Lattice-Based Signature from Key Consensus

Leixiao Cheng\textsuperscript{1}, Boru Gong\textsuperscript{2} and Yunlei Zhao\textsuperscript{2}

\textsuperscript{1} School of Mathematical Sciences, Fudan University, Shanghai, China
\textsuperscript{2} School of Computer Science, Fudan University, Shanghai, China

Abstract. Given the current research status in lattice-based cryptography, it is commonly suggested that lattice-based signature could be subtler and harder to achieve. Among them, Dilithium [DLL\textsuperscript{+}17, LDK\textsuperscript{+}17] is one of the most promising signature candidates for the post-quantum era, for its simplicity, efficiency, small public key size, and resistance against side channel attacks. The design of Dilithium is based on a list of pioneering works (e.g., [Lyu09, Lyu12, BG14]), and has very remarkable performance by very careful and comprehensive optimizations in implementation and parameter selection. Whether better trade-offs on the already remarkable performance of Dilithium can be made is left in [CRYSTALS] as an interesting open question.

In this work, we provide new insights in interpreting the design of Dilithium, in terms of key consensus previously proposed in the literature for key encapsulation mechanisms (KEM) and key exchange (KEX). Based on the deterministic version of the optimal key consensus with noise (OKCN) mechanism, originally developed in [JZ16] for KEM/KEX, we present signature from key consensus with noise (SKCN), which could be viewed as generalization and optimization of Dilithium. The construction of SKCN is generic, modular and flexible, which in particular allows a much broader range of parameters for searching better tradeoffs among security, computational efficiency, and bandwidth. For example, on the recommended parameters, compared with Dilithium our SKCN scheme is more efficient both in computation and in bandwidth, while preserving the same level of post-quantum security. In addition, using the same routine of OKCN for both KEM/KEX and digital signature eases (hardware) implementation and deployment in practice, and is useful to simplify the system complexity of lattice-based cryptography in general.

Keywords: Digital signature · Module-LWE · Module-SIS · post-quantum cryptography

1 Introduction

Over the last decades, lattice have emerged as a very attractive foundation for cryptography. Ever since the seminal work of Ajtai [Ajt96] connecting the average-case complexity of lattice problems to their complexity in the worst case, there has been an intriguing and fruitful efforts to base cryptographic schemes on worst-case lattice assumptions. In addition to their unique theoretical niche, lattice-based schemes enjoy many potential advantages: their asymptotic efficiency and conceptual simplicity (usually requiring only linear operations on small integers); their resistance so far to cryptanalysis by quantum algorithms; and the guarantee that their random instances are “as hard as possible” [Reg09, BLP+13].

Given the importance of digital signature scheme in modern cryptography, it is natural to consider building practical and provably secure digital signature schemes based on lattice assumptions. Generally speaking, lattice-based signature schemes are designed by following either of the following paradigms: hash-and-sign paradigm [DH76, BR93, GPV08], and Fiat-Shamir heuristic [FS87, Lyu09, DDL13]. Nevertheless, given the current research status in lattice-based cryptography, it is commonly suggested that lattice-based signature could be subtler and harder to achieve. For instance, there are more than twenty submissions of lattice-based key encapsulation mechanisms to NIST post-quantum cryptography (NIST-PQC), but only five lattice-based signature...
submissions [NIST]. Among them, Falcon [PFH+17], and pqNTRUSign [ZCHW17] follow the hash-and-sign paradigm; Dilithium [DLL+17] and qTESLA [BAA+17] follow the Fiat-Shamir heuristic. Now, Dilithium [DLL+17], qTESLA [BAA+17] and Falcon [PFH+17] are in the second round submissions of NIST-PQC.

In this work, we focus on the study of Dilithium [DLL+17, LDK+17]. Dilithium is one of the best lattice-based signature schemes that follow the Fiat-Shamir paradigm, and is one of the most promising lattice-based signature candidates. Some salient features of Dilithium include: simplicity (both for the algorithmic design and for the algebraic structure of the underlying lattice), efficiency, small public key size, and resistance against side channel attacks. Its design is based on a list of pioneering works (e.g., [Lyu09, Lyu12, BG14] and more), with very careful and comprehensive optimizations in implementation and parameter selection. Whether better trade-offs on the already remarkable performance of Dilithium can be made is left in [CRYSTALS] as an interesting open question.

1.1 Our contributions

In this work, we present generalization and optimization of Dilithium. This is enabled by new insights in interpreting the design of Dilithium, in terms of symmetric key consensus previously proposed in the literature for achieving key encapsulation mechanisms (KEM) and key exchange (KEX) [Reg09, LPR10, LP11, DKL12, Pei14, BCD+16, JZ16]. Based on the deterministic version of the optimal key consensus with noise (OKCN) mechanism, originally developed in [JZ16] for highly practical KEM/KEX schemes, we present signature from key consensus with noise (SKCN). The construction of SKCN is generic, modular and flexible, which in particular allows a much broader range of parameters.

