4 domain ratio. The prefactor and the language-inclusion step therefore require separate checks against the same declared OWF domain.

C Additional Boundary Analyses

This appendix complements the Lynx-128 boundary in Section 5 with further relation-level checks and the FAEST witness-encoding example from Section 6.3.

C.1 When the zero case is carried by the relation

A strict inverse equation $xy = 1$ proves $x \neq 0$ and $y \neq 0$ without revealing either value. A totalized inverse can instead be encoded by the exact pair

$$x^2y = x, \quad xy^2 = y,$$

which forces $(x, y) = (0, 0)$ at the exceptional input; Proposition 3 proves the equivalence. Both choices close the proof-domain branch exploited in the main attacks.

Lynx-128 is the closest negative control: its direct inverse equations reject the zero assignments used against the larger dimensions.

FAEST v2.0 provides a separate example of carrying a KeyGen restriction into the proved relation. In the Round-2 specification, `FAEST.KeyGen` resamples while the first two key bits are both one [3, Fig. 7.1, line 3, p. 56]. The

`ZK.OWFConstraints` routine explicitly appends the constraint $w[0]w[1] = 0$, with the degree conversion used by its proof system [3, Fig. 6.22, line 5, p. 54]. On bit-valued witness coordinates, this polynomial excludes exactly the rejected pattern $(1, 1)$. Both the prover and verifier invoke that constraint routine [3, Fig. 6.23, line 12 of each routine, p. 55]. Thus the specified proof checks the key-domain condition on hidden coordinates without revealing the key.

C.2 Distinct FAEST Signatures with Fixed Signing Randomness

Fix a FAEST v2 key pair whose OWF computation contains a zero input to a norm-compressed S-box. Also fix a message m and signing randomness ρ ; deterministic signing uses $\rho = 0^\lambda$ [3, Fig. 7.2]. A signer knowing sk computes the canonical extended witness W , then replaces the four bits representing $c = 0$ by a nonzero $c' \in \mathbb{F}_{16}$. By (Norm), the resulting $W' \neq W$ preserves the inverse-norm residual and all reconstructed inverse conjugates. All other witness coordinates remain unchanged, so W' satisfies the complete OWF constraints for the same key.

The fixed key, message, and ρ determine the seed and IV, hence the initial VOLE commitment and witness mask u . The witnesses give distinct serialized correction strings

$$d = W \oplus u_{[0,\ell)}, \quad d' = W' \oplus u_{[0,\ell)}, \quad d \oplus d' = W \oplus W' \neq 0.$$

Using W' , the signer recomputes the second Fiat–Shamir challenge, the Quick–Silver responses, and the final challenge and commitment opening. Completeness gives an accepted signature on successful termination of the ordinary grinding loop. For any two successfully completed runs with different auxiliary choices, the signatures differ because d is serialized in the signature [3, Fig. 7.2]. In particular, if all 16 auxiliary choices yield successful termination, the resulting signatures are pairwise distinct. This statement is conditional on termination and does not assert success for every fixed choice of signing randomness.

The prescribed `Sign` algorithm remains deterministic for fixed ρ ; the alternative runs modify witness extension while retaining the same derived randomness and initial commitment. Every modified witness decodes to the original OWF preimage, and constructing these signatures requires sk . Thus the construction establishes representation non-uniqueness, not an EUF-CMA or SUF-CMA forgery.

D The Lynx-192 Public Linear-System Route

This appendix details the Lynx-192 constructor of Section 5.5. A successful output is an equivalent witness when $\text{Lynx}_{\text{iv}}(k) \neq t$. Let

$$\mathcal{K} = \{k \in \mathbb{E} : L_0(k) + C_1 = 0\}.$$

For any selected $k \in \mathcal{K}$, compute $v_1 = S_1(k + C_0)$ and define the public $\mathbb{K}$ -linear map

$$T_k = M_{L_1(v_1)} \circ L_2 + L_3,$$

where M_a denotes multiplication by the public field element a .

