Compressed verification for post-quantum signatures

Gustavo Banegas, Anaëlle Le Dévéhat, **Benjamin Smith**

22/04/2025 // WRACH // Roscoff

Équipe-Projet GRACE, **Inria** and **École polytechnique**

The post-quantum transition

Quantum computers are coming!

We need to move to *post-quantum systems*, which

- run on classical systems
- resist quantum and classical attacks.

The transition to post-quantum systems is a long-term project.

There are many **choices to be made**, and comparing options is complicated.

Signatures at classical 128-bit security level / NIST Level 1

	Candidate	Paradigm	PK (B)	Sig (B)
Pre-quantum	Ed25519	ECC	32	64
New standards	ML-DSA (Level II)	Structured lattice	1 312	2 420
	Falcon-512	Structured lattice	897	666
	SPHINCS+-128s	Hash	32	7 856
	SPHINCS+-128f	Hash	32	17 088
Round 4/On-ramp	Wave822	Unstructured code	3 677 390	$\leq$ 822
	Squirrels-I	Unstructured lattice	681 780	1 019

To structure or not to structure

Using **structured variants** of the underlying cryptographic problem *(e.g. moving from LWE to Module- or Ring-LWE):*

Pros:

- **Improves** parameter sizes dramatically
- *Can* **improve** runtime performance

Cons:

- **Adds strong hypotheses** to security arguments
- Historically, algebraic structure is an important **new attack vector**

The goal

"Conservative" signatures on unstructured code or lattice problems are **absolutely, spectacularly impractical** for applications where PKs are transmitted with signatures (e.g. TLS certs).

They are merely **totally impractical** for applications where a long-term PK is stored/cached by the verifier to check multiple signatures:

- *Software updates*
- *Authentication (e.g.* `ssh`*)*
- *Root certificates*
- ...

Our goal: to reduce the storage and computational cost for verifying conservative code- and lattice-based signatures.

Bernstein's Rabin–Williams signature verification trick

Rabin–Williams signatures (pre-quantum)

Rabin–Williams signatures: faster than RSA, security equivalent to factoring.

- **Private key**: primes $p \equiv 3 \pmod 8$ and $q \equiv 7 \pmod 8$
- **Public key**: $N = pq$. For 128-bit security: take $\log_2 N \approx 3072$.
- **Signature** on M under N: tuple $\sigma = (e, f, s)$ with $e = \pm 1$, $f \in \{1, 2\}$, $s \in [0, N)$, and
$$efs^2 \equiv \mathsf{Hash}(M) \pmod N. \tag{1}$$
- **Verification**: check (1). Cost: Hashing M, plus one modular squaring mod N.

Bernstein's trick (1997)

Suppose the verifier will verify multiple signatures under the same key N.

First, we lift the verification equation

$$efs^2 \equiv \mathsf{Hash}(M) \pmod{N} \qquad \text{to} \qquad efs^2 = \mathsf{Hash}(M) + kN \text{ in } \mathbb{Z}$$

for some integer k (about the size of N).

Expand signatures to (e, f, s, k).

> *The verifier can efficiently check the latter equation modulo a random 128-bit prime (or several smaller primes) with negligible chance of error. The verifier can amortize the prime-generation cost across any number of signatures by keeping the prime (or prime list) secret and reusing it.*

Signatures: $2\times$ longer. Reduced PK $N \bmod \ell$ (for secret prime ℓ): $24\times$ smaller.

The informal security argument

What's really happening: the verifier checks **homomorphic images** of the signatures under a secret *ring* homomorphism from $\mathbb{Z}$.

The only useful homomorphisms from $\mathbb{Z}$ are $\mathbb{Z} \to \mathbb{Z}/\ell\mathbb{Z}$ for prime (or composite) ℓ

If an adversary can forge for this verifier, i.e. find (e, f, s, k) s.t. $efs^2 \not\equiv \mathsf{Hash}(M) \pmod{N}$ but $efs^2 \equiv \mathsf{Hash}(M) + kN \pmod{\ell}$, then they can find ℓ (and vice versa).

- The adversary is essentially reduced to exhaustive search for ℓ using queries to a verification oracle.
- Each failed forgery attempt (e, f, s, k) reveals *only* that $\ell \nmid (efs^2 - h - kN)$.
- Prime ℓ: each forgery attempt tests $\leq 2 \times 3072/128 = 48$ candidates for ℓ.

