A signature scheme based on Module-LIP

HAWK: Module-LIP makes lattice signatures fast, compact and simple

Léo Ducas, Eamonn W. Postlethwaite, **Ludo N. Pulles**, Wessel van Woerden

7 October 2021

Centrum Wiskunde & Informatica, Amsterdam

NIST PQC Signature finalists

NIST PQC Signature finalists

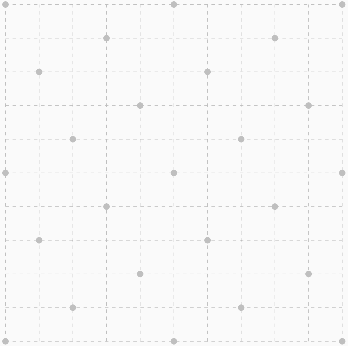
NIST recommends Dilithium for general use.

NIST PQC Signature finalists

NIST recommends Dilithium for general use.

Hash-and-sign

FALCON uses the hash-and-sign design.



Sign(m):

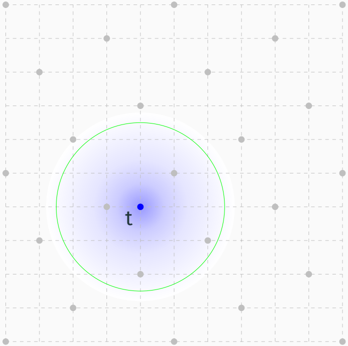
- Hash m to a target $\mathbf{t}$.
- Sample a nearby lattice point $\mathbf{s}$ using a trapdoor basis.

Verify($m, \mathbf{s}$):

- Hash m to a target $\mathbf{t}$.
- Check $\mathbf{s} \in \Lambda$ and $\|\mathbf{s} - \mathbf{t}\|$ small.

Hash-and-sign

FALCON uses the hash-and-sign design.



Sign(m):

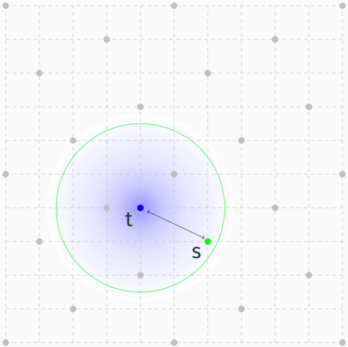
- Hash m to a target $\mathbf{t}$.
- Sample a nearby lattice point $\mathbf{s}$ using a trapdoor basis.

Verify($m, \mathbf{s}$):

- Hash m to a target $\mathbf{t}$.
- Check $\mathbf{s} \in \Lambda$ and $\|\mathbf{s} - \mathbf{t}\|$ small.

Hash-and-sign

FALCON uses the hash-and-sign design.



Sign(m):

- Hash m to a target $\mathbf{t}$.
- Sample a nearby lattice point $\mathbf{s}$ using a trapdoor basis.

Verify($m, \mathbf{s}$):

- Hash m to a target $\mathbf{t}$.
- Check $\mathbf{s} \in \Lambda$ and $\|\mathbf{s} - \mathbf{t}\|$ small.

Falcon

- ✓ Falcon has small keys and signatures.
- × Gaussian sampling is complicated because it requires high-precision floats.
  - Emulating floats is slow on *constrained devices*.
  - Masking is difficult.
  - Fundamental to the class of NTRU lattices.

- Sampling on $\mathbb{Z}^n$ is easy.
- How can we hide $\mathbb{Z}^n$?

Falcon

- ✓ Falcon has small keys and signatures.
- × Gaussian sampling is complicated because it requires high-precision floats.
  - Emulating floats is slow on *constrained devices*.
  - Masking is difficult.
  - Fundamental to the class of NTRU lattices.

- Sampling on $\mathbb{Z}^n$ is easy.
- How can we hide $\mathbb{Z}^n$?

Falcon

✓ Falcon has small keys and signatures.

× Gaussian sampling is complicated because it requires high-precision floats.

- Emulating floats is slow on *constrained devices*.
- Masking is difficult.
- Fundamental to the class of NTRU lattices.

- Sampling on $\mathbb{Z}^n$ is easy.
- How can we hide $\mathbb{Z}^n$?

Falcon

✓ Falcon has small keys and signatures.

× Gaussian sampling is complicated because it requires high-precision floats.

- Emulating floats is slow on *constrained devices*.
- Masking is difficult.
- Fundamental to the class of NTRU lattices.

- Sampling on $\mathbb{Z}^n$ is easy.
- How can we hide $\mathbb{Z}^n$?

Falcon

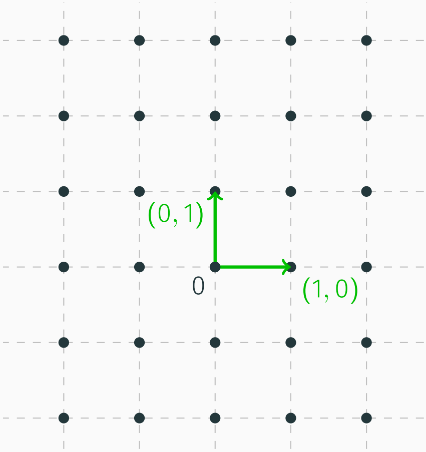
- ✓ Falcon has small keys and signatures.
- × Gaussian sampling is complicated because it requires high-precision floats.
  - Emulating floats is slow on *constrained devices*.
  - Masking is difficult.
  - Fundamental to the class of NTRU lattices.