Algorithm 3 RelWit_{Lynx192}(pk = (iv, t))

1. Solve $L_0(k) = C_1$ over the public $\mathbb{K}$ -coordinate representation. If inconsistent, return $\perp$; otherwise select the canonical solution returned by Gaussian elimination.
2. For that k , compute $v_1 = S_1(k + C_0)$ and construct T_k .
3. Solve

$$T_k(v_2) = t + L_3(k + v_1).$$

If inconsistent, return $\perp$; otherwise return $W^* = (k, v_1, v_2)$.

Theorem 5 (Lynx-192 witness construction). *For any supplied $k \in \mathcal{K}$, if*

$$T_k(v_2) = t + L_3(k + v_1)$$

is consistent, Gaussian elimination computes a Lynx-192 relation witness in polynomial time. Exhaustively searching all of $\mathcal{K}$ costs

$$2^{\dim \ker L_0} \text{poly}(n)$$

in the worst case. If $\mathcal{K}$ is a singleton and T_k is invertible, the route succeeds for every output t of that public relation.

Algorithm 3 runs in polynomial time using the canonical solution; the theorem also quantifies the cost of searching the remaining affine solutions.

Proof. For $k \in \mathcal{K}$, the second branch residual

$$u^{2^{59}} - uv_2$$

is zero for every v_2 because $u = L_0(k) + C_1 = 0$. The semantic value $v_1 = S_1(k + C_0)$ satisfies Br_1 . Using $\mathbb{K}$ -linearity of L_3 , the direct output residual is zero exactly when

$$\begin{aligned} L_1(v_1)L_2(v_2) + L_3(k + v_1 + v_2) &= t, \\ (M_{L_1(v_1)} \circ L_2 + L_3)(v_2) &= t + L_3(k + v_1). \end{aligned}$$

Any solution therefore completes all three QuickSilver residuals. Gaussian elimination is polynomial time for a supplied k . The solution set of $L_0(k) = C_1$ is either empty or an affine coset of $\ker L_0$, which gives the exhaustive-search bound. If $\mathcal{K}$ is a singleton and T_k is invertible, the second system has a unique solution for every right-hand side.

For a fixed supplied $k \in \mathcal{K}$ with $r = \text{rank}(T_k)$, the affine output map has exactly 2^r values. An independent uniform output $t \in \mathbb{E}$ therefore gives success probability 2^{r-192} . For several candidates k , the successful set is a union of affine images, whose intersections must not be double-counted. For honestly generated keys, success is measured under the actual KeyGen distribution, as below.

For the concrete submitted fixed-matrix parameter set with the archived key-sampling rule, let

$$p_{192}^{\text{fixed}} = \Pr_{(\text{pk}, \text{sk}) \leftarrow \text{KeyGen}_{192}^{\text{fixed}}} [\text{RelWit}_{\text{Lynx192}}(\text{pk}) \neq \perp].$$

Proposition 1 gives an adversary making no signing queries with success at least $p_{192}^{\text{fixed}}(1 - \delta_{\text{comp}})$. Appendix F.2 records a successful fixed-matrix instance.

E Repair Summary

Table 7 summarizes the exact local graph encodings proved in Section 7 and separates them from domain-restricting nonzero checks. The first four rows encode exact total graphs; the last row restricts the witness domain. Cost $(\Delta W, \Delta E)$ counts additional hidden field elements and the net increase in quadratic equations. The baseline is the single conditional equation for each graph: $t^{2^e} = ut$, $xy = 1$ (or $x^2y = x$), $x^{2^i}y = x^{2^{2^i}}x$, and $AB = wz$, respectively. The last row counts one additional nonzero check, not the total cost of enforcing a restricted domain.

Table 7. Local repair templates.

Semantic graph	Zero-safe encoding	Cost
$u = t^{2^e-1}$	retain $t^{2^e} = ut$; add $u = tr$	(1, 1)
$y = \text{Inv}_0(x)$	$x^2y = x$ and $xy^2 = y$	(0, 1)
$y = x^{2^{2^i}-2^i+1}$	retain $x^{2^i}y = x^{2^{2^i}}x$; add $y^{2^i}y = x^{2^{3^i}}x$	(0, 1)
$w = AB \text{Inv}_0(z)$	$p = AB$, $zp = z^2w$, and $w = zr$	(2, 2)
restricted: $a \neq 0$	retain the conditional equation; add $aq = 1$	(1, 1)

E.1 Power-map permutation checks

The inverse-permutation notation used in the case studies and their repairs is justified by the following parameter checks. For a Mersenne map,

$$\gcd(2^e - 1, 2^n - 1) = 2^{\gcd(e, n)} - 1.$$

The submitted AIM2 tuples $(n; e_0, \dots, e_*)$ are (128; 49, 91, 3), (192; 17, 47, 5), and (256; 11, 141, 7, 3). Every listed exponent is coprime to n , so the corresponding maps permute $\mathbb{F}_{2^n}$.

