

# 耐量子計算機暗号における 電子署名のマスクング

東京都立大学大学院理学研究科数理科学専攻 修士 2 年

うえもと まさと  
植元 雅斗

## 目次

1	要旨	1
2	イントロダクション	1
3	耐量子計算機暗号の署名方式 Masking-Dilithium	6
3.1	Dilithium の概要	6
3.2	Dilithium のアルゴリズム	7
3.3	Masking 化の説明	11
3.4	liboqs における Masking-Dilithium の実装の必要性	11
4	研究結果	11
4.1	Masking 関数	11
4.2	Unmasking 関数	14
4.3	Masking と Unmasking の対応付け	15
4.4	計算量の評価	18
4.5	今後の課題	18

## 1 要旨

近年の暗号業界では、量子計算機の研究の進展による暗号方式の危殆化が懸念されている。米国国立標準技術研究所（NIST: National Institute of Standards and Technology）は 2022 年に CRYSTALS-Dilithium を耐量子計算機暗号の標準アルゴリズムの 1 つとして選定した [1]。しかし、CRYSTALS-Dilithium はサイドチャネル攻撃への脆弱性が指摘されている [2]。近年はサイドチャネル攻撃への耐性を持たせるマスキング技術が注目されている。Migliore らは、ACNS2019 において CRYSTALS-Dilithium にマスキングと呼ばれる手法を施すことにより、サイドチャネル攻撃への耐性を持つ Masking-Dilithium を提案しているが [2]、NIST の定めている Dilithium のパラメータを適用した実装評価はなされていない。本稿論文では、NIST の定めている Dilithium のパラメータを適用した Masking-Dilithium をオープンソースである liboqs 上で実装することで、Masking-Dilithium の実装評価を行った。結果として、Masking-Dilithium は liboqs 上で同様に実装された CRYSTALS-Dilithium に比べて署名生成の速度が大幅に低下してしまうことが分かった。速度低下は、Masking-Dilithium を実用化する上で問題の 1 つであり、今後の展望として、Masking-Dilithium のアルゴリズムの高速化に関する研究や Julia 言語等での実装を検討したい。なお、マスキングされるデータを unmasked data、マスキングされたデータを masked data と本稿では呼ぶ。

キーワード: Masking-Dilithium, 耐量子計算機暗号, 量子計算機, liboqs

## 2 イントロダクション

昨今、契約や取引などの業務において、様々な場面でペーパーレス化が推奨されている。しかし、改ざんの痕跡が不明瞭であったり、原本のコピーが容易であるリスクが存在する。これらの回避方法の 1 つとして、本論文で取り扱う電子署名がある。まず、電子署名の一連の流れについて概説をする。送信者 Alice がメッセージに署名し、受信者 Bob がその署名を検証となっている。まず、Alice が秘密鍵でメッセージを暗号化する。Alice はメッセージと署名を Bob に送信する。Bob は受信した署名を Alice の公開鍵で復号化する。Bob は署名を復号化して得られたメッセージとアリスから直接受信したメッセージを比較して、改竄の有無を検証する。ここからは、電子署名の定義を与える。

**定義 1.** [6, p.160 定義 9.1]

$\{0,1\}^*$  を  $\{0,1\}$  の有限個の直積とする。電子署名とは、以下の性質を満たす 3 つの確率的多項式時間アルゴリズム (Gen, Sign, Verify) の組である:

1. Gen とは公開鍵と秘密鍵の対  $(P_a, S_a)$  を出力するアルゴリズムである。このことを  $(P_a, S_a) \in \text{Gen}$  と表記する。
2. Sign とは秘密鍵  $S_a$  とメッセージ  $m \in \{0,1\}^*$  に対して、署名  $\sigma$  を出力するアルゴリズムである。このことを  $\sigma = \text{Sign}(S_a, m)$  と表記する。
3. Verify とは公開鍵  $P_a$  とメッセージ  $m \in \{0,1\}^*$  及び、署名  $\sigma$  に対して、1 ビット  $b \in \{0,1\}$  を出力するアルゴリズムである。ここで、1 は署名が正しいことを意味し、0 は正しくないことを意味する。このことを  $b = \text{Verify}(P_a, m, \sigma)$  と表記する。

ここで、送信者になりすまして、署名偽造することを攻撃と呼び、攻撃する人・機械などを攻撃者と呼ぶ。こ

のとき、署名に関する情報をどの程度の攻撃者が利用できるかによって、攻撃の種類が分類される [6, p.161 9.1.2]:

- **受動的攻撃:** 公開鍵などの公開情報のみを利用する攻撃.
  - **直接攻撃:** 公開鍵のみを利用する攻撃.
  - **既知平文攻撃:** 攻撃者は平文の集合に対する正しい署名を入手し、これらの署名情報を利用して第三の平文の署名を偽造する攻撃. しかし、攻撃者はこれらの平文を選ぶことができない.
- **能動的攻撃:** 攻撃のための情報を、積極的に署名者から入手できる状態の攻撃.
  - **一般的選択平文攻撃:** 攻撃者は任意に選択した平文の集合に対する正しい署名を入手し、これらの署名情報を利用して第三の平文の署名を偽造する攻撃. 攻撃者はこれらの平文を選ぶが、平文は固定されており、署名を見る前に全ての平文のリストを作っておく必要がある.
  - **適応的選択平文攻撃:** 攻撃者は毎回適応的に（署名結果をみながら次の平文を選ぶ）任意に選んだ平文に対して、真の署名者に署名させ、そこで得た情報を利用して第三の平文の署名を偽造する攻撃.

さらに、署名の偽造の解読のレベルの分類を与える:

- **全面的解読:** 署名者の秘密鍵が計算できること.
- **一般的偽造:** 署名アルゴリズムと機能的に等価なアルゴリズムを効率的に見つけられて任意の平文の署名が偽造可能になること.
- **選択的偽造:** 攻撃者があらかじめ選んだ特定の平文に対する署名文において、偽造ができること.
- **存在的偽造:** 少なくとも 1 つの平文に対する署名文の偽造ができること.

電子署名について概説する. 整数の素因数分解や離散対数などの現在使われているアルゴリズムの安全性に関わる問題を量子計算機では高速に計算できるため、これらの数学的問題に安全性を帰着させた RSA 暗号と楕円曲線暗号は多項式時間で解かれるアルゴリズムが発見されている. 従って、量子計算機に耐性のある電子署名が必要となる. NIST は 2030 年までに量子計算機の実用化が実現すると発表している. 特にアメリカや中国などが量子計算機の研究開発に活発である. これらを踏まえて NIST では 2017 年から耐量子計算機暗号と呼ばれる、量子計算機に耐性のある暗号の標準化を目指すための公募を開始した [1]. ここで扱われている暗号のことを耐量子計算機暗号 (PQC: Post-Quantum Cryptography) と呼ぶ.

NIST は 2016 年に PQC の標準化計画を発表した. これ以降に安全性を考慮した検討が、Round4 まで進んでいる. 結果として Round4 には 3 つの署名方式が残った. 最終候補まで残ったのは FALCON と SPHINCS<sup>+</sup> と CRYSTALS-Dilithium である. なお、SPHINCS<sup>+</sup> 以外の 2 つは格子暗号をベースに制作されている. CRYSTALS-Dilithium との違いを確認するために概要を述べる.

FALCON について述べる. Hash-and-Sign 型の署名方式であり、 $X^{n+1}$  を定義多項式とする NTRU 格子上の SIS 問題の困難性を安全性の根拠としている. 高速フーリエサンプリングを利用しており、他の 2 つと比べて高速に鍵生成と署名が可能である [14, p33]. 高速フーリエサンプリングを用いるため、定義多項式の次数が  $2^k$  の形となり、パラメータ選択の自由度に制限がある. NIST の提案方式では安全性レベル 1 および 5 のパラメータセットのみが提案されている [14, p33].

SPHINCS<sup>+</sup> について述べる. これはハッシュ関数に基づく署名方式である [14, p100]. 従って SPHINCS<sup>+</sup> には格子暗号に依存した署名方式のみの存在を避けることが期待されている [1].

