Masking Floating-Point Number Multiplication and Addition of Falcon

Keng-Yu Chen, Jiun-Peng Chen

October 16th, 2024

IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2024, No. 2, pp. 276–303. DOI:10.46586/tches.v2024.i2.276-303

Masking Floating-Point Number Multiplication and Addition of Falcon

First- and Higher-order Implementations and Evaluations

Keng-Yu Chen[1] and Jiun-Peng Chen[1,2]

[1] National Taiwan University, Taipei, Taiwan, r11921066@ntu.edu.tw
[2] Academia Sinica, Taipei, Taiwan, jpchen@ieee.org

Accepcted by Cryptographic Hardware and Embedded Systems (CHES) 2024

Table of Contents

Table of Contents

Introduction

- To defend the potential threat from large-scale quantum computers, the US National Institute of Standards and Technology (NIST) initiated standardization process for post-quantum cryptography in 2016.

Introduction

- To defend the potential threat from large-scale quantum computers, the US National Institute of Standards and Technology (NIST) initiated standardization process for post-quantum cryptography in 2016.
- In 2022, four selected algorithms – CRYSTALS-Kyber, CRYSTALS-Dilithium, FALCON, and SPHINCS+ were expected to be part of NIST's post-quantum cryptographic standards.

Theoretical and Real-World Security

In theory, these algorithms can base their security on problems that are considered still hard given the advantage of quantum computing.

Theoretical and Real-World Security

In theory, these algorithms can base their security on problems that are considered still hard given the advantage of quantum computing.

1. CRYSTALS-Kyber: Module Learning With Errors (MLWE)
2. CRYSTALS-Dilithium: Module Short Integer Solution (MSIS)
3. FALCON: NTRU Problem and SIS on NTRU lattices
4. SPHINCS+: Security of the used hash function families

Theoretical and Real-World Security

In theory, these algorithms can base their security on problems that are considered still hard given the advantage of quantum computing.

1. CRYSTALS-Kyber: Module Learning With Errors (MLWE)
2. CRYSTALS-Dilithium: Module Short Integer Solution (MSIS)
3. FALCON: NTRU Problem and SIS on NTRU lattices
4. SPHINCS+: Security of the used hash function families

In practice, the implementations of these algorithms need side-channel countermeasure, and there exist attacks on FALCON that have not been addressed (until our paper).

Attacks on FALCON

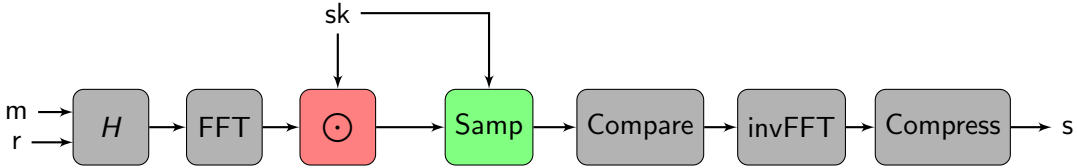

Figure: A graphical overview of FALCON.Sign.

	Attack	Countermeasure
Pre-image Vector Computation	[KA21; Gue+22]	
Gaussian Sampler over Lattices	[Gue+22; Zha+23]	[Gue+22; Zha+23]

Attacks on FALCON

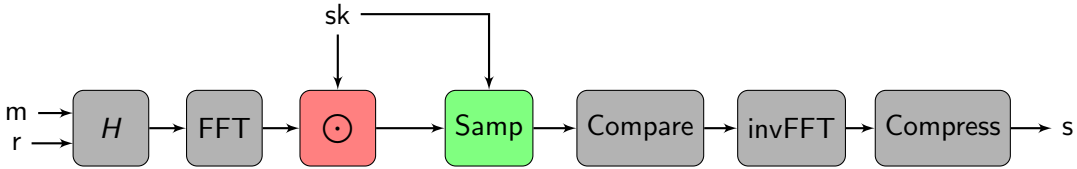

Figure: A graphical overview of FALCON.Sign.

	Attack	Countermeasure
Pre-image Vector Computation	[KA21; Gue+22]	**[This Paper]**
Gaussian Sampler over Lattices	[Gue+22; Zha+23]	[Gue+22; Zha+23]

Notation

Throughout the presentation, we assume

Notation

Throughout the presentation, we assume

- For a variable x, the jth bit of x is written as $x^{(j)}$.

Notation

Throughout the presentation, we assume

- For a variable x, the jth bit of x is written as $x^{(j)}$.
- The ith bit to jth bit ($j \geq i$) of x is represented by $x^{[j:i]}$.

