SPRUCE: SmallWood Proofs for Robust, Unlinkable Anonymous Credential Evidence via PACS and ISIS$_f$-BBS

Jonas Lazard*and Carsten Baum†

DTU Compute, Denmark

December 9, 2025

1 Todos

- Section 7: how to choose parameters so all works together (basically - the process) Jonas
- Section 6: better description of SmallWood and why/how it works + diagrams Jonas

Future todos:

1. likely better structure to begin with describing the NTRU sampler and the ZK proof and then have the section that describes the Blind Signature scheme as putting this all together.
2. Derive randomness for initial commitment generating t interactively or use ZK-friendly ZK hash

2 Introduction

Anonymous credentials enable users to prove possession of attributes without revealing them, while resisting profiling and linkability. Achieving this in the post-quantum setting with practical sizes remains challenging. We combine a proof-friendly, lattice-based Blind Signature (vSIS-BBS [3] combined with an NTRU sampler [6]) with the SmallWood hash-based NIZK proof system [7] to obtain compact, verifiable presentations. Lena: We then show how to extend SPRUCE to get the first post-quantum version of an ARC-style anonymous credential token.

Contributions. In more detail, this work combines the following primitives:

1. We implement an instantiation of the vSIS-BBS blind signature [3]. This signature is tailored for proof systems: the message generated by the holder does not involve computing a full-fledged cryptographic hash function, simplifying the proof size towards the issuer and the verifier.
2. We adopt the trapdoor-based Gaussian preimage sampler tuned for stable per-slot tails from [6]. This sampler generates the preimages necessary for the aforementioned signature scheme.

*Electronic address: jonas.lazard@student-cs.fr
†Electronic address: cabau@dtu.dk

3. We implement the SmallWood [7] NIZK proof system which is linear in the witness size, but succinct in the size of the statement being proven. We then encode the degree-2 signature relation and range membership via SmallWood tailored constraints.

4. We present implementation-driven optimizations (Merkle multiproofs, trimming, compact headers) that reduce the proof size without altering algebraic checks.

5. We combine all building blocks, together with carefully chosen parameters. This yields a Blind Signature scheme with ≈2 kB signatures and ≈15–20 kB proofs.

6. Lena: ARC tokens are unlinkable tokens with constant issuance size, regardless of many presentations they have. We show how do add SPRUCE to get a post-quantum ARC token in under XX kB of proof.

2.1 How the pieces connect in our construction

At a high level (see Fig. 1):

1. **Commitment from vSIS ($\mathcal{H} \to \mathcal{I}$).** The holder computes $C = \mathsf{Commit}(\mathsf{pp}, m; r)$ using a vSIS-hash–based commitment (computational binding from vSIS/SIS-type assumptions; computational hiding from randomized hashing). *No* attribute is revealed to $\mathcal{I}$.

2. **Blind signature on C ($\mathcal{I} \to \mathcal{H}$).** Via $\mathsf{Blind}/\mathsf{Sign}/\mathsf{Unblind}$, $\mathcal{I}$ signs the blinded commitment; the holder obtains a signature σ on C without disclosing m.

3. **Zero-knowledge presentation ($\mathcal{H} \to \mathcal{V}$).** Using SmallWood's hash-based PCS/PIOP stack, $\mathcal{H}$ proves knowledge of (m, r, σ') such that $C = \mathsf{Commit}(\mathsf{pp}, m; r)$ and $\mathsf{Verify}(\mathsf{ipk}, C, \sigma') = 1$, and optionally that m satisfies predicates (e.g., ranges). $\mathcal{V}$ outputs accept/reject.

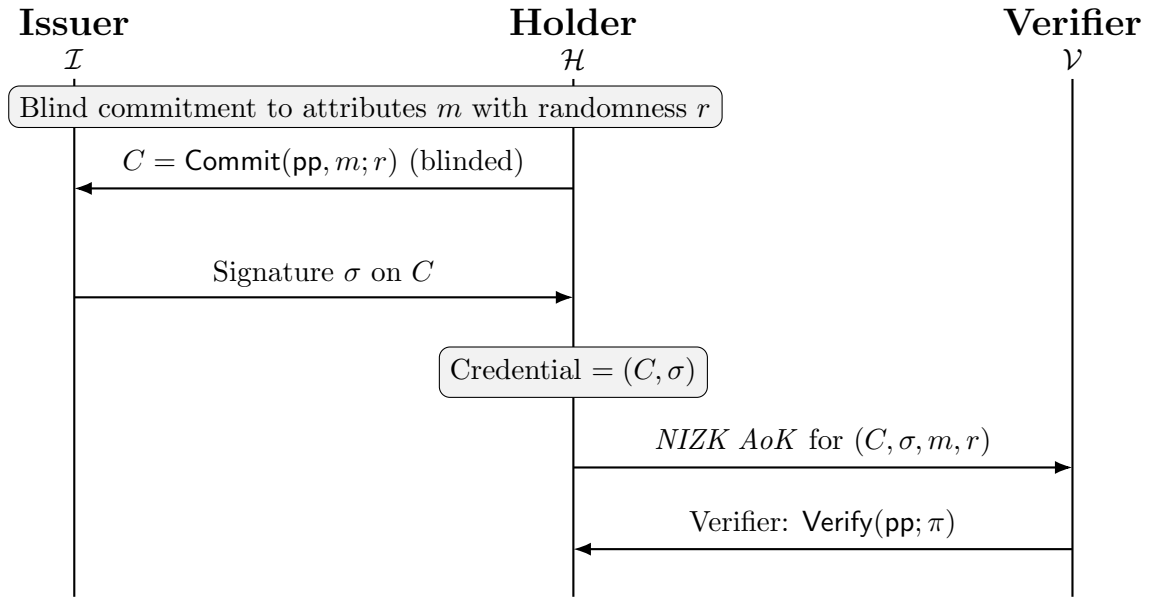


Figure 1: Issuance and presentation flow. *Default showing transmits only the proof π; neither r nor a deterministic tag is revealed, yielding unlinkability.*

The public parameters for our construction are a trapdoored matrix $\mathbf{A} \in R_q^{n \times m}$ (verification map) for an ISIS instance and a public matrix $\mathbf{B} \in R_q^{n \times \ell}$ (hash domain). The message and randomness are an ISIS preimage relation with a BBS-style *rational hash*; its output $\mathbf{t}$ serves as

the commitment. For a message u and signing randomness χ, the vSIS construction defines a rational function $h_{u,\chi}$ and sets the target

$$\mathbf{t} \;\coloneqq\; h_{u,\chi}(\mathbf{B}) \;\in R_q^n.$$

A signature is precisely a short preimage $\mathbf{s}$ such that

$$\boxed{\mathbf{A}\mathbf{s} \;\equiv\; \mathbf{t} \pmod q} \qquad \text{with} \quad \|\mathbf{s}\| \text{ small.}$$

This is the verification equation we will prove in zero knowledge later on.

Concrete choices of $h_{u,\chi}$ translate the verification equations into low-degree polynomial relations over R_q Previous work [4] formalizes this pattern through a general signature skeleton $\Sigma_{\mathrm{ISIS_f}}$ where verification checks $\mathbf{A}\mathbf{s} = h_{u,\chi}(\mathbf{B}) \pmod q$ and $\mathbf{s}, \chi, u$ is short; our instantiation follows that template.

3 Preliminaries - a'la Carsten

Notation. We let $\mathrm{negl}(n)$ denote any positive function that is negligible in n, i.e. upper-bounded by $1/|p(n)|$ for any polynomial p and large enough n. Let λ denote the security parameter and we write PPT to denote probabilistic algorithms whose runtime is polynomial in λ. For $a, b \in \mathbb{Z}, a < b$ write $[a, b]$ to denote the set $[a, a+1, \ldots, b-1, b]$ and use $[b]$ as a short-hand for $[1, b]$.

3.1 Commitment Schemes

Definition 3.1 (Non-interactive commitment)**.** A *commitment scheme* consists of the following polynomial-time algorithms:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$: Takes the security parameter as input and outputs a commitment public key pp. pp also describes the message space $\mathcal{M}$, randomness space $\mathcal{R}$ and commitment space $\mathcal{C}$.
- $\mathsf{Commit}(\mathsf{pp}, \mu, r) \to c$: Takes a commitment public key pp, a message $\mu \in \mathcal{M}$ and randomness $r \in \mathcal{R}$ as input and outputs a commitment $c \in \mathcal{C}$.

We say that a commitment scheme is correct if for any honestly generated pp, the algorithm Commit is deterministic and always terminates except with negligible probability. In addition, we require that it is binding and hiding:

Definition 3.2 (Binding)**.** A commitment scheme $(\mathsf{Setup}, \mathsf{Commit})$ is *binding* if there exists a negligible function negl, such that for any PPT algorithm $\mathcal{A}$ we have that

$$\Pr\left[\begin{array}{c} c_1 = c_2, \\ c_1 \neq \bot,\ \mu_1 \neq \mu_2 \end{array} \middle| \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda), \\ (\mu_1, \mu_2, r_1, r_2) \leftarrow \mathcal{A}(\mathsf{pp}), \\ c_1 \leftarrow \mathsf{Commit}(\mathsf{pp}, \mu_1, r_1), \\ c_2 \leftarrow \mathsf{Commit}(\mathsf{pp}, \mu_2, r_2) \end{array}\right] \leq \mathrm{negl}(\lambda)\,.$$

Definition 3.3 (Hiding)**.** A commitment scheme $(\mathsf{Setup}, \mathsf{Commit})$ is *hiding* if there exists a negligible function negl, such that for any PPT algorithm $\mathcal{A}$ we have that

$$\left|\Pr\left[b = \mathcal{A}(c, \mathsf{state}) \middle| \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda), \\ (\mu_0, \mu_1, \mathsf{state}) \leftarrow \mathcal{A}(\mathsf{pp}), \\ b \xleftarrow{\$} \{0,1\},\ r \xleftarrow{\$} \mathcal{R}, \\ c \leftarrow \mathsf{Commit}(\mathsf{pp}, \mu_b, r)) \end{array}\right] - \frac{1}{2}\right|$$

is bounded by $\mathrm{negl}(\lambda)$.

3.2 Blind Signatures

We provide the correctness and security definitions of Blind signatures [2].

Definition 3.4 (Blind Signature)**.** A round-optimal *Blind signature* scheme consists of the following five PPT algorithms:

- $\mathsf{KeyGen}(1^\lambda) \to (\mathtt{sk}, \mathtt{pk})$: Takes the security parameter as input and outputs a secret key $\mathtt{sk}$ and a public key $\mathtt{pk}$.
- $\mathsf{Sign}_1(\mathtt{pk}, \mu) \to (\rho_1, S_{\mathcal{U}})$: This is the first part of a signing protocol between a user $\mathcal{U}$ and a signer $\mathcal{S}$, where $\mathcal{U}$ uses a public key and a message $\mu \in \{0,1\}^*$, to compute a state $S_{\mathcal{U}}$ and a first message ρ_1, which they send to $\mathcal{S}$.
- $\mathsf{Sign}_2(\mathtt{sk}, \rho_1) \to \rho_2$: The second part of the interaction, where $\mathcal{S}$ uses their signing key $\mathtt{sk}$ and a first message ρ_1 and outputs a second message ρ_2, which they send to $\mathcal{U}$.
- $\mathsf{Sign}_3(\rho_2, S_{\mathcal{U}}) \to \mathsf{sig}$: The last part of the interaction, where $\mathcal{U}$ uses ρ_2 and $S_{\mathcal{U}}$ to derive a signature sig or $\perp$ if the signing protocol failed.
- $\mathsf{Verify}(\mathtt{pk}, \mu, \mathsf{sig}) \to \{0,1\}$: A verification algorithm that takes as input a public key $\mathtt{pk}$, a message μ and a signature sig and outputs a bit to indicate whether the signature is deemed valid (1) or invalid (0).

All the algorithms take the security parameter λ as input (sometimes implicitly).

We require Blind signatures to satisfy three properties: correctness, which says that honestly generated signatures should verify with probability close to 1, blindness, which says that the signer cannot link signatures to the interaction with which they were created, and existential unforgeability, which says that if a user is allowed to execute the signing protocol Q times with chosen messages it cannot obtain valid signatures for any other messages.

Definition 3.5 (Correctness)**.** A Blind signature $(\mathsf{KeyGen}, (\mathsf{Sign}_i)_{i\in[3]}, \mathsf{Verify})$ is *correct* if there exists a negligible function negl, such that for any message μ we have

$$\Pr\left[\mathsf{Verify}(\mathtt{pk}, \mu, \mathsf{sig}) = 1 \;\middle|\; \begin{array}{c} (\mathtt{sk}, \mathtt{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda), \\ (\rho_1, S_{\mathcal{U}}) \leftarrow \mathsf{Sign}_1(\mathtt{pk}, \mu), \\ \rho_2 \leftarrow \mathsf{Sign}_2(\mathtt{sk}, \rho_1), \\ \mathsf{sig} \leftarrow \mathsf{Sign}_3(\rho_2, S_{\mathcal{U}}) \end{array}\right]$$

is 1 except with probability $\mathrm{negl}(\lambda)$.

Definition 3.6 (Blindness)**.** A Blind signature $(\mathsf{KeyGen}, (\mathsf{Sign}_i)_{i\in[3]}, \mathsf{Verify})$ has *blindness* if for every polynomial-time three-part stateful adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$ there exists a negligible function negl such that for any two messages μ_0, μ_1 we have

$$\left|\Pr\left[\mathcal{A}_3(\mathsf{sig}^0, \mathsf{sig}^1) = b \;\middle|\; \begin{array}{c} \mathtt{pk} \leftarrow \mathcal{A}_1(1^\lambda), b \xleftarrow{\$} \{0,1\}, \\ (\rho_1^0, S_{\mathcal{U}}^0) \leftarrow \mathsf{Sign}_1(\mathtt{pk}, \mu_0), \\ (\rho_1^1, S_{\mathcal{U}}^1) \leftarrow \mathsf{Sign}_1(\mathtt{pk}, \mu_1), \\ \rho_2^b, \rho_2^{1-b} \leftarrow \mathcal{A}_2(\rho_1^b, \rho_1^{1-b}), \\ \mathsf{sig}^0 \leftarrow \mathsf{Sign}_3(\rho_2^0, S_{\mathcal{U}}^0), \\ \mathsf{sig}^1 \leftarrow \mathsf{Sign}_3(\rho_2^1, S_{\mathcal{U}}^1), \\ \text{If } \perp \in \{\mathsf{sig}^0, \mathsf{sig}^1\} \\ \text{then } (\mathsf{sig}^0, \mathsf{sig}^1) \leftarrow (\perp, \perp) \end{array}\right] - \frac{1}{2}\right|$$

is bounded by $\mathrm{negl}(\lambda)$.