- Sampling on $\mathbb{Z}^n$ is easy.
- How can we hide $\mathbb{Z}^n$?

Hiding $\mathbb{Z}^n$ with a rotation

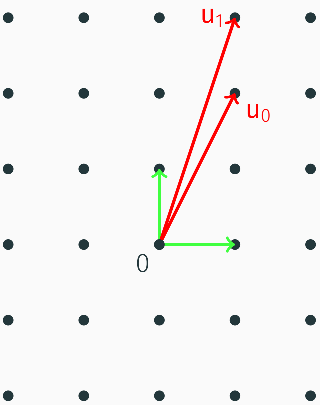
Hiding $\mathbb{Z}^n$ with a rotation

Good basis ~~(Secret key)~~



Λ

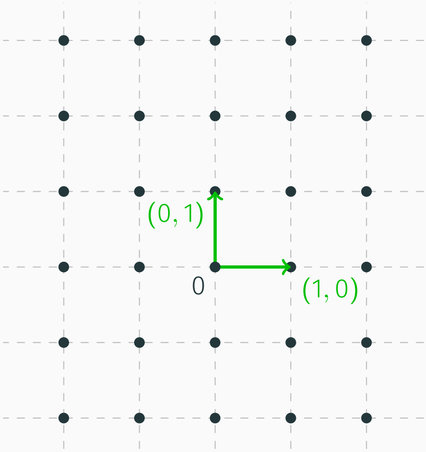
Bad basis (Public key)



Λ

Hiding $\mathbb{Z}^n$ with a rotation

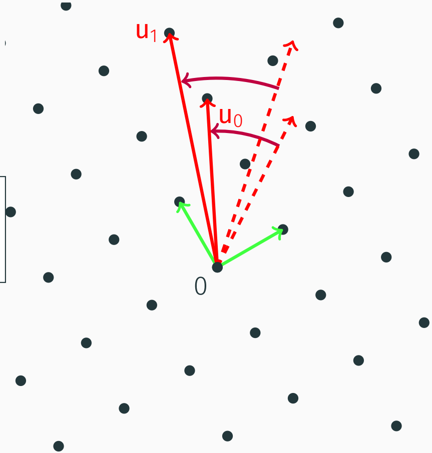
Good basis ~~(Secret key)~~



Λ

$$\xrightarrow[\text{(Secret key)}]{O \in \mathcal{O}_n(\mathbb{R})}$$

Bad basis (Public key)



$O \cdot \Lambda$

Hiding $\mathbb{Z}^n$ with the Lattice Isomorphism Problem (LIP)

Lattice Isomorphism Problem

Given $\mathcal{L}(B) \cong \mathcal{L}(B')$ for $B, B' \in \mathrm{GL}_n(\mathbb{R})$, find $O \in \mathcal{O}_n(\mathbb{R})$ s.t.

$$\mathcal{L}(B') = O \cdot \mathcal{L}(B).$$

- How can we avoid using the floating points in O and B'?
- Make the embedding *implicit*, but keep the geometry.

Lattice Isomorphism Problem (bases)

Given $\mathcal{L}(B) \cong \mathcal{L}(B')$ for $B, B' \in \mathrm{GL}_n(\mathbb{R})$, find $O \in \mathcal{O}_n(\mathbb{R})$ and $U \in \mathrm{GL}_n(\mathbb{Z})$ s.t.

$$B' = O \cdot B \cdot U.$$

- How can we avoid using the floating points in O and B′?
- Make the embedding *implicit*, but keep the geometry.

Hiding $\mathbb{Z}^n$ with the Lattice Isomorphism Problem (LIP)

Lattice Isomorphism Problem (bases)

Given $\mathcal{L}(B) \cong \mathcal{L}(B')$ for $B, B' \in \mathrm{GL}_n(\mathbb{R})$, find $O \in \mathcal{O}_n(\mathbb{R})$ and $U \in \mathrm{GL}_n(\mathbb{Z})$ s.t.

$$B' = O \cdot B \cdot U.$$

- How can we avoid using the floating points in **O** and **B′**?
- Make the embedding *implicit*, but keep the geometry.

Hiding $\mathbb{Z}^n$ with the Lattice Isomorphism Problem (LIP)

Lattice Isomorphism Problem (bases)

Given $\mathcal{L}(B) \cong \mathcal{L}(B')$ for $B, B' \in \mathrm{GL}_n(\mathbb{R})$, find $O \in \mathcal{O}_n(\mathbb{R})$ and $U \in \mathrm{GL}_n(\mathbb{Z})$ s.t.

$$B' = O \cdot B \cdot U.$$

- How can we avoid using the floating points in **O** and **B′**?
- Make the embedding *implicit*, but keep the geometry.

