Mitaka

A Simpler, Parallelizable, Maskable Variant of Falcon

Thomas Espitau, Akira Takahashi,
Mehdi Tibouchi, Alexandre Wallet

NIST 3rd Workshop

Lattice signatures

Two finalists are based on structured lattices:

FALCON

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

✓ compact, fast

✗ restricted parameter set, quite hard to implement and protect against side-channels

CRYSTALS-DILITHIUM

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

✗ larger bandwdith

✓ large range of parameter sets, easier to implement and protect against side-channels

Two finalists are based on structured lattices:

FALCON	CRYSTALS-DILITHIUM
"Hash-and-sign" in lattices [GPV'08] + **NTRU trapdoors** [DLP'14]	**Fiat-Shamir "with abort"** [Lyu12] + module lattices

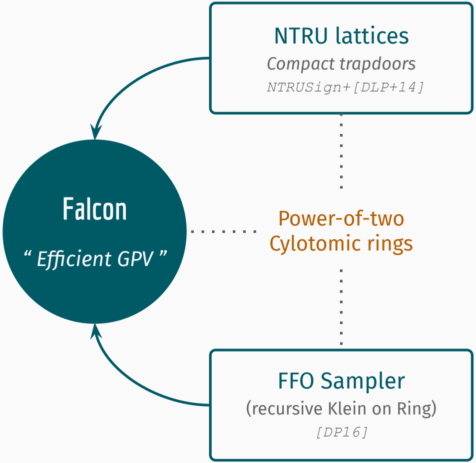
Introducing: [Mitaka]

trying to reach best of both worlds

✓ compact, fast

✓ large range of parameters sets

✓ easier to implement and protect against side-channels

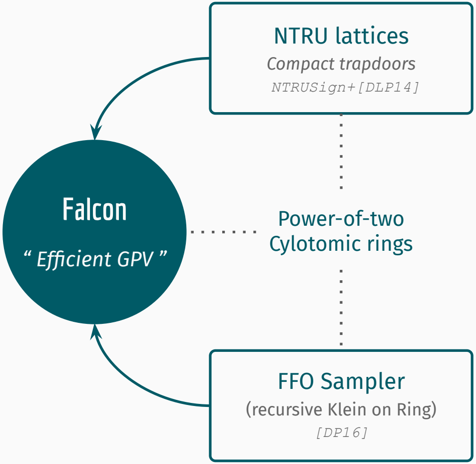


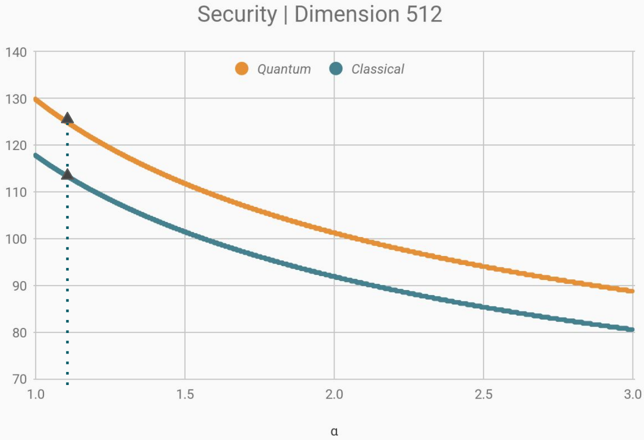
NTRU lattices: free rank 2 modules over cyclotomic rings

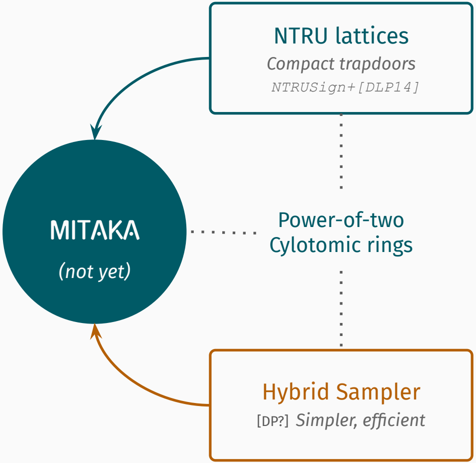
Quasi-linear **thanks to the ring** *but*