Notation

Throughout the presentation, we assume

- For a variable x, the jth bit of x is written as $x^{(j)}$.
- The ith bit to jth bit ($j \geq i$) of x is represented by $x^{[j:i]}$.
- A sequence of n variables $(x_1, x_2, \cdots, x_n)$ (e.g. shares of variable x) is written as $(x_i)_{1 \leq i \leq n}$, or simply (x_i).

Notation

Throughout the presentation, we assume

- For a variable x, the jth bit of x is written as $x^{(j)}$.
- The ith bit to jth bit ($j \geq i$) of x is represented by $x^{[j:i]}$.
- A sequence of n variables $(x_1, x_2, \cdots, x_n)$ (e.g. shares of variable x) is written as $(x_i)_{1 \leq i \leq n}$, or simply (x_i).
- For a proposition P, $[\![P]\!] = 1$ if and only if P is true and 0 if otherwise.

Table of Contents

Table of Contents

Introduction to Falcon

- A NIST-standardized digital signature
- Use the Gentry-Peikert-Vaikuntanathan (GPV) framework [GPV08] with NTRU lattices

KeyGen

Public Key: $\mathbf{A} \in \mathbb{Z}_q^{N \times M}$

Secret Key: Short $\mathbf{B} \in \mathbb{Z}_q^{M \times M}$

$$\mathbf{B}\mathbf{A}^T = \mathbf{0} \bmod q$$

Sign(m)

A short signature $\mathbf{s}$ s.t.

$$\mathbf{s}\mathbf{A}^T = H(\mathrm{m}) \bmod q$$

$H : \{0,1\}^* \to \{0,1\}^N$

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

Check

1. $\mathbf{s}$ is short
2. $\mathbf{s}\mathbf{A}^T = H(\mathrm{m}) \bmod q$

Introduction to FALCON

KeyGen

Sign(m)

Verify(m, **s**)

Introduction to FALCON

KeyGen

Secret Key: Short polynomials $f, g, F, G \in \mathbb{Z}[x]/(x^N + 1)$ such that $fG - gF = q$ and

$$\mathbf{B} = \left[\begin{array}{c|c} g & -f \\ \hline G & -F \end{array}\right]$$

Sign(m)

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

Introduction to FALCON

KeyGen

Secret Key: Short polynomials $f, g, F, G \in \mathbb{Z}[x]/(x^N + 1)$ such that $fG - gF = q$ and

$$\mathbf{B} = \left[\begin{array}{c|c} g & -f \\ \hline G & -F \end{array}\right]$$

Public Key: Polynomial $h = gf^{-1} \bmod q$ and

$$\mathbf{A} = \left[\begin{array}{c|c} 1 & h \end{array}\right]$$

Note that

$$\mathbf{B}\mathbf{A}^T = \mathbf{0} \bmod q$$

Sign(m)

Verify(m, **s**)

Introduction to Falcon

KeyGen

Secret Key: Short polynomials $f, g, F, G \in \mathbb{Z}[x]/(x^N + 1)$ such that $fG - gF = q$ and

$$\mathbf{B} = \left[\begin{array}{c|c} g & -f \\ \hline G & -F \end{array}\right]$$

Public Key: Polynomial $h = gf^{-1} \bmod q$ and

$$\mathbf{A} = \left[\begin{array}{c|c} 1 & h \end{array}\right]$$

Note that

$$\mathbf{B}\mathbf{A}^T = \mathbf{0} \bmod q$$

Sign(m)

A short signature $\mathbf{s}$ such that

$$\mathbf{s}\mathbf{A}^T = H(\mathrm{r} \| \mathrm{m}) \bmod q$$

where H is a hash function and r is the random salt

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

Introduction to FALCON

KeyGen

Secret Key: Short polynomials $f, g, F, G \in \mathbb{Z}[x]/(x^N + 1)$ such that $fG - gF = q$ and

$$\mathbf{B} = \left[\begin{array}{c|c} g & -f \\ \hline G & -F \end{array}\right]$$

Public Key: Polynomial $h = gf^{-1} \bmod q$ and

$$\mathbf{A} = \left[\begin{array}{c|c} 1 & h \end{array}\right]$$

Note that

$$\mathbf{B}\mathbf{A}^T = \mathbf{0} \bmod q$$

Sign(m)

A short signature $\mathbf{s}$ such that

$$\mathbf{s}\mathbf{A}^T = H(\mathrm{r} \| \mathrm{m}) \bmod q$$

where H is a hash function and r is the random salt

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

Check

1. $\mathbf{s}$ is short
2. $\mathbf{s}\mathbf{A}^T = H(\mathrm{r} \| \mathrm{m}) \bmod q$