古典コンピュータの 1 桁を 1 ビットと呼ぶ. しかし、量子計算機では観測して明確に区別できる 2 つの状態

があって、それらの重ね合わせを 1 量子ビットと呼ぶ。論文や書籍では 1q-bit と表記されることもある。重ね合わせの状態であれば、どのような現象も 1 量子ビットとして使える。例えば、液体の中に溶け込んでいる原子の核スピンを使う核磁気共鳴量子コンピュータや超伝導素子を使う方法もある [21]。現在主流である量子コンピュータは粒子のスピン (いわゆる粒子の回転方向) を利用して、従来の 1 や 0 が決定し演算している。粒子のスピンを増やすには、電気的な抵抗や周囲の熱が量子ビットの状態に、影響を与えないようにする必要がある。同様に、粒子自体が電荷を持っているので自転すると磁石のような性質を持つてしまうので、量子ビットの状態に影響を与えないようにする必要がある。そのために利用されている技術が超伝導技術である。超伝導状態では物質によって、磁束が侵入しないだけでなく、電気抵抗も存在しない状態に近づく。超伝導状態にするには絶対零度まで物質の温度を下げることを目的とした大型な装置が必要であるため、大きなコストがかかる [22]。現在までの量子計算機の成功例としては以下が挙げられる:

- 2018 年: Bristlecone (Google) 72 量子ビット [7]
- 2020 年: 九章 (中国科学技術大学) 76 量子ビット [8]
- 2022 年: IBM Quantum System Two (IBM) 433 量子ビット [9]
- 2023 年 3 月: 超伝導量子コンピュータ (理化学研究所) 64 量子ビット [10]
- 2023 年 11 月: IBM Quantum System One with Eagle プロセッサー (東京大学と日本 IBM) 127 量子ビット [11]

実用化には約 100 万量子ビットが必要であると予想されているが、NIST の発表のように近い将来量子計算機が実用化される準備は整い始めている。

さて、NIST から有力視されている暗号の 1 つに格子暗号がある。なお、SPHINCS<sup>+</sup> のようにハッシュ関数を利用した暗号方式も存在する。まず、格子についての定義を与える。

## 定義 2. (格子)

ベクトル空間  $\mathbb{R}^m$  の  $n$  個のベクトル  $\mathbf{b}_1, \dots, \mathbf{b}_n$  の整数係数の線型結合全体の集合を

$$L(\mathbf{b}_1, \dots, \mathbf{b}_n) = \left\{ \sum_{i=1}^n a_i \mathbf{b}_i : a_i \in \mathbb{Z} \right\}$$

と表す。線型独立なベクトル  $\mathbf{b}_1, \dots, \mathbf{b}_n \in \mathbb{R}^m$  の整数係数の線型結合全体の集合  $L = L(\mathbf{b}_1, \dots, \mathbf{b}_n)$  を  $\mathbb{R}^m$  の格子と呼ぶ。

この格子を元にした問題が格子問題である。格子問題は LWE (Learning With Errors) 問題および SIS (Short Integer Solution) 問題を含む数学的問題の総称である。まず、LWE 問題について述べるために用語と表記を定義する。

## 定義 3. (内積)

$\mathbf{x} = (x_1, \dots, x_m), \mathbf{y} = (y_1, \dots, y_m) \in \mathbb{R}^m$  に対して、 $\mathbb{R}^m$  上の内積を

$$\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x} \mathbf{y}^\top = \sum_{i=1}^m x_i y_i$$

と定義する。

## 定義 4. (ベクトルの合同式)

$\mathbf{x}, \mathbf{y} \in \mathbb{R}^m$  に対し,  $1 \leq i \leq m$  なる任意の整数  $i$  について  $x_i \equiv y_i \pmod{q}$  のことを

$$\mathbf{x} \equiv \mathbf{y} \pmod{q}$$

と表記する.

**定義 5.** (環上の合同式)

$n \in \mathbb{Z}, q$  を奇素数とし, 剰余環  $\mathbb{Z}/q\mathbb{Z}$  を  $\mathbb{Z}_q$  と表記する. このとき,  $R$  と  $R_q$  をそれぞれ環  $\mathbb{Z}[X]/(X^n + 1), \mathbb{Z}_q[X]/(X^n + 1)$  とする.  $\mathbf{x}, \mathbf{y} \in R_q^m$  に対し,  $1 \leq i \leq m$  なる任意の整数  $i$  について  $x_i \equiv y_i \pmod{q}$  のことを

$$\mathbf{x} \equiv \mathbf{y} \pmod{q}$$

と表記する.

**定義 6.** (最大値ノルム)

$\mathbf{x} = (x_1, \dots, x_m)$  に対して, 以下のノルム  $\|\cdot\|_\infty$  を **最大値ノルム** と呼ぶ:

$$\|\mathbf{x}\|_\infty = \max\{|x_1|, \dots, |x_m|\}$$

**定義 7.** (暗号学的擬似乱数生成器)

入力サイズごとに以下を満たす関数  $g$  を定める:

$$g_n : \{0, 1\}^n \rightarrow \{0, 1\}^{l(n)}$$

ただし,  $l(n)$  は  $n$  の多項式で,  $g$  を計算する決定性多項式時間アルゴリズムが存在するとする. このような関数  $g$  が**暗号学的擬似乱数生成器**であるとは, 任意の確率的多項式時間アルゴリズム  $D$  と任意の正多項式  $p$  に対して, 十分大きな  $n$  で,

$$|\Pr[D(g(U_n)) = 1] - \Pr[D(U_{l(n)}) = 1]| < \frac{1}{p(n)}$$

を満たすときをいう. ただし,  $U_n$  は等確率で  $\{0, 1\}^n$  の値をとる確率変数で,  $g(U_n)$  や  $D(U_{l(n)})$  も確率変数となり, 同様に,  $\Pr[g(U_n) = 1]$  や  $\Pr[D(U_{l(n)}) = 1]$  も確率関数となる.

暗号学的擬似乱数生成器のような疑似的な乱数とは対照的に, 理想的な乱数の定義を次で与える.

**定義 8.** (一様ランダム)

ある有限の区間において, 全ての値が同じ確率で出現するような乱数の性質を**一様ランダム**と呼ぶ.

**定義 9.** (一方向性ハッシュ関数) 任意長  $\{0, 1\}^*$  から  $m$  ビットのデータへの関数  $H : \{0, 1\}^* \rightarrow \{0, 1\}^m$  が以下の性質を満たすとき, **一方向性ハッシュ関数**という.

- $\{0, 1\}^* \ni \forall x$  に対して,  $y = H(x)$  の計算が容易である.
- **衝突困難性**  $H(x') = H(x)$  となる  $x' \neq x$  の組を求めることは計算量的に困難である.
- **二次不可逆性**  $\{0, 1\}^* \ni x$  に対して,  $H(x') = H(x)$  となる  $x' \neq x$  を求めることは計算量的に困難である.
- **不可逆性**  $\{0, 1\}^m \ni y = H(x)$  に対して,  $H(x') = y$  となる  $x' \neq x$  を求めることは計算量的に困難である.

**定義 10.** (ランダムオラクル) 一方向性ハッシュ関数を仮定することを**ランダムオラクルモデル (ROM: Random Oracle Model)**と呼ぶ [6, p139]. また, 多数の状態の重ね合わせができるような量子計算が許されている一方向性ハッシュ関数を仮定することを**量子ランダムオラクルモデル (QROM: Quantum Random Oracle Model)**と呼ぶ [23, p.47 chapter2.2].

**定義 11.** [4, p.187 定義 5.2.1]

平均 0, 標準偏差  $\sigma$  の  $\mathbb{Z}$  上の離散 Gauss 分布を  $\chi$  とする. まず, 有限体  $\mathbb{F}_q$  上の秘密ベクトル  $\mathbf{s} \in \mathbb{F}_q^n$  を固定し, 一様ランダムに選ばれた  $\mathbf{a} \in \mathbb{F}_q^n$  と離散 Gauss 分布  $\chi$  からサンプリング  $e \in \mathbb{Z}$  に対して,

$$b \equiv \langle \mathbf{a}, \mathbf{s} \rangle + e \pmod{q}$$

を満たす組  $(\mathbf{a}, b) \in \mathbb{F}_q^n \times \mathbb{F}_q$  を出力する確率分布を  $L_{\mathbf{s}, \chi}$  とする.

1. 与えられた  $(\mathbf{a}, b) \in \mathbb{F}_q^n \times \mathbb{F}_q$  が, 確率分布  $L_{\mathbf{s}, \chi}$  からサンプリングされた元か,  $\mathbb{F}_q^n \times \mathbb{F}_q$  上一様ランダムに生成された元かを判定する問題を**判定 LWE 問題**と呼ぶ.
2. 確率分布  $L_{\mathbf{s}, \chi}$  からサンプリングされた組  $(\mathbf{a}, b)$  に基づいて, ベクトル  $\mathbf{s}$  を復元する問題を**探索 LWE 問題**と呼ぶ.

