Post-Quantum Anonymous Credentials and Blind Signatures from ML-DSA

Thibauld Feneuil, Killian Provin, and Matthieu Rivain

CryptoExperts, Paris, France
thibauld.feneuil@cryptoexperts.com
killian.provin@cryptoexperts.com
matthieu.rivain@cryptoexperts.com

Abstract. Anonymous credentials allow users to prove properties of their identity, such as their age or nationality, without revealing any other personal information. For institutional deployment, it is particularly desirable to build such systems from standardized post-quantum signature schemes, allowing issuers to retain their existing signing procedures. However, proving knowledge of a valid ML-DSA signature in zero knowledge is costly: ML-DSA verification mixes binary operations, notably SHAKE/Keccak, with arithmetic over $\mathbb{F}_q$, together with sampling and decomposition procedures that are hard to arithmetize efficiently. As a result, recent concurrent works produce large proofs, ranging from several hundred kilobytes to approximately 1 MB.

In this paper, we present an efficient zero-knowledge proof of knowledge (ZK-PoK) of an ML-DSA signature. Our approach separates the binary and prime-field computations and relies on $\mathsf{SmallWood}^+$, a hash-based zero-knowledge argument system built on the sum-check and GKR protocols. We use it to build blind signatures and anonymous credentials from ML-DSA.

For ML-DSA-44, our ZK-PoK of a valid signature on a 32-byte private message has a proof size of about 87 KB. Our blind signature, which essentially consists of such a ZK-PoK, achieves a similar size of about 88 KB. We apply our construction to ISO/IEC 18013-5 mdoc credentials instantiated with ML-DSA, including device binding: the holder proves knowledge of an issuer-signed credential and of a device signature on the verifier's challenge under the certified, hidden device key. The resulting presentation proofs have a size of about 200 KB, compared with about 458 KB in a recent concurrent work. Our proof-of-concept implementation, partly written in Python and not yet optimized, generates the ZK-PoK and the blind signature in about 2 seconds and a credential presentation in about 15 seconds; we expect an optimized implementation to be significantly faster.

Table of Contents

1 Introduction . . . 3
2 Preliminaries . . . 5
2.1 The ML-DSA Signature Scheme . . . 5
2.2 Proof Systems and SmallWood$^+$ . . . 7
2.3 Arithmetic Tools . . . 7
3 Proving Knowledge of an ML-DSA Signature: A Two-Field Protocol . . . 10
3.1 Proven Relation . . . 11
3.2 Splitting ML-DSA Verification . . . 12
3.3 Cross-Field Consistency . . . 14
3.4 Overall Protocol . . . 15
3.5 Circuit Specifications . . . 16
3.6 Security of the Protocol . . . 18
4 Binary Circuit . . . 19
4.1 SHAKE Evaluations with GKR . . . 20
4.2 Binding the Key and the Message . . . 20
4.3 Generating the Sampler Streams . . . 20
4.4 Recomputing the Challenge Seed . . . 21
4.5 Public Verification Key or Public Message . . . 21
4.6 Circuit Cost . . . 22
5 Prime-Field Circuit . . . 22
5.1 Bits over the Prime Field . . . 23
5.2 Matrix Expansion . . . 25
5.3 Challenge Sampling . . . 26
5.4 Verification Equation . . . 28
5.5 Response and Hint Bounds . . . 28
5.6 Hint Correction . . . 29
5.7 Common Lookup . . . 31
5.8 Consistency Tag . . . 32
5.9 Completeness and Soundness . . . 34
5.10 Public Verification Key . . . 35
5.11 Circuit Cost . . . 35
6 Applications . . . 37
6.1 Proof of Knowledge of an ML-DSA Signature . . . 37
6.2 Blind Signature . . . 38
6.3 Anonymous Credentials with Device Binding . . . 39
A Soundness of SmallWood$^+$ . . . 43

1 Introduction

Anonymous credentials [CL01] allow a user to obtain a credential from an issuer, and to later prove to a verifier that it holds a valid credential whose attributes satisfy some predicate, such as being over 18, without revealing anything else and without its presentations being linkable. They are becoming a key building block of digital identity. In the European Union, the revised eIDAS regulation [Eur24b] requires the member states to provide their citizens with a European Digital Identity Wallet (EUDI Wallet), whose Architecture and Reference Framework [Eur24a] asks for selective disclosure of attributes and unlinkable presentations, two goals for which cryptographers have recommended anonymous credentials [BBC+24]. Likewise, Google Wallet relies on zero-knowledge proofs for privacy-preserving age verification [Fas26].

Most anonymous credential systems combine a signature of the issuer on the attributes of the user with a zero-knowledge proof of knowledge of this signature. Dedicated signature schemes, such as CL [CL03, CL04], BBS [BBS04], or PS [PS16] signatures, make this proof very efficient. In practice, however, issuers are institutions whose signing keys are held in certified hardware, which only supports standardized algorithms, and credential formats such as ISO/IEC 18013-5 mdoc [II21] are built on standard signatures. Changing the issuance infrastructure is a major obstacle to deployment. This led Frigo and shelat [Fas26] to build anonymous credentials from plain ECDSA signatures: the issuer is left untouched, and the holder proves knowledge of a valid ECDSA signature with a general-purpose zero-knowledge argument. This is the approach deployed in Google Wallet.

ECDSA is however broken by quantum computers [Sho94], and the transition to post-quantum cryptography is underway. NIST standardized its first post-quantum signature schemes in 2024, with ML-DSA [Nat24] as the primary one, and the issuers of digital identities will have to migrate to these schemes. Post-quantum anonymous credentials are thus needed and, for the same reasons as above, they should be built from the standardized signatures. Lattice-based anonymous credentials exist [BLNS23, JRS23], but they rely on dedicated signature schemes. A middle ground is to adapt the standard: zkDilithium [PWFW23] is a variant of Dilithium, the scheme underlying ML-DSA, whose verification is designed to be friendly to zero-knowledge proofs. Such a variant is however no longer the standard, so that issuers cannot rely on certified implementations and existing infrastructure, which defeats the purpose of building on a standardized signature. A zero-knowledge proof of knowledge of an ML-DSA signature is therefore a key building block for post-quantum digital identity: it turns signatures issued with the standard into anonymous credentials and blind signatures, without any change on the issuer side. It is also a demanding statement to prove. ML-DSA verification mixes binary computations, notably the SHAKE hash function, with arithmetic modulo a 23-bit prime, together with rejection samplers and rounding steps that are costly to arithmetize over either field. Making such a proof compact and efficient enough for deployment, while relying on post-quantum assumptions only, is the goal of this work.

Our contributions. We present an efficient zero-knowledge proof of knowledge of an ML-DSA signature, which is, together with the concurrent works discussed below, one of the first proofs designed for this standard. From it, we derive blind signatures and anonymous credentials in which the signer runs unmodified ML-DSA. Our construction splits the verification of ML-DSA between a binary circuit and a prime-field circuit, proves each of them over its native field with the hash-based argument system SmallWood+ [FPR26], and ties them together with a lightweight consistency test. It requires no trusted setup and only relies on hash functions. Specifically, our contributions are the following.

- *A framework for proving ML-DSA verification over two fields.* We formalize the proven relation, including the bounded samplers allowed by the standard, and split it into a binary relation and a prime-field relation that share a small cross-field witness. We show that this split is exact, specify the two circuits accordingly, and prove that the resulting protocol is zero knowledge and round-by-round knowledge sound, hence compatible with the Fiat–Shamir transform, for any underlying commit-and-prove system with these properties.
- *A cross-field consistency test.* To ensure that both circuits use the same cross-field witness, we introduce a test based on a public random binary linear map and a committed mask. Unlike the MAC-based test of [Fas26], it is linear, and thus essentially free, in the binary circuit, and it reduces to affine equations in the prime-field circuit.

– *An optimized prime-field circuit.* While the binary circuit essentially consists of Keccak permutations, the prime-field circuit concentrates the difficulties of ML-DSA verification: rejection sampling, a data-dependent shuffle, and rounding with hints. Our main technical contribution is a set of techniques that avoid emulating these steps inside the circuit. We replace the sequential execution of the two samplers by checks on their outcome, with an ordered-selection argument for the matrix expansion and a memory-consistency argument for the challenge sampling. We give an exact reformulation of the hint correction as one table lookup per coefficient, which we batch with all the range checks of the verification in a single lookup argument. We also pack the bits shared with the binary circuit into elements of a multiplicative subgroup of $\mathbb{F}_q$, which shrinks the committed witness. As a result, all the nonlinear computations of the circuit are shallow layered circuits, and for ML-DSA-44 with a public verification key, the circuit commits to about 16 500 field elements only.
– *Applications and implementation.* We instantiate our protocol for three use cases: a proof of knowledge of an ML-DSA signature on a hidden message, a blind signature whose signer runs unmodified ML-DSA, and anonymous credentials following the ISO/IEC 18013-5 mdoc format with device binding, where a single proof covers the signature of the issuer and a signature of the device under its certified, hidden key. We implement the three applications for all ML-DSA parameter sets, in Python, on top of a SmallWood$^+$ backend [FPR26] provided to us. For ML-DSA-44, the proofs have a size of 86.5 KB for the proof of knowledge, 87.9 KB for the blind signature, and 199.7 KB for the credential presentation. They are generated in 2.125, 2.046, and 15.472 seconds, and verified in 1.737, 1.745, and 13.565 seconds, respectively. These timings are those of a non-optimized prototype: only the Keccak circuits are proven with dedicated C routines, while all the other computations go through generic code orchestrated in Python, so that we expect an optimized implementation to be significantly faster.

Table 1: Zero-knowledge proofs of unmodified ML-DSA-44 signatures. Sizes are in KB (1000 bytes); those of [SG26], reported in KiB (1024 bytes), are converted accordingly. The times of [BB26, SG26] are single-core medians on an AMD EPYC 9754, as measured in [SG26, Tables 4 and 5]. * Our times are medians of three executions of the non-optimized prototype on an Intel Core Ultra 7 255H (Section 6). The security level of [BB26] is a union bound over the whole protocol; that of [SG26] is the largest round error. † Our entries bound the bad-challenge probability per fresh classical random-oracle trial after grinding (Appendix A).

Statement	Work	Security	Size (KB)	Prover (ms)	Verifier (ms)
Proof of knowledge	[SG26], ROM	2^{-114}	218.6	62.0	31.9
	[SG26], QROM	2^{-236}	562.4	146.8	72.1
	This work†	2^{-128}	86.5	2125*	1737*
Blind signature	[BB26]	$\approx 2^{-128}$	1011.9	1761.0	897.9
	This work†	2^{-128}	87.9	2046*	1745*
mdoc presentation	[SG26], ROM	$2^{-112.7}$	458.3	473.7	247.0
	[SG26], QROM	2^{-236}	1043.3	921.7	486.3
	This work†	2^{-128}	199.7	15472*	13565*

Concurrent work. Two recent and independent works also prove knowledge of unmodified ML-DSA signatures with hash-based proof systems, both for ML-DSA-44 only. Table 1 compares them with ours.

– *Baum and Beckmann [BB26]* build blind signatures, optionally issuer-hiding. They prove the binary and the prime-field computations with two VOLE-in-the-Head instances, one over $\mathbb{F}_2$ and one with native arithmetic over $\mathbb{F}_q$, linked by a new batch conversion check called edaBitE. Range checks rely on digit decompositions, and the challenge sampling on an oblivious RAM. Their proofs are about 1 MB, mostly due to the committed digits of the range checks, that is, about 11.5 times larger than ours.
– *Sharp and Grubbs [SG26]* present GordieHowe, a language, a certifying compiler, and a Ligero-based proof system for statements spanning several fields, which they apply to ECDSA, RSA, and ML-

DSA-44. As in our work, the hashes are proven over a binary field, and the challenge sampling with ordered selection and offline memory checking, although they prove the latter in the binary field as well. The arithmetic modulo q is however emulated over the Goldilocks field, with range-checked quotients, and the two fields are linked by a MAC. They give a proof of possession of an ML-DSA-44 signature and an mdoc presentation with device binding in which both the issuer and the device use ML-DSA-44, at parameters targeting either classical adversaries in the ROM or quantum adversaries in the QROM. Their prover is fast, but their proofs are 2.3 to 6.5 times larger than ours, depending on the parameters.

2 Preliminaries

This work deals with zero-knowledge proofs of knowledge of ML-DSA signatures. In this section, we first recall the ML-DSA signature scheme, then give some background on proof systems together with a description of the proof system we rely on, and finally present the arithmetic tools used in our construction.

2.1 The ML-DSA Signature Scheme

ML-DSA is a lattice-based signature scheme standardized by NIST in FIPS 204 [Nat24]. It is parameterized by the prime $q = 2^{23} - 2^{13} + 1$, and most of its computation is performed over the field $\mathbb{F}_q$, more precisely over the polynomial ring $R_q = \mathbb{F}_q[X]/(X^n + 1)$. The remaining part of the computation operates on bits and bytes, mainly in the symmetric primitives, which are all based on Keccak. Table 2 summarizes the parameters of the three ML-DSA parameter sets.

Table 2: ML-DSA parameters [Nat24].

Parameter		ML-DSA-44	ML-DSA-65	ML-DSA-87
n	Coefficients per polynomial	256	256	256
q	Coefficient modulus	8 380 417	8 380 417	8 380 417
d	Bits dropped from $\mathbf{t}$ in the public key	13	13	13
(k, ℓ)	Matrix dimensions	$(4, 4)$	$(6, 5)$	$(8, 7)$
τ	Nonzero challenge coefficients	39	49	60
λ	Collision strength; $\widetilde{c}$ has 2λ bits	128	192	256
γ_1	Coefficient range of the response	2^{17}	2^{19}	2^{19}
γ_2	Low-order rounding range	$(q-1)/88$	$(q-1)/32$	$(q-1)/32$
β	Margin in the response-norm check ($\beta = \tau\eta$)	78	196	120
ω	Maximum hint weight	80	55	75

In what follows, Π denotes the ML-DSA parameter set under consideration.

Notation. Bold lowercase and uppercase symbols denote vectors and matrices of polynomials, respectively. For a vector of polynomials $\mathbf{v} \in R_q^m$, the norm $\|\mathbf{v}\|_\infty$ is the largest absolute value of its coefficients, each coefficient being represented in $\{-(q-1)/2, \ldots, (q-1)/2\}$, and $\mathrm{wt}(\mathbf{v})$ is the number of its nonzero coefficients. For a positive integer a, $\mathsf{bitlen}(a)$ denotes its bit length. We write $\mathbb{B} := \{0, \ldots, 255\}$ for the set of bytes.

NTT representation. For efficiency, ML-DSA relies on the number-theoretic transform NTT: some vectors and matrices of polynomials are directly represented in NTT form, which we indicate with a hat. As in the standard, $\circ$ denotes the (coefficient-wise) multiplication of polynomials in NTT form.

Hash functions and encodings. ML-DSA relies on both SHAKE128 and SHAKE256: SHAKE128 expands the public matrix from its seed, while SHAKE256 is used for all the other hashing steps, including the derivation of the challenge. Following the standard [Nat24], we write

$$H(S, L) = \mathsf{SHAKE256}(S, 8L), \qquad G(S, L) = \mathsf{SHAKE128}(S, 8L),$$

both of which take a bit string S and return a string of $8L$ bits. We denote concatenation by $\|$ and use the conversion, encoding, and decoding functions of the standard, including its bit-order conventions. Throughout, we identify a byte string with its bit string, obtained with the $\mathsf{BytesToBits}$ conversion of the standard, and leave this conversion implicit.

Key generation. An ML-DSA key pair is a Module-LWE instance together with its solution. The public key defines a matrix $\mathbf{A} \in R_q^{k\times\ell}$ and a vector $\mathbf{t} \in R_q^k$, and the secret key contains short vectors $\mathbf{s}_1 \in R_q^\ell$ and $\mathbf{s}_2 \in R_q^k$ such that $\mathbf{t} = \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$. For compactness, the serialized public key pk does not contain $\mathbf{A}$ and $\mathbf{t}$ themselves: it contains a seed ρ, from which the matrix is expanded directly in NTT form (denoted $\widehat{\mathbf{A}}$), and the high-order part $\mathbf{t}_1$ of $\mathbf{t}$, defined by $\mathbf{t} = \mathbf{t}_1 \cdot 2^d + \mathbf{t}_0$ with the d low-order bits of each coefficient in $\mathbf{t}_0$.

Signature verification. ML-DSA follows the Fiat–Shamir with aborts paradigm. A signature σ encodes a triple $(\widetilde{c}, \mathbf{z}, \mathbf{h})$. The *challenge seed* $\widetilde{c} \in \{0,1\}^{2\lambda}$ is a hash of the message and of the signer's commitment; it is expanded into the *challenge* $c \in R_q$, a polynomial with τ coefficients in $\{-1, 1\}$ and all other coefficients equal to zero. The *response* $\mathbf{z} \in R_q^\ell$ is a short vector, and the *hint* $\mathbf{h} \in R_2^k$ allows the verifier to recompute the signer commitment without knowing $\mathbf{t}_0$.

We now recall the ML-DSA verification algorithm: a ZK-PoK of a signature amounts to proving that this algorithm accepts. We consider pure ML-DSA, as specified in FIPS 204 [Nat24]. The message length ℓ_M and the context string ctx are public, with

$$M \in \{0,1\}^{\ell_M}, \qquad \mathsf{ctx} \in \{0,1\}^{8L_{\mathsf{ctx}}}, \qquad 0 \le L_{\mathsf{ctx}} \le 255,$$

where L_{ctx} is the length of ctx in bytes. The external verification algorithm first formats the message as

$$M' = \mathsf{IntegerToBits}(0, 8)\|\mathsf{IntegerToBits}(L_{\mathsf{ctx}}, 8)\|\mathsf{ctx}\|M, \tag{1}$$

and then calls the internal verification algorithm, which we recall in a slightly simplified form in Algorithm 8. We write $\mathsf{Verify}^{II}_{\mathrm{NIST}}(pk, M, \sigma; \mathsf{ctx})$ for the resulting verification predicate. The value $\mathbf{w}'_1$ output by $\mathsf{UseHint}$ is the *reconstructed signing commitment*, which is hashed together with the message to derive the challenge seed.

NIST Algorithm 8 $\mathsf{ML\text{-}DSA.Verify_internal}(pk, M', \sigma)$

Internal function to verify a signature σ for a formatted message M'.

Input: Public key $pk \in \{0,1\}^{256+256k(\mathsf{bitlen}(q-1)-d)}$ and message $M' \in \{0,1\}^*$.
Input: Signature $\sigma \in \{0,1\}^{2\lambda+256\ell(1+\mathsf{bitlen}(\gamma_1-1))+8(\omega+k)}$.
Output: Boolean

1: $(\rho, \mathbf{t}_1) \leftarrow \mathsf{pkDecode}(pk)$
2: $(\widetilde{c}, \mathbf{z}, \mathbf{h}) \leftarrow \mathsf{sigDecode}(\sigma)$
3: **if** $\mathbf{h} = \perp$ **then return** false
4: $\widehat{\mathbf{A}} \leftarrow \mathsf{ExpandA}(\rho)$
5: $tr \leftarrow H(pk, 64)$
6: $\mu \leftarrow H(tr\|M', 64)$
7: $c \in R_q \leftarrow \mathsf{SampleInBall}(\widetilde{c})$
8: $\mathbf{w}'_{\mathrm{Approx}} \leftarrow \mathrm{NTT}^{-1}\Big(\widehat{\mathbf{A}} \circ \mathrm{NTT}(\mathbf{z}) - \mathrm{NTT}(c) \circ \mathrm{NTT}(\mathbf{t}_1 \cdot 2^d)\Big)$
9: $\mathbf{w}'_1 \leftarrow \mathsf{UseHint}(\mathbf{h}, \mathbf{w}'_{\mathrm{Approx}})$ ▷ applied componentwise
10: $\widetilde{c}' \leftarrow H(\mu\|\mathsf{w1Encode}(\mathbf{w}'_1), \lambda/4)$
11: **return** $\big(\|\mathbf{z}\|_\infty < \gamma_1 - \beta\big) \wedge \big(\widetilde{c} = \widetilde{c}'\big)$

2.2 Proof Systems and SmallWood$^+$

Zero-knowledge proofs of knowledge. Let $\mathcal{R}$ be an NP relation, made of pairs (x, w) where x is a public statement and w a secret witness. A zero-knowledge proof of knowledge (ZK-PoK) [GMR89] for $\mathcal{R}$ is an interactive protocol in which a prover $P(x, w)$ convinces a verifier $V(x)$ that it knows a witness w such that $(x, w) \in \mathcal{R}$, without revealing anything else about w. It satisfies the following properties.

- *Completeness:* if $(x, w) \in \mathcal{R}$, the honest prover always convinces the verifier.
- *Knowledge soundness:* there exists an efficient extractor which, given access to any prover P^* that convinces $V(x)$ with probability greater than some ϵ, outputs a witness w such that $(x, w) \in \mathcal{R}$. The quantity ϵ is called the *knowledge error*.
- *Zero knowledge:* there exists an efficient simulator which, given only x, outputs transcripts that are indistinguishable from those of real executions between $P(x, w)$ and $V(x)$.

When knowledge soundness only holds against computationally bounded provers, e.g., provers making a bounded number of queries to a random oracle, the protocol is called an *argument* of knowledge. A public-coin protocol can be made non-interactive with the Fiat–Shamir transform [FS87], which derives the verifier's challenges by hashing the transcript. For multi-round protocols, the soundness of the resulting non-interactive argument is not implied by the soundness of the interactive protocol in general, but it is guaranteed by round-by-round (knowledge) soundness [CCH$^+$19].

SmallWood$^+$. SmallWood [FR26] is a hash-based zero-knowledge argument in the lineage of Ligero, which relies on a Merkle-tree-based polynomial commitment scheme with a degree-enforcing test. Since it only relies on hash functions, it requires no trusted setup and is plausibly post-quantum secure. It achieves the smallest proof sizes among hash-based arguments for witnesses of moderate size, from about 2^6 to 2^{16} field elements.

SmallWood$^+$ [FPR26] improves upon SmallWood by leveraging sum-check-based protocols, such as the GKR protocol, following the approach of Longfellow-ZK [Fas26]. The prover first commits to its witness together with a random mask; the prover and the verifier then run the sum-check-based protocol, in which every prover message is masked; finally, the prover shows in zero knowledge, with SmallWood-style commitment schemes, that the committed values pass the verifier's residual checks. Since the nonlinear work is delegated to the sum-check protocols, the commitment schemes only need to open linear functions or multilinear evaluations of the committed values, and, for layered circuits, only the inputs and outputs of the circuit need to be committed. SmallWood$^+$ further supports the commitment of binary values through a zero-knowledge variant of the ring-switching protocol, which packs several bits into a single element of a larger binary field. As a result, it establishes new state-of-the-art proof sizes for hash-based arguments, e.g., 18 KB and 63 KB to prove one and one hundred Keccak permutations, respectively. SmallWood$^+$ is round-by-round knowledge sound and zero knowledge, and can thus be made non-interactive with the Fiat–Shamir transform.

SmallWood$^+$ comes with two compilers: the first one relies on a single linear commitment scheme, while the second one additionally uses a multilinear polynomial commitment scheme for the witness. In this work, we use the first compiler, instantiated with the SmallWood-style linear commitment scheme of [FPR26]. SmallWood$^+$ is a commit-and-prove system: the values committed at the beginning of the protocol are fixed before the verifier sends any challenge. In our construction, the verifier thus samples the challenges of the randomized tests of Section 2.3 after the commitments, and the prover then proves, in zero knowledge, the resulting challenge-dependent constraints on the committed values.