Introduction to FALCON

To find such a short $\mathbf{s}$, one can first

Introduction to FALCON

To find such a short $\mathbf{s}$, one can first

- Compute $H(\mathrm{m})$

Introduction to Falcon

To find such a short $\mathbf{s}$, one can first

- Compute $H(\mathrm{m})$
- Find a solution $\mathbf{c}$ (not short) where $\mathbf{c}\mathbf{A}^T = H(\mathrm{m}) \bmod q$

Introduction to FALCON

To find such a short $\mathbf{s}$, one can first

- Compute $H(\mathrm{m})$
- Find a solution $\mathbf{c}$ (not short) where $\mathbf{c}\mathbf{A}^T = H(\mathrm{m}) \bmod q$
- Compute the pre-image vector $\mathbf{t} \leftarrow \mathbf{c}\mathbf{B}^{-1}$

Introduction to FALCON

To find such a short $\mathbf{s}$, one can first

- Compute $H(\mathrm{m})$
- Find a solution $\mathbf{c}$ (not short) where $\mathbf{c}\mathbf{A}^T = H(\mathrm{m}) \bmod q$
- Compute the pre-image vector $\mathbf{t} \leftarrow \mathbf{c}\mathbf{B}^{-1}$
- Apply the nearest plane algorithm to find an integer vector $\mathbf{z}$ such that $(\mathbf{t} - \mathbf{z})\mathbf{B}$ is short.

Introduction to FALCON

To find such a short $\mathbf{s}$, one can first

- Compute $H(\mathrm{m})$
- Find a solution $\mathbf{c}$ (not short) where $\mathbf{c}\mathbf{A}^T = H(\mathrm{m}) \bmod q$
- Compute the pre-image vector $\mathbf{t} \leftarrow \mathbf{c}\mathbf{B}^{-1}$
- Apply the nearest plane algorithm to find an integer vector $\mathbf{z}$ such that $(\mathbf{t} - \mathbf{z})\mathbf{B}$ is short.
- $\mathbf{s} \leftarrow (\mathbf{t} - \mathbf{z})\mathbf{B}$. Note that $\mathbf{s}\mathbf{A}^T = H(\mathrm{m}) \bmod q$

Introduction to Falcon

To find such a short $\mathbf{s}$, one can first

- Compute $H(\mathrm{m})$
- Find a solution $\mathbf{c}$ (not short) where $\mathbf{c}\mathbf{A}^T = H(\mathrm{m}) \bmod q$
- Compute the pre-image vector $\mathbf{t} \leftarrow \mathbf{c}\mathbf{B}^{-1}$
- Apply the nearest plane algorithm to find an integer vector $\mathbf{z}$ such that $(\mathbf{t} - \mathbf{z})\mathbf{B}$ is short.
- $\mathbf{s} \leftarrow (\mathbf{t} - \mathbf{z})\mathbf{B}$. Note that $\mathbf{s}\mathbf{A}^T = H(\mathrm{m}) \bmod q$

Randomized Nearest-Plane Algorithm [GPV08]

Randomized Nearest-Plane Algorithm [GPV08]

Input: $\mathbf{t} = \mathbf{c}\mathbf{B}^{-1}, \mathbf{B}$ where $\mathbf{B} = \tilde{\mathbf{B}}\mathbf{U}$ is the Gram-Schmidt Orthogonalization, constant $\sigma > 0$
Output: $\mathbf{z} = (z_1, z_2, \cdots, z_M)$

1: **for** $i = M$ **to** 1 **do**
2: $\quad t'_i \leftarrow t_i + \sum_{j>i} \mathbf{U}_{ij}(t_j - z_j)$
3: $\quad \sigma_i \leftarrow \frac{\sigma}{\|\tilde{\mathbf{b}}_i\|}$ $\quad \{\tilde{\mathbf{b}}_i$ is the i-th row vector of $\tilde{\mathbf{B}}\}$
4: $\quad z_i \leftarrow\!\!\$\ D_{\mathbb{Z},\sigma_i,t'_i}$ $\quad$ {Sample a value z_i from a discrete Gaussian distribution }

Randomized Nearest-Plane Algorithm [GPV08]

Randomized Nearest-Plane Algorithm [GPV08]

Input: $\mathbf{t} = \mathbf{c}\mathbf{B}^{-1}, \mathbf{B}$ where $\mathbf{B} = \tilde{\mathbf{B}}\mathbf{U}$ is the Gram-Schmidt Orthogonalization, constant $\sigma > 0$
Output: $\mathbf{z} = (z_1, z_2, \cdots, z_M)$

