Squirrels

A post-quantum signature scheme based on plain lattices

Master thesis of Guilhem Niot (09/2023)
PQShield, ENS Lyon, EPFL

Plan

01 Post-quantum cryptography?

02 Lattices and the GPV framework

03 Squirrels — High-level design

04 Evaluation

01 Post-quantum cryptography

Quick introduction

Classical cryptography is broken by quantum computers

Quantum computers

The theory of quantum computing was developed in the ‘80s based on physics quantum theory.

- They allow one to perform many computations at a time, and recover a combined result.
- Not practical yet, but believed to be in a few decades.

Shor’s algorithm

Peter Shor developed in 1994 an algorithm for quantum computers allowing one to factor integers in polynomial time.

- The current cryptographic schemes based on RSA and elliptic curves could be broken by quantum computers in just a few hours.

Classical cryptography is broken by quantum computers

Quantum computers

The theory of quantum computing was developed in the ‘80s based on physics quantum theory.

- They allow one to perform many computations at a time, and recover a combined result.
- Not practical yet, but believed to be in a few decades.

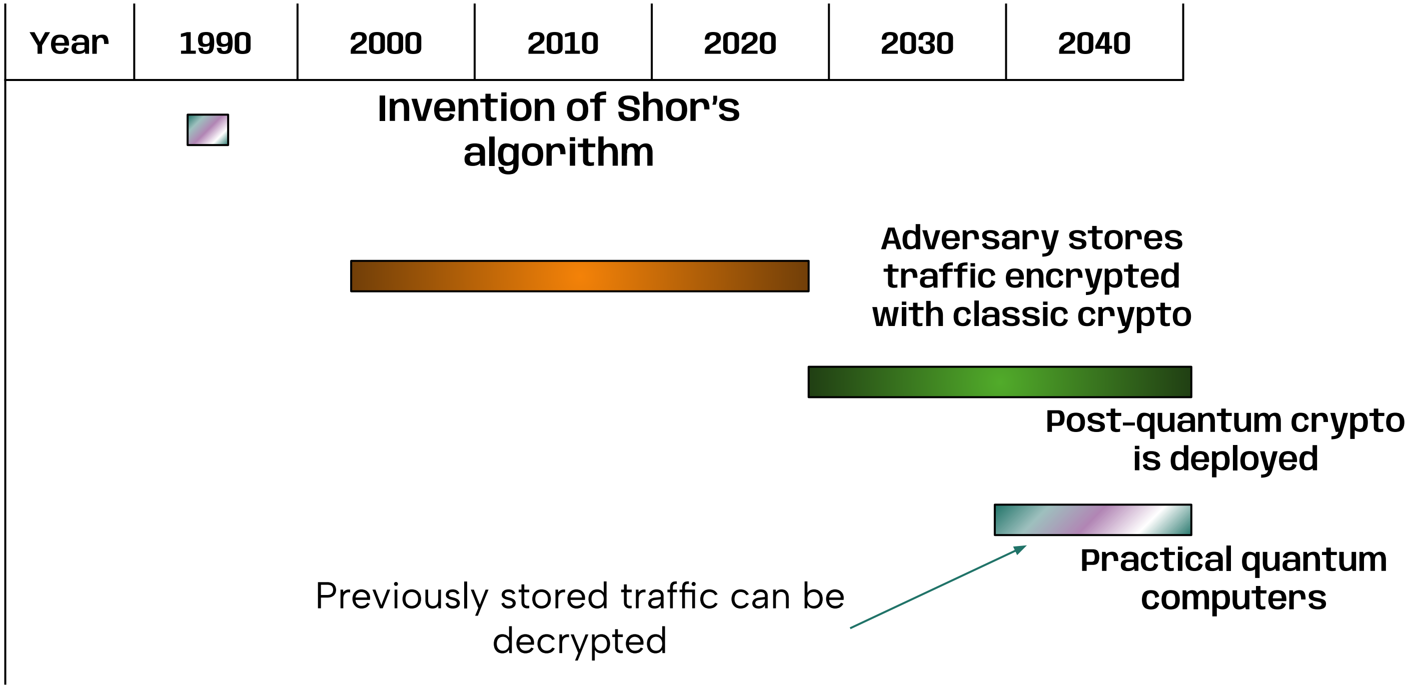
Shor’s algorithm

Peter Shor developed in 1994 an algorithm for quantum computers allowing one to factor integers in polynomial time.

