Isochrony
Constant-time techniques for lattice-based signatures

Mélissa Rossi

Presentation based on

- [CCS’2019] joint work with Barthe, Belaïd, Espitau, Fouque and Tibouchi
- [PQCRYPTO’20] joint work with Howe, Prest and Ricosset
«Constant time» is a confusing term

Constant time does not mean constant execution time
Constant time does not mean constant execution time

« Constant time »

The execution time does not depend on the private key.

⇒ Not necessarily constant!
The execution time does not depend on the private key.

« Constant time »

The execution time does not depend on the private key.

Not necessarily constant!

Better say isochronous?

Howe et al. (2019/1411)
The execution time does not depend on the private key. 

« Constant time » is a confusing term

Constant time does not mean constant execution time

Better say isochronous?

Assumption

+ , − , × , / Constant time on integers

Howe et al. (2019/1411)
Gaussian Distributions and Lattice Based signatures

Fiat-Shamir with aborts Family
- BLISS
- Crystals-Dilithium
- qTesla

Hash and Sign Family
- GPV
- Falcon
Fiat-Shamir with aborts
Family
Fiat-Shamir with aborts
Family

Schnorr-like signatures with aborts

- Lyubashevsky (EC’12)
  Based on SIS, LWE or variants
Schnorr-like signatures with aborts

- Lyubashevsky (EC’12)
  Based on SIS, LWE or variants

**Public parameter:** $a$

**Secret key:** $s$ (short)

**Public key:** $t \leftarrow a \cdot s \mod q$
Schnorr-like signatures with aborts

- Lyubashevsky (EC’12)
  Based on SIS, LWE or variants

Public parameter: \(a\)
Secret key: \(s\) (short)
Public key: \(t \leftarrow a \cdot s \mod q\)

\(\text{Sign}(s, m):\)
1: do
2: \(y \leftarrow Y\)
3: \(c \leftarrow H(ay, m)\)
4: \(z \leftarrow c \cdot s + y\)
5: while Rejected\((z)\)
6: return \((z, c)\)
Gaussian Distributions and Lattice Based signatures

Fiat-Shamir with aborts Family

Schnorr-like signatures with aborts
  - Lyubashevsky (EC’12)
  
  Based on SIS, LWE or variants

Public parameter: \( a \)
Secret key: \( s \) (short)
Public key: \( t \leftarrow a \cdot s \mod q \)

\[
\text{Sign}(s, m):
\begin{align*}
1: & \text{do} \\
2: & y \leftarrow Y \\
3: & c \leftarrow H(ay, m) \\
4: & z \leftarrow c \cdot s + y \\
5: & \text{while Rejected}(z) \\
6: & \text{return } (z, c)
\end{align*}
\]

\[
\text{Verify}(z, c, t, m):
\begin{align*}
1: & v \leftarrow a \cdot z - c \cdot t \\
2: & \text{return } 1 \text{ if } c = H(v, m) \text{ and } z \text{ is small} \text{ else } 0
\end{align*}
\]
Gaussian Distributions and Lattice Based signatures

Fiat-Shamir with aborts
Family

Schnorr-like signatures with aborts

- Lyubashevsky (EC’12)
  Based on SIS, LWE or variants

Public parameter: \(a\)
Secret key: \(s\) (short)
Public key: \(t \leftarrow a \cdot s \mod q\)

**Sign\((s, m)\):**
1: do
2: \(y \leftarrow Y\)
3: \(c \leftarrow H(ay, m)\)
4: \(z \leftarrow c \cdot s + y\)
5: **while** Rejected\((z)\)
6: return \((z, c)\)

**Verify\((z, c, t, m)\):**
1: \(v \leftarrow a \cdot z - c \cdot t\)
2: return 1 if \(c = H(v, m)\) and \(z\) is small else 0

Why?
Rejection Sampling Lemma

- Lyubashevsky (EUROCRYPT'12)

Also called Acceptance-Rejection method

Going from an actual distribution $Y$ to an ideal target distribution $X$
Rejection Sampling Lemma

Also called Acceptance-Rejection method

Going from an actual distribution $Y$ to an ideal target distribution $X$

To sample from a distribution $X$, with density $f$, one uses samples from the distribution $Y$, with density $g$ as follows:

1. Get a sample $y$ from distribution $Y$
2. Accept $y$ as a sample drawn from $X$ with probability $\frac{f(y)}{M \cdot g(y)} = \frac{f(y)}{M \cdot \text{Actual}}$ target
   reject otherwise
Rejection Sampling Lemma

Also called Acceptance-Rejection method

Going from an actual distribution $Y$ to an ideal target distribution $X$

To sample from a distribution $X$, with density $f$, one uses samples from the distribution $Y$, with density $g$ as follows:

1. Get a sample $y$ from distribution $Y$
2. Accept $y$ as a sample drawn from $X$ with probability \[ \frac{f(y)}{M \cdot g(y)} = \text{Target} \]
   reject otherwise

statistically close to

\( (\text{if } M \cdot g(y) \geq f(y) \ \forall y) \)

1. Get a sample $x$ from distribution $X$
2. Accept $x$ with probability \[ \frac{1}{M}, \]
   reject otherwise

Lyubashevsky (EUROCRYPT'12)
Where are the Gaussian distributions in Fiat-Shamir with aborts signatures?

**Key generation:**

- Public parameter: $a$
- Secret key: $s$
- Public key: $t \leftarrow a \cdot s$

- Public parameter: $a$
- Secret key: $s, e$
- Public key: $t \leftarrow a \cdot s + e$

**Signature algorithm:**

1: do  
2: $y \leftarrow Y$  
3: $c \leftarrow H(ay, m)$  
4: $z \leftarrow c \cdot s + y$  
5: while $\text{Rejected}(z)$  
6: return $(z, c)$
Where are the Gaussian distributions in Fiat-Shamir with aborts signatures?

Key generation:

Public parameter: \(a\)
Secret key: \(s\)
Public key: \(t \leftarrow a \cdot s\)

Public parameter: \(a\)
Secret key: \(s, e\)
Public key: \(t \leftarrow a \cdot s + e\)

Signature algorithm:

1: do
2: \(y \leftarrow Y\)
3: \(c \leftarrow H(ay, m)\)
4: \(z \leftarrow c \cdot s + y\)
5: while Rejected(\(z\))
6: return \((z, c)\)
Gaussian Distributions and Lattice Based signatures 1

Where are the Gaussian distributions in Fiat-Shamir with aborts signatures?

Key generation:

Public parameter: \( a \)
Secret key: \( s \)
Public key: \( t \leftarrow a \cdot s \)

Public parameter: \( a \)
Secret key: \( s, e \)
Public key: \( t \leftarrow a \cdot s + e \)

Signature algorithm:

1: do
2: \( y \leftarrow Y \)
3: \( c \leftarrow H(ay, m) \)
4: \( z \leftarrow c \cdot s + y \)
5: while Rejected(\( z \))
6: return \( (z, c) \)

They were originally used for two reasons:

Security reductions
Performance
Gaussians lead to implementation vulnerabilities
Gaussians lead to implementation vulnerabilities

How to evaluate them?

by computing transcendental functions $\exp(.)$ and $\cosh(.)$
Gaussians lead to implementation vulnerabilities

➡ How to evaluate them?

by computing transcendental functions $\exp(.)$ and $\cosh(.)$
Gaussians lead to implementation vulnerabilities

How to evaluate them?
by computing transcendental functions $exp(.)$ and $cosh(.)$

How to sample?
Many possibilities, ex:
Large Cumulative Distribution Tables (CDT)
$u$ uniform in [0,1], recover $y$ with a table of the red curve
Gaussians lead to implementation vulnerabilities

How to evaluate them?
by computing transcendental functions \( \exp(\cdot) \) and \( \cosh(\cdot) \)

How to sample?
Many possibilities, ex:
Large Cumulative Distribution Tables (CDT)
\( u \) uniform in \([0,1]\), recover \( y \) with a table of the red curve
Binary search leaks \( y \) by memory access pattern
Gaussians lead to implementation vulnerabilities

⇒ How to evaluate them?
   by computing transcendental functions \( \exp(\cdot) \) and \( \cosh(\cdot) \)

⇒ How to sample?
   Many possibilities, ex :
   Large Cumulative Distribution Tables (CDT)
   \( u \) uniform in [0,1], recover \( y \) with a table of the red curve
   Binary search leaks \( y \) by memory access pattern
Gaussian distributions are hard to implement securely.