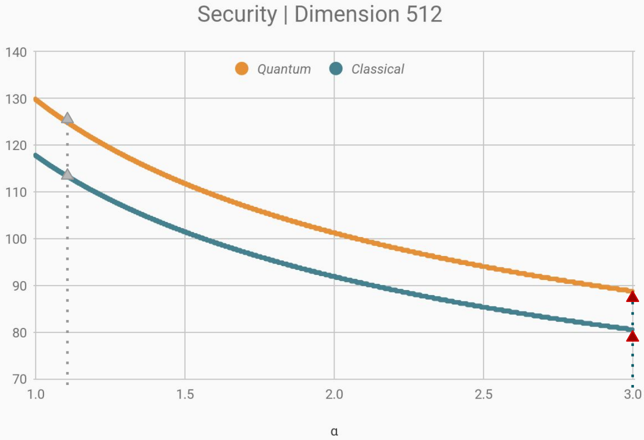
- Few parameter sets
- Complicated implementation
- Complicated masking

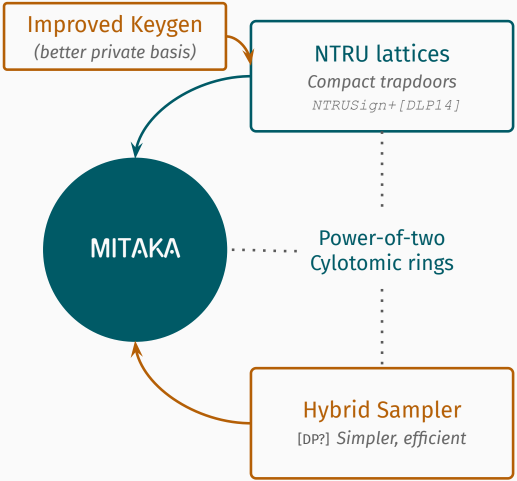
Towards Mitaka

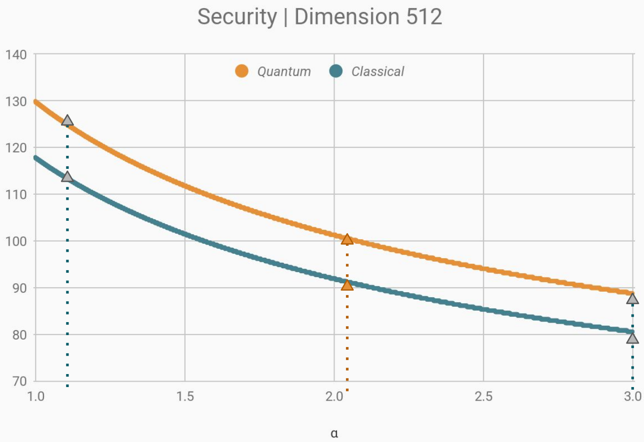


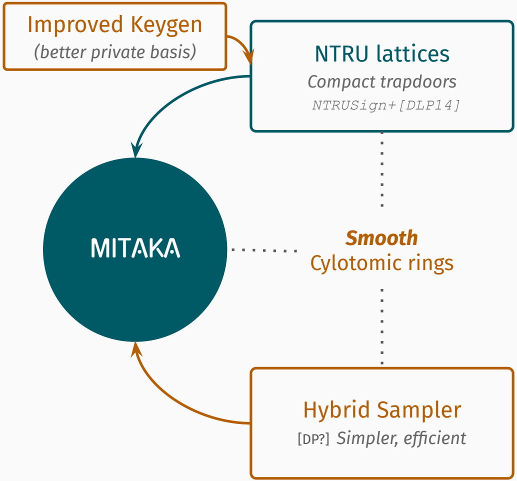


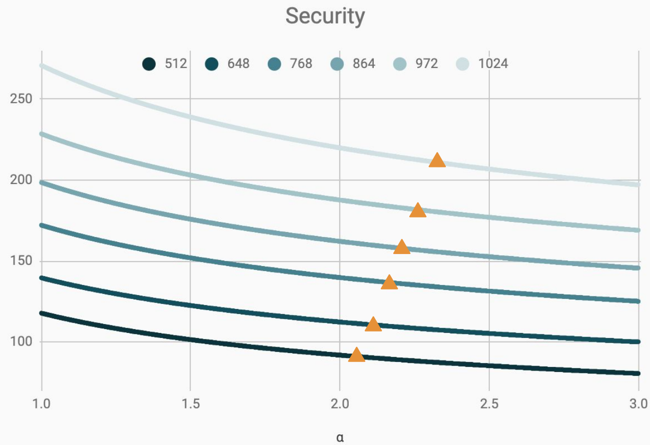

NTRU lattices
Compact trapdoors
NTRUSign+[DLP14]
MITAKA
(not yet)
Power-of-two
Cylotomic rings
Hybrid Sampler
[DP?] Simpler, efficient

