

edaBits in the Exponent (edaBitE) and Issuer-Hiding Blind Signatures from ML-DSA and VOLE-in-the-Head

Carsten Baum¹, Marvin Beckmann¹

Technical University of Denmark, Kongens Lyngby 2800 {cabau, mabeck}@dtu.dk

Abstract. Blind signatures let a user obtain a signature by interacting with an issuer that controls the signing key, while the issuer never sees the message. They form the basis of anonymous credentials, which are the centerpiece of privacy-preserving digital identification to be deployed widely in the coming years. Existing plausibly quantum-secure blind signatures rely on bespoke cryptographic constructions, instead of standardized ones. This is dissatisfying from a cryptographic perspective, and also leads to practical issues: secret signing keys are often stored in secure elements, in which case only signature algorithms supported by certified secure elements can be the basis for deployed protocols.

We give the first blind signature whose issuer runs a NIST-standardized post-quantum signature scheme unmodified. Issuance is just an ML-DSA signature on a commitment, and the blind signature is a zero-knowledge proof of knowledge, using the VOLE-in-the-head zero-knowledge proof system, of the ML-DSA signature. Security reduces to the EUF-CMA security of ML-DSA and to the proof system, with no new assumptions. A blind signature is about 1 MB and takes under a second to produce and half a second to verify. Issuer-hiding and selective disclosure of signed attributes can easily be added on top for a few tens of kilobytes, at a proof size essentially independent of how many issuers the list names. Underlying the construction is edaBitE, a novel batch consistency check between binary and prime-field commitments. In a non-interactive proof, the existing edaBits checks it replaces would be the single largest item in the blind signature. We believe that edaBitE is of independent interest.

Table of Contents

edaBits in the Exponent (edaBitE) and Issuer-Hiding Blind Signatures from ML-DSA and VOLE-in-the-Head	1
<i>Carsten Baum</i> [Ⓔ] , <i>Marvin Beckmann</i> [Ⓔ]	
1 Introduction	4
1.1 Our Contributions	5
1.2 Technical Overview	6
1.3 Related Work	9
2 Preliminaries	10
2.1 Blind Signatures	10
2.2 ML-DSA	11
2.3 VOLE-in-the-Head NIZKs	13
2.4 From vector oblivious linear evaluation (VOLE) to VOLE-in-the-Head (VOLEitH)	14
3 edaBits in the Exponent (edaBitE)	16
4 Implementing the Blind Signature from MLDSA	21
4.1 Proving SHAKE256	21
4.2 The circuit for SampleInBall	22
4.3 Range Checks and Digit Decomposition	25
4.4 Benchmarks and Sizes	27
4.5 Security	27
5 Issuer-Hiding & Attribute Disclosure	28
5.1 Obtaining Issuer-Hiding	28
5.2 Attribute Disclosure	29
A AI Usage	34
B Benchmarking Setup	35
C Additional Preliminaries	36
C.1 Blind Signatures	36
C.2 ML-DSA Verification	37
C.3 Non-Interactive Zero-Knowledge Proofs of Knowledge	37
D Fischlin’s Blueprint for Blind Signatures with VOLEitH Masking for Commitments	39
E Proving SHAKE256	41
F Proof Strategy Details for SampleInBall	42
F.1 The Linear ORAM	42
F.2 The General ORAM	43
G Range Checks in Detail	45
G.1 Digit Domains and Indicators	45
G.2 The Hint, and Recovering w_1	46
G.3 Pricing the Variants	47
H Proof of Theorem 2	48
I Issuer-Hiding and Attributes	49

I.1	Issuer-Hiding Selection Gadget	49
I.2	Proof of Theorem 3.....	50
I.3	Attributes	51

1 Introduction

A digital signature lets a signer authenticate a message using a signing key, verifiable under the matching verification key, and in doing so, the signer learns the message. A blind signature [19] adds a third party so that it need not: a *user* holds a message, an *issuer* signs a blinded form of it, allowing the user afterwards to present a signature to a *verifier*. Blindness says that a collaborating issuer and verifier cannot tell which among multiple issuance protocol runs produced which signature, and one-more unforgeability says that Q issuances yield no more than Q valid signatures.

Blind signatures are of practical interest in the design of privacy-preserving identification schemes such as the EU Digital Identity Wallet (EUDIW), which every EU Member State must soon offer its citizens. While not present in the current draft (which is not unlinkable), future versions might support enhanced privacy notions using blind signature techniques such as longfellow-zk [23]. However, while blind signature-based credentials are the leading approach to privacy, vanilla blind signatures do not suffice. Credentials are issued by many authorities (in the EUDIW example one or more per Member State), so a showing that names its issuer narrows the holder down to that issuer’s population; *issuer-hiding* asks that the verifier learn only that some authority on a list vouched for the credential. Additionally, a credential often carries attributes the issuer knows, so a holder asked to e.g. prove they are over 18 should be able to do exactly that, and not surrender e.g. a birth date; this is *selective disclosure*. A deployment at the scale and lifetime of a nation-state also needs a plan to move to quantum-secure cryptography in the foreseeable future. The classical constructions, from Chaum’s original blind signature scheme [19] to blind Schnorr [45], do not survive a quantum attacker, and while the post-quantum proposals that replace them achieve compact showings, some bound how many issuances may run concurrently, and none builds on a standardized cryptographic primitive (Section 1.3).

Fischlin’s paradigm. A popular approach to constructing blind signatures is to follow Fischlin’s paradigm [17, 18, 22]: the user commits to its message and sends the commitment; the issuer signs that commitment with an *ordinary* signature scheme and returns the signature; the user then proves non-interactively in zero knowledge that it knows a signature, valid under the issuer’s key, on a commitment to the message it is presenting. That zero-knowledge proof is the blind signature, and the verifier checks it against the issuer’s public verification key and the message. The protocol is round-optimal and the issuer sends a single signature, so issuance is cheap, and almost all work falls on the user. Both matter for an issuer that has to serve, say, a population of a whole country.

The signature scheme chosen decides how efficient the blind signature can be, as it defines the relation to prove in zero knowledge, but there is a second restricting factor. An issuer’s key has to be stored somewhere it cannot leak, which in practice means a certified secure element, and a secure element runs the algorithms it was certified for and for which standards exist. A blind signature

that needs a modified signing routine, or a scheme that is not standardized, therefore needs new certified code before an issuer can use it at all. The post-quantum signature standard such a device will hold for general-purpose signing is very likely¹ Module-Lattice-Based Digital Signature Algorithm (ML-DSA) [37]. Interestingly, no blind signature has so far been built on the standard directly.

This is mainly because ML-DSA is not an easy scheme to prove in zero knowledge. Its hashing is SHAKE256, whose Keccak permutation is expensive in circuits; its Σ -protocol challenge is expanded by a Fisher–Yates shuffle whose memory accesses and stopping point both depend on the witness; and its arithmetic is split between $\mathbb{F}_2$ and a prime field, so a proof has to move values between the two and certify every crossing. Efficient constructions usually build *around* obstacles like these: they substitute a hash function that is easier to prove, or a friendlier variant of the challenge generation `SampleInBall`, or simplify the circuit elsewhere. When treating the scheme as fixed by the standard, so no workarounds are available, we then ask the following question:

*Can we prove knowledge of a ML-DSA signature
at a cost consumer hardware can carry?*

1.1 Our Contributions

We give the first blind signature whose issuer runs a *standardized* post-quantum signature scheme *unmodified*. Issuance is a ML-DSA signature on a commitment (which can be implemented using SHAKE256), so the issuer needs no new code and no new key material. A signature takes 0.82s to produce, 0.50s to verify, and is 1.01 MB, on consumer-grade hardware.

Given our construction, both issuer-hiding and selective disclosure are basically free on top. Issuer-hiding costs an additional 47 KB, and the proof size then barely grows with the number of issuers: 192 bytes separate a list of 27 issuers from one of 512. Selective disclosure over signed attributes, enough for a wallet to prove an age bound without revealing a birth date, costs another 27.4 KB. Neither extension uses anything specific to ML-DSA.

Underneath sits a contribution of independent interest. Proving ML-DSA verification means that a proof system either has to simulate $\mathbb{F}_2$ -arithmetic inside $\mathbb{F}_q$ (or vice versa) or has to pass values between $\mathbb{F}_2$ and $\mathbb{F}_q$. In a non-interactive proof, certifying those crossings the standard way using `edaBits` [21, 46] is the single largest item in the whole proof. We replace it with **edaBit in the Exponent** (`edaBitE`), an algebraic batch check that needs no cut-and-choose (i.e. combinatorial techniques) and applies to any VOLE-based proof crossing between a binary and a prime field. The check is independent of the VOLEitH proof system that we use, and might be applied for other use cases of VOLE-based zero knowledge or general secure computation protocols.

¹ The only other standardized post-quantum candidate is SLH-DSA [38], whose verification recomputes long chains of hashes and is a worse fit for a blind signature; see <https://blog.cloudflare.com/ml-dsa-will-have-to-do/> for an overview.

1.2 Technical Overview

Crossing between $\mathbb{F}_2$ and $\mathbb{F}_q$. Our circuit for ML-DSA verification works in two fields at once, $\mathbb{F}_2$ for the hashing and $\mathbb{F}_q$ for the field arithmetic. A VOLEitH proof over each is sound on its own, but nothing ties the two proof instances together: whenever the same value is used in both proof instances, one must therefore prove that an $\mathbb{F}_2$ commitment and an $\mathbb{F}_q$ commitment hold the same value. This is usually done with extended doubly authenticated bits (edaBits), and the state of the art is the cut-and-choose approach of Mystique [46] sacrificing a pool of $n(B_M - 1) + C_M$ random pairs of commitments to show consistency of n commitment pairs, where B_M is a tunable parameter called *bucket size* that depends on the required soundness error.

However, [21, 46] are calibrated for interactive proofs, where 2^{-40} interactive soundness error is customary and buckets have size three or four. Ours is non-interactive and has to reach $2^{-\lambda}$, which for a few hundred conversions forces $B_M = 14$, with every element of the generated pool carrying a committed $\mathbb{F}_q$ element on top of its bits. Our circuit passes only 366 values between the fields, and certifying just those costs about half a megabyte in proof size.

Consistency in the exponent. A conversion is a statement that an $\mathbb{F}_q$ commitment $[x_i]_q$ and the bits $([x_{i,j}]_2)_j$ the circuit already holds denote the same value, that is $x_i = \bar{x}_i$ for $\bar{x}_i := \sum_j 2^{j-1} x_{i,j}$. Our edaBitE check establishes this algebraically for a whole batch at once. The idea is to take a random linear combination of the candidates on each side and compare the two. On the $\mathbb{F}_q$ side this is immediate and linear. On the $\mathbb{F}_2$ side the bits have to be recombined into the same integer, which is not an $\mathbb{F}_2$ -linear operation. Rather than simulate $\mathbb{F}_q$ addition and multiplication inside an $\mathbb{F}_2$ circuit, we lift the statement *into the exponent* of a fixed base $X \in \mathbb{F}_{2^\lambda}$, where the check becomes a weighted product of commitments.

More concretely, an $\mathbb{F}_2$ -VOLE commitment holds a bit, but its algebra is over $\mathbb{F}_{2^\lambda}$, so the bit can be lifted into the exponent by computing

$$X^a = 1 + (X + 1) \cdot a,$$

since the right-hand side is 1 when $a = 0$ and X when $a = 1$; being affine in a , it can be evaluated on a commitment. The product over j with bases $X^{2^{j-1}}$ rebuilds $\bar{x}_i$ in the exponent, and raising those bases to a challenge χ_i makes the product over i the random linear combination:

$$\prod_i \prod_j \left(1 + (X^{\chi_i 2^{j-1}} + 1) \cdot [x_{i,j}]_2 \right) = X^{\sum_i \chi_i \bar{x}_i}.$$

The prover opens that exponent E and the verifier checks it against the same combination of the $\mathbb{F}_q$ commitments. The opening must hide E and, since X^E pins E only modulo $\text{ord}(X)$, bound it: a uniform κ -bit mask, wide enough to cover E , and one carry bit do the hiding, and the opening's fixed length $\kappa + 1$ is the range check. As the mask is committed in $\mathbb{F}_2$ and $\mathbb{F}_q$ alike, the difference of the two checks is free of it and reads $\sum_i \chi_i (x_i - \bar{x}_i) \equiv 0 \pmod q$: one inconsistent pair survives with probability $2/q$, and six repetitions bring that to $2^{-\lambda}$. The masks need no verification: their difference is fixed before the χ_i are sampled.

What it trades. Beyond the n pairs the circuit was going to commit anyway, edaBitE adds only the masks of each repetition – $2t$ elements in $\mathbb{F}_q$ and $t(\kappa + 1)$ bits in $\mathbb{F}_2$ – none of which need checking. What it costs is multiplicative degree: the product has $\sum_i k_i + \kappa$ factors. For a fixed degree bound $d_{\max}$ the prover needs to re-commit and prove consistency every $d_{\max} - 1$ factors, each reduction one $\mathbb{F}_{2^\lambda}$ commitment. The commitment cost falls off as $1/d_{\max}$ while the other savings remain fixed, so Mystique stays cheaper for a small $d_{\max}$ and edaBitE wins once $d_{\max}$ is large enough. The price it pays lands in $\mathbb{F}_{2^\lambda}$, where the Chinese remainder theorem (CRT) full-field VOLEs of the latest FAEST specification [5] make commitments cheap.

A verification circuit for ML-DSA. Our blind signature follows the blueprint of [4], so all that is left to build is a circuit that proves ML-DSA.Verify accepts. Because the issuer runs an *unmodified* implementation, we cannot substitute a friendlier but equivalent statement: the circuit has to follow that algorithm step for step. Algorithm 1 depicts a simplified version of the algorithm that we implement, allowing us to show how we solve the individual bottlenecks in the rest of this subsection. The ring arithmetic, as done in line 3 of the algorithm, is computed over $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^N + 1)$ for $N = 256$.

Algorithm 1 Simplified ML-DSA.Verify(pk, M', σ) [37, Algorithm 8]

Require: Public key $(\hat{\mathbf{A}}, \mathbf{tr}, \mathbf{t}_1)$, message $M' \in \{0, 1\}^*$, signature $(\tilde{\mathbf{c}}, \mathbf{z}, \mathbf{h})$

- 1: $\mu \leftarrow \text{SHAKE256}(\mathbf{tr} \parallel M')$ $\triangleright$ message representative
- 2: $\mathbf{c} \leftarrow \text{SampleInBall}(\tilde{\mathbf{c}})$
- 3: $\mathbf{w}_{\text{appr}} \leftarrow \hat{\mathbf{A}} \cdot \mathbf{z} - \mathbf{c} \cdot \mathbf{t}_1 \cdot 2^{13}$
- 4: $\mathbf{w}_1 \leftarrow \text{UseHint}(\mathbf{h}, \mathbf{w}_{\text{appr}})$
- 5: $\tilde{\mathbf{c}}' \leftarrow \text{SHAKE256}(\mu \parallel \text{w1Encode}(\mathbf{w}_1))$
- 6: **return** 1 iff $\|\mathbf{z}\|_\infty \leq \gamma_z$ and $\tilde{\mathbf{c}} = \tilde{\mathbf{c}}'$

Everything the verifier of the blind signature holds is public: the matrix $\hat{\mathbf{A}}$, the public-key digest $\mathbf{tr}$ and the rounded key $\mathbf{t}_1$ all come out of $\mathbf{pk}$, and M' is the commitment the issuer signed. The user proves it knows a signature: the challenge seed $\tilde{\mathbf{c}}$, the response $\mathbf{z}$ and the hint $\mathbf{h}$ (and also the randomness used in the commitment). The circuit of Algorithm 1 recomputes the rest from those, and the statement to be proven is simply that line 6 returns 1.

Line 2 expands the seed $\tilde{\mathbf{c}}$ into the challenge $\mathbf{c}$, lines 3–5 run forward from $\mathbf{c}$ to a fresh seed $\tilde{\mathbf{c}}'$, and line 6 compares it against the $\tilde{\mathbf{c}}$ contained in the signature. The algorithm requires two range checks: $\|\mathbf{z}\|_\infty \leq \gamma_z$ on line 6, and the bound on the low part of $\mathbf{w}_{\text{appr}}$ to check consistency of UseHint returning $\mathbf{w}_1$ in line 4.

The work done in the proof splits along the two fields as follows. Over $\mathbb{F}_2$ sit the two SHAKE256 calls of lines 1 and 5 and the SampleInBall of line 2. For SHAKE256 we reuse the circuit of [4] and only re-tune its parameters. Over $\mathbb{F}_q$ sit line 3's matrix-vector product and line 4's hint, where every product pairs

a committed value with a public one, so the arithmetic itself is free and only the two range checks cost. The circuit crosses between the two sides at exactly two points, 366 values in all, for which we use edaBitE. That leaves constructing efficient checks for **SampleInBall** and the two range checks, which we now describe.

SampleInBall and oblivious access. **SampleInBall** expands $\tilde{c}$ into a sparse challenge of length $N = 256$ by a Fisher–Yates shuffle whose swap positions come from the hash stream, so it is a data-dependent transformation of committed values: both the variables it accesses and the round it stops in depend on the secret input. A circuit used in a zero-knowledge proof cannot branch cheaply, so it must do the same work every round and hide which variables were read or changed. This suggests the use of an oblivious random access memory (ORAM) – over an array whose *elements* are committed as well. We benchmark and compare two designs. A linear ORAM reads and writes every one of the N positions each round, multiplexed by an equality indicator on the secret index; with only $N = 256$ entries this is viable, but it commits $2N$ values per round. A general ORAM [48] used in a zero-knowledge proof reads and writes what the round really needs, and performs a single consistency check after all ORAM accesses. We cast **SampleInBall** with a general ORAM access pattern. The design via a general ORAM commits 40 values each round, against the linear ORAM’s 528.

Range checks and digit decomposition. ML-DSA accepts only if $\|z\|_\infty$ is small, and recovering w_1 from the hint means decomposing w_{appr} into a high and a low part with the low part bounded. Both can be expressed as membership checks in $[0, C]$ for a public constant C , and both are 18 bits wide. Committing those 18 bits over $\mathbb{F}_2$ makes the bound almost free, but the value is then committed for the wrong modulus, and getting it back into $\mathbb{F}_q$ requires costly conversions. Committing base- B digits in $\mathbb{F}_q$ needs no conversion, but requires a circuit that asserts the vanishing product $\prod_{k < B} (x - k) = 0$ for each committed digit, in the style of [1]. Raising B then reduces the number of required commitments by increasing degree: fewer digits per coordinate, but a comparison of degree $\ell(B - 1)$, which runs from 18 at $B = 2$ to 189 at $B = 64$. These two checks and the hint they carry are three fifths of the proof size, so optimizing this matters; Section 4 benchmarks different choices.

Going beyond blind signatures. The verification circuit consumes the issuer’s public key as constants ($\hat{A}$, t_1 and tr). We hand it a public *table* containing the keys of n issuers instead, and let it select one obliviously. The signer’s index can be expressed in $\lceil \log n \rceil$ committed bits, a doubling gadget turns those into a one-hot vector f in both fields, and the row the rest of the circuit then reads is $\sum_j f[j] \cdot (tr_j, \hat{A}_j, t_{1,j})$. An OR-proof over the n candidate statements [20] would prove the same statement, but runs the circuit n times instead of once.

What it costs is that $\hat{A}$ and t_1 stop being constants, which increases the proof size by around 47 KB over its $N \cdot K$ coordinates (independent of n), plus the commitment to the index. However, the latter commitment barely impacts

proof size: 192 bytes separate $n = 27$ from $n = 512$. Prover and verifier time do grow, however, since both evaluate the n products before evaluating the signature verification circuit.

Attribute disclosure. We additionally let the issuer sign $\text{com}||\text{aux}$ for a block of attributes, which needs nothing new in our construction because the message representative μ is already a SHAKE256 evaluation the circuit proves. The attributes stay committed and are never opened; what the circuit proves over them are low-degree functions such as an age check, or that the credential has not expired – each a bitwise comparison against a public constant. The whole extension costs 27.4 KB: one more Keccak permutation, because including aux in the signature requires the use of an additional SHAKE256 absorption round. Further attributes cost further absorbs, which are however small compared to the size of the blind signature as a whole.