2.3 Arithmetic Tools

In this section, we describe randomized tests that are useful to design zero-knowledge proofs. In all of them, we assume that some elements of $\mathbb{F}_q$ have been committed, and we want to check that they satisfy some property. The verifier then sends random challenges, and the prover shows that the committed values satisfy an arithmetic constraint that depends on these challenges. All committed values are fixed before the challenges are sampled.

Let $\mathbb{K}$ be a field extension of $\mathbb{F}_q$ of degree $r = [\mathbb{K} : \mathbb{F}_q]$. The challenges are sampled from $\mathbb{K}$, which is chosen large enough for the soundness errors of the tests to be negligible. We fix a public basis $(\theta_1, \ldots, \theta_r)$ of $\mathbb{K}$ over $\mathbb{F}_q$, so that each element of $\mathbb{K}$ is represented by r coordinates in $\mathbb{F}_q$; every constraint over $\mathbb{K}$ below is thus expressed as r constraints over $\mathbb{F}_q$ on these coordinates.

Tuple compression. Assume that we want to perform tests on tuples $\mathbf{z} \in \mathbb{F}_q^\nu$ with $\nu \leq r$. Instead of working on the tuples themselves, we can perform these tests on compressed versions of them. Each tuple $\mathbf{z} \in \mathbb{F}_q^\nu$ is compressed into

$$\psi(\mathbf{z}) = \sum_{h=1}^{\nu} \theta_h z_h \in V_\nu, \qquad V_\nu = \mathrm{span}_{\mathbb{F}_q}(\theta_1, \ldots, \theta_\nu), \tag{2}$$

using the first ν elements of the public basis of $\mathbb{K}$. For scalars ($\nu = 1$), we simply have $\psi(x) = \theta_1 x$. Since $\theta_1, \ldots, \theta_\nu$ are linearly independent over $\mathbb{F}_q$, the map ψ is an $\mathbb{F}_q$-linear bijection from $\mathbb{F}_q^\nu$ onto the subspace V_ν, which has q^ν elements. In particular, distinct tuples always have distinct compressions. In practice, this paper compresses at most triples, which requires $r \geq 3$; this is the case for the extensions we use.

Multiset Equality. Assume that L_x tuples $\mathbf{x}_1, \ldots, \mathbf{x}_{L_x} \in \mathbb{F}_q^\nu$ have been committed together with L_x selection bits $(\delta_i^x)_i \in \{0,1\}^{L_x}$, and similarly that L_y tuples $\mathbf{y}_1, \ldots, \mathbf{y}_{L_y} \in \mathbb{F}_q^\nu$ have been committed together with L_y selection bits $(\delta_j^y)_j \in \{0,1\}^{L_y}$. We assume that the selection bits are constrained to be Boolean (e.g., by $\delta(\delta - 1) = 0$). Our goal is to check that the multisets of selected tuples

$$\{\{\mathbf{x}_i \mid \delta_i^x = 1\}\} \qquad \text{and} \qquad \{\{\mathbf{y}_j \mid \delta_j^y = 1\}\}$$

are equal, i.e., that they contain the same tuples with the same multiplicities. This can be done with the product argument used in the shuffle argument of Bayer and Groth [BG12]. Although the original argument does not handle selection bits, it is easily adapted to support them. The verifier samples a challenge $\chi \xleftarrow{\$} \mathbb{K}$, and the prover shows that

$$\prod_{i=1}^{L_x} \big[1 + \delta_i^x \big(\chi + \psi(\mathbf{x}_i) - 1\big)\big] = \prod_{j=1}^{L_y} \big[1 + \delta_j^y \big(\chi + \psi(\mathbf{y}_j) - 1\big)\big]. \tag{3}$$

When a selection bit is zero, the corresponding factor equals one and leaves the product unchanged; when it is one, the factor equals $\chi + \psi(\mathbf{x}_i)$ (resp. $\chi + \psi(\mathbf{y}_j)$).

Seen as polynomials in χ, the two sides of (3) are products of monic linear factors, one per selected tuple. Since the compression is injective, distinct tuples give distinct factors. By unique factorization, the two polynomials are equal if and only if the selected multisets are equal. If they differ, their difference is a nonzero polynomial of degree at most $\max\{\sum_i \delta_i^x, \sum_j \delta_j^y\} \leq \max\{L_x, L_y\}$, which has at most that many roots in $\mathbb{K}$. Hence

$$\Pr_{\chi}[\text{accept}] \leq \frac{\max\{L_x, L_y\}}{|\mathbb{K}|}. \tag{4}$$

Table Lookups. Let $\nu < r$ and let $\mathcal{T} \subseteq \mathbb{F}_q^\nu$ be a nonempty public table. Assume that $m < q$ lookup queries $\mathbf{Q}_1, \ldots, \mathbf{Q}_m \in \mathbb{F}_q^\nu$ have been committed, together with a multiplicity $m_{\mathbf{u}} \in \mathbb{F}_q$ for each $\mathbf{u} \in \mathcal{T}$. Our goal is to check that all the queries belong to $\mathcal{T}$, and that each $\mathbf{u} \in \mathcal{T}$ appears exactly $m_{\mathbf{u}}$ times among them.

LogUp [Hab22] performs this check by means of a rational identity on the compressed tuples (2). The verifier samples $\alpha \xleftarrow{\$} \mathbb{K} \setminus V_\nu$, and the prover shows that

$$\sum_{j=1}^{m} \frac{1}{\alpha + \psi(\mathbf{Q}_j)} = \sum_{\mathbf{u} \in \mathcal{T}} \frac{m_{\mathbf{u}}}{\alpha + \psi(\mathbf{u})}. \tag{5}$$

All denominators are nonzero since the compressions lie in the subspace V_ν and $\alpha \notin V_\nu$. If all the queries belong to the table and the multiplicities are their actual numbers of occurrences, the identity holds for every challenge. Conversely, consider both sides as rational functions in α. Since the compression is injective, a query outside the table leaves on the left-hand side a pole that does not appear on the right-hand side, and whose coefficient is its number of occurrences, which is nonzero in $\mathbb{F}_q$ since it lies between 1 and $m < q$. Similarly, a wrong multiplicity leaves a nonzero coefficient at the corresponding

pole. Thus, any incorrect assignment gives a nonzero rational function. Clearing denominators yields a nonzero polynomial of degree at most $m + |\mathcal{T}| - 1$, whose roots are the only challenges for which the test passes. Hence

$$\Pr_{\alpha}[\text{accept}] \leq \frac{m + |\mathcal{T}| - 1}{|\mathbb{K}| - q^{\nu}}. \tag{6}$$

Range checks. LogUp can be used to perform range checks. With the interval table $\mathcal{T} = \{0, \ldots, T\}$ for some $T < q$, checking that $x \in \mathbb{F}_q$ belongs to the table is equivalent to checking that $0 \leq x \leq T$, where x is identified with its representative in $\{0, \ldots, q-1\}$. We write $\mathsf{Range}(x)$ for the property $x \in \mathcal{T}$.

The same table can be used to check smaller ranges. If $2T < q$ and $0 \leq B \leq T$, then

$$\mathsf{Range}(x) \wedge \mathsf{Range}(B - x) \quad \Longleftrightarrow \quad 0 \leq x \leq B. \tag{7}$$

Indeed, if both x and $B - x$ have representatives in $\{0, \ldots, T\}$, the sum of these representatives is at most $2T < q$ and is congruent to B modulo q, hence it equals B as an integer. Although this costs two lookups instead of one, it has the significant advantage that a *single* table serves range checks with many different bounds, as long as they do not exceed T.

Fractions as pairs. To avoid inversions, a common practice is to represent a fraction N/D, with $D \neq 0$, by the pair (N, D). Two fractions are then added with the law

$$(N_1, D_1) + (N_2, D_2) = (N_1 D_2 + N_2 D_1,\ D_1 D_2),$$

which only involves multiplications. This representation is convenient to express the LogUp equation (5) as an arithmetic circuit: it is equivalent to

$$\sum_{j=1}^{m} \big(1, \alpha + \psi(\mathbf{Q}_j)\big) + \sum_{\mathbf{u} \in \mathcal{T}} \big(-m_{\mathbf{u}}, \alpha + \psi(\mathbf{u})\big) = (0, D),$$

where D is the product of all denominators, which is nonzero since $\alpha \notin V_\nu$. Only the numerator thus needs to be checked to be zero.

Ordered Selection. Let $m \leq L < q$. Assume that two lists $\mathbf{v} = (v_1, \ldots, v_L) \in \mathbb{F}_q^L$ and $\mathbf{u} = (u_1, \ldots, u_m) \in \mathbb{F}_q^m$ have been committed, together with selection bits $\boldsymbol{\delta} = (\delta_1, \ldots, \delta_L) \in \{0,1\}^L$. Our goal is to check that $\mathbf{u}$ is exactly the list of the entries of $\mathbf{v}$ whose selection bit is set, *in the same order*:

$$(u_j)_{1 \leq j \leq m} = (v_i)_{1 \leq i \leq L,\ \delta_i = 1}.$$

Let $c_i := \sum_{k=1}^{i} \delta_k$ be the number of selected entries among the first i ones (including the i-th). The property above holds if and only if $\sum_i \delta_i = m$ and $u_{c_i} = v_i$ for every i such that $\delta_i = 1$. It can therefore be checked by the linear constraint $\sum_i \delta_i = m$ together with the multiset equality

$$\{\{(c_i, v_i) \mid \delta_i = 1\}\}_{1 \leq i \leq L} = \{\{(j, u_j)\}\}_{1 \leq j \leq m}. \tag{8}$$

Indeed, the selected indices c_i are exactly $1, \ldots, m$, so each side contains exactly one pair with first coordinate j for every $1 \leq j \leq m$, and the equality forces this pair to be (j, u_j) on both sides. Since $c_i \leq L < q$, the integers c_i are faithfully represented in $\mathbb{F}_q$.

We write $\mathsf{Select}(\mathbf{v}, \boldsymbol{\delta}; \mathbf{u}) = 1$ when the test passes, i.e., when $\sum_i \delta_i = m$ and (3) holds for the pairs of (8). Since $\sum_i \delta_i = m$ is enforced exactly, both sides of (3) have total degree m, and the analysis of the multiset equality shows that an incorrect selection passes the test with probability

$$\Pr_{\chi}[\mathsf{Select}(\mathbf{v}, \boldsymbol{\delta}; \mathbf{u}) = 1] \leq \frac{m}{|\mathbb{K}|}.$$

Memory Consistency. We model a memory as a set of *versions*, i.e., triples (a, v, t) made of an address a, a value v, and the time t at which this version was created. Addresses belong to a public set $\mathcal{A} \subseteq \{0, \ldots, q-1\}$ and values lie in $\mathbb{F}_q$. A memory history $(\mathcal{O}, \mathcal{F}, \mathcal{U})$ consists of

- an initial state $\mathcal{O} := \{(a, v_a^{\text{init}}, 0)\}_{a \in \mathcal{A}}$, which lists the initial value of every address;
- a final state $\mathcal{F} := \{(a, v_a^{\text{final}}, t_a^{\text{final}})\}_{a \in \mathcal{A}}$, which lists the final value of every address together with the time of its last update;
- a list of updates $\mathcal{U} := \{(s, a_s, v_s^{\text{old}}, t_s^{\text{old}}, v_s^{\text{new}})\}_{1 \le s \le |\mathcal{U}|}$, where the s-th update replaces the version $(a_s, v_s^{\text{old}}, t_s^{\text{old}})$ by the version $(a_s, v_s^{\text{new}}, s)$:

$$\underbrace{(a_s, v_s^{\text{old}}, t_s^{\text{old}})}_{\text{deleted}} \quad \longrightarrow \quad \underbrace{(a_s, v_s^{\text{new}}, s)}_{\text{created}}.$$

We denote by $\mathcal{W} := \{(a_s, v_s^{\text{new}}, s)\}_{1 \le s \le |\mathcal{U}|}$ the multiset of created versions and by $\mathcal{D} := \{(a_s, v_s^{\text{old}}, t_s^{\text{old}})\}_{1 \le s \le |\mathcal{U}|}$ the multiset of deleted versions. We assume $|\mathcal{U}| < q$.

We want to check that the memory history is consistent, namely that each update reads the current value at its address, and that applying the updates in order to $\mathcal{O}$ results in $\mathcal{F}$. Adapting the offline memory checking approach based on multiset hashing [CDvD+03], we first check the conservation law

$$\mathcal{O} \uplus \mathcal{W} = \mathcal{D} \uplus \mathcal{F}, \tag{9}$$

where $\uplus$ denotes the multiset union. Since all the versions of $\mathcal{O} \uplus \mathcal{W}$ have distinct address-time pairs, this means that every initial or created version is either deleted exactly once or remains in the final state, and that nothing else is deleted. Conservation alone, however, ignores the order of the updates: an update could delete a version created by a later update. To prevent this, we additionally require each update to satisfy

$$0 \le t_s^{\text{old}} < s \qquad \text{for all } 1 \le s \le |\mathcal{U}|. \tag{10}$$

Let us fix an address and consider its updates in order. The first one can only delete the initial version, since no other version at this address was created before. Each subsequent update must delete the version created by the preceding update at this address: older versions have already been deleted, and (10) excludes the future ones. The only version that is never deleted is the last one, which must therefore be the final one. This argument checks the whole history without reconstructing the intermediate memory states. In particular, an update at an address outside $\mathcal{A}$ cannot satisfy these constraints.

The conservation law is checked with the multiset equality test, and the time constraints (10) with two range checks per update, using (7) with a single lookup table $\mathcal{T} = \{0, \ldots, T\}$ such that $|\mathcal{U}| \le T$ and $2T < q$. We write $\mathsf{MemCheck}(\mathcal{O}, \mathcal{W}; \mathcal{D}, \mathcal{F}) = 1$ when the memory consistency test passes. By a union bound, an inconsistent memory history passes the test with probability

$$\Pr[\mathsf{MemCheck}(\mathcal{O}, \mathcal{W}; \mathcal{D}, \mathcal{F}) = 1] \le \underbrace{\frac{|\mathcal{O}| + |\mathcal{U}|}{|\mathbb{K}|}}_{\text{multiset equality}} + \underbrace{\frac{2|\mathcal{U}| + |\mathcal{T}| - 1}{|\mathbb{K}| - q}}_{\text{lookup}},$$

where the probability is taken over the challenges χ of the multiset equality and α of the lookup.

3 Proving Knowledge of an ML-DSA Signature: A Two-Field Protocol

This section presents our ZK-PoK at a high level. We first define the proven relation (Section 3.1). We then explain how we split it between a binary circuit over $\mathbb{F}_2$ and a prime-field circuit over $\mathbb{F}_q$, and specify what each circuit must check (Section 3.2). We next introduce the test that keeps the values shared by the two circuits consistent (Section 3.3). Finally, we describe the overall protocol, in which both circuits are proven with $\mathsf{SmallWood}^+$ (Section 3.4), specify the two circuits (Section 3.5), and state the security of the protocol (Section 3.6). The two circuits will then be detailed in Sections 4 and 5.

3.1 Proven Relation

Our goal is a ZK-PoK of an ML-DSA signature, possibly with a hidden verification key or a hidden message, depending on the application: in a blind signature, the verification key is public and the message is hidden, whereas for anonymous credentials with device binding, the device's verification key must remain hidden while the signed challenge is public. A hidden verification key or message is only meaningful when it is bound by a further relation proven jointly on the same committed values, e.g., when some attributes are proven on the message or when the verification key is part of a certificate signed under a public verification key.

To cover all cases at once, we describe the most demanding setting, in which the verification key, the message, and the signature are all private. The parameter set Π, the context string ctx, and the message length ℓ_M are public. Settings in which the verification key or the message is public are special cases of this one, obtained by moving these values into the statement; in these cases, some parts of the verification, such as the expansion of the matrix or the computation of tr, no longer need to be proven.

Semantic witness. The semantic witness w_{sem} consists of the key, the message, and the signature components:

$$w_{\text{sem}} = (pk, M, \widetilde{c}, \mathbf{z}, \mathbf{h}) \in \mathcal{D}_{\text{sem}} := \{0,1\}^{\ell_{pk}} \times \{0,1\}^{\ell_M} \times \{0,1\}^{2\lambda} \times R_q^{\ell} \times R_q^{k},$$

where $\ell_{pk} = 256 + 256k(\mathsf{bitlen}(q-1) - d)$ is the bit length of the encoded verification key. We keep pk in its encoded form, since the verification hashes it. For the signature, we use its decoded components $(\widetilde{c}, \mathbf{z}, \mathbf{h}) = \mathsf{sigDecode}(\sigma)$ rather than its encoding σ. This is without loss of generality: the verification only depends on σ through this triple, and the possible outputs of $\mathsf{sigDecode}$ other than $\perp$ are exactly the triples in which $\mathbf{z}$ has all its coefficients in $\{-\gamma_1 + 1, \ldots, \gamma_1\}$ and $\mathbf{h}$ has all its coefficients in $\{0,1\}$, i.e., $\mathbf{h} \in R_2^k$, with $\mathrm{wt}(\mathbf{h}) \leq \omega$. These triples form the domain $\mathcal{D}_{\text{sig}}$ of $\mathsf{sigEncode}$, which maps each of them to a signature that decodes back to it. Proving knowledge of a valid signature is thus equivalent to proving knowledge of a triple in this domain that passes the verification checks. We do not restrict the witness to this domain, since a prover can commit to arbitrary $\mathbf{z}$ and $\mathbf{h}$: membership in $\mathcal{D}_{\text{sig}}$ is instead a condition of the relation.

Bounded sampling. The verification involves two samplers, which read the output of an extendable-output function (XOF) and rely on rejection sampling:

- $\mathsf{ExpandA}$ samples each of the $k\ell$ entries of $\widehat{\mathbf{A}}$ with $\mathsf{RejNTTPoly}$, which reads the SHAKE128 output seeded by ρ (and the position of the entry) by chunks of three bytes. Each chunk, called a *candidate*, yields a 23-bit integer, which is accepted if it is smaller than q and rejected otherwise; the entry consists of the first n accepted candidates.
- $\mathsf{SampleInBall}$ expands the challenge seed $\widetilde{c}$ into the challenge c. It reads the SHAKE256 output seeded by $\widetilde{c}$: its first eight bytes provide the signs of the τ nonzero coefficients of c. The following bytes are *candidates* for the positions of these coefficients: for each of the τ positions to fill, candidates are read until one falls in the allowed range.

The number of rejected candidates, and thus the amount of XOF output consumed, is not bounded a priori, whereas a circuit of fixed size requires such a bound. FIPS 204 [Nat24, Appendix C] allows implementations to bound the number of candidates, and specifies the following minimal budgets, which we use throughout:

$$B_A = 298 \text{ candidates per matrix entry}, \qquad B_c = 213 \text{ candidates for the challenge}.$$

They correspond to $3B_A = 894$ bytes of SHAKE128 output per matrix entry, and to $8 + B_c = 221$ bytes of SHAKE256 output for the challenge. If the XOF output is uniformly random, each call exceeds its budget with probability at most 2^{-256} [Nat24, Appendix C]. In comparison, $n = 256$ accepted candidates are needed per matrix entry, and $\tau \leq 60$ for the challenge. We write $\mathsf{ExpandA}_{B_A}$ and $\mathsf{SampleInBall}_{B_c}$ for the bounded samplers, which return the output of the ML-DSA samplers when these terminate within their budgets, and $\perp$ otherwise.

Remark 1. The budget $B_c = 213$ of FIPS 204 is dimensioned for the largest challenge weight, $\tau = 60$ (ML-DSA-87). For ML-DSA-44 ($\tau = 39$) and ML-DSA-65 ($\tau = 49$), an exact computation shows that

148 and 178 candidates, respectively, already give a failure probability of at most 2^{-256} per call. These smaller budgets would slightly reduce the cost of proving the challenge sampling, since each candidate is a byte shared by the two circuits and subject to an acceptance check, but they would not change the number of Keccak permutations: the SHAKE256 output spans two blocks in all cases. We keep the budgets of the standard, which specifies them as minimal values.

Proven relation. Let $x = (\Pi, \mathsf{ctx}, \ell_M, B_A, B_c)$ be the statement. In the default setting where the verification key, the message, and the signature are all private, it only consists of public parameters. With $(\rho, \mathbf{t}_1) = \mathsf{pkDecode}(pk)$ and $\sigma = \mathsf{sigEncode}(\widetilde{c}, \mathbf{z}, \mathbf{h})$ when $(\widetilde{c}, \mathbf{z}, \mathbf{h}) \in \mathcal{D}_{\mathrm{sig}}$, the relation is defined, for every $w_{\mathrm{sem}} \in \mathcal{D}_{\mathrm{sem}}$, by

$$(x, w_{\mathrm{sem}}) \in \mathcal{R}_{\text{ML-DSA}} \iff \begin{cases} (\widetilde{c}, \mathbf{z}, \mathbf{h}) \in \mathcal{D}_{\mathrm{sig}}, \\ \mathsf{Verify}^{\Pi}_{\mathrm{NIST}}(pk, M, \sigma; \mathsf{ctx}) = 1, \\ \mathsf{ExpandA}_{B_A}(\rho) \neq \bot, \\ \mathsf{SampleInBall}_{B_c}(\widetilde{c}) \neq \bot. \end{cases} \tag{11}$$

The first condition states that the signature components are well formed. The last two conditions exclude the valid signatures for which a sampler exceeds its budget. For honestly generated keys and signatures, ρ and $\widetilde{c}$ are hash outputs, so that this happens with probability at most $(k\ell + 1) \cdot 2^{-256}$ when SHAKE is modeled as a random oracle; this is the completeness error induced by the bounded samplers. When the verification key or the message is public, the corresponding value pk or M is moved from the witness w_{sem} to the statement x; the conditions of (11) are unchanged, except that the condition $\mathsf{ExpandA}_{B_A}(\rho) \neq \bot$ is dropped when the verification key is public: the verifier then expands the matrix itself, outside the circuits, so that the number of candidates read by ExpandA need not be bounded.

3.2 Splitting ML-DSA Verification

ML-DSA verification interleaves two kinds of computation. On the one hand, it evaluates SHAKE several times and manipulates binary encodings, which are naturally expressed over $\mathbb{F}_2$. On the other hand, it performs polynomial arithmetic and norm checks modulo q, which are naturally expressed over $\mathbb{F}_q$. Arithmetizing the whole verification over a single field would force one of these two parts into an unnatural representation: bit decompositions of every Keccak state over $\mathbb{F}_q$, or modular arithmetic emulated with binary adders over $\mathbb{F}_2$.

To prove the relation $\mathcal{R}_{\text{ML-DSA}}$, we therefore split the verification algorithm into a binary part, which covers the hash computations and the binary encodings, and a prime-field part, which covers the polynomial computations and the bounds. The two parts share a few intermediate values, and each of them is checked by a circuit over its own field, $C^{(2)}$ over $\mathbb{F}_2$ and $C^{(q)}$ over $\mathbb{F}_q$, specified in Section 3.5. We explain in turn how the samplers, the witness, and the relation are split.