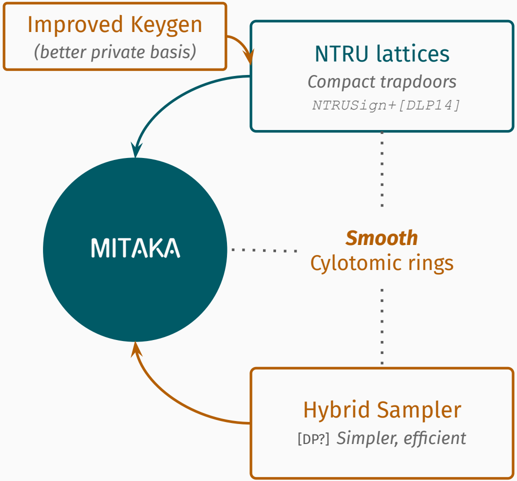

Security | Dimension 512
Quantum
Classical
140
130
120
110
100
90
80
70
1.0
1.5
2.0
2.5
3.0
α

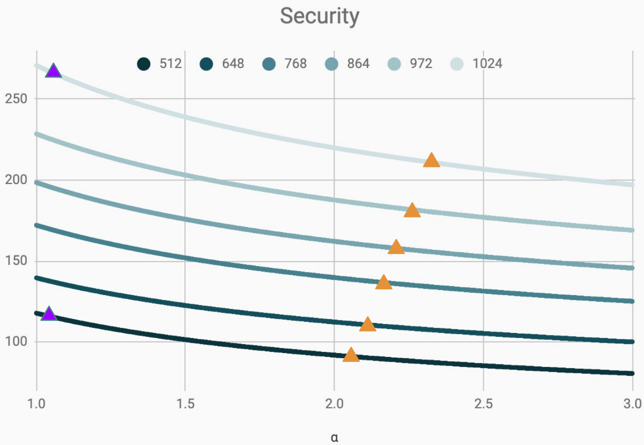

Improved Keygen
(better private basis)
NTRU lattices
Compact trapdoors
NTRUSign+[DLP14]
MITAKA
Power-of-two
Cylotomic rings
Hybrid Sampler
[DP?] Simpler, efficient


Security | Dimension 512
Quantum
Classical
140
130
120
110
100
90
80
70
1.0
1.5
2.0
2.5
3.0
α


Improved Keygen
(better private basis)
NTRU lattices
Compact trapdoors
NTRUSign+[DLP14]
MITAKA
Smooth
Cylotomic rings
Hybrid Sampler
[DP?] Simpler, efficient


Security
512
648
768
864
972
1024
250
200
150
100
1.0
1.5
2.0
2.5
3.0
α


Improved Keygen
(better private basis)
NTRU lattices
Compact trapdoors
NTRUSign+[DLP14]
MITAKA
Smooth
Cylotomic rings
Hybrid Sampler
[DP?] Simpler, efficient


Security
512
648
768
864
972
1024
250
200
150
100
1.0
1.5
2.0
2.5
3.0
α

Hash-and-sign over lattices

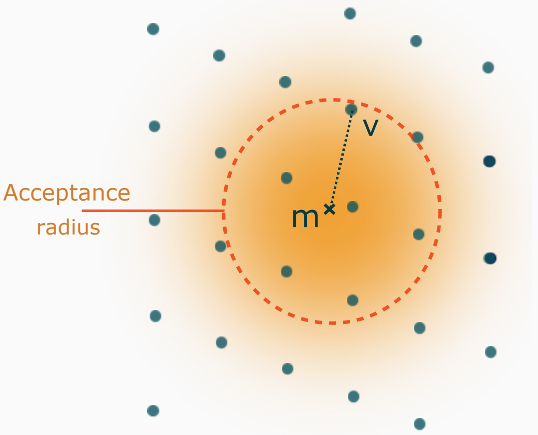
Simplified $\mathrm{Sign}_{\mathbf{sk},\sigma}(\mathrm{msg})$:

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

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

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

"Good basis" **sk** of $\mathcal{L}$, bad basis **pk**



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

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

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

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

Requirements

Hard Forgery $\Rightarrow$ σ small $\Rightarrow$ **sk** has short vectors

Hard to compute **sk** just from **pk**	Easy to generate **pk** just from **sk**

sk is called *"a trapdoor"*

Generating trapdoors is an interesting challenge

[HPSS'00, AP'09, MP'12, DLP'14, CGM'19, GL'20, CPSWX'20...]