BLISS was the first practical implementation of a lattice based signature scheme.

BLISS
Fiat-Shamir with aborts
Bimodal Gaussians

» Ducas et al (CRYPTO’13)
Gaussian distributions are hard to implement securely.

BLISS was the first practical implementation of a lattice based signature scheme.

- Fiat-Shamir with aborts
- Bimodal Gaussians
- Ducas et al (CRYPTO’13)

Many side channel attacks targeting Gaussian distributions (timing)

- Groot Bruinderink et al. CHES’2016
- Espitau et al. SAC’2016
- Pessl et al. ACM-CCS'2017
- Espitau et al. ACM-CCS’2017
- Bootle et al. ASIACRYPT’2018
- Barthe et al. ACM-CCS’2019
Gaussian distributions are hard to implement securely.

BLISS was the first practical implementation of a lattice based signature scheme.

- BLISS
- Fiat-Shamir with aborts
- Bimodal Gaussians

Many side channel attacks targeting Gaussian distributions (timing)

- Groot Bruinderink et al. CHES’2016
- Espitau et al. SAC’2016
- Pessl et al. ACM-CCS’2017
- Espitau et al. ACM-CCS’2017
- Bootle et al. ASIACRYPT’2018
- Barthe et al. ACM-CCS’2019

New designs:

Gaussians are now avoided in Fiat-Shamir with aborts lattice signature schemes... but there is a price to pay in terms of performance

- Ducas et al. (CRYPTO’13)
Gaussian distributions are hard to implement securely

BLISS was the first practical implementation of a lattice based signature scheme.

- BLISS
  - Fiat-Shamir with aborts
  - Bimodal Gaussians

Many side channel attacks targeting Gaussian distributions (timing)

- Groot Bruinderink et al. CHES’2016
- Espitau et al. SAC’2016
- Pessl et al. ACM-CCS'2017
- Espitau et al. ACM-CCS’2017
- Bootle et al. ASIACRYPT’2018
- Barthe et al. ACM-CCS’2019

New designs:

Gaussians are now avoided in Fiat-Shamir with aborts lattice signature schemes... but there is a price to pay in terms of performance

- Crystals-Dilithium: Only uniform distributions
  - Ducas et al (NIST-PQC’17)
- qTesla: Gaussian sampling only in the keygen
  - Bindel et al (NIST-PQC’17)
Gaussian Distributions and Lattice Based signatures 2

- Fiat-Shamir with abort Signatures
- Hash and Sign Signatures
Gaussian Distributions and Lattice Based signatures 2

Hash and Sign
Signatures

- Gentry, Peikert and Vaikuntanathan (STOC'08)
Gaussian Distributions and Lattice Based signatures 2

Hash and Sign Signatures

- Gentry, Peikert and Vaikuntanathan (STOC’08)

**KeyGen()**

1: Generate matrices A, B
   - such that \( BA = 0 \)
   - B has small coefficients

2: \( pk \leftarrow A \)

3: \( sk \leftarrow B \)
Gaussian Distributions and Lattice Based signatures

Hash and Sign Signatures

- Gentry, Peikert and Vaikuntanathan (STOC'08)

KeyGen()  
1: Generate matrices $A, B$
   such that $BA = 0$
   $B$ has small coefficients
2: $pk \leftarrow A$
3: $sk \leftarrow B$

Sign$(m, sk)$
1: Compute $c$ such that $cA = H(m)$
2: $v \leftarrow$ a vector in $\Lambda(B)$ close to $c$
3: $s \leftarrow c - v$
Gaussian Distributions and Lattice Based signatures 2

Hash and Sign
Signatures

- Gentry, Peikert and Vaikuntanathan (STOC’08)

**KeyGen()**

1: Generate matrices $A, B$
   such that
   \[
   BA = 0
   \]
   $B$ has small coefficients
2: $pk \leftarrow A$
3: $sk \leftarrow B$

**Sign(m,sk)**

1: Compute $c$ such that $cA = H(m)$
2: $v \leftarrow$ a vector in $\Lambda(B)$ close to $c$
3: $s \leftarrow c - v$