Splitting the Samplers. Each sampler is split into a Stream routine, evaluated in $C^{(2)}$, which evaluates SHAKE and produces the candidates from the output stream, and a Build routine, evaluated in $C^{(q)}$, which builds the sampled polynomial from these candidates.

Matrix sampling. ExpandA.Stream produces the SHAKE128 candidates derived from ρ together with their validity bits, and ExpandA.Build takes the first n valid candidates of each stream, in order, as the NTT coefficients of the corresponding matrix entry. Let $S_A \in \{0,1\}^{B_A \cdot k \cdot \ell \cdot 24}$ denote the stream defined by:

$$S_A = (\mathbf{v}^A, \mathbf{e}^A) = \mathsf{ExpandA.Stream}_{B_A}(\rho), \qquad \widehat{\mathbf{A}} = \mathsf{ExpandA.Build}(S_A),$$

where, for each matrix entry $f = (r, s)$, with $0 \leq r < k$ and $0 \leq s < \ell$, and each $1 \leq i \leq B_A$, $\mathbf{v}^A$ contains the i-th candidate $v^A_{f,i}$ of the corresponding stream, as a 23-bit integer, and $\mathbf{e}^A$ contains its validity bit $e^A_{f,i}$, equal to 1 if and only if $v^A_{f,i} < q$.

Challenge sampling. SampleInBall.Stream produces the sign bits and the candidate bytes from the SHAKE256 stream seeded by $\widetilde{c}$, and SampleInBall.Build reconstructs from them the polynomial returned by the ML-DSA sampler. Let $S_c \in \{0,1\}^{B_c \cdot 8 + \tau}$ denote the stream defined by:

$$S_c = (\mathbf{j}, \mathbf{s}) = \mathsf{SampleInBall.Stream}_{B_c}(\widetilde{c}), \qquad c = \mathsf{SampleInBall.Build}(S_c),$$

where $\mathbf{j} = (j_1, \ldots, j_{B_c})$ contains the B_c candidate bytes, and $\mathbf{s} = (s_1, \ldots, s_\tau)$ contains the sign bits of the τ nonzero coefficients. Unlike the matrix stream, this stream carries no validity bits. Indeed, a candidate j_i is accepted if it does not exceed the position currently being filled, which depends on the number of candidates accepted before it; the acceptance of a candidate is thus not a function of the candidate alone. These decisions are therefore made in $C^{(q)}$, which keeps track of the number of accepted candidates (Section 5.3).

In both cases, composing Stream and Build gives the bounded sampler of Section 3.1. Their implementations are given in Sections 4.3, 5.2, and 5.3.

Splitting the Witness. The semantic witness w_{sem} is split accordingly, and it is completed with the intermediate values shared by the two parts. This gives three components:

- the input witness $w_{\text{in}}^{(2)} = (pk, M, \widetilde{c})$, which only enters $C^{(2)}$;
- the input witness $w_{\text{in}}^{(q)} = (\mathbf{z}, \mathbf{h})$, which only enters $C^{(q)}$;
- the cross-field witness $w_{\text{cross}} = (\mathbf{t}_1, S_A, S_c, \mathbf{w}_1')$, which enters both circuits.

We denote by $\mathcal{D}_{\text{cross}}$ the domain in which the cross-field witness lies, which is:

$$\mathcal{D}_{\text{cross}} = \underbrace{\{0, \ldots, 2^{10}-1\}^{nk}}_{\mathbf{t}_1} \times \underbrace{\{0,1\}^{24k\ell B_A}}_{S_A} \times \underbrace{\{0,1\}^{8B_c+\tau}}_{S_c} \times \underbrace{\{0, \ldots, 2^{b_w}-1\}^{nk}}_{\mathbf{w}_1'},$$

where $b_w = \mathsf{bitlen}(m_\gamma - 1)$ and $m_\gamma = (q-1)/(2\gamma_2)$.

With this split, the binary circuit $C^{(2)}$ recomputes the key coefficients $\mathbf{t}_1$ and the sampler streams S_A, S_c from $w_{\text{in}}^{(2)}$, and checks that they match the values provided in w_{cross}. Similarly, $C^{(q)}$ recomputes the reconstructed signing commitment $\mathbf{w}_1'$ from $w_{\text{in}}^{(q)}$ and $(\mathbf{t}_1, S_A, S_c)$, and checks that it matches the value provided in w_{cross}. The reconstructed signing commitment $\mathbf{w}_1'$ provided in w_{cross} is in turn the input of the final hash in $C^{(2)}$. Figure 1 summarizes, for each component of the cross-field witness, which circuit checks it and which circuit uses it. Each circuit takes as input a copy of w_{cross} in its own field, as described in Section 3.5.

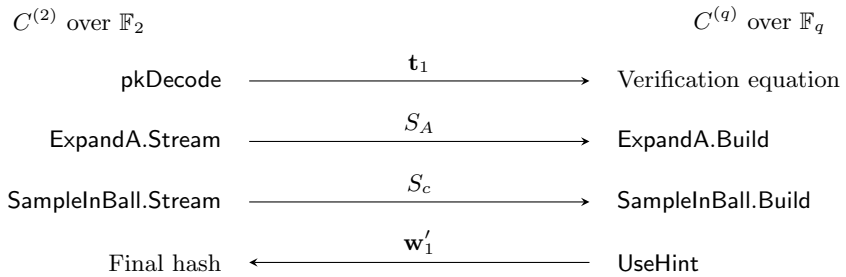

Fig. 1: Cross-field witness w_{cross}. Both circuits receive it as part of the witness: each arrow goes from the circuit that recomputes and checks a value to the circuit that uses it as an input.

Remark 2. When the verification key is public, $\mathbf{t}_1$ and $\widehat{\mathbf{A}}$ are public constants computed by the verifier, so that $\mathbf{t}_1$ and S_A are no longer shared: the cross-field witness reduces to $w_{\text{cross}} = (S_c, \mathbf{w}_1')$. A public message does not change the cross-field witness.

Splitting the Relation. Relations $\mathcal{R}^{(2)}$ and $\mathcal{R}^{(q)}$ below take the cross-field witness $w_{\text{cross}} \in \mathcal{D}_{\text{cross}}$ as input. They follow Algorithm 8: assignments ($\leftarrow$) define intermediate values, each relation holds if and only if all its checks pass, and the comments give the corresponding lines of Algorithm 8. As before, M' is the formatted message of (1).

Relation $\mathcal{R}^{(2)}\big(w_{\text{in}}^{(2)}, w_{\text{cross}}\big)$, with $w_{\text{in}}^{(2)} = (pk, M, \widetilde{c})$ and $w_{\text{cross}} = (\mathbf{t}_1, S_A, S_c, \mathbf{w}_1')$	
$(\rho, \mathbf{t}_1^*) \leftarrow \mathsf{pkDecode}(pk)$; **check** $\mathbf{t}_1^* = \mathbf{t}_1$	▷ line 1
check $S_A = \mathsf{ExpandA.Stream}_{B_A}(\rho)$	▷ line 4
$tr \leftarrow H(pk, 64)$; $\mu \leftarrow H(tr \| M', 64)$	▷ lines 5–6
check $S_c = \mathsf{SampleInBall.Stream}_{B_c}(\widetilde{c})$	▷ line 7
check $\widetilde{c} = H(\mu \| \mathsf{w1Encode}(\mathbf{w}_1'), \lambda/4)$	▷ lines 10–11

Relation $\mathcal{R}^{(q)}\big(w_{\text{in}}^{(q)}, w_{\text{cross}}\big)$, with $w_{\text{in}}^{(q)} = (\mathbf{z}, \mathbf{h})$ and $w_{\text{cross}} = (\mathbf{t}_1, S_A, S_c, \mathbf{w}_1')$	
check $\mathbf{h} \in R_2^k$ and $\mathrm{wt}(\mathbf{h}) \le \omega$	▷ lines 2–3
$\widehat{\mathbf{A}} \leftarrow \mathsf{ExpandA.Build}(S_A)$; **check** $\widehat{\mathbf{A}} \neq \perp$	▷ line 4
$c \leftarrow \mathsf{SampleInBall.Build}(S_c)$; **check** $c \neq \perp$	▷ line 7
$\mathbf{w}'_{\text{Approx}} \leftarrow \mathrm{NTT}^{-1}\big(\widehat{\mathbf{A}} \circ \mathrm{NTT}(\mathbf{z}) - \mathrm{NTT}(c) \circ \mathrm{NTT}(2^d \mathbf{t}_1)\big)$	▷ line 8
check $\mathbf{w}_1' = \mathsf{UseHint}(\mathbf{h}, \mathbf{w}'_{\text{Approx}})$	▷ line 9
check $\|\mathbf{z}\|_\infty < \gamma_1 - \beta$	▷ line 11

The function $\mathsf{UseHint}$ acts coefficientwise, and the bound on the response uses centered representatives. The check on $\mathbf{h}$ and the bound on $\mathbf{z}$ together imply the first condition of (11), namely $(\widetilde{c}, \mathbf{z}, \mathbf{h}) \in \mathcal{D}_{\text{sig}}$, since $\gamma_1 - \beta < \gamma_1$.

Lemma 1 (Split Verification). *For every semantic witness* $w_{\text{sem}} = (w_{\text{in}}^{(2)}, w_{\text{in}}^{(q)}) \in \mathcal{D}_{\text{sem}}$*,*

$$(x, w_{\text{sem}}) \in \mathcal{R}_{\text{ML-DSA}} \iff \exists\, w_{\text{cross}} \in \mathcal{D}_{\text{cross}} :\ \mathcal{R}^{(2)}\big(w_{\text{in}}^{(2)}, w_{\text{cross}}\big) \wedge \mathcal{R}^{(q)}\big(w_{\text{in}}^{(q)}, w_{\text{cross}}\big).$$

Proof. Assume that both relations hold for some w_{cross}. The checks of $\mathcal{R}^{(q)}$ on $\mathbf{h}$ and $\mathbf{z}$ give $(\widetilde{c}, \mathbf{z}, \mathbf{h}) \in \mathcal{D}_{\text{sig}}$. Merging the two relations line by line then gives an execution of Algorithm 8 on (pk, M', σ), with $\sigma = \mathsf{sigEncode}(\widetilde{c}, \mathbf{z}, \mathbf{h})$, in which both samplers terminate within their budgets and which returns true. Hence $(x, w_{\text{sem}}) \in \mathcal{R}_{\text{ML-DSA}}$. Conversely, the accepting execution of the bounded verification defines $\mathbf{t}_1$, the two streams, and $\mathbf{w}_1'$, and the resulting w_{cross} satisfies both relations; in particular, the check on $\mathbf{h}$ holds since $(\widetilde{c}, \mathbf{z}, \mathbf{h}) \in \mathcal{D}_{\text{sig}}$.

3.3 Cross-Field Consistency

The relations $\mathcal{R}^{(2)}$ and $\mathcal{R}^{(q)}$ must hold for the same cross-field witness. However, the two circuits are proven separately, and each of them receives its own copy of w_{cross}, represented in its own field (Section 3.5). We therefore need to check that the two copies represent the same cross-field witness. For this purpose, we follow and improve the approach of Frigo and shelat [Fas26]. The principle is that both circuits compute a *tag* of their copy, under randomness that is fixed only after the copies are committed, and that both tags must be equal. In their construction, the tag is an information-theoretic MAC $x \mapsto ax + b$ over $\mathbb{F}_{2^{128}}$, under a private key jointly sampled by the prover and the verifier. Instead of a MAC under a private key, our tag is a public random linear map, chosen by the verifier, combined with a private mask committed by the prover. This is sufficient to ensure zero-knowledge and soundness of the cross-field consistency check, while saving the multiplications by a private key in both circuits: the tag is a linear function of the committed values over the field in which it is defined.

The tag has to be computed in both fields: it is native in one circuit and emulated in the other. A first option is a tag over $\mathbb{F}_q$, namely random linear combinations of the words of w_{cross}. It is free in $C^{(q)}$, but $C^{(2)}$ would have to emulate arithmetic modulo q with binary adders. The second option is a tag over $\mathbb{F}_2$, namely random parities of the bits of w_{cross}. It is linear, and thus essentially free, in $C^{(2)}$, while $C^{(q)}$ has to recover these bits from its own representation of w_{cross} and to express their parities over $\mathbb{F}_q$ (Section 5.8). We choose the second option. Accordingly, the tag is computed on a binary encoding of the

cross-field witness. Let $\mathsf{Enc}\colon \mathcal{D}_{\text{cross}} \to \mathbb{F}_2^{N_{\text{cross}}}$ be the bijection that concatenates the binary encodings of the components of w_{cross}, with $N_{\text{cross}} = 10nk+24k\ell B_A+8B_c+\tau+nkb_w$. Concretely, Enc concatenates the groups $(\mathbf{t}_1, \mathbf{v}^A, \mathbf{e}^A, \mathbf{j}, \mathbf{s}, \mathbf{w}'_1)$ in this order, with matrix row and column order, then candidate or coefficient order within each group; key coefficients use 10 bits, matrix candidates 23 bits, challenge candidates 8 bits, and high parts b_w bits, while validity bits and signs are single bits, and the least significant bit of each word comes first. We write $u_{\text{cross}} = \mathsf{Enc}(w_{\text{cross}})$ for the resulting vector.

A tag over $\mathbb{F}_q$ would avoid committing to the bit decompositions of the shared values in $C^{(q)}$, but evaluating it in $C^{(2)}$ would require costly adder circuits to emulate arithmetic modulo q. We instead define the tag over $\mathbb{F}_2$: its evaluation is linear in $C^{(2)}$, and its verification in $C^{(q)}$ reduces to affine equations (Section 5.8).

Let $u \in \mathbb{F}_2^{N_{\text{cross}}}$ be a committed vector, and let $\kappa \in \mathbb{F}_2^{N_{\text{tag}}}$ be a mask committed together with u, where N_{tag} denotes the length of the tag. After the commitment, the verifier samples a matrix $R \xleftarrow{\$} \mathbb{F}_2^{N_{\text{tag}} \times N_{\text{cross}}}$, and the prover reveals the tag

$$\mathsf{tag}_R(u;\kappa) = \kappa + R\,u \in \mathbb{F}_2^{N_{\text{tag}}}. \tag{12}$$

Lemma 2 (Consistency Test). *Let $u, u' \in \mathbb{F}_2^{N_{\text{cross}}}$ and $\kappa, \kappa' \in \mathbb{F}_2^{N_{\text{tag}}}$ be fixed before R is sampled.*

- Completeness: *if $u = u'$ and $\kappa = \kappa'$, then $\mathsf{tag}_R(u;\kappa) = \mathsf{tag}_R(u';\kappa')$ for every R.*
- Soundness: *if $u \neq u'$, then $\Pr_R\big[\mathsf{tag}_R(u;\kappa) = \mathsf{tag}_R(u';\kappa')\big] = 2^{-N_{\text{tag}}}$.*
- Simulatability: *if κ is uniform and independent of u and R, then $\mathsf{tag}_R(u;\kappa)$ is uniform and independent of (u, R).*

Proof. Completeness is immediate. For soundness, the two tags are equal if and only if $R\,(u+u') = \kappa+\kappa'$. Since $u + u' \neq 0$ and the rows of R are independent and uniform, $R\,(u + u')$ is uniform over $\mathbb{F}_2^{N_{\text{tag}}}$, while $\kappa + \kappa'$ is fixed. Simulatability follows from the uniformity of the one-time pad κ.

We apply the test of Lemma 2 to the two copies of w_{cross}. Let $u^{(2)}_{\text{cross}}$ and $u^{(q)}_{\text{cross}}$ denote the vectors of $\mathbb{F}_2^{N_{\text{cross}}}$ encoded by the copies given to $C^{(2)}$ and $C^{(q)}$, whose formats are specified in Section 3.5; for an honest prover, both equal u_{cross}. Each circuit receives its own mask as input and checks its own side of:

$$\mathsf{tag}_R\big(u^{(2)}_{\text{cross}};\kappa^{(2)}\big) \;=\; y \;=\; \mathsf{tag}_R\big(u^{(q)}_{\text{cross}};\kappa^{(q)}\big) \tag{13}$$

where $y \in \mathbb{F}_2^{N_{\text{tag}}}$ is the claimed public tag value sent by the prover. The honest prover samples a fresh uniform mask in each execution and uses it in both circuits, $\kappa^{(2)} = \kappa^{(q)}$. By Lemma 2, honest copies always pass the test, two copies encoding distinct vectors pass it with probability $2^{-N_{\text{tag}}}$, and the public tag y can be simulated thanks to the randomness of the masks and the zero-knowledge property of the underlying commitment scheme.

Each side of (13) is thus integrated into one circuit: the binary side into $C^{(2)}$, and the prime-field side into $C^{(q)}$. While computing the tag is trivial on the binary side, the computation on the prime-field side is more subtle: $C^{(q)}$ holds its mask $\kappa^{(q)}$ and its copy of w_{cross} as elements of $\mathbb{F}_q$, so that it must constrain them to encode bits and express parities over $\mathbb{F}_q$, which requires auxiliary values in addition to the mask. This is detailed in Section 5.8.

3.4 Overall Protocol

Our protocol uses $\mathsf{SmallWood}^+$ as a commit-and-prove system (Section 2.2): the prover first commits to a vector of field elements, and later proves that the committed values satisfy a circuit, which may depend on public values fixed after the commitment. Such a system consists of a *commit protocol* and of a *prove protocol*, both interactive, and the public values form the *statement* of the prove protocol [FPR26]. We use two *instances* of this system, a *binary instance* over $\mathbb{F}_2$ and a *prime-field instance* over $\mathbb{F}_q$, and tie them together in three phases. First, the prover commits to the private inputs of $C^{(2)}$ in the binary instance and to those of $C^{(q)}$ in the prime-field instance. Second, a common challenge phase takes place for the consistency test and the randomized tests: the verifier sends its challenges, and the prover announces the tag, together with the further public values that the prime-field circuit needs to check it. Only then does each instance prove its circuit, whose public inputs are now fixed.

Circuit inputs. The private inputs of each circuit consist of its input witness, its copy of the cross-field witness, its mask for the consistency test, and *auxiliary values*:

$$w_{\text{ext}}^{(2)} = \big(w_{\text{in}}^{(2)}, w_{\text{cross}}^{(2)}, \kappa^{(2)}, \mathsf{aux}^{(2)}\big), \qquad w_{\text{ext}}^{(q)} = \big(w_{\text{in}}^{(q)}, w_{\text{cross}}^{(q)}, \kappa^{(q)}, \mathsf{aux}^{(q)}\big).$$

The auxiliary values are supplied by the prover, and mostly derived from the witness. In the binary circuit, $\mathsf{aux}^{(2)}$ consists of intermediate states of the SHAKE computations, which allow all the Keccak permutations to be proven in parallel (Section 4.1). In the prime-field circuit, $\mathsf{aux}^{(q)}$ gathers the values that the prover supplies to the checks of $C^{(q)}$, such as the prefix counts of the samplers, lookup multiplicities, and memory versions used by the randomized tests, as well as the random pads pad of the consistency test; they are specified in Section 5.

Besides their private inputs, both circuits take public inputs, which are only fixed after the commitments: challenges sampled by the verifier and values announced by the prover. We separate the two kinds of inputs by a semicolon, and write

$$C^{(2)}\big(w_{\text{ext}}^{(2)}; R, y\big) \qquad \text{and} \qquad C^{(q)}\big(w_{\text{ext}}^{(q)}; \mathsf{ch}^{(q)}, R, y, e\big),$$

where R is the matrix of the consistency test, $\mathsf{ch}^{(q)}$ gathers the challenges of the randomized tests of $C^{(q)}$, y is the public tag, and e is a second public value of the consistency test. Indeed, on the prime-field side, each bit of the tag is checked through an integer equation that involves a quotient depending on R, which the prover cannot commit to in advance. It commits instead to random pads pad, included in $\mathsf{aux}^{(q)}$, and later announces in e the quotients masked by these pads (Section 5.8). A circuit outputs 1 when it accepts. Both circuits also depend on the statement x, which we leave implicit.

Protocol. The protocol proceeds as follows.

1. *Commitments.* The prover samples a uniform mask $\kappa \in \mathbb{F}_2^{N_{\text{tag}}}$, which it uses in both instances as $\kappa^{(2)} = \kappa^{(q)} = \kappa$, and uniform pads pad, which are part of $\mathsf{aux}^{(q)}$. The prover and the verifier then run the commit protocols of the two instances, independently of each other: the binary instance commits to $w_{\text{ext}}^{(2)}$, and the prime-field instance commits to $w_{\text{ext}}^{(q)}$.
2. *Challenges.* The verifier samples R and $\mathsf{ch}^{(q)}$.
3. *Announcements.* The prover sends the tag y and the masked quotients e, which it computes from R and its private inputs. The public inputs (R, y) and $(\mathsf{ch}^{(q)}, R, y, e)$ are then fixed: they are the statements of the two prove protocols.
4. *Proofs.* The prover and the verifier run the prove protocols of the two instances, independently of each other: the binary instance proves that $C^{(2)}\big(w_{\text{ext}}^{(2)}; R, y\big) = 1$, and the prime-field instance proves that $C^{(q)}\big(w_{\text{ext}}^{(q)}; \mathsf{ch}^{(q)}, R, y, e\big) = 1$. The verifier accepts if and only if both proofs are valid.

The two instances use independent random oracles, obtained by domain separation. Section 3.5 specifies the two circuits and shows that their joint acceptance implies the validity of the signature, up to a small error. The knowledge soundness and zero knowledge of the composed protocol are established in Section 3.6, which also discusses the non-interactive version obtained with the Fiat–Shamir transform.

3.5 Circuit Specifications

We now specify the two circuits: the format in which they receive the cross-field witness, the conditions that they check, and the guarantee given by their joint acceptance.

Format of the cross-field witness. Each circuit receives the cross-field witness in a format suited to its field. Recall that $u_{\text{cross}} = \mathsf{Enc}(w_{\text{cross}}) \in \mathbb{F}_2^{N_{\text{cross}}}$ is its binary encoding (Section 3.3). The binary circuit receives this vector itself: $w_{\text{cross}}^{(2)} = u_{\text{cross}}$. The prime-field circuit receives the same bits as elements of $\mathbb{F}_q$, in one of two forms:

– By default, a bit b is given as the field element $b \in \{0, 1\} \subset \mathbb{F}_q$, and $C^{(q)}$ enforces $b(b-1) = 0$.

– The bits of the matrix stream S_A are *packed*. For each matrix candidate, the 24 bits formed by the candidate and its validity bit are split into three groups of $s = 8$ bits, and each group $(b_0, \ldots, b_{s-1})$ is given as the single field element $\zeta = g^{\sum_{i<s} 2^i b_i}$, where $g \in \mathbb{F}_q^\times$ is a public element of order 2^s, which exists since 2^{13} divides $q-1$. The circuit $C^{(q)}$ enforces $\zeta^{2^s} = 1$, so that ζ lies in the subgroup generated by g and determines a unique group of bits, which the circuit then extracts (Section 5.1).

When all the elements of $w_{\text{cross}}^{(q)}$ satisfy their constraint, this copy encodes a unique vector of $\mathbb{F}_2^{N_{\text{cross}}}$, namely the vector $u_{\text{cross}}^{(q)}$ of Section 3.3.