- The current cryptographic schemes based on RSA and elliptic curves could be broken by quantum computers in just a few hours.

→ We need to develop new schemes to protect our data in the future. That’s *post-quantum cryptography*.

The urgency to create and deploy post-quantum secure schemes



Year	1990	2000	2010	2020	2030	2040

Invention of Shor's algorithm

Adversary stores traffic encrypted with classic crypto

Post-quantum crypto is deployed

Practical quantum computers

Previously stored traffic can be decrypted

"Store now, Decrypt later"

Research and standardization efforts

New hardness assumptions

To develop post-quantum asymmetric primitives, new hardness assumptions had to be made.

To cite a few class of problems:

- Lattices
- Error codes
- Isogenies
- Hash-based

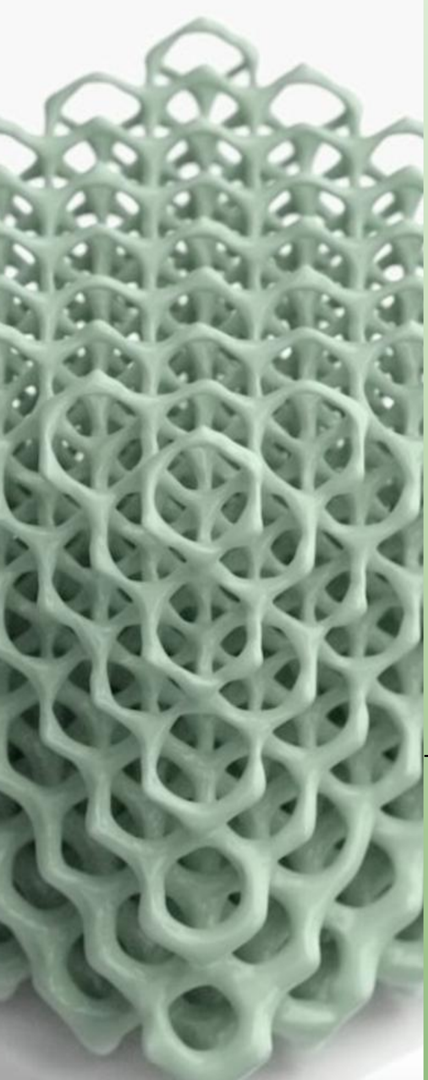
NIST standardization

First call in 2016 for post-quantum proposals of KEMs (*Key Encapsulation Mechanism*) and Signature schemes.

Led to standardization in 2022 of the KEM Kyber (Lattices), and the signatures Dilithium, Falcon (Lattices) and Sphincs$^+$ (Hash).

Second call for signatures in 2022

Little variety in the signatures, and both are based on structured problems. → Call for alternatives.



Lattices and the GPV framework

02

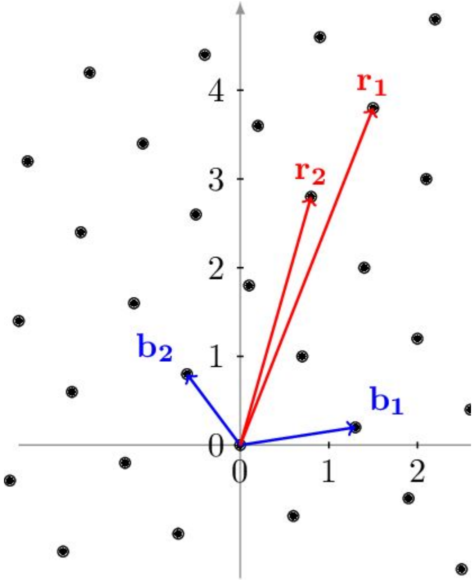
Lattices

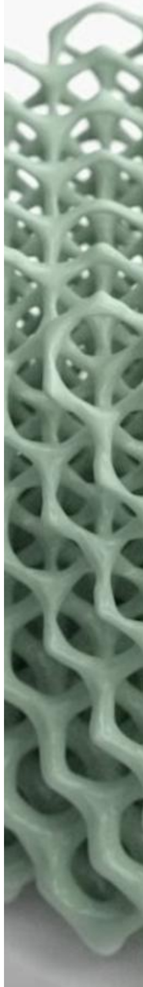
A set of vectors…

Mathematically, a lattice is a set of vectors spanned by a basis:

$$\mathcal{L} = \{\sum x_i b_i \text{ s.t.} x_i \in \mathbb{Z}\}$$

… and given a random basis, it is hard to find a short and quasi orthogonal basis