1: **for** $i = M$ **to** 1 **do**
2: $\quad t_i' \leftarrow t_i + \sum_{j>i} \mathbf{U}_{ij}(t_j - z_j)$
3: $\quad \sigma_i \leftarrow \frac{\sigma}{\|\tilde{\mathbf{b}}_i\|}$ $\quad \{\tilde{\mathbf{b}}_i$ is the i-th row vector of $\tilde{\mathbf{B}}\}$
4: $\quad z_i \leftarrow\!\!\$\ D_{\mathbb{Z},\sigma_i,t_i'}$ $\quad$ {Sample a value z_i from a discrete Gaussian distribution }

Lemma 4.5 in [GPV08]

If $\sigma \geq \|\tilde{\mathbf{B}}\| \cdot \omega(\sqrt{\log(n)}) = \max_i \|\tilde{\mathbf{b}}_i\| \cdot \omega(\sqrt{\log(n)})$, then $\mathbf{z}\mathbf{B} \overset{\Delta}{\sim} D_{\mathcal{L}(\mathbf{B}),\sigma,\mathbf{c}}$.

Randomized Nearest-Plane Algorithm [GPV08]

Randomized Nearest-Plane Algorithm [GPV08]

Input: $\mathbf{t} = \mathbf{c}\mathbf{B}^{-1}, \mathbf{B}$ where $\mathbf{B} = \tilde{\mathbf{B}}\mathbf{U}$ is the Gram-Schmidt Orthogonalization, constant $\sigma > 0$
Output: $\mathbf{z} = (z_1, z_2, \cdots, z_M)$

1: **for** $i = M$ **to** 1 **do**
2: $\quad t_i' \leftarrow t_i + \sum_{j>i} \mathbf{U}_{ij}(t_j - z_j)$
3: $\quad \sigma_i \leftarrow \frac{\sigma}{\|\tilde{\mathbf{b}}_i\|}$ $\quad \{\tilde{\mathbf{b}}_i$ is the i-th row vector of $\tilde{\mathbf{B}}\}$
4: $\quad z_i \leftarrow\!\!\$\ D_{\mathbb{Z},\sigma_i,t_i'}$ $\quad$ {Sample a value z_i from a discrete Gaussian distribution }

Lemma 4.5 in [GPV08]

If $\sigma \geq \|\tilde{\mathbf{B}}\| \cdot \omega(\sqrt{\log(n)}) = \max_i \|\tilde{\mathbf{b}}_i\| \cdot \omega(\sqrt{\log(n)})$, then $\mathbf{z}\mathbf{B} \overset{\Delta}{\sim} D_{\mathcal{L}(\mathbf{B}),\sigma,\mathbf{c}}$.

FALCON uses fast Fourier nearest plane algorithm [DP16] to further speed up.

Introduction to FALCON

In Falcon,

Introduction to FALCON

In Falcon,

- Short secret polynomials $f, g, F, G \in \mathbb{Z}[x]/(x^N + 1)$ where

$$fG - gF = q \qquad \mathbf{B} = \left[\begin{array}{c|c} g & -f \\ \hline G & -F \end{array}\right]$$

Introduction to Falcon

In Falcon,

- Short secret polynomials $f, g, F, G \in \mathbb{Z}[x]/(x^N + 1)$ where

$$fG - gF = q \qquad \mathbf{B} = \left[\begin{array}{c|c} g & -f \\ \hline G & -F \end{array}\right]$$

- Public polynomial $h = gf^{-1} \bmod q$ and $\mathbf{A}^T = \left[\begin{array}{c} 1 \\ \hline h \end{array}\right]$

Introduction to Falcon

In Falcon,

- Short secret polynomials $f, g, F, G \in \mathbb{Z}[x]/(x^N + 1)$ where

$$fG - gF = q \qquad \mathbf{B} = \left[\begin{array}{c|c} g & -f \\ \hline G & -F \end{array}\right]$$

- Public polynomial $h = gf^{-1} \bmod q$ and $\mathbf{A}^T = \left[\begin{array}{c} 1 \\ \hline h \end{array}\right]$
- $\mathbf{c} = \left[\begin{array}{c|c} c & 0 \end{array}\right]$, where $c = H(\mathrm{r} \| \mathrm{m})$ for the message m and a random salt r.
- Short polynomials s_1 and s_2 where

$$s_1 + s_2 h = \left[\begin{array}{c|c} s_1 & s_2 \end{array}\right] \mathbf{A}^T = H(\mathrm{r} \| \mathrm{m}) \bmod q$$

Introduction to Falcon

Sign (Simplified)

Input: Message m, secret key sk, bound $\lfloor \beta^2 \rfloor$
Output: Signature sig