1.3 Related Work

Closest to our idea is zkDilithium [41], which also proves a Dilithium verifier in zero knowledge, but modifies the scheme to suit the proof system: it swaps SHAKE256 for an arithmetization-friendly hash and adapts SampleNBall, so what it proves is a variant of Dilithium rather than ML-DSA as standardized. Those two substitutions remove exactly what is expensive for us: the Keccak permutations, the shuffle, and arithmetic across different moduli. Closest in construction is PoMFRIT [4], a Fischlin-style VOLEitH blind signature whose conservative variant also leaves the signer running an ordinary signing algorithm. However, their work is built on the MAYO signature scheme [11], a NIST additional-signatures candidate that is currently in submission but not an approved standard. A similar approach was investigated in parallel by Bouillaguet et al. [16], who however only provide estimates of the respective signing and verification time. BRaccoon [27] is another related construction which builds on Raccoon [40], a previous NIST additional signature candidate. Similar to zkDilithium, they modify the existing signature towards better efficiency. For pre-quantum signatures, the most prominent work is longfellow-zk [23] which uses a succinct proof system instead.

Other post-quantum blind signature constructions. There exist a substantial number of lattice-based blind signature constructions [2, 3, 12, 13, 15, 28, 29, 32, 35, 39], most with much smaller communication than ours. None of them is built on a standardized algorithm, though, so none works for a signature a deployed device already produces. Additionally, some works such as [15, 35] rely on novel lattice-based assumptions that are not as well-established.

Interesting alternatives are code-based and isogeny-based blind signatures, such as [25, 26, 30, 33, 34]. As with the lattice constructions they show better performance than ours, but none builds upon standardized schemes.

Issuer-hiding. Issuer-hiding was introduced for attribute-based credentials in [14] and further studied in this setting in [31, 36, 44]. We cheaply obtain it as obliviously selecting an item from a public list is very efficient in our chosen proof system.

Concurrent work. Concurrently with our work, Cloudflare² published a VOLEitH-based proof of knowledge of an ML-DSA signature. Instead of committing to values in $\mathbb{F}_q$, they reformulate the verification circuit to work over $\mathbb{F}_2$ only, without fully simulating arithmetic modulo q . Their main variant derives the witness from a short seed via a quadratic local PRG and publishes only its difference to the witness, which reduces the proof to 64 KB at the cost of an *additional assumption*. Committing to the full $\mathbb{F}_2$ witness instead, their proof has a size of 913 KB, comparable to ours (about 1 MB), with substantially faster proving and verification times (123 ms / 58 ms), which we largely attribute to the optimized FAEST code base.

2 Preliminaries

Notation. We use λ to denote the statistical security parameter. Let $[x]$ for $x \in \mathbb{N}$ be short-hand for the set $\{1, \dots, x\}$.

2.1 Blind Signatures

A blind signature [19] lets a user obtain a signature on a message that the issuer never sees, and the issuer cannot later link to the interaction that produced it. We use the round-optimal, three-message form, and state it over a *list* of issuer keys $\mathcal{I} = (\mathbf{pk}_1, \dots, \mathbf{pk}_n)$. $|\mathcal{I}| = 1$ recovers the standard *single-issuer* notions.

Section C.1 gives the syntax: **KeyGen**, a signing protocol $(\text{Sign}_i)_{i \in [3]}$ and **Verify**, each taking λ , sometimes implicitly. Correctness says honestly generated signatures verify. Blindness says an issuer running the protocol twice cannot tell which signature came from which interaction. One-more unforgeability says Q completed runs yield valid signatures on at most Q distinct messages.

Correctness and blindness are standard, and Section C.1 gives them formally. A blind signature is *correct* if, for any message and any honestly generated $(\mathbf{sk}, \mathbf{pk})$ with $\mathbf{pk} \in \mathcal{I}$, an honest run of the Sign_i yields a $\mathbf{sig}$ that $\text{Verify}(\mathcal{I}, \mathbf{msg}, \mathbf{sig})$ accepts with overwhelming probability. It has *blindness* if no adversary that picks $\mathcal{I}$, runs the issuer’s side of two sessions on $\mathbf{msg}_0, \mathbf{msg}_1$ in an unknown order and is then shown both signatures can tell which session produced which.

For unforgeability the list of issuer keys has to be honestly generated. Queries and forgeries are counted per sub-list $\mathcal{J} \subseteq \mathcal{I}$, so that $\mathcal{J} = \mathcal{I}$ is the usual global bound and $\mathcal{J} = (\mathbf{pk}_j)$ bounds what is attributable to issuer j alone.

² <https://github.com/cloudflareresearch/vole-mole/pull/49>, published on 12.10.2026.

Definition 1 (One-more unforgeability). A (multi-issuer) blind signature $(\text{KeyGen}, (\text{Sign}_i)_{i \in [3]}, \text{Verify})$ is one-more unforgeable if for every polynomial-time adversary $\mathcal{A}$ there exists a negligible function negl such that

$$\Pr \left[\begin{array}{l} \forall 1 \leq i < j \leq \ell : (\text{msg}_i \neq \text{msg}_j), \\ \forall 1 \leq i \leq \ell : (\text{Verify}(\mathcal{I}_i, \text{msg}_i, \text{sig}_i) = 1), \\ \forall 1 \leq i \leq \ell : (\mathcal{I}_i \subseteq \mathcal{J} \subseteq \mathcal{I}), \\ \ell > Q_{\mathcal{J}} \end{array} \mid \begin{array}{l} \forall j \in [n] : (\text{pk}_j, \text{sk}_j) \leftarrow \text{KeyGen}(1^\lambda) \\ \mathcal{I} \leftarrow (\text{pk}_1, \dots, \text{pk}_n), \\ (\mathcal{J}, \{(\mathcal{I}_i, \text{msg}_i, \text{sig}_i)\}_{i \in [\ell]}) \leftarrow \\ \mathcal{A}(\text{Sign}_2(\text{sk}_j, \cdot))_{j \in [n]}(\mathcal{I}) \end{array} \right]$$

is bounded by $\text{negl}(\lambda)$, where $\mathcal{J}$ is the sub-list of issuer keys $\mathcal{A}$ names, and $Q_{\mathcal{J}}$ is the number of queries $\mathcal{A}$ made to the oracles $\text{Sign}_2(\text{sk}_j, \cdot)$ with $\text{pk}_j \in \mathcal{J}$; we write Q_j for $Q_{(\text{pk}_j)}$. Both ℓ and the total number of queries are polynomial in λ .

For $|\mathcal{I}| > 1$ the natural question is whether a signature reveals which issuer of $\mathcal{I}$ produced it. *Issuer-hiding* ensures that the verifier only learns that some issuer of the list vouched for the message, and nothing beyond that. The notion was introduced for attribute-based credentials by Bobolz et al. [14]. It applies to blind signatures directly, and we state it in the form used for the anonymity of ring signatures, where the adversary is handed every signing key. In the game (Definition 7, Section C.1) the adversary picks a list $\mathcal{I}$ and two indices i_0, i_1 whose keys it contains, one of them signs, and it must guess which. The remaining entries of $\mathcal{I}$ are unconstrained; together with holding every sk_j , this is the analog of anonymity under full-key-exposure from ring signatures [10].

2.2 ML-DSA

A blind signature in this work is a proof that ML-DSA.Verify accepts, so verification, and not signing, is what our circuit has to evaluate. We therefore introduce ML-DSA [37] through its verification alone, in the simplified form of Algorithm 1, together with the three of its subroutines that a circuit cannot get for free; Section C.2 states the algorithm in full and we refer to [37] for the rest. Every concrete number we quote is that of ML-DSA-44 , the parameter set we use.

Table 1: The ML-DSA-44 parameters used in this paper. We write ν for the challenge weight that [37] calls τ .

symbol	meaning	value
q	modulus of $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^N + 1)$	$2^{23} - 2^{13} + 1$
N	ring degree	256
(K, L)	dimensions of $\mathbf{A}$	(4, 4)
ν	non-zero coefficients of the challenge $\mathbf{c}$	39
γ_z	bound on $\ \mathbf{z}\ _\infty$, $\gamma_1 - \beta - 1$	130 993
γ_2	half-width of the low part $\mathbf{w}_0$	95 232
M	number of values of $\mathbf{w}_1$, $(q - 1)/(2\gamma_2)$	44
ω	maximum Hamming weight of the hint $\mathbf{h}$	80

What the Circuit Pays For. What Algorithm 1 leaves out of the standard’s own algorithm (Section C.2) is free in zero knowledge. `pkDecode` and `ExpandA` read public data only, so $\hat{\mathbf{A}}$ and $\mathbf{t}_1$ are computed outside the circuit, and `sigDecode` and `w1Encode` are bit reorderings. The NTT is $\mathbb{F}_q$ -linear and every product of the matrix-vector step pairs a committed value with a public one, leading to no extra cost. What remains are the four steps that are non-linear: the two SHAKE256 evaluations, `SampleInBall`, `UseHint`, and the norm bound on $\mathbf{z}$.

SHAKE256. Every hash in ML-DSA is SHAKE256, an extendable-output function (XOF) built from the Keccak permutation in a sponge that absorbs and squeezes at a rate of $r = 1088$ bits (136 bytes). Verification calls it twice: once to derive the message representative $\boldsymbol{\mu}$ from $\mathbf{tr}$ and the message, which for an empty context string is $\mathbf{M}' = 0\|0\|\mathbf{msg}$, and once to rehash $\boldsymbol{\mu}$ against the recovered $\mathbf{w}_1$ and compare the result with the seed $\tilde{\mathbf{c}}$ that produced the Σ -protocol challenge.

SampleInBall. `SampleInBall` (Algorithm 2) expands the seed $\tilde{\mathbf{c}} \in \{0, 1\}^{256}$ into the challenge $\mathbf{c} \in \mathcal{R}_q$, a polynomial with exactly ν non-zero coefficients, all ± 1 . It squeezes 8 bytes for the signs and then runs a Fisher–Yates shuffle whose positions come from rejection sampling over the remaining byte stream, one byte per attempt. The number of attempts is not fixed in advance. [37, Table 3] asks implementations not to bound it at all, and any that do to allow at least 221 bytes of SHAKE256 output per call – a limit a correct execution reaches with probability about 2^{-256} . Past the 8 sign bytes that is $221 - 8 = 213$ iterations.

Algorithm 2 `SampleInBall`($\tilde{\mathbf{c}}$) [37, Algorithm 29]

Require: Seed $\tilde{\mathbf{c}} \in \{0, 1\}^{256}$

Ensure: $\mathbf{c} \in \mathcal{R}_q$ with ν non-zero coefficients, all ± 1

```

1:  $\mathbf{c} \leftarrow 0^N$ 
2:  $\text{ctx} \leftarrow \text{SHAKE256.Init}()$ ,  $\text{ctx} \leftarrow \text{SHAKE256.Absorb}(\text{ctx}, \tilde{\mathbf{c}})$ 
3:  $(\text{ctx}, s) \leftarrow \text{SHAKE256.Squeeze}(\text{ctx}, 8)$ 
4:  $\mathbf{b} \leftarrow \text{BytesToBits}(s)$   $\triangleright \nu$  sign bits,  $\nu \leq 64$ 
5: for  $i = N - \nu$  to  $N - 1$  do
6:   repeat  $\triangleright$  at most  $\theta_{\max}$  iterations in total
7:      $(\text{ctx}, j) \leftarrow \text{SHAKE256.Squeeze}(\text{ctx}, 1)$ 
8:   until  $j \leq i$ 
9:    $c_i \leftarrow c_j$ ,  $c_j \leftarrow (-1)^{b[i+\nu-N]}$   $\triangleright$  swap, then place the sign
10: return  $\mathbf{c}$ 
```

Decompose and UseHint. The signer commits to the high part of $\mathbf{w}$ and sends a hint that lets the verifier recover it from the approximation $\mathbf{w}_{\text{appr}}$ alone. `Decompose` splits $r \in \mathbb{Z}_q$ into $r_1 \in [0, M)$ and a low part r_0 with $r \equiv 2\gamma_2 r_1 + r_0$, and `UseHint` moves r_1 by ± 1 modulo M when the hint bit is set (Section C.2).

2.3 VOLE-in-the-Head NIZKs

We use standard definitions of for Non-interactive Zero-Knowledge Proofs of Knowledge (NIZKs) and recall their definition in Section C.3. All the NIZKs in our construction are VOLE-in-the-Head (VOLEitH) proofs [7]. We introduce these as Commit-and-Prove NIZKs on an abstract level. We follow the presentation of [4], which is compatible with the optimizations of [1, 6]. A more thorough introduction can be found in [9].

Commitments from VOLEs. We begin by defining the notion of a subfield VOLE.

Definition 2 (Subfield VOLE). *Let $\mathbb{F}$ be a finite field with a finite extension $\mathbb{F}_{\text{ext}}$ of size at least 2^λ . Moreover, let P and V denote two parties usually considered as prover and verifier. A subfield VOLE correlation between both parties has P holding vectors $\mathbf{u} \in \mathbb{F}^\ell$ and $\mathbf{v} \in \mathbb{F}_{\text{ext}}^\ell$, while V has a global key $\Delta \in \mathbb{F}_{\text{ext}}$ together with $\mathbf{x} = \mathbf{u}\Delta - \mathbf{v}$.*

Each triple (u_i, v_i, x_i) is an information-theoretic message authentication code (MAC) on u_i , opened when P sends u_i, v_i : P cannot change u_i without knowing Δ , and v_i hides it from V . We write $[u_i]_q$ for such a correlation where $\mathbb{F} = \mathbb{F}_q$ and call it a commitment: in a designated-verifier setting, a MAC is one.

Evaluating $\mathbb{F}$ -circuits in NIZKs using VOLE Commitments. We assume that the NIZK can generate random subfield VOLE correlations between P and V over $\mathbb{F}$, and that $w \in \mathbb{F}^*$ is a witness for a statement x describing a circuit with $\mathbb{F}$ consisting of addition and multiplication gates with a single output wire. We say that $(x, w) \in \mathcal{R}$ iff the circuit described by x on input w evaluates to 0.

Linear operations: Because the VOLE relation is linear in Δ , linear operations can be performed on the commitments without interaction between P and V . The linear coefficients can come from $\mathbb{F}_{\text{ext}}$ which lead to committed secrets being from $\mathbb{F}_{\text{ext}}$ instead of the base field $\mathbb{F}$. This fact has no impact on the soundness of the NIZK but is crucial to this work.

Inputs: Linear operations allow P to derandomize random VOLE correlations to w by sending the difference to the random VOLE secret to V .

Output: To verify the output of a circuit evaluation on commitments, P opens the output commitment (in our work: *assert* its value) towards the verifier.

Multiplications: Multiplication in VOLEitH is handled as in QuickSilver [47] by letting commitments carry a *degree*. A degree- d commitment $[a]$ means that P holds a polynomial ρ_a of degree d whose leading coefficient is a , while V holds the value $\rho_a(\Delta)$. A VOLE relation is the case $d = 1$ with $\rho_a(t) = at - v$. Parties multiply commitments locally, where P sets $\rho_{ab} = \rho_a \cdot \rho_b$ while V computes $\rho_a(\Delta) \cdot \rho_b(\Delta)$. Multiplication is free of communication, but the degree of the resulting commitment equals the sum of the degrees of ρ_a, ρ_b .

Asserts and degree reduction. The presented multiplication method leads to increases in computation and memory use by P as degrees of commitments grow. Moreover, it increases the communication cost of asserting a value of a commitment. Namely, to assert $a = 0$ for a degree- d commitment $[a]$, P sends ρ_a masked by $d-1$ fresh random VOLEs correlations, where all masking VOLE correlations must have secrets over $\mathbb{F}_{\text{ext}}$ rather than $\mathbb{F}$ for this to be zero-knowledge. V checks both that the result evaluates at Δ to its local value $\rho_a(\Delta)$ and that the degree is $d-1$ (as $a = 0$). A non-zero a fails this check except with probability $d/2^\lambda$. Since prover runtime and proof size scale with the largest degree of an assert, a circuit has a degree budget rather than a multiplication budget, and we let d_{max} denote that bound. A commitment that would exceed it must be *degree-reduced*: the prover commits to its secret again using an independent random degree-1 VOLE and asserts the difference of both commitments to be 0.

Bits are committed with $\mathbb{F} = \mathbb{F}_2$ inside $\mathbb{F}_{2^\lambda}$, and ML-DSA elements with $\mathbb{F} = \mathbb{F}_q$ for the prime $q = 2^{23} - 2^{13} + 1$, inside $\mathbb{F}_{q^r}$ for the smallest r carrying λ bits, $r = 6$. We write $[x]_2$ and $[x]_q$ for the instance a commitment belongs to.

2.4 From VOLE to VOLEitH

The aforementioned proof system describes a designated-verifier construction, since the binding of the commitments relies on P not learning Δ . In [7], the authors described the VOLEitH technique that allows public verifiability and thus the construction of Commit-and-Prove VOLE NIZKs (in the random-oracle model). Their main two observations are that the aforementioned proof system still works even if the verifier learns its share of the VOLE correlation and in fact also Δ after the prover finished its proof. In addition, such VOLE correlations can be generated via [42] from τ vector commitments, which in turn can be cheaply generated and opened using GGM and Merkle trees. Here, τ is a parameter of the proof system that trades off communication and computation.

A VOLEitH proof can be split into two phases: We write π_1 for the prover’s first message, which carries the vector commitments together with additional values necessary to generate consistent subfield VOLEs. All remaining messages generated as part of the NIZK are then denoted as π_2 . One committed $\mathbb{F}_2$ value communicates τ bits in VOLEitH, while one committed $\mathbb{F}_q$ value requires $\tau \lceil \log_2 q \rceil$ bits when packed tightly. At $\lambda = 128$ and $\tau = 16$ that results in 2 B and 46 B per commitment respectively. However, these are *subfield VOLE* commitments: to perform an *assert* statement for a commitment of degree d , P requires $d-1$ random $\mathbb{F}_{\text{ext}}$ -VOLE correlations. While those can be generated from $\mathbb{F}_2$ or $\mathbb{F}_q$ -correlations, this substantially increases the proof size as d grows.

Full-Field random VOLEs via CRT. The original VOLEitH approach from [7] only generates random VOLE correlations where the secret lies in a subfield. The latest FAEST specification [5] replaces this with an efficient algorithm where $\bar{u} \in \mathbb{F}_{\text{ext}}$. To do so, they represent $\bar{u}$ and the challenge Δ as polynomials over $\mathbb{F}_2$ of degree less than λ . $\bar{u}\Delta$ is then determined by its residues modulo any set of pairwise coprime moduli whose degrees sum to (roughly) 2λ . We refer

to [5] for the full construction and its security proof. Our implementation makes use of the [5] VOLE correlation generation algorithm. Running their reference table generator on our parameter set ($\tau = 16$, no grinding) yields the same $n_{\text{mult}} \approx 2.5\lambda$ as the FAEST sets [5, Appendix B]: 39 B per $\mathbb{F}_{2^\lambda}$ correlation.

Conversions between $\mathbb{F}_2$ and $\mathbb{F}_q$. To prove the ML-DSA verification circuit, we run two instances of the aforementioned NIZK proof in parallel. However, additional care must be taken to prove equality of values committed in $\mathbb{F}_2$ and $\mathbb{F}_q$. Towards this, we recapitulate the notion of an edaBit.

Definition 3 (edaBit). *An edaBit of width $m := \lceil \log_2 q \rceil$ is a pair of commitments*

$$\left([a]_q, ([a_1]_2, \dots, [a_m]_2) \right)$$

such that $a = \sum_{j \in [m]} a_j 2^{j-1}$.

edaBits were introduced for secure multi-party computation (MPC) by Escudero et al. [21] and brought to VOLE-based zero knowledge proofs by Mystique [46] (and independently [8]). Applying their techniques to commitments in a Boolean circuit, the prover can commit the $\mathbb{F}_q$ element of an edaBit directly, and symmetrically an $\mathbb{F}_q$ element's bit decomposition can be committed directly in $\mathbb{F}_2$. The resulting pairs are then checked using the consistency protocol of [21]. We write **ConvertB2A** and **ConvertA2B** for the two directions, and what a conversion costs equals the costs to verify well-formedness of the pair.

Mystique establishes well-formedness by *cut-and-choose with bucketing* [46, Section 4.1]. Its protocol uses a set of $n(B_M - 1) + C_M$ random committed candidate edaBits and a random selection of C_M of these edaBits is opened in full to check that the relation between the $\mathbb{F}_2$ and $\mathbb{F}_q$ commitments is consistent. The remaining $n(B_M - 1)$ random edaBits are randomly partitioned into n buckets, with each of the circuit's n conversion pairs assigned to one of the buckets for checking via *sacrificing* the $B_M - 1$ random edaBits in the bucket using a combine-and-open check.