Remark 3. The matrix stream is packed because it accounts for most of the cross-field witness when the verification key is hidden: $24k\ell B_A$ of its N_{cross} bits, i.e., 114 432 out of 132 559 for ML-DSA-44. The copy $w_{\text{cross}}^{(q)}$ then consists of $N_{\text{cross}} - 21k\ell B_A$ field elements, i.e., 32 431 for ML-DSA-44. For the other components, the gain does not compensate for the additional layers of the extraction circuit. The mask $\kappa^{(q)}$ and the Boolean auxiliary values of $C^{(q)}$ are given in the default form. When the verification key is public, S_A is not part of the cross-field witness and no value is packed.

Binary circuit. The circuit $C^{(2)}$, described in Section 4, uses no randomized test. Its acceptance exactly characterizes the binary relation and the binary side of the consistency test: for all $w_{\text{in}}^{(2)}$, $w_{\text{cross}}^{(2)}$, $\kappa^{(2)}$, R, and y,

$$\begin{aligned} &\exists\, \mathsf{aux}^{(2)} :\ C^{(2)}\big(w_{\text{in}}^{(2)}, w_{\text{cross}}^{(2)}, \kappa^{(2)}, \mathsf{aux}^{(2)}; R, y\big) = 1 \\ &\qquad\Longleftrightarrow\quad \mathcal{R}^{(2)}\big(w_{\text{in}}^{(2)}, w_{\text{cross}}\big) \ \text{ and } \ \mathsf{tag}_R\big(u_{\text{cross}}^{(2)}; \kappa^{(2)}\big) = y \end{aligned} \tag{14}$$

where $u_{\text{cross}}^{(2)} = w_{\text{cross}}^{(2)}$, and $w_{\text{cross}} = \mathsf{Enc}^{-1}(u_{\text{cross}}^{(2)})$ is the cross-field witness encoded by this copy, Enc being the binary encoding of Section 3.3.

Prime-field circuit. The circuit $C^{(q)}$, described in Section 5, satisfies the same kind of characterization for the prime-field relation and the prime-field side of the consistency test. Since it relies on randomized tests, one of the two implications only holds up to a soundness error $\epsilon^{(q)}$:

$$\begin{aligned} &\exists\, \mathsf{aux}^{(q)}, e :\ C^{(q)}\big(w_{\text{in}}^{(q)}, w_{\text{cross}}^{(q)}, \kappa^{(q)}, \mathsf{aux}^{(q)}; \mathsf{ch}^{(q)}, R, y, e\big) = 1 \\ &\qquad\overset{\epsilon^{(q)}}{\Longleftrightarrow}\quad \mathcal{R}^{(q)}\big(w_{\text{in}}^{(q)}, w_{\text{cross}}\big) \ \text{ and } \ \mathsf{tag}_R\big(u_{\text{cross}}^{(q)}; \kappa^{(q)}\big) = y \end{aligned} \tag{15}$$

where $w_{\text{cross}} = \mathsf{Enc}^{-1}(u_{\text{cross}}^{(q)})$ is the cross-field witness encoded by the copy of $C^{(q)}$. The right-hand side includes the requirements that $w_{\text{cross}}^{(q)}$ is a valid encoding, so that $u_{\text{cross}}^{(q)}$ and w_{cross} are defined, and that $\kappa^{(q)}$ consists of bits. The equivalence is to be read as follows, for every R.

– *From right to left (completeness):* if the right-hand side holds, there exist auxiliary values $\mathsf{aux}^{(q)}$ for which the circuit accepts for all challenges $\mathsf{ch}^{(q)}$, with suitable public values e.
– *From left to right (soundness):* if the circuit accepts, for any challenges and public values, then $w_{\text{cross}}^{(q)}$ is a valid encoding, $\kappa^{(q)}$ consists of bits, and $\mathsf{tag}_R\big(u_{\text{cross}}^{(q)}; \kappa^{(q)}\big) = y$: this part of the right-hand side holds without error. As for the relation, once the private inputs, including $\mathsf{aux}^{(q)}$, are fixed, the probability over $\mathsf{ch}^{(q)}$ that the circuit accepts for some public values (y, e) while $\mathcal{R}^{(q)}\big(w_{\text{in}}^{(q)}, w_{\text{cross}}\big)$ does not hold is at most $\epsilon^{(q)}$.

These properties are established by Proposition 4, with the error $\epsilon^{(q)}$ given in (46).

Joint acceptance. The following lemma states that the joint acceptance of the two circuits is equivalent to the validity of the signature, up to the errors of the consistency test and of the randomized tests.

Lemma 3 (Circuit Acceptance). *Let* $w_{\text{sem}} = (w_{\text{in}}^{(2)}, w_{\text{in}}^{(q)}) \in \mathcal{D}_{\text{sem}}$.

– Completeness: *if* $(x, w_{\text{sem}}) \in \mathcal{R}_{\text{ML-DSA}}$, *then for every mask* $\kappa \in \mathbb{F}_2^{N_{\text{tag}}}$, *there exist private inputs* $w_{\text{ext}}^{(2)}$ *and* $w_{\text{ext}}^{(q)}$ *with input witnesses* $w_{\text{in}}^{(2)}, w_{\text{in}}^{(q)}$ *and masks* $\kappa^{(2)} = \kappa^{(q)} = \kappa$ *such that, for all challenges* $(R, \mathsf{ch}^{(q)})$, *both circuits accept for some public values* (y, e).

– Soundness: *for all private inputs* $w_{\text{ext}}^{(2)}$ *and* $w_{\text{ext}}^{(q)}$ *with input witnesses* $w_{\text{in}}^{(2)}, w_{\text{in}}^{(q)}$*, the probability over* $(R, \mathsf{ch}^{(q)})$ *that there exist public values* (y, e) *such that*

$$C^{(2)}\big(w_{\text{ext}}^{(2)}; R, y\big) = 1, \qquad C^{(q)}\big(w_{\text{ext}}^{(q)}; \mathsf{ch}^{(q)}, R, y, e\big) = 1, \qquad (x, w_{\text{sem}}) \notin \mathcal{R}_{\text{ML-DSA}}$$

is at most $\max\big\{2^{-N_{\text{tag}}},\, \epsilon^{(q)}\big\}$.

Proof. Completeness: Lemma 1 gives a cross-field witness w_{cross} that satisfies both relations. Let both copies encode $u_{\text{cross}} = \mathsf{Enc}(w_{\text{cross}})$, and let $y = \mathsf{tag}_R(u_{\text{cross}}; \kappa)$. By (14), some $\mathsf{aux}^{(2)}$ makes $C^{(2)}$ accept, and by the completeness of $C^{(q)}$, some $\mathsf{aux}^{(q)}$ makes $C^{(q)}$ accept with the same tag y and suitable public values e. *Soundness:* Fix $w_{\text{ext}}^{(2)}$ and $w_{\text{ext}}^{(q)}$, and assume that both circuits accept for some (y, e). By (14) and by the exact part of the soundness of $C^{(q)}$, the copy $w_{\text{cross}}^{(q)}$ is a valid encoding and $\mathsf{tag}_R(u_{\text{cross}}^{(2)}; \kappa^{(2)}) = y = \mathsf{tag}_R(u_{\text{cross}}^{(q)}; \kappa^{(q)})$, where the two vectors and the two masks are determined by the private inputs, before the challenges are sampled. We distinguish two cases, which only depend on the private inputs. If $u_{\text{cross}}^{(2)} \neq u_{\text{cross}}^{(q)}$, the two tags are equal with probability $2^{-N_{\text{tag}}}$ over R, by Lemma 2. Otherwise, both copies encode the same cross-field witness w_{cross}, for which $\mathcal{R}^{(2)}$ holds by (14). If $(x, w_{\text{sem}}) \notin \mathcal{R}_{\text{ML-DSA}}$, Lemma 1 then shows that $\mathcal{R}^{(q)}$ does not hold for w_{cross}, so that $C^{(q)}$ accepts with probability at most $\epsilon^{(q)}$ over $\mathsf{ch}^{(q)}$. In both cases, the probability is at most $\max\big\{2^{-N_{\text{tag}}},\, \epsilon^{(q)}\big\}$.

3.6 Security of the Protocol

We now state the security of the protocol of Section 3.4. It follows from Lemma 3 and from the security of $\mathsf{SmallWood}^{+}$ as a commit-and-prove system, for which we use the definitions of [FPR26]: honest-verifier zero knowledge (HVZK) in the random oracle model [FPR26, Definition 7], and round-by-round knowledge soundness [FPR26, Definition 9]. The latter notion comes with a *commitment extractor*, which recovers the committed values from the commit protocol alone, before the statement of the prove protocol is known.

Each instance is a commit-and-prove system for the acceptance of its circuit: its witness is the vector of private inputs, and its statement is the tuple of public inputs. The corresponding relations are

$$\mathcal{R}_{\text{cp}}^{(2)} = \big\{\big(w_{\text{ext}}^{(2)}, (R, y)\big) : C^{(2)}\big(w_{\text{ext}}^{(2)}; R, y\big) = 1\big\},$$
$$\mathcal{R}_{\text{cp}}^{(q)} = \big\{\big(w_{\text{ext}}^{(q)}, (\mathsf{ch}^{(q)}, R, y, e)\big) : C^{(q)}\big(w_{\text{ext}}^{(q)}; \mathsf{ch}^{(q)}, R, y, e\big) = 1\big\}.$$

For $f \in \{2, q\}$, we assume that the instance over $\mathbb{F}_f$ is $(Q, \epsilon_{\text{zk}}^{(f)})$-HVZK and has $(Q, \epsilon_{\text{ro}}^{(f)}, \epsilon_{\text{rbr}}^{(f)})$-round-by-round knowledge soundness for $\mathcal{R}_{\text{cp}}^{(f)}$, where Q bounds the number of random-oracle queries, $\epsilon_{\text{rbr}}^{(f)}$ is the error per verifier message, and $\epsilon_{\text{ro}}^{(f)}$ bounds the probability of the bad events on the random oracle. For $\mathsf{SmallWood}^{+}$, these properties and the corresponding errors are given by [FPR26, Theorem 2 and Remark 6], the acceptance of each circuit being proven with a round-by-round sound sum-check-based interactive proof.

Theorem 1 (Security of the Protocol). *The protocol of Section 3.4 is a zero-knowledge argument of knowledge for the relation* $\mathcal{R}_{\text{ML-DSA}}$*: it is complete,* $(Q, \epsilon_{\text{zk}})$*-HVZK, and has* $(Q, \epsilon_{\text{ro}}, \epsilon_{\text{rbr}})$*-round-by-round knowledge soundness, with*

$$\epsilon_{\text{zk}} \le \epsilon_{\text{zk}}^{(2)} + \epsilon_{\text{zk}}^{(q)}, \qquad \epsilon_{\text{ro}} \le \epsilon_{\text{ro}}^{(2)} + \epsilon_{\text{ro}}^{(q)}, \qquad \epsilon_{\text{rbr}} \le \max\big\{\epsilon_{\text{rbr}}^{(2)},\, \epsilon_{\text{rbr}}^{(q)},\, 2^{-N_{\text{tag}}},\, \epsilon^{(q)}\big\},$$

where $\epsilon^{(q)}$ *is the soundness error of the randomized tests of* $C^{(q)}$ *(Proposition 4).*

Proof (Sketch). Completeness: For $(x, w_{\text{sem}}) \in \mathcal{R}_{\text{ML-DSA}}$, Lemma 3 gives private inputs for which both circuits accept for all challenges, with the public values (y, e) computed by the honest prover. Both statements then hold, and the two instances accept by completeness.

Knowledge soundness: Since the commit protocols of the two instances are independent, as are their prove protocols, we view the protocol as a sequence of five phases: the two commit protocols, the challenge phase, and the two prove protocols. The extractor runs the commitment extractors of the two instances, which gives $w_{\text{ext}}^{(2)}$ and $w_{\text{ext}}^{(q)}$, and outputs the input witnesses $w_{\text{sem}} = (w_{\text{in}}^{(2)}, w_{\text{in}}^{(q)})$ that they contain. These

values only depend on the commit protocols: they are fixed before $(R, \mathsf{ch}^{(q)})$ is sampled. The state function follows those of the two instances during their commit and prove protocols. In between, once the challenges are sent, the state is alive if $(x, w_{\text{sem}}) \in \mathcal{R}_{\text{ML-DSA}}$, or if some public values (y, e) make both circuits accept on the extracted private inputs, and it is doomed otherwise. When the extracted witness is not valid, the soundness part of Lemma 3 bounds the probability that the challenges lead to an alive state by $\max\{2^{-N_{\text{tag}}}, \epsilon^{(q)}\}$. From a doomed state, whatever public values (y, e) the prover announces, one of the two statements is false for the extracted private inputs: the corresponding instance starts its prove protocol in a doomed state, which it leaves with probability at most $\epsilon^{(f)}_{\text{rbr}}$ per verifier message. Finally, the bad events on the random oracles are those of the two instances, which gives ϵ_{ro} by a union bound.

Zero knowledge: In an honest execution, the mask κ and the pads pad are uniform. The tag y is then uniform and independent of the witness and of R, by Lemma 2, and the pads pad make e uniform conditioned on y (Section 5.8). The simulator thus samples R, $\mathsf{ch}^{(q)}$, y, and e uniformly at random, and runs the simulators of the two instances on the statements (R, y) and $(\mathsf{ch}^{(q)}, R, y, e)$, which is possible since the two instances use independent random oracles. The public inputs being the only values shared by the two instances, a hybrid argument bounds the distinguishing advantage by $\epsilon^{(2)}_{\text{zk}} + \epsilon^{(q)}_{\text{zk}}$.

Non-interactive version. The protocol is public-coin, and round-by-round knowledge soundness is the property that makes the Fiat–Shamir transform applicable to multi-round protocols [CCH+19]. We apply this transform along the structure of the protocol, which forms a tree. Its trunk consists of the commit protocols of the two instances, of the challenges $(R, \mathsf{ch}^{(q)})$, which are derived by hashing the statement x together with the transcripts of both commit protocols, and of the public values (y, e). The two prove protocols then form two independent branches: the challenges of each of them are derived from the trunk and from the messages of its own branch only. This structure matches the analysis above, in which the state of an instance only depends on the trunk and on the transcript of its own prove protocol.

4 Binary Circuit

The binary circuit $C^{(2)}$ implements the relation $\mathcal{R}^{(2)}$ of Section 3.2, together with the binary side of the consistency test of Section 3.3. As specified in Sections 3.4 and 3.5, its private inputs are $w^{(2)}_{\text{ext}} = \big(w^{(2)}_{\text{in}}, w^{(2)}_{\text{cross}}, \kappa^{(2)}, \mathsf{aux}^{(2)}\big)$, with $w^{(2)}_{\text{in}} = (pk, M, \widetilde{c})$, and its public inputs are the matrix R and the tag y. Its copy of the cross-field witness is the binary encoding itself, $w^{(2)}_{\text{cross}} = u^{(2)}_{\text{cross}} \in \mathbb{F}_2^{N_{\text{cross}}}$, whose bits give the words of

$$w_{\text{cross}} = (\mathbf{t}_1, S_A, S_c, \mathbf{w}'_1) = \mathsf{Enc}^{-1}\big(u^{(2)}_{\text{cross}}\big).$$

Since Enc is a bijection from $\mathcal{D}_{\text{cross}}$ onto $\mathbb{F}_2^{N_{\text{cross}}}$, every bit string is a valid copy, and $C^{(2)}$ needs no constraint on its format. Following the lines of $\mathcal{R}^{(2)}$, the circuit

- decodes pk, checks that its key coefficients equal $\mathbf{t}_1$, and computes tr and μ (Section 4.2);
- recomputes the sampler streams from ρ and $\widetilde{c}$, and checks that they equal S_A and S_c (Section 4.3);
- encodes $\mathbf{w}'_1$ and checks the final hash against $\widetilde{c}$ (Section 4.4);
- checks its side of the tag equation (13), $y = \mathsf{tag}_R\big(u^{(2)}_{\text{cross}}; \kappa^{(2)}\big)$. Since R and y are public, this check is $\mathbb{F}_2$-linear in $u^{(2)}_{\text{cross}}$ and $\kappa^{(2)}$, and thus costs no AND gate.

These items cover the four checks of $\mathcal{R}^{(2)}$ and the binary side of (13), and $C^{(2)}$ checks nothing else. The auxiliary values $\mathsf{aux}^{(2)}$ are the intermediate Keccak states of Section 4.1; the Keccak circuit checks them against the permutation, so that they are determined by the other private inputs, and the honest states satisfy all its constraints. Since $C^{(2)}$ uses no randomized test, it therefore satisfies the exact equivalence (14). We describe the general case in which the key and the message are hidden; Section 4.5 lists the simplifications when they are public. In all cases, the statement fixes the context string ctx and the message length ℓ_M (Section 3.1), so the input length of every SHAKE call is public and $C^{(2)}$ is a fixed circuit.

Proving $C^{(2)}$. The binary instance proves $C^{(2)}$ with the GKR protocol [GKR08], as supported by $\mathsf{SmallWood}^+$ [FPR26]. We therefore express the nonlinear computations of $C^{(2)}$ as layered circuits, whose inputs and outputs are linear functions of committed values. Most of this work consists of SHAKE128 and SHAKE256 evaluations, for which we use the layered circuit for the Keccak permutation of [FPR26] (Section 4.1). The only other nonlinear computation is that of the validity bits of the matrix candidates, which uses a small additional layered circuit (Section 4.3). All other operations of $C^{(2)}$, namely the decodings, the encodings, and the comparisons with words of w_{cross} or with $\widetilde{c}$, are linear.

4.1 SHAKE Evaluations with GKR

GKR reduces a claim on the multilinear extension of the output layer of a layered circuit to a claim on that of its input layer. In $\mathsf{SmallWood}^+$, both claims are checked by opening linear combinations of committed values [FPR26], so the inputs and outputs need not be committed separately when they are linear functions of committed values.

We prove all the permutations of $C^{(2)}$ in parallel with the Keccak circuit of [FPR26]. Their input and output states are linear functions of public constants, of the bits of $w_{\text{in}}^{(2)}$ and $w_{\text{cross}}^{(2)}$, and of auxiliary values. In particular, $\mathbf{t}_1$ (through pk) and $\mathbf{w}_1'$ enter the input layer directly, while the bits of S_A, S_c, and $\widetilde{c}$ are used to evaluate the output-layer claim, which implicitly checks that the computed outputs equal them. The chaining within and between SHAKE calls goes through auxiliary values: the prover commits to the states between consecutive permutations (including tr and μ), except for bits already in $w_{\text{cross}}^{(2)}$, and to the unused output bits of the last permutation of each call. Depending on the flattening parameter of the Keccak circuit [FPR26], it also commits to the states between some rounds of each permutation.

4.2 Binding the Key and the Message

The encoded key pk determines the matrix seed ρ, the key coefficients $\mathbf{t}_1$, and the digest used to hash the message. To bind them together, the circuit works directly on the bits of pk. Following $\mathsf{pkDecode}$ [Nat24], the first 32 bytes of pk give ρ and its remaining bits give the nk coefficients of $\mathbf{t}_1^*$, with ten bits per coefficient. Since every ten-bit word is admissible, decoding is a mere relabeling of bits and requires no range check. In particular, the check $\mathbf{t}_1^* = \mathbf{t}_1$ of $\mathcal{R}^{(2)}$ is free: the positions of $w_{\text{cross}}^{(2)}$ that hold $\mathbf{t}_1$ are directly the corresponding bits of pk. From the same bits, the circuit then computes

$$tr = H(pk, 64), \qquad \mu = H(tr \| M', 64)\,, \tag{16}$$

where M' is the formatted message defined in (1).

4.3 Generating the Sampler Streams

The circuit evaluates the Stream routines of Section 3.2, with the fixed budgets B_A and B_c of Section 3.1, and checks the two corresponding lines of $\mathcal{R}^{(2)}$:

$$S_A = \mathsf{ExpandA.Stream}_{B_A}(\rho), \qquad S_c = \mathsf{SampleInBall.Stream}_{B_c}(\widetilde{c}).$$

The Build routines, which select the accepted candidates, are left to $C^{(q)}$: the streams contain every candidate, including those that are later rejected.

Matrix stream. To evaluate $\mathsf{ExpandA.Stream}_{B_A}(\rho)$, for each matrix entry $f = (r, s)$, with $0 \le r < k$ and $0 \le s < \ell$, $C^{(2)}$ computes

$$G(\rho \| \mathsf{IntegerToBytes}(s, 1) \| \mathsf{IntegerToBytes}(r, 1),\, 3B_A)\,, \tag{17}$$

where, following FIPS 204, the column index precedes the row index. Each group of three output bytes yields one candidate: its first 23 bits form the candidate $v_{f,i}^A$, and its last bit is discarded [Nat24]. These B_A candidates, together with their validity bits, form the part of S_A associated with f. Fixing f to lighten notation, we write $v_i = v_{f,i}^A$, with bits $(v_{i,0}, \ldots, v_{i,22})$, and $e_i = e_{f,i}^A$ for its validity bit.

The validity bits tell $C^{(q)}$ which candidates are admissible, that is, which integers v_i are less than q. Since $q = 2^{23} - 2^{13} + 1$, a candidate is at least q exactly when its ten upper bits are all one and its thirteen lower bits are not all zero. For $1 \le i \le B_A$, $C^{(2)}$ thus checks

$$e_i = 1 + \left(\prod_{a=13}^{22} v_{i,a}\right)\left(1 + \prod_{a=0}^{12}(1 + v_{i,a})\right) \qquad \text{in } \mathbb{F}_2, \tag{18}$$

where the two factors detect these two conditions. This is the only nonlinear computation of $C^{(2)}$ outside Keccak. We prove it with a small additional layered circuit, alongside the Keccak circuit: as in Section 4.1, its input layer consists of the candidate bits and its output layer of the validity bits, both read from $w^{(2)}_{\text{cross}}$. Its two products are computed by binary trees of depth $\lceil \log_2 13 \rceil = 4$, followed by a final multiplication, that is, 22 multiplications per candidate in 5 layers. Section 5.2 then checks that each matrix entry is formed from the first n valid candidates of its stream, in order.

Challenge stream. To evaluate $\mathsf{SampleInBall.Stream}_{B_c}(\widetilde{c})$, the circuit computes $Y = H(\widetilde{c}, 8 + B_c)$ from the challenge seed, and we write Y_b for bit b of this stream and $Y[a]$ for byte a, both indexed from zero. As in FIPS 204, the first eight bytes are reserved for the signs, which suffices for all three parameter sets, and the remaining B_c bytes are the candidate indices. The circuit thus checks that $S_c = (\mathbf{j}, \mathbf{s})$ satisfies

$$\begin{aligned} s_g &= Y_{g-1}, && 1 \le g \le \tau, \\ j_i &= Y[7+i], && 1 \le i \le B_c, \end{aligned} \tag{19}$$

the unused sign bits being ignored. Note that i indexes every candidate, including those later rejected, whereas g is the rank of an acceptance: the g-th accepted candidate uses the g-th sign s_g, as checked in Section 5.3.

4.4 Recomputing the Challenge Seed

The last check of $\mathcal{R}^{(2)}$ connects the arithmetic part of the verification to the challenge seed of the signature. While $C^{(q)}$ checks that $\mathbf{w}'_1$ is the output of $\mathsf{UseHint}$, $C^{(2)}$ encodes $\mathbf{w}'_1$ and checks that hashing it with μ gives back $\widetilde{c}$.