Definition 3.7 (One-more unforgeability)**.** A Blind signature $(\mathsf{KeyGen}, (\mathsf{Sign}_i)_{i\in[3]}, \mathsf{Verify})$ is *one-more unforgeable* if for every polynomial-time adversary $\mathcal{A}$ making at most Q queries (Q is a function of λ, bounded by a polynomial) to the signing oracle Sign_2 there exists a negligible function negl such that

$$\Pr\left[\begin{array}{c}\forall 1 \le i < j \le Q+1 : \\ (\mu_i \neq \mu_j)\,, \\ \forall 1 \le i \le Q+1 : \\ (\mathsf{Verify}(\mathtt{pk}, \mu_i, \mathsf{sig}_i) = 1)\end{array}\middle|\begin{array}{c}(\mathtt{sk}, \mathtt{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda), \\ \{(\mu_i, \mathsf{sig}_i)\}_{i\in[Q+1]} \leftarrow \\ \mathcal{A}^{\mathsf{Sign}_2(\mathtt{sk},\cdot)}(\mathtt{pk})\end{array}\right] \le \mathrm{negl}(\lambda)\,.$$

3.3 Non-Interactive Zero-Knowledge Arguments

We now describe the model of Non-Interactive Zero-Knowledge Arguments (NIZKs) that we use in this work.

Computational model. We model computation by applying a circuit C to an input $\mathbf{w} \in \mathbf{F}_p^\ell$. C has ℓ input wires and a single output wire. We say that the input $\mathbf{w}$, or witness, is valid if $C(\mathbf{w}) = 0$ and invalid otherwise.

C consists of wires and gates, where each gate has at least one input wire and exactly one output wire. The input wires of each gate are either the input wires of C, or output wires of other gates. The output of a gate may be input to multiple other gates, and the wires and gates form an acyclic directed graph. There are 3 types of gates in a circuit:

Linear gates apply public linear functions $\mathtt{lin}(\alpha, \beta, \mathbf{x}) = \langle \alpha, \mathbf{x}\rangle + \beta$ on the input wires $\mathbf{x} \in \mathbf{F}_p^h$. $\alpha \in \mathbf{F}_p^h, \beta \in \mathbf{F}_p$ are public inputs to the gate, while the output is a single element in $\mathbf{F}_p$. Note that h may be arbitrarily large.

Multiplication gates take as input a vector $\mathbf{x} \in \mathbf{F}_p^h$, compute the product of all input wires $\mathtt{mul}(\mathbf{x}) = \prod_{i\in[h]} x_i$ and output the product as an element in $\mathbf{F}_p$. h can be larger than 2.

Assert gates take as input a wire value $x \in \mathbf{F}_p$ and output $\top$ if $x = 0$ and $\bot$ otherwise. Note that the output wire of an assert gate may therefore never be input to another gate.

We define the evaluation of C on input $\mathbf{w}$ as follows: $\mathbf{w}$ is initially assigned to the input wires of the circuit, and then consecutively all gates whose input wires are fully assigned are evaluated and the output of the gate function is assigned to the output wire(s). This process is repeated until a) the single output wire of the circuit has a value assigned to it; and b) all *assert gates* have been evaluated. If all assert gates have output $\top$ then $C(\mathbf{w})$ is equivalent to the value assigned to the output wire of C, otherwise $C(\mathbf{w}) = \bot$.

NIZKs. We now define NIZK proof systems in the random oracle (RO) model.

Definition 3.8. A *NIZK proof system* Π is a tuple $\Pi = (\mathsf{Prove}, \mathsf{Verify})$ of PPT algorithms that have access to the random oracle H:

- $\mathsf{Prove}^{\mathsf{H}}(C, \mathbf{w}) \to \pi$: Takes a circuit C and a witness $\mathbf{w}$ and outputs a proof π.
- $\mathsf{Verify}^{\mathsf{H}}(C, \pi) \to \{0, 1\}$: Takes a circuit C, and a proof π, and outputs a bit to indicate whether the proof is deemed valid (1) or invalid (0).

In the following, we drop the superscript H from $\mathsf{Prove}, \mathsf{Verify}$ to simplify notation.

Definition 3.9 (Correctness)**.** A NIZK proof system $(\mathsf{Prove}, \mathsf{Verify})$ in the RO model has *correctness error* ϵ_{cor} if for any C and for all $\mathbf{w} \in \mathbf{F}_q^\ell$ such that $C(\mathbf{w}) = 0$,

$$\Pr\left[\mathsf{Verify}(C, \pi) = 1 | \pi \leftarrow \mathsf{Prove}(C, \mathbf{w})\right] \ge 1 - \epsilon_{\mathsf{cor}}(1^\lambda)$$

$(\mathsf{Prove}, \mathsf{Verify})$ is *correct* if $\epsilon_{\mathsf{cor}} = \mathrm{negl}$.

Definition 3.10 (Non-interactive Zero-Knowledge)**.** A simulator Sim for a NIZK proof system $\Pi = (\mathsf{Prove}, \mathsf{Verify})$ is a PPT algorithm with implicit input 1^λ. Sim on input C outputs π. Let $\mathcal{A}$ be a stateful algorithm and let

$$\mathsf{Real}_{\mathcal{A}}(1^\lambda) = \Pr\left[b = 1 \,|\, b \leftarrow \mathcal{A}^{\mathsf{H}, \mathcal{O}_{\mathsf{Prove}}}(1^\lambda)\right]$$
$$\mathsf{Ideal}_{\mathcal{A}}(1^\lambda) = \Pr\left[b = 1 \,|\, b \leftarrow \mathcal{A}^{\mathsf{H}, \mathcal{O}_{\mathsf{Sim}}}(1^\lambda)\right]$$

The adversary has black-box access to the random oracle H and to an oracle $\mathcal{O}_{\mathsf{Prove}}$ or $\mathcal{O}_{\mathsf{Sim}}$. These oracles take $C, \mathbf{w}$ as input, check if $C(\mathbf{w}) = 0$ and then do the following:

- $\mathcal{O}_{\mathsf{Prove}}(C, \mathbf{w})$ computes $\pi \leftarrow \mathsf{Prove}^{\mathsf{H}}(C, \mathbf{w})$ and outputs π.
- $\mathcal{O}_{\mathsf{Sim}}(C, \mathbf{w})$ computes $\pi \leftarrow \mathsf{Sim}^{\mathsf{H}}(C)$.

The simulator Sim is allowed to *program* the random oracle H on any fresh input to H, i.e. if $\mathsf{H}(m)$ has not been queried by any party before, then $\mathcal{A}$ resp. Sim is free to choose $\mathsf{H}(m)$, otherwise, programming fails.

We define the advantage of $\mathcal{A}$ by $\left|\mathsf{Real}_{\mathcal{A}}(1^\lambda) - \mathsf{Ideal}_{\mathcal{A}}(1^\lambda)\right|$. We call Sim a *Non-interactive Zero-Knowledge Black-box* simulator for Π if for any PPT $\mathcal{A}$ the advantage is negligible.

Knowledge soundness for *black-box* extraction in the RO model for NIZKs is defined as follows:

Definition 3.11. Let Π be a non-interactive proof system and KExt be a PPT algorithm. Define the knowledge soundness experiment as follows:

1. The experiment obtains C as input.
2. Run $\mathcal{A}^{\mathsf{H}}(1^\lambda, C)$ until it outputs π. Let Q be all RO queries made by $\mathcal{A}$.
3. Return 0 if $\mathsf{Verify}(C, \pi) = 0$.
4. Return 1 if $C(\mathsf{KExt}(C, \pi, Q)) \neq 0$, else return 0.

The advantage of $\mathcal{A}$ is defined as the probability that the experiment returns 1. If there exist KExt such that for every PPT $\mathcal{A}$ the advantage is negligible, then we call Π knowledge sound.

3.4 Lattice Primitives

Lattice preliminaries and evaluation basis. Let $q \geq 2$ be an integer modulus and let φ be a power of two. We work in the cyclotomic ring $R \coloneqq \mathbb{Z}[X]/(X^\varphi + 1)$ and its q-adic reduction $R_q \coloneqq R/qR$. For vectors over R we use boldface (e.g. $\mathbf{a} \in R^m$), and we write $\|\cdot\|_2,\ \|\cdot\|_\infty$ for coefficient-wise norms on R (extended component-wise to R^m). Via the NTT/evaluation basis, Hadamard product $\odot$ and Hadamard division $\oslash$ act coordinate-wise; division is understood only when all denominator coordinates are nonzero.

We work with a NTT-friendly prime q such that :

$$\operatorname{ord}_{2\varphi}(q) = 1 \quad \Longleftrightarrow \quad q \equiv 1 \pmod{2\varphi}.$$

We thus can identify $R_q \cong \mathbf{F}_q^N$ (with $N = \varphi$) since the cyclotomic polynomial $X^\varphi + 1$ splits completely over $\mathbf{F}_q$.

Definition 3.12 (SIS and ISIS over R_q)**.** Fix integers $n, m, \varphi \geq 1$ where φ is a power of 2 and let $q \geq 2$ be a modulus. Define R_q as above.

- **The homogeneous SIS problem.** Given $\mathbf{A} \in R_q^{n \times m}$ and a bound $\beta > 0$, find a non-zero $\mathbf{s} \in R^m$ such that $\mathbf{A}\mathbf{s} \equiv \mathbf{0} \pmod{q}$ and $\|\mathbf{s}\| \leq \beta$ (for a prescribed norm, e.g., ℓ_∞ or ℓ_2).

- **The inhomogeneous SIS (ISIS) problem.** Given $(\mathbf{A}, \mathbf{t}) \in R_q^{n \times m} \times R_q^n$ and $\beta > 0$, find $\mathbf{s} \in R^m$ with $\mathbf{A}\mathbf{s} \equiv \mathbf{t} \pmod q$ and $\|\mathbf{s}\| \leq \beta$.

We say that an attacker $\mathcal{A}$ solves the SIS problem if $\mathcal{A}$, on input $\mathbf{A}, \beta$ as well as all parameters of the SIS instance, outputs a valid solution $\mathbf{s}$. A similar definition can be made for the ISIS problem. On a formal level, we will assume that for the SIS/ISIS instances that we use, no PPT algorithm $\mathcal{A}$ solves them with non-negligible advantage. Concretely, we will later choose q, n, m, φ such that attacks require a runtime of at least 2^λ.

3.4.1 Trapdoors

Let $\mathbf{c} \in \mathbb{R}^m$ and $s > 0$, then the Gaussian function on $\mathbb{R}^m$ with center $\mathbf{c}$ and parameter s is defined as

$$\forall \mathbf{x} \in \mathbb{R}^m : \ \rho_{s,\mathbf{c}}(\mathbf{x}) := \exp -\pi \|\mathbf{x} - \mathbf{c}\|^2 / s^2.$$

Definition 3.13 (Discrete Gaussian Distribution)**.** For any $\mathbf{c} \in \mathbb{R}^m$, real $s > 0$ and m-dimensional lattice Λ, the *discrete Gaussian distribution* over Λ is defined as

$$\forall \mathbf{x} \in \Lambda : \ D_{\Lambda,s,\mathbf{c}} := \frac{\rho_{s,\mathbf{c}}(\mathbf{x})}{\rho_{s,\mathbf{c}}(\Lambda)}$$

If $\mathbf{c} = \mathbf{0}$, i.e. the distribution is centered, then we drop $\mathbf{c}$ from the definition.

For a matrix $\mathbf{A} \in R_q^{n \times m}$ and target $\mathbf{t} \in R_q^n$, denote by $\Lambda_\perp^{\mathbf{t}}(\mathbf{A}) \coloneqq \{\mathbf{e} \in R^m : \mathbf{A}\mathbf{e} \equiv \mathbf{t} \ (\bmod \ q)\}$ the corresponding coset of the lattice.

Definition 3.14 (Lattice trapdoor suite)**.** Let $R_q = R/qR$. A lattice trapdoor suite consists of two PPT algorithms $(\mathsf{TrapGen}, \mathsf{SampPre})$:

- $(\mathbf{A}, \mathsf{td}_\mathbf{A}) \leftarrow \mathsf{TrapGen}(1^\lambda, 1^n, 1^m, q)$ outputs $\mathbf{A} \in R_q^{n \times m}$ together with a trapdoor $\mathsf{td}_\mathbf{A}$.
- $\mathbf{u} \leftarrow \mathsf{SampPre}(\mathsf{td}_\mathbf{A}, \mathbf{v}, s)$, on input $\mathbf{v} \in R_q^n$ and parameter $s > 0$, samples a vector $\mathbf{e} \in \Lambda_\perp^{\mathbf{t}}(\mathbf{A})$.

We now define security for trapdoor sampler:

Definition 3.15 (Secure Trapdoor sampler)**.** Let $(\mathsf{TrapGen}, \mathsf{SampPre})$ be a lattice trapdoor suite. Let $\mathcal{A}$ be any algorithm and let

$$\mathsf{Real}_\mathcal{A}(1^\lambda) = \Pr\left[b = 1 \,|\, b \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Real},G}, \mathcal{O}_{\mathsf{Real},S}}(1^\lambda)\right]$$
$$\mathsf{Ideal}_\mathcal{A}(1^\lambda) = \Pr\left[b = 1 \,|\, b \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Ideal},G}, \mathcal{O}_{\mathsf{Ideal},S}}(1^\lambda)\right]$$

The adversary has black-box access to two pairs of oracles. These oracles do the following

- $\mathcal{O}_{\mathsf{Real},G}()$ upon first call computes $(\mathbf{A}, \mathsf{td}_\mathbf{A}) \leftarrow \mathsf{TrapGen}()$, stores them and outputs $\mathbf{A}$. It ignores all further calls.
- $\mathcal{O}_{\mathsf{Real},S}()$ samples $\mathbf{v} \leftarrow R_q^n$ uniformly at random, computes $\mathbf{s} \leftarrow \mathsf{SampPre}(\mathsf{td}_\mathbf{A}, \mathbf{v}, s)$ and outputs $(\mathbf{s}, \mathbf{v})$.
- $\mathcal{O}_{\mathsf{Ideal},G}()$ upon first call samples $\mathbf{A} \leftarrow R_q^{n \times m}$ uniformly at random, stores $\mathbf{A}$ and outputs it. It ignores all further calls.
- $\mathcal{O}_{\mathsf{Ideal},S}()$ samples $\mathbf{e} \leftarrow D_{\Lambda_\perp^{\mathbf{t}}(\mathbf{A}),s}$, compute $\mathbf{v} = \mathbf{A}\mathbf{e}$ and outputs $(\mathbf{s}, \mathbf{v})$.