**Verify(m,pk,s)**

Accept iff:

\[
\begin{align*}
\text{s is short} \\
\text{sA} = H(m)
\end{align*}
\]
KeyGen()
1: Generate matrices A, B such that \( BA = 0 \) and B has small coefficients
2: \( pk \leftarrow A \)
3: \( sk \leftarrow B \)

Sign(m,sk)
1: Compute c such that \( cA = H(m) \)
2: \( v \leftarrow \) a vector in \( \Lambda(B) \) close to c
3: \( s \leftarrow c - v \)

Verify(m,pk,s)
Accept iff:
\[
\begin{align*}
\{ & s \text{ is short} \\
& sA = H(m) 
\}
\]
Gaussian distributions are not « implementation friendly »

Side channel attacks targeting the Gaussian rejection sampling

- Espitau et al. SAC’2016
- Fouque et al EUROCRYPT’2019
Main takeaway of this presentation

Gaussian distributions can actually be

- ✔ Simpler to implement
- ✔ Provably resistant to timing attacks
Main takeaway of this presentation

Gaussian distributions can actually be

- ✔ Simpler to implement
- ✔ Provably resistant to timing attacks

1. Galactics: a polynomial approximation tool for Gaussians
2. Isochronous BLISS
3. Isochronous FALCON
Approximation tool

A tool to use polynomials instead of (bimodal) Gaussians

- Using Renyi divergence and Sobolev spaces
- Proof of concept developed in sage8.3
- Polynomials have integer coefficients (Lattice reduction)
Take two cryptographic schemes
- One with distribution $\mathcal{D}$
- One with an approximate distribution $\mathcal{D}'$ with the same support

For keeping the ‘same’ bit security for both schemes with at most $2^{64}$ signature queries (NIST suggestion),

$$\left\| 1 - \frac{\mathcal{D}'}{\mathcal{D}} \right\|_\infty \leq 2^{-45}.$$
Renyi divergence result

- Prest ASIACRYPT’17

Take two cryptographic schemes
- One with distribution $\mathcal{D}$
- One with an approximate distribution $\mathcal{D}'$ with the same support

For keeping the ‘same’ bit security for both schemes with at most $2^{64}$ signature queries (NIST suggestion),

$$\left\| 1 - \frac{\mathcal{D}'}{\mathcal{D}} \right\|_\infty \leq 2^{-45} \cdot \sqrt{\frac{2 \cdot \lambda}{2 \cdot (2 \cdot \lambda + 1)^2 \cdot Q_{\mathcal{D}'}}}$$
Renyi divergence result

Take two cryptographic schemes
- One with distribution $\mathcal{D}$
- One with an approximate distribution $\mathcal{D}'$ with the same support

For keeping the ‘same’ bit security for both schemes with at most $2^{64}$ signature queries (NIST suggestion),

$$\left\| 1 - \frac{\mathcal{D}'}{\mathcal{D}} \right\|_\infty \leq 2^{-45}. \sqrt{\frac{2 \cdot \lambda}{2 \cdot (2 \cdot \lambda + 1)^2 \cdot Q_{ss}}}$$

$\exp(\cdot)$ $\cosh(\cdot)$

Goal: $\mathcal{D} =$ (a transcendental function on an interval)

$\mathcal{D}' =$ a polynomial
Other Polynomial Approximations

How to choose $\mathcal{D}'$?

- **Taylor development**
  
  Not precise enough

\[
\mathcal{D}'(x) = \mathcal{D}(0) + \mathcal{D}'(0) \cdot x + \cdots + \frac{\mathcal{D}^{(d)}(0)}{d!} \cdot x^d
\]
Other Polynomial Approximations

How to choose $\mathcal{D}'$?

- **Taylor development**
  
  Not precise enough

- **Padé approximants** (rational function approximation)
  - Prest ASIACRYPT’17

  Two polynomials, higher degrees

$$
\mathcal{D}'(x) = \mathcal{D}(0) + \mathcal{D}^{(1)}(0) \cdot x + \cdots + \frac{\mathcal{D}^{(d)}(0)}{d!} \cdot x^d
$$

$$
\mathcal{D}'(x) = \frac{P(x)}{Q(x)}
$$
Other Polynomial Approximations

How to choose $\mathcal{D}'$ ?