1: Sample salt $\mathsf{r} \leftarrow \{0,1\}^{320}$ uniformly
2: $c \leftarrow H(\mathsf{r} \| \mathsf{m})$
3: Compute the pre-image vector $\mathbf{t} \leftarrow \left[\begin{array}{c|c} c & 0 \end{array} \right] \cdot \mathbf{B}^{-1}$
4: **repeat**
5: $\quad \mathbf{z} = \mathsf{ffSampling}(\mathbf{t}, \mathsf{sk})$
6: $\quad \mathbf{s} = \left[\begin{array}{c|c} s_1 & s_2 \end{array} \right] = (\mathbf{t} - \mathbf{z})\mathbf{B}$
7: **until** $\|\mathbf{s}\|^2 \leq \lfloor \beta^2 \rfloor$
8: $\mathsf{sig} \leftarrow (\mathsf{r}, s_2)$

Verify (Simplified)

Input: Message m, signature sig
Input: Bound $\lfloor \beta^2 \rfloor$
Output: Accept or Reject

1: $c \leftarrow H(\mathsf{r} \| \mathsf{m})$
2: $s_1 \leftarrow c - s_2 h \bmod q$
3: **if** $\|(s_1, s_2)\|^2 \leq \lfloor \beta^2 \rfloor$ **then**
4: $\quad$ Accept
5: **else**
6: $\quad$ Reject

Introduction to Falcon

Sign (Simplified)

Input: Message m, secret key sk, bound $\lfloor \beta^2 \rfloor$
Output: Signature sig

1: Sample salt $\mathsf{r} \leftarrow \{0,1\}^{320}$ uniformly
2: $c \leftarrow H(\mathsf{r} \| \mathsf{m})$
3: Compute the pre-image vector $\mathbf{t} \leftarrow \left[\begin{array}{c|c} c & 0 \end{array} \right] \cdot \mathbf{B}^{-1}$
4: **repeat**
5: $\quad \mathbf{z} = \mathsf{ffSampling}(\mathbf{t}, \mathsf{sk})$
6: $\quad \mathbf{s} = \left[\begin{array}{c|c} s_1 & s_2 \end{array} \right] = (\mathbf{t} - \mathbf{z})\mathbf{B}$
7: **until** $\|\mathbf{s}\|^2 \leq \lfloor \beta^2 \rfloor$
8: $\mathsf{sig} \leftarrow (\mathsf{r}, s_2)$

Verify (Simplified)

Input: Message m, signature sig
Input: Bound $\lfloor \beta^2 \rfloor$
Output: Accept or Reject

1: $c \leftarrow H(\mathsf{r} \| \mathsf{m})$
2: $s_1 \leftarrow c - s_2 h \bmod q$
3: **if** $\|(s_1, s_2)\|^2 \leq \lfloor \beta^2 \rfloor$ **then**
4: $\quad$ Accept
5: **else**
6: $\quad$ Reject

Table of Contents

Fast-Fourier Transform

The pre-image vector computation includes polynomial multiplications

$$\mathbf{t} = \left[\begin{array}{c|c} c & 0 \end{array}\right] \cdot \mathbf{B}^{-1} = \frac{1}{q} \left[\begin{array}{c|c} c \cdot -F & c \cdot f \end{array}\right]$$

Fast-Fourier Transform

The pre-image vector computation includes polynomial multiplications

$$\mathbf{t} = \left[\begin{array}{c|c} c & 0 \end{array} \right] \cdot \mathbf{B}^{-1} = \frac{1}{q} \left[\begin{array}{c|c} c \cdot -F & c \cdot f \end{array} \right]$$

To speed up, the pre-image vector computation is performed after a Fourier transform:

$$\frac{1}{q} \left[\begin{array}{c|c} \mathrm{FFT}(c) \odot \mathrm{FFT}(-F) & \mathrm{FFT}(c) \odot \mathrm{FFT}(f) \end{array} \right]$$

Fast-Fourier Transform

The pre-image vector computation includes polynomial multiplications

$$\mathbf{t} = \left[\begin{array}{c|c} c & 0 \end{array} \right] \cdot \mathbf{B}^{-1} = \frac{1}{q} \left[\begin{array}{c|c} c \cdot -F & c \cdot f \end{array} \right]$$

To speed up, the pre-image vector computation is performed after a Fourier transform:

$$\frac{1}{q} \left[\begin{array}{c|c} \mathrm{FFT}(c) \odot \mathrm{FFT}(-F) & \mathrm{FFT}(c) \odot \mathrm{FFT}(f) \end{array} \right]$$

Therefore, the pre-image vector computation is essentially coefficient-wise complex number multiplications.