We define the advantage of $\mathcal{A}$ by $\left|\mathsf{Real}_\mathcal{A}(1^\lambda) - \mathsf{Ideal}_\mathcal{A}(1^\lambda)\right|$. We call the trapdoor suite *secure* if for any potentially computationally unbounded $\mathcal{A}$, the advantage is negligible.

3.4.2 Rational Hashes

Definition 3.16 (Rational hash family H)**.** Fix public $B = (B_0, b_1) \in R_q^{n\times\ell}$, message u, and randomness $\chi = (x_0, x_1)$ in prescribed ranges. Define $h_{u,\chi} : R_q^{n\times\ell} \to R_q^n$ by

$$h_{u,\chi}(B) \;=\; B_0 \cdot (1; u; x_0) \;\oslash\; \big(b_1 - \mathbf{1}_n \cdot x_1\big),$$

The denominator is required to be nonzero in *every* slot (i.e., coordinate-wise invertible in R_q). We call $h_{u,\chi}$ the *rational hash* and $t = h_{u,\chi}(B)$ the *commitment image* of (u, χ).

Security assumptions for rational hashes. Our $H = \{h_{u,\chi}\}$ is analyzed in the framework given by A.Dubois *et.al* [4] as follows.

- **Natural predicate $P_{\gamma,\varepsilon}$.** We restrict function sets by the predicate $P_{\gamma,\varepsilon}$: (i) the family is *ε-strongly linearly independent* (SLI) and (ii) every denominator polynomial has coefficients bounded by γ. This prevents degenerate choices (e.g. adaptively zeroing denominators) and rules out linear-dependency pitfalls. [4, Defs 6-7]
- **Hinted vSIS hardness.** For function families augmented with $P_{\gamma,\varepsilon}$, the signature's EUF security reduces to *strong* hinted-vSIS (s-Hint-vSIS): given short preimages of Q hints $\{h_{u_i,\chi_i}(B)\}$, producing a short preimage for a *fresh* target $h_{u^\star,\chi^\star}(B)$ is hard. [4, §3.1]
- **Equivalence to GenISIS$_f$.** Our concrete family $h_{u,\chi}(B)$ instantiates $f(\cdot)$ in the generalized ISIS$_f$ assumption, and s-EUF-RMA of the vSIS scheme is *equivalent* to GenISIS$_f$ (and conversely to s-$\$$Hint-vSIS under the same parameters). This pins the security definition to a standard preimage-resistance style assumption with oracle access. [4, Thm 7-8]
- **SLI in practice.** When R_q *splits* into a product of fields, ordinary linear independence implies SLI with explicit ε. This corresponds to the setting induced by the condition : $q \equiv 1 \bmod 2\varphi$.
- **Robustness note.** The work from A.Dubois *et.al* also analyzes linear-combination attacks and shows why the "natural" predicate (denominator bound + SLI) is the right guardrail for rational families (e.g., BB/BBS-type forms), ensuring that easy linear relations in $\mathrm{im}(h_{u,\chi})$ do not yield short forgeries.

Signed Lifting When encoding integers into $\mathbb{F}_q$, the *signed lifting* $\langle i\rangle_q$ is defined by $\langle i\rangle_q = i \pmod q$ for $i \geq 0$ and $\langle i\rangle_q = q - ((-i) \pmod q)$ for $i < 0$.

Definition 3.17 (Vanishing polynomial for set membership)**.** For a bound $B \in \mathbb{Z}_{>0}$, define

$$P_B(X) \;=\; \prod_{i=-B}^{B} \big(X - \langle i\rangle_q\big) \;\in \mathbb{Z}_q[X].$$

For any $x \in \mathbb{Z}_q$, the predicate $x \in [-B, B]$ (w.r.t. signed lifting) is equivalent to $P_B(x) \equiv 0 \pmod q$.

When instantiating the rational hash, each component is restricted coefficient-wise to a bounded alphabet:

$$u \in [-B, B]^{kN}, \qquad x_0 \in [-B, B]^{k_0 N}, \qquad x_1 \in [-B, B]^N.$$

A verifier can therefore validate that t is well-formed by verifying a ZK argument of knowledge of a preimage u, x_0, x_1 which fulfill the aforementioned bounds, where the computed circuit is given by Definition 3.17.

4 Gaussian Preimage Sampler for vSIS–BBS (Antrag–Hybrid over NTRU)

The first main primitive of our scheme is the Gaussian pre-image sampler used by our signature construction. At a high level, given a public linear map $\mathbf{A} \in R_q^{n \times m}$ and a target $\mathbf{t} \in R_q^n$ produced by the vSIS–BBS hash, we sample a short vector $\mathbf{s} \in R^m$ such that

$$\mathbf{As} \equiv \mathbf{t} \pmod{q}, \quad \text{i.e.,} \quad \mathbf{s} \in \Lambda_{\perp}^{\mathbf{t}}(\mathbf{A}) \coloneqq \{\, \mathbf{x} \in R^m : \mathbf{Ax} \equiv \mathbf{t} \,(\mathrm{mod} q)\,\}.$$

Our concrete sampler is a ring–NTRU instantiation of Prest's *hybrid* two-plane Gaussian sampler with the *Antrag* trapdoor-generation method, which attains high trapdoor quality while keeping key generation simple and fast (cf. [4]). The parameters that we need to control for said sampler are per-coefficient standard deviation of the distributions and the outputed norm of the pre-image.

4.1 NTRU trapdoor form and quality

We work with the standard rank-2 NTRU module over the cyclotomic ring $R = \mathbb{Z}[X]/(X^n + 1)$ and its reduction R_q. An NTRU public key is $h = gf^{-1} \bmod q$ for secret $f, g \in R$, and the corresponding module is

$$\mathsf{L}_{\mathsf{NTRU}}(h) \;=\; \{(u, v) \in R^2 : uh - v \equiv 0 \,(\mathrm{mod} q)\}.$$

A *trapdoor basis* of $\mathsf{L}_{\mathsf{NTRU}}(h)$ is

$$B_{f,g} \;\coloneqq\; \begin{pmatrix} f & F \\ g & G \end{pmatrix}, \qquad \text{with} \qquad fG - gF \;=\; q,$$

whose first column (f, g) and second column (F, G) are short and satisfy $g/f \equiv G/F \equiv h \pmod{q}$. The trapdoor $(f, g; F, G)$ is *good* when its quality α is close to 1, where

$$\alpha^2 \;=\; \max_{1 \le i \le d} \max\left\{ \frac{|\varphi_i(f)|^2 + |\varphi_i(g)|^2}{q}, \frac{q}{|\varphi_i(f)|^2 + |\varphi_i(g)|^2} \right\},$$

i.e., each slot energy $S_i \coloneqq |\varphi_i(f)|^2 + |\varphi_i(g)|^2$ lies in the window

$$\frac{q}{\alpha^2} \;\le\; S_i \;\le\; \alpha^2 q \qquad \text{for all embeddings } \varphi_i.$$

Equivalently, $(|\varphi_i(f)|, |\varphi_i(g)|)$ lies in the *annulus arc* $A^+(\sqrt{q}/\alpha,\ \alpha\sqrt{q}) \subset \mathbb{R}^2_{\ge 0}$. With this metric, the hybrid sampler's Gaussian width scales essentially like $\alpha\sqrt{q}$: smaller α means tighter signatures and higher security for a fixed q, n.

4.2 Sampler definition and parameter tuning

Definition (hybrid two-plane). Given $(\mathbf{A}, \mathsf{td})$ and a target $\mathbf{t}$, the signer runs a ring-based two-plane sampler in the evaluation domain:

$$\mathsf{SamplePreimage}(\mathbf{A}, \mathsf{td}, \mathbf{t}; \{\sigma_k[i]\}) \;\longrightarrow\; \mathbf{s} \in \Lambda_{\perp}^{\mathbf{t}}(\mathbf{A}),$$

We implement the following path: per-slot widths are calibrated from the trapdoor's per-slot Gram–Schmidt lengths, and draws are performed slot-wise in two planes. [5]

Symbols that resurface later.

- q modulus; d ring degree; trapdoor quality $\alpha > 1$ measured at keygen.
- Annulus radii (r, R) used by *keygen in the Fourier domain* (Antrag, §4.3).
- $R_{\text{acc}} > 0$ acceptance radius in the coefficient domain.
- $\widetilde{b}_k[i]$ per-slot ring GSO lengths (two planes $k \in \{1, 2\}$).

Per-slot calibration. The slot-wise widths are

$$\sigma_k[i] \;=\; \sqrt{\frac{\alpha^2 q\, R_{\text{acc}}^2}{\|\widetilde{b}_k[i]\|^2} \;-\; R_{\text{acc}}^2} \qquad (k \in \{1, 2\}),$$

equalizing acceptance across slots, avoiding weak slots, and keeping variance consistent with the measured key quality α.

Hiding threshold (smoothing). Let $\eta_\varepsilon(\Lambda)$ be the smoothing parameter. A GPV-style sufficient condition for statistical hiding is to enforce a floor $\sigma_{\min}$ with

$$\sigma_k[i] \;\geq\; \sigma_{\min} \;\text{ for all } k, i, \quad \text{and} \quad \sigma_{\min} \;\geq\; \eta_\varepsilon\big(\Lambda_\perp(\mathbf{A})\big).$$

We take $\eta_\varepsilon(\mathbb{Z}^{2\varphi}) = \sqrt{\ln\big(4\varphi(1+2^\lambda)\big)/\pi}$ (see §9) and raise all slot widths $\sigma_k[i] \geq \sigma_{\min}$ with slack. In practice we apply a global multiplicative knob `SigmaScale` to raise all $\sigma_k[i]$ above $\sigma_{\min}$ with slack (empirically 1.05–1.20 depending on (d, q)), and we monitor rejection and residual L_∞ histograms.

What we require (and rely on). For our anonymous-credential setting we use: (i) *correctness* ($\mathbf{As} \equiv \mathbf{t} \pmod q$ and the norm test fails only with negligible probability); (ii) *(approx.) Gaussianity* of $\mathbf{s}$ over the coset; (iii) *per-slot calibration* as above to enforce $\|\mathbf{s}\| \leq \beta$ with overwhelming probability. These follow from the usual hybrid two-plane analysis once the keygen per-slot windows hold.

Tuning at a glance (defaults).

- **Trapdoor quality α.** Measured at keygen (e.g. $\alpha \approx 1.23$ at $d = 1024$), setting the global $\approx \alpha\sqrt{q}$ scale that drives acceptance and variance.
- **Keygen annulus (r, R).** We target the middle third of the theoretical window (cf. §4.3):

  $$r = \Big(\tfrac{1}{3}\alpha + \tfrac{2}{3}\alpha^{-1}\Big)\sqrt{q}, \qquad R = \Big(\tfrac{2}{3}\alpha + \tfrac{1}{3}\alpha^{-1}\Big)\sqrt{q},$$

  which leaves rounding slack yet keeps repetition low in practice.
- **Acceptance radius R_{acc}.** A small multiplicative slack on R_{acc} lifts all $\sigma_k[i]$ uniformly and smooths acceptance; we follow the hybrid-B defaults in our implementation.

4.3 Annulus sampling (Antrag key generation)

Antrag replaces trial-and-error keygen with direct *annulus sampling* in the Fourier domain and compensates rounding by sampling from a slightly *narrower* annulus before inverse embedding [5]. For each conjugate pair (there are $d/2$ independent slots), we draw $\big(|\varphi_i(f)|, |\varphi_i(g)|\big)$ uniformly in the arc $A^+(r, R)$ with

$$\sqrt{q}/\alpha \;<\; r \;<\; R \;<\; \alpha\sqrt{q},$$

using polar sampling (uniform in θ and in ρ^2), attach random phases, inverse-embed (IFFT) to coefficients, and *round* to obtain $(f, g) \in R^2$. Choosing the "middle third" $r = (\frac{1}{3}\alpha + \frac{2}{3}\alpha^{-1})\sqrt{q},\ R = (\frac{2}{3}\alpha + \frac{1}{3}\alpha^{-1})\sqrt{q}$ makes the per-slot window $q/\alpha^2 \le |\varphi_i(f)|^2 + |\varphi_i(g)|^2 \le \alpha^2 q$ hold *after* rounding with high probability, with practical repetition rates $M \approx 3$–4 at our parameters. In short: (i) the per-slot window needed by the hybrid sampler is enforced *by construction*; (ii) keygen stays simple (independent slot draws + one IFFT + rounding); (iii) α can be kept close to 1 with modest rejection, reducing signing variance and tightening norm bounds [5].

4.4 From trapdoor to preimage: the hybrid two-plane sampler

Let t be the coefficient lift of $\mathbf{t}$ and $c_1 \coloneqq \mathrm{center}_q(t) \in (-\frac{q}{2}, \frac{q}{2}]^n$. With trapdoor basis $B_{f,g}$:

Centers. Build evaluation-domain centers from c_1.

Per-slot widths. Precompute ring GSO lengths $\|\widetilde{b}_k[i]\|$ and set

$$\sigma_k[i] \;=\; \sqrt{\frac{\alpha^2 q\, R_{\mathrm{acc}}^2}{\|\widetilde{b}_k[i]\|^2} \;-\; R_{\mathrm{acc}}^2} \qquad (k \in \{1, 2\}).$$

Draw, round, residual. Draw a Gaussian pair per slot, reconstruct (v_1, v_2) in the coefficient domain, round to nearest integers, set $s_1 \leftarrow -v_1^{\mathrm{rnd}}$, and the centered residual $s_2 \coloneqq \mathrm{center}_q\big(h \star s_1 + c_1\big)$.

Accept. Accept iff the norm predicate (L_2 and/or L_∞) holds on (s_1, s_2). On acceptance set $s_0 \coloneqq \mathrm{center}_q\big(t - h \star s_1\big)$, output (s_0, s_1), and retain s_2 (bounded by the predicate) for the ZK witness component.

Algorithm 1 $\mathsf{SampPre}(\mathsf{td}_A, t; s)$ — two-plane hybrid sampler (sketch)

1: $c \leftarrow \mathrm{center}_q(t) \in (-\frac{q}{2}, \frac{q}{2}]^n$.
2: Using the NTRU trapdoor in td_A, define per-slot covariance (width) $\sigma[i]$ calibrated to slot quality to avoid weak slots.
3: For each slot i, sample $(s_0[i], s_1[i]) \leftarrow \mathcal{D}_{\mathbb{Z}^2, \Sigma_i}$ on the two NTRU planes centered to satisfy $A\mathbf{s} \equiv t \pmod q$.
4: Return $\mathbf{s} = (s_0, s_1)$ (mapped back to coefficient domain).