For the Lynx Mersenne maps,

$$\gcd(2^e - 1, 2^n - 1) = 1 \quad \text{for } (n, e) \in \left\{ (192, 13), (192, 59), (256, 5), \right. \\ \left. (256, 53), (384, 13) \right\}.$$

For the Kasami exponent $d_i = 2^{2i} - 2^i + 1$,

$$\gcd(d_i, 2^n - 1) = 1 \quad \text{for } (n, i) \in \{(384, 30), (512, 6), (512, 90)\}.$$

In particular, $d_6 = 4033$ and $\gcd(4033, 2^{512} - 1) = 1$. Thus every power map used as a permutation in the repairs is bijective on the stated field.

E.2 Composition of exact local graphs

The following theorem makes the composition argument in Section 7 explicit. It follows the compositional circuit-verification pattern formalized by Coglio et al. [12]; its forward projection is closely related to the output-consistency criterion studied by Zequal [31].

Theorem 6 (Composition of exact graph repairs). *Let iv denote the public computation parameters, x a hidden input, and y a public output. Order the acyclic semantic computation as $z_0 = x$ and $z_i = g_i(\text{iv}, z_{<i})$ for $1 \leq i \leq N$, with output z_N . For each gate, let η_i be a fresh auxiliary tuple that appears in no other gate, and suppose*

$$\exists \eta_i : G_i(\text{iv}, z_{\leq i}, \eta_i) = 0 \quad \Longleftrightarrow \quad z_i = g_i(\text{iv}, z_{<i}).$$

Define $R^0((\text{iv}, y), (x, z_1, \dots, z_N, \eta_1, \dots, \eta_N)) = 1$ exactly when every $G_i = 0$ and $z_N = y$. Then

$$\exists z_1, \dots, z_N, \eta_1, \dots, \eta_N : R^0 = 1 \quad \Longleftrightarrow \quad \text{Eval}_{\text{iv}}(x) = y.$$

Thus R^0 is a sound and complete encoding under the decoder selecting x .

Proof. For the forward direction, $z_0 = x$ is fixed. Inductively, if $z_{<i}$ equals the semantic prefix, the forward implication for G_i fixes $z_i = g_i(\text{iv}, z_{<i})$. Hence $z_N = \text{Eval}_{\text{iv}}(x)$, and the output equality gives $\text{Eval}_{\text{iv}}(x) = y$. Conversely, compute the semantic trace from (iv, x) , use the reverse implication at each gate to choose its fresh tuple η_i , and concatenate those tuples. All $G_i = 0$ and $z_N = y$ then hold.

Freshness preserves independent choices of auxiliary variables. For example, the power repair at $t = u = 0$ allows every r , all representing the same semantic gate value. Reusing an auxiliary at unrelated gates could impose incompatible quotient values and destroy completeness. Uniqueness is unnecessary: the forward proof fixes the semantic wires regardless of the local choices, and Dec ignores those choices.

E.3 Proof details for the local repairs

Proposition 2. If $t = 0$, $u = tr$ gives $u = 0$; if $t \neq 0$, division in $t^{2^e} = ut$ gives $u = t^{2^e-1}$. Conversely, take $r = 0$ at $t = 0$ and $r = t^{2^e-2}$ otherwise.

Proposition 3. At $x = 0$, $xy^2 = y$ forces $y = 0$. At $x \neq 0$, $x^2y = x$ gives $xy = 1$, and the second equation follows. Both equations hold on the resulting totalized inverse graph.

Proposition 4. For $x \neq 0$, divide the first equation by x^{2^i} to obtain $y = x^{2^{2^i}-2^i+1}$. For $x = 0$, the companion forces $y^{2^i+1} = 0$ and hence $y = 0$. Conversely, the first identity follows from the exponent definition and the companion from $(2^{2^i} - 2^i + 1)(2^i + 1) = 2^{3^i} + 1$.