Floating-Point Number

A complex number is represented by two 64-bit floating-point numbers (FPNs). An FPN is composed of sign bit s, exponent e, and mantissa $\tilde{m}$

Figure: A 64-bit Floating-Point Number

The value is $(-1)^s \cdot 2^{e-1023} \cdot \underbrace{(1 + \tilde{m} \cdot 2^{-52})}_{\times 2^{52} = m}$

For convenience, we may use (s, e, m) to represent an FPN.

Floating-Point Number Arithmetic

FPN multiplication (FprMul) is proceeded by

FPN addition (FprAdd) is proceeded by

Floating-Point Number Arithmetic

FPN multiplication (FprMul) is proceeded by

1. Sign bit XOR

FPN addition (FprAdd) is proceeded by

Floating-Point Number Arithmetic

FPN multiplication (FprMul) is proceeded by

1. Sign bit XOR
2. Exponent Addition

FPN addition (FprAdd) is proceeded by

Floating-Point Number Arithmetic

FPN multiplication (FprMul) is proceeded by

1. Sign bit XOR
2. Exponent Addition
3. Mantissa Multiplication

FPN addition (FprAdd) is proceeded by

Floating-Point Number Arithmetic

FPN multiplication (FprMul) is proceeded by

1. Sign bit XOR
2. Exponent Addition
3. Mantissa Multiplication
4. Right-shifting the mantissa

FPN addition (FprAdd) is proceeded by

Floating-Point Number Arithmetic

FPN multiplication (FprMul) is proceeded by

1. Sign bit XOR
2. Exponent Addition
3. Mantissa Multiplication
4. Right-shifting the mantissa
5. Packing and rounding (FPR)

FPN addition (FprAdd) is proceeded by

Floating-Point Number Arithmetic

FPN multiplication (FprMul) is proceeded by

1. Sign bit XOR
2. Exponent Addition
3. Mantissa Multiplication
4. Right-shifting the mantissa
5. Packing and rounding (FPR)

FPN addition (FprAdd) is proceeded by

1. Making the first operand $\geq$ the second

Floating-Point Number Arithmetic

FPN multiplication (FprMul) is proceeded by

1. Sign bit XOR
2. Exponent Addition
3. Mantissa Multiplication
4. Right-shifting the mantissa
5. Packing and rounding (FPR)

FPN addition (FprAdd) is proceeded by

1. Making the first operand $\geq$ the second
2. Right-shifting the second operand

Floating-Point Number Arithmetic

FPN multiplication (FprMul) is proceeded by

1. Sign bit XOR
2. Exponent Addition
3. Mantissa Multiplication
4. Right-shifting the mantissa
5. Packing and rounding (FPR)

FPN addition (FprAdd) is proceeded by

1. Making the first operand $\geq$ the second
2. Right-shifting the second operand
3. Mantissa Addition / Subtraction

Floating-Point Number Arithmetic

FPN multiplication (FprMul) is proceeded by

1. Sign bit XOR
2. Exponent Addition
3. Mantissa Multiplication
4. Right-shifting the mantissa
5. Packing and rounding (FPR)

FPN addition (FprAdd) is proceeded by

1. Making the first operand $\geq$ the second
2. Right-shifting the second operand
3. Mantissa Addition / Subtraction
4. Normalizing the result

Floating-Point Number Arithmetic

FPN multiplication (FprMul) is proceeded by

1. Sign bit XOR
2. Exponent Addition
3. Mantissa Multiplication
4. Right-shifting the mantissa
5. Packing and rounding (FPR)

FPN addition (FprAdd) is proceeded by

1. Making the first operand $\geq$ the second
2. Right-shifting the second operand
3. Mantissa Addition / Subtraction
4. Normalizing the result
5. Packing and rounding (FPR)

Sticky Bit

In floating-point arithmetic, when shifted right, the mantissa maintains a sticky bit

$$100100100 \gg 4 \rightarrow 1001 \underbrace{1}_{\text{Sticky}}$$

It indicates whether there exists any 1 after the least significant bit. In the above example,

$$\text{sticky bit} = 0 \vee [\![(0100) \neq 0]\!] = [\![(00100) \neq 0]\!]$$

Floating-Point Number Packing and Rounding

FPR

Input: Sign bit s, exponent e, and 55-bit mantissa z
Output: FPN x packed by s, e, z