Lemma 1 (Soundness of Cut-and-Choose with Bucketing, [46, Lemma 1]). *Let $C_M \geq B_M - 1$. If at least one of the n edaBit pairs is inconsistent, the check accepts with probability at most $\binom{n(B_M-1)+C_M}{B_M-1}^{-1}$.*

The only other edaBits algorithm we know of is due to Agarwal et al. [1]. They extend a small cut-and-choose-verified seed with a low-degree pseudorandom generator (PRG) simultaneously in $\mathbb{F}_2$ and $\mathbb{F}_q$ rather than bucketing each instance. However, this introduces an additional assumption lacking substantial cryptanalysis. Additionally, we later only need a small number of conversions – on the order of the seed length or below.

3 edaBits in the Exponent (edaBitE)

We now describe the edaBitE approach in more detail and prove its security. The main idea enabling the approach is the following lemma, which shows how to lift a binary commitment into the exponent. This lets us compute a linear combination of $\mathbb{F}_2$ secrets without $\mathbb{F}_2$ addition or multiplication circuits.

Lemma 2 (Bit Commitment as Exponent). *For bits $b_1, \dots, b_k \in \{0, 1\}$, $b = \sum_{j \in [k]} b_j 2^{j-1}$ and any $X \in \mathbb{F}_{2^\lambda}$,*

$$\prod_{j \in [k]} \text{elift}_X(2^{j-1}, b_j) = X^{\sum_j b_j 2^{j-1}} = X^b,$$

where $\text{elift}_Y(\text{exp}, a) := 1 + (Y^{\text{exp}} + 1) \cdot a$ for a base Y .

Proof. For a single bit a and any exponent exp ,

$$1 + (X^{\text{exp}} + 1) \cdot 0 = \text{elift}_X(\text{exp}, 0) = 1 = X^0$$

if $a = 0$ and

$$1 + (X^{\text{exp}} + 1) = \text{elift}_X(\text{exp}, 1) = X^{\text{exp}}$$

if $a = 1$ (using $1 + 1 = 0$ in characteristic two). The j -th factor is therefore $X^{b_j 2^{j-1}}$, so the product over $j \in [k]$ is X^b . $\square$

Applied to committed bits $[b_j]_2$, the left-hand side of Lemma 2 reconstructs a value *in the exponent*, and only the product across the k factors accumulates degree. The total degree is thereby k , assuming all commitments at the beginning of the computation are of degree 1.

We first describe the idea of edaBitE (Algorithm 3) without hiding. Fix the public base X to be a generator of $\mathbb{F}_{2^\lambda}^\times$, so that $\text{ord}(X) = 2^\lambda - 1$, and take n candidate pairs, $([x_i]_q, ([x_{i,1}]_2, \dots, [x_{i,k_i}]_2))_{i \in [n]}$, and recall $\bar{x}_i = \sum_{j \in [k_i]} x_{i,j} 2^{j-1}$. After fixing the commitments, draw $\chi_1, \dots, \chi_n \leftarrow \mathbb{F}_q$. The prover computes

$$e := \sum_{i \in [n]} \chi_i x_i \bmod q,$$

and reveals the integer $E := \sum_{i \in [n]} \chi_i \bar{x}_i$ for which

$$P := \prod_{i \in [n]} \prod_{j \in [k_i]} \text{elift}_{X^{\chi_i}}(2^{j-1}, [x_{i,j}]_2) = X^E.$$

The latter relation is over $\mathbb{F}_{2^\lambda}$ and uses Lemma 2. The verifier accepts if $0 \leq E < \text{ord}(X)$, $P = X^E$ and $E \equiv e \bmod q$.

Lemma 3 (Soundness without Hiding). *Let every commitment be fixed before $\chi_1, \dots, \chi_n$ are drawn from $\mathbb{F}_q$, and let $q \sum_{i \in [n]} 2^{k_i} \leq \text{ord}(X)$. If $\bar{\mathbf{x}} \neq \mathbf{x}$ as vectors in $\mathbb{F}_q^n$, then $0 \leq E < \text{ord}(X)$, $P = X^E$ and $E \equiv e \pmod{q}$ hold simultaneously with probability at most $1/q$ over $\chi_1, \dots, \chi_n$, independent of X .*

Proof. Once the commitments are fixed, Lemma 2 gives $P = X^D$ for the integer $D := \sum_{i \in [n]} \chi_i \bar{x}_i$ determined by the committed bits. Since $X^E = P$ determines E only modulo $\text{ord}(X)$, the range check leaves the single candidate $E = D \bmod \text{ord}(X)$, and $D < q \sum_i 2^{k_i} \leq \text{ord}(X)$ ensures $E = D$. Acceptance therefore requires $D \equiv e \pmod{q}$, that is $\sum_{i \in [n]} \chi_i (\bar{x}_i - x_i) \equiv 0 \pmod{q}$. Pick i_0 with $\bar{x}_{i_0} \neq x_{i_0}$ in $\mathbb{F}_q$. After fixing all other $\chi_i, i \neq i_0$ there is a unique χ_{i_0} that must be sampled such that the equation holds. The probability for this event is $1/q$. $\square$

Remark 1. Repeating the check with independent samples of χ means that the soundness error of the check multiplies. Thus, by repeating it $\lceil \lambda / \log_2 q \rceil$ times it can be reduced to $2^{-\lambda}$. For $\lambda = 128$ and $q = 2^{23} - 2^{13} + 1$, this is 6 repetitions.

Size of the Exponent. The range check on E is necessary as X^E fixes E only modulo $\text{ord}(X)$, whereas $E \equiv e \pmod{q}$ is a statement about the integer E . A prover with $\bar{x} \neq x$ could otherwise set $D := \sum_i \chi_i \bar{x}_i$, compute e as before, and send $E := D + a \cdot \text{ord}(X)$ for $a \equiv (e - D) \cdot \text{ord}(X)^{-1} \bmod q$. Without the range check, this E would be accepted for arbitrary commitments. Since $\chi_i < q$ and $\bar{x}_i < 2^{k_i}$, the honest exponent satisfies $E < q \sum_i 2^{k_i} \leq 2^\kappa$ for

$$\kappa := \left\lceil \log_2 \sum_{i \in [n]} 2^{k_i} \right\rceil + \lceil \log_2 q \rceil,$$

and we assume from here that $2^{\kappa+1} < \text{ord}(X)$.

Making the check Zero Knowledge. So far, revealing e, E leaks information about the secrets $x_{i,j}$. To make the check simulatable, we add two masks, each committed in both $\mathbb{F}_2$ and $\mathbb{F}_q$: a κ -bit mask s_{low} for the lower bits, and a single bit s_{high} for the carry of $s_{\text{low}} + E$. Then, write the masked sum as

$$E + s_{\text{low}} = c \cdot 2^\kappa + E_{\text{low}},$$

where the carry $c \in \{0, 1\}$ is a bit because $E, s_{\text{low}} < 2^\kappa$. The prover opens $E_{\text{low}} \parallel \beta \in \{0, 1\}^{\kappa+1}$, where $\beta := c \oplus s_{\text{high}}$ is the masked carry. The fixed length of this opening enforces $E_{\text{low}} < 2^\kappa$. Both sides now recompute the carry as $[c]_2 = [s_{\text{high}}]_2 + \beta$ and $[c]_q = \beta + (1 - 2\beta)[s_{\text{high}}]_q$. The verifier accepts if

$$\underbrace{P \cdot \prod_{\ell \in [\kappa]} \text{elift}_X(2^{\ell-1}, [s_{\text{low}, \ell}]_2)}_{=X^{E+s_{\text{low}}}} = \underbrace{X^{E_{\text{low}}} \cdot \text{elift}_X(2^\kappa, [c]_2)}_{=X^{c \cdot 2^\kappa + E_{\text{low}}}} \quad \text{and}$$

$$\sum_{i \in [n]} \chi_i [x_i]_q + [s_{\text{low}}]_q = 2^\kappa [c]_q + E_{\text{low}} \pmod{q}$$

hold. The left-hand side of the first check has degree $\sum_{i \in [n]} k_i + \kappa$ over $\mathbb{F}_{2^\lambda}$; all others degree 1. Both underbraces use the trick from Lemma 2: on the left applied to the candidates and to s_{low} 's bits, and on the right to the single bit

c at position κ . Since $E_{\text{low}} < 2^\kappa$ and $c \in \{0, 1\}$, both exponents stay below $2^{\kappa+1} < \text{ord}(X)$, so the first check is equivalent to the integer identity

$$\sum_{i \in [n]} \chi_i \bar{x}_i + s_{\text{low}} = E_{\text{low}} + c 2^\kappa.$$

The second check asserts the relation modulo q . We summarize this analysis:

Lemma 4 (Perfect Hiding). *Fix any $E < 2^\kappa$ and let $s_{\text{low}} \leftarrow \{0, \dots, 2^\kappa - 1\}$, $s_{\text{high}} \leftarrow \{0, 1\}$ be uniformly random and independent. Write*

$$E_{\text{low}} := (E + s_{\text{low}}) \bmod 2^\kappa, \quad c := \lfloor (E + s_{\text{low}}) / 2^\kappa \rfloor, \quad \beta := c \oplus s_{\text{high}}.$$

Then (E_{low}, β) are uniformly random in $\{0, \dots, 2^\kappa - 1\} \times \{0, 1\}$.

Proof. Adding a uniform value modulo 2^κ is a bijection of $\mathbb{Z}_{2^\kappa}$, so E_{low} is uniform for every fixed E . Conditioned on E_{low} the carry is a fixed value c , which the uniform s_{high} then masks perfectly. $\square$

In Algorithm 3 below we describe the full check, putting together the technique from Lemma 3 with Lemma 4.

Algorithm 3 edaBitE

Require: Candidate pairs $([x_i]_q, ([x_{i,j}]_2)_{j \in [k_i]})_{i \in [n]}$ where $k_i \leq \lceil \log_2 q \rceil$
Public base $X \in \mathbb{F}_{2^\lambda}^\times$; $\kappa := \lceil \log_2 \sum_{i \in [n]} 2^{k_i} \rceil + \lceil \log_2 q \rceil$, with $2^{\kappa+1} < \text{ord}(X)$
Ensure: $x_i = \bar{x}_i$ in $\mathbb{F}_q$ for every $i \in [n]$, where $\bar{x}_i := \sum_{j \in [k_i]} x_{i,j} 2^{j-1}$
At $k_i = \lceil \log_2 q \rceil$, assert a bitwise length check over the $\mathbb{F}_2$ commitments: $\bar{x}_i < q$.
1: **for** $r = 1$ to t **do**
2: $s_{\text{low}}^{(r)} \leftarrow \{0, 1\}^\kappa$, committed as $([s_{\text{low}, \ell}^{(r)}]_2)_{\ell \in [\kappa]}$ and $[s_{\text{low}}^{(r)}]_q$ $\triangleright$ low-part mask
3: $s_{\text{high}}^{(r)} \leftarrow \{0, 1\}$, committed as $[s_{\text{high}}^{(r)}]_2$ and $[s_{\text{high}}^{(r)}]_q$ $\triangleright$ carry mask
4: $(\chi_1^{(r)}, \dots, \chi_n^{(r)}) \leftarrow \mathbb{F}_q^n$ for $r \in [t]$
5: **for** $r = 1$ to t **do**
6: $E^{(r)} \leftarrow \sum_{i \in [n]} \chi_i^{(r)} \bar{x}_i \in \mathbb{Z}$, $E_{\text{low}}^{(r)} \leftarrow (E^{(r)} + s_{\text{low}}^{(r)}) \bmod 2^\kappa$
7: $c^{(r)} \leftarrow \lfloor (E^{(r)} + s_{\text{low}}^{(r)}) / 2^\kappa \rfloor$, $\beta^{(r)} := c^{(r)} \oplus s_{\text{high}}^{(r)}$
8: Publish $E_{\text{low}}^{(r)} \parallel \beta^{(r)} \in \{0, 1\}^{\kappa+1}$ $\triangleright$ masked low part and carry
Algebraic consistency check in the circuit:
9: $[c^{(r)}]_2 \leftarrow [s_{\text{high}}^{(r)}]_2 + \beta^{(r)}$, $[c^{(r)}]_q \leftarrow \beta^{(r)} + (1 - 2\beta^{(r)})[s_{\text{high}}^{(r)}]_q$
10: $P^{(r)} \leftarrow \prod_{i \in [n]} \prod_{j \in [k_i]} \text{elift}_{X^{\times i}}(2^{j-1}, [x_{i,j}]_2)$
11: $Q^{(r)} \leftarrow \prod_{\ell \in [\kappa]} \text{elift}_X(2^{\ell-1}, [s_{\text{low}, \ell}^{(r)}]_2)$
12: **assert** $P^{(r)} \cdot Q^{(r)} = X^{E_{\text{low}}^{(r)}} \text{elift}_X(2^\kappa, [c^{(r)}]_2)$ $\triangleright$ over $\mathbb{F}_{2^\lambda}$
13: **assert** $\sum_{i \in [n]} \chi_i^{(r)} [x_i]_q + [s_{\text{low}}^{(r)}]_q = 2^\kappa [c^{(r)}]_q + E_{\text{low}}^{(r)}$ $\triangleright$ over $\mathbb{F}_q$
 $P^{(r)} \cdot Q^{(r)}$ gets degree-reduced every $d_{\text{max}} - 1$ factors over $\mathbb{F}_{2^\lambda}$.

To see why it should be immediate to combine both Lemmas, note that subtracting one proven equation from the other cancels the masks and the carry, leaving precisely $\sum_{i \in [n]} \chi_i (x_i - \bar{x}_i) \equiv 0 \pmod q$ from the proof of Lemma 5.

Consistency of the Masks. Both s_{low} and s_{high} used in the $\mathbb{F}_q$ check are committed before the challenges χ_i are drawn. They need not be checked for consistency with their $\mathbb{F}_2$ commitments: any difference is fixed before the challenges are drawn, so it only shifts the target of the check by a constant and leaves the argument of Lemma 3 intact.

Lemma 5 (Soundness with Hiding). *Assume $\bar{x}_i < q$ for all $i \in [n]$. Write $\delta_i := x_i - \bar{x}_i$, $\delta_{\text{low}} := [s_{\text{low}}]_q - (s_{\text{low}} \bmod q)$ and $\delta_{\text{high}} := [s_{\text{high}}]_q - s_{\text{high}}$, let every commitment be fixed before χ is drawn, and let $E_{\text{low}} < 2^\kappa$. If $\delta_i \neq 0$ for some i then the checks in Algorithm 3 pass with probability at most $2/q$, for arbitrary δ_{low} and δ_{high} .*

Proof. The assumption $\bar{x}_i < q$ restricts δ_i to $(-q, q)$, so $\delta_i \equiv 0 \bmod q$ iff $\delta_i = 0$. Substituting errors in the check and cancelling the integer identity above leaves

$$\sum_{i \in [n]} \chi_i \delta_i \equiv -\delta_{\text{low}} + 2^\kappa(1 - 2\beta) \delta_{\text{high}} \bmod q,$$

whose right-hand side is fixed by the commitments except for the opened bit β . Pick i_0 with $\delta_{i_0} \neq 0$ and condition on $(\chi_i)_{i \neq i_0}$. As q is prime and $\delta_{i_0} \neq 0$ in $\mathbb{F}_q$, it is invertible, so for either value of β exactly one $\chi_{i_0} \in \mathbb{F}_q$ satisfies the congruence. The failure event is the union of those two and therefore happens with probability at most $2/q$. $\square$

Note that for applying Lemma 4 in practice we can let s_{low} and s_{high} simply be the values of random VOLE correlations. The factor 2 in Lemma 5 does not impact the total number of repetitions in our application, because $(2/q)^6 \approx 2^{-132}$. We now summarize all proven statements about Algorithm 3:

Theorem 1 (edaBitE). *Let q be prime, let X generate $\mathbb{F}_{2^\lambda}^\times$, and let the candidate pairs, their widths, and the parameters κ and t be as in Algorithm 3. Then Algorithm 3 satisfies the following.*

- Completeness. *If $x_i = \bar{x}_i$ in $\mathbb{F}_q$ for every $i \in [n]$, every assert of every repetition holds.*
- Soundness. *If $x_i \neq \bar{x}_i$ in $\mathbb{F}_q$ for some $i \in [n]$, then all $2t$ asserts hold with probability at most $(2/q)^t \leq 2^{-\lambda}$ over the challenges.*
- Zero knowledge. *The protocol transcript can be perfectly simulated without knowledge of x_i .*

Proof. Completeness follows as Lemma 2 makes the $\mathbb{F}_{2^\lambda}$ assert the integer identity $E^{(r)} + s_{\text{low}}^{(r)} = c^{(r)}2^\kappa + E_{\text{low}}^{(r)}$, which is how P obtains $E_{\text{low}}^{(r)}$ and $c^{(r)}$ to begin with, and the $\mathbb{F}_q$ assert is that identity modulo q with $x_i = \bar{x}_i$.

Soundness follows from applying Lemma 5 once per repetition. Its requirements are met: the published $\kappa+1$ bits have a fixed length, so $E_{\text{low}}^{(r)} < 2^\kappa$ whatever P sends, and Algorithm 3 guarantees $\bar{x}_i < q$, by the width bound. Since the commitments of every repetition are fixed before $\chi^{(r)}$ is sampled, the bad event of repetition r is determined by $\chi^{(r)}$ alone and happens with probability at most $2/q$; the $\chi^{(r)}$ are independent across repetitions.

Zero knowledge follows from Lemma 4, being applied once per repetition. $\square$

Remark 2. edaBitE requires only few extra commitments in $\mathbb{F}_q$, a small mask in $\mathbb{F}_2$, and commitments for degree reduction in $\mathbb{F}_{2^\lambda}$. Moreover, it is a batch consistency check: it takes n candidate edaBit pairs $([x_i]_q, ([x_{i,j}]_2)_{j \in [k_i]})$ and accepts only if $x_i = \bar{x}_i$ in $\mathbb{F}_q$ for all of them, where $\bar{x}_i := \sum_{j \in [k_i]} x_{i,j} 2^{j-1}$.

A further advantage is that widths may be mixed: each candidate edaBit pair i may use a different width k_i , and only $\sum_{i \in [n]} k_i$ determines the cost. What the check does assume of a candidate is $\bar{x}_i < q$ (Lemma 5). For candidates with $k_i \leq \log_2 q$ this is guaranteed regardless. For full width ($k_i = \lceil \log_2 q \rceil$) candidates a bitwise length check over the $\mathbb{F}_2$ commitments can enforce the bound on $\bar{x}_i$, requiring an assert of a degree $\lceil \log q \rceil$ commitment determined by a range check.

Comparison with Mystique. Mystique’s cut-and-choose sacrifices $(B_M - 1)n + C_M$ edaBits in both fields (Lemma 1); edaBitE adds nothing to the n pairs beyond the masks per repetition: $2t$ elements in $\mathbb{F}_q$ and $t(\kappa + 1)$ bits in $\mathbb{F}_2$, none of which need checking. These are for the whole batch rather than per width, since one check covers all. In comparison, [46] must fix the maximal edaBit width ahead of time, leading to wasteful generation of edaBits that are too large.

However, in our check the product has $\sum_{i \in [n]} k_i + \kappa$ factors per repetition and accumulates one degree per factor, so with $d_{\max}$ the degree bound in $\mathbb{F}_{2^\lambda}$ we reduce every $d_{\max} - 1$ factors. Each reduction commits to the running product by taking a random $\mathbb{F}_{2^\lambda}$ correlation (39 B, Section 2.4) and publishing the λ -bit difference between the product and that correlation’s random value (16 B), so 55 B in total. This cost falls off as $1/d_{\max}$, while the $\mathbb{F}_q$ -side saving is fixed once n , B_M and C_M are chosen. For a small $d_{\max}$ the reduction cost dominates and Mystique remains cheaper overall, while for a large enough $d_{\max}$ it is outweighed by the unconditional savings of edaBitE.

An exact comparison between the approaches depends on the candidate pairs, the proof system’s degree bound $d_{\max}$ and, if VOLEitH is used. Table 2 compares the two consistency checks for $n = 512$ candidates at full width.

Table 2: Certifying $n = 512$ pairs of the full width $m = 23$, at $B_M = 14$, $C_M = 13$, $t = 6$, $d_{\max} = 197$ and $\kappa = 55$. Sizes are at $\lambda = 128$ and $\tau = 16$, with the $\mathbb{F}_{2^\lambda}$ rate from Section 2.4, and include the openings. The second number 256 B generates each $\mathbb{F}_{2^\lambda}$ commitment naïvely, as λ subfield commitments of τ bits each.