• Taylor development
  Not precise enough

• Padé approximants (rational function approximation)
  › Prest ASIACRYPT’17

  Two polynomials, higher degrees

• Minimax computations : Sollya software package
  › Brisebarre and Chevillard IEEE’07
  › Chevillard, Joldes and Lauter ICMS’10
  › Zhao, Steinfeld and Sakzad 2018/1234

Floating point arithmetics

$$\mathcal{D}'(x) = \mathcal{D}(0) + \mathcal{D}^{(1)}(0) \cdot x + \cdots + \frac{\mathcal{D}^{(d)}(0)}{d!} \cdot x^d$$

$$\mathcal{D}'(x) = \frac{P(x)}{Q(x)}$$

$$\mathcal{D}' = \arg \min_{\deg(P) \leq d} \left( \sup_{x \in I} \left( 1 - \frac{P(x)}{\mathcal{D}(x)} \right) \right)$$
Our alternative Polynomial Approximation

How to choose $\mathcal{D}'$?

\[
\exp(\cdot) \quad \cosh(\cdot)
\]
Our alternative Polynomial Approximation

How to choose $D'$?

Transcendent functions $\exp(\cdot)$, $\cosh(\cdot)$ → Degree $d$ polynomial in $\mathbb{R}[x]$ → Degree $d$ polynomial in $\mathbb{Z}[x]$ with $\eta$ bit coefficients
Towards a polynomial approximation

We want
\[
\left\| 1 - \frac{\mathcal{D}'}{\mathcal{D}} \right\|_\infty \leq 2^{-45} \text{ and } \mathcal{D}' \in \mathbb{Z}[x].
\]

It corresponds to finding the closest element of 1 in $\mathbb{Z}[x]/\mathcal{D}(x)$ for the $\| \cdot \|_\infty$ norm.
Towards a polynomial approximation

We want \[ \left\| 1 - \frac{\mathcal{D}'}{\mathcal{D}} \right\|_\infty \leq 2^{-45} \] and \( \mathcal{D}' \in \mathbb{Z}[x] \).

It corresponds to finding the closest element of 1 in \( \mathbb{Z}[x]/\mathcal{D}(x) \) for the \( \| \cdot \|_\infty \) norm.

In a first attempt, let us look for the closest element of 1 in \( \mathbb{R}[x]/\mathcal{D}(x) \) for the \( \| \cdot \|_\infty \) norm.
Towards a polynomial approximation

We want
\[
\left\| 1 - \frac{D'}{D} \right\|_{\infty} \leq 2^{-45} \text{ and } D' \in \mathbb{Z}[x].
\]

It corresponds to finding the closest element of 1 in \(\mathbb{Z}[x]/D(x)\) for the \(\| \cdot \|_{\infty}\) norm.

In a first attempt, let us look for the closest element of 1 in \(\mathbb{R}[x]/D(x)\) for the \(\| \cdot \|_{\infty}\) norm.

An orthogonal projection could be great.
Towards a polynomial approximation

We want \[ \left\| 1 - \frac{D'}{D} \right\|_\infty \leq 2^{-45} \text{ and } D' \in \mathbb{Z}[x]. \]

It corresponds to finding the closest element of 1 in \( \mathbb{Z}[x]/D(x) \) for the \( \| \cdot \|_\infty \) norm.

In a first attempt, let us look for the closest element of 1 in \( \mathbb{R}[x]/D(x) \) for the \( \| \cdot \|_\infty \) norm.

An orthogonal projection could be great.

The \( \| \cdot \|_\infty \) norm is not Euclidean, no easy projection on a subspace.
Towards a polynomial approximation

We want \( \left\| 1 - \frac{\mathcal{D}'}{\mathcal{D}} \right\|_\infty \leq 2^{-45} \) and \( \mathcal{D}' \in \mathbb{Z}[x] \).

It corresponds to finding the closest element of 1 in \( \mathbb{Z}[x]/\mathcal{D}(x) \) for the \( \| \cdot \|_\infty \) norm.

In a first attempt, let us look for the closest element of 1 in \( \mathbb{R}[x]/\mathcal{D}(x) \) for the \( \| \cdot \|_\infty \) norm.

An orthogonal projection could be great.

The \( \| \cdot \|_\infty \) norm is not Euclidean, no easy projection on a subspace.