Keeping the geometry & making the embedding implicit

- The Gram matrix is invariant under rotations:

$$(\mathsf{O} \cdot \mathsf{B})^\top \cdot (\mathsf{O} \cdot \mathsf{B}) = \mathsf{B}^\top \cdot \underbrace{\mathsf{O}^\top \cdot \mathsf{O}}_{\mathbb{I}_n} \cdot \mathsf{B} = \mathsf{B}^\top \cdot \mathsf{B} = \mathsf{Q}.$$

- Make the Gram matrix $\mathbf{Q}' = \mathbf{B}'^\top \cdot \mathbf{B}' = \mathbf{U}^\top \mathbf{Q} \mathbf{U}$ public but keep $\mathbf{U}$ secret.
- Cholesky decomposition on $\mathbf{Q}'$ gives the explicit embedding: basis $\mathbf{B}'$.
- [DvW22][1] shows a generic way to go from a sampleable lattice to a signature scheme that reduces to a variant of LIP.
- Hence we can make a signature scheme on $\mathbb{Z}^n$ with sk $= \mathbf{U}$ and pk $= \mathbf{U}^\top \cdot \mathbf{U}$.
- But how do we make this competitive?

[1] Léo Ducas, Wessel van Woerden: On the Lattice Isomorphism Problem, Quadratic Forms, Remarkable Lattices, and Cryptography. In: EUROCRYPT 2022

Keeping the geometry & making the embedding implicit

- The Gram matrix is invariant under rotations:

$$(\mathbf{O} \cdot \mathbf{B})^\top \cdot (\mathbf{O} \cdot \mathbf{B}) = \mathbf{B}^\top \cdot \underbrace{\mathbf{O}^\top \cdot \mathbf{O}}_{\mathbb{I}_n} \cdot \mathbf{B} = \mathbf{B}^\top \cdot \mathbf{B} = \mathbf{Q}.$$

- Make the Gram matrix $\mathbf{Q}' = \mathbf{B}'^\top \cdot \mathbf{B}' = \mathbf{U}^\top \mathbf{Q} \mathbf{U}$ public but keep $\mathbf{U}$ secret.
- Cholesky decomposition on $\mathbf{Q}'$ gives the explicit embedding: basis $\mathbf{B}'$.
- [DvW22][1] shows a generic way to go from a sampleable lattice to a signature scheme that reduces to a variant of LIP.
- Hence we can make a signature scheme on $\mathbb{Z}^n$ with $\mathsf{sk} = \mathbf{U}$ and $\mathsf{pk} = \mathbf{U}^\top \cdot \mathbf{U}$.
- But how do we make this competitive?

[1] Léo Ducas, Wessel van Woerden: On the Lattice Isomorphism Problem, Quadratic Forms, Remarkable Lattices, and Cryptography. In: EUROCRYPT 2022

Keeping the geometry & making the embedding implicit

- The Gram matrix is invariant under rotations:

$$(\mathbf{O} \cdot \mathbf{B})^\top \cdot (\mathbf{O} \cdot \mathbf{B}) = \mathbf{B}^\top \cdot \underbrace{\mathbf{O}^\top \cdot \mathbf{O}}_{\mathbb{I}_n} \cdot \mathbf{B} = \mathbf{B}^\top \cdot \mathbf{B} = \mathbf{Q}.$$

- Make the Gram matrix $\mathbf{Q}' = \mathbf{B}'^\top \cdot \mathbf{B}' = \mathbf{U}^\top \mathbf{Q} \mathbf{U}$ public but keep $\mathbf{U}$ secret.
- Cholesky decomposition on $\mathbf{Q}'$ gives the explicit embedding: basis $\mathbf{B}'$.
- [DvW22][1] shows a generic way to go from a sampleable lattice to a signature scheme that reduces to a variant of LIP.
- Hence we can make a signature scheme on $\mathbb{Z}^n$ with $\mathrm{sk} = \mathbf{U}$ and $\mathrm{pk} = \mathbf{U}^\top \cdot \mathbf{U}$.
- But how do we make this competitive?

[1] Léo Ducas, Wessel van Woerden: On the Lattice Isomorphism Problem, Quadratic Forms, Remarkable Lattices, and Cryptography. In: EUROCRYPT 2022

Keeping the geometry & making the embedding implicit

- The Gram matrix is invariant under rotations:

$$(\mathbf{O} \cdot \mathbf{B})^\top \cdot (\mathbf{O} \cdot \mathbf{B}) = \mathbf{B}^\top \cdot \underbrace{\mathbf{O}^\top \cdot \mathbf{O}}_{\mathbb{I}_n} \cdot \mathbf{B} = \mathbf{B}^\top \cdot \mathbf{B} = \mathbf{Q}.$$

- Make the Gram matrix $\mathbf{Q}' = \mathbf{B}'^\top \cdot \mathbf{B}' = \mathbf{U}^\top \mathbf{Q} \mathbf{U}$ public but keep $\mathbf{U}$ secret.
- Cholesky decomposition on $\mathbf{Q}'$ gives the explicit embedding: basis $\mathbf{B}'$.
- [DvW22][1] shows a generic way to go from a sampleable lattice to a signature scheme that reduces to a variant of LIP.
- Hence we can make a signature scheme on $\mathbb{Z}^n$ with $\mathrm{sk} = \mathbf{U}$ and $\mathrm{pk} = \mathbf{U}^\top \cdot \mathbf{U}$.
- But how do we make this competitive?