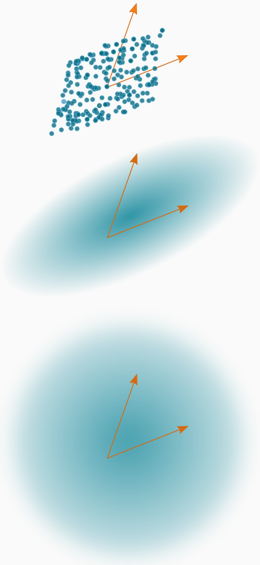
Sampling over (structured) lattices

Lattice Gaussian samplers = decoding + randomization

CVP solvers	**Gaussian samplers**
Babai's Round-off: $\mathbf{u} = \mathbf{B}\lceil \mathbf{B}^{-1}\mathbf{t} \rfloor$	Randomize the whole integer rounding
Babai's Nearest Plane: "adaptive" rounding on each $\mathbb{R}\widetilde{\mathbf{b}_i}$	Randomize each integer rounding

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

Without randomization *(not a Gaussian sampler)*

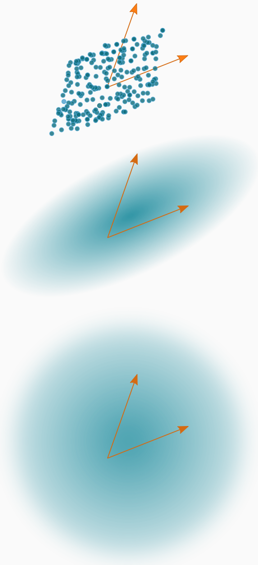


Outputs $\mathbf{z} = \mathbf{B}\lceil \mathbf{B}^{-1}\mathbf{t} \rfloor$

Without randomization *(not a Gaussian sampler)*

Randomize rounding w/ discrete Gaussians

(leaks the lattice basis)



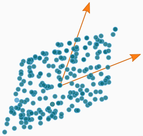
$\mathbf{y} \leftarrow \lceil \mathbf{B}^{-1}\mathbf{t} \rfloor_r$

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

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

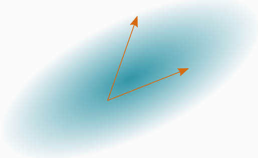
Randomized Babai Rounding : Peikert's approach

Without randomization *(not a Gaussian sampler)*



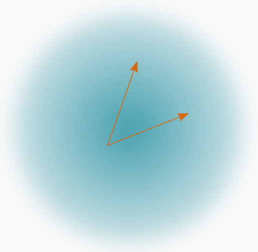
Randomize rounding w/ discrete Gaussians

(leaks the lattice basis)



[P'10] add Gaussian perturbation to *"smooth out"* the lattice

(works!)



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

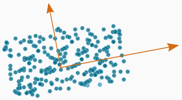
$\mathbf{x} \leftarrow \sigma \cdot \mathcal{N}(0, 1)$

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

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

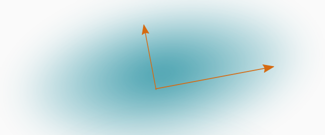
Without randomization

(not a Gaussian sampler)



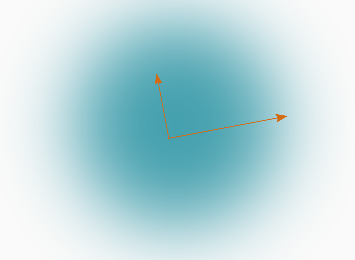
Randomize rounding of each $t_i \in \mathbb{R}$

(leaks Gram-Schmidt basis)



On each $\mathbb{R}\widetilde{\mathbf{b}}_i$, rescale adaptively

$$s_i := \frac{s}{\|\widetilde{\mathbf{b}}_i\|}$$



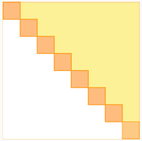
Klein$(\mathbf{B}, \widetilde{\mathbf{B}}, \mathbf{t}, s_i, r)$