Let $m_\gamma = (q-1)/(2\gamma_2)$. In w_{cross}, each coefficient of $\mathbf{w}'_1$ is a word of $\mathsf{bitlen}(m_\gamma - 1)$ bits, that is, six bits for ML-DSA-44 and four bits for ML-DSA-65 and ML-DSA-87. The encoding concatenates these words in increasing polynomial index, then increasing coefficient index, with the least significant bit of each word first. It is thus a mere arrangement of bits. For ML-DSA-65 and ML-DSA-87, every four-bit word lies in $\{0, \ldots, m_\gamma - 1\} = \{0, \ldots, 15\}$, and this arrangement is exactly $\mathsf{w1Encode}(\mathbf{w}'_1)$. For ML-DSA-44, a six-bit word may also take the values 44 to 63, on which FIPS 204 does not define $\mathsf{w1Encode}$. Since $\mathcal{R}^{(2)}$ takes any $w_{\text{cross}} \in \mathcal{D}_{\text{cross}}$ as input, we extend $\mathsf{w1Encode}$ to these words by the same arrangement of bits. The circuit then checks

$$H(\mu \| \mathsf{w1Encode}(\mathbf{w}'_1), \lambda/4) = \widetilde{c}, \tag{20}$$

where $\widetilde{c}$ is the same seed as the one that generates the challenge stream in (19). No range check on $\mathbf{w}'_1$ is needed in $C^{(2)}$: for a valid witness, the words lie in $\{0, \ldots, m_\gamma - 1\}$ because $\mathcal{R}^{(q)}$ checks $\mathbf{w}'_1 = \mathsf{UseHint}(\mathbf{h}, \mathbf{w}'_{\text{Approx}})$ (Section 5.6).

4.5 Public Verification Key or Public Message

The previous sections describe the case in which the key and the message are hidden. When one of them is public, some computations leave $C^{(2)}$. When pk is public, the verifier decodes it and derives $\widehat{\mathbf{A}} = \mathsf{ExpandA}(\rho)$ itself. Since this computation is performed outside the circuits, it needs no budget, and the condition on $\mathsf{ExpandA}_{B_A}$ disappears from the relation (11) (Section 3.1). The values $\mathbf{t}_1$ and $\widehat{\mathbf{A}}$ become public constants of $C^{(q)}$, and $\mathbf{t}_1$ and S_A leave w_{cross}. The circuit $C^{(2)}$ thus no longer evaluates $\mathsf{pkDecode}$, tr, the matrix streams, or their validity bits. In this case, $C^{(2)}$ still computes μ from the public digest tr and the hidden message. A public message alone changes little: M becomes a public constant in the input of the hash that computes μ, but this hash must still be evaluated in $C^{(2)}$, since μ also depends on tr, and thus on the hidden key. Only when both the key and the message are public does the verifier compute μ itself, which then becomes a public constant in the input of the final hash. Table 3, in the next section, summarizes which computations remain in each case.

4.6 Circuit Cost

The cost of $C^{(2)}$ is driven by the number of Keccak permutations, which determines both the size of the GKR circuit and the number of auxiliary values committed between permutations (Section 4.1). Absorbing L bytes in a SHAKE instance of rate r bytes takes $\lfloor L/r \rfloor + 1$ permutations, and squeezing O bytes takes $\lceil O/r \rceil - 1$ more, with $r = 168$ for SHAKE128 and $r = 136$ for SHAKE256. Table 3 gives the resulting counts. The circuit for the validity bits (18) is comparatively small: its $22\,k\ell B_A$ multiplications amount to between 2.7 and 9.6 times the $24 \cdot 1600$ multiplications of the χ layers of one permutation, and its inputs and outputs are already in $w^{(2)}_{\text{cross}}$, so it requires no auxiliary value.

Table 3: Keccak permutations in $C^{(2)}$, where $N_\mu = \lfloor (66 + L_{\mathsf{ctx}} + \ell_M/8)/136 \rfloor + 1$. The message length ℓ_M is in bits and is assumed to be a multiple of 8.

Computation	Required when	ML-DSA-44	ML-DSA-65	ML-DSA-87
Matrix streams (6 per entry)	hidden pk	96	180	336
Key digest tr	hidden pk	10	15	20
Message digest μ	hidden pk or M	N_μ	N_μ	N_μ
Challenge stream	always	2	2	2
Final challenge $\widetilde{c}$	always	7	7	9
Total, hidden pk		$115 + N_\mu$	$204 + N_\mu$	$367 + N_\mu$
Total, public pk and hidden M		$9 + N_\mu$	$9 + N_\mu$	$11 + N_\mu$
Total, public pk and M		9	9	11

5 Prime-Field Circuit

The prime-field circuit $C^{(q)}$ implements the relation $\mathcal{R}^{(q)}$ of Section 3.2, together with the prime-field side of the consistency test of Section 3.3. As specified in Sections 3.4 and 3.5, its private inputs are $w^{(q)}_{\text{ext}} = \big(w^{(q)}_{\text{in}}, w^{(q)}_{\text{cross}}, \kappa^{(q)}, \mathsf{aux}^{(q)}\big)$, with $w^{(q)}_{\text{in}} = (\mathbf{z}, \mathbf{h})$, and its public inputs are the challenges $\mathsf{ch}^{(q)}$ of its randomized tests, the matrix R, the tag y, and the announcements e. Its copy $w^{(q)}_{\text{cross}}$ of the cross-field witness gives the bits of $u^{(q)}_{\text{cross}}$ as elements of $\mathbb{F}_q$, packed for the matrix stream S_A and one per element otherwise. Through $w_{\text{cross}} = \mathsf{Enc}^{-1}(u^{(q)}_{\text{cross}}) = (\mathbf{t}_1, S_A, S_c, \mathbf{w}'_1)$, the key coefficients and the sampler streams are inputs of the computation, while $\mathbf{w}'_1$ is its claimed output. Following the lines of $\mathcal{R}^{(q)}$, the circuit

- checks that $w^{(q)}_{\text{cross}}$, $\mathbf{h}$, $\kappa^{(q)}$, and the Boolean auxiliary values encode bits, and decodes the packed matrix stream (Section 5.1);
- builds the matrix entries $\widehat{\mathbf{A}}$ from the candidates of S_A (Section 5.2);
- builds the challenge c from the candidates and signs of S_c (Section 5.3);
- computes $\mathbf{w}'_{\text{Approx}}$ and checks that $\mathbf{w}'_1 = \mathsf{UseHint}(\mathbf{h}, \mathbf{w}'_{\text{Approx}})$ (Sections 5.4 and 5.6);
- checks the bound on $\mathbf{z}$ and the weight of $\mathbf{h}$ (Section 5.5);
- checks its side of the tag equation (13), $y = \mathsf{tag}_R\big(u^{(q)}_{\text{cross}}; \kappa^{(q)}\big)$. Unlike in $C^{(2)}$, this linear map over $\mathbb{F}_2$ must be expressed over $\mathbb{F}_q$ (Section 5.8).

Section 5.9 combines these checks and shows that $C^{(q)}$ satisfies the specification (15). We describe the general case in which the verification key is hidden; Section 5.10 lists the simplifications when it is public, and Section 5.11 summarizes the cost of $C^{(q)}$.

Proving $C^{(q)}$. As for $C^{(2)}$ (Section 4.1), the prime-field instance proves the nonlinear computations of $C^{(q)}$ with GKR, as shallow layered circuits. Their inputs may also involve the challenges of the randomized tests, and their outputs are either required to be zero or linear in committed values. These computations are

- the Boolean constraints $b(b-1) = 0$ and the decoding of the packed matrix stream (Section 5.1);

– the local constraints of the samplers (Sections 5.2 and 5.3), which use one or two multiplication layers;
– the coefficient-wise products $\widehat{\mathbf{A}}\,\widehat{\mathbf{z}}$ and $\widehat{c}\,\widehat{\mathbf{t}_1}$ of the verification equation in the NTT domain (Section 5.4), which form one multiplication layer;
– the product trees of the multiset equalities behind the ordered selections and the memory check (Sections 5.2 and 5.3);
– the sum of fractions of the lookup argument (Section 2.3), which performs all the range and hint-correction checks of the circuit at once and is computed as a balanced tree of fraction additions (Section 5.7).

All other operations, namely the recomposition of words from the bits of $w^{(q)}_{\text{cross}}$, the NTTs, the limb decompositions, the hint index j, and the parity checks of the consistency test, are linear. The challenges of the multiset equalities and of the lookup are sampled from the extension field $\mathbb{K}$ of $\mathbb{F}_q$, of degree r, introduced with its basis $(\theta_1, \ldots, \theta_r)$ in Section 2.3, so the product and fraction trees compute over $\mathbb{K}$. Each element of $\mathbb{K}$ is represented by its coordinates in this basis, and a multiplication in $\mathbb{K}$ becomes a bilinear layer on these coordinates over $\mathbb{F}_q$. The trees thus remain layered circuits of degree two over $\mathbb{F}_q$. Many of these subcircuits are identical copies, such as the $k\ell$ matrix selections or the per-coefficient constraints, and are proved in parallel. The circuits of the randomized tests depend on challenges sampled after the commitments; this is compatible with GKR, which is run after these challenges are known.

Range checks. We fix a public bound T (Table 5) satisfying

$$T \geq \max\{n-1, \omega, 2\tau-1\}, \qquad 2T < q, \qquad nk + T < q.$$

We write $\mathcal{T} = \{0, \ldots, T\}$ and $\mathsf{Range}(x)$ for $x \in \mathcal{T}$, as in Section 2.3. The inequalities refer to integer representatives, with centered representatives for the response coefficients. All range checks and hint-correction checks are performed together by the common lookup of Section 5.7. Two bounds checked by the circuit are larger than T, on the response and on the hint offset; they are handled with two range-checked limbs, which imposes a further condition on T, stated in Section 5.5.

Conditions on the parameters. Besides the conditions on T, the subsections below rely on the following conditions on the parameters:

$$B_A, B_c < q, \qquad 2(\gamma_1 - \beta - 1) < q, \qquad 2^{b_w+1} < q, \qquad N_Q < q, \qquad 2N_{\text{cross}} + 1 < q,$$

where b_w, N_Q, and N_{cross} are defined in Sections 5.6, 5.7, and 3.3. They prevent the counts, the bounds, the packed hint corrections, the lookup multiplicities, and the parity sums from wrapping modulo q, and hold with a large margin for all ML-DSA parameter sets, where $q \approx 2^{23}$.

5.1 Bits over the Prime Field

Several values of $C^{(q)}$ are bits held as elements of $\mathbb{F}_q$: the entries of $w^{(q)}_{\text{cross}}$, the hint $\mathbf{h}$, the mask $\kappa^{(q)}$, and some auxiliary values. Following the format of Section 3.5, they are given in one of two forms.

Boolean form. By default, a bit b is given as the field element $b \in \{0,1\} \subset \mathbb{F}_q$, and $C^{(q)}$ enforces $b(b-1) = 0$. This form is used for

– the bits of $\mathbf{t}_1$, S_c, and $\mathbf{w}'_1$ in $w^{(q)}_{\text{cross}}$;
– the hint $\mathbf{h}$ (Section 5.6);
– the selection bits $a^A_{f,i}$ and a_i of the samplers, which are differences of committed prefix counts (Sections 5.2 and 5.3);
– the mask $\kappa^{(q)}$ and the pads pad of the consistency test (Section 5.8).

Each constraint is a single multiplication layer whose input is affine in committed values, and whose output must be zero.

Packed form. The bits of the matrix stream S_A are packed. Since 2^{13} divides $q-1$, we fix $s=8$ and a public element $g \in \mathbb{F}_q^\times$ of order exactly 2^s. For each candidate, the 24 bits formed by $v^A_{f,i}$ and $e^A_{f,i}$ are split into three groups of s bits, and a group $(b_0, \dots, b_{s-1})$ is given as the single field element

$$\zeta = g^E, \qquad E = \sum_{i=0}^{s-1} 2^i b_i.$$

The circuit computes the squaring chain

$$P_0 = \zeta, \qquad P_{m+1} = P_m^2 \quad (0 \le m < s),$$

so that $P_m = \zeta^{2^m}$, and enforces $P_s = 1$. Since $\mathbb{F}_q^\times$ is cyclic, the solutions of $\zeta^{2^s} = 1$ form the subgroup generated by g, so ζ determines a unique group of s bits. We encode the bits in the exponent rather than as the integer $\sum_i 2^i b_i$, since the latter would require dense polynomials of degree about 2^s to check its range and to extract each bit, hence about 2^s multiplications per element. In the subgroup, the membership check and the extraction of the bits both derive from the squaring chain, for a total of $s + s(s-1)/2$ multiplications per element, i.e., 36 for $s=8$.

To decode ζ, note that squaring multiplies the exponent by two modulo 2^s, and thus erases its most significant bit: $P_m = g^{2^m (E \bmod 2^{s-m})}$ only depends on the $s-m$ lowest bits. In particular, the least significant bit is the only one isolated by a power, $P_{s-1} = (-1)^{b_0}$. The circuit therefore recovers the bits from the least to the most significant, going back down the chain and removing from P_{s-1-i} the contribution of the bits already decoded:

$$b_i = \frac{1 - P_{s-1-i}\, C_i}{2}, \qquad C_i = \prod_{k<i} \bigl(1 + (g^{-2^{s-1-i+k}} - 1)\, b_k\bigr) \qquad (0 \le i < s). \tag{21}$$

To see this, split the exponent into the bits below, at, and above position i:

$$E = E_{<i} + 2^i b_i + 2^{i+1} E_{>i}, \qquad E_{<i} = \sum_{k<i} 2^k b_k, \qquad E_{>i} = \sum_{k>i} 2^{k-i-1} b_k.$$

By induction on i, assume that $b_0, \dots, b_{i-1}$ are bits. Each factor of C_i then equals $g^{-2^{s-1-i+k} b_k}$, so that $C_i = g^{-2^{s-1-i} E_{<i}}$ cancels the lower bits of $P_{s-1-i} = g^{2^{s-1-i} E}$:

$$P_{s-1-i}\, C_i = g^{2^{s-1-i}(2^i b_i + 2^{i+1} E_{>i})} = g^{2^{s-1} b_i} \left(g^{2^s}\right)^{E_{>i}} = (-1)^{b_i},$$

since $g^{2^{s-1}} = -1$ and $g^{2^s} = 1$. The decoded values are thus the bits of the group, and the subgroup check guarantees that they are Boolean without further constraint.

As a layered circuit, the squaring chain takes s multiplication layers. The corrections are sequential, since b_i depends on $b_0, \dots, b_{i-1}$. To avoid recomputing each product C_i, the circuit maintains, for every bit t not yet decoded, the partially corrected power

$$Q_{t,i} = P_{s-1-t} \prod_{k<i} \bigl(1 + (g^{-2^{s-1-t+k}} - 1)\, b_k\bigr), \qquad Q_{t,0} = P_{s-1-t},$$

so that $b_i = (1 - Q_{i,i})/2$ is linear in $Q_{i,i}$. Once b_i is known, a single multiplication layer updates all the remaining powers in parallel,

$$Q_{t,i+1} = Q_{t,i} \bigl(1 + (g^{-2^{s-1-t+i}} - 1)\, b_i\bigr) \qquad (i < t < s),$$

whose second factor is affine in b_i. Since b_0 is available after $s-1$ squarings, and each of the $s-1$ remaining bits requires one update layer, the decoding has depth $2s-2$; the subgroup check, which uses the s-th squaring, runs in parallel with the first update. Values produced before the layer that uses them, such as the powers P_{s-1-t}, the decoded bits, and the subgroup check, are carried to that layer by identity gates.

The inputs of this circuit are the committed packed elements, and its outputs are the bits of S_A. These bits are not committed: the decoding circuit is composed with the circuits that use them, namely the layered circuits of Section 5.2, which take its outputs as inputs, and the parity checks of Section 5.8, which are linear in them. The decoded bits are fixed by the packed elements before the challenges are sampled.

Encoded vector. When all these constraints hold, the copy $w^{(q)}_{\text{cross}}$ determines a unique vector $u^{(q)}_{\text{cross}} \in \mathbb{F}_2^{N_{\text{cross}}}$, whose bits are the Boolean entries and the decoded bits, in the order of Enc. Conversely, every vector of $\mathbb{F}_2^{N_{\text{cross}}}$ is encoded by a copy satisfying these constraints. In the following, the words of $w_{\text{cross}} = \mathsf{Enc}^{-1}(u^{(q)}_{\text{cross}})$ are recomposed from these bits by affine combinations $\sum_a 2^a b_a$ in $\mathbb{F}_q$.

5.2 Matrix Expansion

This subsection implements ExpandA.Build: each matrix entry must consist of the first n valid candidates of its stream, in order. For the stream $f = (r, s)$, write $v^A_{f,i}$ for its i-th candidate and $e^A_{f,i}$ for its validity bit, both part of S_A and checked by $C^{(2)}$. In $C^{(q)}$, their bits are decoded from the packed elements of $w^{(q)}_{\text{cross}}$ (Section 5.1), and the candidate $v^A_{f,i}$ is recomposed from its 23 bits as a residue modulo q, whereas validity refers to the full 23-bit word: a candidate outside $[0, q-1]$ has the same residue as a valid one. This is why S_A carries the validity bits alongside the candidates.

Selected positions. Instead of implementing a loop that appends each valid candidate, we check which positions are selected, and then extract them in order. The prover commits to the counts of selected positions $N^A_{f,1}, \ldots, N^A_{f,B_A-1}$, while the endpoints $N^A_{f,0} = 0$ and $N^A_{f,B_A} = n$ are public constants. The selection bits are their differences $a^A_{f,i} = N^A_{f,i} - N^A_{f,i-1}$, which are affine in them. The first line of

$$\begin{gathered} N^A_{f,0} = 0, \qquad N^A_{f,i} = N^A_{f,i-1} + a^A_{f,i}, \qquad N^A_{f,B_A} = n, \\ (n - N^A_{f,i-1})(a^A_{f,i} - e^A_{f,i}) = 0 \end{gathered} \tag{22}$$

then holds by construction, and the circuit checks the second line, together with the Boolean constraints on the selection bits (Section 5.1). Since the selection bits are Boolean and $B_A < q$, the prefix counts are integers between zero and B_A. As long as the count is below n, the last equation forces $a^A_{f,i} = e^A_{f,i}$. Once it reaches n, every further selection bit is zero, since the count cannot decrease and must end at n. The selection bits therefore mark exactly the first n valid candidates.

Ordered extraction. Let $\mathbf{v}^A_f$ and $\mathbf{a}^A_f$ be the sequences of candidates and selection bits, and $\widehat{\mathbf{A}}_f$ the claimed matrix entry, in NTT order. The ordered-selection test of Section 2.3 checks that $\widehat{\mathbf{A}}_f$ is the list of the candidates of $\mathbf{v}^A_f$ whose selection bit is set, in the same order:

$$\mathsf{Select}(\mathbf{v}^A_f, \mathbf{a}^A_f; \widehat{\mathbf{A}}_f) = 1. \tag{23}$$

This test is a multiset equality on pairs made of a selected candidate and its rank among the selected ones. Here the rank is the prefix count $N^A_{f,i}$, which is thus shared by both checks.

Proposition 1 (Bounded Matrix Expansion). *Fix a stream f, candidates $\mathbf{v}^A_f$ with validity bits $\mathbf{e}^A_f$, and a claimed entry $\widehat{\mathbf{A}}_f$, and assume $B_A < q$. There exist Boolean selection bits satisfying* (22) *and the ordered-selection property* (23) *if and only if at least n candidates of the stream are valid and $\widehat{\mathbf{A}}_f$ consists of the first n of them, in order.*

Proof. Suppose the constraints hold. By the argument above, the selection bits mark exactly the first n valid candidates, so at least n candidates are valid, and the ordered-selection property makes $\widehat{\mathbf{A}}_f$ the list of these candidates, in order. Conversely, if at least n candidates are valid, set $a^A_{f,i} = 1$ exactly at the positions of the first n of them. The prefix counts then reach n at the last of these positions, the last equation of (22) holds at every position, and the ordered-selection property holds by definition of $\widehat{\mathbf{A}}_f$.

Layered circuits. The last equation of (22) is a single multiplication layer, whose inputs $n - N^A_{f,i-1}$ and $a^A_{f,i} - e^A_{f,i}$ are linear in committed values and in the decoded bits, and whose outputs must be zero. The product check (3) behind Select uses one layer to compute the factors $1 + a^A_{f,i}\big(\chi + \psi(N^A_{f,i}, v^A_{f,i}) - 1\big)$, followed by a balanced product tree of depth $\lceil \log_2 B_A \rceil$. On the other side, the factors are linear in the committed entries of $\widehat{\mathbf{A}}_f$, and their tree has depth $\lceil \log_2 n \rceil$. The circuit outputs the difference of the two roots, which must be zero. The $k\ell$ matrix entries give identical circuits, proved in parallel. Since both circuits read the candidates and validity bits decoded in Section 5.1, they are composed with the decoding circuit, whose $2s - 2$ layers are placed below them.

5.3 Challenge Sampling

This subsection implements SampleInBall.Build: from the candidate bytes $\mathbf{j} = (j_1, \ldots, j_{B_c})$ and the sign bits $\mathbf{s} = (s_1, \ldots, s_\tau)$ of S_c, it reconstructs the polynomial returned by the bounded sampler, which reads at most B_c candidate bytes after the eight sign bytes and returns $\perp$ if fewer than τ of them are accepted. NIST Algorithm 29 recalls the ML-DSA sampler [Nat24].

Unlike for the matrix, the control flow of this sampler is sequential: the number of earlier acceptances fixes the threshold of the next candidate, and each array update depends on the previous ones. We therefore proceed in three steps. We check the acceptance decisions locally, extract the accepted bytes with Select, and check the array updates with a memory check.

NIST Algorithm 29 $\mathsf{SampleInBall}(\widetilde{c})$

Input: Seed $\widetilde{c} \in \mathbb{B}^{\lambda/4}$
Output: $c \in R_q$, with coefficients in $\{-1, 0, 1\}$ and weight τ
1: $c \leftarrow 0$
2: $\mathsf{st} \leftarrow \mathsf{H.Absorb}(\mathsf{H.Init}(), \widetilde{c})$
3: $(\mathsf{st}, Y) \leftarrow \mathsf{H.Squeeze}(\mathsf{st}, 8)$
4: $(s_1, \ldots, s_{64}) \leftarrow \mathsf{BytesToBits}(Y)$
5: **for** $g = 1, \ldots, \tau$ **do**
6: $\quad I_g \leftarrow n - \tau + g - 1$
7: $\quad$ **repeat**
8: $\quad\quad (\mathsf{st}, J_g) \leftarrow \mathsf{H.Squeeze}(\mathsf{st}, 1)$
9: $\quad$ **until** $J_g \leq I_g$
10: $\quad c[I_g] \leftarrow c[J_g]$
11: $\quad c[J_g] \leftarrow 1 - 2s_g$
12: Return c

Acceptance decisions. For each candidate j_i, the prover supplies a comparison slack e_i. As in Section 5.2, it also commits to the prefix counts $N_1, \ldots, N_{B_c-1}$ of the accepted candidates, while the endpoints $N_0 = 0$ and $N_{B_c} = \tau$ are public constants. The selection bit $a_i = N_i - N_{i-1}$, set when the candidate is accepted, is affine in them. We require $B_c < q$. In

$$N_0 = 0, \qquad N_i = N_{i-1} + a_i, \qquad N_{B_c} = \tau, \qquad a_i(a_i - 1) = 0, \qquad 1 \le i \le B_c, \tag{24}$$

the first three equations hold by construction, and the circuit checks the Boolean constraints. The counts are then nondecreasing integers from zero to τ. Let $I_g = n - \tau + g - 1$ be the position filled in round g of Algorithm 29. While fewer than τ candidates have been accepted, that is, while $N_{i-1} < \tau$, the candidate j_i belongs to round $N_{i-1} + 1$, and it is accepted if and only if $j_i \le I_{N_{i-1}+1}$. We express this comparison through a nonnegative slack:

$$e_i = \begin{cases} I_{N_{i-1}+1} - j_i & a_i = 1, \\ j_i - I_{N_{i-1}+1} - 1 & a_i = 0. \end{cases} \tag{25}$$

We combine the two cases in one equation, and disable the comparison after the τ-th acceptance with the factor $\tau - N_{i-1}$:

$$(\tau - N_{i-1})\big(e_i - (2a_i - 1)(I_{N_{i-1}+1} - j_i) + (1 - a_i)\big) = 0, \qquad \mathsf{Range}(e_i). \tag{26}$$

Before the τ-th acceptance, the correct decision gives $e_i \in [0, n-1]$, while the wrong one gives a negative slack of magnitude at most n, whose residue is at least $q - n > T$ for the ML-DSA parameters. The range check therefore enforces the correct decision. After the τ-th acceptance, the comparison is disabled, and the final count and the Boolean selection bits force all remaining selection bits to zero.