Public relations. Verification and downstream proofs use

$$h \star s_1 + s_0 \equiv t \pmod q \qquad \text{and} \qquad \begin{bmatrix} I & h \end{bmatrix} \cdot \begin{pmatrix} s_2 \\ s_1 \end{pmatrix} \equiv c_1 \pmod q,$$

with s_1, s_2 bounded as enforced at signing time.

5 The SmallWood Proof System: DECS, LVCS and PCS/PIOP

In this section, we give a quick overview over the SmallWood ZK proof system. It consists of the following components components:

- **DECS** (Degree-Enforcing Commitment Scheme): a commitment layer that enforces low-degree structure of committed vectors/polynomials and supports proximity checks.
- **LVCS** (Linear Vector Commitment System): a vector commitment with linear operations and succinct openings; used to batch and route constraints.

- **PCS** (Polynomial Commitment Scheme): commit to a polynomial and prove evaluations/relations at points.
- **PIOP** (Polynomial IOP): an interactive oracle proof where the prover posts low-degree polynomials and the verifier queries them at random points; compiled via a PCS to a succinct (N)IZK.

SmallWood composes these layers (DECS $\to$ LVCS $\to$ PCS/PIOP) with hash-based commitments to obtain compact arguments. Soundness and (H)VZK are inherited from SmallWood under *transparent* ROM/Fiat–Shamir with grinding [8].

Instantiation note. Our concrete PACS instance (witness layout, degree-2 cleared identity, range and norm subroutines) is detailed in Section 7. Here we keep the presentation generic and cite SmallWood for soundness/HVZK.

5.1 Domains and Packing

Fix a base field $\mathbf{F}_q$ and a packing grid $\Omega = \{\omega_1, \ldots, \omega_s\} \subset \mathbf{F}_q$ of size s. Every witness *row* is encoded as a polynomial of degree $\leq s + \ell - 1$ whose values at Ω are the s packed coordinates and whose values at an extra, disjoint set $\Omega' = \{\omega'_1, \ldots, \omega'_\ell\} \subset \mathbf{F}_q \setminus \Omega$ provide ℓ blinding points for honest-verifier zero knowledge (HVZK). All identities and checks are enforced coefficient-wise over $\mathbf{F}_q$. We use the SmallWood knobs $(\rho, \ell', \ell, N, n_{\text{cols}}, d_Q, \eta)$ with bounds as in [8]. In what follows, "$\equiv \bmod q$" and all Hadamard operations are understood coefficient-wise on the chosen evaluation domain.

5.2 DECS (Degree-Enforcing Commitments)

The DECS layer is the degree-soundness backbone of SmallWood. It lets the prover commit (via a Merkle tree) to evaluations and then enforce that these evaluations come from polynomials of the targeted low degree, rather than from arbitrary high-degree data. In the overall argument, DECS appears both in the initial commitment phase and in the final opening phase.

Commitment phase. The prover interpolates each packed row into $P_j \in \mathbf{F}_q[X]_{\leq s+\ell-1}$, samples η masking polynomials $M_1, \ldots, M_\eta$ of the same degree bound, and commits to all evaluations on the NTT domain via a Merkle tree; the root root is the DECS commitment.

Deriving the batching matrix. From root, both parties deterministically derive a challenge matrix $\Gamma \in \mathbf{F}_q^{\eta \times r}$ (ROM/Fiat–Shamir with grinding counters), which binds masks to rows.

Opening and checks. For any opened set of NTT indices, the verifier (i) authenticates each leaf against root and (ii) checks per-slot masked linear relations

$$\mathrm{NTT}(R_k)[e] \stackrel{?}{=} M_k(e) + \sum_{j=0}^{r-1} \Gamma_{k,j}\, P_j(e) \pmod q,$$

together with $\deg R_k \leq s + \ell - 1$ for all $k \leq \eta$. This yields small-domain polynomial-binding and HVZK as in SmallWood's DECS formalization. (Algorithmic details: see Appendix A.)

Soundness knob and bound (DECS). Let N be the Merkle domain size and $d_{\text{decs}} = n_{\text{cols}} + \ell - 1$ the enforced degree. For one DECS round with a *uniform* challenge matrix distribution D_Γ over $\mathbf{F}_q^{\eta \times r}$, the polynomial-binding error is

$$\boxed{\varepsilon_1 \;=\; \frac{\binom{N}{d_{\text{decs}}+2}}{|\mathbf{F}_q|^\eta}}$$

More generally, SmallWood gives $\varepsilon_{\text{decs}} = \binom{N}{d_{\text{decs}}+2}\varepsilon_D + Q^2/2^{2\lambda}$ (Thm. 1), with $\varepsilon_D := \max_{v \neq 0, u} \Pr_{\Gamma \leftarrow D_\Gamma}[\Gamma v + u = 0]$; under uniform D_Γ, $\varepsilon_D = |\mathbf{F}_q|^{-\eta}$. In our NIZK composition, the global RO-collision term is accounted for later (see *Final NIZK soundness*) [8, Eq. (8), Thm. 1].

5.3 LVCS: committing rows and verifying linear maps end-to-end

The LVCS layer packages many row-polynomials together and gives the verifier linear access to (masked) linear maps on the rows, checked at verifier-chosen coordinates: from a single commitment, the verifier can ask for small batches of openings that certify linear relations among rows. Concretely, witness rows are arranged as low-degree polynomials and organized in layers, LVCS queries then certify linear combinations across these rows while deferring degree soundness to DECS.

Commit phase. Rows $r_1, \ldots, r_{n_{\text{rows}}} \in \mathbf{F}_q^s$ are interpolated into $P_1, \ldots, P_{n_{\text{rows}}}$ of degree $\leq s + \ell - 1$ with ℓ blinding points on Ω'. DECS yields a Merkle-root commitment root; this realizes an LVCS commitment to the matrix of rows.

Preparing masked linear forms. Given a coefficient matrix $C = \{c_{k,j}\}$, the prover forms $\bar{v}_k[i] = \sum_j c_{k,j}\,\text{mask}_j[i]$ for $i < \ell$. These masked prefixes match the slab opened in DECS, preserving HVZK while binding future function queries.

Evaluation phase. The verifier selects a tail set E of ℓ indices disjoint from Ω and the masked slab, requests DECS openings on both regions, checks the masked equalities $\sum_j c_{k,j} \cdot \text{Pvals}[i][j] \stackrel{?}{=} \bar{v}_k[i]$, and reconstructs $Q_k(X) = \sum_j c_{k,j} P_j(X)$ from the public prefix plus $\bar{v}_k$ to validate tail evaluations via LVCS/DECS. Soundness follows from DECS polynomial-binding with a Schwartz–Zippel tail test. (Algorithmic details: see Appendix A.)

Soundness knob and bound (LVCS). Let $d_{\text{row}} := n_{\text{cols}} + \ell - 1$ and let ℓ be the number of DECS openings per evaluation. SmallWood's LVCS function-binding error splits as

$$\boxed{\varepsilon_{\text{LVCS}} \;=\; \underbrace{\varepsilon_1}_{\text{DECS round above}} \;+\; \underbrace{\frac{\binom{d_{\text{row}}}{\ell}}{\binom{N}{\ell}}}_{\text{Schwartz–Zippel on the tail}}},$$

i.e., the LVCS layer inherits the DECS error ε_1 and adds the tail-check term $\binom{d_{\text{row}}}{\ell}/\binom{N}{\ell}$ (Thm. 3) [8]. In the round-by-round view used later, we will denote the tail term alone as $\varepsilon_4 := \binom{n_{\text{cols}}+\ell-1}{\ell}\Big/\binom{N}{\ell} = \binom{d_{\text{row}}}{\ell}\Big/\binom{N}{\ell}$ and keep ε_1 separate.

5.4 PCS and the PIOP interface

SmallWood-PCS wraps the LVCS/DECS stack into a standard polynomial commitment interface used by the PIOP. Compared to using DECS "directly" as a PCS, this wrapper allows openings at arbitrary field points (not just the N Merkle indices), which lowers verifier cost and lets us keep polynomial degrees smaller.

PCS from LVCS. SmallWood stacks LVCS so that evaluations of a vector polynomial $P = (P_1, \ldots, P_n)$ are obtained as LVCS openings via arranged coefficient blocks and blinding. This yields a hash-based PCS: correctness and HVZK inherit from LVCS/DECS, and polynomial-binding reduces to LVCS function-binding.

PIOP as a polynomial-oracle client. The PIOP treats the PCS as a polynomial oracle: *commit* asks PCS/LVCS/DECS to commit to (P, M); *query* asks PCS to open a few evaluations at Fiat–Shamir points. Parallel constraints F_j and aggregated families F'_u are combined into :

$$Q_i(X) = M_i(X) + \sum_j \Gamma'_{i,j}(X)\, F_j(X) + \sum_u \gamma'_{i,u}\, F'_u(X). \tag{1}$$

The verifier checks point relations at ℓ' random points and runs masked Ω-sum tests (repeated ρ times); with d_Q given by Eq. (2), SmallWood's PIOP soundness satisfies [8, Thm. 7].

Soundness knobs and bounds (PIOP). Let ℓ' be the number of oracle point checks and ρ the number of masked Ω-sum repetitions. With batching randomness Γ' drawn uniformly, SmallWood's PIOP soundness (Theorem 7) satisfies

$$\boxed{\varepsilon_{\mathrm{PIOP}} \;\le\; \underbrace{\frac{1}{|\mathbf{F}|^{\rho}}}_{\varepsilon_2:\ \Omega\text{-sum tests}} \;+\; \underbrace{\frac{\binom{d_Q}{\ell'}}{\binom{|S|}{\ell'}}}_{\varepsilon_3:\ \text{point checks}}},$$

where d_Q is given by Eq. (2) and $S \subseteq \mathbf{F} \setminus \Omega$ is the PIOP query set (default $|S| = |\mathbf{F}| - s$). Thus ρ and ℓ' are independent soundness levers [8, Thm. 7].

Lemma 5.1 (PACS degrees and d_Q)**.** *Let $d = \max_j \deg(f_j)$ and $d' = \max_u \deg(f'_u)$ for the parallel and aggregated constraint families, respectively. With packing factor s and ℓ' oracle openings, the maximal degree checked at PIOP openings is*

$$d_Q = \max\Big(d(\ell' + s - 1) + (s - 1),\ d'(\ell' + s - 1)\Big). \tag{2}$$

Proof sketch. *See SmallWood [8, Fig. 6 & Eq. (3)]: $F_j(X) = f_j(P(X), \Theta(X))$ and $Q_i(X) = M_i(X) + \sum_j \Gamma'_{i,j}(X) F_j(X) + \sum_u \gamma'_{i,u} F'_u(X)$, with $\deg \Gamma'_{i,j} \le s - 1$; Theorem 7 then applies.*

Fiat–Shamir compilation with grinding. We compile the public-coin protocol into a NIZK by hashing the transcript with domain-separated labels and short grinding counters, yielding SmallWood-ARK with straightline extractability in the ROM. (Algorithmic details: see Appendix A.)

Final NIZK soundness (with grinding). Let $\varepsilon_1, \varepsilon_2, \varepsilon_3, \varepsilon_4$ be the *interactive* round errors defined above (DECS, PIOP Ω-sum, PIOP point checks, LVCS tail). With grinding parameters $\kappa_1, \ldots, \kappa_4$ and random-oracle query counts $Q_0, \ldots, Q_4$, SmallWood gives the compiled NIZK bound

$$\boxed{\varepsilon_{\mathrm{ARK}} \;\le\; \frac{Q_0^2 + \cdots + Q_4^2}{2^{2\lambda}} \;+\; \frac{Q_1\,\varepsilon_1}{2^{\kappa_1}} \;+\; \frac{Q_2\,\varepsilon_2}{2^{\kappa_2}} \;+\; \frac{Q_3\,\varepsilon_3}{2^{\kappa_3}} \;+\; \frac{Q_4\,\varepsilon_4}{2^{\kappa_4}}},$$

so each κ_i directly recovers λ-bits of concrete soundness in its round (Thm. 9) [8].

Soundness knobs at a glance.

$$\varepsilon_1 = \frac{\binom{N}{d_{\text{decs}}+2}}{|\mathbf{F}|^{\eta}}, \quad \varepsilon_2 = \frac{1}{|\mathbf{F}|^{\rho}}, \quad \varepsilon_3 = \frac{\binom{d_Q}{\ell'}}{\binom{|S|}{\ell'}}, \quad \varepsilon_4 = \frac{\binom{n_{\text{cols}}+\ell-1}{\ell}}{\binom{N}{\ell}}.$$

Choose (η, N) for DECS, $(\rho, \ell', |S|)$ for PIOP, and (ℓ, n_{cols}) for LVCS; then set d_Q from Eq. (2). Final NIZK error uses Theorem 9 with grinding (κ_i) [8].

How range/norm subroutines integrate. Columnwise range-membership and norm-boundedness constraints (for u, x_0, x_1) are added among the *parallel* constraints and therefore must vanish at the opened PIOP points; they do not alter the aggregated families. This matches our "small-set + ℓ_2-bounded" witness language inherited from the vSIS–BBS framework.

Unlinkability from ROM zero-knowledge. A presentation publishes only the NIZK transcript (domain-separated FS rounds); no deterministic function of the target t is revealed. Hence SmallWood's ROM HVZK implies unlinkability for repeated showings in our language. For the witness structure and proof-friendly signatures we follow the vSIS line [4].

5.5 Small-field PACS/PIOP via an Extension Field

Idea. When the base field $\mathbf{F}$ is small, we lift the *PIOP algebra* to an extension field $K/\mathbf{F}$ of degree θ while keeping the *PCS/LVCS/DECS layer* over $\mathbf{F}$. We set

$$\ell' = 1 \quad \text{and} \quad \rho = 1,$$

so the verifier opens a single PIOP point over K and runs one Ω-sum check. With $S = K \setminus \Omega$, the PIOP soundness becomes

$$\varepsilon \;\le\; \frac{1}{|K|} \;+\; \frac{\binom{d_Q}{1}}{\binom{|K|}{1}} \;=\; \frac{1+d_Q}{|K|},$$

which is negligible as soon as $|K|$ exceeds the security target.