	$\mathbb{F}_q$ (46 B)	$\mathbb{F}_2$ (2 B)	$\mathbb{F}_{2^\lambda}$ (55/256 B)	Openings (bits)	Size (KB)
<i>Mystique</i> [46]					
The n candidate pairs	512	11 776	—	—	47.1
Sacrificed edaBits	6669	153 387	—	153 387	632.7
Total	7181	165 163	—	153 387	679.8
<i>edaBitE</i>					
The n candidate pairs	512	11 776	—	—	47.1
Mask pairs $s_{\text{low}}, s_{\text{high}}$	12	336	—	336	1.3
Degree reductions	—	—	360	—	19.8/92.2
Total	524	12 112	360	336	68.2/140.5

4 Implementing the Blind Signature from MLDSA

Conceptually, we follow the standard Fischlin-style approach for single-issuer blind signatures [22]. The user sends the issuer a commitment $\text{com} = H(\text{msg}, \pi_1) \oplus \mathbf{r}$ to its message, the issuer signs com with an unmodified ML-DSA implementation, and the blind signature is a proof that ML-DSA.Verify accepts that signature on that commitment. Both $\mathbf{r}$ and π_1 come from the proof itself – $\mathbf{r}$ is VOLE randomness reserved for the purpose and π_1 is the proof’s first message. This is the construction of [4] (details recalled in Section D with the resulting scheme Algorithm 11). Only the circuit $C_{\text{pk}, \text{msg}}$ that the proof is about changes, and that is the subject of this section. We can alternatively let the commitment be $H(\text{msg}, \mathbf{r})$, at the expense of an additional SHAKE evaluation inside the proof.

The circuit has several components, and we sketch the whole of it in Algorithm 4 mirroring ML-DSA verification (Algorithm 1). We treat each component in turn, before we present benchmarks and explain the final cost estimates. We analyze the components via standalone circuits to explain our design decisions.

Algorithm 4 Full Circuit Layout for BMLDSA $C_{\text{pk}, \text{msg}}$

- 1: Obtain public inputs: $(\hat{\mathbf{A}}, t_1, \mathbf{tr}) \leftarrow \text{pk}, H(\text{msg}, \pi_1)$
 - 2: Obtain private inputs: $[\mathbf{r}]_2, [\tilde{\mathbf{c}}]_2, [\mathbf{c}]_q, [\tilde{\mathbf{z}}]_q = [\text{Decomp}(\mathbf{z})]_q, [\tilde{\mathbf{w}}_0]_q, [\mathbf{h}]_q$
 - 3: Implicit private inputs: $[\text{st}_{\mathbf{c}}]_2, [\text{st}_{\boldsymbol{\mu}}]_2, [\text{st}_{\tilde{\mathbf{c}}}]_2$ ▷ States for all shake evaluations

 - 4: $\text{SampleInBallC}([\tilde{\mathbf{c}}]_2, [\text{st}_{\mathbf{c}}]_2, [\mathbf{c}]_q)$
 - 5: $[\mathbf{z}]_q \leftarrow \text{RecomputeSignatureC}([\tilde{\mathbf{z}}]_q)$ ▷ Length check and combining
 - 6: $[\mathbf{w}_{\text{appr}}]_q \leftarrow \text{WApproxC}(\hat{\mathbf{A}}, [\mathbf{z}]_q, [\mathbf{c}]_q, t_1)$
 - 7: $[\mathbf{w}_1]_2 \leftarrow \text{UsingHintC}([\mathbf{h}]_q, [\tilde{\mathbf{w}}_0]_q, [\mathbf{w}_{\text{appr}}]_q)$
 - 8: $[\text{com}]_2 \leftarrow H(\text{msg}, \pi_1) \oplus [\mathbf{r}]_2$
 - 9: $[\boldsymbol{\mu}]_2 \leftarrow \text{ShakeC}(\mathbf{tr} \| 0 \| 0 \| [\text{com}]_2)$ ▷ One absorb
 - 10: **assert** $[\tilde{\mathbf{c}}]_2 = \text{ShakeC}([\boldsymbol{\mu}]_2 \| [\mathbf{w}_1]_2)$ ▷ Seven absorbs
-

WApproxC recomputes $\mathbf{w}_{\text{appr}} = \text{NTT}^{-1}(\hat{\mathbf{A}} \circ \text{NTT}(\mathbf{z}) - \text{NTT}(\mathbf{c}) \circ \text{NTT}(t_1 \cdot 2^{13}))$ from $[\mathbf{z}]_q, [\mathbf{c}]_q$ and the public $\hat{\mathbf{A}}$ and t_1 , and every step of it is linear in the commitments. Since $\hat{\mathbf{A}}$ is fixed, we load it already in number theoretic transform (NTT) form, so only $\mathbf{z}, \mathbf{c}$ and t_1 pass through a forward transform and only the combined result through an inverse one. The NTT is linear and can directly be applied to the commitments without any degree increase. The remaining products $\hat{\mathbf{A}}_{i,j} \circ [\tilde{\mathbf{z}}_j]_q$ and $[\tilde{\mathbf{c}}]_q \circ \hat{t}_{1,i}$ are each a pair of a committed value with a public one, so the degree is preserved.³

4.1 Proving SHAKE256

A circuit for proving SHAKE256 was already designed in [4], and we use it unchanged, re-tuning only its two parameters (F, B): how many of Keccak’s 24

³ Section 5.1 modifies this part, where $\hat{\mathbf{A}}$ and t_1 are no longer public constants.

rounds are run forward and backward between committed state boundaries. We re-tune because [4] optimized for size and compared only $(F, B) \in \{(1, 0), (4, 2)\}$ for inputs short enough to fit a single absorb. Our longest **SHAKE256** call in our circuit absorbs 7 blocks (Algorithm 4) and the size is dominated by the $\mathbb{F}_q$ side, so runtime savings are important here. Benchmarking that 7-absorb call on its own across every plan dividing 24 (Table 6), the extremes are (1, 0) at 210 ms to prove and 556 KB, and (4, 2) at 454 ms and 109 KB. We fix (3, 0), at 172 ms and 198 KB, as it is the best balance; more details and the full circuit in Section E.

4.2 The circuit for **SampleInBall**

Proving **SampleInBall** (Algorithm 2) is hard, because the algorithm is data-dependent in two ways: each iteration swaps two coefficients at positions derived from the secret stream, and the loop stops after a secret number of iterations. Both the positions and the stopping point have to stay hidden, so the circuit must do the same work in every round independent of the witness.

We first express Algorithm 2 as a Fisher–Yates shuffle, which sets the last ν coefficients to -1 or 1 from the first 64 stream bits and lets the rest of the stream perform the shuffle. We run the circuit in $\mathbb{F}_2$ to spend as few $\mathbb{F}_q$ commitments as possible, so we never carry $\mathbf{c}$ itself. Instead, we keep two indicator vectors $\mathbf{c}_\pm$ recording whether a position holds $+1$ or -1 , and convert them to $\mathbb{F}_q$ at the end. **SampleInBallC** (Algorithm 5) has three steps: one **SHAKE256** call producing the stream, one shuffle over $\mathbb{F}_2$, and one batch of conversions. The conversion packs each challenge coefficient into 2 bits, so one conversion covers 11 coefficients. Packing this way costs $\lceil N/11 \rceil = 24$ conversions of 22 bits.

The circuit also needs a fixed round budget $\theta_{\max}$, since it has to simulate work in every iteration and can never terminate early. Choosing a small round budget does not cost soundness, but it might leave an honest user unable to prove its signature. At $\theta_{\max} = 128$ the $8 + \theta_{\max} = 136$ byte stream is exactly one **SHAKE256** rate block, the largest budget a single squeeze affords. An honest signature exceeds that budget only if at least 90 of its first 128 byte draws are rejected, which happens with probability at most $\binom{128}{90} (38/256)^{90} < 2^{-139}$. Accounting for the per-position rejection probabilities rather than the worst case tightens this. At $\theta_{\max} = 213$ the stream is the 221 bytes that [37, Table 3] names as the smallest limit an implementation may impose, so an honest signature outruns the budget only with probability about 2^{-256} , and the extra 85 bytes cost a second squeeze (so one more **SHAKE256** permutation).

The stream that is input into the Fisher–Yates shuffle is the output of the **SHAKE256** circuit, so it arrives as degree-1 commitments over $\mathbb{F}_2$. Each round swaps two entries at secret positions, which is an ORAM access. A *general* ORAM pays only for the entries a round actually touches, at the price of a small statistical loss. The obvious alternative is a linear ORAM, which scans the whole array on every access and so needs nothing beyond **VOLEITH** and the **QuickSilver** framework, but commits $\Theta(N)$ values per round whatever the round does. Section F.1 gives that construction, together with the plain shuffle (Algo-

Algorithm 5 SampleInBallC($[\tilde{c}]_2, [st_c]_2, [c]_q$)

Require: Conversions pack 11 coefficients, i.e. 22 bits each

```

1:  $h \leftarrow \text{ShakeC}([\tilde{c}]_2, [st_c]_2)$  ▷ One absorb, one squeeze
2:  $([c_+]_2, [c_-]_2) \leftarrow \text{SampleInBallShuffleGORAM}(h, \theta_{\max}, \nu)$ 
3: for  $x = 0$  to  $N - 1$  do
4:   assert  $[c_x]_q \cdot ([c_x]_q - 1) \cdot ([c_x]_q + 1) = 0$  ▷  $c_x \in \{-1, 0, 1\}$ 
5:   for  $block = 0$  to  $N - 1$  step 11 do
6:      $end \leftarrow \min(block + 11, N)$ 
7:      $\beta \leftarrow ([c_{+,x}]_2 + [c_{-,x}]_2 + 1 \parallel [c_{+,x}]_2)_{x \in [block, end)}$  ▷ Two bits encoding  $c_x + 1$ 
8:      $[v]_q \leftarrow \text{ConvertB2A}(\beta, 2(end - block))$ 
9:     assert  $[v]_q = \sum_{x=block}^{end-1} ([c_x]_q + 1) \cdot 4^{x-block}$ 
10: return  $[c]_q$ 

```

rithm 13) whose swap it realizes directly, and Table 3 prices the two against each other. The per-round control below is common to both ORAM approaches.

The Round Condition and the Counter. A round is active iff $j \leq i \wedge i \neq 0$; otherwise no swap happens. The comparison $j \leq i$ is done bit-wise: for each position $x \in \{0, \dots, 7\}$ we form a check for $j[x] < i[x]$, of degree 2, and a linear check for $i[x] = j[x]$. They are combined as

$$j[0] < i[0] \vee (i[0] = j[0] \wedge (j[1] < i[1] \vee (\dots)))$$

reaching degree at most 9. The $i \neq 0$ is an equality indicator against a constant.

Lemma 6 (Equality Indicator). *Fix a public constant $y \in \{0, \dots, 255\}$ with bit decomposition $y_1, \dots, y_8$, and an index $[x]_2 = ([x_1]_2, \dots, [x_8]_2)$. Then,*

$$eq_{xy} := \prod_{k=1}^8 (1 + x_k + y_k)$$

evaluates to 1 if $x = y$ and to 0 otherwise.

Proof. Over $\mathbb{F}_2$, $1 + x_k + y_k = 1$ iff $x_k = y_k$ and 0 otherwise. A product of bits is 1 exactly when every factor is 1, i.e., when $x_k = y_k$ for every k . $\square$

Multiplying the comparison by $1 \oplus eq_{i0}$ brings the result to degree 17, and it defines the per-round bit *do* (degree-reduced), which is 1 exactly when the round is active. The counter advances only on an active round, so rather than branching we always add *do*. Each carry is one more multiplication, which would leave the bits of i at degrees 1, 2, $\dots$, 8, so we reduce each carry as it is produced: $\log_2 N - 1 = 7$ commitment degree reductions per round. In the first $\nu - 1 = 38$ rounds i cannot yet have wrapped from 255 to 0, so the carry chain is shorter and *do* is the bare comparison; only afterwards does it also need the factor $1 \oplus eq_{i0}$.

The General ORAM. A general ORAM defers correctness to a single check at the end, which is what lets a round pay only for the variables it reads and writes. Whenever the circuit reads an array entry, it appends the triple (address, value, timestamp of last write to address) to a list `reads`, the prover supplying value and timestamp as witness; whenever it writes, it appends (address, new value, current timestamp) to a list `writes`. At the beginning `writes` holds all N entries at timestamp 0, and after the loop every entry is read once more in a teardown read. Each round accesses the two positions i and j , so the lists have

$$n_{\text{acc}} = |\text{reads}| = |\text{writes}| = N + 2\theta_{\text{max}}$$

entries, 512 at $\theta_{\text{max}} = 128$ and 682 at $\theta_{\text{max}} = 213$. A permutation check on the two lists guarantees soundness (Lemmas 7 and 8). Algorithm 6 gives the circuit.

Algorithm 6 SampleInBallShuffleGORAM($h, \theta_{\text{max}}, \nu$)

```

1:  $\mathbf{c}_{\pm} \leftarrow 0^{256}$ ,  $\mathbf{c}_{-}[N - \nu..] \leftarrow h[0..\nu]$ ,  $\mathbf{c}_{+}[N - \nu..] \leftarrow 1^{\nu} \oplus h[0..\nu]$ 
2: append  $(x, \mathbf{c}_{\pm}[x], 0)$  to writes ▷ every position  $x$ , written at time 0
3:  $i \leftarrow \text{BytesToBits}(N - \nu)$ 
4: for  $\text{rnd} = 0$  to  $\theta_{\text{max}} - 1$  do
5:    $j \leftarrow h[64 + \text{rnd} \cdot 8 .. 64 + (\text{rnd} + 1) \cdot 8]$  ▷ sign bits always the first 64 bits
6:    $do \leftarrow i \neq 0 \wedge j \leq i$ 
7:    $(\mathbf{c}_{\pm}[i], t_i), (\mathbf{c}_{\pm}[j], t_j) \leftarrow$  prover's claimed last write to  $i$  and  $j$ 
8:   assert  $t_i < 2\text{rnd} + 1 \wedge t_j < 2\text{rnd} + 2$ 
9:   append  $(i, \mathbf{c}_{\pm}[i], t_i)$  and  $(j, \mathbf{c}_{\pm}[j], t_j)$  to reads
10:   $\mathbf{c}'_{\pm}[i] \leftarrow \mathbf{c}_{\pm}[i] + do \cdot (\mathbf{c}_{\pm}[i] + \mathbf{c}_{\pm}[j])$ 
11:   $\mathbf{c}'_{\pm}[j] \leftarrow \mathbf{c}_{\pm}[j] + do \cdot (\mathbf{c}_{\pm}[j] + \mathbf{c}_{\pm}[i])$ 
12:  append  $(i, \mathbf{c}'_{\pm}[i], 2\text{rnd} + 1)$  and  $(j, \mathbf{c}'_{\pm}[j], 2\text{rnd} + 2)$  to writes
13:   $i \leftarrow i + do$  ▷ bit-addition with carry-over and overflow
14: assert  $i = 0$  ▷ enough shuffles were performed
15: for  $x = 0$  to  $N - 1$  do
16:    $(\mathbf{c}_{\pm}[x], t_x) \leftarrow$  final write to  $x$ 
17:   append  $(x, \mathbf{c}_{\pm}[x], t_x)$  to reads
18: permutation-check: reads  $\sim$  writes
19: return  $\mathbf{c}_{+} - \mathbf{c}_{-}$  ▷  $\mathbf{c}_{\pm}$  embedded as  $\mathbb{F}_q^{256}$  vectors

```

The computation of do and the conditional increment of i are as before. An address is $\log_2 N = 8$ bits and a value is the two bits $\mathbf{c}_{\pm}$. Timestamps run up to $2\theta_{\text{max}}$ and need $\lceil \log_2(2\theta_{\text{max}} + 1) \rceil$ bits, which is 9 for both budgets we report. A tuple is therefore at most $10 + \lceil \log_2(2\theta_{\text{max}} + 1) \rceil \leq 19$ bits.

Permutation Check. We use the check from [48]. We pack a tuple $v = (v_1, \dots, v_m)$ into a single field element by reading its bits as coordinates in the polynomial basis $1, \alpha, \dots, \alpha^{127}$ of $\mathbb{F}_{2^{128}}$,

$$\text{lift}(v) := \sum_{k=1}^m v_k \cdot \alpha^{k-1} \in \mathbb{F}_{2^{128}},$$

which is $\mathbb{F}_2$ -linear and therefore applies to the underlying VOLEitH commitments, so lift can be computed on committed tuples directly. Since $m < 128$, lift is injective on the tuples that occur. Given a challenge $\chi \leftarrow \mathbb{F}_{2^{128}}$, drawn after every entry of `reads` and `writes` has been committed, the prover asserts

$$\prod_{j \in [n_{\text{acc}}]} (\chi - \text{lift}(\text{reads}[j])) = \prod_{j \in [n_{\text{acc}}]} (\chi - \text{lift}(\text{writes}[j])).$$

Lemma 7 (Soundness of the Permutation Check). *If reads and writes are not permutations of each other (as tuples), the permutation check passes for a uniform $\chi \in \mathbb{F}_{2^{128}}$ with probability at most $n_{\text{acc}}/2^{128}$.*

More details can be found in Section F.2. One challenge gives a loss of $n_{\text{acc}}/2^{128} \leq 2^{-118}$ for either budget. Repeating for t independent challenges $\chi_1, \dots, \chi_t$ yields $(n_{\text{acc}}/2^{128})^t$, costing one more product run and its degree reductions.

From Permutation to Consistency. Lemma 7 only states that every claimed read matches *some* recorded write to the same address, not that it returns the *most recent* one. Our construction can be identified as an instance of the read/write RAM formalism of [48, Game 4], proving the consistency (see Section F.2).

Lemma 8 (Consistency, via [48, Game 4, Lemma 6]). *For every address a , the lists of reads and writes to it equals entry for entry and in order: the m -th write to a is read exactly once, by the read accompanying the $(m+1)$ -th write to a or by the teardown read.*

Our timestamp bound is an exact $\mathbb{F}_2$ identity rather than the separate set-membership gadget [48]’s protocol uses, so it contributes no probabilistic error.

Degree Reductions. The products of the permutation check are values in $\mathbb{F}_{2^{128}}$, so each degree reduction costs a whole field element rather than a bit. We let each product run accumulate to degree d_{max} before reducing and use $\lfloor (n_{\text{acc}} - 2)/(d_{\text{max}} - 1) \rfloor$ commitments per product. Inside the loop we still reduce do and the two selectors $do \cdot (c_{\pm}[i] + c_{\pm}[j])$, so the ORAM only ever records degree-1.

Cost. Each round carries do , a carry chain, 8 bits defining the next j , the two selectors above and two read tuples, each 2 value bits and a timestamp of $\lceil \log_2(2\theta_{\text{max}} + 1) \rceil = 9$ bits. That is 40 $\mathbb{F}_2$ -commitments per round in place of the linear ORAM’s 528, and $N \cdot 11$ for the final teardown pass. edaBitE fixed $d_{\text{max}} = 197$, so we need 12 degree reductions. Table 3 compares the ORAMs.

4.3 Range Checks and Digit Decomposition

The ML-DSA verification algorithm enforces two bounds, so our circuit has to on committed values. Verification accepts only if $\|z\|_{\infty} \leq \gamma_z$ for $\gamma_z := \gamma_1 - \beta - 1$, which `RecomputeSignatureC` (Algorithm 7) enforces while handing $[z]_q$ to the rest

Table 3: Standalone cost of `SampleInBallShuffle` under both ORAM constructions. The linear ORAM (Section F.1) scans the whole array every round; *Short* and *Fast* differ in whether they reduce the $2N$ equality indicators. The general ORAM runs $t = 2$ repetitions of the permutation check at $d_{\max} = 197$.

ORAM	$\theta_{\max}$	Prove Time (ms)	Verify Time (ms)	Size (KB)
Linear (Short)	128	353.08	54.84	139.379
	213	610.47	85.91	229.139
Linear (Fast)	128	127.91	79.61	270.396
	213	213.36	126.61	447.196
General	128	81.06	56.28	29.347
	213	85.97	61.74	36.367

of the circuit. Recovering $\mathbf{w}_1$ from the hint means splitting $\mathbf{w}_{\text{appr}}$ into a high and a low part with the low part bounded, which `UsingHintC` (Algorithm 14) does coordinate-wise. We shift each check such that each is a membership test for $[0, C]$ with a public constant C , and both are 18 bits wide. The two checks are the same but with different constants, so we describe them together. They are responsible for more than half the total cost, so we evaluate different trade-offs.