Accepted bytes. Write $\mathbf{a} = (a_i)_i$. The prover supplies the accepted bytes $\mathbf{J} = (J_1, \ldots, J_\tau)$, and we check

$$\mathsf{Select}(\mathbf{j}, \mathbf{a}; \mathbf{J}) = 1,$$

whose ranks are again the counts N_i. The ordered-selection property makes J_g the byte accepted in round g, which is paired with the sign bit s_g of S_c.

Array updates. In round g, let $\sigma_g = 1 - 2s_g$ be the sign written at the accepted byte J_g. The sampler copies the value at J_g to I_g, then writes the sign at J_g:

$$c[I_g] \leftarrow c[J_g], \qquad c[J_g] \leftarrow \sigma_g. \tag{27}$$

We check these updates with a simple extension of the memory check of Section 2.3, which differs in two respects. First, each round performs two writes, so the versions created in round g have times $2g-1$ and $2g$ instead of the index of an update. Second, we initialize only the addresses $p < n - \tau$, with zero versions, and introduce each I_g at its first write, without deleting an earlier version: this write creates the first version of I_g, which plays the role of its initial version. This is possible because no earlier round can access I_g, as shown in Lemma 4. The prover supplies the version (J_g, v_g, t_g) deleted in round g and the claimed final versions, which give the four lists below.

List	Versions supplied to the check
Initial $\mathcal{O}$	$(p, 0, 0)$ for $0 \le p < n - \tau$
Created $\mathcal{W}$	$(I_g, v_g, 2g-1)$ and $(J_g, \sigma_g, 2g)$ in each round g
Deleted $\mathcal{D}$	(J_g, v_g, t_g), deleted at time $2g$
Final $\mathcal{F}$	$(p, c[p], t_p^{\text{fin}})$ for $0 \le p < n$

Each round thus creates two versions but deletes only one: the write at I_g replaces nothing, and the write at J_g deletes the version whose value v_g is copied to I_g. We check the conservation law

$$\mathcal{O} \uplus \mathcal{W} = \mathcal{D} \uplus \mathcal{F} \tag{28}$$

with the multiset equality test of Section 2.3, on $n+\tau$ versions per side, and the time bounds $0 \le t_g < 2g$ with two range checks on t_g and $2g - 1 - t_g$, included in the common lookup of Section 5.7. The time bounds ensure that a version is created before it is deleted.

Lemma 4 (Array Updates). *For the version lists above, assume exact memory conservation and*

$$0 \le J_g \le I_g, \qquad 0 \le t_g < 2g \qquad (1 \le g \le \tau).$$

Then the claimed array c is obtained from the zero array by applying (27) *in order.*

Proof. Initial versions. The initialization above gives each address exactly one first version: a zero version below $n - \tau$, or the version created at time $2g - 1$ for I_g. Earlier rounds cannot access I_g because $J_h \le I_h < I_g$ for $h < g$. The memory argument of Section 2.3 only uses that each address has a unique first version, created before any deletion at this address, and that each deletion removes a version created strictly earlier. It therefore extends to these lists: conservation and the time bounds force each deletion to use the latest version at its address, and the last remaining version is the final one.

One round. Suppose the remaining versions agree with the array of the ML-DSA sampler on the addresses introduced before round g. There are two cases.

- If $J_g < I_g$, the deleted version contains the current value at J_g. The first write copies this value to I_g, and the second replaces it by σ_g at J_g.
- If $J_g = I_g$, the version created at time $2g - 1$ is deleted immediately at time $2g$:

  $$(I_g, v_g, 2g-1) \longrightarrow (I_g, \sigma_g, 2g).$$

  The intermediate value v_g has no effect on the resulting array. The ML-DSA sampler also leaves σ_g at this address: its first assignment copies the cell to itself, and its second writes the sign.

Both cases give the state of the ML-DSA sampler after round g. Induction covers all τ rounds. At that point every address has been introduced, so the final versions give exactly the output of the ML-DSA sampler. □

Proposition 2 (Bounded Challenge Reconstruction). *Fix candidate bytes* $\mathbf{j}$*, sign bits* $\mathbf{s}$*, and a claimed polynomial c. Assume $B_c < q$, $T \ge \max\{n-1, 2\tau-1\}$, and $2T < q$. There exist auxiliary values satisfying* (24) *and* (26)*, the exact range and ordered-selection properties, and memory conservation with its time bounds if and only if the bounded sampler completes on* $(\mathbf{j}, \mathbf{s})$ *and returns c.*

Proof. Suppose the stated properties hold. The acceptance and ordered-selection arguments above give the first τ accepted bytes, with $0 \le J_g \le I_g$. Together with memory conservation and the time bounds, Lemma 4 then identifies the claimed c with the output of the bounded sampler.

Conversely, use the selection bits and accepted bytes of a completed bounded execution. Choose the active slacks as in (25); after completion, keep the counts at τ and set the remaining slacks to zero. For $J_g < I_g$, take (v_g, t_g) from the current version at J_g. For $J_g = I_g$, take $(v_g, t_g) = (0, 2g-1)$. The final version at each address is its last version. Every version is either deleted once or kept at the end, so conservation holds. The slacks lie in $[0, n-1]$ and the range-checked times in $[0, 2\tau-1]$, both contained in $[0, T]$. These choices satisfy all the constraints. □

Layered circuits. The selection bits and the positions $I_{N_{i-1}+1}$ are affine in the committed prefix counts. The acceptance constraint (26) has degree three and uses two multiplication layers: the first computes $(2a_i - 1)(I_{N_{i-1}+1} - j_i)$, and the second multiplies the resulting linear combination by $\tau - N_{i-1}$; the outputs must be zero. As in Section 5.2, the ordered selection of the accepted bytes is a product circuit of depth $1 + \lceil \log_2 B_c \rceil$ on one side and $\lceil \log_2 \tau \rceil$ on the other. The memory check has no selection bits, so its factors are linear in committed values, and its two product trees have depth $\lceil \log_2(n+\tau) \rceil$.

5.4 Verification Equation

With the matrix and challenge fixed by the preceding checks, the circuit computes $\mathbf{w}'_{\mathrm{Approx}}$ in the NTT representation of ML-DSA:

$$\widehat{w}'_{\mathrm{Approx},r,i} = \sum_{s=0}^{\ell-1} \widehat{A}_{r,s,i}\widehat{z}_{s,i} - 2^d \widehat{c}_i \widehat{t}_{1,r,i}. \tag{29}$$

Here $\widehat{\mathbf{z}} = \mathrm{NTT}(\mathbf{z})$, $\widehat{c} = \mathrm{NTT}(c)$, and $\widehat{\mathbf{t}}_1 = \mathrm{NTT}(\mathbf{t}_1)$ are computed in the circuit, where c is the claimed polynomial of Section 5.3, and $\mathbf{t}_1$ is recomposed from its bits in $w^{(q)}_{\mathrm{cross}}$. The NTT is linear, so these transforms cost no multiplication; the only products are the $nk\ell$ products $\widehat{A}_{r,s,i}\widehat{z}_{s,i}$ and the nk products $\widehat{c}_i\widehat{t}_{1,r,i}$. To make the hint-correction lookup queries below linear in committed values, the prover commits to the nk coefficients of $\mathbf{w}'_{\mathrm{Approx}}$ as auxiliary values, whose NTT is linear in them. With GKR, the products then form one multiplication layer, whose inputs are linear in the committed claimed matrix entries, response limbs, final versions of c, and bits of $\mathbf{t}_1$. The circuit outputs

$$\sum_{s=0}^{\ell-1} \widehat{A}_{r,s,i}\widehat{z}_{s,i} - 2^d \widehat{c}_i \widehat{t}_{1,r,i} - \widehat{w}'_{\mathrm{Approx},r,i},$$

which must be zero, and which is linear in the products and in the committed coefficients. Section 5.6 applies UseHint to these coefficients.

5.5 Response and Hint Bounds

We now enforce the response norm and the hint weight with range checks. For the response, we describe a two-limb technique, which Section 5.6 reuses for the hint correction.

Two-limb bounds. A range check only bounds values by T. To check that a value $x \in \mathbb{F}_q$ lies in $\{0, \dots, N\}$ for a public bound $N < q$ larger than T, the prover commits to two limbs $H, L \in \mathbb{F}_q$, from which x is recomposed as

$$x = \Lambda H + L. \tag{30}$$

The public parameters $\Lambda, H_{\max}$ satisfy

$$\Lambda H_{\max} + T = N, \qquad 1 \le \Lambda \le T + 1, \qquad H_{\max} \le T, \tag{31}$$

and the circuit bounds the limbs with three range checks:

$$\mathsf{Range}(H), \qquad \mathsf{Range}(L), \qquad \mathsf{Range}(H_{\max} - H). \tag{32}$$

The low limb is bounded by a single range check, and the high limb by a range check and a complementary one. Since $2T < q$, Equation (7) gives exactly $0 \le H \le H_{\max}$ and $0 \le L \le T$, hence $0 \le \Lambda H + L \le N < q$, so that x lies in $\{0, \dots, N\}$. Conversely, the condition $\Lambda \le T + 1$ makes the intervals $[\Lambda H, \Lambda H + T]$, for $0 \le H \le H_{\max}$, overlap or be adjacent, so that they cover $\{0, \dots, N\}$: every value of this set admits limbs. The conditions (31) only require $N - T$ to factor as $\Lambda H_{\max}$ with $\Lambda \le T + 1$ and $H_{\max} \le T$.

Response bound. Let $Z = \gamma_1 - \beta - 1$, with $2Z < q$. The bound $\|\mathbf{z}\|_\infty < \gamma_1 - \beta$ holds if and only if every coefficient z_i has a representative in $\{-Z, \ldots, Z\}$, that is, $z_i + Z \in \{0, \ldots, 2Z\}$. We apply the two-limb technique with $N = 2Z$ and public parameters (Λ_z, H_z) satisfying (31) (Table 5): rather than committing to z_i directly, the prover commits to two limbs $z_{1,i}, z_{0,i}$, from which z_i is recomposed as

$$z_i = \Lambda_z z_{1,i} + z_{0,i} - Z, \tag{33}$$

and the circuit performs the three range checks (32) on $(H, L) = (z_{1,i}, z_{0,i})$. Limbs satisfying these checks exist if and only if $|z_i| \le Z$.

Hint weight. The hint $\mathbf{h}$ is a Boolean vector (Section 5.1). The weight bound $\sum_i h_i \le \omega$ is enforced by the single range check

$$\mathsf{Range}\left(\omega - \sum_{i=1}^{nk} h_i\right).$$

Since $\omega \le T$ and $nk + T < q$, this membership is equivalent to the bound over the integers.

The range checks of this subsection are lookup queries that are affine in committed values. Their realization as a layered circuit for GKR is described in Section 5.7, together with the other lookup queries.

5.6 Hint Correction

This subsection checks $\mathbf{w}_1' = \mathsf{UseHint}(\mathbf{h}, \mathbf{w}'_{\mathrm{Approx}})$, where $\mathbf{w}'_{\mathrm{Approx}}$ is computed in Section 5.4. The check is done coefficient by coefficient. For one coefficient, write

$$r = w'_{\mathrm{Approx},i}, \qquad h = h_i, \qquad w = w'_{1,i}.$$

Here $r \in \mathbb{F}_q$ is a committed coefficient, h is a bit of the hint, and $w = \sum_{a<b_w} 2^a b_a$ is recomposed from its b_w bits in $w^{(q)}_{\mathrm{cross}}$, where $b_w = \mathsf{bitlen}(m_\gamma - 1)$; in particular, $0 \le w < 2^{b_w}$. We want to check $w = \mathsf{UseHint}(h, r)$.

The correction of ML-DSA. Let $r^+ \in \{0, \ldots, q-1\}$ be the integer representative of r. Decompose [Nat24] computes $r_0 = r^+ \bmod^\pm 2\gamma_2 \in \{1-\gamma_2, \ldots, \gamma_2\}$ and $r_1 = (r^+ - r_0)/(2\gamma_2)$, except when $r^+ - r_0 = q-1$, in which case it returns $r_1 = 0$ and decreases r_0 by one. In all cases, $r = 2\gamma_2 r_1 + r_0$ in $\mathbb{F}_q$ and $0 \le r_1 < m_\gamma$. The hint then keeps the high part or adjusts it by one, depending on the sign of the low part:

$$\mathsf{UseHint}(h, r) = \begin{cases} r_1 & h = 0, \\ (r_1 + 1) \bmod m_\gamma & h = 1 \text{ and } r_0 > 0, \\ (r_1 - 1) \bmod m_\gamma & h = 1 \text{ and } r_0 \le 0. \end{cases}$$

Blocks of length γ_2. The output of UseHint only depends on the hint h, on the high part r_1, and on the sign of the low part r_0; the last two pieces of information have a simple description. Since $q - 1 = 2m_\gamma\gamma_2$, the integers $1 - \gamma_2, \ldots, q - 1$ split into the $2m_\gamma + 1$ consecutive blocks

$$B_j = \{\gamma_2(j-1) + 1, \ldots, \gamma_2 j\}, \qquad 0 \le j \le 2m_\gamma,$$

of γ_2 integers each (Figure 2). Every $r \in \mathbb{F}_q$ has a representative in at least one block, and in two blocks, B_0 and B_{2m_γ}, exactly when $r^+ \in \{q + 1 - \gamma_2, \ldots, q - 1\}$. As shown below, the high part of r is $\lfloor j/2 \rfloor \bmod m_\gamma$, and its low part is positive exactly when j is odd. The output of UseHint is therefore given by the function

$$F_{m_\gamma}(j, h) = \begin{cases} (j/2 - h) \bmod m_\gamma & j \text{ even}, \\ ((j-1)/2 + h) \bmod m_\gamma & j \text{ odd}, \end{cases} \tag{34}$$

which adds the hint to the high part on a block where the low part is positive, and subtracts it on the other blocks.

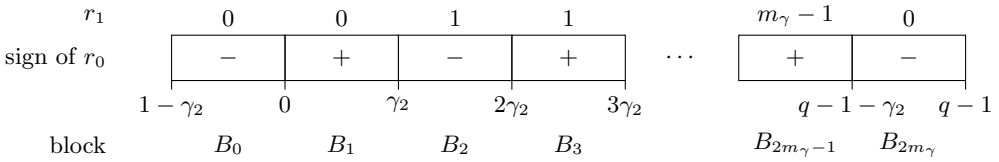

Fig. 2: The blocks B_j of length γ_2, with the extreme integers of some of them. For r with a representative in B_j, the high part is $r_1 = \lfloor j/2 \rfloor \bmod m_\gamma$ (above) and the low part r_0 is positive exactly when j is odd (inside). The first and last blocks overlap modulo q.

Lemma 5 (Correction by Blocks). *Let $r \in \mathbb{F}_q$, $h \in \{0,1\}$, and $0 \le j \le 2m_\gamma$ such that r has a representative in B_j. Then* $\mathsf{UseHint}(h, r) = F_{m_\gamma}(j, h)$.

Proof. Let x be the representative of r in B_j, which is unique since $\gamma_2 < q$. We show that Decompose returns $r_1 = \lfloor j/2 \rfloor \bmod m_\gamma$, with $r_0 > 0$ if and only if j is odd; substituting into UseHint then gives $F_{m_\gamma}(j, h)$.

- If $1 \le j \le 2m_\gamma - 1$, then $1 \le x \le q-1-\gamma_2$, so $x = r^+$. Write $j = 2a+1$ or $j = 2a$, with $a < m_\gamma$. Then $x = 2\gamma_2 a + r_0$ with $r_0 \in \{1, \ldots, \gamma_2\}$ or $r_0 \in \{1-\gamma_2, \ldots, 0\}$ respectively, so that $r_0 = r^+ \bmod^\pm 2\gamma_2$, and $r^+ - r_0 = 2\gamma_2 a < q - 1$: Decompose returns (a, r_0).
- If $j = 0$, then $1 - \gamma_2 \le x \le 0$. For $x = 0$, Decompose returns $(0, 0)$. For $x < 0$, $r^+ = q + x$ has $r^+ \bmod^\pm 2\gamma_2 = x + 1$ and $r^+ - (x+1) = q - 1$, so Decompose returns $(0, x)$. In both cases, $r_1 = 0$ and $r_0 \le 0$.
- If $j = 2m_\gamma$, then $q - \gamma_2 \le x \le q-1$, so $x = r^+$, $r^+ \bmod^\pm 2\gamma_2 = x - q + 1$, and $r^+ - (x - q + 1) = q - 1$: Decompose returns $(0, x - q)$, with $r_1 = 0 = m_\gamma \bmod m_\gamma$ and $r_0 < 0$.

□

When r has representatives in both B_0 and B_{2m_γ}, the lemma applies to both blocks, which indeed give the same output $(-h) \bmod m_\gamma$.

What the circuit checks. The circuit lets the prover designate a block B_j containing a representative of r, and the offset t of this representative in the block, that is,

$$r = \gamma_2(j-1) + t + 1 \quad \text{in } \mathbb{F}_q, \qquad 0 \le t < \gamma_2, \qquad 0 \le j \le 2m_\gamma, \tag{35}$$

and then checks the claimed output $w = F_{m_\gamma}(j, h)$ with a lookup. The offset t and the index j are determined as follows.

- *Offset.* Rather than committing to t directly, the prover commits to two limbs $H, L \in \mathbb{F}_q$, from which t is recomposed as
$$t = \Lambda_h H + L, \tag{36}$$
and bounds them with the two-limb technique of Section 5.5, for $N = \gamma_2 - 1$ and public parameters (Λ_h, H_h) satisfying (31) (Table 5). The three range checks (32) then enforce exactly $0 \le t < \gamma_2$, and every offset in this interval admits limbs.
- *Index.* The index j is not committed, but derived affinely from r and t:
$$j = 1 + 2m_\gamma(t + 1 - r) \quad \text{in } \mathbb{F}_q. \tag{37}$$
Since $2m_\gamma\gamma_2 = q - 1$, the inverse of γ_2 modulo q is $-2m_\gamma$, so this expression is equivalent to the equality of (35). The bound $0 \le j \le 2m_\gamma$ is enforced by the lookup below.

The output is checked with a lookup (Section 2.3) on triples, in the public table

$$\mathcal{U}_{m_\gamma} = \big\{ \big(j,\, h,\, F_{m_\gamma}(j, h)\big) : 0 \le j \le 2m_\gamma,\ h \in \{0,1\} \big\} \subset \mathbb{F}_q^3, \tag{38}$$

which has $4m_\gamma + 2$ entries. The circuit checks $(j,\, h,\, w) \in \mathcal{U}_{m_\gamma}$. Since $2m_\gamma < q$, this membership enforces $0 \le j \le 2m_\gamma$ and $w = F_{m_\gamma}(j, h)$. In practice, this lookup is merged with all the range checks of the circuit into a single lookup on pairs (Section 5.7).

Proposition 3 (Exact Hint Correction). *For an ML-DSA parameter set, let the public parameters* (Λ_h, H_h) *satisfy* (31) *for* $N = \gamma_2 - 1$, *with* $2T < q$. *Let* $r \in \mathbb{F}_q$, $h \in \{0,1\}$, *and* $0 \le w < 2^{b_w}$. *With* t, j *defined by* (36) *and* (37), *the range conditions* (32) *on* (H, L) *and the table membership* $(j, h, w) \in \mathcal{U}_{m_\gamma}$ *admit auxiliary values* $H, L \in \mathbb{F}_q$ *if and only if*

$$w = \mathsf{UseHint}(h, r).$$

Proof. Suppose the range conditions and the table membership hold. The limbs give an integer t with $0 \le t < \gamma_2$, and the table gives an integer j with $0 \le j \le 2m_\gamma$ and $w = F_{m_\gamma}(j, h)$. By the index item, (35) holds, so the integer $\gamma_2(j-1)+t+1 \in B_j$ is a representative of r, and Lemma 5 gives $w = \mathsf{UseHint}(h, r)$.

Conversely, assume $w = \mathsf{UseHint}(h, r)$. Choose a block B_j containing a representative of r, and let t be the offset of this representative in the block, so that (35) holds. The offset item gives limbs H, L with $t = \Lambda_h H + L$ that satisfy the three range conditions, and (37) then gives this j. By Lemma 5, $w = F_{m_\gamma}(j, h)$, so $(j, h, w) \in \mathcal{U}_{m_\gamma}$. □

Each coefficient thus requires the auxiliary limbs H, L, three range checks, and one lookup query in $\mathcal{U}_{m_\gamma}$, all affine in committed values. As for the bounds of Section 5.5, their realization as a layered circuit for GKR is described in Section 5.7.

5.7 Common Lookup

The range checks and the hint corrections of the previous subsections are all lookups: each of them requires a value, called a lookup query, to belong to a public table (Section 2.3). This subsection merges their tables into a single one, so that one lookup argument checks all the queries, and describes the corresponding circuit.

Common table and lookup queries. Rather than running a lookup on scalars for the range checks and a lookup on triples for the hint correction, we merge all the lookup queries into a single lookup on pairs, which needs a single fraction tree and a single vector of multiplicities. To this end, we pack the hint h and the output w of a correction query into a single coordinate, which turns the table (38) into

$$\mathcal{U}'_{m_\gamma} = \left\{ \left(j,\ 1 + F_{m_\gamma}(j,h) + 2^{b_w} h\right) : 0 \le j \le 2m_\gamma,\ h \in \{0,1\} \right\} \subset \mathbb{F}_q^2, \tag{39}$$

and the query $(j, h, w) \in \mathcal{U}_{m_\gamma}$ into

$$\left(j,\ 1 + w + 2^{b_w} h\right) \in \mathcal{U}'_{m_\gamma}. \tag{40}$$

Since w and $F_{m_\gamma}(j, h)$ fit in b_w bits and h is a single bit, the packed coordinate is an integer between 1 and $2^{b_w+1} < q$ that determines h and w. This membership is thus equivalent to $(j, h, w) \in \mathcal{U}_{m_\gamma}$. A range check $\mathsf{Range}(x)$, on a value $x \in \mathbb{F}_q$, is represented by the lookup query $(x, 0)$, and the common table is

$$\mathcal{S} = (\mathcal{T} \times \{0\}) \cup \mathcal{U}'_{m_\gamma}. \tag{41}$$

The shift by one makes the second coordinate of the correction entries and queries nonzero. The two parts of $\mathcal{S}$ are therefore disjoint, so $|\mathcal{S}| = T + 4m_\gamma + 3$, and each query can only match an entry of its own part: the query $(x, 0)$ of a range check lies in $\mathcal{S}$ if and only if $x \in \mathcal{T}$, and a correction query lies in $\mathcal{S}$ if and only if it lies in $\mathcal{U}'_{m_\gamma}$.