Lattices

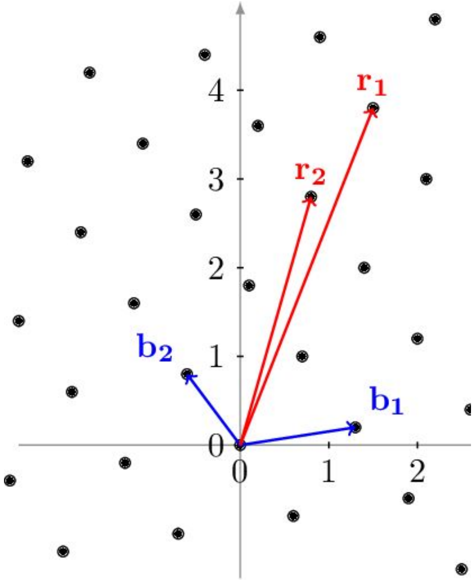
A set of vectors…

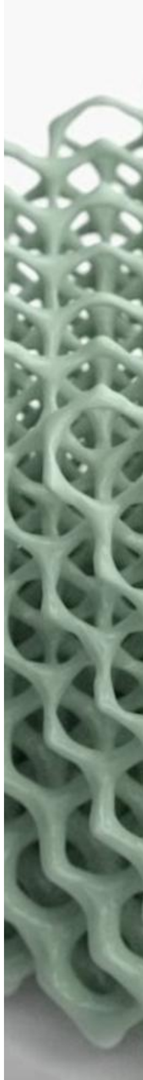
Mathematically, a lattice is a set of vectors spanned by a basis:

$$\mathcal{L} = \{\sum x_i b_i \text{ s.t.} x_i \in \mathbb{Z}\}$$

It is even hard to find one short vector in a random lattice given a random basis.

… and given a random basis, it is hard to find a short and quasi orthogonal basis



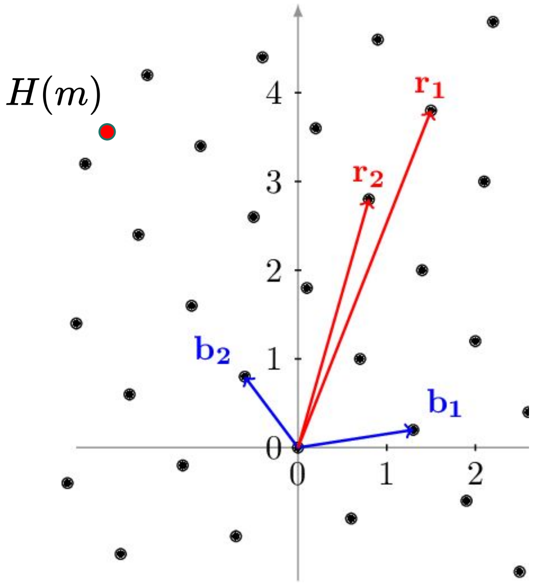


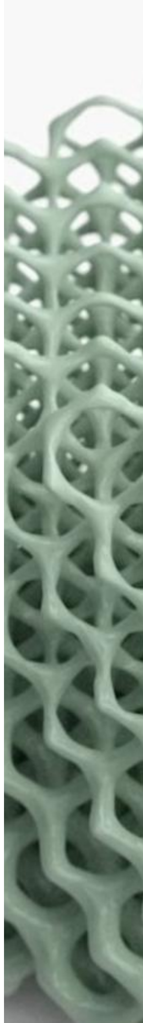
Hash and sign signatures with the GPV framework

It is possible to design signatures from this problem.
One way is to use the GPV framework, as Falcon does.

The signer initially samples a short and quasi orthogonal basis, and publishes a bad basis with long vectors.

1. To sign a message, the signer first hashes it to a point in $\mathbb{Z}^n$
2. Then, he uses the short basis to find a vector close to that hash, and *belonging* to the lattice.
3. The signature is that vector.