**注意 1.** ただし,  $(\mathbf{a}, b)$  は公開されているベクトルである. この公開されているベクトルと区別するために, 公開されていないベクトル  $\mathbf{s}$  を秘密ベクトルと表記する.

以下,  $S_\eta = \{\omega \in R : \|\omega\|_\infty \leq \eta\}$  とする.

**定義 12.** (Module-LWE 問題)

Module-LWE 分布を  $R_q^k \times R_q$  上で定義された分布とする. ただし, 各組  $(a_i, b_i) \in R_q^k \times R_q$  は,  $b_i = a_i^T s_1 + s_{2,i}$  とする. また,  $a_i$  は  $R_q^k$  から一様ランダムに選ばれる. ここで, 集合  $S_\eta^k \times S'_\eta$  から一様に選ばれた秘密ベクトル  $\mathbf{s} = (s_1, s_2) \in S_\eta^k \times S'_\eta$  を全ての組  $(a_i, b_i)$  に対して固定する.

1. 与えられた  $m$  個の組が, Module-LWE 分布から選ばれたサンプルの組  $(a_i, b_i)$  か, ランダムに生成されたサンプルの組  $(a_i, b'_i)$  かを判定する問題を**判定 Module-LWE 問題**と呼ぶ.
2. Module-LWE 分布からサンプリングされた全ての組  $(a_i, b_i)$  に基づいて, 秘密ベクトル  $\mathbf{s}$  を復元する問題を**探索 Module-LWE 問題**と呼ぶ.

次に SIS (Short Integer Solution) 問題についてである. 判定 LWE 問題は SIS 問題に帰着される.  $q \in \mathbb{Z}_{\geq 0}$  と,  $0 < \beta < q$  を満たす  $\beta \in \mathbb{R}$  を固定する.

**定義 13.** [4, p.188 定義 5.2.2]

各成分が剰余環  $\mathbb{Z}/q\mathbb{Z}$  上一様ランダムに選ばれた  $n \times m$  整数行列  $\mathbf{X}$  を与える.

$\|\mathbf{y}\| \leq \beta$  かつ  $\mathbf{y}\mathbf{X}^T \equiv 0 \pmod{q}$  を満たす非零ベクトル  $\mathbf{y} \in \mathbb{Z}^m$  を見つける問題を **SIS 問題**と呼ぶ.

**定義 14.** (Module-SIS 問題)

$R_q^{l \times k}$  上一様ランダムに選ばれた行列  $\mathbf{A}$  を与える.

$0 \leq \|\mathbf{y}\|_\infty \leq \beta$  かつ  $[\mathbf{A} | I] \cdot \mathbf{y} \equiv 0 \pmod{q}$  を満たす非零ベクトル  $\mathbf{y} \in R_q^{l+k}$  を見つける問題を **Module-SIS 問題**と呼ぶ.

**定義 15.** (SelfTarget-MSIS 問題)

ハッシュ関数  $H : \{0,1\}^n \rightarrow B_{60}$  を  $R_q$  上で定義する．ここで、 $B_\tau$  は  $R$  の元の集合である．ただし、ちょうど  $\tau$  個の多項式の係数が  $-1$  か  $1$  であり、残りの多項式の係数が  $0$  である．

$R_q^{m \times k}$  上一様ランダムに選ばれた行列  $\mathbf{A}$  と、 $R_q^m$  上一様ランダムに選ばれたベクトル  $\mathbf{t}$  を与える．

$R_q$  において、以下の条件を満たす非零ベクトル  $\mathbf{y} \in R_q^{k+m+1}$  と満たすメッセージ  $M$  を見つける問題を **SelfTarget-MSIS 問題** と呼ぶ．

- $r_1 \in R_q^k, r_2 \in R_q^m$  として、 $\mathbf{y} = \begin{bmatrix} r_1 \\ c \\ r_2 \end{bmatrix}$  を満たす  $0 \leq \|\mathbf{y}\|_\infty \leq \beta$ ．
- $c \in B_{60}$  として、 $H(M \| [\mathbf{A} | \mathbf{t} | I] \cdot \mathbf{y}) = c$ ．

これらを踏まえて、次の補題を示す．

**補題 1.** SIS 問題は LWE 問題に帰着する．

この補題を示すために、 $q$ -ary 格子を定義する．

**定義 16.** [4, p.188 定義 5.2.2]

$q\mathbb{Z}^m \subseteq \mathbf{L} \subseteq \mathbb{Z}^m$  を満たす完全階数の  $m$  次元格子  $\mathbf{L}$  を  **$q$ -ary 格子** と呼ぶ．

$m > n$  に対し、任意の  $q \in \mathbb{Z}_{\geq 0}$  と  $n \times m$  整数行列  $\mathbf{X}$  に対する  $m$  次元  $q$ -ary 格子を以下のように定義する．

$$\Lambda_q(\mathbf{X}) = \{\mathbf{y} \in \mathbb{Z}^m : \mathbf{y} \equiv \mathbf{s}\mathbf{X} \pmod{q} \text{ を満たす } \mathbf{s} \in \mathbb{Z}^n \text{ が存在する.}\},$$

$$\Lambda_q^\perp(\mathbf{X}) = \{\mathbf{y} \in \mathbb{Z}^m : \mathbf{y}\mathbf{X}^\top \equiv \mathbf{s}\mathbf{X} \pmod{q}\}$$

補題 1 の証明．奇素数の剰余パラメータ  $q$  における LWE 問題のサンプル数を  $m$  として、 $m$  個の LWE サンプルの組を  $(\mathbf{A}, \mathbf{b}) \in \mathbb{F}_q^{m \times n} \times \mathbb{F}_q$  とする．ここで  $n \times m$  の転置行列  $\mathbf{A}^\top$  に対する SIS 問題の短い解ベクトル  $\mathbf{v} \in \Lambda_q^\perp(\mathbf{A}^\top)$  が得られると仮定する．このとき、LWE サンプルの組  $(\mathbf{A}, \mathbf{b})$  は関係式  $\mathbf{b} \equiv \mathbf{s}\mathbf{A}^\top + \mathbf{e}$  を満たすので、

$$\langle \mathbf{v}, \mathbf{b} \rangle \equiv \langle \mathbf{v}, \mathbf{s}\mathbf{A}^\top + \mathbf{e} \rangle \equiv \langle \mathbf{v}\mathbf{A}, \mathbf{s} \rangle + \langle \mathbf{v}, \mathbf{e} \rangle \equiv \langle \mathbf{v}, \mathbf{e} \rangle \pmod{q}$$

が成り立つ．

さらに、ベクトル  $\mathbf{e}$  の全ての成分  $e_i$  は離散 Gauss 分布  $\chi$  からサンプリングされた元なので、

$$|\langle \mathbf{v}, \mathbf{e} \rangle| \leq \|\mathbf{e}\| \|\mathbf{v}\| \approx \sigma \|\mathbf{v}\| \leq \sigma \beta$$

が期待できる．故に  $\sigma \beta \ll q$  ならば、絶対値  $|\langle \mathbf{v}, \mathbf{e} \rangle|$  から LWE サンプルの組  $(\mathbf{A}, \mathbf{b})$  は確率分布  $L_{\mathbf{s}, \chi}$  からサンプリングされたものか判定できる．つまり、 $|\langle \mathbf{v}, \mathbf{e} \rangle|$  の値が十分に小さい場合、 $(\mathbf{A}, \mathbf{b})$  に含まれている全ての組  $(\mathbf{a}_i, b_i)$  が  $L_{\mathbf{s}, \chi}$  からサンプリングされたものと判定できる．

□

### 3 耐量子計算機暗号の署名方式 Masking-Dilithium

#### 3.1 Dilithium の概要

CRYSTALS-Dilithium は 2017 年に Shi Bai ら [13, p3.chapter.1] によって NIST へ提出された方式である．NIST の Round3 報告書によると、Dilithium はそのシンプルさから実用的な実装に向いている．最



最終的に Round4 で標準化の発表がされた。Dilithium の安全性の根拠に関してまず、ROM の元で、秘密鍵復元の困難性が Module-LWE 問題に、署名の偽造不可能性が SelfTarget-MSIS 問題にそれぞれ帰着される [13,p22.chapter.6]。同様に、QROM の元で、つまり量子計算機のメモリにおいても鍵復元、署名偽造が同様に Module-LWE 問題、SelfTarget-MSIS 問題にそれぞれ帰着される。ただし、Module-SIS 問題まで帰着されることは知られていない [13, p23.chapter.6]。