[1] Léo Ducas, Wessel van Woerden: On the Lattice Isomorphism Problem, Quadratic Forms, Remarkable Lattices, and Cryptography. In: EUROCRYPT 2022

Keeping the geometry & making the embedding implicit

- The Gram matrix is invariant under rotations:

$$(\mathbf{O} \cdot \mathbf{B})^\top \cdot (\mathbf{O} \cdot \mathbf{B}) = \mathbf{B}^\top \cdot \underbrace{\mathbf{O}^\top \cdot \mathbf{O}}_{\mathbb{I}_n} \cdot \mathbf{B} = \mathbf{B}^\top \cdot \mathbf{B} = \mathbf{Q}.$$

- Make the Gram matrix $\mathbf{Q}' = \mathbf{B}'^\top \cdot \mathbf{B}' = \mathbf{U}^\top \mathbf{Q} \mathbf{U}$ public but keep $\mathbf{U}$ secret.
- Cholesky decomposition on $\mathbf{Q}'$ gives the explicit embedding: basis $\mathbf{B}'$.
- [DvW22][1] shows a generic way to go from a sampleable lattice to a signature scheme that reduces to a variant of LIP.
- Hence we can make a signature scheme on $\mathbb{Z}^n$ with $\mathsf{sk} = \mathbf{U}$ and $\mathsf{pk} = \mathbf{U}^\top \cdot \mathbf{U}$.
- But how do we make this competitive?

[1] Léo Ducas, Wessel van Woerden: On the Lattice Isomorphism Problem, Quadratic Forms, Remarkable Lattices, and Cryptography. In: EUROCRYPT 2022

Keeping the geometry & making the embedding implicit

- The Gram matrix is invariant under rotations:

$$(\mathbf{O} \cdot \mathbf{B})^\top \cdot (\mathbf{O} \cdot \mathbf{B}) = \mathbf{B}^\top \cdot \underbrace{\mathbf{O}^\top \cdot \mathbf{O}}_{\mathbb{I}_n} \cdot \mathbf{B} = \mathbf{B}^\top \cdot \mathbf{B} = \mathbf{Q}.$$

- Make the Gram matrix $\mathbf{Q}' = \mathbf{B}'^\top \cdot \mathbf{B}' = \mathbf{U}^\top \mathbf{Q} \mathbf{U}$ public but keep $\mathbf{U}$ secret.
- Cholesky decomposition on $\mathbf{Q}'$ gives the explicit embedding: basis $\mathbf{B}'$.
- [DvW22][1] shows a generic way to go from a sampleable lattice to a signature scheme that reduces to a variant of LIP.
- Hence we can make a signature scheme on $\mathbb{Z}^n$ with $\mathrm{sk} = \mathbf{U}$ and $\mathrm{pk} = \mathbf{U}^\top \cdot \mathbf{U}$.
- But how do we make this competitive?

[1] Léo Ducas, Wessel van Woerden: On the Lattice Isomorphism Problem, Quadratic Forms, Remarkable Lattices, and Cryptography. In: EUROCRYPT 2022

Making the [DvW22] signature scheme of $\mathbb{Z}^n$ competitive

How do we make this competitive?

1. We add extra structure.
2. We compress keys and signatures.
3. We hash to targets in $\frac{1}{2}\mathbb{Z}^n$ so we can use precomputed distribution tables for sampling from $\mathbb{Z}$ and $\mathbb{Z} + \frac{1}{2}$.

How do we make this competitive?

1. We add extra structure.
2. We compress keys and signatures.
3. We hash to targets in $\frac{1}{2}\mathbb{Z}^n$ so we can use precomputed distribution tables for sampling from $\mathbb{Z}$ and $\mathbb{Z} + \frac{1}{2}$.

How do we make this competitive?

1. We add extra structure.
2. We compress keys and signatures.
3. We hash to targets in $\frac{1}{2}\mathbb{Z}^n$ so we can use precomputed distribution tables for sampling from $\mathbb{Z}$ and $\mathbb{Z} + \frac{1}{2}$.

Adding extra structure

Adding extra structure

- Replace $\mathbb{Z}^{2n}$ by $R \oplus R$, where $R = \mathbb{Z}[\zeta_{2n}] = \mathbb{Z}[X]/(X^n + 1) \cong \mathbb{Z}^n$ for n a power of 2.
- The unimodular transformation is secret:

$$\mathrm{sk} = \mathbf{U} = \begin{bmatrix} \mathbf{u}_0 & \mathbf{u}_1 \end{bmatrix} = \left[\begin{array}{c|c} f & F \\ g & G \end{array}\right],$$

with $f, g \in R$ sampled from a (narrow) discrete Gaussian.
- Then, F, G are computed s.t. $fG - gF = 1$ (NTRU equation).
- This is basically Falcon's KeyGen with $q = 1$.
- The geometry is public:

$$\mathrm{pk} = \mathbf{Q} = \mathbf{U}^* \cdot \mathbf{U} = \begin{bmatrix} \mathbf{u}_0^*\mathbf{u}_0 & \mathbf{u}_0^*\mathbf{u}_1 \\ \mathbf{u}_1^*\mathbf{u}_0 & \mathbf{u}_1^*\mathbf{u}_1 \end{bmatrix}.$$

Adding extra structure

- Replace $\mathbb{Z}^{2n}$ by $R \oplus R$, where $R = \mathbb{Z}[\zeta_{2n}] = \mathbb{Z}[X]/(X^n + 1) \cong \mathbb{Z}^n$ for n a power of 2.
- The unimodular transformation is secret:

$$\mathrm{sk} = \mathbf{U} = \begin{bmatrix} \mathbf{u}_0 & \mathbf{u}_1 \end{bmatrix} = \left[\begin{array}{c|c} f & F \\ g & G \end{array}\right],$$

  with $f, g \in R$ sampled from a (narrow) discrete Gaussian.
- Then, F, G are computed s.t. $fG - gF = 1$ (NTRU equation).
- This is basically FALCON's KeyGen with $q = 1$.
- The geometry is public:

$$\mathrm{pk} = \mathbf{Q} = \mathbf{U}^* \cdot \mathbf{U} = \begin{bmatrix} \mathbf{u}_0^* \mathbf{u}_0 & \mathbf{u}_0^* \mathbf{u}_1 \\ \mathbf{u}_1^* \mathbf{u}_0 & \mathbf{u}_1^* \mathbf{u}_1 \end{bmatrix}.$$

Adding extra structure

- Replace $\mathbb{Z}^{2n}$ by $R \oplus R$, where $R = \mathbb{Z}[\zeta_{2n}] = \mathbb{Z}[X]/(X^n + 1) \cong \mathbb{Z}^n$ for n a power of 2.
- The unimodular transformation is secret:

$$\mathrm{sk} = \mathbf{U} = \begin{bmatrix} \mathbf{u}_0 & \mathbf{u}_1 \end{bmatrix} = \left[\begin{array}{c|c} f & F \\ g & G \end{array}\right],$$

  with $f, g \in R$ sampled from a (narrow) discrete Gaussian.
- Then, F, G are computed s.t. $fG - gF = 1 \quad$ (NTRU equation).
- This is basically FALCON's KeyGen with $q = 1$.
- The geometry is public:

$$\mathrm{pk} = \mathbf{Q} = \mathbf{U}^* \cdot \mathbf{U} = \begin{bmatrix} \mathbf{u}_0^* \mathbf{u}_0 & \mathbf{u}_0^* \mathbf{u}_1 \\ \mathbf{u}_1^* \mathbf{u}_0 & \mathbf{u}_1^* \mathbf{u}_1 \end{bmatrix}.$$

Adding extra structure

- Replace $\mathbb{Z}^{2n}$ by $R \oplus R$, where $R = \mathbb{Z}[\zeta_{2n}] = \mathbb{Z}[X]/(X^n + 1) \cong \mathbb{Z}^n$ for n a power of 2.
- The unimodular transformation is secret:

$$\mathrm{sk} = \mathbf{U} = \begin{bmatrix} \mathbf{u}_0 & \mathbf{u}_1 \end{bmatrix} = \left[\begin{array}{c|c} f & F \\ g & G \end{array}\right],$$

with $f, g \in R$ sampled from a (narrow) discrete Gaussian.

- Then, F, G are computed s.t. $fG - gF = 1$ (NTRU equation).
- This is basically FALCON's KeyGen with $q = 1$.
- The geometry is public:

$$\mathrm{pk} = \mathbf{Q} = \mathbf{U}^* \cdot \mathbf{U} = \begin{bmatrix} \mathbf{u}_0^*\mathbf{u}_0 & \mathbf{u}_0^*\mathbf{u}_1 \\ \mathbf{u}_1^*\mathbf{u}_0 & \mathbf{u}_1^*\mathbf{u}_1 \end{bmatrix}.$$

Adding extra structure

- Replace $\mathbb{Z}^{2n}$ by $R \oplus R$, where $R = \mathbb{Z}[\zeta_{2n}] = \mathbb{Z}[X]/(X^n + 1) \cong \mathbb{Z}^n$ for n a power of 2.
- The unimodular transformation is secret:

$$\mathrm{sk} = \mathbf{U} = \begin{bmatrix} \mathbf{u}_0 & \mathbf{u}_1 \end{bmatrix} = \left[\begin{array}{c|c} f & F \\ & \\ g & G \end{array}\right],$$

  with $f, g \in R$ sampled from a (narrow) discrete Gaussian.
- Then, F, G are computed s.t. $fG - gF = 1 \quad$ (NTRU equation).
- This is basically FALCON's KeyGen with $q = 1$.
- The geometry is public:

$$\mathrm{pk} = \mathbf{Q} = \mathbf{U}^* \cdot \mathbf{U} = \begin{bmatrix} \mathbf{u}_0^* \mathbf{u}_0 & \mathbf{u}_0^* \mathbf{u}_1 \\ \mathbf{u}_1^* \mathbf{u}_0 & \mathbf{u}_1^* \mathbf{u}_1 \end{bmatrix}.$$