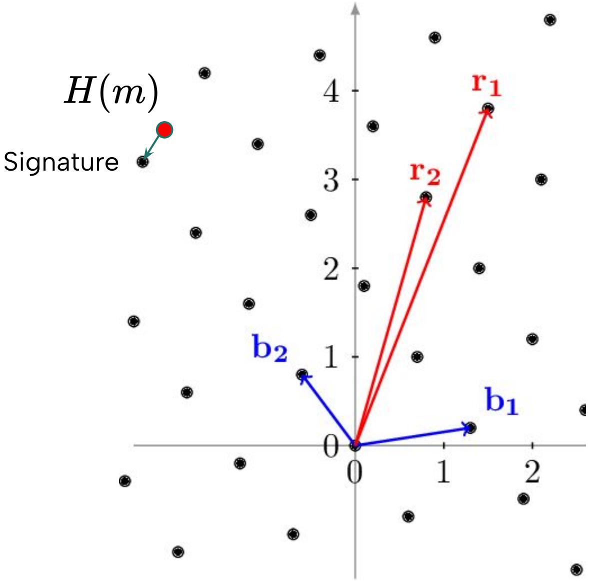
Hash and sign signatures with the GPV framework

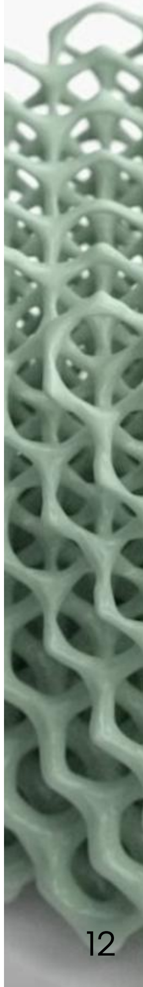
It is possible to design signatures from this problem. One way is to use the GPV framework, as Falcon does.

The signer initially samples a short and quasi orthogonal basis, and publishes a bad basis with long vectors.

1. To sign a message, the signer first hashes it to a point in $\mathbb{Z}^n$
2. Then, he uses the short basis to find a vector close to that hash, and *belonging* to the lattice.
3. The signature is that vector.

The bad basis allows one to verify that this vector is in the lattice, and a bound on the distance with the hash ensures unforgeability.

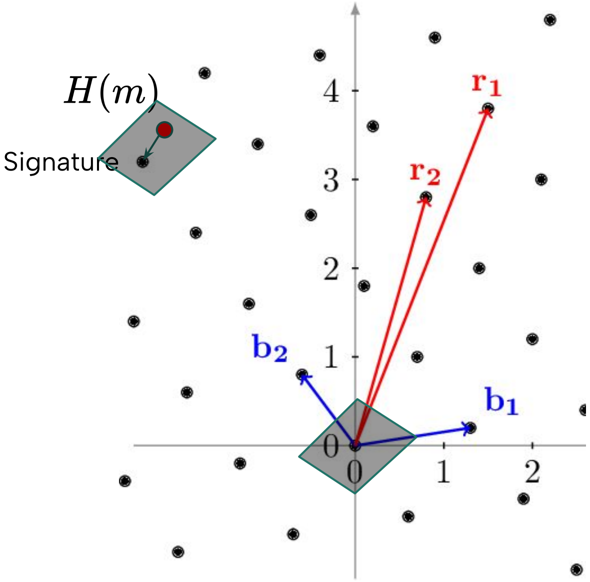


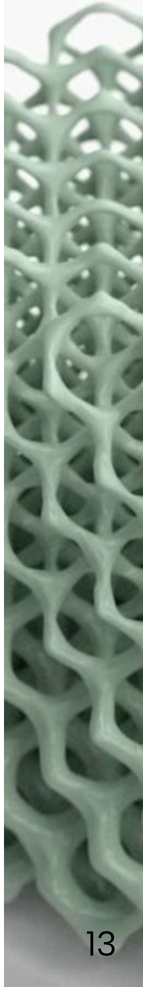


Hash and sign signatures with the GPV framework

But… how to find a vector close to the hash?

Original idea was to use Babai’s Nearest Plane algorithm, which maps the space with the parallelepiped formed by the Gram-Schmidt vectors.