### 3.2 Dilithium のアルゴリズム

ここから Dilithium のアルゴリズムに関して記載していく。以降では安全性レベル 1 である 128 ビットのハッシュ関数を使ったパラメータを記述している。他にもレベル 3 の 192 ビット、レベル 5 の 256 ビットも存在している。NIST の発表したガイドラインによると安全レベルとその概要は以下のものである [19]:

- レベル 1 は最低レベルのセキュリティを提供し、例えば、IC カードなどに利用されている。
- レベル 3 はマルチユーザーシステムでのソフトウェア暗号化などで利用されている。
- レベル 5 は非常に安全度が高いものであるため、開発の重点からは外されている。

本研究では汎用性が高く、暗号化モジュールの基本的なセキュリティ要件を満たすレベル 1[19] で実装した。また、安全性のレベルとバイトサイズは下記の通りである:

安全性レベル	公開鍵サイズ	秘密鍵サイズ	平文サイズ	暗号文サイズ
レベル 1	800	1632	32	768
レベル 3	1184	2400	32	1088
レベル 5	1568	3168	32	1568

**定義 17.** ハッシュ関数は、与えられたデータを固定長のハッシュ値に変換する関数である。ハッシュ関数は以下の特性を持つ必要がある:

- 任意の長さのメッセージを受け取り、固定長のハッシュ値を生成する。
- 同じメッセージに対しては常に同じハッシュ値を生成する。
- 異なるメッセージに対しては異なるハッシュ値を生成する確率が非常に高い状態である。
- ハッシュ値から元のメッセージを復元することは、実用上不可能である。
- 少しでも異なる入力に対しては異なるハッシュ値が生成されるような性質を持つ (衝突耐性)。

ハッシュ関数とはデータの整合性の検証、データの一意の識別、データの非可逆な変換などの様々な用途で利用される関数である。この関数は代入された値に対してハッシュ値を返す。

Gen と Sign と Verify のアルゴリズムを記述するために、何度か使われている関数を述べる [13, p.13 Figure4]。

ここで、必要な記号を下記のように定義する:

- $q = 8380417 (= 2^{23} - 2^{13} + 1)$
- 環  $R_q = \mathbb{Z}_q[X]/(X^{256} + 1)$
- $k \in \mathbb{Z}$  において、ある実数  $a$  に対して、 $x \equiv a \pmod{\pm 2\eta} \stackrel{\text{def}}{\iff} x \pm 2\eta \cdot k = a$
- $\tilde{S}_\eta = \{\omega \in R : \omega \pmod{\pm 2\eta}\}$

なお、署名失敗のことを  $\perp$  と表記する [6, p.135 定義 8.1].

まず、ハッシュ関数が使われている関数は以下の通りである:

- ExpandA 関数は一様なシード値  $\rho \in \{0, 1\}^{256}$  から行列  $\mathbf{A} \in R_q^{k \times l}$  を生成する関数である.
- ExpandS 関数はシード値  $\rho'$  からベクトル  $(\mathbf{s}_1, \mathbf{s}_2) \in S_\eta^l \times S_\eta^k$  を生成する関数である.
- ExpandMask 関数はシード値  $\rho'$  から  $\kappa$ (一度だけ使われる数) とベクトル  $\mathbf{y} \in \tilde{S}_\eta^l$  を生成する関数である.
- H 関数はさまざまなシード値から可変長のハッシュ値を生成する関数である.

---

**Algorithm 1:** Power2Round $_q(r, d)$  [13, p.12 Figure.4]

---

**Input:**  $(r, d)$

```

1  $r \leftarrow r \pmod{+q}$ ;
2  $r_0 \leftarrow r \pmod{\pm 2^d}$ ;
3 return  $((r - r_0)/2^d, r_0)$ ;
```

---



---

**Algorithm 2:** Decompose $_q(r, \alpha)$  [13, p.12 Figure .4]

---

**Input:**  $(r, \alpha)$

```

1  $r \leftarrow r \pmod{+q}$ ;
2  $r_0 \leftarrow r \pmod{\pm \alpha}$ ;
3 if  $r - r_0 = q - 1$  then
4    $r_1 \leftarrow 0, r_0 \leftarrow r_0 - 1$ ;
5 end
6 else
7    $r_1 \leftarrow (r - r_0)/\alpha$ 
8 end
9 return  $(r_1, r_0)$ ;
```

---



---

**Algorithm 3:** HighBits $_q(r, \alpha)$  [13, p.12 Figure .4]

---

**Input:**  $(r, \alpha)$

```

1  $(r_1, r_0) \leftarrow \text{Decompose}_q(r, \alpha)$ ;
2 return  $r_1$ ;
```

---



---

**Algorithm 4:** LowBits $_q(r, \alpha)$  [13, p.12 Figure .4]

---

**Input:**  $(r, \alpha)$

```

1  $(r_1, r_0) \leftarrow \text{Decompose}_q(r, \alpha)$ ;
2 return  $r_0$ ;
```

---

---

**Algorithm 5:**  $\text{MakeHint}_q(z, r, \alpha)$  [13, p.12 Figure.4]

---

**Input:**  $(z, r, \alpha)$ 

```
1  $r_1 \leftarrow \text{HighBits}_q(r, \alpha);$   
2  $v_1 \leftarrow \text{HighBits}_q(r + z, \alpha);$   
3 return  $\llbracket r_1 \neq v_1 \rrbracket;$ 
```

---

---

**Algorithm 6:**  $\text{UseHint}_q(h, r, \alpha)$  [13, p.12 Figure.4]

---

**Input:**  $(h, r, \alpha)$ 

```
1  $m \leftarrow (q - 1)/\alpha;$   
2  $(r_1, r_0) \leftarrow \text{Decompose}_q(r, \alpha);$   
3 if  $h = 1$  and  $r_0 > 0$  then  
4    $r_1 \leftarrow (r_1 + 1) \pmod{+m};$   
5 end  
6 else  
7    $r_1 \leftarrow (r_1 - 1) \pmod{+m};$   
8 end  
9 return  $r_1;$ 
```

---

ここまですが使われている関数である.

ここから Gen と Sign と Verify のアルゴリズムを記述する.

---

**Algorithm 7:**  $(P_a, S_a) \in \text{Gen}$ [13, p.13 Figure.4]

---

```
1  $\xi \leftarrow \{0, 1\}^{256};$   
2  $(\rho, \rho', K) \leftarrow \text{H}(\xi) \in \{0, 1\}^{256} \times \{0, 1\}^{512} \times \{0, 1\}^{512};$   
3  $\mathbf{A} \leftarrow \text{ExpandA}(\rho) \in R_q^{k \times l};$   
4  $(\mathbf{s}_1, \mathbf{s}_2) \leftarrow \text{ExpandS}(\rho') \in S_\eta^l \times S_\eta^k;$   
5  $\mathbf{t} \leftarrow \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2;$   
6  $tr \leftarrow \text{H}(\rho \parallel \mathbf{t}_1) \in \{0, 1\}^{256};$   
7 return  $P_a = (\rho, \mathbf{t}_1), S_a = (\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0);$ 
```