Compressed verification

The general approach

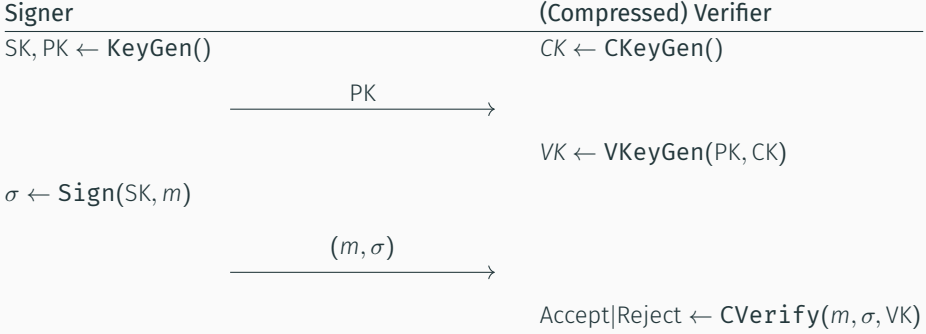
Bernstein's trick was designed to **save time** in RW verification:
working mod a 128-bit ℓ is faster than working mod a 3072-bit N.

We observe that it also **saves space**
because ℓ is much smaller than N

Idea: apply the same idea to post-quantum code- and lattice-based signatures.

Replace secret $(\cdot \bmod \ell) : \mathbb{Z} \to \mathbb{Z}/\ell\mathbb{Z}$ with a secret homomorphism $\phi : \mathcal{M} \to \overline{\mathcal{M}}$
where $\mathcal{M}$ is the cryptosystem's code or lattice, and $\overline{\mathcal{M}}$ is a much smaller module.

Compressed verification: the protocol



CKeyGen: generate secret compression key CK (prime/homomorphism).
VKeyGen: compress public key PK to private verification key VK.
CVerify: verify with private VK in place of public PK.

Compressing Wave

Wave signatures in theory

Public key: a random-looking matrix $\mathbf{R} \in \mathbb{F}_3^{(n-k)\times k}$.

Now $\mathbf{M} := (\mathbf{I}_{n-k}|\mathbf{R})^\top \in \mathbb{F}_3^{n\times(n-k)}$ is a parity-check matrix for a ternary linear code.

Signatures: $\sigma = (\text{salt}, \mathbf{s})$ with $\mathbf{s} \in \mathbb{F}_3^n$.

Verification: Accept $\iff$ CONSTRAINT($\mathbf{s}$) and $\mathbf{sM} = \sum_{i=0}^{n-1} \mathbf{s}_i \mathbf{M}_i = \mathsf{Hash}(\text{salt} \parallel m)$.
Here, Hash maps $\{0,1\}^*$ into $\mathbb{F}_3^{n-k}$.

Use the fact that $\mathbf{M} = (\mathbf{I}_{n-k}|\mathbf{R})^\top$: rewrite verification equation as

$$\text{CONSTRAINT}(\mathbf{s}) \quad \text{and} \quad \sum_{i=0}^{n-1} \mathbf{c}_i \mathbf{M}_i = \mathbf{0}_{n-k} \quad \text{where} \quad \mathbf{c} := \mathbf{s} - (\mathsf{Hash}(\text{salt} \parallel m) \parallel \mathbf{0}_k)\,.$$

Compressed verification for Wave

Wave verification: public key $\mathbf{R} \in \mathbb{F}_3^{(n-k)\times k}$. We write $\mathbf{M} := (\mathbf{I}_{n-k}|\mathbf{R})^\top \in \mathbb{F}_3^{n\times(n-k)}$.

`Verify`: accept $\sigma = (\text{salt}, \mathbf{s})$ iff

$$\textsc{Constraint}(\mathbf{s}) \quad \text{and} \quad \sum_{i=0}^{n-1} \mathbf{c}_i \mathbf{M}_i = \mathbf{0}_{n-k} \quad \text{where} \quad \mathbf{c} := \mathbf{s} - (\mathsf{Hash}(\text{salt} \parallel m) \parallel \mathbf{0}_k)\,.$$

`VKeyGen`: Store $\overline{\mathbf{M}} := (\mathbf{I}_{n-k}|\mathbf{R})^\top \mathbf{P} = \mathbf{M}\mathbf{P} \in \mathbb{F}_3^{n\times d}$ where $\mathbf{P} \xleftarrow{\$} \mathbb{F}_3^{(n-k)\times d}$.
For λ-bit security: can take $d \approx \log_3(2)\lambda$.