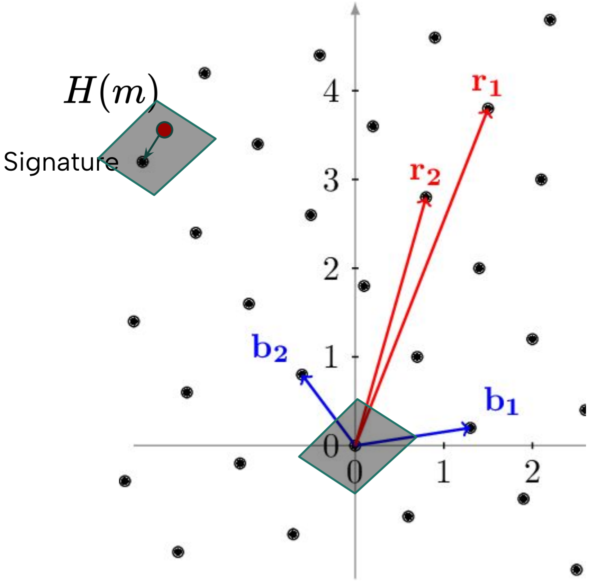


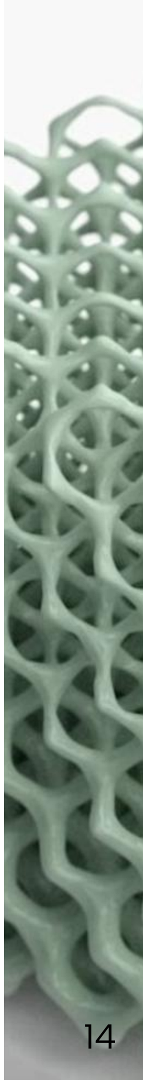
Hash and sign signatures with the GPV framework

But… how to find a vector close to the hash?

Original idea was to use Babai's Nearest Plane algorithm, which maps the space with the parallelepiped formed by the Gram-Schmidt vectors.

… but this leaks the shape of the parallelepiped which is directly related to the good secret basis. It is thus unsecure, and allows an adversary to forge signatures after observing enough of them.

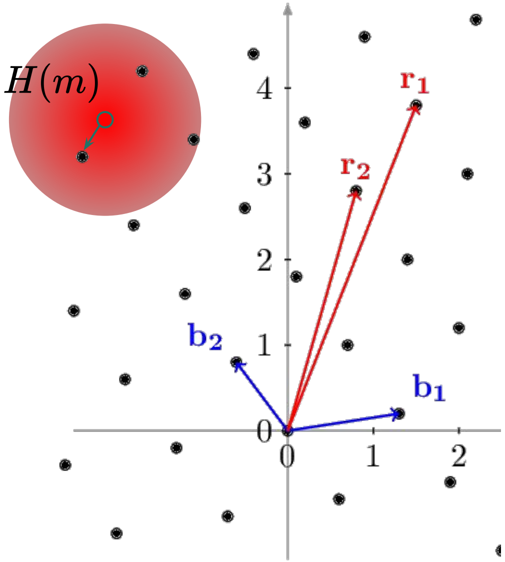


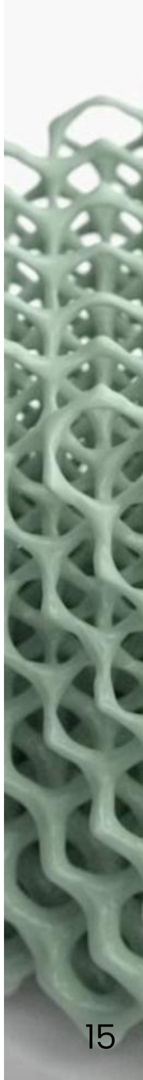


Hash and sign signatures with the GPV framework

But… how to find a vector close to the hash?

Instead, we use Gaussian sampling. Klein proposed a randomized version of Babai Nearest Plane algorithm, returning a point independent of the secret basis.

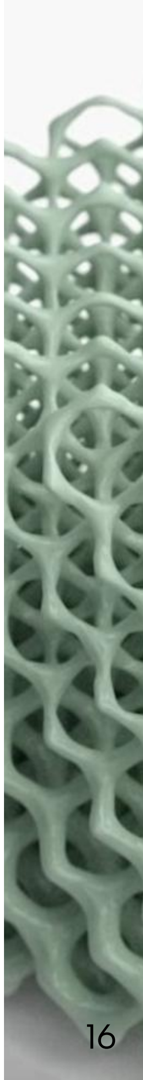




Module and NTRU lattices

NIST lattice finalists optimize the size of public keys and signatures and efficiency by using specific classes of lattices based on polynomials.

They fix a ring $\mathbb{Z}_q[X]/(X^n+1)$, and do polynomial multiplications instead of using matrices.
These can be efficiently implemented using Number Theoretic Transform (NTT).