Over-approximate \( \| \cdot \|_\infty \) by a Euclidean norm.
When functional analysis meets cryptography

**Sobolev $H^2$ norm**

For $u$ and $v$ differentiable functions on $I$:

$$\langle u, v \rangle = \frac{1}{|I|} \int_I uv + |I| \int_I u'v'$$

Corresponding norm:

$$|u|^2_S = \frac{1}{|I|} \int_I u^2 + |I| \int_I u'^2$$

Equivalence with $\| \cdot \|_\infty$:

$$\|u\|_\infty \leq \sqrt{2} \cdot |u|_S$$
When functional analysis meets cryptography

Sobolev $H^2$ norm

For $u$ and $v$ differentiable functions on $I$:

$$\langle u, v \rangle = \frac{1}{|I|} \int_I uv + |I| \int_I u'v'$$

Corresponding norm:

$$|u|^2_S = \frac{1}{|I|} \int_I u^2 + |I| \int_I u'^2$$

Equivalence with $\| \cdot \|_\infty$:

$$\|u\|_\infty \leq \sqrt{2} \cdot |u|^1_S$$

Let us look for the closest element of 1 in $\mathbb{R}[x]/\mathcal{D}(x)$ for the $| \cdot |_S$ norm.
Proof of concept (Available on Github at https://github.com/espitau/GALACTICS)
Proof of concept (Available on Github at https://github.com/espitau/GALACTICS)
Galactics: a polynomial approximation tool for Gaussians

Isochronous BLISS

Isochronous FALCON
BLISS Signature Scheme

- Ducas et al (CRYPTO’13)

Elements are polynomials in $\mathcal{R} = \frac{\mathbb{Z}[x]}{x^n + 1}$

**Sign** ($m, pk = \mathbf{a}, sk = S = (s_1, s_2)$):

1. $y_1, y_2 \leftarrow \mathcal{D}_{\mathbb{Z}_n, \sigma}$
2. $u \leftarrow \zeta \cdot \mathbf{a} \cdot y_1 + y_2$
3. $c \leftarrow H(\lceil u \rceil_d \mod p, m)$
4. choose a random bit $b$
5. $z_1 \leftarrow (-1)^b cs_1 + y_1$
6. $z_2 \leftarrow (-1)^b cs_2 + y_2$
7. restart except wp

$$\frac{1}{M \exp \left( -\frac{\|S c\|^2}{2\sigma^2} \right) \cosh \left( \frac{\langle z, Sc \rangle}{\sigma^2} \right)}$$
8. return $(z_1, z_2, c)$

We polynomially approximate $\exp(\ )$ and $\cosh(\ )$ using the GALACTICS tool.

⇒ Also used for the Gaussian Sampling
BLISS Signature Scheme

- Ducas et al (CRYPTO’13)

Elements are polynomials in $\mathcal{R} = \frac{\mathbb{Z}[x]}{x^n + 1}$

**Sign** $(m, pk = a, sk = S = (s_1, s_2))$:

1. $y_1, y_2 \leftarrow \mathcal{D}_{\mathbb{Z}^n, \sigma}$
2. $u \leftarrow \zeta \cdot a \cdot y_1 + y_2$
3. $c \leftarrow H([u]_d \mod p, m)$
4. choose a random bit $b$
5. $z_1 \leftarrow (-1)^b cs_1 + y_1$
6. $z_2 \leftarrow (-1)^b cs_2 + y_2$
7. restart except wp
   
   $$
   M \exp\left(-\frac{\|Sc\|^2}{2\sigma^2}\right) \cosh\left(\frac{\langle z, Sc \rangle}{\sigma^2}\right)
   $$

8. return $(z_1, z_2, c)$

We polynomially approximate $\exp(\cdot)$ and $\cosh(\cdot)$ using the GALACTICS tool.