Compression

Compressing keys and signatures

- Encode the secret key like Falcon, dropping $G = (1 + gF)/f$.
- From the public key $\mathbf{Q} = \begin{bmatrix} \mathbf{Q}_{00} & \mathbf{Q}_{01} \\ \mathbf{Q}_{10} & \mathbf{Q}_{11} \end{bmatrix} = \begin{bmatrix} \mathbf{u}_0^* \mathbf{u}_0 & \mathbf{u}_0^* \mathbf{u}_1 \\ \mathbf{u}_1^* \mathbf{u}_0 & \mathbf{u}_1^* \mathbf{u}_1 \end{bmatrix}$ only store $\mathbf{Q}_{00}$ and $\mathbf{Q}_{01}$ as we can recover:

$$\mathbf{Q}_{10} = \mathbf{Q}_{01}^* \quad \text{and} \quad \mathbf{Q}_{11} = \frac{1 + \mathbf{Q}_{10}\mathbf{Q}_{01}}{\mathbf{Q}_{00}}.$$

- We can drop s_0 from a signature $\mathbf{s} = (s_0, s_1)$, and recover s_0 (almost always) during verification with a ring generalization of Babai's round-off algorithm.

Compressing keys and signatures

- Encode the secret key like FALCON, dropping $G = (1 + gF)/f$.
- From the public key $\mathbf{Q} = \begin{bmatrix} \mathbf{Q}_{00} & \mathbf{Q}_{01} \\ \mathbf{Q}_{10} & \mathbf{Q}_{11} \end{bmatrix} = \begin{bmatrix} \mathbf{u}_0^* \mathbf{u}_0 & \mathbf{u}_0^* \mathbf{u}_1 \\ \mathbf{u}_1^* \mathbf{u}_0 & \mathbf{u}_1^* \mathbf{u}_1 \end{bmatrix}$ only store $\mathbf{Q}_{00}$ and $\mathbf{Q}_{01}$ as we can recover:

$$\mathbf{Q}_{10} = \mathbf{Q}_{01}^* \quad \text{and} \quad \mathbf{Q}_{11} = \frac{1 + \mathbf{Q}_{10}\mathbf{Q}_{01}}{\mathbf{Q}_{00}}.$$

- We can drop s_0 from a signature $\mathbf{s} = (s_0, s_1)$, and recover s_0 (almost always) during verification with a ring generalization of Babai's round-off algorithm.

Compressing keys and signatures

- Encode the secret key like FALCON, dropping $G = (1 + gF)/f$.
- From the public key $\mathbf{Q} = \begin{bmatrix} \mathbf{Q}_{00} & \mathbf{Q}_{01} \\ \mathbf{Q}_{10} & \mathbf{Q}_{11} \end{bmatrix} = \begin{bmatrix} \mathbf{u}_0^* \mathbf{u}_0 & \mathbf{u}_0^* \mathbf{u}_1 \\ \mathbf{u}_1^* \mathbf{u}_0 & \mathbf{u}_1^* \mathbf{u}_1 \end{bmatrix}$ only store $\mathbf{Q}_{00}$ and $\mathbf{Q}_{01}$ as we can recover:
$$\mathbf{Q}_{10} = \mathbf{Q}_{01}^* \quad \text{and} \quad \mathbf{Q}_{11} = \frac{1 + \mathbf{Q}_{10}\mathbf{Q}_{01}}{\mathbf{Q}_{00}}.$$
- We can drop s_0 from a signature $\mathbf{s} = (s_0, s_1)$, and recover s_0 (almost always) during verification with a ring generalization of Babai's round-off algorithm.

Dropping s_0

Sign algorithm $\mathsf{Sign}(\mathbf{U}, m)$:

1. Hash to $\mathbf{h} = H(m) \in \{0,1\}^{2n} \quad `\subseteq' R^2$.
2. Sample $\mathbf{x} \in R^2$ close to $\frac{1}{2}\mathbf{U} \cdot \mathbf{h}$.
3. Return $\mathbf{s} = \mathbf{U}^{-1} \cdot \mathbf{x}$.

– Note: $\mathbf{Us}$ is close to $\frac{1}{2}\mathbf{Uh}$, so $s_0\mathbf{u}_0$ is close to $\frac{1}{2}\mathbf{Uh} - s_1\mathbf{u}_1$.

$\Longrightarrow$ Use Babai's round-off (or nearest plane) algorithm:

$$s_0' = \left\lceil \frac{h_0}{2} + \frac{\mathsf{Q}_{01}}{\mathsf{Q}_{00}} \left(\frac{h_1}{2} - s_1 \right) \right\rfloor.$$

– Reject key pairs for which Q_{00} is "too small" $\Longrightarrow$ recovery works "always" ($\geq 1 - 2^{-100}$).

Dropping s_0

Sign algorithm $\mathrm{Sign}(\mathbf{U}, m)$:

1. Hash to $\mathbf{h} = H(m) \in \{0,1\}^{2n} \quad `\subseteq' R^2$.
2. Sample $\mathbf{x} \in R^2$ close to $\frac{1}{2}\mathbf{U} \cdot \mathbf{h}$.
3. Return $\mathbf{s} = \mathbf{U}^{-1} \cdot \mathbf{x}$.

– Note: $\mathbf{U}\mathbf{s}$ is close to $\frac{1}{2}\mathbf{U}\mathbf{h}$, so $s_0\mathbf{u}_0$ is close to $\frac{1}{2}\mathbf{U}\mathbf{h} - s_1\mathbf{u}_1$.