Specialized PIOP soundness (small field). With $\ell' = 1$ and $\rho = 1$ over $K/\mathbf{F}$ of size $|K|$, the PIOP error specializes to

$$\boxed{\varepsilon_{\text{PIOP}} \;\le\; \underbrace{\frac{1}{|K|}}_{\varepsilon_2} \;+\; \underbrace{\frac{\binom{d_Q}{1}}{\binom{|K|}{1}}}_{\varepsilon_3} \;=\; \frac{1+d_Q}{|K|}},$$

to be combined with ε_4 (LVCS tail over $\mathbf{F}$) and ε_1 (DECS) [8]. See also [8, §5.4] for the exact PACS/PIOP specialization over K and [8, Tab. 2] for the LVCS row/query accounting with θ-fold coordinate openings.

Witness polynomials over K, commitments over $\mathbf{F}$. The PIOP witness polynomials $P = (P_1, \ldots, P_n)$ and the single masking polynomial M live in $K[X]$. We commit to them using SmallWood LVCS over $\mathbf{F}$ by evaluating on a packing set $\Omega = \{\omega_1, \ldots, \omega_s\} \subset \iota(\mathbf{F}) \subset K$, where $\iota : \mathbf{F} \hookrightarrow K$ is the canonical embedding, so that $P_j(\omega_k) \in \mathbf{F}$ for all $k \le s$. Let $\varphi : \mathbf{F}^{\theta} \to K$ be a fixed $\mathbf{F}$-linear isomorphism. For each j, form a single *column*

$$A_j \;=\; (a_1, \ldots, a_{s+\theta})^{\top} \in \mathbf{F}^{s+\theta}$$

with

$$(a_1, \ldots, a_s) = (P_j(\omega_1), \ldots, P_j(\omega_s)), \qquad (a_{s+1}, \ldots, a_{s+\theta}) = \varphi^{-1}\big(P_j(\omega_{s+1})\big),$$

where $\omega_{s+1} \in K \setminus \Omega$ is one extra K-evaluation that supplies the PIOP's point randomness.

Clarification (from one K-evaluation to θ $\mathbf{F}$-openings). Fix an $\mathbf{F}$-basis $(\beta_t)_{t=1}^{\theta}$ of K compatible with φ. For any $e \in K$ there exists $v_e \in K^{\,s+\theta}$ (a K-linear functional depending only on e and Ω) such that $v_e \cdot A_j = P_j(e)$. Write $\varphi^{-1}(v_e) = (v_e^{(1)}, \ldots, v_e^{(\theta)})$ with $v_e^{(t)} \in \mathbf{F}^{s+\theta}$. Then

$$P_j(e) = v_e \cdot A_j = \sum_{t=1}^{\theta} \big\langle v_e^{(t)},\, A_j \big\rangle\, \beta_t,$$

so answering one evaluation over K boils down to *exactly θ parallel $\mathbf{F}$-linear LVCS openings* (one inner product per $\mathbf{F}$-coordinate). No extra randomness beyond the masking M is required to answer K-linear combinations.

Matrix shape, rows, and queries. Stacking $\{A_j\}_{j \le n}$ in layers yields the global LVCS matrix A over $\mathbf{F}$: the n witness columns occupy $\lceil n/n_{\text{cols}} \rceil$ layers of $(s+\theta)$ rows. The (single) masking polynomial over K contributes an extra block that expands into θ $\mathbf{F}$-rows per K-row. Altogether,

$$n_{\text{rows}} \;=\; \left\lceil \frac{n}{n_{\text{cols}}} \right\rceil (s+\theta) \;+\; \left\lceil \frac{d_Q}{n_{\text{cols}}} + 1 \right\rceil \theta\,, \tag{3}$$

where the "+1" accounts for the masking block. Since one evaluation over K materializes as θ $\mathbf{F}$-coordinate openings, the number of LVCS queries is

$$m \;=\; \left\lceil \frac{n}{n_{\text{cols}}} + 1 \right\rceil \theta\,. \tag{4}$$

Where layers run. The PIOP (oracle algebra, batching, and the single-point check) runs over K with $S = K \setminus \Omega$ and $\deg P_i \le s$ (since $\ell' = 1$), while the PCS/LVCS/DECS commitment, degree enforcement, and Merkle openings remain over $\mathbf{F}$ with unchanged DECS degree $d_{\text{decs}} = n_{\text{cols}} + \ell - 1$.

Row degree vs. relation degree. DECS enforces the *row degrees* $\deg P_i, \deg M_i \le n_{\text{cols}} + \ell - 1$ for the committed polynomials. This is distinct from the *relation degree* d_Q used in the PIOP soundness (Eq. (2)), which depends on the algebraic degrees of the constraint polynomials (f_j, f'_u) rather than on n_{cols}.

Why the single extra K-point suffices (intuition). The s packed values $P_j(\omega_1), \ldots, P_j(\omega_s) \in \mathbf{F}$ fix the degree-$\le s$ behavior on Ω, and the single additional value $P_j(\omega_{s+1}) \in K$ binds the evaluation outside Ω. Together they determine the K-linear functional v_e for any challenge $e \in K$ and let the verifier reconstruct $P_j(e)$ via the θ $\mathbf{F}$-coordinate openings described above.

Lemma 5.2 (Single K-point suffices)**.** *Let $P(X) \in \mathbf{F}[X]_{\le s+\ell-1}$ be a row-polynomial with evaluations fixed on $\Omega \subset \mathbf{F}$. Let $\omega_{s+1} \in K \setminus \Omega$ for an extension $K/\mathbf{F}$ of degree θ. Then for any $e \in K$, the K-linear functional $v_e : P \mapsto P(e)$ is determined by the θ $\mathbf{F}$-coordinate openings of $P(\omega_{s+1})$ together with the values on Ω.*

Proof sketch. With an $\mathbf{F}$-basis of K and the isomorphism $\varphi : \mathbf{F}^{\theta} \to K$, represent $P(\omega_{s+1})$ by its θ $\mathbf{F}$-coordinates. These, together with $\{P(\omega) : \omega \in \Omega\}$ and the interpolation relations, determine all K-linear functionals v_e; see SmallWood for the explicit matrix form and accounting [8, §5.4, Tab. 2].

6 Security-Claim Map

Claim	Assumption/Game	Where proved	External dep.
sEUF (non-blind)	$\mathrm{GenISIS}_f$ hardness	§??, §??	[4, Thm. 5]
One-more UF (blind)	$\mathrm{IntGenISIS}_f$	§?? (Blind issuance)	[4, Thm. 6, Fig. 1]
PACS knowledge soundness	ROM FS + ARK Thm. 9	§??	[8, Thm. 9]
ZK/HVZK	ARK definitions	§??	[8, §5]
Unlinkability (Show)	ZK + hiding	Prop. ??	standard

7 How Everything Merges Together: the Rational Hash and its Encoding in SmallWood

In this section, we will first describe the BBS-style rational hash and its proof-friendly identity, then explain how witness rows are packed and committed, how each constraint family is built and wired, how batches Q_i are formed and checked in the PIOP, and finally how the small-field variant changes costs and soundness.

7.1 Rational hash, target, and cleared identity

Definition (hash and ISIS target). Public parameters fix $A \in R_q^{n\times m}$ and $(B_0, b_1) \in R_q^n \times R_q^n$. For a message u and randomness $\chi = (x_0, x_1)$ we set

$$t \;:=\; h_{u,\chi}(B) \;=\; B_0 \cdot (1; u; x_0) \;\oslash\; \big(b_1 - \mathbf{1}_n \cdot x_1\big) \;\in R_q^n,$$

and the preimage relation is $As \equiv t \pmod q$ with $\|s\| \le \beta$

Guard (denominator invertibility). We enforce slot-wise invertibility of $\big(b_1 - \mathbf{1}_n \cdot x_1\big)$ by resampling x_1 until every slot is invertible in R_q; this realizes the message-subspace guard and legitimizes the cleared form below.

Exact evaluation (implementation contract). We compute t in the evaluation/NTT domain and keep it there until needed downstream:

Algorithm 2 $\mathsf{ComputeBBSHash}(B_0, b_1; u, x_0, x_1) \to t$ (evaluation/NTT domain)

1: Lift u, x_0, x_1 to the evaluation domain.
2: $r \leftarrow B_0^{(\mathrm{const})} \;+\; B_0^{(u)} \odot u \;+\; B_0^{(x)} \odot x_0; \quad d \leftarrow b_1 - x_1.$
3: If any slot of d is non-invertible, resample x_1; else set $t \leftarrow r \odot d^{-1}$.
4: **return** t.

Cleared identity (degree-≤ 2). Writing $X_1 = \mathbf{1}_n \cdot x_1$, the equality $As \equiv t$ is equivalent (under the guard) to

$$(b_1 \odot A)\, s \;-\; (As)\, X_1 \;-\; B_0 \cdot (1; u; x_0) \;\equiv\; 0 \pmod q, \tag{5}$$

which has total degree ≤ 2 in (s, u, x_0, x_1). This is the main equality constraint we will encode in PACS/PIOP.

7.2 Packing model and committed rows

Grid and degrees. Let $\Omega = \{\omega_1, \ldots, \omega_s\} \subset \mathbf{F}_q$ be the evaluation grid (take distinct NTT nodes, with $S_0 := \sum_{\omega \in \Omega} 1 = |\Omega| \not\equiv 0 \pmod{q}$). Each committed row is interpolated to a polynomial with degree $\leq s + \ell - 1$ by fixing its values on Ω and ℓ blinded values on $S \subset \mathbf{F}_q \setminus \Omega$. DECS enforces the degree bound.

Packing policy. Signature rows s use *coefficient packing*: column j carries the j-th coefficient of the ring element. The message/randomness rows (u, x_0) are *column-constant* (degree-0) rows over Ω. The scalar x_1 is carried by a column-constant row X_1.

Witness blocks (to localize products). We form the per-column blocks

$$w_1 := (s, u, x_0), \qquad w_2 := x_1, \qquad w_3 := w_1 \cdot w_2,$$

with the product taken slot-wise. Denote by $W_1(X), W_2(X), W_3(X)$ the corresponding row-polynomial stacks.

Public tables as row polynomials. The public linear maps A, $(b_1 \odot A)$ and the routed coefficients from B_0 are similarly lifted to *public* row polynomials: $A_j(X)$, $(b_1 \odot A)_j(X)$, and $B^{(\cdot)}_{0,j}(X)$.

7.3 Constraint assortment and wiring

Now, let us describe the exact constraint families and show how they wire to (5) in order to have them verify the signature validity.

(C1) Parallel product gates (slot-wise, quadratic). For every gated component t in w_1,

$$F^{\mathrm{par}}_t(X) \;:=\; W_3^{(t)}(X) \;-\; W_1^{(t)}(X)\, W_2(X) \;\equiv\; 0 \tag{6}$$

pins $W_3 = W_1 \cdot W_2$ on Ω (and at fresh points via the PIOP tail).

(C2) Aggregated cleared identity (row-polynomial form). For each public row index j,

$$\begin{aligned} F^{\mathrm{bbs}}_j(X) := & \underbrace{\sum_{k \in \mathcal{S}} (b_1 \odot A)_{j,k}(X)\, S_k(X)}_{(b_1 \odot A)s} \\ & \underbrace{- \sum_{k \in \mathcal{S}} A_{j,k}(X)\, W_3^{(k)}(X)}_{(As)\, x_1} \\ & \underbrace{- \Big(B^{(0)}_{0,j}(X) + \sum_{t \in \mathcal{U}} B^{(u,t)}_{0,j}(X)\, U_t(X) + \sum_{\tau \in \mathcal{X}} B^{(x_0,\tau)}_{0,j}(X)\, X_{0,\tau}(X) \Big)}_{B_0 \cdot (1; u; x_0)} \equiv 0. \end{aligned} \tag{7}$$

Here $\mathcal{S}$ indexes signature rows, $\mathcal{U}$ message rows, and $\mathcal{X}$ the x_0 rows. This is the packed analogue of (5).

(C3) Alphabet membership for (u, x_0, x_1) (parallel, high-degree but bounded). For each $P \in \{u, x_0, x_1\}$ we enforce a single vanishing-polynomial row

$$P_B\big(P(\cdot)\big) \;\equiv\; 0 \quad \text{on } \Omega, \qquad P_B(X) = \prod_{i=-B}^{B} \big(X - \langle i \rangle_q\big).$$

(C4) ℓ_∞ chain for signature rows (parallel). For each signature row and each column value v we write a balanced-radix decomposition with digit membership, carries, and recomposition:

$$P_{\mathcal{D}}(d_k) = 0, \qquad R\,d_k + c_k = d'_k + c_{k+1}, \qquad v = \sum_{k=0}^{L-1} R^k d_k,$$

with $R = 2^W$ and balanced digit sets. The digit vanishing degree drives the parallel-degree parameter d, while carry/recomposition are quadratic. This certifies the ℓ_∞ bound used by your global ℓ_2 budget.

Low-degree vs. degree-constrained. (C1)–(C2) are genuinely low-degree; (C3)–(C4) are high-degree but degree-bounded. The maximal degree among parallel constraints determines the PIOP degree parameter d_Q.

7.4 Batching, masks, and PIOP checks

What the PIOP sees. All rows live under the same LVCS/DECS commitment. The verifier checks linear combinations at verifier-chosen coordinates; PCS wraps these into point openings.

Batch polynomials. We form ρ batches

$$Q_i(X) \;=\; M_i(X) \;+\; \sum_t \Gamma'_{i,t}\, F_t^{\mathrm{par}}(X) \;+\; \sum_j \gamma'_{i,j}\, F_j^{\mathrm{bbs}}(X),$$

with Fiat–Shamir weights (Γ', γ'). Each M_i is picked so that $\Sigma_\Omega Q_i = 0$.

Mask construction (large-field case). Let $S_k := \sum_{\omega\in\Omega} \omega^k$. Choose random $a_1, \ldots, a_{d_Q}$ and set

$$a_0 \;=\; -\,S_0^{-1}\Big[\sum_{k=1}^{d_Q} a_k\, S_k \;+\; \sum_t \Gamma'_{i,t}\, \Sigma_\Omega F_t^{\mathrm{par}} \;+\; \sum_j \gamma'_{i,j}\, \Sigma_\Omega F_j^{\mathrm{bbs}}\Big],$$

then $M_i(X) = \sum_{k=0}^{d_Q} a_k X^k$. The verifier checks Q_i on Ω and at fresh tail points and enforces $\Sigma_\Omega Q_i = 0$.

Soundness knobs (pointer). The error splits into DECS degree, masked-sum, tail, and LVCS tail-binding components; the degree term uses the *true* degrees from (C3)–(C4). Parameters instantiations will be discussed later.

7.5 Small-field variant

Let $K/\mathbf{F}$ be an extension of degree θ. We run the PIOP over K while keeping LVCS/DECS over $\mathbf{F}$.