---

---

**Algorithm 8:**  $\sigma = \text{Sign}(S_a, m)$  [13, p.13 Figure.4]

---

**Input:**  $S_a = (\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0), M$

```

1  $\mathbf{A} \leftarrow \text{ExpandA}(\rho) \in R_q^{k \times l};$ 
2  $\mu \leftarrow H(tr \| M) \in \{0, 1\}^{512};$ 
3  $\kappa \leftarrow 0;$ 
4  $(\mathbf{z}, \mathbf{h}) \leftarrow \perp;$ 
5  $\rho' \leftarrow H(K \| \mu) \in \{0, 1\}^{512};$ 
6 while  $(\mathbf{z}, \mathbf{h}) = \perp$  do
7    $\mathbf{y} \leftarrow \text{ExpandMask}(\rho', \kappa) \in S_{\gamma_1}^l;$ 
8    $\mathbf{w} \leftarrow \mathbf{A}\mathbf{y};$ 
9    $\mathbf{w}_1 \leftarrow \text{HighBits}_q(\mathbf{w}, 2\gamma_2);$ 
10   $\tilde{c} \in \{0, 1\}^{256} \leftarrow H(\mu \| \mathbf{w}_1);$ 
11   $c \in B_\tau \leftarrow \text{SampleInBall}(\tilde{c});$ 
12   $\mathbf{z} \leftarrow \mathbf{y} + c\mathbf{s}_1;$ 
13   $\mathbf{r}_0 \leftarrow \text{LowBits}_q(\mathbf{w} - c\mathbf{s}_1, 2\gamma_2);$ 
14  if  $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$  or  $\|\mathbf{r}_0\|_\infty \geq \gamma_2 - \beta$  then
15     $(\mathbf{z}, \mathbf{h}) \leftarrow \perp;$ 
16  end
17  else
18     $\mathbf{h} \leftarrow \text{MakeHint}_q(-c\mathbf{t}_0, \mathbf{w} - c\mathbf{s}_2 + c\mathbf{t}_0, 2\gamma_2);$ 
19    if  $\|c\mathbf{t}_0\|_\infty \geq \gamma_2$  then
20       $(\mathbf{z}, \mathbf{h}) \leftarrow \perp;$ 
21    end
22  end
23   $\kappa \leftarrow \kappa + l;$ 
24 end
25 return  $\sigma = (\tilde{c}, \mathbf{z}, \mathbf{h});$ 

```

---

**Algorithm 9:**  $b = \text{Verify}(P_a, m, \sigma)$  [13, p.13 Figure.4]

---

**Input:**  $P_a = (\rho, \mathbf{t}_1), M, \sigma = (\tilde{c}, \mathbf{z}, \mathbf{h})$

```

1  $\mathbf{A} \in R_q^{k \times l} \leftarrow \text{ExpandA}(\rho);$ 
2  $\mu \in \{0, 1\}^{512} \leftarrow H(H(\rho \| \mathbf{t}_1) \| M);$ 
3  $c \leftarrow \text{SampleInBall}(\tilde{c});$ 
4  $\mathbf{w}'_1 \leftarrow \text{UseHint}_q(h, \mathbf{A}\mathbf{z} - c\mathbf{t}_1 \cdot 2^d, 2\gamma_2);$ 
5 return  $\llbracket \|\mathbf{z}\|_\infty < \gamma_1 - \beta \rrbracket$  and  $\llbracket \tilde{c} = H(\mu \| \mathbf{w}'_1) \rrbracket;$ 

```

---

### 3.3 Masking 化の説明

ここからは Masking 化する際に検討した点を説明する。Masking は、パスワードや秘密鍵などの情報を別のランダムなデータでマスクすることで、サイドチャネル攻撃からの情報漏洩を防ぐことを目的としている。この技術は、暗号において物理的な攻撃からの保護を強化するために導入された。

機器が発する電気信号を読み取って、暗号を解読するための攻撃をサイドチャネル攻撃と呼ぶ。どんなに優れた暗号方式でも機器から発する微弱な信号はコードやモニターへ無防備のままに向かう。オンライン上で優れていてもオフラインが安全とは限らない。この点を突いた攻撃がサイドチャネル攻撃である。なお、機器が発する電気信号を読み取って、モニター越しに見える情報を盗聴するための攻撃はテンペスト攻撃と呼ぶ。

一般的なマスキングの種類について解説する [3]:

- **シンプルマスキング**: パスワードや鍵などの秘密情報を単一のランダムなマスク値で保護する方法。秘密情報とマスク値を単純に組み合わせることで、サイドチャネル攻撃から保護する。
- **Shamir の秘密分散**: 秘密情報を複数のシェアに分割し、それぞれのシェアにランダムなマスク値を加えて保護する方法。しきい値を持つ一定数のシェアがあれば、秘密情報を再構築できる。
- **ダブルマスキング**: 秘密情報を 2 つの異なるマスクで保護する方法。これにより、各マスクの間に依存関係がなくなり、攻撃者が両方のマスクを同時に突破することが困難になる。

本論文で扱っている Masking は分割した後ランダムなマスク値に XOR 演算をしているものであるシンプルマスキングである。また、関数内で暗号化に利用される値が保持されることは Masking が必要な関数の性質である。

### 3.4 liboqs における Masking-Dilithium の実装の必要性

unmasked の弱点について述べる。関数において差動電力解析 (DPA: Differential Power Analysis) を行い、観察されたピークが実際に利用可能な漏洩に対応していることが確認されている [2, p.7 Section.4]。従って unmasked の状況下では Masking 処理が必要である。

また、liboqs は MIT ライセンスに基づいてリリースされた耐量子計算機暗号アルゴリズム用のオープンソース C ライブラリである [20]。このライブラリの拡充によって、耐量子計算機暗号が発展していく。以上より、liboqs における Masking-Dilithium の実装が必要である。

## 4 研究結果

ここから、Masking-Dilithium を liboqs 上で実装する時に検討した点について説明する。

### 4.1 Masking 関数

Masking-Dilithium が記述された論文からオープンソースである liboqs 上で Masking-Dilithium を実装するために変えるべきパラメータに関して記述する。なお、CRYSTALS-Dilithium と liboqs のパラメータは同様なので省略している。

#### 4.1.1 変更不要のパラメータ

- 定義多項式の  $x^{N+1}$  の  $N$  (256)
- 環  $R_q$  の  $q$  ( $8380417 = (2^{23} - 2^{13} + 1)$ )
- 署名 (2432 バイト)
- 公開鍵 (800 バイト)

項目	liboqs	Masking
秘密鍵の長さ	2528	3616
行列 $A$ ( $L$ 行 $\times$ $K$ 列)	$(4 \times 4)$	$(5 \times 4)$
落とされたビット数	(13bits)	(14bits)
$\beta$	78	235
$\omega$	80	32

#### 4.1.2 Masking 関数のアルゴリズム

鍵生成と署名において使われる関数の中で, Masking 化されるべき関数は下記の通りである (2) :

- MakeHints 関数
  - Decompose 関数
    - \* HighBits 関数
    - \* LowBits 関数
- sampling 関数

ここでは, まず, Masking された関数のアルゴリズムを記述する.

---

##### Algorithm 10: Masking-HighBits<sub>q</sub>

---

**Input:**  $(r)_{0 \leq i \leq t}, \beta$

- 1  $(mask) \leftarrow \beta - 1;$
  - 2  $(d)_{0 \leq i \leq t} \leftarrow \text{arith-mask}((mask \ll 1) + 1);$
  - 3  $(r_1)_{0 \leq i \leq t} \leftarrow \text{arith-add}((r)_{0 \leq i \leq t}, (d)_{0 \leq i \leq t});$
  - 4  $(r_1)_{0 \leq i \leq t} \leftarrow \text{arith-rshift}((r_1)_{0 \leq i \leq t}, \log_2 \beta);$
  - 5 **return**  $(r_1)_{0 \leq i \leq t};$
- 

---

##### Algorithm 11: Masking-LowBits<sub>q</sub>

---

**Input:**  $(r)_{0 \leq i \leq t}, \beta$

- 1  $(r_0)_{0 \leq i \leq t} \leftarrow \text{arith-mask-bool-convert}((r)_{0 \leq i \leq t});$
  - 2  $(r_0)_{0 \leq i \leq t} \leftarrow \text{bool-lshift}((r_0)_{0 \leq i \leq t}, \omega - \log_2 \beta);$
  - 3  $(b_0)_{0 \leq i \leq t} \leftarrow \text{bool-maskfromsign}((r_0)_{0 \leq i \leq t});$
  - 4  $(b_0)_{0 \leq i \leq t} \leftarrow \text{bool-lshift}((b_0)_{0 \leq i \leq t}, \log_2 \beta);$
  - 5  $(r_0)_{0 \leq i \leq t} \leftarrow \text{bool-rshift}((r_0)_{0 \leq i \leq t}, \omega - \log_2 \beta);$
  - 6  $(r_0)_{0 \leq i \leq t} \leftarrow \text{bool-xor}((r_0)_{0 \leq i \leq t}, (b_0)_{0 \leq i \leq t});$
  - 7 **return**  $(r_0)_{0 \leq i \leq t};$
-

---

**Algorithm 12:** Masking-Decompose

---

**Input:**  $(r)_{0 \leq i \leq t}, \beta$ 

- 1  $(r_1)_{0 \leq i \leq t} \leftarrow \text{Masking-HighBits}_q((r)_{0 \leq i \leq t}, \beta);$
  - 2  $(r_0)_{0 \leq i \leq t} \leftarrow \text{Masking-LowBits}_q((r)_{0 \leq i \leq t}, \beta);$
  - 3 **return**  $(r_0)_{0 \leq i \leq t}, (r_1)_{0 \leq i \leq t};$
- 