`CVerify`: accept $\sigma = (\text{salt}, \mathbf{s})$ iff

$$\textsc{Constraint}(\mathbf{s}) \quad \text{and} \quad \sum_{i=0}^{n-1} \mathbf{c}_i \overline{\mathbf{M}}_i = \mathbf{0}_{n-k} \quad \text{where} \quad \mathbf{c} := \mathbf{s} - (\mathsf{Hash}(\text{salt} \parallel m) \parallel \mathbf{0}_k)\,.$$

Wave signatures: compressed verification

For Wave, compressed verification replaces $(n-k) \times k$ trits with $n \times d$ trits where $d \approx \log_3(2)\lambda \ll k = (n-k)$.

Pros:
- Compression factor 30-40×
- Save 30% of the verification time

Cons:
- Incompatible with signature truncation $\implies$ sigs 2× longer
- Detailed explanation of multiplication by a random matrix makes for an extra boring talk

...So **let's skip the details**.

Compressing Squirrels

Squirrels (Espitau–Niot–Sun–Tibouchi, 2023)

Squirrels is a GPV signature over cocyclic lattices, and most lattices are cocyclic.

- Global system parameter: $\Delta = \prod_{j=1}^{m} \ell_j$, with each ℓ_j a 30-bit prime
  Why? Because all arithmetic mod Δ is done using the CRT.
  ...A bit like RNS, but with multiplicative depth 1.
- Hash maps into $[0, q)^n$ where $q = 4096$
- Public key: a list $(v_1, \ldots, v_n)$ of integers mod Δ *(Convention: $v_n = -1$)*
- Signature: $\mathbf{s} \in \mathbb{Z}^n$ such that

$$\|\mathbf{s}\|_2 \leq \beta \quad \text{and} \quad \sum_{i=1}^{n} c_i v_i \equiv 0 \pmod{\Delta} \quad \text{where} \quad \mathbf{c} = \mathbf{s} - \mathsf{Hash}(r, M)\,.$$

For **Level I**: dimension $n = 1034$; $m = 165 \implies \log_2 \Delta = 5048$; bound $\log_2 \beta \approx 21$.

Squirrels: lifting to $\mathbb{Z}$

We want to choose a secret $\mathbb{Z}$-*module* homomorphism from $(\mathbb{Z}/\Delta\mathbb{Z})^n$,
but there aren't many of these (we are limited to the global primes $\ell_1, \ldots, \ell_m$).

Instead, we lift the verification equation to $\mathbb{Z}$:

$$\sum_{i=1}^{n} c_i v_i = k\Delta \quad \text{for some } k \in \mathbb{Z} \quad \text{where} \quad \mathbf{c} = \mathbf{s} - \mathsf{Hash}(r, M)\,.$$

This k is unique, and must be small: $-\lfloor\sqrt{n\lfloor\beta^2\rfloor}\rfloor \le k \le (n-1)(q-1) + \lfloor\sqrt{n\lfloor\beta^2\rfloor}\rfloor$.

Now, choose a random 128-bit prime modulus π (or a product of smaller primes)
and check the linear equation mod π.

Squirrels: reducing mod π

We need to check

$$\sum_{i=1}^{n} c_i v_i = k\Delta \quad \text{for some } k \in \mathbb{Z} \quad \text{where} \quad \mathbf{c} = \mathbf{s} - \mathsf{Hash}(r, M)\,.$$

- Reducing $\Delta \bmod \pi$: precomputation.
- The integers c_i and k are already smaller than π.
- Reducing $v_i \bmod \pi$: amortised over many verifications.

But this ignores a very nice feature of Squirrels: using the CRT and $\Delta = \prod_{j=1}^{m} \ell_i$ to do all the computation modulo 30-bit primes instead of a 5048-bit Δ.

We want to compute each $v_i \bmod \pi$ on the fly, from the vectors $(v_i \bmod \ell_j)_{j=1}^{m}$, ***without*** *reconstructing the big integer v_i.*

The explicit CRT

Main algorithmic tool: the **Explicit CRT**.

Precompute integers $q_1, \ldots, q_j$ satisfying $q_j(\Delta/\ell_j) \equiv 1 \pmod{\ell_j}$.

Explicit CRT: If $0 \le v_i < \Delta$ and $(v_{i,1}, \ldots, v_{i,s}) = (v_i \bmod \ell_1, \ldots, v_i \bmod \ell_m)$, then

$$v_i = \alpha\Delta - \lfloor \alpha \rfloor \Delta \qquad \text{where} \qquad \alpha = \sum_{j=1}^{m} \frac{v_{i,j} q_j}{\ell_j}. \tag{2}$$

We can compute $v_i \bmod \pi$ by computing $(\alpha\Delta - \lfloor \alpha \rfloor \Delta) \bmod \pi$.