Algorithm 7 `RecomputeSignatureC` $(([\tilde{z}_j]_q)_{j \in [N \cdot L]})$

```

1: for  $j = 1$  to  $N \cdot L$  do
2:    $([x_1]_q, \dots, [x_\ell]_q) \leftarrow [\tilde{z}_j]_q$  ▷ base- $B$  digits of  $z_j + \gamma_z$ 
3:    $acc \leftarrow 1$ 
4:   for  $k = \ell$  down to 1 do
5:     assert  $\prod_{t=0}^{B-1} ([x_k]_q - t) = 0$  ▷  $x_k$  is a valid base- $B$  digit
6:      $acc \leftarrow eq_{c_k}([x_k]_q) \cdot acc + less([x_k]_q, c_k)$  ▷  $c_k$ :  $k$ -th digit of  $C = 2\gamma_z$ 
7:   assert  $acc = 1$  ▷  $z_j + \gamma_z \leq C$ , i.e.  $\|z_j\|_\infty \leq \gamma_z$ 
8:    $[z_j]_q \leftarrow \sum_{k=1}^{\ell} [x_k]_q \cdot B^{\ell-k} - \gamma_z$  ▷ Recombine and then reverse shift
9: return  $([z_j]_q)_{j \in [N \cdot L]}$ 

```

Bits or Digits. Committed as 18 $\mathbb{F}_2$ bits, a range check is almost free: a wire is a bit by construction, so the domain needs no enforcing, and the $<$ comparison against bits of C is schoolbook, walking from the most significant bit down and multiplying in one linear factor per step. Obtaining the $\mathbb{F}_q$ value then costs one conversion per coordinate, and the whole cost moves to the `edaBitE` check.

Committed instead as ℓ base- B digits in $\mathbb{F}_q$ with $B^\ell \geq 2^{18}$, no conversion is needed, but: (1) a committed $\mathbb{F}_q$ wire is not confined to $\{0, \dots, B-1\}$, so the circuit checks a vanishing product, and (2) the equality indicators are Lagrange basis polynomials on that domain (Section G.1). Each digit then multiplies in a degree- $(B-1)$ factor, so the comparison reaches degree $\ell(B-1)$. A $<$ check against digits of C is then a summation of those equality indicators.

Raising B trades the number of commitments with degree: an $\mathbb{F}_q$ element costs $\tau \lceil \log_2 q \rceil / 8 = 46$ bytes to generate against 2 bytes for a bit commitment, so fewer digits per coordinate, but each degree costs an $\mathbb{F}_{q^6}$ mask and coefficient. Section G compares all eight variants (6 different digits and pure conversions Mystique and edaBitE). We consider $B = 8$ the best trade-off.

Computing w_0 . Bounding w_0 needs slightly more care, but still a decomposition and works essentially the same way. The application of the hint based on the length-bounded selectors is added, but the technical details are neither interesting nor impactful enough, so we defer the details to Section G.2.

Packing the Conversion. Each $w_{1,i}$ is only 6 bits wide, so rather than convert them one at a time we pack three consecutive coordinates into one $\mathbb{F}_q$ element, $v = \sum_{i < 3} w_{1,i} \cdot 2^{6i}$, and run a single 18-bit conversion on v , 342 in total.

4.4 Benchmarks and Sizes

The preceding subsections benchmarked the computation and communication cost of each component on its own. We now combine them for the whole verification (Table 4) fixing general ORAM with $\theta_{\max} = 213$, $B = 8$ for both range checks, and the (3,0) SHAKE256 verification circuit.

The digits the two range checks commit amount to 600 KB together, the 10 Keccak permutations account for a little over 250 KB, and the conversions under edaBitE for a little under 100 KB, and finally the **SampleInBall** shuffle for around 20 KB. The rest are degree masks and fixed VOLEitH setup costs.

The circuit crosses between the two fields exactly twice, to turn $\mathbf{c}_{\pm}$ into $\mathbf{c}$ and w_1 back into bits (Algorithm 4). These are 366 conversions in total, 24 for the challenge and 342 for w_1 . Using the state-of-the-art conversion check from Mystique they are the single largest item in the whole proof, ahead of both the range check and SHAKE256, thus benefiting strongly from edaBitE.

Table 4: Comparison of the full construction under general ORAM with $\theta_{\max} = 213$, $B = 8$, and the (3,0) SHAKE256 plan, across both edaBit variants.

Conversion technique	Runtime in ms			Proof Size (KB)
	Sign ₁	Sign ₃	Verify	
Mystique	150.84	1422.70	347.89	1486.979
edaBitE	197.41	815.29	504.04	1011.909

4.5 Security

Our construction follows [22], instantiated with ML-DSA as the signature scheme, the VOLEitH NIZK, and using the commitment $\text{com} = H(\text{msg}, \pi_1) \oplus \mathbf{r}$ as in [4]. Therefore, security follows directly as in [4].

Theorem 2. *Let ML-DSA be EUF-CMA secure, let H be modeled as a random oracle, and let $VOLeith$ be zero-knowledge and straightline-extractable knowledge-sound with extraction error ε . Then Algorithm 11 satisfies Definitions 1, 5 and 6 for $\mathcal{I} = (\text{pk})$: blindness reduces to zero-knowledge, and one-more unforgeability reduces to the EUF-CMA security of ML-DSA under Q signing queries, with an additive loss of $(Q + 1)\varepsilon$ against an adversary returning $Q + 1$ forgeries.*

Correctness and blindness follow from the argument system’s completeness and zero knowledge, with $\mathbf{r}$ acting as a one-time pad on com , and unforgeability forwards every Sign_2 query to the EUF-CMA oracle and extracts a signature from each of the ℓ proofs, one of which was never queried. The one place a random oracle is used is in showing that the com_i values are pairwise distinct. More detail can be found in Section H and [4].

The oracle is H , which the verifier evaluates in the clear and the circuit takes as a public input (Algorithm 4), so it is free to be any hash and constrains none of the SHAKE256 calls the circuit does prove. Our implementation instantiates it with SHAKE256 as well, but outside the circuit and at no cost.

The concrete soundness has to be considered, and by a union bound it is

$$\underbrace{(2/q)^6 \approx 2^{-132}}_{\text{edaBitE (Theorem 1)}} + \underbrace{(n_{\text{acc}}/2^{128})^2 \approx 2^{-238}}_{\text{ORAM permutation check (Lemma 7)}} + \underbrace{197/2^{128}}_{\text{soundness of NIZK}}$$

We added additional grinding step to boost the soundness, and retrieve the 8 bits lost due to $d_{\text{max}} = 197$ when choosing parameters for our implementation.

5 Issuer-Hiding & Attribute Disclosure

We modify Algorithm 4 to achieve *issuer-hiding* and the selective disclosure of *attributes*. Neither extension relies on anything specific to ML-DSA, and all the techniques we discuss here can be applied one-to-one to e.g. PoMFRIT [4]. The circuit below replaces a single public key by a public table, and the correct row for the execution of the remaining circuit is then chosen via a one-hot-vector. The attribute assignment itself is done via the issuer signing the commitment together with auxiliary data aux corresponding to the credentials of the user.

5.1 Obtaining Issuer-Hiding

The circuit of Section 4 consumes the issuer’s key as public constants: WApproxC reads $\hat{\mathbf{A}}$ and $\mathbf{t}_1$, and the $\boldsymbol{\mu}$ is the output of a SHAKE256 call absorbing $\mathbf{tr}$. IssuerHidingC (Algorithm 15) derives those from a public table of n issuer keys, leaving the remaining arithmetic unchanged. The prover commits to the issuer’s index as $\lceil \log n \rceil$ bits over $\mathbb{F}_q$ and, through one conversion, over $\mathbb{F}_2$; a doubling gadget expands them into the one-hot vector $\mathbf{f}$ in each field, and the selected row is $\sum_j \mathbf{f}[j] \cdot (\mathbf{tr}_j, \hat{\mathbf{A}}_j, \mathbf{t}_{1,j})$. When n is not a power of two the padding entries are dropped and $\sum_j \mathbf{f}_q[j] = 1$ is asserted, confining the index to $[0, n)$. An OR-proof over the n candidate statements [20] would also achieve issuer-hiding, but runs the circuit n times and proves that one execution was correct. We run it once, and only pay in that the public parameters are no longer constants.

The Circuit. Write $C_{\mathcal{I},\text{msg}}$ for the result. It differs from $C_{\text{pk},\text{msg}}$ in three places: the key material becomes a public table, and the one-hot-vector selection as described above is prepended to select one row of it, and $\mathbf{w}_{\text{appr}}$ is degree-reduced before `UsingHintC`, which the non-hiding path avoids only because its $\hat{\mathbf{A}}$ and $\mathbf{t}_1$ are constants. That reduction re-commits a value and asserts the two agree, so it constrains nothing new: $C_{\mathcal{I},\text{msg}}$ accepts exactly those $(\beta, \mathbf{r}, \sigma)$ for which σ is a valid ML-DSA signature on `com` under the row β selects. The fully spelled-out selection gadget `IssuerHidingC` can be found in Section I.1.

Security. Neither the commitment nor the message flow changes, so only the two multi-issuer properties need an argument of their own.

Theorem 3 (Security with Issuer-Hiding). *Let ML-DSA be EUF-CMA secure, let $\mathbf{H}$ be modeled as a random oracle, and let VOLEitH be zero-knowledge and straightline-extractable knowledge-sound with extraction error ε . Then Algorithm 11, instantiated with the circuit $C_{\mathcal{I},\text{msg}}$, satisfies Definitions 1 and 5 to 7 for a list of n issuers. One-more unforgeability loses a factor n against Theorem 2 and reduces to EUF-CMA under $\max_j Q_j$ signing queries. The issuer-hiding advantage is at most twice the zero-knowledge advantage of VOLEitH .*

The full argument is deferred to Section I.2. Correctness and blindness are inherited unchanged. For unforgeability, extraction recovers the committed index bits along with the signature, so each forgery names one honest row of the table. A reduction plants its EUF-CMA challenge key in a guessed row of the sub-list $\mathcal{J}$ the forgeries name, and generates the other $n - 1$ key pairs itself, which is where the factor n comes from. Only one issuer’s queries require the signing oracle: $\max_j Q_j$ bounds the number of queries. For issuer-hiding, $C_{\mathcal{I},\text{msg}}$ is fixed by $\mathcal{I}$ and `msg` alone, with the index entering only as a witness, so both values of b prove the same statement and the zero-knowledge property ensures issuer-hiding.

Cost. $C_{\mathcal{I},\text{msg}}$ adds one reduction over the $N \cdot K$ coordinates of $\mathbf{w}_{\text{appr}}$ in $\mathbb{F}_q$, ≈ 47 KB. What is left of the dependence is the $\lceil \log n \rceil$ committed index bits. The indicator entries themselves keep whatever degree the doubling gave them, and $\mathbf{w}_{\text{appr}}$ is reduced regardless. Prover and verifier time grow with n , since both evaluate the n products for each selected value, but the rest of the circuit keeps the cost it had. For better runtime, one could reduce the n one-hot vectors in both fields and get a linear size scaling in the number of issuers. We report exact measurements in Table 5 together with those of the next subsection.

5.2 Attribute Disclosure

A credential is rarely just a signature on a message: the issuer knows who the user is, and the verifier wants to learn a predicate over the user’s identity, e.g., their age. We therefore let the issuer sign a block of attributes `aux` alongside `com`, so that the message representative becomes $\mu = \text{SHAKE256}(\text{tr} \parallel 0 \parallel 0 \parallel [\text{com}]_2 \parallel [\text{aux}]_2)$.

The attributes enter as $\mathbb{F}_2$ commitments and are never opened, and neither are the values derived from them, so the verifier only learns which predicates

hold and not `aux`. Low-degree predicates, do not add anything to the proof size beyond the attribute bits themselves. We provide a concrete example in Section 1.3 of which we report the measurements in Table 5. The only change in costs are: the attribute block pushes the μ absorb past one SHAKE256 block, and that second permutation costs 25.6 KB using the (3, 0)-configuration for the SHAKE256 circuit, while the attribute bits require an additional 1.8 KB.

Table 5: Prove/verify time and proof size across the number of candidate issuers n , under general ORAM with $\theta_{\max} = 213$, $B = 8$, edaBitE for the conversions, and the (3, 0) SHAKE256 circuit. Each cell reads *issuer-hiding* / *credential*, the latter adding attribute disclosure under the “at least 18” policy.

total issuers N	Runtime in ms			Proof Size (KB)
	Sign ₁	Sign ₃	Verify	
27	202.28/209.99	873.74/874.76	522.96/519.41	1059.253/1086.629
128	198.93/202.56	942.56/964.68	561.52/569.10	1059.350/1086.726
512	201.72/204.74	1306.00/1311.30	760.25/766.39	1059.445/1086.821

Acknowledgments. We thank Lucjan Hanzlik for a helpful discussion and Elias Nørholm for implementing the $\mathbb{F}_q$ field and its field extension $\mathbb{F}_{q^6}$. MB acknowledges support from the Danish Ministry of Defense Acquisition and Logistics Organization (FMI). CB was supported by research grant VIL53029 from VILLUM FONDEN and the ACPA project supported by the Independent Research Fund Denmark (DRF). We used Claude to support writing and parts of the coding (more details in Section A); all technical content and proofs are the authors’ own.

References

1. Agarwal, A., Baum, C., Braun, L., Scholl, P.: Low-bandwidth mixed arithmetic in VOLE-based ZK from low-degree PRGs. In: Fehr, S., Fouque, P.A. (eds.) EU-ROCRYPT 2025, Part IV. LNCS, vol. 15604, pp. 396–426. Springer, Cham (May 2025). https://doi.org/10.1007/978-3-031-91134-7_14
2. Agrawal, S., Kirshanova, E., Stehlé, D., Yadav, A.: Practical, round-optimal lattice-based blind signatures. In: Yin, H., Stavrou, A., Cremers, C., Shi, E. (eds.) ACM CCS 2022. pp. 39–53. ACM Press (Nov 2022). <https://doi.org/10.1145/3548606.3560650>
3. Argo, S., Güneysu, T., Jeudy, C., Land, G., Roux-Langlois, A., Sanders, O.: Practical post-quantum signatures for privacy. In: Luo, B., Liao, X., Xu, J., Kirda, E., Lie, D. (eds.) ACM CCS 2024. pp. 1523–1537. ACM Press (Oct 2024). <https://doi.org/10.1145/3658644.3670297>
4. Baum, C., Beckmann, M., Beullens, W., Mukherjee, S., Rechberger, C.: Concretely efficient blind signatures based on {VOLE-in-the-Head} proofs and the {MAYO} trapdoor. In: 35th USENIX Security Symposium (USENIX Security 26). pp. 6325–6344 (2026)
5. Baum, C., Beullens, W., Braun, L., Delpech de Saint Guilhem, C., Kloß, M., Majenz, C., Mukherjee, S., Orsini, E., Ramacher, S., Rechberger, C., Roy, L., Scholl, P.: FAEST: Algorithm specifications, version 3.0. Tech. rep., faest.info (2026), <https://faest.info/faest-spec-v3.0.pdf>
6. Baum, C., Beullens, W., Mukherjee, S., Orsini, E., Ramacher, S., Rechberger, C., Roy, L., Scholl, P.: One tree to rule them all: Optimizing GGM trees and OWFs for post-quantum signatures. In: Chung, K.M., Sasaki, Y. (eds.) ASIACRYPT 2024, Part I. LNCS, vol. 15484, pp. 463–493. Springer, Singapore (Dec 2024). https://doi.org/10.1007/978-981-96-0875-1_15
7. Baum, C., Braun, L., Delpech de Saint Guilhem, C., Kloß, 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.) CRYPTO 2023, Part V. LNCS, vol. 14085, pp. 581–615. Springer, Cham (Aug 2023). https://doi.org/10.1007/978-3-031-38554-4_19
8. Baum, C., Braun, L., Munch-Hansen, A., Razet, B., Scholl, P.: Appenzeller to brief: Efficient zero-knowledge proofs for mixed-mode arithmetic and Z2k. In: Vigna, G., Shi, E. (eds.) ACM CCS 2021. pp. 192–211. ACM Press (Nov 2021). <https://doi.org/10.1145/3460120.3484812>
9. Baum, C., Dittmer, S., Scholl, P., Wang, X.: Sok: vector OLE-based zero-knowledge protocols. DCC **91**(11), 3527–3561 (2023). <https://doi.org/10.1007/s10623-023-01292-8>
10. Bender, A., Katz, J., Morselli, R.: Ring signatures: Stronger definitions, and constructions without random oracles. In: Halevi, S., Rabin, T. (eds.) TCC 2006. LNCS, vol. 3876, pp. 60–79. Springer, Berlin, Heidelberg (Mar 2006). https://doi.org/10.1007/11681878_4
11. Beullens, W.: MAYO: Practical post-quantum signatures from oil-and-vinegar maps. In: AlTawy, R., Hülsing, A. (eds.) SAC 2021. LNCS, vol. 13203, pp. 355–376. Springer, Cham (Sep / Oct 2022). https://doi.org/10.1007/978-3-030-99277-4_17
12. Beullens, W., Lyubashevsky, V., Nguyen, N.K., Seiler, G.: Lattice-based blind signatures: Short, efficient, and round-optimal. In: Meng, W., Jensen, C.D., Cremers, C., Kirda, E. (eds.) ACM CCS 2023. pp. 16–29. ACM Press (Nov 2023). <https://doi.org/10.1145/3576915.3616613>

13. Blazy, O., Mallordy, L.B., Wen, W.: Concurrently secure compact blind signatures from module-sis. Cryptology ePrint Archive (2026)
14. Bobolz, J., Eidens, F., Krenn, S., Ramacher, S., Samelin, K.: Issuer-hiding attribute-based credentials. In: Conti, M., Stevens, M., Krenn, S. (eds.) CANS 21. LNCS, vol. 13099, pp. 158–178. Springer, Cham (Dec 2021). https://doi.org/10.1007/978-3-030-92548-2_9
15. Bootle, J., Lyubashevsky, V., Nguyen, N.K., Sorniotti, A.: A framework for practical anonymous credentials from lattices. In: Handschuh, H., Lysyanskaya, A. (eds.) CRYPTO 2023, Part II. LNCS, vol. 14082, pp. 384–417. Springer, Cham (Aug 2023). https://doi.org/10.1007/978-3-031-38545-2_13
16. Bouillaguet, C., Feneuil, T., Maire, J., Rivain, M., Sauvage, J., Vergnaud, D.: Blinding post-quantum hash-and-sign signatures. In: 2026 IEEE Symposium on Security and Privacy. pp. 1–18. IEEE Computer Society Press (May 2026). <https://doi.org/10.1109/SP63933.2026.00032>
17. Camenisch, J., Lysyanskaya, A.: An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 93–118. Springer, Berlin, Heidelberg (May 2001). https://doi.org/10.1007/3-540-44987-6_7
18. Camenisch, J., Lysyanskaya, A.: A signature scheme with efficient protocols. In: Ciamato, S., Galdi, C., Persiano, G. (eds.) SCN 02. LNCS, vol. 2576, pp. 268–289. Springer, Berlin, Heidelberg (Sep 2003). https://doi.org/10.1007/3-540-36413-7_20
19. Chaum, D.: Blind signatures for untraceable payments. In: Chaum, D., Rivest, R.L., Sherman, A.T. (eds.) CRYPTO'82. pp. 199–203. Plenum Press, New York, USA (1982). https://doi.org/10.1007/978-1-4757-0602-4_18
20. yu Chiang, J.H., Damgård, I., Duro, W.R., Engan, S., Kolby, S., Scholl, P.: Post-quantum threshold ring signature applications from VOLE-in-the-head. In: Huang, C.Y., Chen, J.C., Shieh, S.P., Lie, D., Cortier, V. (eds.) ACM CCS 2025. pp. 4664–4678. ACM Press (Oct 2025). <https://doi.org/10.1145/3719027.3744854>
21. Escudero, D., Ghosh, S., Keller, M., Rachuri, R., Scholl, P.: Improved primitives for MPC over mixed arithmetic-binary circuits. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part II. LNCS, vol. 12171, pp. 823–852. Springer, Cham (Aug 2020). https://doi.org/10.1007/978-3-030-56880-1_29
22. Fischlin, M.: Round-optimal composable blind signatures in the common reference string model. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 60–77. Springer, Berlin, Heidelberg (Aug 2006). https://doi.org/10.1007/11818175_4
23. Frigo, M., shelat, a.: Anonymous credentials from ECDSA. Cryptology ePrint Archive, Report 2024/2010 (2024), <https://eprint.iacr.org/2024/2010>
24. Galois, Inc.: swanky: A suite of rust libraries for secure computation. <https://github.com/GaloisInc/swanky> (2019)
25. Hanzlik, L., Lai, Y.F., Mula, M., Paracucchi, E., Slamanig, D., Tang, G.: Tanuki: New frameworks for (concurrently secure) blind signatures from post-quantum group actions. In: Hanaoka, G., Yang, B.Y. (eds.) ASIACRYPT 2025, Part IV. LNCS, vol. 16248, pp. 35–69. Springer, Singapore (Dec 2025). https://doi.org/10.1007/978-981-95-5113-2_2
26. Hanzlik, L., Lai, Y.F., Paracucchi, E., Persichetti, E.: Post-quantum blind signature from standard group action assumptions and more. In: Daemen, J., Thomé, E. (eds.) EUROCRYPT 2026, Part I. LNCS, vol. 16541, pp. 277–307. Springer, Cham (May 2026). https://doi.org/10.1007/978-3-032-25291-3_10