How the interface works. A single evaluation in K expands to its θ $\mathbf{F}$-coordinates for LVCS queries (row/query replication by θ). Batches live in $K[X]$, masks M_i are built in $K[X]$ so that $\sum_{\omega\in\Omega} Q_i(\omega) = 0$ holds *in* K, and DECS enforces the same row-degree bound as in the large-field case.

Gains and trade-offs. Because $|K| = |\mathbf{F}|^\theta$, we can lower both the amount of tail points ℓ' tail point and masked-sum batch ρ while preserving the target soundness (PIOP error $\le (1+d_Q)/|K|$). This reduces challenge material. The trade-off is a factor-θ replication of LVCS queries/rows (over $\mathbf{F}$), but the total opening count still drops when θ is chosen to meet the security level.

8 Optimizations for the SmallWood Implementation

This section explains the proof–size optimizations we implement, why they are sound, and how they reduce the two dominant payloads of the argument. *RowOpening* authenticates a set of row leaves under the rows commitment and carries, per opened leaf, the residue (for data rows and mask rows), the leaf index and nonce, and the Merkle authentication material. *MOpening* is the analogous opening for the mask commitment on tail leaves. Everything else in the transcript is comparatively small. Our strategy is purely structural: eliminate redundancy, share Merkle path material across leaves, and encode residues/descriptors compactly—leaving all algebraic checks (DECS/LVCS/PCS/PIOP) unchanged.

Trimming and extra FS hash. Following SmallWood-ARK, we elide redundant evaluations by adding an extra FS hash and reconstruct omitted Q/R coefficients. This yields

$$|\pi| \;=\; |\pi_{\text{naive}}| + 2\lambda \;-\; \big(\eta\ell + \rho(\ell'+1) + m\ell\big)\log_2 |\mathbf{F}|\,,$$

with straightline extraction and final soundness as in [8, Thm. 9]. Our small-field choice aligns with [8, §5.4, Tab. 2].

Trimming MR to d_Q+1 coefficients The masked rows R_k are degree bounded by construction. We serialize only the first d_Q+1 coefficients of each R_k and the verifier reconstructs the tail as zeros. MR therefore drops from ηN elements to $\eta(d_Q{+}1)$. The degree check already present in the verification path guarantees soundness.

Single combined row opening Masked and tail indices are carried in one opening for the row commitment. The verifier splits this combined opening when it enforces masked equalities on the head slab and tail equalities on the challenged indices. This enables node sharing across the two subsets and avoids duplicated headers. *Verifier rule.* Given the union of masked/tail positions, Verify recomputes the root *only* from the **sorted, deduplicated** union and **rejects** if any descriptor references a node outside the union slice at that level/side.

Sixteen byte Merkle nodes Merkle nodes are represented as 16-byte strings derived with the XOF `SHAKE` and truncated to 128 bits (collision resistance aligned with λ=128).[1] This halves path bytes versus 32-byte nodes while preserving the target λ. At $N = 1024$ the depth is 10 which yields a saving of about 160 bytes per opened leaf before any sharing.

Leaner leaf headers Residues are stored in 32 bit lanes since $q < 2^{20}$. The leaf index fits in 16 bits for $N = 1024$. The per leaf nonce is 16 bytes. These choices shrink the fixed header of every leaf and directly benefit both RowOpening and MOpening.

Merkle multiproof

Principle. Given leaves $L[0], \ldots, L[N-1]$ and an opening set S, the prover builds one global sibling pool and per-leaf descriptors. Concretely, `EvalOpen` deduplicates every sibling encountered across all paths in S into a single slice `Nodes` and, for each opened leaf t, records the per-level indices `PathIndex`$[t][lvl]$ that point into `Nodes`.

Correctness. The verifier side first normalises the proof: `EnsureMerkleDecoded` expands any compact/frontier form back into the canonical `Nodes` and `PathIndex` so consumers can iterate per-leaf paths. It then reconstructs each path for leaf t by fetching the sibling bytes at

[1] Any XOF/HASH with at least 128-bit collision resistance is acceptable; the choice does not affect algebraic checks.

level *lvl* as `Nodes`[`PathIndex`[t][*lvl*]] and passes the resulting chain to `VerifyPath`, which rehashes level by level to the public root.

Savings. Compared to sending $|S| \log_2 N$ raw siblings, the union `Nodes` removes duplicates across leaves, while `PathBits` shrinks descriptors and the frontier form replaces many explicit siblings with bitmaps plus a smaller deduplicated pool.

Residue and descriptor packing

Twenty bit residues. Since $\lceil \log_2 q \rceil \leq 20$, residues are bit packed into 20 bit streams in row major order. This replaces 32 bit lanes with a representation that uses 62.5 percent of the bytes. The verifier reads either the packed form or a materialised slice through uniform accessors.

Dropping derivable items

All challenges and points that are deterministically derived from the transcript are recomputed by the verifier. This includes the tail indices, the extra K point in the small field path, and the LVCS coefficient matrix. The mask mixing seed is also dropped since it is derived from commitment roots. This reduces size and removes alternative encodings of the same proof.

Size model and main levers

Let depth $= \log_2 N$, `node_len` $= 16$, `index_len` $= 2$, `nonce_len` $= 16$. For the combined RowOpening

$$\begin{aligned}\text{RowOpening} \;\approx\; & 2\ell \cdot \Big((r+\eta)\frac{\text{bits}}{8} + \texttt{index\_len} + \texttt{nonce\_len} \Big) \\ & + \; |\texttt{Nodes}| \cdot \texttt{node\_len} \; + \; \sum_{s \in S} \big|\texttt{PathIndex}[s]\big|,\end{aligned}$$

with bits $\in \{20, 32\}$. MOpening mirrors the same structure with ℓ leaves and the mask row count of the instance. After trimming, MR contributes $\eta(d_Q{+}1)$ elements. The largest size levers are the layout parameters that control how many values exist and are opened. The encoding improvements above shrink the opening terms uniformly across all runs.

Verifier-side canonical encodings (required). To prevent malleability and cross-run linkable quirks, Verify enforces:

- Sorted, deduplicated opening index sets (per round).
- The union opening set defines the only admissible indices; reject if any path descriptor points outside this set at its (level,side).
- Merkle node size fixed at **16 bytes**; paths are fully specified with level/side tags.
- Residue packing fixed to **20-bit** limbs where used; no alternative packings accepted.
- Fiat–Shamir transcripts include round labels and explicit grinding counters κ_i (reject if missing).

9 Parameter Constraints and Chosen Parameters

Parameters are governed by three classes of constraints: (i) sampler/statistical (*smoothing*, per-slot σ); (ii) vSIS–BBS correctness (cleared identity, invertibility budget); (iii) SmallWood soundness (DECS/LVCS and PACS budgets). We summarize each and point to §10 for the instantiated bundles.

This section summarizes the constraints that govern parameters across all building blocks (vSIS–BBS, the Gaussian preimage sampler, and the SmallWood proof system), and states a concrete parameter set we use in our implementation.

9.1 Global constraints

Smoothing factor. $\eta_\varepsilon(\mathbb{Z}^{2\varphi}) = \sqrt{\ln\big(4\varphi(1+2^\lambda)\big)/\pi} \approx 5.558.$

Smoothing and sampler widths. Let $\eta_\varepsilon(\Lambda)$ denote the smoothing parameter. For statistical hiding in hash-and-sign and for Gaussian preimage sampling, it suffices to choose per-slot widths $\sigma_k[i]$ so that $\sigma_k[i] \geq \sigma_{\min} \geq \eta_\varepsilon\big(\Lambda_\perp(\mathbf{A})\big)$ (see §4). In our hybrid sampler, σ scales with the NTRU trapdoor quality α as $\sigma \approx \alpha\sqrt{q}$.

Definition 9.1 (Admissibility)**.** A parameter tuple $(q, B, \beta, \{\sigma_k\}, N, n_{\text{cols}}, d_Q, \eta, \rho, \ell, \ell')$ is *admissible* if

1. $B \ll q$, so that vanishing-polynomial checks over $[-B, B]$ do not wrap modulo q;
2. each sampler width satisfies $\sigma_k[i] \geq \eta_\varepsilon\big(\Lambda_\perp(\mathbf{A})\big)$ for the relevant cosets;
3. the invertibility condition $\Pr[(b_1 - \mathbf{1}_\varphi \cdot x_1)[j] = 0] \leq B/q$ holds, enabling rejection sampling on x_1;
4. SmallWood degree bounds are respected: $d_{\text{decs}} = n_{\text{cols}} + \ell - 1$, the mask degree d_Q follows Eq. (2), and the DECS domain size N matches the commitment grid.

Admissibility reference. All parameter choices below are made to satisfy Def. 9.1 (no wrap-around, invertibility budget, per-slot smoothing).
Checklist.

- $B \ll q$ so vanishing polynomials do not wrap modulo q.
- Per-slot sampler width $\sigma(\cdot) \geq \eta_\varepsilon$ on the relevant cosets.
- DECS domain N and degree bounds (d_{decs}, d_Q) satisfy the prescribed inequalities (Eq. (2) and [8]).

Range bounds and norm constraints. We restrict u, x_0, x_1 coefficient-wise to $[-B, B]$ using vanishing-polynomial membership checks (SmallWood parallel constraints, §5). For the signature vector s we enforce $\|s\|_\infty \leq \beta$ (implied by the global ℓ_2 bound below).

SmallWood soundness budget (explicit). We map $(\rho, \ell', \ell, N, n_{\text{cols}}, d_Q, \eta)$ to SmallWood's four error terms:

$$\varepsilon_{\text{decs}} \leq \binom{N}{d_{\text{decs}}+2} q^{-\eta}, \quad \varepsilon_{\text{sum}} = q^{-\rho}, \quad \varepsilon_{\text{tail}} = \frac{\binom{d_Q}{\ell'}}{\binom{|S|}{\ell'}}, \quad \varepsilon_{\text{lvcs}} = \frac{\binom{n_{\text{cols}}+\ell-1}{\ell}}{\binom{N}{\ell}},$$

and use Fiat–Shamir grinding counters $(\kappa_1, \ldots, \kappa_4)$ as in [8] to achieve $\varepsilon \leq 2^{-\lambda}$. Instantiated values are listed in §10.

ISIS relation and cleared identity. Correctness requires the ISIS preimage relation $As \equiv h_{u,\chi}(B) \pmod q$ and the cleared identity of Eq. Equation (5). These drive the linear and quadratic rows in SmallWood.

Full-witness ℓ_2 bound. If $u, x_0, x_1 \in [-B, B]$ coefficient-wise, then

$$\|(s, u, x_0, x_1)\|_2^2 \;\le\; B_s^2 \;+\; (kN)B^2 \;+\; (k_0N)B^2 \;+\; NB^2,$$

for a preimage budget B_s derived from the SIS/vSIS parameters (see §7).
Remark (resampling cost). With $x_1 \leftarrow \mathrm{Unif}([-B, B])$ independent of $b_1[j]$, we have $\Pr[b_1[j] - x_1 \equiv 0 \pmod q] \le (2B{+}1)/q$ per slot; the expected trials are $(1 - (2B{+}1)/q)^{-1}$.

Lemma 9.2 (Invertibility by resampling)**.** *Let x_1 be sampled uniformly from $[-B, B]^N$ with $B \ll q$ and $b_1 \leftarrow U(R_q^{\varphi})$ be public. Then*

$$\Pr\left[\exists j : (b_1 - \mathbf{1}_{\varphi} \cdot x_1)[j] = 0\right] \;\le\; N \cdot \frac{2B+1}{q}.$$

Resampling x_1 until invertible succeeds in expected at most $1/\big(1 - N(2B{+}1)/q\big)$ trials.

9.2 Concrete parameters

Below is the parameter instantiation for the lattice signature. The parameters were tuned using the open source lattice-estimator project [1] and we did not change them throughout the performance testing of the Zero-Knowledge Proof.

Knob / Result	**Symbol**	**Value (this instantiation)**
Ring degree	(φ)	1024
Modulus	(q)	1038337
Security target	(λ)	128 (targeting ≥ 128 bits)
Smoothing factor (natural log)	$\eta_\varepsilon(\mathbb{Z}^{2\varphi})$	$\sqrt{\frac{\ln\left(4\varphi(1+2^{\lambda})\right)}{\pi}} \;\approx\; 5.558$
Trapdoor constant	(c_{trap})	1.17
Preimage Gaussian	(s)	$s = c_{\mathrm{trap}}\sqrt{q}\,\eta_\varepsilon \;\approx\; 6.626 \times 10^3$ $(\log_2 s \approx 12.69)$
Sampler Standard deviation	(σ)	$\sigma = \frac{s}{\sqrt{2\pi}} \;\approx\; 2.643 \times 10^3$ $(\log_2 \sigma \approx 11.37)$
Message / randomness arity	(ℓ_m, ℓ_r)	$(1, 2)$
Verifier bound	(β)	$\beta \ge s\sqrt{3 + \ell_m + \ell_r}\sqrt{\varphi} \;\approx\; 5.194 \times 10^5$ $(\log_2 \beta \approx 18.99)$
Estimated security	–	≥ 128 bits (≈ 193)

Table 1: Parameters for $n = \varphi = 1024$ and $q = 1038337$. Logs are base-2 unless 'ln' is explicit.

We instantiate $(\ell, \ell', \rho, d_Q, n_{\mathrm{cols}}, N, \eta, \kappa_i)$ in §10, satisfying the budget above.

10 Experimental Results

Scope. We report end-to-end prover time, proof size, and soundness (in bits, via the composition in §5.4). All runs use the small-field variant with extension degree $\theta > 1$; the opening degree d_Q is as defined in Eq. (2). Mechanistic details of PACS/PIOP, DECS/LVCS, and the small-field compilation are recalled earlier; here we focus on the measured trade-offs.

Measurement grid. We sweep the main knobs that influence communication and soundness : packing $(n_{\mathrm{cols}}, \ell)$, point checks ℓ', masks ρ, extension degree θ, and DECS window η. We report the resulting sizes, prover time (Time taken to build the NIZK Transcript), and bit security.

ProofKB	Time (s)	Bits	NCols	ℓ	ℓ'	ρ	θ	η
9.32	0.266	133.44	6	22	2	2	4	17
10.07	0.228	146.93	4	24	2	2	4	17
11.52	0.250	133.22	4	28	2	2	4	17
14.06	0.630	198.01	4	34	8	5	2	26
14.17	0.654	192.26	6	34	8	5	2	26
14.24	0.670	198.48	4	34	8	6	2	26

Table 2: Prover time, proof size, and achieved soundness across packing and PIOP knobs (small-field, $\theta > 1$). Sizes exclude public parameters.