- Precompute $\Delta \bmod \pi$
- Precompute each $Q_j := q_j \Delta/\ell_j \bmod \pi$;
- Computing $\alpha\Delta = \sum_j v_{i,j} Q_j$ is then easy;
- The challenge is to compute $\lfloor \alpha \rfloor \bmod \pi$ without computing α.

Approximating α

Lemma: Let $\alpha_1, \ldots, \alpha_m$ be non-negative real numbers, and set $\alpha := \sum_{j=1}^{n} \alpha_j$. Fix some integer $a \geq \log_2 m + 1$ (fixed-point precision). If we let

$$f := \left\lfloor \frac{m}{2^a} + \frac{1}{2^a} \sum_{j=1}^{m} \lfloor 2^a \alpha_j \rfloor \right\rfloor,$$

then $f = \lfloor \alpha \rfloor$ or $\lfloor \alpha \rfloor + 1$. Further, if $\alpha - \lfloor \alpha \rfloor < 1 - m/2^a$ then f is exactly $\lfloor \alpha \rfloor$.

In our case: easily compute fixed-point approximations to each $\alpha_j = x_j q_j / \ell_j$ with small precision $a \geq \log_2 m + 1$; the ECRT result is correct an error of at most 1.

Squirrels: what about k?

Hence: for each $1 \le i \le n$, given the PK entry $v_i = (v_i \bmod \ell_1, \ldots, v_i \bmod \ell_m)$, we can easily compute

$$\overline{v}_i := v_i + \epsilon_i \Delta \bmod \pi \qquad \text{where} \qquad \epsilon_i \in \{0,1\} \text{ is unknown.}$$

Now, verification is

$$\sum_{i=1}^{n} c_i \cdot \overline{v}_i = k' \Delta \qquad \text{where} \qquad k' := k + \sum_{i=1}^{n} \epsilon_i \text{ is unknown.}$$

But k' is short (about the same size as k)! So we can precompute $\Delta^{-1} \pmod{\pi}$, and verify using only 32-bit arithmetic, checking

$$\|\mathbf{s}\|_2^2 \le \lfloor \beta^2 \rfloor \quad \text{and} \quad \left| \frac{1}{\Delta} \sum_{i=1}^{n} c_i \cdot \overline{v}_i \bmod \pi \right| \le \beta' \quad \text{where} \quad c_i = s_i + h_i \,.$$

(If $\sum_i c_i \overline{v}_i \neq k' \Delta$ with k' short, then $\sum_i c_i \overline{v}_i / \Delta$ is a random, larger element mod π.)

Compressed Squirrels (ouch!)

Compressed Squirrels:

- Public keys: same $((v_i \bmod \ell_j)_{j=1}^{m})_{i=1}^{n}$ ($1034 \times 165 \times 4 = 682440$ bytes)
- Signatures: same (r, s), no need to include k (1019 bytes)
- Verification key: π and $(\overline{v}_i)_{i=1}^{n}$ ($(1034 + 1) \times 16 = 16560$ bytes)

Need extra storage for ECRT coefficients to compress incoming PKs, but these depend only on the global ℓ_i and can be re-used across several public keys.

In practice: don't use a 128-bit prime π; use e.g. 4 or 5 31-bit primes to maintain practical advantages of Squirrels.

Performance

Results

Instance	Reference |σ| (B)	Reference |PK| (B)	Compressed |VK| (B)	Compressed |VK|/|PK|	Verif. time (kCycles) Ref.	Verif. time (kCycles) Comp.	Verif. time (kCycles) Speedup
Squirrels-I	1 019	**681 780**	**20 700**	*3.04%*	280	254	*9.3×*
Wave822	822*	**3 677 390**	**207 968**	*5.65%*	1 101	771	*30×*
Squirrels-III	1 554	**1 629 640**	**49 824**	*3.06%*	551	520	*5.7×*
Wave1249	1 249*	**7 867 598**	**304 192**	*3.86%*	2 330	1 892	*18.8×*
Squirrels-V	2 025	**2 786 580**	**90 598**	*3.25%*	916	898	*1.9×*
Wave1644	1 644*	**13 632 308**	**400 416**	*2.94%*	3 911	3221	*17.4×*

Timings: C reference implementations (& our C code for compressed verification) running on an Intel Core i7-1365U processor.