---

**Algorithm 13:** Masking-MakeHint

---

**Input:**  $(r)_{0 \leq i \leq t}, (z)_{0 \leq i \leq t}, \beta$ 

- 1  $(r_1)_{0 \leq i \leq t} \leftarrow \text{Masking-HighBits}_q((r)_{0 \leq i \leq t}, \beta);$
  - 2  $(a)_{0 \leq i \leq t} \leftarrow \text{arith-add}((r)_{0 \leq i \leq t}, (z)_{0 \leq i \leq t});$
  - 3  $(a_1)_{0 \leq i \leq t} \leftarrow \text{Masking-HighBits}_q((a)_{0 \leq i \leq t}, \beta);$
  - 4  $(t)_{0 \leq i \leq t} \leftarrow \text{bool-xor}((r_1)_{0 \leq i \leq t}, (a_1)_{0 \leq i \leq t});$
  - 5  $(t)_{0 \leq i \leq t} \leftarrow \text{bool-lshif}((r_1)_{0 \leq i \leq t}, (r_1)_{0 \leq i \leq t}, \omega - 1);$
  - 6  $c \leftarrow \text{bool-fullxor}((t)_{0 \leq i \leq t});$
  - 7 **return**  $c;$
- 

次に、後述する Unmasking 関数とこれらの関数の対応付けのために必要な改良が加えられ、実際に実装されたアルゴリズムを記述する。

---

**Algorithm 14:** Masking-HighBits $_q$ -2

---

**Input:**  $(r)_{0 \leq i \leq t}, \beta$ 

- 1  $(mask) \leftarrow \beta - 1;$
  - 2  $(ran)_{0 \leq i \leq t} \leftarrow H(\kappa);$
  - 3  $(d)_{0 \leq i \leq t} \leftarrow ((ran)_{0 \leq i \leq t} \text{ xor } ((mask \ll 1) + 1));$
  - 4  $(r_1)_{0 \leq i \leq t} \leftarrow ((r)_{0 \leq i \leq t} + (d)_{0 \leq i \leq t});$
  - 5  $(r_1)_{0 \leq i \leq t} \leftarrow ((r_1)_{0 \leq i \leq t} \gg \log_2 \beta);$
  - 6 **return**  $(r_1)_{0 \leq i \leq t}, (ran)_{0 \leq i \leq t};$
-

---

**Algorithm 15:** Masking-LowBits $_q$ -2

---

**Input:**  $(r)_{0 \leq i \leq t}, \beta$ 

```
1  $(ran_0)_{0 \leq i \leq t} \leftarrow H(\kappa);$ 
2  $(d)_{0 \leq i \leq t} \leftarrow ((ran)_{0 \leq i \leq t} \text{ xor } ((r)_0);$ 
3 for  $i \leftarrow 1$  to  $t - 1$  do
4    $(ran_1)_i \leftarrow H(\kappa);$ 
5    $(x)_i \leftarrow ((ran)_i \text{ xor } ((r)_i);$ 
6    $(r_0)_{0 \leq i \leq t} \leftarrow (r)_{0 \leq i \leq t} + (d)_{0 \leq i \leq t};$ 
7 end
8  $(r_0)_{0 \leq i \leq t} \leftarrow ((r_0)_{0 \leq i \leq t} \gg \omega - \log_2 \beta);$ 
9  $(b_0)_{0 \leq i \leq t} \leftarrow ((r)_{0 \leq i \leq t} \gg \omega - 1);$ 
10  $(b_0)_{0 \leq i \leq t} \leftarrow (-1 \times (b_0)_{0 \leq i \leq t});$ 
11  $(b_0)_{0 \leq i \leq t} \leftarrow ((b_0)_{0 \leq i \leq t} \ll \log_2 \beta);$ 
12  $(r_0)_{0 \leq i \leq t} \leftarrow ((r_0)_{0 \leq i \leq t} \gg \omega - \log_2 \beta);$ 
13  $(r_0)_{0 \leq i \leq t} \leftarrow ((r_0)_{0 \leq i \leq t} \text{ xor } (b_0)_{0 \leq i \leq t});$ 
14 return  $(r_0)_{0 \leq i \leq t}, (ran_0)_{0 \leq i \leq t}, (ran_1)_{0 \leq i \leq t};$ 
```

---

## 4.2 Unmasking 関数

ここから新たに, Unmasking のアルゴリズムを与える. このアルゴリズムは Masking-Dilithium を提唱した論文には `arith::unmask` という表記で, 存在だけが記述されているが [2, p.11 Fig.4], 明確なアルゴリズムは提起されていない. 本論文ではここからそのアルゴリズムを与える. なお, Unmasking 関数を制作する上で重要な点は, ランダムな値を保持する機構を制作し, Unmasking 関数へ渡す点である. このランダムな値を Unmasking へ代入させて Masking と同じ計算をさせてその値と XOR 演算をさせる. その結果, 上位の数ビットのみが HighBits 関数に代入した時と結果と同じ値になる.

---

**Algorithm 16:** Unmasking-HighBits $_q$ -2

---

**Input:**  $(r)_{0 \leq i \leq t}, \beta, (ran)_{0 \leq i \leq t}$ 

```
1  $(mask) \leftarrow \beta - 1;$ 
2  $(d)_{0 \leq i \leq t} \leftarrow ((ran)_{0 \leq i \leq t} \text{ xor } ((mask \ll 1) + 1));$ 
3  $(r_1)_{0 \leq i \leq t} \leftarrow ((r)_{0 \leq i \leq t} - (d)_{0 \leq i \leq t});$ 
4  $(r_1)_{0 \leq i \leq t} \leftarrow ((r_1)_{0 \leq i \leq t} \ll \log_2 \beta);$ 
5 return  $(r_1)_{0 \leq i \leq t};$ 
```

---



---

**Algorithm 17:** Unmasking-LowBits<sub>q</sub>-2

---

**Input:**  $(r_0)_{0 \leq i \leq t}, (ran_0)_{0 \leq i \leq t}, (ran_1)_{0 \leq i \leq t}$ 

```
1  $(d)_{0 \leq i \leq t} \leftarrow ((ran)_{0 \leq i \leq t} \text{ xor } ((r)_0));$ 
2 for  $i \leftarrow 1$  to  $t - 1$  do
3    $(x)_i \leftarrow ((ran)_i \text{ xor } ((r)_i));$ 
4    $(r_0)_{0 \leq i \leq t} \leftarrow (r)_{0 \leq i \leq t} - (d)_{0 \leq i \leq t};$ 
5 end
6  $(r_0)_{0 \leq i \leq t} \leftarrow ((r)_{0 \leq i \leq t});$ 
7  $(r_0)_{0 \leq i \leq t} \leftarrow ((r_0)_{0 \leq i \leq t} \gg \omega - \log_2 \beta);$ 
8  $(b_0)_{0 \leq i \leq t} \leftarrow ((r)_{0 \leq i \leq t} \gg \omega - 1);$ 
9  $(b_0)_{0 \leq i \leq t} \leftarrow (-1 \times (b_0)_{0 \leq i \leq t});$ 
10  $(b_0)_{0 \leq i \leq t} \leftarrow ((b_0)_{0 \leq i \leq t} \ll \log_2 \beta);$ 
11  $(r_0)_{0 \leq i \leq t} \leftarrow ((r_0)_{0 \leq i \leq t} \gg \omega - \log_2 \beta);$ 
12  $(r_0)_{0 \leq i \leq t} \leftarrow ((r)_{0 \leq i \leq t} \text{ xor } (b)_{0 \leq i \leq t});$ 
13 return  $(r_0)_{0 \leq i \leq t};$ 
```