Findings. We measure the results for two different regimes/soundness parameters goals : First, a compact/low-latency band around $\lambda \geq 128$ bits, obtained with $\theta = 4$ and very few point checks ($\ell' = 2$), yielding 9–12 KB proofs and ≈ 0.23–0.27 s proving. Within this band, retuning (n_{cols}, ℓ) shifts load between row openings and query payloads while keeping the opening degree governed by Eq. (2), explaining the modest variation in size and bits. Second, a high-security band at $\lambda \geq 192$ bits, reached with $\theta = 2$, a higher $\ell' = 8$, several masks ($\rho \in \{5, 6\}$), and a wider DECS window, giving ≈ 14 KB proofs in ≈ 0.63–0.67 s. The movement from $(\theta, \ell', \rho) = (4, 2, 2)$ to $(2, 8, 5\text{–}6)$ trades replication against per-check entropy exactly as predicted by the earlier calculus, without altering the underlying construction.

Guidance. For $\lambda \approx 128$–147 bits, $(n_{\text{cols}}, \ell, \ell', \rho, \theta, \eta) = (6, 22, 2, 2, 4, 17)$ or $(4, 24, 2, 2, 4, 17)$ delivers 9–10 KB proofs with the fastest proving. For $\lambda \approx 192$–198.5 bits, $(4, 34, 8, 5\text{–}6, 2, 26)$ attains ≈ 14 KB at sub-second proving. Further knob interactions are as discussed earlier.

A Algorithmic details for DECS/LVCS/PIOP

Algorithm 3 $\mathsf{DECS.DeriveGamma}(\mathsf{root}, \eta, r, q) \to \Gamma$ *(Verifier and Prover)*

Initialize 64-bit counter $ctr \leftarrow 0$; let $L \leftarrow \left\lfloor \frac{2^{64}}{q} \right\rfloor \cdot q$ ▷ bias-free sampling bound
for $k = 0$ to $\eta - 1$ **do** ▷ one mask-row combination per k
  for $j = 0$ to $r - 1$ **do** ▷ one coefficient per committed row
    repeat
      $x \leftarrow \mathsf{SHA256}(\mathsf{root} \parallel \mathsf{LE64}(ctr))$ interpreted as 64-bit little-endian
      $ctr \leftarrow ctr + 1$
    until $x < L$ ▷ rejection ensures uniform mod-q
    $\Gamma_{k,j} \leftarrow x \bmod q$
return Γ

Algorithm 4 DECS.VerifyEval(root, Γ, R, open) *(Verifier)*

Require: Ring R_q with modulus q and size N; params $(d, \eta, \mathsf{NonceBytes})$; row count r fixed at setup.

Require: $R = \{R_k\}_{k<\eta}$ in coefficient domain; opening open = (Indices, Pvals, Mvals, Paths, Nonces).

Ensure: ACCEPT/REJECT.

1: **(Shape checks)** For each opened slot t: check $\mathrm{Pvals}[t] \in [0,q)^r$, $\mathrm{Mvals}[t] \in [0,q)^\eta$, and $|\mathrm{Nonces}[t]| = \mathsf{NonceBytes}$; reject on mismatch.

2: **(Precompute NTT of R)** For each $k < \eta$, set $R_k^\star \leftarrow \mathrm{NTT}(R_k)$.

3: **for** $t = 1$ to $|\mathrm{Indices}|$ **do**

4: $\quad e \leftarrow \mathrm{Indices}[t]$; reject if $e \notin \{0, \ldots, N-1\}$.

5: $\quad$ **(Merkle authentication)** Build

$$\mathrm{leaf}(e) \;:=\; \big(\mathrm{Pvals}[t][0..r-1] \;;\; \mathrm{Mvals}[t][0..\eta-1] \;;\; e \;;\; \mathrm{Nonces}[t]\big).$$

6: $\quad$ Verify $\mathsf{VerifyPath}(\mathrm{leaf}(e), \mathrm{Paths}[t], \mathsf{root}, e)$; reject on failure.

7: $\quad$ **(Masked relation)** For each $k < \eta$:

$$\mathrm{rhs}_k \leftarrow \mathrm{Mvals}[t][k] \;+\; \sum_{j=0}^{r-1} \Gamma_{k,j} \cdot \mathrm{Pvals}[t][j] \mod q, \qquad \mathrm{lhs}_k \leftarrow R_k^\star[e];$$

require $\mathrm{lhs}_k = \mathrm{rhs}_k$ for all k; else reject.

8: **return** ACCEPT

Algorithm 5 LVCS.CommitInitWithParams(ringQ, $\{r_j\}_{j<r}$, ℓ, params) $\to$ (root, ProverKey) *(Prover)*

Require: ring R_q of size N, rows $r_j \in \mathbf{F}_q^{n_{\mathrm{cols}}}$, $\ell > 0$, DECS params (Degree $= d$, η, NonceBytes)

1: **(Row masks)** For each $j < r$, sample $\mathrm{mask}_j \in \mathbf{F}_q^\ell$ uniformly ▷ HVZK blinding points

2: **(Interpolate rows)** For each $j < r$, set $P_j(X) \leftarrow \mathsf{interpolateRow}(\mathrm{ringQ}, r_j, \mathrm{mask}_j, n_{\mathrm{cols}}, \ell)$

3: **(Commit via DECS)** Initialize a DECS prover on $\{P_j\}$ and obtain Merkle root root

4: **(Derive Γ)** Compute $\Gamma \leftarrow \mathsf{DECS.DeriveGamma}(\mathsf{root}, \eta, r, q)$

5: **(Cache)** Store NTT lifts of P_j and of the DECS masks for later openings

6: **return** (root, ProverKey$\{\cdots\}$)

Algorithm 6 LVCS.CommitStep1(root) $\to \Gamma$ and LVCS.AcceptGamma(Γ) *(Verifier)*

1: Store root (bind the session)

2: Derive $\Gamma \leftarrow \mathsf{DECS.DeriveGamma}(\mathsf{root}, \eta, r, q)$

3: Optionally replace with application-provided Γ via AcceptGamma (determinism/debug)

4: **return** Γ

Algorithm 7 LVCS.CommitStep2($\{R_k\}_{k<\eta}$) $\to \{\mathrm{ACCEPT}, \mathrm{REJECT}\}$ *(Verifier)*

1: **for** each R_k **do** ▷ prover-sent masked rows in coefficient form

2: $\quad$ Reject if $\deg(R_k) > d$ ▷ quick degree guard before any openings

3: **return** ACCEPT

Algorithm 8 LVCS.EvalInitMany(ringQ, ProverKey, $\{\text{Coeffs}_k\}_{k<m}) \to \{\bar{v}_k\}_{k<m}$ *(Prover)*

Require: m linear forms with coefficients $\text{Coeffs}_k \in \mathbf{F}_q^r$

$q \leftarrow \text{ringQ.Modulus}[0]$; $\ell \leftarrow |\text{mask}_j|$
for $k = 0$ to $m-1$ **do**
  $\bar{v}_k \in \mathbf{F}_q^\ell \leftarrow \mathbf{0}$
  for $j = 0$ to $r-1$ **do**
    for $i = 0$ to $\ell - 1$ **do**
      $\bar{v}_k[i] \leftarrow \mathsf{MulAddMod64}(\bar{v}_k[i], \text{Coeffs}_k[j], \text{mask}_j[i], q)$
return $\{\bar{v}_k\}_{k<m}$

Algorithm 9 LVCS.ChooseE(ℓ, n_{cols}) $\to E$ *(Verifier)*

Require: $n_{\text{cols}} = |\Omega|$; ring size N; masked slab is $[n_{\text{cols}}, n_{\text{cols}} + \ell)$

Ensure $0 < \ell \le N - n_{\text{cols}}$; set $maskStart \leftarrow n_{\text{cols}}$, $maskEnd \leftarrow n_{\text{cols}} + \ell$
Sample $E \subset \{maskEnd, \ldots, N-1\}$ with $|E| = \ell$ uniformly without replacement
return E

Algorithm 10 LVCS.EvalFinish(ProverKey, E) $\to$ Opening *(Prover)*

open $\leftarrow$ ProverKey.DecsProver.EvalOpen(E)
return Opening{DECSOpen = open}

Algorithm 11 LVCS.EvalStep2($\{\bar{v}_k\}_{k<m}$, E, open, C, v_{targets}) $\to \{\text{ACCEPT}, \text{REJECT}\}$ *(Verifier)*

Require: $\bar{v}_k \in \mathbf{F}_q^\ell$, E with $|E| = \ell$, opening open, coeffs $C \in \mathbf{F}_q^{m \times r}$, public prefixes $v_{\text{targets}} \in (\mathbf{F}_q^{n_{\text{cols}}})^m$

$(maskStart, maskEnd) \leftarrow (n_{\text{cols}}, n_{\text{cols}} + \ell)$; split open into maskOpen for $[maskStart, maskEnd)$ and tailOpen for E; reject on index anomalies
DECS: Verify DECS.VerifyEvalAt on both maskOpen and tailOpen; reject on failure
Masked relations: For each $k < m$ and each masked $i < \ell$, check

$$\sum_{j=0}^{r-1} C[k,j] \cdot \text{maskOpen.Pvals}[i][j] \stackrel{?}{=} \bar{v}_k[i] \pmod q$$

Reconstruct Q_k**:** For each $k < m$,
  $Q_k(X) \leftarrow \mathsf{interpolateRow}(\text{ringQ}, v_{\text{targets}}[k], \bar{v}_k, n_{\text{cols}}, \ell)$; $Q_k^\star \leftarrow \text{NTT}(Q_k)$
Tail checks: For each opened tail index $idx \in E$ and each $k < m$,

$$Q_k^\star[idx] \stackrel{?}{=} \sum_{j=0}^{r-1} C[k,j] \cdot \text{tailOpen.Pvals}[\cdot][j] \pmod q$$

return ACCEPT

Algorithm 12 PACS.PIOPCommitOracle$_{\text{Prover}}$(params) $\to$ pcsCom

1: **(Prover)** Interpolate each row of the packed witness into $P_1, \ldots, P_n \in \mathbf{F}[X]_{\le s+\ell'-1}$ with ℓ' random blind evaluations outside Ω. ▷ HVZK
2: **(Prover)** Sample $M_1, \ldots, M_\rho \in \mathbf{F}[X]_{\le d_Q}$ s.t. $\sum_{\omega\in\Omega} M_i(\omega) = 0$ for all i. ▷ Row-sum masks
3: **(Prover→PCS)** Call PCS.Commit$(P \| M)$; receive pcsCom (an opaque handle that includes LVCS/DECS state).
4: **return** pcsCom.

Algorithm 13 PACS.PIOPBatchCombine$_{\text{Prover}}$(pcsCom, Γ') $\to Q$

Require: Batching randomness $\Gamma' = \big(\{\Gamma'_{i,j}(X)\}_{i\le\rho,\, j\le m_1},\ \{\gamma'_{i,u}\}_{i\le\rho,\, u\le m_2}\big)$.
1: **(Prover)** Using local P, M, form for each $i = 1, \ldots, \rho$:

$$Q_i(X) \ :=\ M_i(X) \ +\ \sum_{j=1}^{m_1} \Gamma'_{i,j}(X)\, F_j(X) \ +\ \sum_{u=1}^{m_2} \gamma'_{i,u}\, F'_u(X),$$

where F_j, F'_u are the parallel/aggregated constraint polynomials evaluated symbolically on P. ▷ No openings
2: **(Prover)** Output $Q = (Q_1, \ldots, Q_\rho)$.

Algorithm 14 PACS.PIOPOpenAndCheck(pcsCom, Q) $\to \{\text{ACCEPT}, \text{REJECT}\}$

1: **(Verifier)** Sample query set $E' \subset S$ with $|E'| = \ell'$; send E' to the prover (Fiat–Shamir in NIZK).
2: **(Prover→PCS)** Ask PCS.Eval(pcsCom, E', $P|_{E'}$, $M|_{E'}$) to open the oracle at E'.
3: **(Verifier)** Check degree bounds (via DECS) and that

$$\forall i,\ \forall e \in E' : \quad Q_i(e) \ \stackrel{?}{=}\ M_i(e) \ +\ \sum_{j} \Gamma'_{i,j}(e)\, F_j(e) \ +\ \sum_{u} \gamma'_{i,u}\, F'_u(e).$$

4: **(Verifier)** Using the PCS/LVCS view of (P, M) on Ω, verify $\sum_{\omega\in\Omega} Q_i(\omega) = 0$ for all i. ▷ Masked Ω-sum test
5: **return** ACCEPT/REJECT.

Algorithm 15 PACS.ToNIZK (*Fiat–Shamir wrapper with grinding*)

1: **(Prover)** Sample salt $\in \{0,1\}^{2\lambda}$.
2: Derive challenges by hashing the running transcript with proof-of-work (grinding) counters:

$$\begin{aligned}
h_1 &\leftarrow \mathsf{XOF}_1(\mathsf{salt} \,\|\, \mathsf{PCS.Root}); & \Gamma &\leftarrow \mathsf{XOF}'_1(h_1)\\
h_2 &\leftarrow \mathsf{XOF}_2(h_1 \,\|\, R); & \Gamma' &\leftarrow \mathsf{XOF}'_2(h_2)\\
h_3 &\leftarrow \mathsf{XOF}_3(h_2 \,\|\, Q); & E' &\leftarrow \mathsf{XOF}'_3(h_3)\\
h_4 &\leftarrow \mathsf{XOF}_4(h_3 \,\|\, \{v_k\}, \{\bar{v}_k\}, \mathsf{msg}); & E &\leftarrow \mathsf{XOF}'_4(h_4)
\end{aligned}$$

and include the counters $\{\mathsf{ctr}_i\}$ that witness the all-zero prefixes.
3: **(Prover)** Package the PCS/LVCS/DECS openings and short Q-coefficient tails as per the size-optimized layout.
4: **(Verifier)** Recompute all $\{h_i\}$, challenges and checks as in PACS.PIOPOpenAndCheck.

B Extending SPRUCE to anonymous credentials

We target ARC as an anonymous credential schemes for two reasons:

Simplicity ARC uses re-randomization to generate several tokens. In contrast, balance-based schemes like Anonymous Credit Tokens (ACT) [12] require homomorphic encryption to maintain encrypted balances.

Parallel Spends ARC presentations can be generated and spent concurrently. Each ARC presentation is independently verifiable without cryptographic state updates. Balance-based schemes like ACT require sequential spending to update the state inbetween spends.

Our goal is a post-quantum variant of ARC that preserves these properties while providing lattice-based security.