We made efforts to thoroughly search and test a large set of parameters in order to achieve better trade-offs among security, efficiency, and bandwidth. On the recommended parameters, compared with Dilithium our SKCN scheme is more efficient both in computation and in bandwidth, while preserving the same level of post-quantum security. This work also further justifies and highlights the desirability of OKCN, originally developed in [JZ16] for highly practical KEM/KEX, as the same routine can be used for both KEM/KEX and digital signature, which eases (hardware) implementation and deployment in practice, and is useful to simplify the system complexity of lattice-based cryptography in general.

2 Preliminaries

For any real number $x \in \mathbb{R}$, let $\lfloor x \rfloor$ denote the largest integer that is no more than $x$, and $\lceil x \rceil := \lfloor x + 1/2 \rfloor$. For any $i, j \in \mathbb{Z}$ such that $i < j$, denote by $[i, j]$ the set of integers $\{i, i+1, \cdots, j-1, j\}$. For the positive integers $r, \alpha > 0$, let $r \mod \alpha$ denote the unique integer $r' \in [0, \alpha - 1]$ such that $\alpha | (r' - r)$, and let $r \mod^\pm \alpha$ denote the unique integer $r'' \in [-\lfloor \frac{\alpha - 1}{2} \rfloor, \lceil \frac{\alpha + 1}{2} \rceil]$ such that $\alpha | (r'' - r)$. For a positive integer $q$ and an element $x \in \mathbb{Z}_q$, we write $|x|_q$ for $|x| \mod q$, and let $|x|_q$ denote the absolute value of $x \mod^\pm q$. For every $a = \sum_{i=0}^{n-1} a_i \cdot \underaccent{bar}{x}^i \in \mathcal{R}_q$, $a_i \in \mathbb{Z}_q$, define $\text{Power2Round}_{q,d}(a) \overset{\text{def}}{=} \sum a'_i \cdot \underaccent{bar}{x}^i$, where $a'_i \overset{\text{def}}{=} (a_i - (a_i \mod^\pm 2^d)) / 2^d$.

For a finite set $S$, $|S|$ denotes its cardinality, and $x \leftarrow S$ denotes the operation of picking an element uniformly at random from the set $S$. We use standard notations and conventions below for writing probabilistic algorithms, experiments and interactive protocols. For an arbitrary probability distribution $\mathcal{D}$, the notation $x \leftarrow \mathcal{D}$ denotes the operation of picking an element according to the pre-defined distribution $\mathcal{D}$. We say that a positive function $f(\lambda) > 0$ is negligible in $\lambda$, if for every $c > 0$ there exists a positive $\lambda_c > 0$ such that $f(\lambda) < 1/\lambda^c$ for all $\lambda > \lambda_c$. 


2.0.1 Digital signature scheme

A digital signature scheme II consists of three probabilistic polynomial-time algorithms (KeyGen, Sign, Verify). KeyGen is the key generation algorithm that, on input the security parameter $\lambda$, outputs $(pk, sk)$. Sign is the signing algorithm that, on input the secret key $sk$ as well as the message $\mu \in \{0, 1\}^*$ to be signed, outputs the signature $\sigma$. Verify is the deterministic verification algorithm that, on input the public key $pk$ as well as the message/signature pair $(\mu, \sigma)$, outputs $b \in \{0, 1\}$, indicating whether it accepts the incoming $(\mu, \sigma)$ as a valid one or not. We say a signature scheme $\Pi = (\text{KeyGen}, \text{Sign}, \text{Verify})$ is correct, if any sufficiently large $\lambda$, any $(pk, sk) \leftarrow \text{KeyGen}(\lambda)$ and any $\mu \in \{0, 1\}^*$, it holds

$$\Pr[\text{Verify}(pk, \mu, \text{Sign}(sk, \mu)) = 1] = 1.$$ 

2.0.2 (S)EU-CMA

The security for a signature scheme $\Pi = (\text{KeyGen}, \text{Sign}, \text{Verify})$, is defined in the following security game between a challenger and an adversary $A$.

- Setup. Given $\lambda$, the challenger runs $(pk, sk) \leftarrow \text{KeyGen}(\lambda)$. The public key $pk$ is given to adversary $A$, whereas the secret key $sk$ is kept in private.

- Challenge. Suppose $A$ makes at most $q_s$ signature queries. Each signature query consists of the following steps: (1) $A$ adaptively chooses the message $\mu_i \in \{0, 1\}^*$, $1 \leq i \leq q_s$, based upon its entire view, and sends $\mu_i$ to the signer; (2) Given the secret key $sk$ as well as the message $\mu_i$ to be signed, the challenger generates and sends back the associated signature, denoted $\sigma_i$, to $A$.