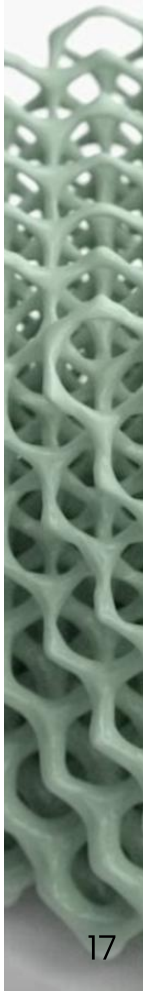
Module and NTRU lattices

NIST lattice finalists optimize the size of public keys and signatures and efficiency by using specific classes of lattices based on polynomials.

They fix a ring $\mathbb{Z}_q[X]/(X^n + 1)$, and do polynomial multiplications instead of using matrices.
These can be efficiently implemented using Number Theoretic Transform (NTT).

Polynomial multiplications can be translated in matrix – vector multiplications, and can thus be translated into lattice problems.

With proper parameters, no attack improvement is known compared to plain random lattices.
But it is a worry, and motivated the additional NIST call for signatures.



03 Squirrels

A digital signature scheme based on plain lattices

The core idea

Why designing Squirrels?

Create a scheme relying on plain lattice assumptions, with high security guarantees.

No structure, average to worst case reductions.

Trade-offs?

The lack of structure implies having a much larger public key.
Signature size remains small.

Optimizing the GPV framework for plain lattices

Squirrels is designed with the goal of optimizing the GPV framework for plain lattices.

Two main observations:

1. The size of signatures is proportional to the size of the largest secret Gram-Schmidt vector. → Our key generation should make it small.

2. We need an efficient way to verify lattice membership, without inverting a very large matrix or solving a linear system.

Optimizing the GPV framework for plain lattices

Squirrels is designed with the goal of optimizing the GPV framework for plain lattices.

Two main observations:

1. The size of signatures is proportional to the size of the largest secret Gram–Schmidt vector.
   → Our key generation should make it small.

   Solution: Sequential sampling of vectors depending on the previous ones, with a target Gram–Schmidt norm.

2. We need an efficient way to verify lattice membership, without inverting a very large matrix or solving a linear system.

Optimizing the GPV framework for plain lattices

Squirrels is designed with the goal of optimizing the GPV framework for plain lattices.

Two main observations:

1. The size of signatures is proportional to the size of the largest secret Gram-Schmidt vector.
   → Our key generation should make it small.

   Solution: Sequential sampling of vectors depending on the previous ones, with a target Gram-Schmidt norm.

2. We need an efficient way to verify lattice membership, without inverting a very large matrix or solving a linear system.

   Solution: Use a subclass of plain lattices with large density, that allows easy membership check.

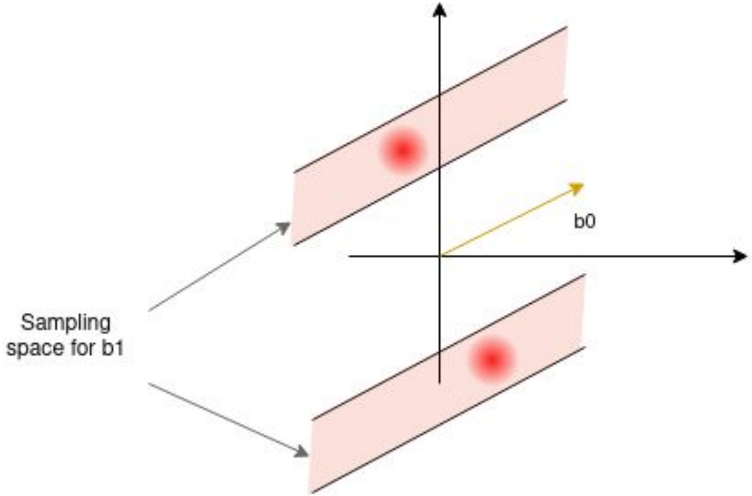
Optimizing the GPV framework for plain lattices

Squirrels’ key generation

Secret vectors are generated sequentially.

1. A continuous candidate is sampled with a constrained norm in the orthogonal of the previous ones.

2. It is then lifted to an integral vector by rounding its coordinates, which introduces a small difference on the Gram-Schmidt norm.