- $\mathbf{v} = 0, \mathbf{c} = \mathbf{t}$
- **for** $i = \dim(\mathbf{B})$ **to** 1:
  - $t_i = \left\lceil \frac{\langle \mathbf{c}, \widetilde{\mathbf{b}}_i \rangle}{\|\widetilde{\mathbf{b}}_i\|^2} \right\rfloor_{s_i}$
  - $\mathbf{v} = \mathbf{v} + t_i \mathbf{b}_i$
  - $\mathbf{c} = \mathbf{c} - t_i \mathbf{b}_i$
- **Outputs** $\mathbf{v}$

Hybrid = Klein decoding + Peikert randomization in dim $\geqslant 1$.

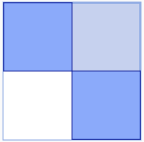
Well-suited for *module lattices*:
rank 2 $\mathcal{R}$-module = rank $2d$ lattice.

Klein



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

Hybrid



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

Example: $\mathcal{R}$ power-of-2 cyclotomic

Hybrid($\mathbf{B}$, $\widetilde{\mathbf{B}}_{\mathcal{R}}$, $\mathbf{t}$, s_1, s_2)

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

for $i = 2$ **to** 1:

$\quad t_i = \textbf{Peikert}\left(\mathbf{I}, \frac{\langle \mathbf{c}, \widetilde{\mathbf{b}}_i \rangle_{\mathcal{R}}}{\langle \widetilde{\mathbf{b}}_i, \widetilde{\mathbf{b}}_i \rangle_{\mathcal{R}}}, s_i, r\right)$

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

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

Outputs v

Operations in $\mathcal{R}$ instead of $\mathbb{Z}$ $\Rightarrow$ need "good FFT domain"

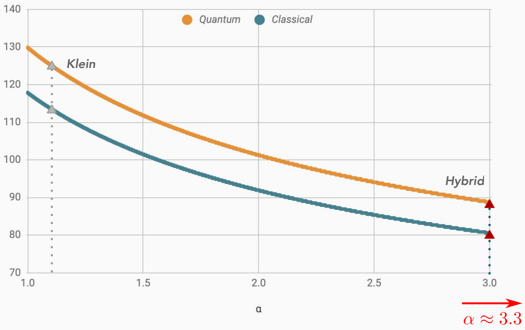
	Quality	Pros	Cons
Peikert	$s_1(\mathbf{B})$ (largest sing. value)	fast simple	worst quality (*lower security*)
Klein	$\max_i \lVert\widetilde{\mathbf{b}_i}\rVert$ (Gram-Schmidt)	best quality (*higher security*)	slower more involved
Hybrid	$s_1(\widetilde{\mathbf{B}})$	**Good tradeoffs when $\mathcal{R}$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

When $\mathcal{R} = \mathbb{Z}[x]/(x^d + 1)$, $d = 2^n$, and for NTRU q-ary lattices, qualities are $\alpha\sqrt{q}$

Asymptotic quality

Sampler	$\alpha\sqrt{q}$	Best achievable α
Peikert	$s_1(\mathbf{B})$	$O(d^{1/4}\sqrt{\log d})$
Hybrid	$s_1(\widetilde{\mathbf{B}})$	$O(d^{1/8}\log^{1/4} d)$
Klein	$\max_i \|\widetilde{\mathbf{b}_i}\|$	$O(1)$

Concrete bitsecurity as a function of α, $d = 512$



Improving the Keygen

NTRU lattice $\mathcal{L}_{\mathsf{NTRU}}(a)$

$$f, g \in \mathcal{R} \to a := f^{-1}g \ [q]$$

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

Trapdoor

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

$$\underbrace{\begin{bmatrix} f & g \\ ? & ? \end{bmatrix}}_{=B} \begin{bmatrix} a \\ -1 \end{bmatrix} = 0 \ [q]$$

NTRU lattice $\mathcal{L}_{\mathrm{NTRU}}(a)$

$$f, g \in \mathcal{R} \to a := f^{-1}g \ [q]$$

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

Trapdoor

Short basis **B** of $\mathcal{L}_{\mathrm{NTRU}}(a)$ with **good quality** wrt. a sampler.

$$\underbrace{\begin{bmatrix} f & g \\ ? & ? \end{bmatrix}}_{=B} \begin{bmatrix} a \\ -1 \end{bmatrix} = 0 \ [q]$$