⇒ Also used for the Gaussian Sampling
BLISS Signature Scheme

- Ducas et al (CRYPTO’13)

Elements are polynomials in $\mathcal{R} = \frac{\mathbb{Z}[x]}{x^n + 1}$

**Sign** ($m, pk = a, sk = S = (s_1, s_2)$):

1. $y_1, y_2 \leftarrow \mathcal{D}_{\mathbb{Z}^n, \sigma}$
2. $u \leftarrow \zeta \cdot a \cdot y_1 + y_2$
3. $c \leftarrow H([u]_d \mod p, m)$
4. choose a random bit $b$
5. $z_1 \leftarrow (-1)^b cs_1 + y_1$
6. $z_2 \leftarrow (-1)^b cs_2 + y_2$
7. restart except wp
   
   $\frac{1}{M \exp \left(-\frac{\|Sc\|^2}{2\sigma^2}\right) \cosh \left(\frac{\langle z, Sc \rangle}{\sigma^2}\right)}$

8. return $(z_1, z_2, c)$

We polynomially approximate $\exp(\ )$ and $\cosh(\ )$ using the GALACTICS tool.

Also used for the Gaussian Sampling
Performance (in kcycles)

Recall that:
- and they are more conservative than BLISS

<table>
<thead>
<tr>
<th>Method</th>
<th>Number of kilocycles for one signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original BLISS</td>
<td>150</td>
</tr>
<tr>
<td>Our isochronous BLISS</td>
<td>300</td>
</tr>
<tr>
<td>Dilithium ref</td>
<td>450</td>
</tr>
<tr>
<td>Dilithium avx2</td>
<td>600</td>
</tr>
<tr>
<td>qTesla-I ref</td>
<td>20</td>
</tr>
</tbody>
</table>
Performance (in kcycles)

Number of kilocycles for one signature

The performance loss for isochrony is small
Recall that:

Crystals-Dilithium: Only uniform distributions
qTesla: Gaussian sampling only in the keygen
and they are more conservative than BLISS.
Roadmap

1. Galactics: a polynomial approximation tool for Gaussians
2. Isochronous BLISS
3. Isochronous FALCON
Falcon Gaussian sampler

**Algorithm SampleZ(σ, μ)**

Require: \( μ ∈ [0,1), σ ≤ σ_0 \)
Ensure: \( z ∼ D_{\mathbb{Z},σ,μ} \)

1. \( z_0 ← \text{Basesampler()} \)
2. \( b ← \{0,1\} \text{ uniformly} \)
3. \( z ← (2b - 1) \cdot z_0 + b \)
4. \( x ← -\frac{(z - μ)^2}{2σ^2} + \frac{z_0^2}{2σ_0^2} \)
5. Accept with probability \( \exp(x) \)
   Restart to 1. otherwise
Falcon Gaussian sampler

Algorithm SampleZ(\(\sigma, \mu\))

Require: \(\mu \in [0,1), \sigma \leq \sigma_0\)
Ensure: \(z \sim D_{\mathbb{Z}, \sigma, \mu}\)

1. \(z_0 \leftarrow \text{Basesampler()}\)
2. \(b \leftarrow \{0,1\} \text{ uniformly}\)
3. \(z \leftarrow (2b - 1) \cdot z_0 + b\)
4. \(x \leftarrow - \frac{(z - \mu)^2}{2\sigma^2} + \frac{z_0^2}{2\sigma_0^2}\)
5. Accept with probability \(\exp(x)\)
   Restart to 1. otherwise
Algorithm SampleZ$(\sigma, \mu)$

Require: $\mu \in [0,1), \sigma \leq \sigma_0$

Ensure: $z \sim D_{\mathbb{Z}, \sigma, \mu}$

1. $z_0 \leftarrow$ Basesampler()
2. $b \leftarrow \{0,1\}$ uniformly
3. $z \leftarrow (2b - 1) \cdot z_0 + b$
4. $x \leftarrow -\frac{(z - \mu)^2}{2\sigma^2} + \frac{z^2}{2\sigma_0^2}$
5. Accept with probability $\exp(x)$
   Restart to 1. otherwise

1. $\sigma_0$
3. $b = 0, \; b = 1$
Falcon Gaussian sampler

Algorithm SampleZ(σ, μ)

Require: μ ∈ [0,1), σ ≤ σ₀
Ensure: z ~ D_{Z,σ,μ}

1. z₀ ← Basesampler()
2. b ← {0,1} uniformly
3. z ← (2b − 1) · z₀ + b
4. x ← − \frac{(z − μ)^2}{2σ^2} + \frac{z_0^2}{2σ_0^2}
5. Accept with probability \exp(x)
   Restart to 1. otherwise

\[
P_{\text{accept}} = \frac{\exp \left( -\frac{(z − μ)^2}{2σ^2} \right)}{\exp \left( -\frac{z_0^2}{2σ_0^2} \right)}
\]
Algorithm SampleZ(\(\sigma, \mu\))