$\Longrightarrow$ Use Babai's round-off (or nearest plane) algorithm:

$$s_0' = \left\lceil \frac{h_0}{2} + \frac{Q_{01}}{Q_{00}} \left(\frac{h_1}{2} - s_1 \right) \right\rfloor .$$

– Reject key pairs for which Q_{00} is "too small" $\Longrightarrow$ recovery works "always" ($\geq 1 - 2^{-100}$).

Dropping s_0

Sign algorithm $\mathrm{Sign}(\mathbf{U}, m)$:

1. Hash to $\mathbf{h} = H(m) \in \{0,1\}^{2n}$ '$\subseteq$' R^2.
2. Sample $\mathbf{x} \in R^2$ close to $\frac{1}{2}\mathbf{U} \cdot \mathbf{h}$.
3. Return $\mathbf{s} = \mathbf{U}^{-1} \cdot \mathbf{x}$.

– Note: $\mathbf{U}\mathbf{s}$ is close to $\frac{1}{2}\mathbf{U}\mathbf{h}$, so $s_0\mathbf{u}_0$ is close to $\frac{1}{2}\mathbf{U}\mathbf{h} - s_1\mathbf{u}_1$.

$\Longrightarrow$ Use Babai's round-off (or nearest plane) algorithm:

$$s_0' = \left\lceil \frac{h_0}{2} + \frac{\mathrm{Q}_{01}}{\mathrm{Q}_{00}} \left(\frac{h_1}{2} - s_1 \right) \right\rfloor .$$

– Reject key pairs for which Q_{00} is "too small" $\Longrightarrow$ recovery works "always" ($\geq 1 - 2^{-100}$).

Dropping s_0

Sign algorithm Sign($\mathbf{U}, m$):

1. Hash to $\mathbf{h} = H(m) \in \{0,1\}^{2n} \quad$ '$\subseteq$' R^2.
2. Sample $\mathbf{x} \in R^2$ close to $\frac{1}{2}\mathbf{U} \cdot \mathbf{h}$.
3. Return $\mathbf{s} = \mathbf{U}^{-1} \cdot \mathbf{x}$.

– Note: $\mathbf{Us}$ is close to $\frac{1}{2}\mathbf{Uh}$, so $s_0\mathbf{u}_0$ is close to $\frac{1}{2}\mathbf{Uh} - s_1\mathbf{u}_1$.

$\Longrightarrow$ Use Babai's round-off (or nearest plane) algorithm:

$$s_0' = \left\lfloor \frac{h_0}{2} + \frac{\mathbf{Q}_{01}}{\mathbf{Q}_{00}} \left(\frac{h_1}{2} - s_1 \right) \right\rceil.$$

– Reject key pairs for which $\mathbf{Q}_{00}$ is "too small" $\Longrightarrow$ recovery works "always" ($\geq 1 - 2^{-100}$).

Dropping s_0

Sign algorithm $\mathrm{Sign}(\mathbf{U}, m)$:

1. Hash to $\mathbf{h} = H(m) \in \{0,1\}^{2n}$ '$\subseteq$' R^2.
2. Sample $\mathbf{x} \in R^2$ close to $\frac{1}{2}\mathbf{U} \cdot \mathbf{h}$.
3. Return $\mathbf{s} = \mathbf{U}^{-1} \cdot \mathbf{x}$.

– Note: $\mathbf{Us}$ is close to $\frac{1}{2}\mathbf{Uh}$, so $s_0\mathbf{u}_0$ is close to $\frac{1}{2}\mathbf{Uh} - s_1\mathbf{u}_1$.

$\implies$ Use Babai's round-off (or nearest plane) algorithm:

$$s_0' = \left\lceil \frac{h_0}{2} + \frac{\mathbf{Q}_{01}}{\mathbf{Q}_{00}} \left(\frac{h_1}{2} - s_1 \right) \right\rfloor .$$

– Reject key pairs for which $\mathbf{Q}_{00}$ is "too small" $\implies$ recovery works "always" ($\geq 1 - 2^{-100}$).

Performance of Hawk

Performance of HAWK

- HAWK has an *isochronous* implementation in C, using lots of code from FALCON.

	FALCON-512	HAWK-512		FALCON-1024	HAWK-1024	
KeyGen*	7.95 ms	4.25 ms	$\downarrow$ /1.9	23.60 ms	17.88 ms	$\downarrow$ /1.3
Sign*	193 µs	50 µs	$\downarrow$ /3.9	382 µs	99 µs	$\downarrow$ /3.9
Verify*	50 µs	19 µs	$\downarrow$ /2.6	99 µs	46 µs	$\downarrow$ /2.2
|sk| (bytes)	1281	1153	$\downarrow$ /1.1	2305	2561	$\uparrow$ $\times$1.1
|pk| (bytes)	897	1006 ± 6	$\uparrow$ $\times$1.2	1793	2329 ± 11	$\uparrow$ $\times$1.29
|sig| (bytes)	652 ± 3	542 ± 4	$\downarrow$ /1.20	1261 ± 4	1195 ± 6	$\downarrow$ /1.06

Table 1: Performance on an i5-4590 @3.30GHz CPU. *: AVX2 implementation using floats.

Performance of HAWK

- HAWK has an *isochronous* implementation in C, using lots of code from FALCON.
- When floating points are unavailable, FALCON emulates these, but HAWK signs with the NTT instead.