E.4 Parameter-by-parameter Lynx repair derivation

For $n = 128$, Proposition 3 gives the two totalized inverse graphs. For $n = 192$, Proposition 2 gives both power graphs. For $n = 256$, that proposition gives Br_1 and the specification's unmultiplied Br_2 directly. For the snapshot Br_2 , the companion $v_2 = ur_2$ forces $v_2 = 0$ at $u = 0$; at $u \neq 0$, cancelling one u in the additionally multiplied equation gives the same Mersenne graph. At $n = 384$, the first branch follows from Proposition 2 and the second from Proposition 4 with $i = 30$. At $n = 512$, the latter proposition with input and output exchanged gives the first branch at $i = 6$, and its ordinary direction at $i = 90$ gives the second. These are exactly the maps in Table 4; inverse notation is licensed by the explicit gcd checks in Appendix E.1.

For $n \in \{128, 192\}$, $AB = w$ is the direct semantic output equation. For the larger dimensions, Proposition 5 gives $w = AB \text{Inv}_0(z)$, including $z = 0$. Substituting $w = t + h$ and $h = L_3(k + v_1 + v_2)$ yields $t = \text{Lynx}_{\text{iv}}(k)$. Every local implication is reversible, so Theorem 6 proves both relation-encoding soundness and relation-encoding completeness.

The total-graph increments are

n	128	192	256	384	512
ΔW	0	2	4	3	2
ΔE	2	2	4	4	4

A restricted-domain design may retain cleared equations only after the nonzero domain is made part of both KeyGen and the semantic relation and every cancelled factor is proved nonzero: v_1 and u for Lynx, additionally z at $n \geq 256$, and every t_j, t_* for AIMer.

F Implementation Validation

This appendix documents the reference implementations, KAT records, and timing measurements. The original source archives, buildable drivers, full records, and prior rebuild results are available at

<https://anonymous.4open.science/r/NBWA-C415/>

The repository can be browsed online or downloaded as a ZIP archive. All artifact paths below are relative to its root; `README.md` lists the dependencies and reproduction commands, and `MANIFEST.sha256` records the file checksums.

F.1 Reference versions and verifier boundary

The original archives are identified by SHA-256:

- AIMer package: d7bbdb28e70dad021da2057bc0b4802585770cbc2d7449acfc
cd770d8bc624521;
- Lynxer package: 9267541884ce9fe10827197b1b72c3b06f0e93081d1327b6
77e40b4cdc0ee8a8.

The Lynxer archive records upstream commit 3fd635a52bf26d9159a2d53e0e7246b42accf800.

The supplied AIMer witness source is `harness/AIMer/sign.c`. It corresponds to the upstream `Reference_Implementation/sign.c` and changes witness generation in `run_phase_1`. Later transcript phases remain unchanged. The suffix starting at `crypto_sign_verify_internal`, including the public verification entry points, is byte-for-byte identical to the archive, as is `aim2.c`.

For Lynxer, the KAT driver constructs the witness externally and calls the existing `faest_sign` routine. The prover and verifier in `ref/lynx_proof.c` parse $k \parallel v_1 \parallel v_2$ and evaluate the equations in Table 5. After line-ending normalization, `ref/faest_impl.c`, `ref/lynx_proof.c`, `ref/lynx_owf.c`, and `ref/owf.h` are identical to the archive. The constants C0, C1, and C2 correspond to our subscripts; the snapshot sets $C_2 = 0$ at dimensions 256 and 384 and derives it publicly at 512.

F.2 KAT records and reproducibility

Tables 2 and 6 list the seven retained examples: AIMer-128f/192f/256f and fixed-matrix Lynxer-192f/256f/384f/512f, respectively. The Lynxer-192f key was selected because its public completion system is consistent. For each key, the records contain an original KAT signature on m_0 , a distinct forgery on m_0 , and a forgery on a fresh message m_1 : seven controls and fourteen forgeries in total. The control uses the KAT secret key; the attack witness uses public data.

The script `tools/reproduce_kats.py` rebuilds the examples from archived sources. It checks verifier acceptance, signature distinctness, the direct-OWF diagnosis, and agreement with the full records.

The file `validation/KAT_RECHECK.json` records the prior seven-example rebuild. Tables 3 and 8 show the two 256-bit cases; `output/vectors/REFERENCE_KAT_VECTORS.json` contains their complete public inputs, witnesses, diagnostic outputs, outcomes, and signature hashes. Full signature bytes are in `vectors/`.

Table 8. Lynxer-256f KAT example (fixed matrices).

Item	Value or result
<i>Public inputs</i>	
Original message m_0	0xd81c4d8d...bb556ac8 (33 bytes)
Fresh message m_1	Null-branch forgery: new message.
iv	0x8626ed79...872bfb8f
t	0x81bec683...f001abf3
<i>Equivalent witness and relation checks</i>	
$k' = C_2$	0
v'_1	0
v'_2	0x1c02958e...383817d5
Constraint residuals	Br ₁ , Br ₂ , and Out residuals are zero
Lynx _{iv} (k')	0xa60350b6...4ab30c60 $\neq t$
<i>Unmodified verifier</i>	
Honest σ_0 on m_0	Accept
Forged σ_0^* on m_0	Accept; $\sigma_0^* \neq \sigma_0$
Forged σ_1^* on m_1	Accept
Signature size	14,772 bytes
<i>Note.</i> The KAT seed 0x06155023...d2e1ffa1 is reproduction metadata, not an attack input. ... omits intermediate hexadecimal digits; the first and last eight are shown. Full values are in the anonymous repository’s KAT data (Appendix F).	

F.3 Reference timing baseline

Table 9 uses the AIMer-256f and fixed-matrix Lynxer-256f keys. The run uses GCC 13.3.0 with `-O2` under WSL2/Linux on an Intel Core i9-13900KF, pins the process to one reported CPU, performs three warmups and 30 measured repetitions, and calls the unmodified verifier outside the timed interval after every signature.

Table 9. Reference timing baseline for two retained KAT public keys. Entries are medians in milliseconds, over 30 measured runs after three warmups.

Target	public setup	forgery API call	Verify acc.
AIMer-256f	0.296	11.610	30/30
Lynxer-256f, fixed	0.399	14.172	30/30

The public-setup column times witness reconstruction from the public key. For AIMer-256f, the forgery column calls `crypto_sign_signature` with the attack path enabled; that API generates signing randomness internally and repeats some public reconstruction. For Lynxer-256f, it calls `faest_sign` with the equivalent witness and a pregenerated ρ . The columns report separate durations at these scheme-specific API boundaries, including shared setup work where performed by each API.

The repository supplies `tools/reproduce_timings.py`, the timing drivers, raw per-run CSV files, and `validation/TIMING_BASELINE.json` with the build commands, environment metadata, and source and input hashes. The reported medians are computed from those CSV files.

F.4 Seed-derived matrix storage layout

The seed-derived sampler allocates each matrix as `uint64_t L0[512][8]`, with an eight-word row stride. The OWF and QuickSilver routines cast this storage to rows of $n/64$ words, consuming strides 2, 3, 4, 6, 8 at $n = 128, 192, 256, 384, 512$, respectively. Only the last stride matches the allocation. Fixed-matrix constants are copied into the compact representation consumed at their dimension.

Source locations and hashes for this storage-layout comparison are recorded in `validation/STATIC_SOURCE_AUDIT.json`.

F.5 Evidence scope and artifact checks

The KAT validation uses the speed variants and the fixed-matrix C reference implementation for Lynxer. Its selected 192-bit example demonstrates a consistent public completion system; the success probability over KeyGen remains unestimated. Theorem 1 and Theorem 2 establish the broader parameter and public-map coverage. The matrix-layout findings come from static source inspection; the optimized and corrected-stride seed-derived implementations are outside the KAT validation.

The tool `tools/check_materials.py` checks the manifest and stored KAT records; `tools/audit_evidence.py` additionally recomputes timing statistics and checks source hashes. These data audits complement the executable KAT and timing drivers. The provenance report `validation/EVIDENCE_PROVENANCE.md` inventories the evidence and identifies historical aggregates whose original experiment records are unavailable.