Require: \(\mu \in [0,1), \sigma \leq \sigma_0\)
Ensure: \(z \sim D_{z,\sigma,\mu}\)

1. \(z_0 \leftarrow \text{Basesampler}()\)
2. \(b \leftarrow \{0,1\} \text{ uniformly}\)
3. \(z \leftarrow (2b - 1) \cdot z_0 + b\)
4. \(x \leftarrow -\frac{(z - \mu)^2}{2\sigma^2} + \frac{z_0^2}{2\sigma_0^2}\)
5. Accept with probability \(\exp(x)\)
   Restart to 1. otherwise
**Algorithm SampleZ(σ, μ)**

Require: μ ∈ [0, 1), σ ≤ σ₀  
Ensure: z ∼ D_{z, σ, μ}

1. \( z_0 \leftarrow \text{Basesampler}() \)
2. \( b \leftarrow \{0, 1\} \) uniformly
3. \( z \leftarrow (2b - 1) \cdot z_0 + b \)
4. \( x \leftarrow -\frac{(z - μ)^2}{2σ^2} + \frac{z_0^2}{2σ_0^2} \)
5. Accept with probability \( \exp(x) \)

Restart to 1. otherwise

---

 Isochrony and portability modifications

(1) Basesampler with a CDT  
(2) Polynomial approximation for \( \exp \)  
(3) Make the number of iterations independent from the secret
Implementations

Number of sig computed in one second

Ref round 1 (not isochronous) vs Our PoC (isochronous)

- N = 512
- N = 1024

See https://falcon-sign.info/falcon-impl-20190802.pdf
Implementations

Number of sig computed in one second

- Ref round 1 (not isochronous)
- Our PoC (isochronous)

The performance loss for isochrony and portability do not change the ranking of Falcon

https://github.com/PQClean/PQClean/

https://falcon-sign.info/falcon-impl-20190802.pdf
Implementations

The performance loss for isochrony and portability do not change the ranking of Falcon

Recent implementations done by Thomas Pornin

See https://github.com/PQClean/PQClean/
And https://falcon-sign.info/falcon-impl-20190802.pdf
## Implementations

Constant time and integers help Cortex M4 implementations

<table>
<thead>
<tr>
<th>Falcon-512 (168 MHz)</th>
<th>Dynamic signatures (in milliseconds)</th>
<th>Memory (in bytes of extra RAM, not counting the key)</th>
</tr>
</thead>
<tbody>
<tr>
<td>First M4 implementation (Oder et al. PQCRYPTO 2019)</td>
<td>479</td>
<td>50508</td>
</tr>
<tr>
<td>Recent Constant time and integers (Thomas Pornin) <a href="https://github.com/mupq/pqm4">https://github.com/mupq/pqm4</a></td>
<td>243</td>
<td>36864</td>
</tr>
</tbody>
</table>
GALACTICS:

From transcendant functions to polynomials with integer coefficients

+ Simple rejection techniques for diminishing the CDT sizes
GALACTICS:

From transcendant functions to polynomials with integer coefficients

+ Simple rejection techniques for diminishing the CDT sizes

Helps to make BLISS isochronous and portable
Helps to make FALCON isochronous and portable
Conclusion

GALACTICS:

From transcendant functions to polynomials with integer coefficients

+ Simple rejection techniques for diminishing the CDT sizes

Helps to make BLISS isochronous and portable
Helps to make FALCON isochronous and portable

Gaussian distributions can now be

☑ Simpler to implement — portability
☑ Provably resistant to timing attacks
**Conclusion**

**GALACTICS:**

- From transcendant functions to polynomials with integer coefficients
  
  + Simple rejection techniques for diminishing the CDT sizes

Helps to make BLISS isochronous and portable

Helps to make FALCON isochronous and portable

Gaussian distributions can now be

- Simpler to implement — portability
- Provably resistant to timing attacks

**Perspectives**

Also helps for the **masking** countermeasure:

- ✓ See our CCS paper for the application to BLISS (without implementation)
- □ Ongoing work for Falcon