Optimizing the GPV framework for plain lattices

Using co-cyclic lattices

A co-cyclic lattice is a special case of lattices for which there exists $d \in \mathbb{N}, w \in \mathbb{R}^n$ and:

$$\mathcal{L} = \{x \in \mathbb{R}^n \; | < x, w >= 0 \bmod d\}$$

These lattices have a density of more than 80% among integer lattices.

When fixing the determinant appropriately, we can directly compute w from the Hermite Normal Form (HNF) of the lattice with high probability.

$$\mathrm{HNF} = \begin{bmatrix} 1 & \dots & \\ 0 & 1 & v_{\text{check}} \\ 0 & \dots & \det(\mathcal{L}) \end{bmatrix}$$

and

$$w = (v_{\text{check}}, -1)$$

$$d = \det(\mathcal{L})$$

Optimizing the GPV framework for plain lattices

Using co-cyclic lattices

A co-cyclic lattice is a special case of lattices for which there exists $d \in \mathbb{N}, w \in \mathbb{R}^n$ and:

$$\mathcal{L} = \{x \in \mathbb{R}^n \mid < x, w >= 0 \bmod d\}$$

$$w = (v_{\text{check}}, -1)$$

In practice, we would like to *verify* $< x, w >= 0 \bmod \det(\mathcal{L})$ **without big integers** (det ~ 3000 bits).

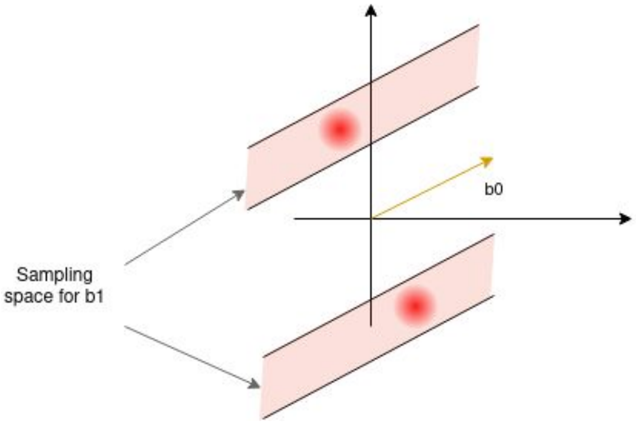
We can fix the determinant to be a product of large primes, and use Chinese Remainder Theorem.

Optimizing the GPV framework for plain lattices

Fixing the determinant

Recall, determinant = product of Gram–Schmidt norms.

1. In the key generation, we thus maintain $\sum_{i=1}^{k} \log ||\tilde{b}_i||$ close to $k/n \cdot \log(\det(\mathcal{L}))$ by varying the sampling bounds.



Optimizing the GPV framework for plain lattices

Fixing the determinant

Recall, determinant = product of Gram–Schmidt norms.

1. In the key generation, we thus maintain $\sum_{i=1}^{k} \log ||\tilde{b}_i||$ close to $k/n \cdot \log(\det(\mathcal{L}))$ by varying the sampling bounds.

2. The last vector is fixed so as to obtain the target determinant.
   Determinant expansion on the last vector gives:

$$\det(\mathcal{L}) = \sum_i (-1)^i b_{\mathrm{last},i} \cdot \mathrm{Minor}_i$$

Method: obtain Bezout coefficients for a few minors, then multiply by target det.

04

Evaluation

Sizes

	PK size (bytes)	Sig size (bytes)
Squirrels I	666000	1019
Falcon I	897	666
Dilithium II	1312	2420

Speed

	Keygen	Sign	Verify
Squirrels I	34s	600/s	13000/s
Falcon I	8ms	6000/s	28000/s
Dilithium II	0.05ms	6900/s	19400/s

Conclusion

Conclusion

- Squirrels offers an alternative to schemes based on structured lattices with stronger security assumptions.
  Submitted to NIST 2022 Call for Additional Digital Signature Schemes.
  - **Small signature size**, between Falcon and Dilithium. **Efficient** to sign and verify.
  - But, large public key and slow to generate.

Conclusion

- Squirrels offers an alternative to schemes based on structured lattices with stronger security assumptions.
  Submitted to NIST 2022 Call for Additional Digital Signature Schemes.
  - **Small signature size**, between Falcon and Dilithium. **Efficient** to sign and verify.
  - But, large public key and slow to generate.

- Practical contributions, with the optimization of the GPV framework
  - Novel usage of co-cyclic lattices, and key generation technique
  - New algorithm to efficiently compute a batch of matrix minors

Thanks!

Questions?