27. Hanzlik, L., Manulis, M., Mula, M., Pulval-Dady, A., Silde, T., Slamanig, D.: Braccoon: Concurrently secure blind lattice signatures from raccoon. *Cryptology ePrint Archive* (2026)
28. Jeudy, C., Roux-Langlois, A., Sanders, O.: Lattice signature with efficient protocols, application to anonymous credentials. In: Handschuh, H., Lysyanskaya, A. (eds.) *CRYPTO 2023, Part II*. LNCS, vol. 14082, pp. 351–383. Springer, Cham (Aug 2023). https://doi.org/10.1007/978-3-031-38545-2_12
29. Jeudy, C., Sanders, O.: Improved lattice blind signatures from recycled entropy. In: Kalai, Y.T., Kamara, S.F. (eds.) *CRYPTO 2025, Part I*. LNCS, vol. 16000, pp. 477–513. Springer, Cham (Aug 2025). https://doi.org/10.1007/978-3-032-01855-7_16
30. Katsumata, S., Lai, Y.F., LeGrow, J.T., Qin, L.: CSI-otter: isogeny-based (partially) blind signatures from the class group action with a twist. *DCC* **92**(11), 3587–3643 (2024). <https://doi.org/10.1007/s10623-024-01441-7>
31. Katz, J., Sefranek, M.: Issuer hiding for BBS-based anonymous credentials. In: Bai, S., Persichetti, E. (eds.) *PKC 2026, Part III*. LNCS, vol. 16553, pp. 437–465. Springer, Cham (May 2026). https://doi.org/10.1007/978-3-032-26737-5_14
32. Klooß, M., Lai, R.W.F., Reichle, M.: Blind signatures from arguments of inequality. *Cryptology ePrint Archive, Report 2025/1886* (2025), <https://eprint.iacr.org/2025/1886>
33. Kuchta, V., LeGrow, J.T., Persichetti, E.: Post-quantum (partially) blind signatures from matrix code equivalence. *Cryptography and Communications* (2026). <https://doi.org/10.1007/s12095-026-00879-x>, scheme MEBS; based on the MEDS group action. Earlier version: *Cryptology ePrint Archive, Paper 2025/274*
34. Lai, Y.F., Persichetti, E.: LEAF: Compact and efficient blind signature from code-based assumptions. *Cryptology ePrint Archive, Report 2025/1585* (2025), <https://eprint.iacr.org/2025/1585>
35. Lyubashevsky, V., Seiler, G., Steuer, P.: The LaZer library: Lattice-based zero knowledge and succinct proofs for quantum-safe privacy. In: Luo, B., Liao, X., Xu, J., Kirda, E., Lie, D. (eds.) *ACM CCS 2024*. pp. 3125–3137. ACM Press (Oct 2024). <https://doi.org/10.1145/3658644.3690330>
36. Mir, O., Bauer, B., Griffy, S., Lysyanskaya, A., Slamanig, D.: Aggregate signatures with versatile randomization and issuer-hiding multi-authority anonymous credentials. In: Meng, W., Jensen, C.D., Cremers, C., Kirda, E. (eds.) *ACM CCS 2023*. pp. 30–44. ACM Press (Nov 2023). <https://doi.org/10.1145/3576915.3623203>
37. National Institute of Standards and Technology: Module-lattice-based digital signature standard. Tech. Rep. NIST Federal Information Processing Standards (FIPS) 204, U.S. Department of Commerce, Gaithersburg, MD (aug 2024). <https://doi.org/10.6028/NIST.FIPS.204>, <https://doi.org/10.6028/NIST.FIPS.204>
38. National Institute of Standards and Technology: Stateless hash-based digital signature standard. Tech. Rep. NIST Federal Information Processing Standards (FIPS) 205, U.S. Department of Commerce, Gaithersburg, MD (aug 2024). <https://doi.org/10.6028/NIST.FIPS.205>, <https://doi.org/10.6028/NIST.FIPS.205>
39. del Pino, R., Katsumata, S.: A new framework for more efficient round-optimal lattice-based (partially) blind signature via trapdoor sampling. In: Dodis, Y., Shrimpton, T. (eds.) *CRYPTO 2022, Part II*. LNCS, vol. 13508, pp. 306–336. Springer, Cham (Aug 2022). https://doi.org/10.1007/978-3-031-15979-4_11
40. del Pino, R., Katsumata, S., Prest, T., Rossi, M.: Raccoon: A masking-friendly signature proven in the probing model. In: Reyzin, L., Stebila, D. (eds.) *CRYPTO 2024, Part I*. LNCS, vol. 14920, pp. 409–444. Springer, Cham (Aug 2024). https://doi.org/10.1007/978-3-031-68376-3_13

41. Policharla, G.V., Westerbaan, B., Faz-Hernández, A., Wood, C.A.: Post-quantum privacy pass via post-quantum anonymous credentials. Cryptology ePrint Archive, Report 2023/414 (2023), <https://eprint.iacr.org/2023/414>
42. Roy, L.: SoftSpokenOT: Quieter OT extension from small-field silent VOLE in the minicrypt model. In: Dodis, Y., Shrimpton, T. (eds.) CRYPTO 2022, Part I. LNCS, vol. 13507, pp. 657–687. Springer, Cham (Aug 2022). https://doi.org/10.1007/978-3-031-15802-5_23
43. RustCrypto Developers: `ml-dsa`: Pure rust implementation of the module-lattice-based digital signature standard (FIPS 204). <https://github.com/RustCrypto/signatures/tree/master/ml-dsa>, version 0.1.1 (2025)
44. Sanders, O., Traoré, J.: Compact issuer-hiding authentication, application to anonymous credential. Cryptology ePrint Archive, Report 2023/1845 (2023), <https://eprint.iacr.org/2023/1845>
45. Schnorr, C.P.: Efficient identification and signatures for smart cards. In: Brassard, G. (ed.) CRYPTO'89. LNCS, vol. 435, pp. 239–252. Springer, New York (Aug 1990). https://doi.org/10.1007/0-387-34805-0_22
46. Weng, C., Yang, K., Xie, X., Katz, J., Wang, X.: Mystique: Efficient conversions for zero-knowledge proofs with applications to machine learning. In: Bailey, M., Greenstadt, R. (eds.) USENIX Security 2021. pp. 501–518. USENIX Association (Aug 2021), <https://www.usenix.org/conference/usenixsecurity21/presentation/weng>
47. Yang, K., Sarkar, P., Weng, C., Wang, X.: QuickSilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field. In: Vigna, G., Shi, E. (eds.) ACM CCS 2021. pp. 2986–3001. ACM Press (Nov 2021). <https://doi.org/10.1145/3460120.3484556>
48. Yang, Y., Heath, D.: Two shuffles make a RAM: Improved constant overhead zero knowledge RAM. In: Balzarotti, D., Xu, W. (eds.) USENIX Security 2024. USENIX Association (Aug 2024), <https://www.usenix.org/conference/usenixsecurity24/presentation/yang-yibin>

A AI Usage

We used Claude (Opus 4.8 and 5.0), through the Claude Code interface, in preparing this work. We list here the notable help. All technical content and proofs are the authors’ own, and the authors take responsibility for parts written with Claude’s assistance.

Writing. Claude helped with condensing passages where length was constrained, removing redundancy, and proofreading for spelling, grammar and punctuation. Individual passages were originally drafted with its help and then revised by us. It was also used for consistency checks across the manuscript – notation and terminology, acronyms, cross-references, and agreement between numbers quoted in different sections – and for cross-checking numerical claims against the standard, the cited specifications and our implementation.

Modifications to swanky. The proof system is our fork of `swanky` [24], and essentially all the fork-specific work lives in its `schmivitz` crate. The README of

<https://anonymous.4open.science/r/swanky-0321> lists all changes we made, and records there, as here, which parts were written with Claude’s assistance.

The largest of those were the CRT full-field multiplication and its VOLEHash check, both translated into code from the FAEST v3 specification [5]. Otherwise, it was used in a supporting capacity for implementing the edaBitE check; and carrying the existing $\mathbb{F}_2$ VOLE generation over to $\mathbb{F}_q$.

Debugging. It also helped with debugging and identifying errors within the code base, that were then reviewed by us.

B Benchmarking Setup

All numbers in this paper come from the reference implementation, measured on one machine with a single fixed build configuration. Both repositories are available anonymously: the proof system at <https://anonymous.4open.science/r/swanky-0321> and the circuits and protocol at https://anonymous.4open.science/r/blind_mldsa-800C.

Machine and toolchain. An Apple M3 Pro (12 cores, 18GiB) running macOS 26.5, with `rustc` 1.95, which both repositories pin. The proof system is our fork of `swanky`’s `schmivitz` crate [24]; the circuits and the blind-signature protocol are a separate crate that depends on it.

Build configuration. Both of the following are required to reproduce our timings, and neither is a default:

- `-C target-cpu=native -cfg aes_armv8`.
- `lto = true, codegen-units = 1, opt-level = 3`. Cargo takes profile settings from the top-level package, so the proof system does *not* inherit them from its own manifest when it is built as a dependency.

Measurement. Reported times are means using `criterion`⁴ for repeated samples: 20 for the protocol-level benchmarks of Tables 4 and 5, 10 for the standalone circuits of Tables 6 to 8, and `criterion`’s default 100 for `SampleInBall`. The full `criterion` report is included with the artifact as `benchmarks.zip`. The README gives the dictionary between the benchmark identifiers and the names used here. The proof sizes are deterministic and function of the witness and the statement, so each reported size is exact. Sig_1 is the user’s commitment, Sig_3 the proof, and Ver verification; the signer’s own work is an unmodified ML-DSA signature and is not reported as it solely depends on the implementation used for ML-DSA. The signatures we prove about are produced by the `ml-dsa` crate [43] at ML-DSA-44.

⁴ <https://github.com/bheisler/criterion.rs>

Fixed choices. The proof system runs at a degree bound of $d_{\max} = 197$ throughout, which is what Sections 3 and 4.2 refer to. Throughout, the commitment is the masked $\text{com} = H(\text{msg}, \pi_1) \oplus \mathbf{r}$ of Section 4 rather than the hashed variant of [4] (which is also implemented as blind signature, but not benchmarked fully), and the full-field masks are CRT-based.

Reproducing. Check both repositories out side by side, since the circuits depend on the proof system by relative path, and run `cargo bench` from the latter. Each benchmark writes its own table of this paper into `target/` as it finishes, so the numbers reported here are regenerated rather than transcribed. The proofs need a larger stack than the default when running test. The `criterion` reports can be regenerated using `cargo bench`. Both build flags above are committed in the repositories and need no action. The report for the runtime numbers is `criterion`’s default report.

C Additional Preliminaries

C.1 Blind Signatures

We briefly introduce the syntax we use for blind signatures, and then recall standard security properties for blind signatures. We formulate the notion as a multi-issuer blind signature, but by restricting the issuer set to 1 public key, we recover the standard formalism.

Definition 4 ((Multi-Issuer) Blind Signature). *A multi-issuer blind signature is a tuple of five algorithms defined as follows.*

- $\text{KeyGen}(1^\lambda) \rightarrow (\text{sk}, \text{pk})$: Takes the security parameter as input and outputs a secret key sk and a public key pk .
- $\text{Sign}_1(\mathcal{I}, \text{msg}) \rightarrow (\rho_1, S_{\text{User}})$: This is the first part of a signing protocol between a user User and a signer Signer , where User uses the issuer list $\mathcal{I}$ and a message $\text{msg} \in \{0, 1\}^*$ to compute a state S_{User} and a first message ρ_1 .
- $\text{Sign}_2(\text{sk}, \rho_1) \rightarrow \rho_2$: Signer uses their signing key sk and a first message ρ_1 to output a second message ρ_2 .
- $\text{Sign}_3(\rho_2, S_{\text{User}}) \rightarrow \text{sig}$: Lastly, User uses ρ_2 and S_{User} to derive a signature sig or $\perp$ if the signing protocol failed.
- $\text{Verify}(\mathcal{I}, \text{msg}, \text{sig}) \rightarrow \{0, 1\}$: A verification algorithm that takes as input an issuer list $\mathcal{I}$, a message msg and a signature sig and outputs a bit to indicate whether the signature is deemed valid (1) or invalid (0).

Definition 5 (Correctness). *A blind signature $(\text{KeyGen}, (\text{Sign}_i)_{i \in [3]}, \text{Verify})$ is correct if there exists a negligible function negl such that for any message msg , any list $\mathcal{I}$ and any (sk, pk) in the support of KeyGen with $\text{pk} \in \mathcal{I}$,*

$$\Pr \left[\text{Verify}(\mathcal{I}, \text{msg}, \text{sig}) = 1 \mid \begin{array}{l} (\rho_1, S_{\text{User}}) \leftarrow \text{Sign}_1(\mathcal{I}, \text{msg}), \\ \rho_2 \leftarrow \text{Sign}_2(\text{sk}, \rho_1), \\ \text{sig} \leftarrow \text{Sign}_3(\rho_2, S_{\text{User}}) \end{array} \right] \geq 1 - \text{negl}(\lambda).$$

Definition 6 (Blindness). A blind signature $(\text{KeyGen}, (\text{Sign}_i)_{i \in [3]}, \text{Verify})$ has blindness if for every polynomial-time three-part stateful adversary $\text{Adv} = (\text{Adv}_1, \text{Adv}_2, \text{Adv}_3)$ there exists a negligible function negl such that for any two messages $\text{msg}_0, \text{msg}_1$

$$\Pr \left[\text{Adv}_3(\text{sig}^0, \text{sig}^1) = b \mid \begin{array}{l} \mathcal{I} \leftarrow \text{Adv}_1(1^\lambda), b \leftarrow \{0, 1\}, \\ (\rho_1^0, S_{\text{User}}^0) \leftarrow \text{Sign}_1(\mathcal{I}, \text{msg}_0), \\ (\rho_1^1, S_{\text{User}}^1) \leftarrow \text{Sign}_1(\mathcal{I}, \text{msg}_1), \\ \rho_2^b, \rho_2^{1-b} \leftarrow \text{Adv}_2(\rho_1^b, \rho_1^{1-b}), \\ \text{sig}^0 \leftarrow \text{Sign}_3(\rho_2^0, S_{\text{User}}^0), \\ \text{sig}^1 \leftarrow \text{Sign}_3(\rho_2^1, S_{\text{User}}^1), \\ \text{If } \perp \in \{\text{sig}^0, \text{sig}^1\} \\ \text{then } (\text{sig}^0, \text{sig}^1) \leftarrow (\perp, \perp) \end{array} \right] - \frac{1}{2} \leq \text{negl}(\lambda).$$

Definition 7 (Issuer-hiding). A blind signature is issuer-hiding if for every polynomial-time two-part stateful adversary $\text{Adv} = (\text{Adv}_1, \text{Adv}_2)$ there exists a negligible function negl such that

$$\Pr \left[\text{Adv}_2(\text{sig}) = b \mid \begin{array}{l} (\text{pk}_j, \text{sk}_j) \leftarrow \text{KeyGen}(1^\lambda) \text{ for } j \in [n], \\ (\mathcal{I}, \text{msg}, i_0, i_1) \leftarrow \text{Adv}_1(1^\lambda, (\text{pk}_j, \text{sk}_j)_{j \in [n]}), \\ b \leftarrow \{0, 1\}, \\ (\rho_1, S_{\text{User}}) \leftarrow \text{Sign}_1(\mathcal{I}, \text{msg}), \\ \rho_2 \leftarrow \text{Sign}_2(\text{sk}_{i_b}, \rho_1), \\ \text{sig} \leftarrow \text{Sign}_3(\rho_2, S_{\text{User}}) \end{array} \right] - \frac{1}{2} \leq \text{negl}(\lambda),$$

where Adv_1 must output indices $i_0, i_1 \in [n]$ and a list $\mathcal{I}$ containing both pk_{i_0} and pk_{i_1} .

C.2 ML-DSA Verification

Algorithm 8 is ML-DSA verification as [37] states it, and Algorithms 9 and 10 are the two subroutines the circuit has to redo on committed values (Section 4.3). The `pkDecode`, `ExpandA`, `sigDecode` and `w1Encode` steps and the NTT are what Algorithm 1 drops, for the reasons given in Section 2.2.

One detail of `Decompose` is easy to miss, and Section G.2 depends on it. Its centered remainder on line 2 alone would put r_0 in $(-\gamma_2, \gamma_2]$, but the special case on lines 3–5 – taken exactly when that remainder would give $r_1 = M -$ decrements it once more. The true range is therefore the closed $[-\gamma_2, \gamma_2]$, and the endpoint $r_0 = -\gamma_2$ is reached for a single r , namely $r = q - \gamma_2$.

C.3 Non-Interactive Zero-Knowledge Proofs of Knowledge

We now define NIZKs. In our work, these are defined with respect to a random oracle RO.

Definition 8 (NIZKs in the random-oracle model). Let $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{W}$ be a relation. A transparent non-interactive zero-knowledge argument of knowledge for $\mathcal{R}$ in the random-oracle model, with oracle RO, is a pair of algorithms $(\text{Prove}^{\text{RO}}, \text{Verify}^{\text{RO}})$:

Algorithm 8 ML-DSA.Verify(pk, M', σ) [37, Algorithm 8]

Require: Public key pk, message M' $\in \{0, 1\}^*$, signature σ

```
1:  $(\rho, t_1) \leftarrow \text{pkDecode}(\text{pk})$ 
2:  $(\tilde{c}, z, h) \leftarrow \text{sigDecode}(\sigma)$   $\triangleright$  challenge seed, response, hint
3: if  $h = \perp$  then return 0  $\triangleright$  hint was not properly encoded
4:  $\hat{A} \leftarrow \text{ExpandA}(\rho)$   $\triangleright A$  in NTT form
5:  $tr \leftarrow \text{SHAKE256}(\text{pk}, 64)$   $\triangleright$  public-key digest
6:  $\mu \leftarrow \text{SHAKE256}(tr \parallel M', 64)$   $\triangleright$  message representative
7:  $c \leftarrow \text{SampleInBall}(\tilde{c})$   $\triangleright$  verifier's challenge, Algorithm 2
8:  $w_{\text{appr}} \leftarrow \text{NTT}^{-1}(\hat{A} \circ \text{NTT}(z) - \text{NTT}(c) \circ \text{NTT}(t_1 \cdot 2^{13}))$ 
9:  $w_1 \leftarrow \text{UseHint}(h, w_{\text{appr}})$   $\triangleright$  recover the signer's commitment, Algorithm 10
10:  $\tilde{c}' \leftarrow \text{SHAKE256}(\mu \parallel w_1, 32)$ 
11: return 1 iff  $\|z\|_\infty \leq \gamma_z$  and  $\tilde{c} = \tilde{c}'$ 
```

Algorithm 9 Decompose(r) [37, Algorithm 36]

Require: $r \in \mathbb{Z}_q$

Ensure: Integers (r_1, r_0)

```
1:  $r^+ \leftarrow r \bmod q$ 
2:  $r_0 \leftarrow r^+ \bmod^\pm(2\gamma_2)$ 
3: if  $r^+ - r_0 = q - 1$  then
4:    $r_1 \leftarrow 0$ 
5:    $r_0 \leftarrow r_0 - 1$ 
6: else
7:    $r_1 \leftarrow (r^+ - r_0)/(2\gamma_2)$ 
8: return  $(r_1, r_0)$ 
```

Algorithm 10 UseHint(h, r) [37, Algorithm 40]

Require: Boolean h , $r \in \mathbb{Z}_q$

Ensure: $r_1 \in \mathbb{Z}$ with $0 \leq r_1 < M$

```
1:  $M \leftarrow (q - 1)/(2\gamma_2)$ 
2:  $(r_1, r_0) \leftarrow \text{Decompose}(r)$ 
3: if  $h = 1$  and  $r_0 > 0$  then
4:   return  $(r_1 + 1) \bmod M$ 
5: if  $h = 1$  and  $r_0 \leq 0$  then
6:   return  $(r_1 - 1) \bmod M$ 
7: return  $r_1$ 
```

- $\pi \leftarrow \text{Prove}^{\text{RO}}(w \mid \mathcal{R}, x)$ is probabilistic: on a statement-witness pair $(x, w) \in \mathcal{R}$ it outputs a proof π .
- $b \leftarrow \text{Verify}^{\text{RO}}(\pi \mid \mathcal{R}, x)$ is deterministic: it outputs a bit $b \in \{0, 1\}$ accepting or rejecting π .