---

---

**Algorithm 18:** Unmasking-Decompose

---

**Input:**  $(r)_{0 \leq i \leq t}, \beta$ 

```
1  $((r_1)_{0 \leq i \leq t}, (ran)_{0 \leq i \leq t}) \leftarrow \text{Masking-HighBits}_q\text{-}2((r)_{0 \leq i \leq t}, \beta);$ 
2  $((r_0)_{0 \leq i \leq t}, (ran_0)_{0 \leq i \leq t}, (ran_1)_{0 \leq i \leq t}) \leftarrow \text{Masking-LowBits}_q\text{-}2((r)_{0 \leq i \leq t}, \beta);$ 
3  $(r_1)_{0 \leq i \leq t} \leftarrow \text{Unasking-HighBits}_q((r)_{0 \leq i \leq t}, \beta, (ran)_{0 \leq i \leq t});$ 
4  $(r_0)_{0 \leq i \leq t} \leftarrow \text{Unasking-LowBits}_q((r)_{0 \leq i \leq t}, (ran_0)_{0 \leq i \leq t}, (ran_1)_{0 \leq i \leq t});$ 
5 return  $((r_0)_{0 \leq i \leq t}, (r_1)_{0 \leq i \leq t});$ 
```

---

---

**Algorithm 19:** Unmasking-MakeHint

---

**Input:**  $(r)_{0 \leq i \leq t}, (z)_{0 \leq i \leq t}, \beta$ 

```
1  $((r_1)_{0 \leq i \leq t}, (ran)_{0 \leq i \leq t}) \leftarrow \text{Masking-HighBits}_q\text{-}2((r)_{0 \leq i \leq t}, \beta);$ 
2  $(r_1)_{0 \leq i \leq t} \leftarrow \text{Unasking-HighBits}_q((r)_{0 \leq i \leq t}, \beta, (ran)_{0 \leq i \leq t});$ 
3  $(a)_{0 \leq i \leq t} \leftarrow ((r)_{0 \leq i \leq t} + (z)_{0 \leq i \leq t});$ 
4  $(a_1)_{0 \leq i \leq t}, (ran)_{0 \leq i \leq t} \leftarrow \text{Masking-HighBits}_q\text{-}2((a)_{0 \leq i \leq t}, \beta);$ 
5  $((a_1)_{0 \leq i \leq t} \leftarrow \text{Unasking-HighBits}_q((a_1)_{0 \leq i \leq t}, (ran)_{0 \leq i \leq t});$ 
6  $(t)_{0 \leq i \leq t} \leftarrow ((r_1)_{0 \leq i \leq t} \text{ xor } (a_1)_{0 \leq i \leq t});$ 
7  $(t)_{0 \leq i \leq t} \leftarrow ((t)_{0 \leq i \leq t} \ll \omega - 1);$ 
8  $c \leftarrow ((t)_{0 \leq i \leq t} \text{ xor } c);$ 
9 return  $c;$ 
```

---

### 4.3 Masking と Unmasking の対応付け

ここから、実際に Unmasking-HighBits 関数が正しく使用可能かの判定として、下記の定理を示す。

**主定理 1.** (Unmasking の有用性)

任意の入力値  $x$  に対して, Masking-HighBits 関数へ値を代入した後, その値を Unmasking-HighBits 関数に代入して計算された出力値  $y$  を考える. このとき, 同じ入力値  $x$  に対して, HighBits 関数へ代入して計算された出力値  $y'$  は  $y$  と一致する.

この主定理は次の主定理に帰着できる.

**主定理 2.** (Unmasking の数値的な有用性)

$n \leq 5$  の自然数  $n$ , 整数  $k$  に対して,  $x_n, y_n \in \{0, 1\}^{2k}$  とする. また,

$$\begin{aligned} M(x_1) = x_2, HB(x_2) = x_3, Un(x_3) = x_4, HB(x_4) = x_5 \\ HB(y_1) = y_2 \end{aligned}$$

とする. このとき,

$$x_1 = y_1 \Rightarrow x_5 = y_2$$

となる.

この主定理 2 を証明して, Unmasking-HighBits 関数が正しく使用可能かの判定をする.

**証明.** 数学的帰納法で示す.

$k = 1$  のとき, 次の補題を数値計算して示す.

**補題 2.**  $n \leq 5$  の自然数  $n$  に対して,  $x_n, y_n \in \{0, 1\}^2$  とする. また,

$$\begin{aligned} M(x_1) = x_2, HB(x_2) = x_3, Un(x_3) = x_4, HB(x_4) = x_5 \\ HB(y_1) = y_2 \end{aligned}$$

とする. このとき,

$$x_1 = y_1 \Rightarrow x_5 = y_2$$

となる.

**準備 1.** まず, 下記の疑似関数を 2 桁の二進数で設定する.

- **HB: HighBits** 上位 1 桁を取得する関数
- **M: Mask** 各桁にランダムに生成された Mask 値を XOR する関数
- **Un: Unmask** 各桁に受け取った Mask 値を XOR する関数

**証明.** 補題の証明全パターンを数値計算して確認する. つまり Mask 値 (= 00, 01, 10, 11) に対して入力値 (= 00, 01, 10, 11) を数値計算する. 下記の表はその結果である.

Mask 値	入力値: $x$	$UM(HB(M(x)))$	$HB(UM(HB(M(x))))$	$HB(x)$
00	00	00	00	00
00	01	00	00	00
00	10	10	10	10
00	11	10	10	10
01	00	01	00	00
01	01	01	00	00
01	10	11	10	10
01	11	11	10	10
10	00	00	00	00
10	01	00	00	00
10	10	10	10	10
10	11	10	10	10
11	00	01	00	00
11	01	01	00	00
11	10	11	10	10
11	11	11	10	10

このとき, どの入力値でも  $HB(UM(HB(M(x))))=HB(x)$  となっているので補題 2 が示された. □

$HB(UM(HB(M(x))))$  の数値計算までどのような計算例になるのかを述べる.

**例 1.** 入力値を  $10 \in \{0, 1\}^2$  とすると,  $x_5$  の計算は下記ようになる.

$$\begin{aligned}
M(10) &= 11 & (\Leftrightarrow 10 \text{ XOR } 01 = 11) \\
HB(11) &= 10 & (\Leftrightarrow \text{上位は } 1) \\
Un(10) &= 11 & (\Leftrightarrow 10 \text{ XOR } 01 = 11) \\
HB(11) &= 10 & (\Leftrightarrow \text{上位は } 1)
\end{aligned}$$

同様に, 入力値を  $10 \in \{0, 1\}^2$  とすると,  $y_2$  の計算は下記ようになる.

$$HB(10) = 10 \quad (\Leftrightarrow \text{上位は } 1)$$

この結果は補題 2 において,

$$x_1 = y_1 \Rightarrow x_5 = y_2$$

この式の成立例として, 確認できる.

主定理 2 の証明に戻る.

次に, 下記の疑似関数を  $2k$  桁の 2 進数で改めて設定する.

- **HB: HighBits** 上位  $k$  桁を取得する関数
- **M: Mask** 各桁にランダムに生成された Mask 値を XOR する関数

- **Un:** Unmask 各桁に受け取った Mask 値を XOR する関数

任意の自然数  $s$  について,  $k = s$  のときの成立を仮定する.

$k = s + 1$  のときを考える.

仮定より, 上位  $2s$  桁では  $x_1 = y_1 \Rightarrow x_5 = y_2$  が成立している. 上位  $2s + 1$  桁目に着目すると, 下記の計算結果の表として確認できる.

Mask 値	$2s + 1$ 桁目	$(\text{HB}(\text{M}(x)))$	$\text{HB}(\text{UM}(\text{HB}(\text{M}(x))))$	$\text{HB}(x)$
0	0	1	0	0
0	1	0	1	1
1	0	0	0	0
1	1	1	1	1

よって, 上位  $2s + 1$  桁で  $x_5 = y_2$  が示された.

以上, 補題 2 と主定理 2 から Unmasking の有用性に関する主定理 1 が証明できた. すなわち, Unmasking-HighBits 関数を正しく使用できることを示した.  $\square$

#### 4.4 計算量の評価

$\sigma = \text{Sign}(S_a, m)$  の 13 行目において,  $\text{LowBits}_q$  関数が使われている. この関数は  $\text{Masking-LowBits}_q-2$  関数へ  $\text{Masking}$  化されて使用するが, これらの計算量に差がある.

まず,  $\text{LowBits}_q$  関数の計算量  $\mathcal{O}$  は  $\omega$  の次数に依存して,  $\mathcal{O}(k)$  となる. 一方,  $\text{Masking-LowBits}_q-2$  関数の計算量  $\mathcal{O}$  は  $\omega$  の次数に加えて, 関数内部でループが存在するため,  $\mathcal{O}(k^2)$  となる. 従って,  $\text{Masking}$  した際に計算量が増加してしまう. これは  $\sigma = \text{Sign}(S_a, m)$  に時間がかかることを意味する.