- Output. Finally, $A$ outputs a pair of $(\mu, \sigma)$, and wins if (1) $\text{Verify}(pk, \mu, \text{Sign}(\mu, \sigma)) = 1$ and (2) $(\mu, \sigma) \not\in \{(\mu_1, \sigma_1), \cdots, (\mu_{q_s}, \sigma_{q_s})\}$.

We say the signature scheme $II$ is strongly existentially unforgeable under adaptive chosen-message attack, if the probability that every p.p.t. attacker $A$ wins in the foregoing game is negligible. A weaker model, i.e., the EU-CMA model, could be define by requiring that $A$ wins if and only if (1) $\text{Verify}(pk, \mu, \sigma) = 1$ and (2) $\mu \not\in \{\mu_1, \mu_2, \cdots, \mu_{q_s}\}$. Then $II$ is called (standard) existentially unforgeable under adaptive chosen-message attack, if no efficient adversary can win in this weaker game with non-negligible probability.

2.0.3 Module-LWE and Module-SIS

In this work, we always have $n = 256$ and $q = 1952257$. Also, let $R$ and $R_q$ denote the rings $\mathbb{Z}[x]/(x^n + 1)$ and $\mathbb{Z}_q[x]/(x^n + 1)$, respectively. For the element $w = \sum_{i=0}^{n-1} w_i x^i \in R$, its $\ell_{\infty}$-norm is defined as $\|w\|_{\infty} := \max \|w_i\|_{\infty}$. Likewise, for the element $w = (w_1, \cdots, w_k) \in R^k$, its $\ell_{\infty}$-norm is defined as $\|w\|_{\infty} := \max \|w_i\|_{\infty}$. In particular, when the other parameters are clear from the context, let $S_q \subseteq R$ denote the set of elements $w \in R$ such that $\|w\|_{\infty} \leq \eta$.

The hard problems underlying the security of our signature scheme are Module-LWE (MLWE), Module-SIS (MSIS) (as well as a variant of MSIS problem). They were well studied in [LS15] and could be seen as a natural generalization of the Ring-LWE [LPR10] and Ring-SIS problems [LM06, PR06], respectively. Fix the parameter $\ell \in \mathbb{N}$. The Module-LWE distribution induced by $s \in R_q^k$ is the distribution of the random pair $(a_i, b_i)$ over the support $R_q^\ell \times R_q$, where $a_i \leftarrow R_q^\ell$ is taken uniformly at random, and $b_i := a_i^T s + \epsilon_i$ with $\epsilon_i \leftarrow S_q$ fresh for every sample. Given arbitrarily many samples drawn from the Module-LWE distribution induced by $s \leftarrow S_q^k$, the (search) Module-LWE problem asks to recover $s$. And the associated Module-LWE assumption states that given $A \leftarrow R_q^{k \times \ell}$ and $b := As + \epsilon$ where $k = \text{poly}(\lambda)$ and $(s, \epsilon) \leftarrow S_q^k \times S_q^k$, no
efficient algorithm can succeed in recovering $s$ with non-negligible probability, provided that the parameters are appropriately chosen.

Fix $p \in [1, \infty]$. Given $A \leftarrow R_{q}^{h \times t}$ where $h = \text{poly}(\lambda)$, the Module-SIS problem (in $\ell_{p}$-norm) parameterized by $\beta > 0$ asks to find a “short” yet nonzero pre-image $x \in R_{q}^{t}$ in the lattice determined by $A$, i.e., $x \neq 0$, $A \cdot x = 0$ and $\|x\| \leq \beta$. And the associated Module-SIS assumption (in $\ell_{p}$-norm) states that no probabilistic polynomial-time algorithm can find a feasible pre-image $x$ with non-negligible probability, provided that the parameters are appropriately chosen. In the literature, the module-SIS problem in Euclidean norm, i.e., $p = 2$, is well-studied; nevertheless, in this work, we are mostly interested in the Module-SIS problem/assumption in $\ell_{\infty}$-norm, i.e., $p = \infty$.

### 2.0.4 Hashing

As is in [DLL+17, LDK+17], when the other related parameters are clear from the context, for every positive integer $w > 0$, let $B_{w} := \{x \in R \mid \|x\|_{\infty} = 1, \|x\|_{1} = w\} \subseteq R$. In this work, we always have $w = 60$, since the set $B_{60} \subseteq R$ is of size $2^{60} \cdot \left(\frac{n}{60}\right) \approx 2^{256}$ (recall that $n = 256$ by default in this work). Let $H : \{0, 1\}^{*} \rightarrow B_{60}$ be a hash function that is modeled as a random oracle in this work. In practice, to pick a random element in $B_{60}$, we can use an inside-out version of Fisher-Yates shuffle.