Computing B

- Sample f, g Gaussians so that

$$\|(f, g)\| \approx \sqrt{q}$$

- Complete the basis: *unimodularity problem:* Euclid+geometry

Achieve good quality

Sample (f, g)'s until:

- `Falcon`: $\max(\|\widetilde{\mathbf{b}}_1\|, \|\widetilde{\mathbf{b}}_{d+1}\|) \approx 1.17\sqrt{q}$
- `Hybrid`: $s_1(\widetilde{\mathbf{B}})$ as close as possible to $\sqrt{q}$

*Both metrics can be computed **just with** f, g*

(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:* $\|\widetilde{\mathbf{b}}_{d+1}\| \leqslant 1.17\sqrt{q}$?
   else restart;
4) $\mathbf{b}_{d+1} \leftarrow \mathrm{NTRUSolve}(f, g, q)$;
   Compute all needed data;
   Output (pk, sk).

(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:* $\|\widetilde{\mathbf{b}}_{d+1}\| \leqslant 1.17\sqrt{q}$?
   else restart;
4) $\mathbf{b}_{d+1} \leftarrow \mathrm{NTRUSolve}(f, g, q)$;
   Compute all needed data;
   Output (pk, sk).

(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:* $\|\widetilde{\mathbf{b}}_{d+1}\| \leqslant 1.17\sqrt{q}$?
   else restart;

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

4) $\mathbf{b}_{d+1} \leftarrow \mathrm{NTRUSolve}(f, g, q)$;
   Compute all needed data;
   Output (pk, sk).

(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:* $\|\widetilde{\mathbf{b}}_{d+1}\| \leqslant 1.17\sqrt{q}$?

   else restart;

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

   else restart;

4) $\mathbf{b}_{d+1} \leftarrow \mathrm{NTRUSolve}(f, g, q)$;

   Compute all needed data;

   Output (pk, sk).

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

And randomness is expensive.

(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:* $\|\widetilde{\mathbf{b}}_{d+1}\| \leqslant 1.17\sqrt{q}$?
else restart;

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

4) $\mathbf{b}_{d+1} \leftarrow \mathrm{NTRUSolve}(f, g, q)$;
Compute all needed data;
Output (pk, sk).

Solution: *amortize* the rnd generation

+ Reuse randomness

+ Galois automorphisms

= *"Free"* blow-up of search-space

☺ better trapdoors in reasonable time

KeyGen (Std. dev. σ of f and g, number of samples m, n, set $\mathfrak{G}$ of Galois automorphisms)

1) [Sampling]

- Generate $2m$ Gaussians vectors of std. dev. $\sigma/\sqrt{2}$ and store them in two lists F', F''.
- Generate $2m$ Gaussians vectors of std. dev. $\sigma/\sqrt{2}$ and store them in two lists G', G''.

2) [Blowing up]

- Pair two lists $F \leftarrow F' + F'', G \leftarrow G' + G''$
- Let $\mathfrak{G}$ acts on G: $G \leftarrow \bigcup_{\sigma \in \mathfrak{G}} \sigma(G)$

KeyGen (Std. dev. σ of f and g, number of samples m, n, set $\mathfrak{G}$ of Galois automorphisms)

1) [Sampling]

- Generate $2m$ Gaussians vectors of std. dev. $\sigma/\sqrt{2}$ and store them in two lists F', F''.
- Generate $2m$ Gaussians vectors of std. dev. $\sigma/\sqrt{2}$ and store them in two lists G', G''.

2) [Blowing up]

- Pair two lists $F \leftarrow F' + F'', G \leftarrow G' + G''$
- Let $\mathfrak{G}$ acts on G: $G \leftarrow \bigcup_{\sigma \in \mathfrak{G}} \sigma(G)$

3) [Testing] **For** $f \in F, g \in G$ **do**
   If quality-testing(f, g)
      Output $(\text{pk}(f, g), \text{sk}(f, g))$.