1: $e \leftarrow e + 1076$
2: $b \leftarrow [\![e < 0]\!]$
3: $z \leftarrow z \wedge (b - 1)$
4: $b \leftarrow [\![z \neq 0]\!]$
5: $e \leftarrow e \wedge (-b)$
6: $x \leftarrow ((s \ll 63) \vee (z \gg 2)) + e \ll 52$
7: $f \leftarrow \texttt{0XC8} \gg z^{[3:1]}$
8: $x \leftarrow x + f^{(1)}$ {increment if $z^{[3:1]}$ is 011,110 or 111}
9: **return** x

Floating-Point Number Multiplication

FprMul

Input: FPN $x = (sx, ex, mx)$
Input: FPN $y = (sy, ey, my)$
Output: FPN product of x and y

1: $s \leftarrow sx \oplus sy$
2: $e \leftarrow ex + ey - 2100$
3: $z \leftarrow mx \times my$
4: $b \leftarrow [\![z^{[50:1]} \neq 0]\!]$
5: $z \leftarrow z^{[106:51]} \vee b$
6: $z' \leftarrow (z \gg 1) \vee z^{(1)}$
7: $w \leftarrow z^{(106)}$
8: $z \leftarrow z \oplus (z \oplus z') \wedge (-w)$
9: $e \leftarrow e + w$
10: $bx \leftarrow [\![ex \neq 0]\!]$, $by \leftarrow [\![ey \neq 0]\!]$
11: $b \leftarrow bx \wedge by$
12: $z \leftarrow z \wedge (-b)$
13: **return** FPR(s, e, z)

Floating-Point Number Addition

FprAdd

Input: FPNs x and y
Output: FPN sum of x and y

1: $d \leftarrow x^{[63:1]} - y^{[63:1]}$
2: $cs \leftarrow d^{(64)} \vee ((1 - (-d)^{(64)}) \wedge x^{(64)})$
3: $m \leftarrow (x \oplus y) \wedge (-cs)$
4: $x \leftarrow x \oplus m, y \leftarrow y \oplus m$
5: Extract (sx, ex, mx) and (sy, ey, my) from x, y, respectively.
6: $mx \leftarrow mx \ll 3, my \leftarrow my \ll 3$
7: $ex \leftarrow ex - 1078, ey \leftarrow ey - 1078$
8: $c \leftarrow ex - ey$
9: $b \leftarrow [\![c < 60]\!]$
10: $my \leftarrow my \wedge (-b)$
11: $my \leftarrow (my \gg c) \vee [\![my^{[c:1]} \neq 0]\!]$
12: $s \leftarrow sx \oplus sy$
13: $z \leftarrow mx + (-1)^s my$
14: Normalize z, ex to make $z \in [2^{63}, 2^{64})$
15: $z \leftarrow (z \gg 9) \vee [\![z^{[9:1]} \neq 0]\!]$
16: $ex \leftarrow ex + 9$
17: **return** FPR(sx, ex, z)

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Reference I

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. "Private Circuits: Securing Hardware against Probing Attacks". In: *CRYPTO 2003*. Ed. by Dan Boneh. Vol. 2729. LNCS. Springer, Heidelberg, Aug. 2003, pp. 463–481. DOI: 10.1007/978-3-540-45146-4_27.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. "Trapdoors for hard lattices and new cryptographic constructions". In: *40th ACM STOC*. Ed. by Richard E. Ladner and Cynthia Dwork. ACM Press, May 2008, pp. 197–206. DOI: 10.1145/1374376.1374407.

[Ber+10] Guido Bertoni et al. "Building power analysis resistant implementations of Keccak". In: *Second SHA-3 candidate conference*. Vol. 142. Citeseer. 2010.

[GJR+11] Benjamin Jun Gilbert Goodwill, Josh Jaffe, Pankaj Rohatgi, et al. "A testing methodology for side-channel resistance validation". In: *NIST non-invasive attack testing workshop*. Vol. 7. 2011, pp. 115–136.

[Bel+13] Sonia Belaïd et al. "Differential power analysis of HMAC SHA-2 in the Hamming weight model". In: *2013 International Conference on Security and Cryptography (SECRYPT)*. 2013, pp. 1–12.

[Cor+15] Jean-Sébastien Coron et al. "Conversion from Arithmetic to Boolean Masking with Logarithmic Complexity". In: *FSE 2015*. Ed. by Gregor Leander. Vol. 9054. LNCS. Springer, Heidelberg, Mar. 2015, pp. 130–149. DOI: 10.1007/978-3-662-48116-5_7.

[Bar+16] Gilles Barthe et al. "Strong Non-Interference and Type-Directed Higher-Order Masking". In: *ACM CCS 2016*. Ed. by Edgar R. Weippl et al. ACM Press, Oct. 2016, pp. 116–129. DOI: 10.1145/2976749.2978427.