B.1 ARC Credential Structure

The credential binds m_1, a random secret generated by the client, and m_2, a deterministic secret derived from the application context, to the credential. m_1 is used for the tag generation and binds all presentations to the same underlying credential.

m_2 is a deterministic secret derived from the application context. Its concrete usage depends on the application, but it is a public value. It can bind the credential to specific application data (like a place where you can spend your credential). The seperation between m_1 and m_2 allows context-specific credentials without new issuance,

B.2 Credential issuance

B.2.1 Client Credential Request

In classical ARC, the client samples two random scalars $r_1, r_2 \xleftarrow{\$} \mathbb{Z}_p$ and creates Pedersen commitments to hide m_1 and m_2 during issuance.

The post-quantum variant of this issuance using SPRUCE would be to sample (m_1, m_2, r_1, r). The user uses r to compute the Ajtai commitment $c = A \begin{pmatrix} m_1 \\ m_2 \\ r_1 // r \end{pmatrix}$ and a proof that:

1. c is well-formed
2. m_1, m_2, r_1, r are all small lattice elements

Lena: For me, the critical difference here is that we have one commitment instead of two used in the original ARC, where we have one to m_1 and one to m_2. I did think this was a bit weird in the original ARC, but it does have the advantage of context-specific credentials without new issuance. I think if we don't do this we may lose some functionality.

B.2.2 Server Credential Response

To issue a credential in ARC, the server samples a randomization scalar $b \xleftarrow{\$} \mathbb{Z}_p$ and computes the randomized $U = b \cdot G$, which serves as the base for all other credential elements. It will be re-randomized by the client to get an unlinkable credential. Since b is unknown to the client, it prevents credential forgery.

The server computes additional blinding values and issues a blinded MAC which simplifies to $U' = b \cdot (x_0 + x_1 m_1 + x_2 m_2) \cdot G$ after the client removes blinding terms. The final credential is (m_1, U, U', X_1), where $X_1 = x_1 * H$, where x_1 is one of the server's private keys and H is a generator. From this one credential, the client can generate up to n unlinkable presentations.

In the lattice setting, we propose to add a challenge r_2 that the server sends to the user. This randomness r_2 has enough min_entropy to hide the GPV trapdoor. The user sends $t = h_{m_1 \| m_2, r_1 + r_2}(B)$, which is signed by the server after the server verifies the consistency of the earlier messages. This adds another two rounds to the issuance, which is fine since the issuance does not happen too often. Lena: I'm not sure where B is coming from, and how we can prove efficiently that the values are the same as in the commitment. Has this challenge approach been used in lattices before? It seems quite straightforward in any case.

B.3 Credential Presentation

Each presentation re-randomizes the credential elements with fresh randomness and computes a tag $(m_1 + \text{nonce})^{-1} \cdot \mathsf{H}(\text{context}, \text{"Tag"})$. The presentation includes a zero-knowledge proof of:

1. a valid signature possession
2. a correct tag construction
3. a nonce within range $[0, n)$

Since different nonces produce different tags, the same credential with two different nonces creates two unlinkable presentations.

In the lattice setting, F is a PRF. The user chooses a new nonce n and sends $f = F(m_2, n)$ alongside a proof of a hash of a valid GPV signature that contains m_2, and is consistent with n. For the post-quantum construction we of course include a proof that the user has knowledge of a valid preimage of t, which also proves m_1. Only proving signature, nonce and valid signature and proof of nonce in range is not enough, since the proof of the signature is randomized. This means that for the same nonce we can generate several proofs, which is why we evaluate F. Keeping m_2 secret means that we cannot correlate the values. Lena: jetlag, TODO reread later

B.4 Limiting the number of presentations

The presentation limit in ARC is *not* cryptographically enforced by the protocol, but by an application parameter. There are two options to enforce the presentation limit:

Random Nonce Selection The client randomly selects unused nonces from $[0, n)$ and reveals each nonce alongside its presentation. The server then validates that the nonce is in $[0, n)$ and stores observed tags to prevent double-spending. The presentations cannot be linked unless the nonces are reused. However, the client has to keep track of the spent nonces to prevent accidental double-spending.

Range proofs Instead of revealing the nonce, the client adds a zero-knowledge range proof proing that the nonce is in $[0, n)$. using range proofs simplifies the client state, as the client does not have to keep track of spent nonces. However, the additional zero-knowledge proof increases the size of the presentation and verification cost.

It is explicitly the server's responsibility to enforce its rate limit, meaning the server has to keep track of the nonces and proofs to prevent double-spending.

Lena: To construct an ARC-style protocol from a blind signature scheme, we need to add the proofs mentioned in the presentation section and figure out a way to prove both the nonce and the signature. It would be interesting to see which approach is easier (range proofs or random nonces).

B.5 Considerations

Verifiability ARC and other schemes currently uses privately-verifiable MACs (i.e. the issuer must be the verifier, or the issuer has to share the keys with the verifier). I don't really have a good argument for *public* verifiability, however, given the state of VOPRFs, we can argue that we need the public verifiability. We need private verifiability e.g. in the case of PrivacyPass, where we need to ensure the issuer does not segment credentials by using different keys for a subset of parties. Right now, some deployments of PrivacyPass use blind RSA signatures, or ARC with BBS, but both are a lot more expensive than using a VOPRF.

Encoding additional attributes The credential structure in the presentation context should be flexible enough to allow encoding additional attributes beyond rate-limiting (e.g. origin, timestamps, validity period, metadata...) without requiring re-issuance. Especially *late-origin binding*

Binding the presentation limit It's an open question whether the presentation limit n should be cryptographically bound to the credential during issuance. Binding n into the credential would prevent limit equivocation but reduce flexibility. If this is either hard or easy, it would be neat to form an opinion.

B.6 The protocol carsten imagines

To fix our construction in the right way, let's consider the following modified issuance and proof protocol. The message which will be signed during issuance will consist of two parts m_1 and m_2. m_1 is used as actual public credential material that the user can reveal and prove something about. $m_2 \in \mathbf{F}_p^{prf}$ is a uniformly random sequence that we will later use as a key. In addition, we assume that the user uses a nonce when showing a credential. This nonce should be a value from $D \subseteq \mathbb{F}_p^{nonce}$ for a public parameter ℓ. It could just be a small number or some larger value, if we want the nonce to also contain some session context.

Towards constructing the credential scheme, we assume that there exists a secure pseudorandom function

$$PRF : \mathbb{F}_p^{key} \times \mathbb{F}_p^{nonce} \to \mathbb{F}_p^{tag}$$

such that $tag \cdot \log_2(p) \geq 256$. We assume that F_p^{nonce} can be efficiently computed in our ZK proof system. As implementation for PRF we will likely use some conservative parameterization of Poseidon/Hades [10, 11]. While usually running over larger fields, this should still work nicely in our setting. Christian Rechberger suggests that we use Farfalle [**?**] though we might be able to use better parameters.

In the following, we will also use a function *center*. It takes as input a value and reduces it to the interval $[-B, B]$. Basically we want that for any two values $a, b \in [-B, B]$, if either is uniformly random then $center(a + b)$ is uniformly random in $[-B, B]$. This can e.g. be achieved by "wrapping around" the interval boundaries back into the interval, i.e. $center(-B - 1) = B, center(B + 2) = -B + 1$ etc.

Setup Generate a public matrix $\mathbf{A}$ with a trapdoor kept by the Issuer. Also generate a random matrix $\mathbf{A}^c$ (can just be a random string, but not chosen by either issuer or user).

Issuance The user starts with a message $m_1 \in [-B, B]^{k_{m,1}N}$.

1. The user samples $m_2 \in [-B, B]^{(k-k_{m,1})N}$ as well as $r_{U,0} \in [-B, B]^{k_0 N}, r_{U,1} \in [-B, B]^N$ uniformly at random. In addition, it chooses $r \in [-B, B]^{rN}$.

2. Let $\mathbf{A}^c \in R_p^{out \times (k+1+k_0+r)}$ be a uniformly random matrix. Compute $com \leftarrow \mathbf{A}^c \begin{bmatrix} m_1 \\ m_2 \\ r_{U,0} \\ r_{U,1} \\ r \end{bmatrix}$
3. The user sends com to the issuer, together with a ZK proof of plaintext knowledge π_{com} that it knows a preimage of $\mathbf{A}^c$. this proof can likely be skipped, as we have to show the same thing again below.
4. Issuer verifies com using π_{com} and rejects if the proof rejects. Otherwise it chooses $r_{I,0} \in [-B, B]^{k_0 N}, r_{I,1} \in [-B, B]^N$ and sends these to the user.
5. Upon receiving $r_{I,0}, r_{I,1}$ from the Issuer, the User sets

$$\begin{aligned} `r_0 &:= center(r_{U,0} + r_{I,0}) \\ r_1 &:= center(r_{U,1} + r_{I,1}). \end{aligned}$$

   It then sets $m := m_1 \| m_2$, computes

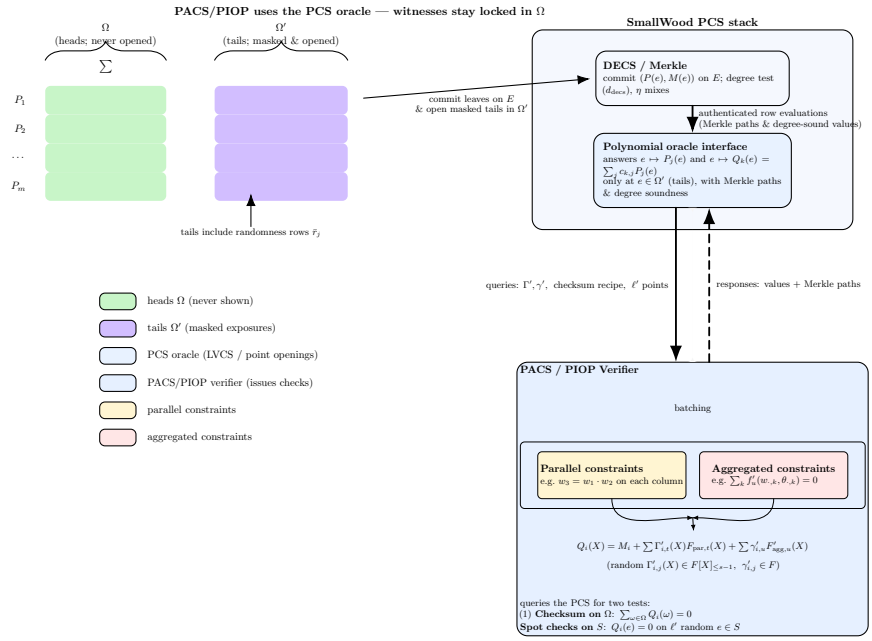
$$t = h_{m,(r_0,r_1)}(B)$$

   and sends t together with a proof π_t to the Issuer. The proof shows that User knows $(m_1, m_2, r_{U,0}, r_{U,1}, r)$ such that
   (a) $com = \mathbf{A}^c(m_1, m_2, r_{U,0}, r_{U,1}, r)^\top$
   (b) $t = h_{m_1 \| m_2, center(r_{U,0}+r_{I,0}), center(r_{U,1}+r_{I,1})}(B)$
   (c) $m_1, m_2, r_{U,0}, r_{U,1}, r$ are from $[-B, B]$

   where $r_{I,0}, r_{I,1}$ are public.
6. Upon verifying π_t, the issuer samples a preimage $\mathbf{u}$ under $\mathbf{A}$ and sends it to the User.
7. The user checks if $\mathbf{t} = \mathbf{Au}$ and that $\mathbf{u}$ is short. If so, then it accepts $\mathbf{u}$ as a valid signature.

Showing To create a showing, the user chooses a previously unknown nonce $nonce \in D$ and computes the following:

1. Set $tag = PRF(m_2, nonce)$.
2. Compute a Non-Interactive ZKPoK π_{show} showing knowledge of $\mathbf{u}, m_1, m_2, r_0, r_1$ such that
   - $\mathbf{Au} = h_{m_1 \| m_2, r_0, r_1}(\mathbf{B})$
   - $tag = PRF(m_2, nonce)$
   - m_1, m_2, r_0, r_1 are over $[-B, B]$
3. Then, send $tag, nonce, \pi_{show}$ to the verifier.
4. The verifier accepts the proof π_{show} if it verifies and if $nonce, tag$ has not been revealed before. could likely also just show that nonce is from some interval and keep it secret as well.



References

[1] M. R. Albrecht, R. Player, and S. Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9(3):169–203, 2015. Preprint: Cryptology ePrint Archive, Report 2015/046. Estimator source: `https://github.com/malb/lattice-estimator`.

[2] D. Chaum. Blind signatures for untraceable payments. pages 199–203, 1982.

[3] A. Dubois, M. Klooß, R. W. F. Lai, and I. K. Y. Woo. Lattice-based proof-friendly signatures from vanishing short integer solutions. pages 452–486, 2025.

[4] A. Dubois, M. Klooß, R. W. F. Lai, and I. K. Y. Woo. Lattice-based proof-friendly signatures from vanishing short integer solutions. Cryptology ePrint Archive, Paper 2025/356, 2025.

[5] T. Espitau, T. T. Q. Nguyen, C. Sun, M. Tibouchi, and A. Wallet. Antrag: Annular NTRU trapdoor generation. Cryptology ePrint Archive, Paper 2023/1335, 2023.

[6] T. Espitau, T. T. Q. Nguyen, C. Sun, M. Tibouchi, and A. Wallet. Antrag: Annular NTRU trapdoor generation - making mitaka as secure as falcon. pages 3–36, 2023.

[7] T. Feneuil and M. Rivain. CAPSS: A framework for SNARK-friendly post-quantum signatures. Cryptology ePrint Archive, Report 2025/061, 2025.

[8] T. Feneuil and M. Rivain. Smallwood: Hash-based polynomial commitments and zero-knowledge arguments for relatively small instances. Cryptology ePrint Archive, Paper 2025/1085, 2025.

[9] C. Gentry, C. Peikert, and V. Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. Cryptology ePrint Archive, Paper 2007/432, 2007.

[10] L. Grassi, D. Khovratovich, C. Rechberger, A. Roy, and M. Schofnegger. Poseidon: A new hash function for zero-knowledge proof systems. pages 519–535, 2021.

[11] L. Grassi, D. Khovratovich, and M. Schofnegger. Poseidon2: A faster version of the poseidon hash function. pages 177–203, 2023.

[12] S. Schlesinger and J. Katz. Anonymous Credit Tokens. Internet-Draft draft-schlesinger-cfrg-act-00, Internet Engineering Task Force, Aug. 2025. Work in Progress.