Table 4: Lookup queries to the common table.

Source	Lookup queries	Count	ML-DSA-44	ML-DSA-65	ML-DSA-87
Response	Limbs (32) of z_i	$3n\ell$	3072	3840	5376
Hint offset	Limbs (32) of t	$3nk$	3072	4608	6144
Hint correction	Query (40)	nk	1024	1536	2048
Acceptance	Slacks e_i	B_c	213	213	213
Memory	t_g and $2g - 1 - t_g$	2τ	78	98	120
Hint weight	$\omega - \sum_i h_i$	1	1	1	1
Total		N_Q	7460	10296	13902

Table 4 lists the queries of the common lookup, by source. Their total number is $N_Q = 3n\ell + 4nk + B_c + 2\tau + 1$. The lookup of Section 2.3 requires $N_Q < q$, so that the multiplicities are not reduced modulo q. Neither N_Q nor $|\mathcal{S}|$ depends on whether the verification key is public (Section 5.10): the lookup queries only involve the response, the hint, and the challenge. The table has $|\mathcal{S}| = T + 1 + 4m_\gamma + 2$ entries. Table 5 gives the bound T, the limb parameters of Sections 5.5 and 5.6, and the resulting sizes of the lookup for the three ML-DSA parameter sets; in all cases, $N_Q < q$ holds.

Table 5: Range parameters and size of the common lookup.

	ML-DSA-44	ML-DSA-65	ML-DSA-87
Range bound T	560	5430	1484
Response limbs (Λ_z, H_z)	$(561, 466)$	$(5431, 192)$	$(997, 1050)$
Hint-offset limbs (Λ_h, H_h)	$(201, 471)$	$(337, 761)$	$(429, 607)$
Table size $\lvert\mathcal{S}\rvert$	739	5497	1551
Number of lookup queries N_Q	7460	10296	13902

Lookup circuit. Let $\mathbf{X}_1, \ldots, \mathbf{X}_{N_Q}$ be the lookup queries collected above. The values determining the queries and the multiplicities $(m_{\mathbf{u}})_{\mathbf{u}\in\mathcal{S}}$ are committed before the verifier samples $\alpha \xleftarrow{\$} \mathbb{K} \setminus V_2$, where $V_2 = \psi(\mathbb{F}_q^2)$ is the image of the compression (2) of pairs. The lookup (5) with $\nu = 2$ then checks all memberships with soundness error at most

$$\frac{N_Q + |\mathcal{S}| - 1}{|\mathbb{K}| - q^2}. \tag{42}$$

Since the table is public, each $\varphi_{\mathbf{u}} = (\alpha + \psi(\mathbf{u}))^{-1}$ is a public coefficient once α is known, and the table contribution

$$\Phi = \sum_{\mathbf{u}\in\mathcal{S}} m_{\mathbf{u}}\varphi_{\mathbf{u}} \tag{43}$$

is affine in the committed multiplicities. Using the fraction representation of Section 2.3, we form the leaves

$$(1, \alpha + \psi(\mathbf{X}_i)) \quad (1 \le i \le N_Q), \qquad (-\Phi, 1).$$

The first N_Q leaves are the fractions $1/(\alpha + \psi(\mathbf{X}_i))$ of the lookup queries, and the last one is the table contribution $-\Phi$, seen as a fraction of denominator 1. A balanced tree adds these fractions. Since $\alpha \notin V_2$, all denominators are nonzero, so a zero numerator at the root is equivalent to (5) for the common table $\mathcal{S}$.

The leaves of this fraction tree are affine in committed values. Indeed, Φ is a linear combination of the committed multiplicities by (43), and every lookup query is affine in committed values: the limbs $z_{1,i}, z_{0,i}, H, L$, the slacks e_i, and the times t_g are committed; the other range arguments $H_{\max} - H$, $2g - 1 - t_g$, and $\omega - \sum_i h_i$ are affine in committed values; j is affine in H, L, and the committed $\mathbf{w}'_{\mathrm{Approx}}$; w is recomposed from committed bits; and h is a committed bit. After padding with the neutral fraction $(0, 1)$, the tree has depth $\lceil \log_2(N_Q + 1) \rceil$, that is, 13 for ML-DSA-44, and each layer applies the addition law of Section 2.3, which has degree two. The output of the circuit is the numerator at the root, which must be zero.

This fraction tree is the largest single layered circuit of $C^{(q)}$ (Table 8). Its $N_Q + 1$ leaves gather all the range and hint-correction checks of the circuit, whereas the product trees of the multiset equalities have at most $\max\{B_A, n + \tau\}$ factors per side, that is, 298 for ML-DSA-44, and the other circuits are copies of constant-size gadgets. With a public verification key, it is also the deepest layered circuit, since the decoding and the ordered selections of the matrix candidates (Section 5.2) disappear.

5.8 Consistency Tag

The prime-field side of the consistency test of Section 3.3 checks

$$\mathsf{tag}_R\big(u^{(q)}_{\mathrm{cross}}; \kappa^{(q)}\big) = y,$$

where $u^{(q)}_{\text{cross}} \in \mathbb{F}_2^{N_{\text{cross}}}$ is the vector encoded by $w^{(q)}_{\text{cross}}$ and y is the public tag (13). Each bit of the tag is a parity, which $C^{(q)}$ expresses as an integer equation with a quotient. Unlike all the other auxiliary values of the circuit, these quotients depend on a challenge, the matrix R, and thus cannot be committed with the private inputs. We first express the parities over $\mathbb{F}_q$, and then explain how the prover provides the quotients.

Parity over the prime field. Write b_i for the i-th bit of $u^{(q)}_{\text{cross}}$: either an entry of $w^{(q)}_{\text{cross}}$ in Boolean form, or a decoded bit of the matrix stream (Section 5.1). The mask $\kappa^{(q)}$ also consists of bits by its Boolean constraints. Since the entries of $R \in \mathbb{F}_2^{N_{\text{tag}} \times N_{\text{cross}}}$ are 0 or 1, row j of R defines the integer

$$\hat{S}_j = \kappa^{(q)}_j + \sum_{i=1}^{N_{\text{cross}}} R_{j,i} b_i,$$

whose parity is the j-th bit of $\mathsf{tag}_R\big(u^{(q)}_{\text{cross}}; \kappa^{(q)}\big)$. This parity is y_j if and only if there is a nonnegative integer Q_j such that

$$\hat{S}_j = y_j + 2Q_j,$$

namely $Q_j = \lfloor \hat{S}_j/2 \rfloor$. Since $\hat{S}_j \le N_{\text{cross}} + 1$, this quotient fits in

$$b_Q = \mathsf{bitlen}\left(\left\lfloor \frac{N_{\text{cross}} + 1}{2} \right\rfloor \right) \tag{44}$$

bits, whatever R, and we represent Q_j by b_Q bits. Since N_{cross} is smaller than $q/2$, this representation raises no issue. Indeed, $2^{b_Q} \le N_{\text{cross}} + 1$ by definition of b_Q, so $\hat{S}_j \le N_{\text{cross}} + 1$ and $y_j + 2Q_j \le 2^{b_Q+1} - 1 \le 2N_{\text{cross}} + 1$ both lie in $[0, q-1]$, for any quotient of b_Q bits. Their equality in $\mathbb{F}_q$ thus implies their equality over the integers, and enforces the parity.

Quotients chosen after the commitments. Everywhere else in this paper, the auxiliary values, such as limbs, prefix counts, or multiplicities, only depend on the private inputs, and the prover commits to them with these inputs. The quotients Q_j are the only exception: they depend on R, which the verifier samples after the commitments (Section 3.4). The prover can thus neither commit to them, nor announce them in the clear, since they would leak information on $u^{(q)}_{\text{cross}}$. We therefore mask them with a one-time pad committed in advance. The prime-field instance commits to a vector $\mathsf{pad} = (\mathsf{pad}_{j,a})$ of independent uniform pad bits, for $1 \le j \le N_{\text{tag}}$ and $0 \le a < b_Q$, which belongs to $\mathsf{aux}^{(q)}$. The number of pads must be fixed at this point, before R is known: this is why b_Q is derived from the bound $\hat{S}_j \le N_{\text{cross}} + 1$, which holds for every R. After receiving R, the honest prover computes the quotients and announces the public bit string $e = (e_{j,a})$ defined by

$$e_{j,a} = \mathsf{pad}_{j,a} \oplus \mathrm{bit}_a(Q_j),$$

where bit_a denotes the bit of weight 2^a. The circuit reconstructs the quotient bits as

$$d_{j,a} = e_{j,a} + (1 - 2e_{j,a})\, \mathsf{pad}_{j,a},$$

and checks

$$\kappa^{(q)}_j + \sum_{i=1}^{N_{\text{cross}}} R_{j,i} b_i - y_j - 2 \sum_{a=0}^{b_Q - 1} 2^a d_{j,a} = 0 \qquad \text{in } \mathbb{F}_q. \tag{45}$$

The pads are bits by their Boolean constraints (Section 5.1). Since $e_{j,a}$ is a public bit, $d_{j,a}$ is then also a bit, namely $\mathsf{pad}_{j,a} \oplus e_{j,a}$, and is affine in the committed pad.

Lemma 6 (Parity Equations). *Assume that the values b_i, the mask $\kappa^{(q)}$, and the pads are bits. Let $R \in \mathbb{F}_2^{N_{\text{tag}} \times N_{\text{cross}}}$ and $y \in \mathbb{F}_2^{N_{\text{tag}}}$.*

- *If some $e \in \mathbb{F}_2^{N_{\text{tag}} \times b_Q}$ satisfies (45) for all rows, then $\mathsf{tag}_R\big(u^{(q)}_{\text{cross}}; \kappa^{(q)}\big) = y$.*
- *Conversely, if $\mathsf{tag}_R\big(u^{(q)}_{\text{cross}}; \kappa^{(q)}\big) = y$, then for any pad bits, the honest announcements e satisfy (45) for all rows.*

Proof. For the first item, the bits $d_{j,a}$ define an integer $Q_j = \sum_a 2^a d_{j,a}$ of b_Q bits, and (45) reads $\hat{S}_j = y_j + 2Q_j$ in $\mathbb{F}_q$. By the discussion above, this equality holds over the integers, so the parity of $\hat{S}_j$ is y_j for every row. For the second item, the parity of $\hat{S}_j$ is y_j, so $\hat{S}_j = y_j + 2Q_j$ with $Q_j = \lfloor \hat{S}_j/2 \rfloor$ of b_Q bits. The honest announcements give $d_{j,a} = \mathrm{bit}_a(Q_j)$, hence (45). □

The announcements do not reveal more than the tag. In an honest execution, the prover uses a common fresh uniform tag mask $\kappa^{(2)} = \kappa^{(q)}$ and fresh independent uniform pads. For fixed R and cross-field witness, y is uniform, and the pads make e uniform even conditioned on y. The public announcements (y, e) can thus be simulated by uniform bits; the zero-knowledge of the whole protocol is argued in Theorem 1.

Layered circuits. Once R, y, and e are public, each equation (45) is affine in the mask, the pads, and the bits b_i. The bits in Boolean form are committed. The decoded bits of the matrix stream are outputs of the decoding circuit of Section 5.1, so the equation is a linear claim on these outputs and on committed values, as for the other outputs of the layered circuits of $C^{(q)}$. Beyond the Boolean constraints, the consistency test thus needs no layered circuit unless packing is used. In that case, its equations are included in the affine GKR layer following the decoding.

5.9 Completeness and Soundness

We now combine the previous subsections to show that $C^{(q)}$ satisfies the specification (15) of Section 3.5. Most constraints of $C^{(q)}$ are deterministic. Its randomized tests are the ordered selections of the matrix entries, one per stream f, the ordered selection of the accepted challenge bytes, the memory conservation of the challenge array, and the common lookup. Their challenges $\mathsf{ch}^{(q)}$ are sampled after the commitments, so that the private inputs, including the auxiliary values, the multiplicities, and the pads, are fixed beforehand; only the public values y and e are chosen afterwards.

Table 6 gives the soundness error of each randomized test, that is, the probability that it accepts when the property it checks is false, from the bounds of Section 2.3. The common lookup gathers all range and hint-correction checks, including the time constraints of the memory check, so the memory row only concerns its conservation law.

Table 6: Soundness errors of the randomized tests of $C^{(q)}$.

Test	Soundness error
Common lookup (42)	$(N_Q + \lvert\mathcal{S}\rvert - 1)/(\lvert\mathbb{K}\rvert - q^2)$
Matrix selection (per stream)	$n/\lvert\mathbb{K}\rvert$
Challenge selection	$\tau/\lvert\mathbb{K}\rvert$
Memory conservation	$(n+\tau)/\lvert\mathbb{K}\rvert$

Since $N_Q > n + \tau$, the soundness error of the common lookup dominates the others, and we set

$$\epsilon^{(q)} = \frac{N_Q + |\mathcal{S}| - 1}{|\mathbb{K}| - q^2}. \tag{46}$$

The consistency test itself is not a test of $C^{(q)}$: its soundness error $2^{-N_{\mathrm{tag}}}$, over the choice of R, is accounted for in Lemma 3. On the side of $C^{(q)}$, the tag equality is checked without error, by Lemma 6.

Proposition 4 (Prime-Field Circuit). *Assume that the conditions on T and on the parameters stated at the beginning of Section 5 hold. Then $C^{(q)}$ satisfies the specification* (15) *with the soundness error* (46)*, whether the verification key is hidden or public.*

Proof. We treat the case of a hidden verification key; with a public one, the matrix entries are public constants, and the same arguments apply without the matrix expansion (Section 5.10). Fix R.

Completeness. Assume that $w_{\text{cross}}^{(q)}$ is a valid encoding, that $\kappa^{(q)}$ consists of bits, and that $\mathcal{R}^{(q)}$ holds for $w_{\text{cross}} = \mathsf{Enc}^{-1}(u_{\text{cross}}^{(q)})$ and $w_{\text{in}}^{(q)}$, and let $y = \mathsf{tag}_R\big(u_{\text{cross}}^{(q)}; \kappa^{(q)}\big)$. We choose all the auxiliary values as in the converse directions of Propositions 1, 2, and 3, which include the slacks and the times of the challenge reconstruction, together with the response limbs of Section 5.5, the honest coefficients of $\mathbf{w}'_{\text{Approx}}$, the honest lookup multiplicities, and arbitrary pad bits. None of them depends on the challenges or on R. All deterministic constraints then hold, and so do the properties checked by the randomized tests, which therefore accept for every choice of $\mathsf{ch}^{(q)}$. Finally, the honest announcements e satisfy (45) by Lemma 6.

Soundness. Fix the private inputs, and assume that the circuit accepts for some (y, e). Since the circuit accepts, all its deterministic constraints hold. In particular, $w_{\text{cross}}^{(q)}$ is a valid encoding and $\kappa^{(q)}$ and the pads consist of bits, by the Boolean and subgroup constraints, and the tag equality then holds by Lemma 6: this is the part of the soundness that holds without error. Assume now that $\mathcal{R}^{(q)}$ does not hold. If all the properties checked by the randomized tests held, the deterministic constraints and Propositions 1, 2, and 3, together with the bounds of Section 5.5, would give $\mathcal{R}^{(q)}$. One of these properties is therefore false. It only depends on the private inputs, and not on (y, e), so we can fix it before $\mathsf{ch}^{(q)}$ is sampled. The circuit accepts only if the corresponding test accepts, which happens with probability at most its soundness error in Table 6, hence at most $\epsilon^{(q)}$. This argument only uses one false property: it requires neither a union bound over the $k\ell$ matrix selections nor independence between the tests. □

5.10 Public Verification Key

When the verification key is public, the verifier derives $\widehat{\mathbf{A}} = \mathsf{ExpandA}(\rho)$ from it, without budget (Section 4.5), and $\widehat{\mathbf{A}}$ and $\mathbf{t}_1$ become public constants of $C^{(q)}$. As noted in Section 3.2, the matrix stream S_A and the key coefficients $\mathbf{t}_1$ then leave the cross-field witness, which reduces to $(S_c, \mathbf{w}'_1)$. Section 5.2 is then removed entirely: there are no matrix candidates, selection bits, or claimed matrix entries to commit, and no packed element to decode. The verification equation (29) also becomes affine: $\widehat{A}_{r,s,i}$ and $\widehat{t}_{1,r,i}$ are public, so each term is a public multiple of a coefficient of $\widehat{\mathbf{z}}$ or $\widehat{c}$. Its multiplication layer disappears, and $\mathbf{w}'_{\text{Approx}}$ is affine in the response limbs and the final versions of c, so it no longer needs to be committed. The consistency test only covers the remaining bits, so that $N_{\text{cross}} = 8B_c + \tau + nkb_w$: the Boolean constraints and the parity terms on $\mathbf{t}_1$ and S_A disappear, and the width (44) of the quotients decreases accordingly. The lookup queries are unchanged (Section 5.7). The layered circuits of $C^{(q)}$ then reduce to the Boolean constraints, the challenge reconstruction, and the fraction tree of the common lookup. Proposition 4 applies with the same soundness error (46); only the row of the matrix selections disappears from Table 6. A public message does not change $C^{(q)}$.

5.11 Circuit Cost

Whether the verification key is hidden or public, the cross-field witness is the largest group of committed values. Keeping the verification key hidden more than triples the number of committed values for ML-DSA-44, mostly because of the key coefficients $\mathbf{t}_1$ and of the matrix expansion: the matrix stream S_A, the prefix counts, and the claimed entries. The matrix expansion also forms the deepest layered circuit, namely the decoding of S_A followed by the ordered selections. With a public verification key, the largest layered circuit is the fraction tree of the common lookup (Section 5.7).

Table 7 lists the values committed by $C^{(q)}$; all other values, such as the selection bits, $\mathbf{z}$, t, j, and the NTT coefficients, are affine in them.

Table 7: Values committed by $C^{(q)}$ for one ML-DSA verification. The rows marked * disappear when the verification key is public.

			ML-DSA-44	
Source	Committed values	Count	with hidden pk	with public pk
Cross-field witness	Entries of $w^{(q)}_{\text{cross}}$ (packed for S_A)	$N_{\text{cross}} - 21k\ell B_A$	32431	7887
Hint	$\mathbf{h}$	nk	1024	1024
Response	Limbs $z_{1,i}, z_{0,i}$	$2n\ell$	2048	2048
Matrix*	Prefix counts $N^A_{f,i}$	$k\ell(B_A - 1)$	4752	–
Matrix*	Claimed entries $\widehat{\mathbf{A}}_f$	$nk\ell$	4096	–
Challenge	Prefix counts N_i and slacks e_i	$2B_c - 1$	425	425
Challenge	Accepted bytes J_g	τ	39	39
Challenge	Deleted versions (v_g, t_g)	2τ	78	78
Challenge	Final versions $(c[p], t^{\text{fin}}_p)$	$2n$	512	512
Verification*	Coefficients of $\mathbf{w}'_{\text{Approx}}$	nk	1024	–
Hint correction	Limbs H, L of the hint offset	$2nk$	2048	2048
Lookup	Multiplicities $m_{\mathbf{u}}$	$\lvert\mathcal{S}\rvert$	739	739
Consistency	Mask $\kappa^{(q)}$ and pads pad	$N_{\text{tag}}(1 + b_Q)$	2304	1664
Total			51520	16464

With a hidden verification key, the N_{cross} bits of the cross-field witness include the $24k\ell B_A$ bits of the matrix stream S_A, which are packed into $3k\ell B_A$ elements, so that $w^{(q)}_{\text{cross}}$ has $N_{\text{cross}} - 21k\ell B_A$ entries. With a public verification key, nothing is packed, the row then counts $N_{\text{cross}} = 8B_c + \tau + nkb_w$ entries, and the quotient width b_Q decreases accordingly. The numerical values use $N_{\text{tag}} = 128$ and the parameters of Table 5, which give $N_{\text{cross}} = 132559$ and $b_Q = 17$ with a hidden verification key, and $N_{\text{cross}} = 7887$ and $b_Q = 12$ with a public verification key.

Table 8 lists the layered circuits proved with GKR, with the number of identical copies of each that are proved in parallel, and their depth, counted in multiplication layers. Unlike in $C^{(2)}$, no intermediate layer is committed. The values inside the trees depend on the challenges of the randomized tests, so they cannot be committed with the private inputs, and cutting a tree would require masking each cut value with a random element of $\mathbb{K}$ committed in advance. In our implementation, this overhead does not pay off.

Table 8: Layered circuits of $C^{(q)}$. The rows marked * disappear when the verification key is public.

Layered circuit	# Parallel copies	Depth
Boolean constraints	N_{bool}	1
Decoding of the matrix stream* (21)	$3k\ell B_A$	$2s - 2$
composed with the selection constraints* (22)	$k\ell B_A$	$+1$
and the ordered selections* (B_A against n factors)	$k\ell$	$+1 + \lceil \log_2 B_A \rceil$
Acceptance constraints (26)	B_c	2
Challenge ordered selection (B_c against τ factors)	1	$1 + \lceil \log_2 B_c \rceil$
Memory conservation ($n + \tau$ factors per side)	1	$\lceil \log_2(n + \tau) \rceil$
Verification products* (29)	$nk\ell + nk$	1
Lookup fraction tree ($N_Q + 1$ leaves over $\mathbb{K}$)	1	$\lceil \log_2(N_Q + 1) \rceil$

In this table, N_{bool} denotes the number of bits given in Boolean form (Section 5.1): $N_{\text{bool}} = N_{\text{cross}} - 23k\ell B_A + nk + B_c + N_{\text{tag}}(1 + b_Q)$ with a hidden verification key, i.e., 26436 for ML-DSA-44, and $N_{\text{bool}} = N_{\text{cross}} + nk + B_c + N_{\text{tag}}(1 + b_Q)$ with a public one, i.e., 10788.

6 Applications

The protocol of the previous sections proves the validity of an ML-DSA signature with a verification key and a message that may each be public or hidden, and it can be combined with further statements on the same committed values. It thus applies to any system in which a standard ML-DSA signature must be shown without being revealed. We illustrate this with three applications of increasing complexity: a proof of knowledge of a signature on a hidden message, a blind signature, and anonymous credentials with device binding. For each of them, we describe the proven statement, the resulting circuits, and the proof sizes and running times for the three ML-DSA parameter sets.

Implementation. We build on the proof-of-concept SmallWood$^+$ backend provided by its authors [FPR26], written in C and Python. The C library provides the arithmetic core, including finite-field operations, commitments, and masked sum-checks. The Python layer calls these routines to assemble the proof protocols and manage their witnesses and transcripts.