	FALCON-512	HAWK-512		FALCON-1024	HAWK-1024	
KeyGen*	7.95 ms	4.25 ms	↓ /1.9	23.60 ms	17.88 ms	↓ /1.3
KeyGen	19.32 ms	13.14 ms	↓ /1.5	54.65 ms	41.39 ms	↓ /1.3
Sign*	193 μs	50 μs	↓ /3.9	382 μs	99 μs	↓ /3.9
Sign	2449 μs	168 μs	↓ /15	5273 μs	343 μs	↓ /15
Verify*	50 μs	19 μs	↓ /2.6	99 μs	46 μs	↓ /2.2
Verify	53 μs	178 μs	↑ ×3.4	105 μs	392 μs	↑ ×3.7
$\|sk\|$ (bytes)	1281	1153	↓ /1.1	2305	2561	↑ ×1.1
$\|pk\|$ (bytes)	897	1006 ± 6	↑ ×1.2	1793	2329 ± 11	↑ ×1.29
$\|sig\|$ (bytes)	652 ± 3	542 ± 4	↓ /1.20	1261 ± 4	1195 ± 6	↓ /1.06

Table 1: Performance on an i5-4590 @3.30GHz CPU. *: AVX2 implementation using floats.

Open questions

Open questions

- Can we use a better lattice (sampling closer to a target)?
- More cryptanalysis on (module-)LIP wanted!

Open questions

- Can we use a better lattice (sampling closer to a target)?
- More cryptanalysis on (module-)LIP wanted!

Conclusion

We hide a rotation of $\mathbb{Z}^n$.

- ✓ Sampling becomes far simpler and faster.
- ✓ Floating points are avoided.
- ✓ We get a fast and compact signature scheme.

Thank you! Questions?

ePrint: `https://ia.cr/2022/1155`
code: `https://github.com/ludopulles/hawk-sign`

Conclusion

We hide a rotation of $\mathbb{Z}^n$.

- ✓ Sampling becomes far simpler and faster.
- ✓ Floating points are avoided.
- ✓ We get a fast and compact signature scheme.

Thank you! Questions?

ePrint: `https://ia.cr/2022/1155`
code: `https://github.com/ludopulles/hawk-sign`

Full Key generation code

$\mathsf{KeyGen}(1^\lambda)$

$f, g \leftarrow D_{\mathbb{Z}^n, \sigma_{\mathrm{pk}}}$

$q_{00} = f^* f + g^* g$

if $2 \mid \mathrm{N}(f)$ **or** $2 \mid \mathrm{N}(g)$ **or** $\|(f, g)\|^2 \leq \sigma_{\mathrm{sec}}^2 \cdot 2n$ **or** $\langle 1, q_{00}^{-1} \rangle \geq \nu_{\mathrm{dec}}$

  restart

$(F, G)^{\mathsf{T}} \leftarrow \mathrm{NTRUSolve}_1(f, g)$, **or restart if it fails**

$(F, G)^{\mathsf{T}} \leftarrow (F, G)^{\mathsf{T}} - \mathrm{ffNP}_R\left(\dfrac{f^* F + g^* G}{q_{00}}, q_{00}\right) \cdot (f, g)^{\mathsf{T}}$

$\mathsf{B} = \begin{pmatrix} f & F \\ g & G \end{pmatrix}.$

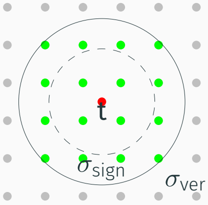
$\mathsf{Q} = \begin{pmatrix} q_{00} & q_{01} \\ q_{10} & q_{11} \end{pmatrix} = \mathsf{B}^* \cdot \mathsf{B}.$

return $(\mathrm{pk}, \mathrm{sk}) = (\mathsf{Q}, \mathsf{B})$

Sign & Verify

$\mathsf{Sign}_{\mathbf{B}}(m)$

$r \leftarrow\!\!\$ \{0,1\}^{\mathrm{saltlen}}$
$\mathbf{h} \leftarrow \mathrm{H}(m \| r)$
$\mathbf{t} \leftarrow \frac{1}{2}\mathbf{B}\mathbf{h}$
$\mathbf{x} \leftarrow D_{\sigma_{\mathrm{sign}},\mathbf{t}}$
if $\|\mathbf{x}-\mathbf{t}\|^2 > 2n \cdot \sigma_{\mathrm{ver}}^2$:
    restart
return $(r, \mathbf{B}^{-1} \cdot \mathbf{x})$



$\mathsf{Verify}_{\mathbf{Q}}(m, (r, \mathbf{s}))$

$\mathbf{h} \leftarrow \mathrm{H}(m \| r)$
return $\left[\!\left[\mathbf{s} \in R \oplus R \text{ and } \left\| \frac{\mathbf{h}}{2} - \mathbf{s} \right\|_{\mathbf{Q}}^2 \leq 2n \cdot \sigma_{\mathrm{ver}}^2 \right]\!\right]$

⚠ Beware:$(r, \mathbf{h} - \mathbf{s})$ would be a weak forgery for m.
Fix: demand first nonzero coefficient of $\frac{\mathbf{h}}{2} - \mathbf{s}$ to be positive.