When it is of no importance for the context, we will drop the RO superscript.

Completeness. For every $(x, w) \in \mathcal{R}$,

$$\Pr \left[\text{Verify}^{\text{RO}}(\pi \mid \mathcal{R}, x) = 1 \mid \pi \leftarrow \text{Prove}^{\text{RO}}(w \mid \mathcal{R}, x) \right] = 1.$$

Straightline-extractable knowledge soundness. There exists a PPT extractor Ext such that for every PPT adversary $\mathcal{A}$

$$\Pr \left[\text{Verify}^{\text{RO}}(\pi \mid \mathcal{R}, x) = 1 \wedge (x, w) \notin \mathcal{R} \mid \begin{array}{l} (Q, \pi) \leftarrow \mathcal{A}^{\text{RO}}(x) \\ w \leftarrow \text{Ext}(Q, \pi) \end{array} \right] \leq \text{negl}(\lambda),$$

where Q is the set of random-oracle query/answer pairs made by $\mathcal{A}$.

Zero knowledge. There exists a PPT simulator Sim , which may program the random oracle, such that for every $(x, w) \in \mathcal{R}$ and every PPT distinguisher $\mathcal{A}$,

$$\left| \Pr[\mathcal{A}^{\text{RO}}(\pi) = 1 \mid \pi \leftarrow \text{Prove}^{\text{RO}}(w \mid \mathcal{R}, x)] - \Pr[\mathcal{A}^{\text{RO}}(\pi) = 1 \mid \pi \leftarrow \text{Sim}^{\text{RO}}(x)] \right| \leq \text{negl}(\lambda).$$

D Fischlin’s Blueprint for Blind Signatures with VOLEith Masking for Commitments

Fischlin’s template [22] turns any signature scheme and any argument system into a blind signature: the user commits to its message, the issuer signs the commitment, and the signature the user publishes is an argument of knowledge of a signature on a commitment to that message. Blindness comes from the commitment and from the zero knowledge of the argument, unforgeability from the signature scheme, and the issuer runs nothing but its own signing algorithm – which is what lets ours be an unmodified ML-DSA. What the template leaves open is the commitment, this could for instance be a SHAKE256-based commitment (provable with Section E) or the masking as it was done in [4].

The Commitment. A VOLEith proof begins with a first message π_1 , carrying the vector commitments and the correction values that align them (Section 2.3), and it fixes every committed $\mathbb{F}_2$ value before any challenge exists. The construction exploits this twice. First, the circuit is run once on no inputs to count the VOLE relations it needs, and 256 of them are reserved as a string $\mathbf{r}$ that nothing else consumes. Being raw VOLE randomness, $\mathbf{r}$ is uniform and hidden from the issuer,

yet it is already committed, so it costs nothing beyond the proof the user has to produce anyway. Second, the message is masked not against a fresh random string but against π_1 itself,

$$\text{com} = H(\text{msg}, \pi_1) \oplus \mathbf{r},$$

which is a one-time pad under $\mathbf{r}$ and hence perfectly hiding. Masking against π_1 is what binds the user to the $\mathbb{F}_2$ relations it has already fixed: were the target independent of π_1 , a user could choose those relations after seeing the issuer's signature and then prove a different statement about the same com . Since $\mathbf{r}$ is fixed by π_1 too, the extractor of Section H recovers it from π_1 at the moment the user queries $H(\text{msg}, \pi_1)$, which is the one place the random oracle is used and all that is needed to make distinct messages give distinct commitments. The commitment is therefore free: no separate commitment scheme, no extra assumption, and no cost beyond randomness for the mask.

Algorithm 11 is the template instantiated this way, with ML-DSA as the signature scheme and the VOLEitH argument of Section 2.3 as the proof. We also implemented the construction with SHAKE256-based commitments, but do not provide the concrete numbers here, as it is just one more iteration of SHAKE256, which is cheap compared to the rest. We inherit all of it unchanged from [4], and the only thing we replace is the circuit $C_{\text{pk}, \text{msg}}$ the argument proves, which is where all of our work goes and which Algorithm 4 lays out.

Algorithm 11 Fischlin-Based ML-DSA Blind Signature, for one issuer $\mathcal{I} = (\text{pk})$

- 1: **KeyGen**(1^λ):
 - 2: Output $(\text{sk}, \text{pk}) \leftarrow \text{ML-DSA.KeyGen}(1^\lambda)$

 - 3: **Sign**₁(pk, msg):
 - 4: $(\pi_1, \mathbf{r}, \text{st}) \leftarrow P_1(C_{\text{pk}, \text{msg}})$ $\triangleright \mathbf{r} \in \{0, 1\}^{256}$
 - 5: $\text{com} \leftarrow H(\text{msg}, \pi_1) \oplus \mathbf{r}$
 - 6: Output $\rho_1 = (\text{com})$ and $S_{\mathcal{U}} = (\text{pk}, \pi_1, \text{msg}, \mathbf{r}, \text{com}, \text{st})$

 - 7: **Sign**₂($\text{sk}, \rho_1 = (\text{com})$):
 - 8: $\sigma \leftarrow \text{ML-DSA.Sign}(\text{sk}, \text{com})$ $\triangleright \text{Sign com.}$
 - 9: Output $\rho_2 = (\sigma)$

 - 10: **Sign**₃($\rho_2 = (\sigma), S_{\mathcal{U}} = (\text{pk}, \pi_1, \text{msg}, \mathbf{r}, \text{com}, \text{st})$):
 - 11: If $\text{ML-DSA.Verify}(\text{pk}, \text{com}, \sigma) \neq 1$ then output $\perp$.
 - 12: Compute $\pi \leftarrow P(C_{\text{pk}, \text{msg}}, (\mathbf{r}, \sigma))$
 - 13: **Output** $\text{sig} = (\pi)$.

 - 14: **Verify**($\text{pk}, \text{msg}, \text{sig} = (\pi)$):
 - 15: Output 1 iff $V(C_{\text{pk}, \text{msg}}, \pi) = 1$, otherwise output 0.
-

E Proving SHAKE256

The circuit recalled here is that of [4]. The input is provided as bit commitments of degree 1 or as constants. It is first padded to a multiple of the input rate $r = 1088$, after which the state absorbs and permutes one block at a time. Inside a permutation, the constants F and B fix how many of Keccak- f 's 24 rounds are evaluated without an intervening degree reduction: the prover commits to the state only at every $(F + B)$ -th round boundary, which is why $F + B$ has to divide 24. Each such segment is closed by running F rounds forward from one committed boundary and B rounds backward from the next, and asserting that the two meet in the middle. Degrees grow at different rates in the two directions: each forward step doubles the degree and each backward step triples it. Committing less often therefore trades prover work, which grows with the accumulated degree, against proof size, which is one commitment per entry.

Algorithm 12 ShakeC($[\mathbf{x}']_2, [\mathbf{st}']_2$)

Require: Fixed (F, B) with $F + B \mid 24$; committed boundary states $[\mathbf{st}']_2$ of $n_{abs} \cdot 24/(F + B)$ blocks of 1600 bits each.

- 1: $[\mathbf{x}]_2 \leftarrow \text{Pad } [\mathbf{x}']_2$ according to the SHAKE256 specification
- 2: $n_{abs} \leftarrow |\mathbf{x}|/r$ $\triangleright$ Total number of absorb steps, $r = 1088$
- 3: $[\mathbf{st}]_2 \leftarrow [0^{1600}]_2$; $\text{ctr} \leftarrow 0$
- 4: **for** $i = 0, \dots, n_{abs} - 1$ **do**
- 5: $[\mathbf{st}[0..r]]_2 \leftarrow [\mathbf{st}[0..r]]_2 + [\mathbf{x}[r \cdot i..r \cdot (i + 1)]]_2$
- 6: **for** $j = 0, \dots, 24/(F + B) - 1$ **do**
- 7: $[\mathbf{st}_{next}]_2 \leftarrow [\mathbf{st}'[\text{ctr} \cdot 1600..(\text{ctr} + 1) \cdot 1600]]_2$
- 8: **assert** $\text{ShakeFwd}_{F,B}([\mathbf{st}]_2, j) = \text{ShakeBwd}_{F,B}([\mathbf{st}_{next}]_2, j)$ $\triangleright$ F/B steps
- fwd/bwd
- 9: $[\mathbf{st}]_2 \leftarrow [\mathbf{st}_{next}]_2$; $\text{ctr} \leftarrow \text{ctr} + 1$
- 10: **return** $[\mathbf{st}[0..r]]_2$

Where the numbers in Table 6 come from. The proof sizes are almost entirely accounted for by the boundary states alone. At (F, B) , each absorb commits to $24/(F + B)$ full Keccak states, and one state is 1600 committed $\mathbb{F}_2$ values. At $\lambda = 128$ and $\tau = 16$ a committed $\mathbb{F}_2$ value costs 2 bytes, so a full state is $1600 \cdot 2 = 3.2$ KB and a single absorb costs

$$\frac{24}{F + B} \cdot 3.2 \text{ KB},$$

ranging from 76.8 KB at $(1, 0)$ to 12.8 KB at $(4, 2)$, not counting the commitments to the 7 input blocks of r bits each (15.2 KB). Over 7 absorbs this becomes 537.6 KB and 89.6 KB respectively. The remaining KB are VOLEitH overhead.

Table 6: Proving SHAKE256 with a total of 7 absorbs, the longest single hash call in our circuit, for all reasonable choices of (F, B).

Scheme (F, B)	Runtime in ms		Proof Size (KB)
	Prove	Verify	
(1, 0)	210.02	157.32	556.227
(2, 0)	141.62	107.47	287.537
(2, 1)	210.91	211.09	197.937
(3, 0)	171.84	91.07	198.157
(3, 1)	229.36	185.10	153.357
(4, 0)	340.23	90.66	153.797
(4, 2)	454.34	198.16	108.997

F Proof Strategy Details for SampleInBall

F.1 The Linear ORAM

A linear ORAM accesses a secret index x by scanning the entire array and applying a conditional update at every position, so the access pattern reveals nothing about x . This is affordable here because the array holds only $N = 256$ entries and an index is 8 bits, which keeps the indicator polynomials at a low degree. The round condition do and the counter increment are as in Section 4.2. Because every access is a scan, the construction realizes the swap of the plain Fisher–Yates shuffle (Algorithm 13) line for line, where the general ORAM has to record it as a read and a write.

Algorithm 13 SampleInBallShuffle($h, \theta_{\max}, \nu$)

```

1:  $\mathbf{c}_+ \leftarrow 0^{256}; \mathbf{c}_- \leftarrow 0^{256}$  ▷ Over  $\mathbb{F}_2$ 
2:  $\mathbf{c}_-[N - \nu..] \leftarrow h[0..\nu]$ 
3:  $\mathbf{c}_+[N - \nu..] \leftarrow 1^\nu \oplus h[0..\nu]$ 
4:  $i \leftarrow \text{BytesToBits}(N - \nu)$ 
5: for  $rnd = 0$  to  $\theta_{\max} - 1$  do
6:   if  $i = 0$  then ▷ can be achieved by adding 1 to  $i$  with overflow
7:     break
8:    $j \leftarrow h[64 + rnd \cdot 8 .. 64 + (rnd + 1) \cdot 8]$  ▷ sign bits always occupy the first 64 bits
9:   if  $j \leq i$  then
10:     $\mathbf{c}_\pm[i] \leftrightarrow \mathbf{c}_\pm[j]$ 
11:     $i \leftarrow i + 1$  ▷ Bit-addition with carry-over and overflow
12: assert  $i = 0$  ▷ enough shuffles were performed
13: return  $\mathbf{c}_+ - \mathbf{c}_-$  ▷  $\mathbf{c}_\pm$  embedded as  $\mathbb{F}_q^{256}$  vectors

```

Reading an Entry. Each equality indicator of Lemma 6 is a product of 8 linear terms and so has degree 8, which makes the entry itself, recovered as $\mathbf{c}_\pm[x] = \sum_{y=0}^{255} eq_{xy} \cdot \mathbf{c}_\pm[y]$, degree 9.

Writing to an Entry. Each round reads $\mathbf{c}_\pm[i], \mathbf{c}_\pm[j]$ and writes them back swapped. Since the positions are secret, the swap has to touch every position y :

$$\mathbf{c}_\pm[y]' := \mathbf{c}_\pm[y] + eq_{yj}(\mathbf{c}_\pm[i] - \mathbf{c}_\pm[y]) + eq_{yi}(\mathbf{c}_\pm[j] - \mathbf{c}_\pm[y]).$$

Multiplying a read value (degree 9) by an indicator (degree 8) puts the update at degree 17. The bit *do* then acts as a multiplexer on the write-back,

$$\mathbf{c}_\pm[y] \leftarrow \mathbf{c}_\pm[y] + do \cdot (\mathbf{c}_\pm[y]' + \mathbf{c}_\pm[y]),$$

so that an inactive round leaves the array untouched. This multiplication is what makes the write-back, and with it the whole circuit, degree 18.

Degree Reductions and the Two Variants. Every degree reduction buys prover time at the cost of one commitment, and there are many reduction patterns. We fix what we consider the necessary set – the write-backs, each round’s *do*, and *i* after the increment – and call it the *Short* variant. The *Fast* variant additionally reduces all $2N$ equality indicators eq_{ix}, eq_{jx} before they are used. That drops the read from degree 9 to 2 and the circuit’s top degree from 18 to 17. This improves runtime, but it also doubles the number of commitments per round.

Cost. Every degree reduction and every committed stream bit is one $\mathbb{F}_2$ commitment, costing 2 bytes. In the Short variant, each round has $2N = 512$ write-backs (each position of $\mathbf{c}_+$ and $\mathbf{c}_-$), 7 carries for *i*, one degree-reduction for *do*, and the 8 stream bits forming *j*. That is 528 commitments, or 1.06 KB per round. Fast adds the $2N = 512$ indicator reductions for a total of 1040 commitments, 2.08 KB per round. The budget $\theta_{\max}$ therefore sets the proof size almost by itself, at 1.06 KB or 2.08 KB per round of headroom. Table 3 prices both variants.

F.2 The General ORAM

This appendix collects the parts of Section 4.2 deferred there: the soundness proof of the permutation check, the identification of our construction as an instance of [48]’s RAM formalism, and the accounting behind Table 3.

Lemma 7 (Soundness of the Permutation Check). *If reads and writes are not permutations of each other (as tuples), the permutation check passes for a uniform $\chi \in \mathbb{F}_{2^{128}}$ with probability at most $n_{\text{acc}}/2^{128}$.*

Proof. Since lift is injective, reads and writes differing implies the polynomials $\prod_j (X - \text{lift}(\text{reads}[j]))$ and $\prod_j (X - \text{lift}(\text{writes}[j]))$ are not identical. Their difference is then a non-zero polynomial of degree at most n_{acc} , which by the Schwartz–Zippel lemma has at most n_{acc} roots in $\mathbb{F}_{2^{128}}$. $\square$

From Permutation to Consistency. Lemma 7 only says that every claimed read matches *some* recorded write to the same address; on its own it does not say the read returns the *most recent* write to that address. Rather than re-derive this, we identify our construction as an instance of the read/write RAM formalism of [48], which gives the following.

Lemma 8 (Consistency, via [48, Game 4, Lemma 6]). *For every address a , the lists of reads and writes to it equals entry for entry and in order: the m -th write to a is read exactly once, by the read accompanying the $(m+1)$ -th write to a or by the teardown read.*

[48, Game 4] models exactly this situation: an address set $\{a_1, \dots, a_n\}$ with one initial write $(a_k, c_{k, \text{begin}}, 0)$ per address, followed by T rounds in which an adversary picks an address $u_i \in [n]$, appends an arbitrary claimed read (a_{u_i}, d_i, t_i) to **reads** and a fresh write (a_{u_i}, c_i, i) to **writes**. The final teardown then adds a final read for each position. The adversary wins iff **reads** $\sim$ **writes** and every claimed timestamp satisfies $(i - t_i) \in \{1, \dots, T\}$. We instantiate the game with $n = N$ addresses and $T = 2\theta_{\max}$ rounds (splitting each of our rounds into its access to i and its access to j), each a single-address read/write exactly as in Game 4. Our write times $2rnd + 1$ and $2rnd + 2$ already coincide with the running round index, so the strict-inequality asserts $t_i < 2rnd + 1$ and $t_j < 2rnd + 2$ give precisely their lower bound $t_i < i$; the upper bound $i - t_i \leq T$ holds automatically, since all our timestamps lie in $\{0, \dots, 2\theta_{\max}\} = \{0, \dots, T\}$. Our teardown pass is their post-loop step.

Unlike [48]’s generic protocol, which enforces the timestamp bound with a second, separate permutation-based set-membership gadget (their Section 4.2) carrying its own Schwartz–Zippel loss, our bound is an *exact* $\mathbb{F}_2$ arithmetic identity: it is a bitwise less-than comparison of committed timestamp bits against a public constant, (like the equality indicator of Lemma 6). The soundness loss is therefore exactly the $(n_{\text{acc}}/2^{128})^t$ of Lemma 7.

Cost Comparison. Taking the 229.1 KB of *Short* at $\theta_{\max} = 213$ as the baseline, the difference to the general ORAM can be read off as a few corrections. Each round saves $528 - 40 = 488$ commitments, 103 944 over the 213 rounds, while the teardown pass adds $N \cdot 11 = 2816$, for a net saving of ≈ 202.3 KB. Two things are then added back: the $t \cdot 2 \cdot \lfloor (n_{\text{acc}} - 2)/(d_{\max} - 1) \rfloor = 12$ reductions of the permutation check over $\mathbb{F}_{2^{128}}$, and the VOLEitH overhead of raising $d_{\max}$ from 18 to 197, which means $d_{\max} - 1 = 196$ full-field $\mathbb{F}_{2^\lambda}$ masks plus one per ORAM degree reduction, 208 in all. The CRT construction of Section 2.4 supplies those for ≈ 8.3 KB, at 39 B correction per mask plus the 16 B difference each of the 12 reductions publishes; bit-by-bit they would take λ subfield rows each, $208 \cdot \lambda \cdot \tau/8 \approx 53$ KB. Even that stays well inside the 202.3 KB the switch saves, so the comparison does not rest on the CRT construction.

G Range Checks in Detail

G.1 Digit Domains and Indicators

A committed $\mathbb{F}_q$ wire is not confined to $\{0, \dots, B-1\}$, so the circuit asserts the vanishing product $\prod_{k=0}^{B-1} (x - k) = 0$ to pin it to the domain, and the indicators become Lagrange basis polynomials on that domain.

Lemma 9. *Fix $t \in \{0, \dots, B-1\}$ and $x \in \mathbb{F}_q$ with $\prod_{k=0}^{B-1} (x - k) = 0$. Then*

$$eq_t(x) := \frac{\prod_{k \neq t} (x - k)}{\prod_{k \neq t} (t - k)}$$

evaluates to 1 if $x = t$ and to 0 for every other $x \in \{0, \dots, B-1\}$.

Proof. Up to the public, precomputed normalizer, eq_t is the degree- $(B-1)$ Lagrange basis polynomial for node t on the domain $\{0, \dots, B-1\}$: it vanishes at every other node $k \neq t$ by construction and is normalized to 1 at t . The hypothesis restricts x to the domain nodes, so this is exhaustive. $\square$

Summing $eq_t(x)$ over $t < c$, or over its complement where that is smaller, gives $less(x, c)$, and the comparison then proceeds exactly as in the bit case, one degree- $(B-1)$ factor per digit.