Our proof-of-concept implements the three ML-DSA applications in Python for all parameter sets. It constructs our binary and prime-field verification circuits and prepares their witnesses. It implements the complete protocol, including witness commitments, shared challenge derivation, consistency announcements, and the generation and verification of both field proofs. The test drivers use the Python `cryptography` library to generate keys and signatures, then run proof generation and verification with SmallWood$^+$ and record proof sizes and running times.

Only the Keccak circuits are proven with dedicated C routines, which exploit their fixed structure. All the other layered circuits, over both $\mathbb{F}_2$ and $\mathbb{F}_q$, use generic GKR code, with C arithmetic kernels but a protocol orchestration in Python. The reported timings mostly reflect this difference of optimization, rather than the overall cost of the circuits. In standalone benchmarks, the dedicated implementation proves and verifies 10 Keccak permutations in 62.2 and 16.1 ms, and 100 permutations in 270.9 and 63.1 ms. In comparison, the binary instance of the ML-DSA-44 proof of knowledge, whose circuit essentially consists of 10 permutations, takes 308 and 193 ms (Table 9): the Keccak permutations, which form the bulk of the binary computation, account for about a fifth of its proving time and less than a tenth of its verification time. The rest of this time, and all the time of the prime-field instance, is spent in the generic code. We therefore expect dedicated implementations of the other layered circuits to bring significant speedups for both instances. All timings were measured with our single-threaded implementation on an Intel Core Ultra 7 255H processor.

Fiat–Shamir grinding. We use symmetric grinding to bound the bad-challenge probability by $2^{-\lambda}$ per fresh classical random-oracle trial, with $\lambda = 128, 192, 256$ for the three ML-DSA parameter sets. The joint draw of R and the arithmetic challenges after both commitments uses four grinding bits for ML-DSA-44 and none for the other two parameter sets. The binary SmallWood$^+$ instance uses up to five grinding bits per round; the prime-field instance uses none. Both parties recompute the counter from the public transcript, so it adds no witness or proof bytes. The measurements include grinding. Appendix A gives the bounds.

6.1 Proof of Knowledge of an ML-DSA Signature

Proven relation. The verification key pk is public. The prover shows knowledge of a message M and a valid signature under pk, according to the public-pk variant of (11). The message length is public, while the message and the signature remain hidden.

Circuit specialization. Since the verification key is public, its hash and matrix expansion can be computed by the verifier. Since ExpandA is evaluated outside the circuits, circuit size imposes no bound on the number of candidates it reads. These computations leave the binary circuit, and the verification equation in the prime-field circuit becomes affine, as described in Sections 4.5 and 5.10. The binary circuit still computes the message digest, since the message is hidden, as well as the hashes involving the signature.

Arithmetic soundness. The prime-field error $\epsilon^{(q)}$ is given by (46). For ML-DSA-44, ML-DSA-65, and ML-DSA-87, respectively, we use $\mathbb{K} = \mathbb{F}_{q^6}, \mathbb{F}_{q^9}, \mathbb{F}_{q^{12}}$, $N_{\text{tag}} = 128, 192, 256$, and the range parameters of Table 5. The tables report the interactive arithmetic error $\max\{\epsilon^{(q)}, 2^{-N_{\text{tag}}}\}$ of Lemma 3.

Performance. We evaluate the proof with a public verification key on hidden 32-byte messages. Table 9 gives the results for the three ML-DSA parameter sets. Each field proof includes its protocol messages; the proof size also includes the consistency announcements. The proof size of $\mathsf{SmallWood}^+$ slightly varies from one execution to another: the authentication paths of the opened leaves are merged in the Merkle tree, so that their total size depends on the opened positions. For each configuration, we report the proof with median total size among three executions. The reported times are medians of three executions. The setup rows measure the initial construction of the circuits, of their wiring, and of the proof parameters; these fixed structures could be precomputed and reused. The proving and verification times cover the remaining per-instance work, including the challenge-dependent computations, which must be performed for each proof.

Table 9: Performance of the ZK-PoK of an ML-DSA signature.

	ML-DSA-44	ML-DSA-65	ML-DSA-87
Interactive arithmetic error	$\approx 2^{-124.99}$	2^{-192}	2^{-256}
Security with grinding (bits)	≥ 128	≥ 192	≥ 256
Proof over $\mathbb{F}_2$ (KB)	29.6	51.1	80.7
Proof over $\mathbb{F}_q$ (KB)	56.6	108.5	157.7
Consistency announcements (KB)	0.2	0.3	0.4
Proof size (KB)	**86.5**	**160.0**	**238.8**
Setup over $\mathbb{F}_2$ (s)	0.1	0.1	0.2
Setup over $\mathbb{F}_q$ (s)	1.0	2.7	5.8
Prove over $\mathbb{F}_2$ (s)	0.3	0.6	1.2
Prove over $\mathbb{F}_q$ (s)	1.8	5.5	10.4
Proving time (s)	**2.1**	**6.1**	**11.6**
Verify over $\mathbb{F}_2$ (s)	0.2	0.3	0.5
Verify over $\mathbb{F}_q$ (s)	1.6	4.6	10.4
Verification time (s)	**1.7**	**4.9**	**10.9**

6.2 Blind Signature

Our blind signature follows the framework of Fischlin [Fis06]: the signer signs a commitment to the message, and the blind signature proves knowledge of a signature on such a commitment. In Fischlin's construction, the user additionally encrypts the commitment and the signature, which allows the security reduction to extract them. This is not needed here, since our argument of knowledge comes with a straight-line extractor in the random oracle model. Moreover, the commitment is a hash of the message with a random string, so that no proof of well-formedness is required at issuance.

Issuance. The signer's verification key pk is public. For a message M, the requester samples a fresh uniform randomizer $r \in \{0,1\}^{2\lambda}$ and computes $D = H(M\|r, \lambda/4)$. It sends D to the signer, which returns an ML-DSA signature on D. The requester checks this signature under pk.

Blind signature generation. The requester proves knowledge of an opening r of the digest $D = H(M\|r, \lambda/4)$ and a valid signature on D under pk. The message M is public, while r, D, and the issued signature remain hidden. This proof is the blind signature on M.

Circuit extension. The binary circuit additionally computes D from the public M and the hidden r. The prime-field circuit and cross-field witness are unchanged, so the arithmetic error is the same as for the proof of knowledge.

Performance. We evaluate the construction on 32-byte messages; computing D then costs one Keccak permutation for each of the three parameter sets. Table 10 gives the blind signature sizes and the signing

and verification times, excluding issuance. The small differences between the prime-field proof sizes of Tables 9 and 10 are only due to the variation of the proof size between executions.

Table 10: Performance of the blind signature.

	ML-DSA-44	ML-DSA-65	ML-DSA-87
Interactive arithmetic error	$\approx 2^{-124.99}$	2^{-192}	2^{-256}
Security with grinding (bits)	≥ 128	≥ 192	≥ 256
Proof over $\mathbb{F}_2$ (KB)	30.9	52.5	82.2
Proof over $\mathbb{F}_q$ (KB)	56.7	109.1	157.9
Consistency announcements (KB)	0.2	0.3	0.4
Blind signature size (KB)	**87.9**	**161.9**	**240.6**
Setup over $\mathbb{F}_2$ (s)	0.1	0.1	0.2
Setup over $\mathbb{F}_q$ (s)	1.0	2.7	6.0
Prove over $\mathbb{F}_2$ (s)	0.3	0.8	1.3
Prove over $\mathbb{F}_q$ (s)	1.7	5.6	10.5
Signing time (s)	**2.0**	**6.4**	**11.8**
Verify over $\mathbb{F}_2$ (s)	0.2	0.4	0.5
Verify over $\mathbb{F}_q$ (s)	1.5	4.7	8.9
Verification time (s)	**1.7**	**5.0**	**9.4**

6.3 Anonymous Credentials with Device Binding

We consider anonymous credentials based on the mdoc structure of ISO/IEC 18013-5 [II21], adapted to ML-DSA and SHA3 item digests. The proven statement follows the mdoc presentation of Frigo and shelat [Fas26, Algorithm 10], with two changes of algorithms. We replace ECDSA by ML-DSA, for both the issuer and the device, to obtain post-quantum credentials. Since the format has to change anyway, we also replace SHA-256 by SHA3 for the item digests: it is the most recent hash standard and the one underlying ML-DSA, so that all the hash computations of the statement rely on the Keccak permutation. A credential certifies attributes of its holder and binds them to a device verification key. The holder proves that a requested attribute was certified by the issuer, that the credential is currently valid, and that the certified device authenticated the verifier's session, without revealing the credential or the device verification key.

Issuance. After checking the holder's attributes, the issuer encodes each of them with a fresh salt and computes its digest. The Mobile Security Object (MSO) contains these digests, the device verification key pk_D, and the validity dates. The issuer signs the MSO. As prescribed by the standard, each signature is actually computed on a COSE signature structure, which wraps the signed payload with a fixed header; we leave this wrapping implicit in the following. Both the issuer and the device use the same ML-DSA parameter set.

Presentation. The profile fixes the verifier's identity and the requested attribute. For each presentation, the verifier specifies a public session containing its identity and a fresh nonce. The device signs the corresponding authentication message. The public statement consists of the issuer verification key pk_I, the session, the requested attribute, and the verification time t_{now}.

The holder proves knowledge of an MSO, a salt, and two ML-DSA signatures satisfying the following conditions:

1. The issuer's signature on the MSO is valid under pk_I.
2. The MSO contains the digest of the requested attribute, computed from its encoding and a private salt.
3. The device's signature on the session authentication message is valid under the verification key pk_D contained in the MSO.

4. The validity dates in the MSO are well-formed UTC timestamps, with $t_{\text{from}} < t_{\text{until}}$, and satisfy $t_{\text{from}} \leq t_{\text{now}} \leq t_{\text{until}}$.

The MSO, device verification key, salt, validity dates, and both signatures remain hidden.

Circuit composition. The relation combines two ML-DSA verifications with different public inputs. The issuer verification uses a public verification key and a hidden message, while the device verification uses a hidden verification key and a public authentication message. The same bits of pk_D are used in the signed MSO and in the device verification, ensuring that the session is authenticated under the certified verification key. Only the device verification requires a private matrix expansion. The two verifications are combined into one binary circuit and one prime-field circuit, sharing a single lookup table and one consistency test on their combined cross-field witness. The attribute digest and validity checks are performed in the binary circuit.

Arithmetic soundness. Each ML-DSA verification contributes $N_Q = 3n\ell + 4nk + B_c + 2\tau + 1$ queries to the prime-field lookup. Both verifications use the same public table $\mathcal{S}$, so the combined circuit performs $2N_Q$ lookup queries without duplicating the table. The lookup remains the dominant prime-field test, giving

$$\epsilon^{(q)} = \frac{2N_Q + |\mathcal{S}| - 1}{|\mathbb{K}| - q^2}.$$

Instantiation. We evaluate a proof-of-age credential: a fixed mdoc profile whose MSO contains a single attribute digest, for the item `age_over_18=true`. This proof of age, also the use case evaluated in [Fas26], is one example: disclosing several attributes, or proving a predicate on a hidden attribute value, only adds item digests and a small predicate circuit to the binary circuit, with a marginal impact on the proof size and timings. Likewise, a credential certifying more attributes only lengthens the MSO hashed for the issuer's signature: each additional digest adds at most about 70 bytes, that is, about half a SHAKE256 block, to a binary circuit of 140 Keccak permutations for ML-DSA-44. The item digest uses SHA3-256, SHA3-384, or SHA3-512, with a salt of 16, 24, or 32 bytes for ML-DSA-44, ML-DSA-65, or ML-DSA-87, respectively. The layout of the MSO, encoded in CBOR [BH20], is fixed, so the digest, device verification key, and validity dates occupy public positions in the hidden MSO. In contrast, the circuit of [Fas26] handles an MSO of hidden length, at most 2231 bytes in their evaluation, in which the fields are located with a parsing witness. Supporting such general layouts would only extend our binary circuit.

Performance. Table 11 reports the presentation proof sizes and the presentation and verification times for this profile.

Table 11: Performance of the anonymous credentials with device binding.

	ML-DSA-44	ML-DSA-65	ML-DSA-87
Interactive arithmetic error	$\approx 2^{-124.06}$	2^{-192}	2^{-256}
Security with grinding (bits)	≥ 128	≥ 192	≥ 256
Proof over $\mathbb{F}_2$ (KB)	78.2	140.5	229.8
Proof over $\mathbb{F}_q$ (KB)	121.2	219.0	349.1
Consistency announcements (KB)	0.3	0.4	0.6
Presentation proof size (KB)	**199.7**	**359.9**	**579.5**
Setup over $\mathbb{F}_2$ (s)	1.7	2.9	5.2
Setup over $\mathbb{F}_q$ (s)	7.7	17.0	34.8
Prove over $\mathbb{F}_2$ (s)	3.2	11.9	27.8
Prove over $\mathbb{F}_q$ (s)	12.2	32.8	75.8
Presentation time (s)	**15.5**	**44.0**	**104.4**
Verify over $\mathbb{F}_2$ (s)	2.4	5.4	11.3
Verify over $\mathbb{F}_q$ (s)	11.1	29.2	65.0
Verification time (s)	**13.6**	**34.9**	**76.8**

References

BB26. Carsten Baum and Marvin Beckmann. edaBits in the exponent (edaBitE) and issuer-hiding blind signatures from ML-DSA and VOLE-in-the-head. Cryptology ePrint Archive, Paper 2026/2473, 2026.

BBC+24. Carsten Baum, Olivier Blazy, Jan Camenisch, Jaap-Henk Hoepman, Eysa Lee, Anja Lehmann, Anna Lysyanskaya, René Mayrhofer, Hart Montgomery, Ngoc Khanh Nguyen, Bart Preneel, abhi shelat, Daniel Slamanig, Stefano Tessaro, Søren Eller Thomsen, and Carmela Troncoso. Cryptographers' feedback on the EU digital identity's ARF. Technical report, version 1.0, June 2024.

BBS04. Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 41–55. Springer, Berlin, Heidelberg, August 2004.

BG12. Stephanie Bayer and Jens Groth. Efficient zero-knowledge argument for correctness of a shuffle. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 263–280. Springer, Berlin, Heidelberg, April 2012.

BH20. Carsten Bormann and Paul Hoffman. Concise binary object representation (CBOR). RFC 8949, 2020.

BLNS23. Jonathan Bootle, Vadim Lyubashevsky, Ngoc Khanh Nguyen, and Alessandro Sorniotti. A framework for practical anonymous credentials from lattices. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part II*, volume 14082 of *LNCS*, pages 384–417. Springer, Cham, August 2023.

CCH+19. Ran Canetti, Yilei Chen, Justin Holmgren, Alex Lombardi, Guy N. Rothblum, Ron D. Rothblum, and Daniel Wichs. Fiat-Shamir: from practice to theory. In Moses Charikar and Edith Cohen, editors, *51st ACM STOC*, pages 1082–1090. ACM Press, June 2019.

CDvD+03. Dwaine E. Clarke, Srinivas Devadas, Marten van Dijk, Blaise Gassend, and G. Edward Suh. Incremental multiset hash functions and their application to memory integrity checking. In Chi-Sung Laih, editor, *ASIACRYPT 2003*, volume 2894 of *LNCS*, pages 188–207. Springer, Berlin, Heidelberg, November / December 2003.

CL01. Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 93–118. Springer, Berlin, Heidelberg, May 2001.

CL03. Jan Camenisch and Anna Lysyanskaya. A signature scheme with efficient protocols. In Stelvio Cimato, Clemente Galdi, and Giuseppe Persiano, editors, *SCN 02*, volume 2576 of *LNCS*, pages 268–289. Springer, Berlin, Heidelberg, September 2003.

CL04. Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 56–72. Springer, Berlin, Heidelberg, August 2004.

Eur24a. European Digital Identity Wallet Cooperation Group. European digital identity wallet – architecture and reference framework (ARF), version 1.4.0. `https://eu-digital-identity-wallet.github.io/eudi-doc-architecture-and-reference-framework/1.4.0/arf/`, 2024.

Eur24b. European Parliament and Council of the European Union. Regulation (EU) 2024/1183 amending regulation (EU) no 910/2014 as regards establishing the European digital identity framework. Official Journal of the European Union, 2024.

Fas26. Matteo Frigo and abhi shelat. Anonymous credentials from ECDSA. *IACR Communications in Cryptology*, 3(1), 2026.

Fis06. Marc Fischlin. Round-optimal composable blind signatures in the common reference string model. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 60–77. Springer, Berlin, Heidelberg, August 2006.

FPR26. Thibauld Feneuil, Killian Provin, and Matthieu Rivain. Sum-checking SmallWood: Smaller hash-based zero-knowledge arguments. Cryptology ePrint Archive, Paper 2026/2880, 2026.

FR26. Thibauld Feneuil and Matthieu Rivain. SmallWood: Hash-based polynomial commitments and zero-knowledge arguments for relatively small instances. *IACR Communications in Cryptology*, 3(2), 2026.

FS87. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Berlin, Heidelberg, August 1987.

GKR08. Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: interactive proofs for muggles. In Richard E. Ladner and Cynthia Dwork, editors, *40th ACM STOC*, pages 113–122. ACM Press, May 2008.

GMR89. Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.

Hab22. Ulrich Haböck. Multivariate lookups based on logarithmic derivatives. Cryptology ePrint Archive, Report 2022/1530, 2022.

II21. International Organization for Standardization and International Electrotechnical Commission. Personal identification — ISO-compliant driving licence — part 5: Mobile driving licence (mDL) application. ISO/IEC 18013-5:2021, 2021.

JRS23. Corentin Jeudy, Adeline Roux-Langlois, and Olivier Sanders. Lattice signature with efficient protocols, application to anonymous credentials. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part II*, volume 14082 of *LNCS*, pages 351–383. Springer, Cham, August 2023.

Nat24. National Institute of Standards and Technology. Module-lattice-based digital signature standard. Federal Information Processing Standards Publication FIPS 204, 2024.

PS16. David Pointcheval and Olivier Sanders. Short randomizable signatures. In Kazue Sako, editor, *CT-RSA 2016*, volume 9610 of *LNCS*, pages 111–126. Springer, Cham, February / March 2016.

PWFW23. Guru-Vamsi Policharla, Bas Westerbaan, Armando Faz-Hernández, and Christopher A Wood. Post-quantum privacy pass via post-quantum anonymous credentials. Cryptology ePrint Archive, Report 2023/414, 2023.

SG26. Chad Sharp and Paul Grubbs. Faster and smaller anonymous credentials from standardized signatures. Cryptology ePrint Archive, Paper 2026/2615, 2026.

Sho94. Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *35th FOCS*, pages 124–134. IEEE Computer Society Press, November 1994.

A Soundness of SmallWood$^+$

This appendix quantifies the round-by-round knowledge soundness error of the two SmallWood$^+$ instances. We use the first compiler of [FPR26], whose soundness error is bounded by the maximum of the errors of its constituent rounds [FPR26, Theorem 2]. We first consider the underlying protocols with uniform interactive challenges.

Binary instance. The challenges of the binary instance are sampled from $\mathbb{F}_{2^\lambda}$, with $\lambda = 128, 192, 256$ for ML-DSA-44, ML-DSA-65, and ML-DSA-87, respectively. In this instance, the largest output layer is that of the batched Keccak circuit of Section 4.1. Let P be the number of permutations and F the flattening factor. Each permutation is split into F blocks of $24/F$ rounds, so that PF blocks are proven in parallel. Each 1600-bit state is padded to 2^{11} coordinates, and the number of blocks is padded to a power of two. The multilinear extension of the output layer therefore has $\ell_{\text{out}} = 11 + \lceil \log_2(PF) \rceil$ variables. The initial GKR test compares the claimed and computed output extensions at a random point, with soundness error at most $\ell_{\text{out}}/2^\lambda$.

In SmallWood$^+$, the prover messages of the sum-check-based protocols are masked by a committed *trace mask*, so that the multiplicative checks of the verifier on these messages become degree-two constraints on the trace mask, called *trace constraints*. For each constituent protocol, these constraints are linearized with a technique inspired from linear PCPs, which replaces m of them by two linear constraints, with soundness error at most $2m/2^\lambda$ [FPR26]. In the batched Keccak circuit, each round of a block contributes one trace constraint, so a linearization involves $m = 24/F$ of them. This is the largest linearization in our binary circuits. We use $F = 6$ for the proofs of knowledge and the blind signatures, and $F = 2$ for the credentials, giving $m = 4$ and $m = 12$, respectively.

Prime-field instance. The challenges of the prime-field instance are sampled from $\mathbb{K} = \mathbb{F}_{q^6}, \mathbb{F}_{q^9}, \mathbb{F}_{q^{12}}$, of size about 2^{138}, 2^{207}, and 2^{276}, that is, at least $2^{\lambda+9.9}$. Its round errors have the same form as above, $c'/|\mathbb{K}|$, where c' is the output dimension of a layered circuit, the degree of a sum-check, or twice the number of trace constraints of a linearization. For batched GKR, the trace constraints of all batch members are linearized together. The ML-DSA-44 credential has $2m = 166$, giving an error $166/q^6 \approx 2^{-130.62}$, below 2^{-128}.

Resulting bounds. For the selected parameters, each round of the commitment schemes (LCS and DECS [FPR26]) and each round of the prime-field SmallWood$^+$ instance has error at most $2^{-\lambda}$. On the binary side, the sum-check degree is at most 3 and the ring-switching batching dimension is at most 8, so these errors are dominated by the initial GKR and trace-linearization bounds. The maximum of the errors of the two instances is therefore bounded by

$$\epsilon_{\text{SW}} := \max\{\epsilon^{(2)}_{\text{rbr}}, \epsilon^{(q)}_{\text{rbr}}\} \le c\, 2^{-\lambda}, \qquad c = \max\{\ell_{\text{out}}, 2m\}.$$

The initial GKR test determines this bound for the proofs of knowledge and the blind signatures, whereas trace linearization dominates for the credentials, with $2m = 24$. Table 12 gives the permutation counts, the resulting output dimensions, and the exponents $\lambda - \log_2 c$ of the error bounds, rounded to two decimals. These exponents are smaller than those of the arithmetic error, so they determine the interactive error per verifier message of Theorem 1.

Table 12: Circuit parameters and exponents (in bits) of the $\mathsf{SmallWood}^{+}$ error bounds under uniform interactive challenges.

Application	Quantity	ML-DSA-44	ML-DSA-65	ML-DSA-87
Proof of knowledge	P	10	10	12
	ℓ_{out}	17	17	18
	$\lambda - \log_2 c$	123.91	187.91	251.83
Blind signature	P	11	11	13
	ℓ_{out}	18	18	18
	$\lambda - \log_2 c$	123.83	187.83	251.83
Anonymous credentials	P	140	234	405
	ℓ_{out}	20	20	21
	$\lambda - \log_2 c$	123.42	187.42	251.42

Grinding in the classical ROM. Fix a transcript prefix and let p_i bound the bad-challenge probability for a fresh draw in round i. We choose the smallest integer $g_i \geq 0$ such that

$$2^{-g_i} p_i \leq 2^{-\lambda}.$$

Both parties test successive counters until g_i designated bits of the SHAKE256 output are zero and disjoint bits yield an admissible challenge. A fresh oracle trial thus qualifies and yields a bad challenge with probability at most $2^{-g_i} p_i$. Over Q fresh classical trials, the bad-challenge contribution is at most $Q\, 2^{-\lambda}$. The selected challenge has the prescribed distribution, so the interactive error per verifier message is unchanged. The random-oracle bad-event terms of Theorem 1 are accounted for separately.