#### 4.5 今後の課題

HighBits などの関数がサイドチャネル攻撃に対して弱いことが実験によって明らかになっている [2]. しかし, 本論文で導入した Unmasking の実験はされていないため, 安全性の検証は必要であることを指摘しておく. ただし, 数値に対して演算を加えている点やランダム値を利用している点から理論上は安全であると予想している. また,  $\text{Masking}$  された関数がある署名段階では計算量が多くにあり結果として時間がかかる傾向にある. 従って, 高速化に定評のある Julia 言語による実装などがこの課題に対して有効であると予想される. 特に確率分布と関連関数用の Julia パッケージである Distributions.jl などが想定される [18]. さらに, 理想的な実装への最適なパラメータ選定を進めているが, 現在完全な完結には至っていない. 特に,  $\text{LowBits}$  関数の  $\text{Masking}$  と  $\text{Unmasking}$  の対応付けが急務の課題である. そして, NIST の Round3 報告書において FALCON との比較が行われ, Dilithium はそのシンプルさから一般的な実装に向いていると発表があった. この発表によると, FALCON は署名の短さからリソースの制限されたデバイスで使われることが期待されている. この FALCON についても, 同様に  $\text{Masking}$  実装が必要な可能性も高いと考えられる.

#### 謝辞

本論文の完成にあたり, 多くの方々にご指導ご鞭撻そして, ご支援を賜りました. 指導教官の横山俊一先生には熱心なご指導を賜りました. 学術面ではもちろん, 論文における日本語の曖昧さに対する意識を持続的に

ご教授頂きました。加えて、未熟な筆者が日々の日常でご迷惑をおかけする中でも匙を投げず、今後の人生観や円滑な人間関係そして、深みのある物に対する姿勢などを継続的に指し示して頂きました。短文ではございますが、ここに筆舌に尽くせないほどの深謝の意を表します。また、同学科の内山先生、並びに内田先生には、本論文の作成にあたり、副査として適切なご助言を賜りました。感謝申し上げます。実装にあたって、株式会社 NTT 研究所研究員の山村氏、宮澤氏には献身的に有益なコメントを頂戴しました。そのご助力に感謝の意を表します。次に、研究室の同輩である田中氏と吉村氏には、日常において迷惑をかけることもありましたが、それでも筆者を見捨てずに共に研究を進める中で有益な助言をして頂いたことを感謝いたします。同学の先輩である西村氏、福原氏、高橋氏並びに、準備会の先輩である笠松氏、前学部時代の同輩である古田氏、友人である今川氏、橋本氏の各位には社会人としてご多忙の中、不躰な筆者に対して専門的な視点からの指摘や論理的なアドバイスを頂き、本研究の全体的な質が向上しました。同学の後輩であった鈴木氏、後輩である生川氏、前学部時代の後輩である谷口氏にも学業が多忙な時期にも関わらずに、多角的な指摘を頂きました。論文の添削を手伝ってくれた両親と弟にも感謝の意を表します。また、筆者のデスクに彩りを与えて頂いたゆんみ氏、元気を与えて頂いたりり氏にも心より感謝いたします。最後に長年に渡り、共にいる納谷氏には今回、多大なご迷惑とご心配をおかけしながらも精神的なサポートを献身的にして頂きました。深い感謝を申し上げます。その他多くの方々のご支援により、この論文が完成しました。改めてお礼申し上げます。

## 参考文献

- [1] NIST: PQC Standardization Process: Announcing Four Candidates to be Standardized, Plus Fourth Round Candidates:  
<https://www.nist.gov/news-events/news/2022/07/pqc-standardization-process-announcing-four-candidates-be-standardized-plus>  
(2023 年 09 月 10 日 閲覧).
- [2] Cryptology ePrint Archive: Masking Dilithium: Efficient Implementation and Side-Channel Evaluation:  
<https://eprint.iacr.org/2019/394>  
(2022 年 01 月 15 日 閲覧).
- [3] pathlock: Data Masking Techniques and Why You Need Them:  
<https://pathlock.com/learn/5-data-masking-techniques-and-why-you-need-them>  
(2023 年 10 月 02 日 閲覧).
- [4] 青野良範, 安田雅哉: 格子暗号解読のための数学的基礎, 近代科学社, 2019.
- [5] 縫田光司: 耐量子計算機暗号, 森北出版, 2020.
- [6] 宮地充子: 代数学から学ぶ暗号理論, 日本評論社, 2019.
- [7] JETRO: 米国における量子コンピュータの現状:  
<https://www.jetro.go.jp/ext-images/-Reports/02/2019/d6b7b154bf6af3b9/201903rpnny.pdf>  
(2023 年 11 月 13 日 閲覧).
- [8] AFP BBNews: 中国の量子コンピューター, 世界最速スパコンで 6 億年要する計算を 200 秒で完了:  
<https://www.afpbb.com/articles/-/3319754>  
(2023 年 12 月 02 日 閲覧).
- [9] IBM Newsroom: IBM, 400 量子ビット超えの量子プロセッサと次世代 IBM Quantum System Two

- を発表:  
<https://jp.newsroom.ibm.com/2022-11-10-IBM-Unveils-400-Qubit-Plus-Quantum-Processor-and-Next-Generation-IBM-Quantum-System-Two>  
(2023 年 12 月 02 日 閲覧).
- [10] 富士通株式会社, 理化学研究所: 超伝導量子コンピュータを開発し, 量子シミュレータと連携可能なプラットフォームを提供:  
<https://www.riken.jp/pr/news/2023/20231005-2/index.html>  
(2023 年 12 月 02 日 閲覧).
- [11] IBM: 東京大学と日本 IBM, 127 量子ビットのプロセッサを搭載した量子コンピューターの導入に合意:  
<https://jp.newsroom.ibm.com/2023-04-21-The-University-of-Tokyo-and-IBM-agree-to-install-a-quantum-computer-with-127-qubit-processor>  
(2023 年 12 月 02 日 閲覧).
- [12] Hindawi: Module-LWE-Based Key Exchange Protocol Using Error Reconciliation Mechanism:  
<https://www.hindawi.com/journals/scn/2022/8299232/>  
(2023 年 12 月 02 日 閲覧).
- [13] Shi Bai: CRYSTALS-Dilithium Algorithm Specifications and Supporting Documentation:  
<https://pq-crystals.org/dilithium/data/dilithium-specification-round3-20210208.pdf>  
(2022 年 01 月 15 日 閲覧).
- [14] CRYPTREC 暗号技術調査ワーキンググループ (耐量子計算機暗号): CRYPTREC 暗号技術ガイドライン (耐量子計算機暗号):  
<https://www.cryptrec.go.jp/report/cryptrec-gl-2004-2022.pdf>  
(2023 年 10 月 02 日 閲覧).
- [15] Adeline Langlois, Damien Stehle: Worst-Case to Average-Case Reductions for Module Lattices:  
<https://eprint.iacr.org/2012/090.pdf>  
(2023 年 12 月 02 日 閲覧).
- [16] ZaHyun Koo: Reduction from Module-SIS to Ring-SIS Under Norm Constraint of Ring-SIS:  
<https://eprint.iacr.org/2020/588.pdf>  
(2023 年 12 月 02 日 閲覧).
- [17] Elena Colette Bakos Lang: Worst-Case to Average-Case Reductions for the SIS Problem Tightness and Security:  
<https://uwspace.uwaterloo.ca/bitstream/handle/10012/14832/BakosLang-Elena.pdf?sequence=3&isAllowed=y>  
(2023 年 12 月 02 日 閲覧).
- [18] Github: Distributions.jl: <https://github.com/JuliaStats/Distributions.jl>  
(2023 年 12 月 25 日 閲覧).
- [19] NIST: SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES:  
<https://csrc.nist.gov/publications/fips/fips140-1/fips1401.html>  
(2023 年 12 月 31 日 閲覧).
- [20] open quantum safe: HOME:  
<https://openquantumsafe.org/>

(2023 年 12 月 31 日 閲覧).

[21] 西野友年: 今度こそわかる量子コンピューター, 講談社, 2019.

[22] 深田萌絵: 量子コンピュータの衝撃, 宝島社, 2020.

[23] SpringerLink: Random Oracles in a Quantum World:

<https://link.springer.com/chapter/10.1007/978-3-642-25385-0-3>

(2023 年 12 月 30 日 閲覧).

[24] 結城浩: 暗号技術入門第 3 版秘密の国のアリス, SB クリエイティブ, 2015.