KeyGen (Std. dev. σ of f and g, number of samples m, n, set $\mathfrak{G}$ of Galois automorphisms)

1) [Sampling]
   - Generate $2m$ Gaussians vectors of std. dev. $\sigma/\sqrt{2}$ and store them in two lists F', F''.
   - Generate $2m$ Gaussians vectors of std. dev. $\sigma/\sqrt{2}$ and store them in two lists G', G''.
2) [Blowing up]
   - Pair two lists $F \leftarrow F' + F''$, $G \leftarrow G' + G''$
   - Let $\mathfrak{G}$ acts on G: $G \leftarrow \bigcup_{\sigma \in \mathfrak{G}} \sigma(G)$
3) [Testing] **For** $f \in F, g \in G$ **do**

   If quality-testing(f, g)

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

Improved keygen

For the generation cost of $4m$ Gaussians, search a space of size

$$\mathrm{Card}(\mathfrak{G}) \cdot m^4$$

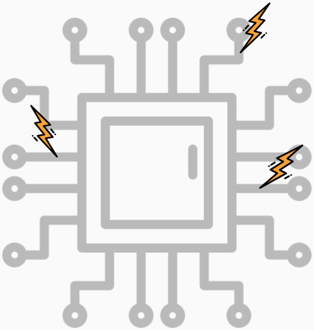
Masking Mitaka

t-probing attacker model [ISW03]

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

Provable security: t-probing security

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



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

- $(x_0, \dots, x_{t-1}) \leftarrow \mathrm{rand}(\mathcal{R})$.
- $x_t = x - (x_0 + \dots + x_{t-1})$.
- 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

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

Precompute

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

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 $x \in \mathcal{R}$

- $(x_0, \ldots, x_{t-1}) \leftarrow \mathrm{rand}(\mathcal{R})$.
- $x_t = x - (x_0 + \cdots + x_{t-1})$.
- 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

Precompute

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

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!

1) [Offline]

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

1) [Offline]

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

2) [Online]

- Generate **discrete Gaussian** samples share-by-share on each random share c_i of $[c] = (c_0, \ldots, c_t)$.

ShareByShareGauss$_r([c])$

for $i = 0$ **to** t:

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

Outputs $(z_0, \ldots, z_t)$

Masking Peikert sampler

1) [Offline]
   - Outputs **continuous Gaussian** samples in *arithmetically* masked form
2) [Online]
   - Generate **discrete Gaussian** samples share-by-share on each random share c_i of $[c] = (c_0, \ldots, c_t)$.
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:

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

Outputs $(z_0, \ldots, z_t)$

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

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

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

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

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

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

Outputs $[c]$

Masking Peikert sampler

1) [Offline]
   - Outputs **continuous Gaussian** samples in *arithmetically* masked form
2) [Online]
   - Generate **discrete Gaussian** samples share-by-share on each random share c_i of $[c] = (c_0, \ldots, c_t)$.
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:

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

Outputs $(z_0, \ldots, z_t)$

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

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

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

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

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

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

Outputs $[c]$

1) [Offline]
   - Outputs **continuous Gaussian** samples in *arithmetically* masked form
2) [Online]
   - Generate **discrete Gaussian** samples share-by-share on each random share c_i of $[c] = (c_0, \dots, c_t)$.
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:

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

Outputs $(z_0, \dots, z_t)$

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

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

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

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

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

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

Outputs $[c]$

☺ No boolean–arithmetic share conversion in the online phase

Wrapping-up

Improved Keygen
(better private basis)

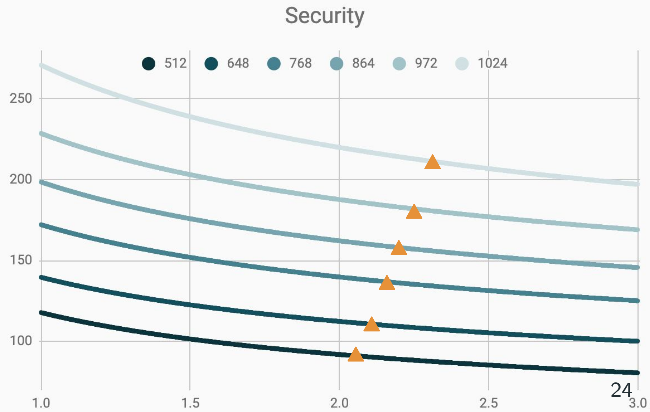
NTRU lattices
Compact trapdoors
NTRUSign+[DLP14]

MITAKA

Smooth
Cylotomic rings

Hybrid Sampler
[DP?] *Simpler, efficient*

Simple | Efficient | Compact | Versatile | Maskable




JB 01
三　鷹
三鷹
미타카
み た か
吉祥寺
JB 02
Kichijōji
Mitaka