G.2 The Hint, and Recovering w_1

Algorithm 14 UsingHintC($([h_i]_q)_{i \in [N \cdot K]}$, $([\tilde{w}_{0,i}]_q)_{i \in [N \cdot K]}$, $([w_{\text{appr},i}]_q)_{i \in [N \cdot K]}$)

```

1:  $sum \leftarrow \sum_{i=1}^{N \cdot K} [h_i]_q$ 
2: assert  $\prod_{x=0}^{\omega} (sum - x) = 0$  ▷ Hamming weight of  $\mathbf{h}$  is at most  $\omega$ 

3: for  $i = 1$  to  $N \cdot K$  do
4:   assert  $[h_i]_q \cdot ([h_i]_q - 1) = 0$  ▷  $h_i$  is a bit
5:    $([x_1]_q, \dots, [x_\ell]_q) \leftarrow [\tilde{w}_{0,i}]_q$  ▷ base- $B$  digits of  $u := w_{0,i} + \gamma_2$ 
6:    $leq_1 \leftarrow 1$ ;  $leq_2 \leftarrow 1$ ;  $leq_3 \leftarrow 1$ 
7:   for  $k = \ell$  down to 1 do
8:     assert  $\prod_{t=0}^{B-1} ([x_k]_q - t) = 0$  ▷  $x_k$  is a valid base- $B$  digit
9:      $leq_1 \leftarrow eq_{c_k}([x_k]_q) \cdot leq_1 + less([x_k]_q, c_k)$  ▷  $c_k$ :  $k$ -th digit of  $2\gamma_2$ ; Lemma 9
10:     $leq_2 \leftarrow eq_{c'_k}([x_k]_q) \cdot leq_2 + less([x_k]_q, c'_k)$  ▷  $c'_k$ :  $k$ -th digit of  $\gamma_2$ 
11:     $leq_3 \leftarrow eq_0([x_k]_q) \cdot leq_3$ 
12:   assert  $leq_1 = 1$  ▷  $u \leq 2\gamma_2$ 
13:    $b \leftarrow leq_2$  ▷  $b = 1 \iff u \leq \gamma_2 \iff w_{0,i} \leq 0$ 
14:    $w_{0,i} \leftarrow \sum_{k=1}^{\ell} [x_k]_q \cdot B^{\ell-k} - \gamma_2$ 
15:    $w_1 \leftarrow ([w_{\text{appr},i}]_q - w_{0,i}) \cdot (2\gamma_2)^{-1}$ 
16:   assert  $leq_3 \cdot w_1 = 0$  ▷  $u = 0 \Rightarrow w_1 = 0$ 
17:    $core \leftarrow \prod_{k=1}^{M-2} (w_1 - k)$ 
18:    $t_a \leftarrow core \cdot w_1$ ;  $t_b \leftarrow core \cdot (w_1 - (M-1))$ 
19:   assert  $t_a \cdot (w_1 - (M-1)) = 0$  ▷  $w_1 \in [0, M]$ 
20:    $eq_0 \leftarrow t_b \cdot ((-1)^{M-1} (M-1)!)^{-1}$ ;  $eq_{M-1} \leftarrow t_a \cdot ((M-1)!)^{-1}$  ▷ Lemma 9
21:    $adj \leftarrow (1 - 2b) + M \cdot (b \cdot eq_0 - (1 - b) \cdot eq_{M-1})$ 
22:    $[w_{1,i}]_q \leftarrow w_1 + [h_i]_q \cdot adj$ 

23: for  $i = 0$  to  $\lceil N \cdot K/3 \rceil - 1$  do
24:    $v \leftarrow \sum_{t=0}^2 [w_{1,3i+t}]_q \cdot 2^{6t}$ 
25:    $(\beta_1, \dots, \beta_{18}) \leftarrow \mathbf{a2b}(v)$ 
26:   append  $(\beta_1, \dots, \beta_{18})$  to  $[w_1]_2$ 
27: return  $[w_1]_2$ 

```

Computing w_0 . The bound on w_0 carries one subtlety the bound on z does not. **Decompose** attains $w_{0,i} = -\gamma_2$ (Section C.2), so the shifted $u := w_{0,i} + \gamma_2$ ranges over $[0, 2\gamma_2]$ – one value more than a step of w_1 covers, and $(w_1, u=2\gamma_2)$ and $(w_1+1, u=0)$ then denote the same $w_{\text{appr},i}$. Taking the narrower window $(0, 2\gamma_2]$ instead is unique but leaves $w_{\text{appr},i} = q - \gamma_2$ with no witness at all, which an honest signature produces with probability $\approx N \cdot K/q \approx 2^{-13}$. We therefore keep the wider window and assert $u = 0 \Rightarrow w_1 = 0$, which removes the second of each colliding pair (**Decompose** returns $w_{0,i} = -\gamma_2$ only with $w_1 = 0$). The circuit therefore runs three comparisons on the same digits rather than one: $u \leq 2\gamma_2$ (asserted), $u \leq \gamma_2$ (kept as the direction indicator $b := [w_{0,i} \leq 0]$ for the hint

application), and $u \leq 0$ (equality test for the tie-break). Undoing the shift and scaling then gives $w_1 = (\mathbf{w}_{\text{appr}} - w_0) \cdot (2\gamma_2)^{-1}$.

Bounding the Hint. Every h_i must be a bit, and the hint's Hamming weight must not exceed $\omega = 80$. Since $N \cdot K < q$ we sum all hint bits directly in $\mathbb{F}_q$ without wraparound and enforce the bound with a single vanishing product $\prod_{x=0}^{\omega} (\sum_i h_i - x) = 0$, which pins the sum to $\{0, \dots, \omega\}$. That product reaches degree $\omega + 1 = 81$, so for the smaller bases the digit comparison adds no degree of its own.

Range-Checking w_1 and Applying the Hint. w_1 must lie in $[0, M)$ for $M := (q - 1)/(2\gamma_2) = 44$. Rather than decompose it as well, we vanish the length- M product directly, and get the two indicators the hint correction needs. Let $\text{core} := \prod_{k=1}^{M-2} (w_1 - k)$, which is $M - 2$ linear factors, and obtain the boundaries

$$t_a := \text{core} \cdot w_1 = \prod_{k=0}^{M-2} (w_1 - k), \quad t_b := \text{core} \cdot (w_1 - (M - 1)) = \prod_{k=1}^{M-1} (w_1 - k).$$

Multiplied by the one remaining factor, $t_a \cdot (w_1 - (M - 1))$ is the degree- M vanishing check. Taken on their own the two are the indicators we need: t_a vanishes everywhere except at $w_1 = M - 1$, so scaled by $1/(M - 1)!$ it is eq_{M-1} , and t_b scaled by $1/(-1)^{M-1}(M - 1)!$ is eq_0 (Lemma 9).

UseHint adds or subtracts 1 from w_1 depending on h and b , wrapping at 0 and M . We fold this into one correction

$$\text{adj} := (1 - 2b) + M \cdot (b \cdot eq_0 - (1 - b) \cdot eq_{M-1}), \quad w_{1,\text{out}} = w_1 + h \cdot \text{adj},$$

where $(1 - 2b)$ takes the direction from the sign bit and the second term only applies at the two boundaries. We also tried carrying $\mathbf{h}$ and this correction as native $\mathbb{F}_2$ bits, which turned out worse than keeping the whole step in $\mathbb{F}_q$.

G.3 Pricing the Variants

Per coordinate a digit variant spends $\ell \cdot 46$ bytes, 828 at $B = 2$ down to 138 at $B = 64$, against $18 \cdot 2 = 36$ bytes on the $\mathbb{F}_2$ side. For $\mathbf{z}$, going from $B = 8$ to $B = 64$ drops $3 \cdot 1024$ committed $\mathbb{F}_q$ elements, 141.3 KB, and adds $189 - 42 = 147$ degrees at 6.46 bytes apiece, so 40.6 KB, for a net saving of 100.7 KB. Committing bits instead removes the $\mathbb{F}_q$ commitments altogether and leaves 1024 conversions as the dominant cost: cheap enough under edaBitE that the trade pays off, and not under Mystique, where certifying them needs about twelve sacrificed edaBits apiece (Lemma 1), 1132 KB on its own.

For $\mathbf{w}_0$ the same accounting holds, but every variant additionally commits h_i and its coordinate of $\mathbf{w}_{\text{appr}}$, and runs the 342 packed conversions. At $B = 8$ those eight $\mathbb{F}_q$ elements per coordinate account for some 375 KB. Committing $\mathbf{w}_0$ as bits trades the ℓ digits for 18 far cheaper ones, but needs three conversions per coordinate where the digits need none – one for $w_{0,i}$ and one each for b and

the equality bit – which is what makes the bit variant less attractive here than it is for z . We provide standalone benchmarks for all variants of Algorithm 7 (Table 7) and UsingHintC (Table 8).

Table 7: Standalone cost of the range check on z , for each base- B digit decomposition and for committing z directly as $\mathbb{F}_2$ bits under both Mystique and edaBitE.

Variant	(B, ℓ)	Prove Time (ms)	Verify Time (ms)	Size (KB)
BitWise	(2, 18)	61.00	50.55	858.657
TwoBits	(4, 9)	51.70	30.07	437.205
ThreeBits	(8, 6)	78.15	25.54	300.033
FourBits	(16, 5)	198.35	27.02	262.037
FiveBits	(32, 4)	481.93	31.91	228.457
SixBits	(64, 3)	1865.60	39.70	199.293
edaBitE	(2, 18)	1416.20	754.99	133.472
Mystique	(2, 18)	2508.70	287.23	1268.237

Table 8: Standalone cost of UsingHintC for each base- B digit decomposition of w_0 , and for committing w_0 directly as $\mathbb{F}_2$ bits under both Mystique and edaBitE. The digit decompositions also use edaBitE for the final conversions; the entry in the list refers to how the range is checked. The Mystique row uses cut-and-bucketing for all conversions.

Variant	(B, ℓ)	Prove Time (ms)	Verify Time (ms)	Size (KB)
BitWise	(2, 18)	132.43	61.54	1017.357
TwoBits	(4, 9)	139.17	42.88	593.421
ThreeBits	(8, 6)	207.80	36.78	452.109
FourBits	(16, 5)	476.47	39.00	405.005
FiveBits	(32, 4)	1029.50	44.30	370.597
SixBits	(64, 3)	4104.80	62.35	341.433
edaBitE	(2, 18)	1685.50	858.84	398.221
Mystique	(2, 18)	6898.20	703.38	3569.771

H Proof of Theorem 2

Theorem 2. *Let ML-DSA be EUF-CMA secure, let H be modeled as a random oracle, and let $VOLeith$ be zero-knowledge and straightline-extractable knowledge-sound with extraction error ε . Then Algorithm 11 satisfies Definitions 1, 5 and 6 for $\mathcal{I} = (\text{pk})$: blindness reduces to zero-knowledge, and one-more unforgeability reduces to the EUF-CMA security of ML-DSA under Q signing queries, with an additive loss of $(Q + 1)\varepsilon$ against an adversary returning $Q + 1$ forgeries.*

Proof. The theorem follows directly from [4, Theorem 1 and 2] instantiated as above.

Correctness. An honest sig is $P(C_{\text{pk}, \text{msg}}, (\mathbf{r}, \sigma))$ for a σ that Sign_3 has already checked against ML-DSA.Verify , and $C_{\text{pk}, \text{msg}}$ accepts exactly those $(\mathbf{r}, \sigma)$ for which that check passes (Section 4). Completeness of the argument system gives $\text{Verify}(\text{pk}, \text{msg}, \text{sig}) = 1$.

Blindness. $\mathbf{r}$ is VOLE randomness used nowhere else, so $\text{com} = H(\text{msg}, \pi_1) \oplus \mathbf{r}$ is a one-time pad and each ρ_1 the issuer sees is uniform and independent of the message it hides. What Adv_3 then sees is the two proofs, each of which is fully simulatable, and the commitment was padded with the non-revealed uniform mask.

One-more unforgeability. For $\mathcal{I} = (\text{pk})$ the sub-list $\mathcal{J}$ of Definition 1 contains one key, and $Q_{\mathcal{J}}$ is the number Q of signing queries and Adv returns $\ell > Q$ forgeries. Because the issuer runs an ordinary ML-DSA.Sign on com , a reduction $\mathcal{B}$ playing EUF-CMA answers every Sign_2 query by relaying com to its own signing oracle, and so makes the same Q signing queries Adv does. It then runs the knowledge extractor on each of the ℓ proofs, at a cost of $\ell\varepsilon$, obtaining $\mathbf{r}_i$ and a σ_i that ML-DSA.Verify accepts on com_i under pk . Only $Q < \ell$ commitments were ever queried, so some com_{i^*} was not, and $\mathcal{B}$ returns $(\text{com}_{i^*}, \sigma_{i^*})$.

The one thing this needs of the commitment is that the com_i be pairwise distinct, and it is the only place the random oracle is used. $\mathbf{r}$ is reserved VOLE randomness and hence fixed by π_1 , so the extractor recovers $\mathbf{r}_i$ from $\pi_{1,i}$ already when Adv queries $H(\text{msg}_i, \pi_{1,i})$, and the masked value is a fresh uniform string for each distinct query. The msg_i are distinct by assumption, hence so are the queries, and the com_i collide with probability at most $\binom{\ell}{2}2^{-256}$. This is the same extractor [4] uses. $\square$

I Issuer-Hiding and Attributes

I.1 Issuer-Hiding Selection Gadget

The one-hot vector expansion, and the public key selection from a publicly known table are spelled out in `IssuerHidingC` (Algorithm 15).

Algorithm 15 IssuerHidingC($(([\beta_i]_q)_{i \in [\lceil \log n \rceil]}, (\mathbf{tr}_j, \hat{\mathbf{A}}_j, \mathbf{t}_{1,j})_{j \in [n]}))$

```

1:  $s \leftarrow 0$ 
2: for  $i = 1$  to  $\lceil \log n \rceil$  do
3:   assert  $[\beta_i]_q \cdot ([\beta_i]_q - 1) = 0$   $\triangleright \beta_i$  is a bit
4:    $s \leftarrow 2 \cdot s + [\beta_i]_q$ 
5:  $([b_1]_2, \dots, [b_{\lceil \log n \rceil}]_2) \leftarrow \text{a2b}(s)$ 

6:  $\mathbf{f}_2 \leftarrow (1)$ ;  $\mathbf{f}_q \leftarrow (1)$   $\triangleright$  each starts as a vector, doubling in length every iteration
7: for  $i = 1$  to  $\lceil \log n \rceil$  do
8:    $\mathbf{f}_2 \leftarrow (\mathbf{f}_2[k] + \mathbf{f}_2[k] \cdot [b_i]_2, \mathbf{f}_2[k] \cdot [b_i]_2)_{k \in [|\mathbf{f}_2|]}$   $\triangleright$  double the prefix, over  $\mathbb{F}_2$ 
9:    $\mathbf{f}_q \leftarrow (\mathbf{f}_q[k] - \mathbf{f}_q[k] \cdot [\beta_i]_q, \mathbf{f}_q[k] \cdot [\beta_i]_q)_{k \in [|\mathbf{f}_q|]}$   $\triangleright$  same, over  $\mathbb{F}_q$ 
10: if  $n < 2^{\lceil \log n \rceil}$  then
11:    $\mathbf{f}_2 \leftarrow \text{degree\_reduce}(\mathbf{f}_2[1..n])$   $\triangleright$  drop the padding entries and degree reduce
12:    $\mathbf{f}_q \leftarrow \text{degree\_reduce}(\mathbf{f}_q[1..n])$ 
13:   assert  $\sum_{k=1}^n \mathbf{f}_q[k] = 1$   $\triangleright$  forces the secret index into  $[0, n)$ 

14:  $\mathbf{tr} \leftarrow \sum_{j=1}^n \mathbf{f}_2[j] \cdot \mathbf{tr}_j$ ;  $\hat{\mathbf{A}} \leftarrow \sum_{j=1}^n \mathbf{f}_q[j] \cdot \hat{\mathbf{A}}_j$ ;  $\mathbf{t}_1 \leftarrow \sum_{j=1}^n \mathbf{f}_q[j] \cdot \mathbf{t}_{1,j}$ 
15: return  $\mathbf{tr}, \hat{\mathbf{A}}, \mathbf{t}_1$ 

```

I.2 Proof of Theorem 3

Theorem 3 (Security with Issuer-Hiding). *Let ML-DSA be EUF-CMA secure, let $\mathbf{H}$ be modeled as a random oracle, and let VOLEitH be zero-knowledge and straightline-extractable knowledge-sound with extraction error ε . Then Algorithm 11, instantiated with the circuit $C_{\mathcal{I}, \text{msg}}$, satisfies Definitions 1 and 5 to 7 for a list of n issuers. One-more unforgeability loses a factor n against Theorem 2 and reduces to EUF-CMA under $\max_j Q_j$ signing queries. The issuer-hiding advantage is at most twice the zero-knowledge advantage of VOLEitH .*

Proof. Correctness and blindness are those of Theorem 2, neither of which inspects the circuit being proved.

For one-more unforgeability, let Adv name a sub-list $\mathcal{J}$ and return $\ell > Q_{\mathcal{J}}$ forgeries $(\mathcal{I}_i, \text{msg}_i, \text{sig}_i)$ with every $\mathcal{I}_i \subseteq \mathcal{J}$. Extraction yields $\mathbf{r}_i$, a signature σ_i and the index bits, hence one row j_i : the one-hot vector admits a single entry, and for n not a power of two the assert $\sum_k \mathbf{f}_q[k] = 1$ rules out an index outside $\mathcal{I}_i$. So σ_i verifies under the honest pk_{j_i} on com_i , and $\text{pk}_{j_i} \in \mathcal{J}$ because $\mathcal{I}_i \subseteq \mathcal{J}$: a signature the adversary obtained from an issuer outside $\mathcal{J}$ verifies under a key no forgery names, and is of no use to it. The com_i are pairwise distinct by the binding argument of Theorem 2, and only $Q_{\mathcal{J}}$ commitments were ever queried to issuers of $\mathcal{J}$, so some com_{i^*} was queried to none of them. A reduction that guesses $\text{pk}_{j_{i^*}} \in \mathcal{J}$, plants its EUF-CMA challenge key in that row and generates the other $n - 1$ pairs itself answers every query, to issuers inside $\mathcal{J}$ and outside alike, and returns $(\text{com}_{i^*}, \sigma_{i^*})$, whence the factor n . Only the queries to row j_{i^*} are relayed to the EUF-CMA signing oracle, every other issuer being one the

reduction can sign for itself, so it makes $Q_{j_i^*} \leq \max_j Q_j$ signing queries however the adversary spreads its total.

For issuer-hiding, $C_{\mathcal{I}, \text{msg}}$ is fixed by $\mathcal{I}$ and msg alone, the index entering only as a witness, so both values of b prove the same statement and both, by correctness, yield an accepting proof. Replacing Sign_3 by the zero-knowledge simulator on it costs one distinguishing advantage and leaves sig a function of $(\mathcal{I}, \text{msg})$, so Adv_2 answers correctly with probability exactly $1/2$; undoing the hop on the other side gives the bound. $\square$

1.3 Attributes

Our implementation offers five independently togglable checks over aux – in our example first and last name, place of birth, birth date and expiration date: exact matches on name and on place of birth, not-yet-expired, age within a range, and today being the holder’s birthday. The holder picks a **Policy** among them. All five are computed whatever the policy says, and only the enabled ones are asserted (Algorithm 16), so the shape of the circuit, and with it the proof size, is independent of the policy. Computing the unused ones costs nothing worth saving in our example.

Algorithm 16 ComputeAllChecks($[x]_2$, Policy)

- 1: parse $[x]_2$ into $(\text{first}, \text{last}, \text{place}, \text{birth}) = (\text{day}, \text{month}, \text{year}), \text{exp} = (\text{day}, \text{month}, \text{year})$
 - 2: In our example: names and places 35 Bytes; days 5 bits; months 4 bits; years 12 bits
 - 3: $r_{\text{name}} \leftarrow (\text{first}_i + \text{Policy.first}_i)_i \parallel (\text{last}_i + \text{Policy.last}_i)_i$
 - 4: $r_{\text{place}} \leftarrow (\text{place}_i + \text{Policy.place}_i)_i$
 - 5: $r_{\text{bday}} \leftarrow (\text{birth.day}_i + \text{Policy.today.day}_i)_i \parallel (\text{birth.month}_i + \text{Policy.today.month}_i)_i$
 - 6: $r_{\text{exp}} \leftarrow [\text{exp} < \text{Policy.today}]$
 - 7: $r_{\text{age}} \leftarrow [\text{birth} > \text{Policy.today} - \text{Policy.min}] \vee [\text{birth} \leq \text{Policy.today} - \text{Policy.max} - 1]$
 - 8: **for** each check $c \in \{\text{name}, \text{place}, \text{exp}, \text{age}, \text{bday}\}$ enabled by Policy **do**
 - 9: **assert** $r_c = 0$ $\triangleright$ Depending on policy: all elements, or just single value
-

Two primitives cover all five checks, and neither commits anything beyond the attribute bits themselves. Exact matches compare committed bits against a public constant and stay at degree 1, while not-expired and the two age bounds reuse the bitwise comparison against a public constant that already bounds the ORAM timestamps in Section 4.2, peaking at the 21 bits of a packed date. An expiration date also gives revocation for free. A credential identifier checked against a public revocation list would extend this at essentially only a runtime

cost, and such a list could drop each entry once the credential behind it expires on its own.