Reference II

[DP16] Léo Ducas and Thomas Prest. "Fast fourier orthogonalization". In: *Proceedings of the ACM on International Symposium on Symbolic and Algebraic Computation*. 2016, pp. 191–198.

[Din+17] A. Adam Ding et al. "Towards Sound and Optimal Leakage Detection Procedure". In: *Smart Card Research and Advanced Applications - 16th International Conference, CARDIS 2017, Lugano, Switzerland, November 13-15, 2017, Revised Selected Papers*. Ed. by Thomas Eisenbarth and Yannick Teglia. Vol. 10728. Lecture Notes in Computer Science. Springer, 2017, pp. 105–122. DOI: 10.1007/978-3-319-75208-2_7. URL: https://doi.org/10.1007/978-3-319-75208-2%5C_7.

[Bar+18] Gilles Barthe et al. "Masking the GLP Lattice-Based Signature Scheme at Any Order". In: *EUROCRYPT 2018, Part II*. Ed. by Jesper Buus Nielsen and Vincent Rijmen. Vol. 10821. LNCS. Springer, Heidelberg, Apr. 2018, pp. 354–384. DOI: 10.1007/978-3-319-78375-8_12.

[BCZ18] Luk Bettale, Jean-Sébastien Coron, and Rina Zeitoun. "Improved High-Order Conversion From Boolean to Arithmetic Masking". In: *IACR TCHES* 2018.2 (2018). https://tches.iacr.org/index.php/TCHES/article/view/873, pp. 22–45. ISSN: 2569-2925. DOI: 10.13154/tches.v2018.i2.22-45.

[Mig+19] Vincent Migliore et al. "Masking Dilithium - Efficient Implementation and Side-Channel Evaluation". In: *ACNS 19*. Ed. by Robert H. Deng et al. Vol. 11464. LNCS. Springer, Heidelberg, June 2019, pp. 344–362. DOI: 10.1007/978-3-030-21568-2_17.

Reference III

[Sch+19] Tobias Schneider et al. "Efficiently Masking Binomial Sampling at Arbitrary Orders for Lattice-Based Crypto". In: *PKC 2019, Part II*. Ed. by Dongdai Lin and Kazue Sako. Vol. 11443. LNCS. Springer, Heidelberg, Apr. 2019, pp. 534–564. DOI: 10.1007/978-3-030-17259-6_18.

[How+20] James Howe et al. "Isochronous Gaussian Sampling: From Inception to Implementation". In: *Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020*. Ed. by Jintai Ding and Jean-Pierre Tillich. Springer, Heidelberg, 2020, pp. 53–71. DOI: 10.1007/978-3-030-44223-1_5.

[Pre+20] Thomas Prest et al. *FALCON*. Tech. rep. available at https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions. National Institute of Standards and Technology, 2020.

[Bos+21] Joppe W. Bos et al. "Masking Kyber: First- and Higher-Order Implementations". In: *IACR TCHES* 2021.4 (2021). https://tches.iacr.org/index.php/TCHES/article/view/9064, pp. 173–214. ISSN: 2569-2925. DOI: 10.46586/tches.v2021.i4.173-214.

[KA21] Emre Karabulut and Aydin Aysu. "FALCON Down: Breaking FALCON Post-Quantum Signature Scheme through Side-Channel Attacks". In: *2021 58th ACM/IEEE Design Automation Conference (DAC)*. 2021, pp. 691–696. DOI: 10.1109/DAC18074.2021.9586131.

[Fri+22] Tim Fritzmann et al. "Masked Accelerators and Instruction Set Extensions for Post-Quantum Cryptography". In: *IACR TCHES* 2022.1 (2022), pp. 414–460. DOI: 10.46586/tches.v2022.i1.414-460.

Reference IV

[Gue+22] Morgane Guerreau et al. “The Hidden Parallelepiped Is Back Again: Power Analysis Attacks on Falcon”. In: *IACR TCHES* 2022.3 (2022), pp. 141–164. DOI: 10.46586/tches.v2022.i3.141-164.

[Hei+22] Daniel Heinz et al. *First-Order Masked Kyber on ARM Cortex-M4*. Cryptology ePrint Archive, Report 2022/058. https://eprint.iacr.org/2022/058. 2022.

[Zha+23] Shiduo Zhang et al. “Improved Power Analysis Attacks on Falcon”. In: *EUROCRYPT 2023, Part IV*. Ed. by Carmit Hazay and Martijn Stam. Vol. 14007. LNCS. Springer, Heidelberg, Apr. 2023, pp. 565–595. DOI: 10.1007/978-3-031-30634-1_19.

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents