New results and trade-offs for Lattice-based Hash-and-sign signatures

Alexandre Wallet

based on joint works with T.Espitau, P.A. Fouque, F.Gerard, M.Rossi, A.Takahashi, M.Tibouchi, Y.Yu

CHARM Seminar, online — 21/01/2022

Lattice-based signatures in NIST's call

As of Round 3, two among three finalists are lattice-based.

FALCON

"Hash-and-sign" in lattices [GPV'08]
+ NTRU trapdoors [DLP'14]

CRYSTALS-DILITHIUM

Fiat-Shamir "with abort" [Lyu12]
+ module lattices

Lattice-based signatures in NIST's call

As of Round 3, two among three finalists are lattice-based.

FALCON

"Hash-and-sign" in lattices [GPV'08]
+ **NTRU trapdoors** [DLP'14]

✓ best bandwith, fast

✗ restricted parameter set
✗ difficult implementation
✗ expensive masking

CRYSTALS-DILITHIUM

Fiat-Shamir "with abort" [Lyu12]
+ module lattices

Lattice-based signatures in NIST's call

As of Round 3, two among three finalists are lattice-based.

FALCON

"Hash-and-sign" in lattices [GPV'08]
+ **NTRU trapdoors** [DLP'14]

✓ best bandwith, fast

✗ restricted parameter set
✗ difficult implementation
✗ expensive masking

CRYSTALS-DILITHIUM

Fiat-Shamir "with abort" [Lyu12]
+ module lattices

Today: `Mitaka`

mitigate *all* these shortcomings!

Other features and results

Specific to Mitaka[1]:

- Simple, cheap masking
- Fixed-point arithmetic friendly (over 2-powers cyclotomic rings) (not today)

For Falcon & Mitaka[2]:

- shorter signatures with elliptic sampling
- trade-off between bandwith and verification speed
- a generic compression technique for gaussian vectors (not today)

Overall: up to 40% **smaller signatures for minimal security loss.**

[1] *Mitaka: a simpler, parallelizable, maskable variant of Falcon*, EUROCRYPT 2022
[2] *Shorter hash-and-sign lattice-based signatures*, CRYPTO 2022

Roadmap

Quickview of the GPV framework, and Falcon's design

Sampling over (structured) lattices

NTRU lattices and their bases

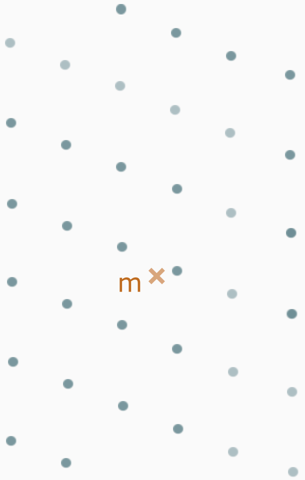
Masking Mitaka

In practice

Making signatures even shorter

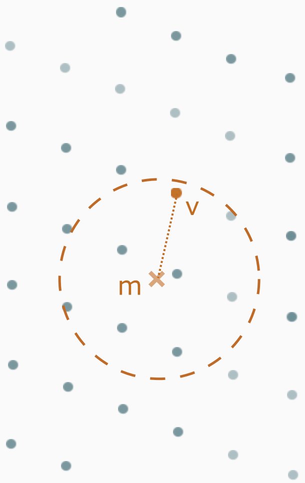
Hash-and-Sign over lattices (in a nutshell)

1) Hash msg as a random point $\mathbf{m}$ in the space



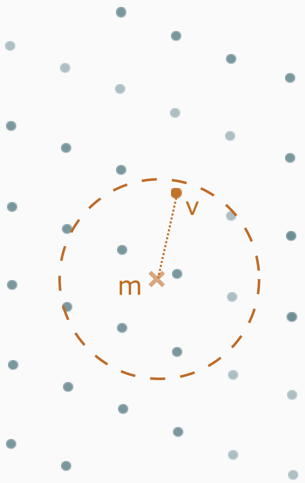
Hash-and-Sign over lattices (in a nutshell)

1) Hash msg as a random point $\mathbf{m}$ in the space
2) Sample a **random** point $\mathbf{v}$ in the lattice, close to $\mathbf{m}$



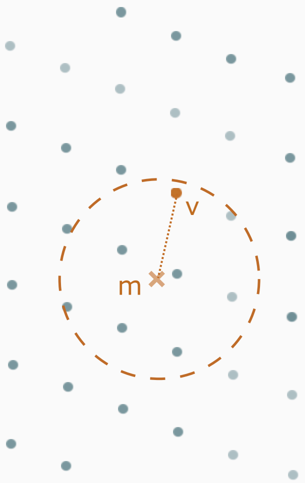
Hash-and-Sign over lattices (in a nutshell)

1) Hash msg as a random point $\mathbf{m}$ in the space
2) Sample a **random** point $\mathbf{v}$ in the lattice, close to $\mathbf{m}$
3) Signature: $\mathbf{s} = \mathbf{m} - \mathbf{v}$



Hash-and-Sign over lattices (in a nutshell)

1) Hash msg as a random point $\mathbf{m}$ in the space
2) Sample a **random** point $\mathbf{v}$ in the lattice, close to $\mathbf{m}$
3) Signature: $\mathbf{s} = \mathbf{m} - \mathbf{v}$



- Enough lattice points close to any target
- Forgery $\sim$ **$\mathbf{CVP}_\gamma$**: should be **hard**
- Public lattice
- Efficient sampling procedure for signer
- The sampler should not leak signer's secrets

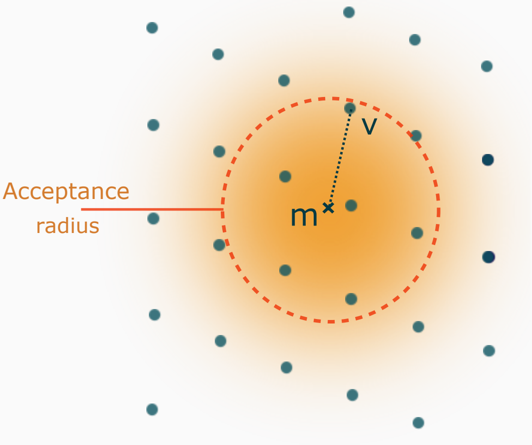
The GPV Framework

Simplified $\text{Sign}_{\mathbf{sk},\sigma}(\text{msg})$:

1. $\mathbf{m} = \mathcal{H}(\text{msg})$
2. $\mathbf{v} \leftarrow \text{GaussianSampler}(\mathbf{sk}, \mathbf{m}, \sigma)$
3. Signature: $\mathbf{s} = \mathbf{m} - \mathbf{v}$.

Simplified $\text{Verif}_{\mathcal{L}=\mathbf{pk}}(\text{msg}, \mathbf{s})$:

1. If $\|\mathbf{s}\|$ too big, reject.
2. If $\mathbf{m} - \mathbf{s} \notin \mathcal{L}$, reject.
3. Accept.



the GPV Framework, explicitely

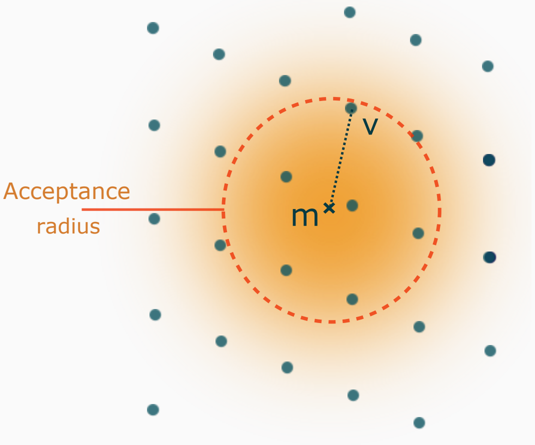
Take $\mathcal{L} = \Lambda_q^{\perp}(\mathbf{A})$ q-ary lattice with basis $\mathbf{B}$, then $\mathbf{AB} = \mathbf{0} \bmod q$

Simplified $\mathrm{Sign}_{\mathbf{B},\sigma}(\mathrm{msg})$:

1. $\mathbf{c}$ such that $\mathbf{Ac} = \mathcal{H}(\mathrm{msg})$
2. $\mathbf{v} \leftarrow$ GaussianSampler$(\mathbf{B}, \mathbf{c}, \sigma)$
3. Signature: $\mathbf{s} = \mathbf{c} - \mathbf{v}$.

Simplified $\mathrm{Verif}_{\mathbf{A}}(\mathrm{msg}, \mathbf{s})$:

1. If $\|\mathbf{s}\|$ too big, refuse.
2. If $\mathbf{As} \neq \mathcal{H}(\mathrm{msg})$, refuse.
3. Accept.



the GPV Framework, explicitely

Take $\mathcal{L} = \Lambda_q^\perp(\mathbf{A})$ q-ary lattice with basis $\mathbf{B}$, then $\mathbf{AB} = \mathbf{0} \bmod q$

Simplified $\mathrm{Sign}_{\mathbf{B},\sigma}(\mathrm{msg})$:

1. $\mathbf{c}$ such that $\mathbf{Ac} = \mathcal{H}(\mathrm{msg})$
2. $\mathbf{v} \leftarrow \mathrm{GaussianSampler}(\mathbf{B}, \mathbf{c}, \sigma)$
3. Signature: $\mathbf{s} = \mathbf{c} - \mathbf{v}$.

Simplified $\mathrm{Verif}_{\mathbf{A}}(\mathrm{msg}, \mathbf{s})$:

1. If $\|\mathbf{s}\|$ too big, refuse.
2. If $\mathbf{As} \neq \mathcal{H}(\mathrm{msg})$, refuse.
3. Accept.

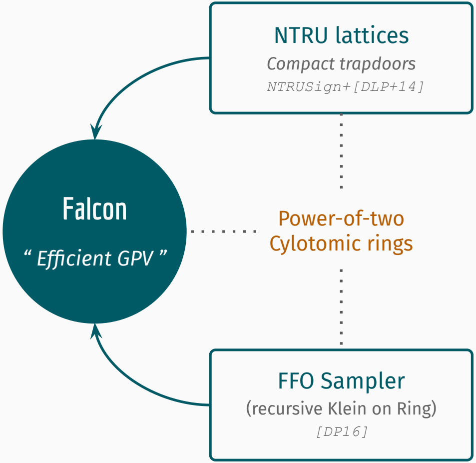
Requirements:

$\mathbf{CVP}_\gamma$ hard $\Rightarrow \sigma$ small $\Rightarrow \mathbf{B}$ has short vectors

Hard to compute $\mathbf{B}$ just from $\mathbf{A}$	Easy to generate $\mathbf{A}$ just from $\mathbf{B}$

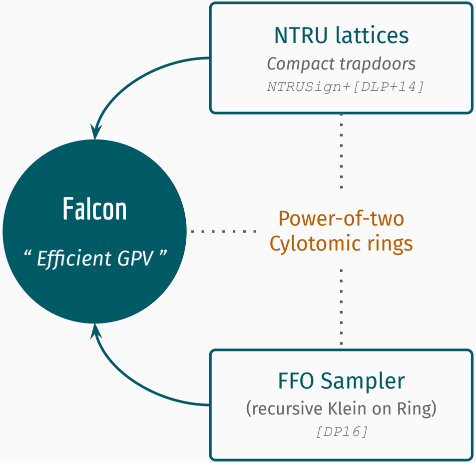
$\mathbf{B}$ is called *"a trapdoor"*

"Falcon: a quest for compactness"



NTRU lattices: free rank 2 modules over (polynomial, cyclotomic) rings

“Falcon: a quest for compactness”



NTRU lattices: free rank 2 modules over (polynomial, cyclotomic) rings

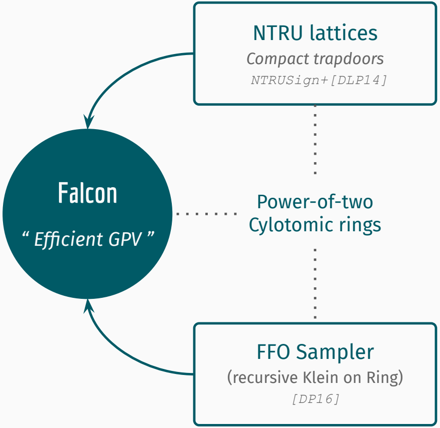
Pros:

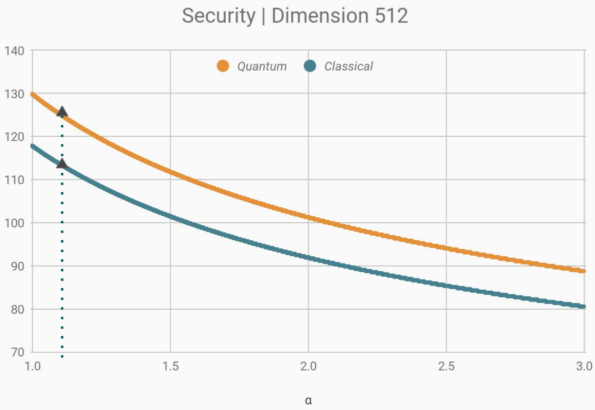
- ✓ **Best bandwith of NIST signatures**
- ✓ Fast signing, fast verification
- ✓ Quasi-linear **thank to the ring**

Cons:

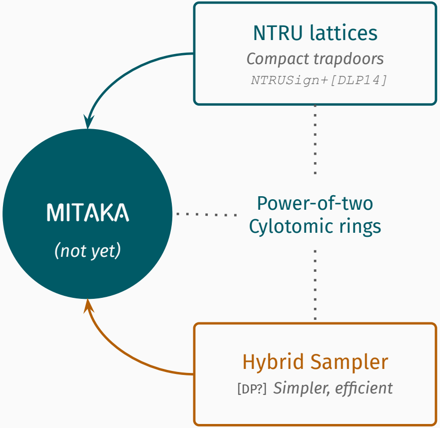
- ✗ Few parameter sets
- ✗ Complicated implementation
- ✗ Expensive protections

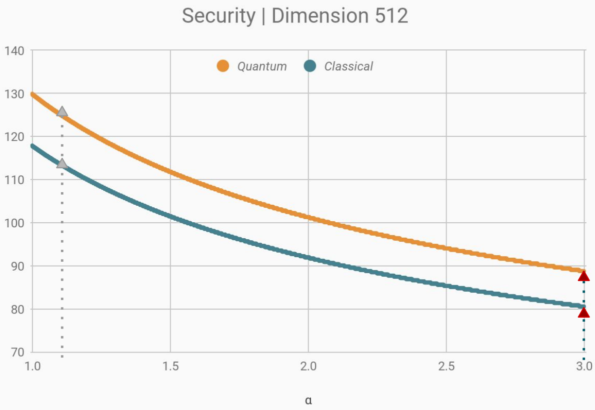
Toward Mitaka



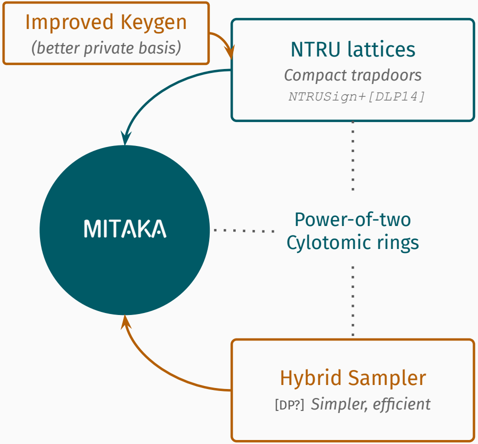


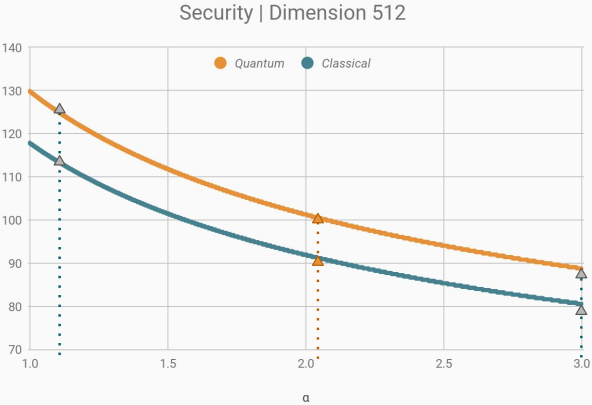
Toward Mitaka



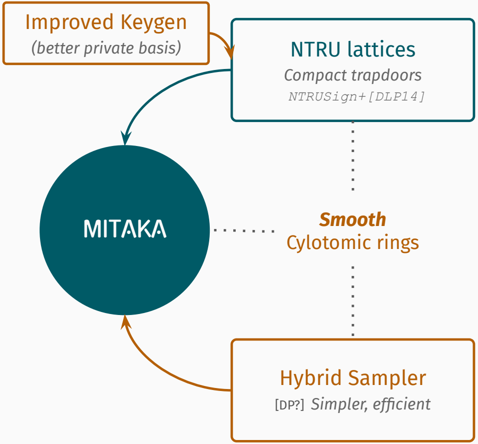


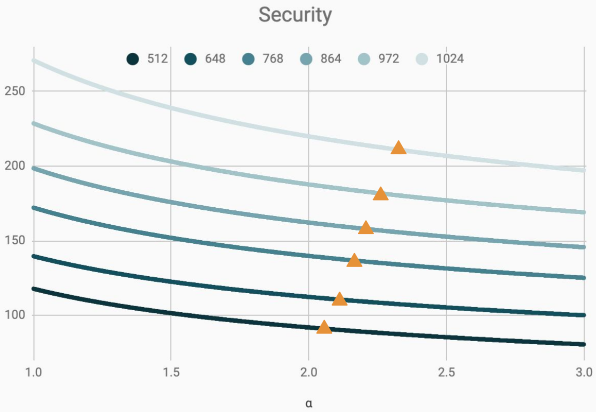
Toward Mitaka



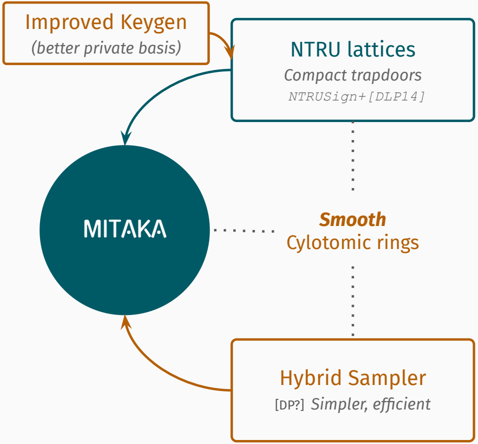


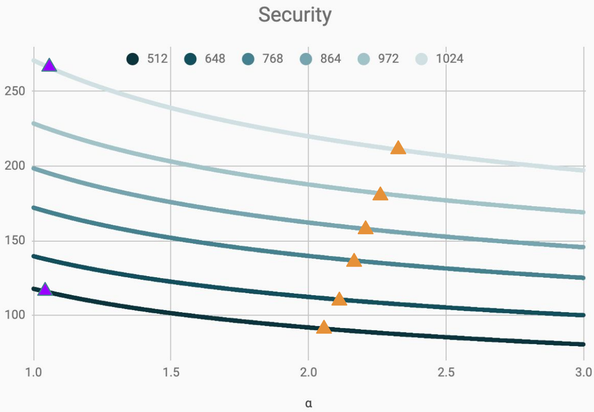
Toward Mitaka





Toward Mitaka





Some Gaussian samplers

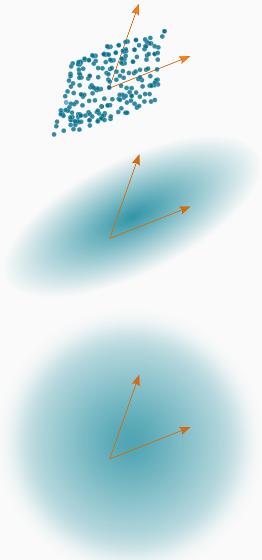
Lattice Gaussian samplers = decoding + randomization

Famous lattice decoders	**made into Gaussian samplers:**
Babai's Round-off: Round target's coords in the lattice basis.	Randomize the roundings
Babai's Nearest Plane: Adaptively round target's coords in the *Gram-Schmidt* basis.	Randomize adaptively

There are also "in-betweens", e.g. *Ducas-Prest hybrid sampler* *(We'll cover that soon)*

Randomized Round-off

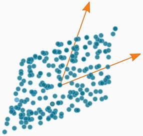
Without randomization



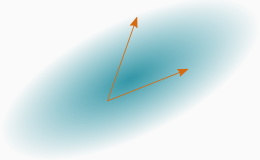
Outputs $\mathbf{z} = \mathbf{B}\lceil \mathbf{B}^{-1}\mathbf{t} \rfloor$

Randomized Round-off

Without randomization



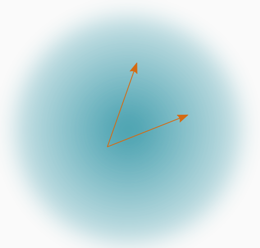
Randomize rounding
w/ discrete Gaussians
Leaks the lattice basis!



$\mathbf{y} \leftarrow \lceil \mathbf{B}^{-1}\mathbf{t} \rfloor_r$

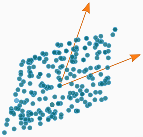
means $\mathbf{y} \hookleftarrow D_{\mathbb{Z}^n - \mathbf{B}^{-1}\mathbf{t}, r}$

Outputs $\mathbf{z} = \mathbf{B}\mathbf{y}$

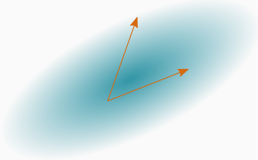


Randomized Round-off

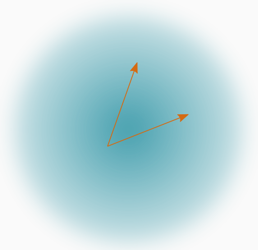
Without randomization



Randomize rounding
w/ discrete Gaussians
Leaks the lattice basis!



Add Gaussian perturbation to
"*smooth out*" the lattice
(C. Peikert, CRYPTO 2010)



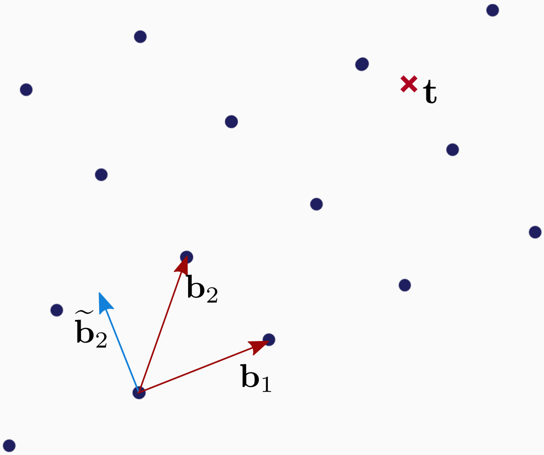
Peikert($\mathbf{B}, \mathbf{t}, \sigma, r$):

$\mathbf{x} \leftarrow \sigma \cdot \mathcal{N}(0,1)$

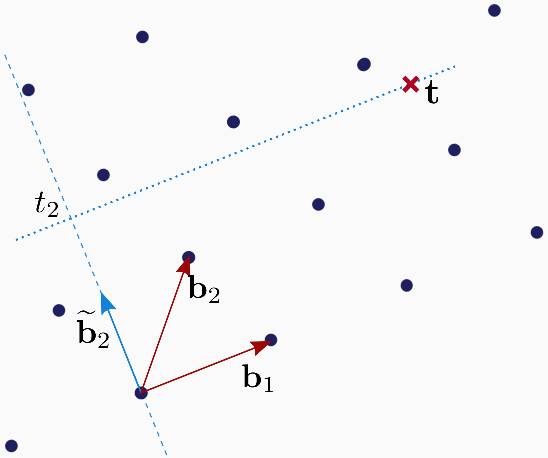
$\mathbf{y} \leftarrow \lceil \mathbf{B}^{-1}\mathbf{t} - \mathbf{x} \rfloor_r$

Outputs $\mathbf{z} = \mathbf{B}\mathbf{y}$.

NearestPlane in pictures



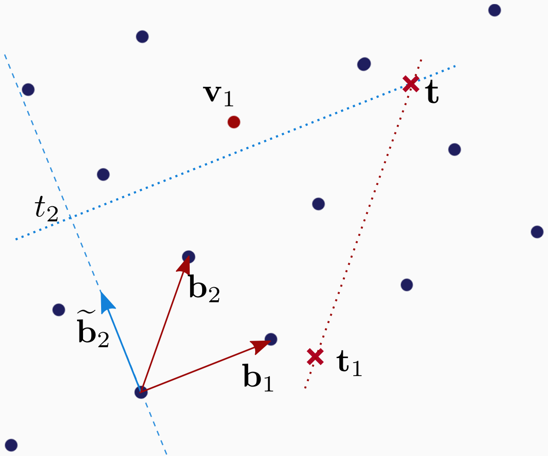
NearestPlane in pictures



Step 1:

$$t_2 := \frac{\langle \mathbf{t}, \widetilde{\mathbf{b}}_2 \rangle}{\|\widetilde{\mathbf{b}}_2\|}, \text{ and } \lceil t_2 \rfloor = 2$$

NearestPlane in pictures

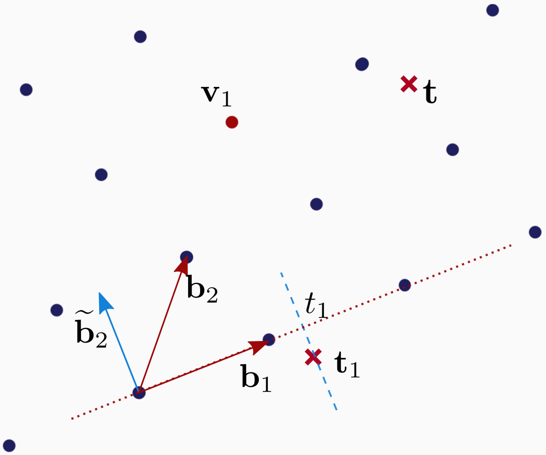


Step 1:

$$t_2 := \frac{\langle \mathbf{t}, \widetilde{\mathbf{b}}_2 \rangle}{\|\widetilde{\mathbf{b}}_2\|}, \text{ and } \lceil t_2 \rfloor = 2$$

$$\mathbf{t}_1 := \mathbf{t} - \lceil t_2 \rfloor \mathbf{b}_2, \text{ and } \mathbf{v}_1 := \lceil t_2 \rfloor \mathbf{b}_2.$$

NearestPlane in pictures



Step 1:

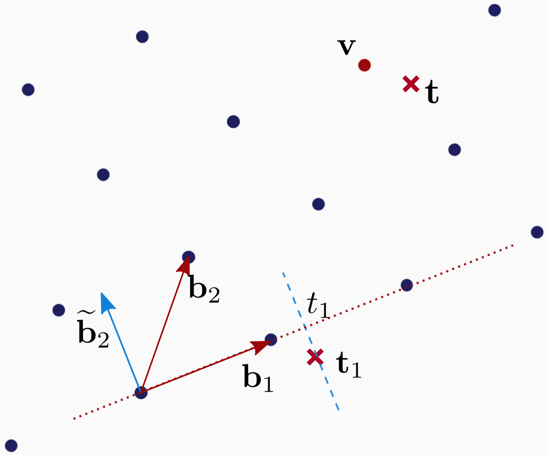
$$t_2 := \frac{\langle \mathbf{t}, \widetilde{\mathbf{b}}_2 \rangle}{\|\widetilde{\mathbf{b}}_2\|}, \text{ and } \lceil t_2 \rfloor = 2$$

$$\mathbf{t}_1 := \mathbf{t} - \lceil t_2 \rfloor \mathbf{b}_2, \text{ and } \mathbf{v}_1 := \lceil t_2 \rfloor \mathbf{b}_2.$$

Step 2:

$$t_1 := \frac{\langle \mathbf{t}_1, \mathbf{b}_1 \rangle}{\|\mathbf{b}_1\|}, \text{ and } \lceil t_1 \rfloor = 1$$

NearestPlane in pictures



Step 1:

$$t_2 := \frac{\langle \mathbf{t}, \widetilde{\mathbf{b}}_2 \rangle}{\|\widetilde{\mathbf{b}}_2\|}, \text{ and } \lceil t_2 \rfloor = 2$$

$$\mathbf{t}_1 := \mathbf{t} - \lceil t_2 \rfloor \mathbf{b}_2, \text{ and } \mathbf{v}_1 := \lceil t_2 \rfloor \mathbf{b}_2.$$

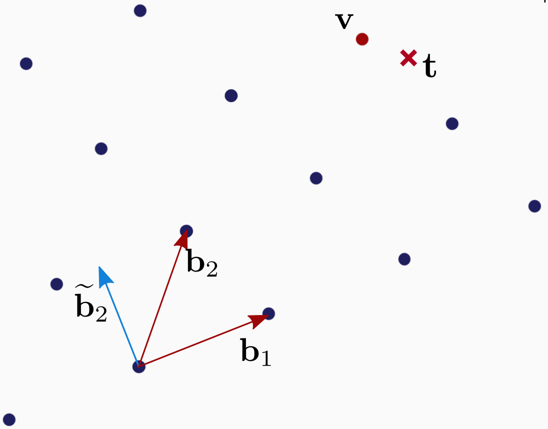
Step 2:

$$t_1 := \frac{\langle \mathbf{t}_1, \mathbf{b}_1 \rangle}{\|\mathbf{b}_1\|}, \text{ and } \lceil t_1 \rfloor = 1$$

$$\mathbf{v} := \mathbf{v}_1 + \lceil t_1 \rfloor \mathbf{b}_1.$$

NearestPlane in pictures

$\mathbf{B}\lfloor\mathbf{B}^{-1}\mathbf{t}\rceil$



Step 1:

$t_2 := \frac{\langle \mathbf{t}, \widetilde{\mathbf{b}}_2 \rangle}{\|\widetilde{\mathbf{b}}_2\|}$, and $\lfloor t_2 \rceil = 2$

$\mathbf{t}_1 := \mathbf{t} - \lfloor t_2 \rceil \mathbf{b}_2$, and $\mathbf{v}_1 := \lfloor t_2 \rceil \mathbf{b}_2$.

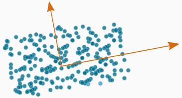
Step 2:

$t_1 := \frac{\langle \mathbf{t}_1, \mathbf{b}_1 \rangle}{\|\mathbf{b}_1\|}$, and $\lfloor t_1 \rceil = 1$

$\mathbf{v} := \mathbf{v}_1 + \lfloor t_1 \rceil \mathbf{b}_1$.

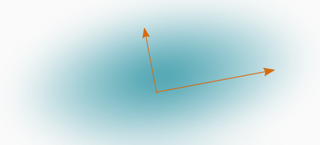
Randomized NearestPlane: Klein's sampler

Without randomization

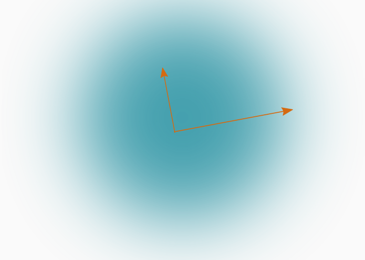


Randomize the rounding of each $t_i \in \mathbb{R}$

Leaks Gram-Schmidt basis!



On each $\mathbb{R}\widetilde{\mathbf{b}}_i$, rescale adaptively: $s_i := \dfrac{s}{\|\widetilde{\mathbf{b}}_i\|}$



Klein$(\mathbf{B}, \widetilde{\mathbf{B}}, \mathbf{t}, s_i, r)$:

$\mathbf{v} = 0, \mathbf{c} = \mathbf{t}$

for $i = n$ **to** 1:

$t_i = \left\lceil \frac{\langle \mathbf{t}, \widetilde{\mathbf{b}}_i \rangle}{\|\widetilde{\mathbf{b}}_i\|^2} \right\rfloor_{s_i}$

$\mathbf{v} = \mathbf{v} + t_i \mathbf{b}_i$

$\mathbf{c} = \mathbf{c} - t_i \mathbf{b}_i$

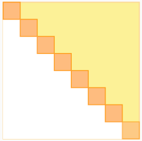
Outputs $\mathbf{v}$

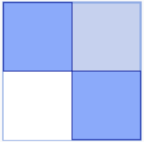
Short interlude: module lattices in 1 min

Euclidean lattices	$\supsetneq$	**Module lattices**
base ring is $\mathbb{Z}$		base ring is $R = \mathbb{Z}[x]/(f(x)) \sim \mathbb{Z}^d$

$$\mathcal{L} = \mathbf{b}_1\mathbb{Z} \oplus \cdots \oplus \mathbf{b}_m\mathbb{Z} = \mathbf{B}\mathbb{Z}^m$$

$$\mathcal{M} = \mathbf{b}_1 R \oplus \cdots \oplus \mathbf{b}_k R \sim [\mathbf{b}_1]\mathbb{Z}^d \oplus \cdots \oplus [\mathbf{b}_k]\mathbb{Z}^d \sim [\mathbf{B}]\mathbb{Z}^{kd}$$





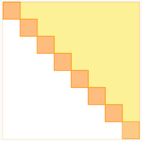
Crypto: $k \leqslant 5$, $d \geqslant 256$.

Hybrid sampling

Hybrid $=$ Klein decoding over R^2

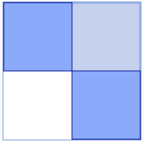
$+$ (Peikert) randomization in R.

Klein



decoding in $2d$
randomization in $\mathbb{Z}$

Hybrid



decoding in rank 2
randomization in $\mathcal{R}$

Ex: R power-of-2 cyclotomic and $k = 2$

Hybrid$(\mathbf{B}, \widetilde{\mathbf{B}}_R, \mathbf{t}, s_1, s_2)$:

$\mathbf{v} = 0, \mathbf{c} = \mathbf{t}$

for $i = 2$ to 1:

$t_i = \mathrm{Peikert}\left(\mathbf{I}, \frac{\langle \mathbf{t}, \widetilde{\mathbf{b}}_i \rangle_R}{\langle \widetilde{\mathbf{b}}_i, \widetilde{\mathbf{b}}_i \rangle_R}, s_i, r\right)$

$\mathbf{v} = \mathbf{v} + t_i \mathbf{b}_i$

$\mathbf{c} = \mathbf{c} - t_i \mathbf{b}_i$

Outputs $\mathbf{v}$

operations in R

$\Rightarrow$ **need "good FFT domain"**

Comparisons of samplers

	Pros	**Cons**	**Quality $\mathcal{Q}(\mathbf{B})$**
Peikert	fast simple	worst quality (*lower security*)	$s_1(\mathbf{B})$ (largest sing. value)
Hybrid	**Good tradeoffs when $\mathcal{R}$ has a *good basis***		$s_1(\widetilde{\mathbf{B}})$
Klein	best quality (*higher security*)	slower more involved	$\max_i \|\widetilde{\mathbf{b}_i}\|$

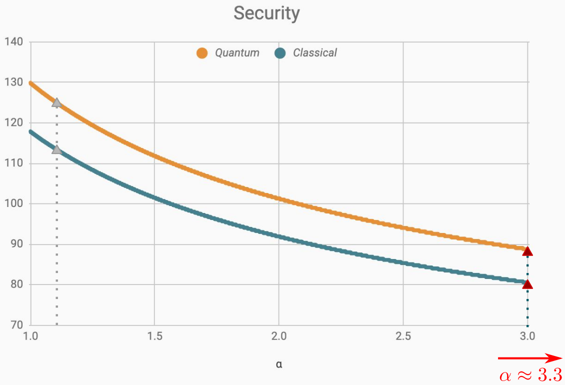
From quality to security

For NTRU-like q-ary lattices, the **quality factor** is $\alpha := \frac{\mathcal{Q}(\mathbf{B})}{\sqrt{q}}$

Concrete bitsecurity as a function of α
over 2-powers cyclotomics

$\alpha_{\mathsf{Falcon}} = 1.17$

$\alpha_{\mathsf{Hybrid}} \geqslant 3.3$ (naively)



NTRU Lattices

$\mathcal{R}$ some ring in a number field
(say, $\mathcal{R} = \mathbb{Z}[x]/(x^d + 1)$ with $d = 512$)

$a = \sum_i a_i X^i$
$[a]$ matrix of multiplication by f

NTRU lattice: let $f, g \in \mathcal{R}$, $a := g/f \bmod q$.

$$\mathcal{L}_{\mathsf{NTRU}}(a) := \Lambda_q^{\perp}((a, -1))$$

$$\begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} a \\ -1 \end{bmatrix} = 0 \bmod q$$

public basis

$$\begin{pmatrix} 1 & a \\ 0 & q \end{pmatrix} \leftrightarrow \left[\begin{array}{c|c} \mathrm{Id}_d & [a] \\ \hline [0] & q\mathrm{Id}_d \end{array}\right]$$

$\mathcal{L}_{\mathsf{NTRU}}(a)$ has rank $2d$ and volume q^d

Expect fundamental quantities to be $\approx \sqrt{q}$

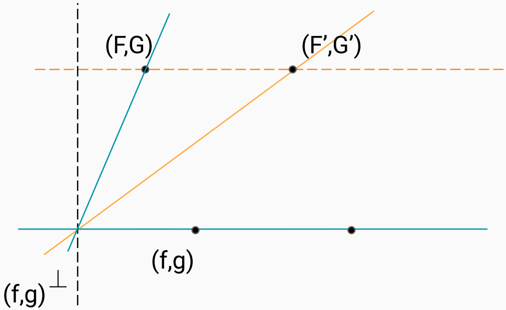
NTRU Trapdoors for signatures

A **trapdoor** is a **short** basis **B** of $\mathcal{L}_{\mathsf{NTRU}}(a)$ with **good quality** wrt. a sampler.

$$\mathbf{B} \begin{bmatrix} a \\ -1 \end{bmatrix} = 0 \bmod q$$

Computing B

- Take f, g so that $\|(f, g)\| \approx \sqrt{q}$
- Complete the basis with a short (F, G):
  "Reverse" Nearest Plane
  (or Euclid algorithm + geometry)



NTRU Trapdoors for signatures

A **trapdoor** is a **short** basis **B** of $\mathcal{L}_{\mathsf{NTRU}}(a)$ with **good quality** wrt. a sampler.

$$\mathbf{B} \begin{bmatrix} a \\ -1 \end{bmatrix} = 0 \bmod q$$

Computing B

- Take f, g so that $\|(f, g)\| \approx \sqrt{q}$
- Complete the basis with a short (F, G): "Reverse" Nearest Plane (or Euclid algorithm + geometry)

Achieve good quality

Sample **Gaussians** (f, g)'s until:

- `Falcon`: $\max(\|f, g\|, \|\widetilde{F}, \widetilde{G}\|) \approx 1.17\sqrt{q}$
- `Mitaka`: $s_1(\widetilde{\mathbf{B}})$ as close as possible to $\sqrt{q}$

Both metrics can be computed just with f, g

Into the key generation algorithm

(naive) **KeyGen**:

1) **Do** $\quad f, g \leftarrow D_{\mathbb{Z}^d, \sqrt{\frac{q}{2d}}}$

   Until f inv. mod q **And** $\|f, g\| \leqslant 1.17\sqrt{q}$;

2) *(F) quality check:* $\|\widetilde{F}, \widetilde{G}\| \leqslant 1.17\sqrt{q}$?

   else restart;

4) $(F, G) \leftarrow \mathrm{BasisCompletion}(f, g, q)$;

   Compute all needed data;

   Output (pk, sk).

Into the key generation algorithm

(naive) **KeyGen**:

1) **Do** $f, g \leftarrow D_{\mathbb{Z}^d, \sqrt{\frac{q}{2d}}}$
   Until f inv. mod q **And** $\|f, g\| \leqslant 1.17\sqrt{q}$;

2) *(F) quality check:* $\|\widetilde{F}, \widetilde{G}\| \leqslant 1.17\sqrt{q}$?
   else restart;

2-bis) *(M) quality check:* $s_1(\widetilde{\mathbf{B}}) \leqslant 2.05\sqrt{q}$?
   else restart;

4) $(F, G) \leftarrow \mathrm{BasisCompletion}(f, g, q)$;
   Compute all needed data;
   Output (pk, sk).

Into the key generation algorithm

(naive) **KeyGen**:

1) **Do** $\quad f, g \leftarrow D_{\mathbb{Z}^d, \sqrt{\frac{q}{2d}}}$
   Until f inv. mod q **And** $\|f, g\| \leqslant 1.17\sqrt{q}$;

2) *(F) quality check:* $\|\widetilde{F}, \widetilde{G}\| \leqslant 1.17\sqrt{q}$?
   else restart;

2-bis) *(M) quality check:* $s_1(\widetilde{\mathbf{B}}) \leqslant 2.05\sqrt{q}$?
   else restart;

4) $(F, G) \leftarrow \mathrm{BasisCompletion}(f, g, q)$;
   Compute all needed data;
   Output (pk, sk).

Randomness is expensive, yet:

- This already happens often in `Falcon`
- Need ***a lot*** of tries to reach 2.05

Into the key generation algorithm

(naive) **KeyGen**:

1) **Do** $f, g \leftarrow D_{\mathbb{Z}^d, \sqrt{\frac{q}{2d}}}$
   Until f inv. mod q **And** $\|f, g\| \leqslant 1.17\sqrt{q}$;
2) *(F) quality check:* $\|\widetilde{F}, \widetilde{G}\| \leqslant 1.17\sqrt{q}$?
   else restart;

2-bis) *(M) quality check:* $s_1(\widetilde{\mathbf{B}}) \leqslant 2.05\sqrt{q}$?
   else restart;

4) $(F, G) \leftarrow \mathrm{BasisCompletion}(f, g, q)$;
   Compute all needed data;
   Output (pk, sk).

Our solution: amortize!

+ Reuse randomness
+ Galois automorphisms

= "Free" blow-up of search-space (say, quartic)

$\Rightarrow$ good trapdoors in reasonable time

Into the key generation algorithm

KeyGen (σ^2 target variance; m, n number of samples; set $\mathcal{S}$ of Galois automorphisms)

1) [Sampling]
   - $\mathcal{F}', \mathcal{F}'' \leftarrow m$ Gaussian vectors of variance $\sigma^2/2$
   - $\mathcal{G} \leftarrow n$ Gaussian vectors of variance σ^2
2) [Blowing up]
   - Pair two lists $\mathcal{F} \leftarrow \mathcal{F}' + \mathcal{F}''$
   - Let $\mathcal{S}$ act on $\mathcal{G}$: $\mathcal{G} \leftarrow \bigcup_{\tau \in \mathcal{S}} \tau(\mathcal{G})$

For the generation cost of $2m + n$ Gaussians, search a space of size

$$\mathrm{Card}(\mathcal{S}) \cdot m^2 n$$

Into the key generation algorithm

KeyGen (σ^2 target variance; m, n number of samples; set S of Galois automorphisms)

1) [Sampling]
   - $\mathcal{F}', \mathcal{F}'' \leftarrow m$ Gaussian vectors of variance $\sigma^2/2$
   - $\mathcal{G} \leftarrow n$ Gaussian vectors of variance σ^2
2) [Blowing up]
   - Pair two lists $\mathcal{F} \leftarrow \mathcal{F}' + \mathcal{F}''$
   - Let S act on $\mathcal{G}$: $\mathcal{G} \leftarrow \bigcup_{\tau \in S} \tau(\mathcal{G})$
3) [Testing] **For** $f \in \mathcal{F}, g \in \mathcal{G}$ **do**

   If quality-testing(f, g)

   Output $(\mathrm{pk}(f, g), \mathrm{sk}(f, g))$.

For the generation cost of $2m + n$ Gaussians, search a space of size

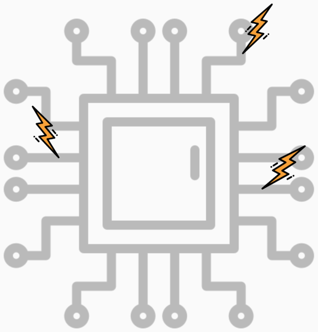
$$\mathrm{Card}(S) \cdot m^2 n$$

Faster with additional tricks (filtering, early aborts, ...)

Modeling side-channel adversaries

- Adversary obtains t intermediate values of the computation
- Successfully models practical **noisy side-channel leakage** [DDF14]

- Any set of at most t intermediate variables is independent of the secret.



Protecting Mitaka from t-probing adversary: an overview

Arithmetic masking of $x \in \mathcal{R}$

- $(x_0, \ldots, x_{t-1}) \leftarrow \mathrm{rand}(\mathcal{R})$.
- $x_t = x - (x_0 + \cdots + x_{t-1})$.
- Secret-share x: $[x] := (x_0, \ldots, x_t)$.
- Masked $a \in \mathbb{R}$ can be approximated by $\frac{[a']}{C}$ with some $a', C \in \mathbb{Z}$

Computation on secret-shares

- **Linear operation** is easy! $z_i = x_i + y_i$
- **Non-linear operation** with masked polynomial multiplication gadget PolyMult

Protecting Mitaka from t-probing adversary: an overview

Arithmetic masking of $x \in \mathcal{R}$

- $(x_0, \ldots, x_{t-1}) \leftarrow \mathrm{rand}(\mathcal{R})$.
- $x_t = x - (x_0 + \cdots + x_{t-1})$.
- Secret-share x: $[x] := (x_0, \ldots, x_t)$.
- Masked $a \in \mathbb{R}$ can be approximated by $\frac{[a']}{C}$ with some $a', C \in \mathbb{Z}$

Computation on secret-shares

- **Linear operation** is easy! $z_i = x_i + y_i$
- **Non-linear operation** with masked polynomial multiplication gadget PolyMult

Precomputed values:

$$[\beta_i] := \left[\frac{\widetilde{\mathbf{b}}_i^*}{\langle \widetilde{\mathbf{b}}_i, \widetilde{\mathbf{b}}_i \rangle_{\mathcal{R}}}\right]$$

MaskHybrid$([\mathbf{B}], [\beta_1], [\beta_2], [s_1], [s_2], [\mathbf{c}])$:

$[\mathbf{v}_2] := [\mathbf{0}]$, $[\mathbf{c}_2] := [\mathbf{c}]$

for $i = 2$ to 1:

$[d_i] = \sum_{j=1}^{2}$ PolyMult$([c_{i,j}], [\beta_{i,j}])$

$[t_i] =$ MaskPeikert$(\mathbf{I}, [d_i], [s_i], r)$

$[\mathbf{v}_{i-1}] = [\mathbf{v}_i] +$ PolyMult$([t_i], [\mathbf{b}_i])$

$[\mathbf{c}_{i-1}] = [\mathbf{c}_i] -$ PolyMult$([t_i], [\mathbf{b}_i])$

Outputs Unmask$([\mathbf{v}_0])$

Protecting Mitaka from t-probing adversary: an overview

Arithmetic masking of $x \in \mathcal{R}$

- $(x_0, \dots, x_{t-1}) \leftarrow \mathrm{rand}(\mathcal{R})$.
- $x_t = x - (x_0 + \cdots + x_{t-1})$.
- Secret-share x: $[x] := (x_0, \dots, x_t)$.
- Masked $a \in \mathbb{R}$ can be approximated by $\frac{[a']}{C}$ with some $a', C \in \mathbb{Z}$

Computation on secret-shares

- **Linear operation** is easy! $z_i = x_i + y_i$
- **Non-linear operation** with masked polynomial multiplication gadget PolyMult

Precomputed values:

$$[\beta_i] := \left[\frac{\widetilde{\mathbf{b}}_i^*}{\langle \widetilde{\mathbf{b}}_i, \widetilde{\mathbf{b}}_i \rangle_{\mathcal{R}}}\right]$$

MaskHybrid$([\mathbf{B}], [\beta_1], [\beta_2], [s_1], [s_2], [\mathbf{c}])$:

$[\mathbf{v}_2] := [\mathbf{0}]$, $[\mathbf{c}_2] := [\mathbf{c}]$

for $i = 2$ to 1:

$[d_i] = \sum_{j=1}^{2}$ PolyMult$([c_{i,j}], [\beta_{i,j}])$

$[t_i] =$ MaskPeikert$(\mathbf{I}, [d_i], [s_i], r)$

$[\mathbf{v}_{i-1}] = [\mathbf{v}_i] +$ PolyMult$([t_i], [\mathbf{b}_i])$

$[\mathbf{c}_{i-1}] = [\mathbf{c}_i] -$ PolyMult$([t_i], [\mathbf{b}_i])$

Outputs Unmask$([\mathbf{v}_0])$

Signing operations outside the sampler are not sensitive!

New gadgets for masking

1) [Offline]

- Outputs **continuous Gaussian** samples in *arithmetically* masked form

New gadgets for masking

1) [Offline]

- Outputs **continuous Gaussian** samples in *arithmetically* masked form

2) [Online]

- sample **discrete Gaussians** share-by-share on each share c_i of $[c] = (c_0, \dots, c_t)$.
  Warning: security loss!

ShareByShareGauss$_r$([c]):

for $i = 0$ to t:

$z_i \leftarrow D_{\mathbb{Z}, c_i, r/\sqrt{t+1}}$

Outputs $(z_0, \dots, z_t)$

New gadgets for masking

1) [Offline]

- Outputs **continuous Gaussian** samples in *arithmetically* masked form

2) [Online]

- sample **discrete Gaussians** share-by-share on each share c_i of $[c] = (c_0, \ldots, c_t)$.
  Warning: security loss!
- a trade-off: rejection-sampling in larger lattice

ShareByShareGauss$_r([c])$:

for $i = 0$ to t:

$z_i \leftarrow D_{\frac{1}{B}\mathbb{Z}, c_i, r/\sqrt{t+1}}$

restart if $\sum\{z_i\} \neq 0$

Outputs $(z_0, \ldots, z_t)$

New gadgets for masking

1) [Offline]

- Outputs **continuous Gaussian** samples in *arithmetically* masked form

2) [Online]

- sample **discrete Gaussians** share-by-share on each share c_i of $[c] = (c_0, \ldots, c_t)$.
  Warning: security loss!
- a trade-off: rejection-sampling in larger lattice

3) [Polynomial multiplication]

- NTT/FFT on arithmetic shares (linear op.)
- Coordinate-wise multiplication with the standard **ISW multiplier**

ShareByShareGauss$_r([c])$:

for $i = 0$ to t:

$z_i \leftarrow D_{\frac{1}{B}\mathbb{Z}, c_i, r/\sqrt{t+1}}$

restart if $\sum\{z_i\} \neq 0$

Outputs $(z_0, \ldots, z_t)$

PolyMult$([a], [b])$:

$[\widehat{a}] = \mathsf{NTT}([a])$

$[\widehat{b}] = \mathsf{NTT}([b])$

for $j = 0$ to $d - 1$:

$[\widehat{c}_j] = \mathsf{Mult}([\widehat{a}_j], [\widehat{b}_j])$

$[c] := \mathsf{iNTT}([\widehat{c}_0], \ldots, [\widehat{c}_{d-1}])$

Outputs $[c]$

New gadgets for masking

1) [Offline]

- Outputs **continuous Gaussian** samples in *arithmetically* masked form

2) [Online]

- sample **discrete Gaussians** share-by-share on each share c_i of $[c] = (c_0, \ldots, c_t)$.
  Warning: security loss!
- a trade-off: rejection-sampling in larger lattice

3) [Polynomial multiplication]

- NTT/FFT on arithmetic shares (linear op.)
- Coordinate-wise multiplication with the standard **ISW multiplier**

ShareByShareGauss$_r([c])$**:**

for $i = 0$ to t:

$z_i \leftarrow D_{\frac{1}{B}\mathbb{Z}, c_i, r/\sqrt{t+1}}$

restart if $\sum\{z_i\} \neq 0$

Outputs $(z_0, \ldots, z_t)$

PolyMult$([a], [b])$**:**

$[\widehat{a}] = \mathsf{NTT}([a])$

$[\widehat{b}] = \mathsf{NTT}([b])$

for $j = 0$ to $d - 1$:

$[\widehat{c}_j] = \mathsf{Mult}([\widehat{a}_j], [\widehat{b}_j])$

$[c] := \mathsf{iNTT}([\widehat{c}_0], \ldots, [\widehat{c}_{d-1}])$

Outputs $[c]$

☺ **No boolean–arithmetic share conversion in the online phase**

About performances

Table 1: Signature by seconds

	Falcon	Mitaka	Ratio
$d = 512$	2800	6300	**2.25**
$d = 1024$	1400	3100	**2.21**

experiments done with a **non-masked & non constant-time** implementation$^{(*)}$ and reusing Falcon's C reference code (as submitted to NIST round 3)

$(*)$: both schemes can be made constant-time, see e.g. Howe et al., PQCrypto 2020

Practical perspectives and open problems

`Mitaka` can use **fixed-point arithmetic only** over cyclotomic 2-smooth

- actually implement it (and target "almost" hardware constraints!)

Challenge: the keygen is then (even more) involved since we need to avoid *continuous* perturbations in Peikert's.

- Can we extend the technique to other cyclotomic rings?
- How efficient can we complete the basis when there is no tower?
- How to maximize the efficiency of the "micro-sieving" without FFT?

- Are there other techniques/approaches to avoid FPA?

A peek in the verification algorithm

A message is as $\mathbf{c} = (0, \mathcal{H}(\text{msg}))$. Signature $\mathbf{s} = (s_1, s_2) \in \mathcal{L}_{\text{NTRU}}$, close to $\mathbf{c}$.

$\text{Verif}_a(\text{msg}, s_1 \in R^2)$:

1. $s_2 \leftarrow a s_1 - \mathcal{H}(\text{msg}) \bmod q$
2. If $\|(s_1, s_2) - \mathbf{c}\|$ too big, reject.
3. Accept.

We send only s_1, so let's **reduce its length**.

Fast thanks to NTT (so, q is **well-chosen**)

A peek in the verification algorithm

A message is as $\mathbf{c} = (0, \mathcal{H}(\mathrm{msg}))$. Signature $\mathbf{s} = (s_1, s_2) \in \mathcal{L}_{\mathsf{NTRU}}$, close to $\mathbf{c}$.

$\mathrm{Verif}_a(\mathrm{msg}, s_1 \in R^2)$:

1. $s_2 \leftarrow a s_1 - \mathcal{H}(\mathrm{msg}) \bmod q$
2. If $\|(s_1, s_2) - \mathbf{c}\|$ too big, reject.
3. Accept.

We send only s_1, so let's **reduce its length**.

Fast thanks to NTT (so, q is **well-chosen**)

High-level ideas:

- Unbalance (s_1, s_2) using **elliptic sampling**
- Trade a few bits of q for efficiency

Challenge and dangers:

- we need yet another keygen
- How to keep the security level?

Elliptic sampling?

Elliptic Gaussian = a spherical Gaussian for another (euclidean) norm.

Any euclidean norm is $\|\mathbf{x}\|_{\mathrm{Q}}^2 = \mathbf{x}^{\mathrm{t}}\mathrm{Q}\mathbf{x}$, with Q positive definite.

(then $\mathrm{Vol}_{\mathrm{Q}}(\mathcal{L})^2 = \det(\mathbf{B}^{\mathrm{t}}\mathrm{Q}\mathbf{B})$)

Elliptic sampling?

Elliptic Gaussian $=$ a spherical Gaussian for another (euclidean) norm.

Any euclidean norm is $\|\mathbf{x}\|_Q^2 = \mathbf{x}^t Q \mathbf{x}$, with Q positive definite.

(then $\mathrm{Vol}_Q(\mathcal{L})^2 = \det(\mathbf{B}^t Q \mathbf{B})$)

- Forgery hardness $\sim$ volume of the decoding cell $\Rightarrow$ keep volume by taking $Q = \begin{bmatrix} \gamma & 0 \\ 0 & \gamma^{-1} \end{bmatrix}$,

where $\gamma \in \mathbb{R}_+^*$ or with all positive embeddings.

- Adapting Klein/Hybrid sampler straightforward: orthogonalize for the form Q

Expected length of s_1 is now shorter by a factor $\sqrt{\gamma}$ compared to the previous case.

the total length stays the same (but for the other norm)

Distribution of secret keys

- the larger γ is, the smaller f is.
- the smaller q is, the smaller f, g, are.

Impact: there are regimes where

- f, g may be sampled below the smoothing parameter of $\mathbb{Z}^d$
- (f, g) may be very short, maybe even close to sparse ternary

Distribution of secret keys

- the larger γ is, the smaller f is.
- the smaller q is, the smaller f, g, are.

Impact: there are regimes where

- f, g may be sampled below the smoothing parameter of $\mathbb{Z}^d$
- (f, g) may be very short, maybe even close to sparse ternary

when this happens, sample it/them directly sparse ternary.

three regimes: small q, pure "twisted" gaussians, and mixed.

Distribution of secret keys

- the larger γ is, the smaller f is.
- the smaller q is, the smaller f, g, are.

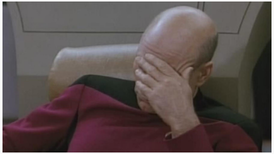
Impact: there are regimes where

- f, g may be sampled below the smoothing parameter of $\mathbb{Z}^d$
- (f, g) may be very short, maybe even close to sparse ternary

when this happens, sample it/them directly sparse ternary.

three regimes: small q, pure "twisted" gaussians, and mixed.

But now, key recovery attacks may be more powerful...



(me two nights before the deadline)

Quick sum-up of the concrete security analysis

On key-recovery:

1) we prove that the best case for the attacker is to find vectors short for $\|\cdot\|_Q$.
2) we adapt hybrid attacks using a geometric argument when f only is sparse
this attack is (mildly) better than the regular hybrid

Quick sum-up of the concrete security analysis

On key-recovery:

1) we prove that the best case for the attacker is to find vectors short for $\|\cdot\|_Q$.
2) we adapt hybrid attacks using a geometric argument when f only is sparse
3) we identify a new attack by (pure) lattice reduction

   if $k = (af - g)/q$, there are ranges for f, g where lattice reduction over $(a, -1, q)^\perp$ to recover (f, g, k) performs better than directly over $\mathcal{L}_{\mathsf{NTRU}}$ to recover (f, g).

Quick sum-up of the concrete security analysis

On key-recovery:

1) we prove that the best case for the attacker is to find vectors short for $\|\cdot\|_Q$.
2) we adapt hybrid attacks using a geometric argument when f only is sparse
3) we identify a new attack by (pure) lattice reduction

On forgery:

1) Smaller q or larger γ are somewhat equivalent for forgery
   Because q/γ^2 drives the security against forgery.

Quick sum-up of the concrete security analysis

On key-recovery:

1) we prove that the best case for the attacker is to find vectors short for $\|\cdot\|_Q$.
2) we adapt hybrid attacks using a geometric argument when f only is sparse
3) we identify a new attack by (pure) lattice reduction

On forgery:

1) Smaller q or larger γ are somewhat equivalent for forgery
2) we highlight that smaller q makes some known attacks performing better

   "forgetting vectors" is useful against ModFalcon, not against Falcon, but it becomes useful again when q decreases.

Quick sum-up of the concrete security analysis

On key-recovery:

1) we prove that the best case for the attacker is to find vectors short for $\|\cdot\|_Q$.
2) we adapt hybrid attacks using a geometric argument when f only is sparse
3) we identify a new attack by (pure) lattice reduction

On forgery:

1) Smaller q or larger γ are somewhat equivalent for forgery
2) we highlight that smaller q makes some known attacks performing better

Also, experimental confirmations of our heuristics, scripts to compute security levels.

Examples of new parameter sets

Parameters for and with $q = 257$ and ellipsoidal Gaussians with $\gamma = 8$. Classical security, size in bytes.

	Falcon–512			Mitaka–512		
	Security	|sig|	|pk|	Security	|sig|	|pk|
Original	123	666	896	102	710	896
Small $q = 257$	118	**425**	**576**	94	**475**	**576**
Ellipsoidal $\gamma = 8$	116	**410**	896	92	**460**	896

	Falcon–1024			Mitaka–1024		
	Security	|sig|	|pk|	Security	|sig|	|pk|
Original	272	1280	1792	233	1405	1792
Small $q = 257$	264	**805**	**1152**	209	**935**	**1152**
Ellipsoidal $\gamma = 8$	261	**780**	1792	204	**905**	1792

NB: these sizes take our generic compression technique into account ($7 - 15\%$ smaller $|\text{sig}|$).

Let's conclude

Further works for optimization:

- Generic Gaussian samplers and application to hash-and-sign (and more?)
- Improved/optimal keygens for lattice trapdoor sampling

Thank you!

Mitaka: a simpler, parallelizable, maskable variant of Falcon, eprint 2021/1486
Shorter hash-and-sign lattice-based signatures, eprint 2022/785