### 2.0.5 Extendable Output Function

The notion of extendable output function follows that of [DLL+17, LDK+17]. An extendable output function $\text{Sam}$ is a function on bit string in which the output can be extended to any desired length, and the notation $y \in S := \text{Sam}(x)$ represents that the function $\text{Sam}$ takes as input $x$ and then produces a value $y$ that is distributed according to the pre-defined distribution $S$ (or according to the uniform distribution over the pre-defined set $S$). The whole procedure is deterministic in the sense that for a given $x$ will always output the same $y$, i.e., the map $x \mapsto y$ is well-defined. For simplicity we always assume that the output distribution of $\text{Sam}$ is perfect, whereas in practice it will be implemented by using some cryptographic hash functions (which are modelled as random oracle in this work) and produce an output that is statistically close to the perfect distribution.

### 3 Building Tools of SKCN

In this section, we first propose the notion of deterministic symmetric key consensus (DKC); then we construct and analyze a concrete DKC instance, i.e., the deterministic symmetric key consensus with noise (DKCN), which is a variant of the optimal key consensus with noise (OKCN) scheme presented in [JZ16]. Based on DKCN, we then define several algorithms/tools, and develop some of their properties. These algorithms will serve as the building tools for our signature scheme to be introduced in Section 4.

Note that although all these algorithms/tools proposed in this section are defined with respect to the finite field $\mathbb{Z}_{q}$ for some positive rational prime $q$, they could be naturally generalized to vectors (as well as the ring $R_{q}$) in the component-wise manner.

**Definition 1.** A DKC scheme $DKC = (\text{params}, \text{Con}, \text{Rec})$, is specified as follows.

- $\text{params} = (q, k, g, d, aux)$ denotes the system parameters, where $q, k, g, d$ are positive integers satisfying $2 \leq k, g \leq q, 0 \leq d \leq \left[ \frac{2}{3}\right]$, and $aux$ denotes some auxiliary values that are usually determined by $(q, k, g, d)$ and could be set to be a special symbol $\emptyset$ indicating “empty”.
• \((k_1,v) \leftarrow \text{Con}(\sigma_1, \text{params})\): On input \((\sigma_1 \in \mathbb{Z}_q, \text{params})\), the deterministic polynomial-time conciliation algorithm \(\text{Con}\) outputs \((k_1,v)\), where \(k_1 \in \mathbb{Z}_k\) is the shared-key, and \(v \in \mathbb{Z}_g\) is a hint signal that will be publicly delivered to the communicating peer to help the two parties reach consensus.

• \(k_2 \leftarrow \text{Rec}(\sigma_2, v, \text{params})\): On input \((\sigma_2 \in \mathbb{Z}_q, v, \text{params})\), the deterministic polynomial-time reconciliation algorithm \(\text{Rec}\) outputs \(k_2 \in \mathbb{Z}_k\).

A DKC scheme is correct, if \(k_1 = k_2\) for any \(\sigma_1, \sigma_2 \in \mathbb{Z}_q\) such that \(|\sigma_1 - \sigma_2|_q \leq d\).

Next, we develop a concrete instance of DKC, i.e., the rounded symmetric key consensus with noise (DKCN) depicted in Algorithm 1. Note that by Theorem 1, as a concrete DKC, DKCN itself is correct, provided that parameters are appropriately set.

Algorithm 1 DKCN: Deterministic Symmetric KC with Noise

1: \(\text{params} := (q, k, g, d, \text{aux} = \emptyset)\)
2: \(\text{procedure Con}(\sigma_1, \text{params})\)
3: \(v := k\sigma_1 \mod q\)
4: \(\text{if } k\sigma_1 - v = kq \text{ then}\)
5: \(k_1 := 0\)
6: \(\text{else}\)
7: \(k_1 := (k\sigma_1 - v)/q\)
8: \(\text{end if}\)
9: \(\text{return } (k_1, v)\)
10: \(\text{end procedure}\)
11: \(\text{procedure Rec}(\sigma_2, v, \text{params})\)
12: \(k_2 := \lfloor (k\sigma_2 - v)/q \rfloor \mod k\)
13: \(\text{return } k_2\)
14: \(\text{end procedure}\)

Theorem 1. When \(k \geq 2, g \geq 2\) and \(2kd < q\), the DKCN scheme \((\text{params}, \text{Con}, \text{Rec})\) depicted in Algorithm 1 is correct.

Before proving theorem 1, we introduce a lemma that was proposed in [JZ16].

Lemma 1 ([JZ16]). For any \(x, y, t, l \in \mathbb{Z}\) where \(t \geq 1\) and \(l \geq 0\), if \(|x - y|_t \leq l\), then there exists \(\theta \in \mathbb{Z}\) and \(\delta \in [-l, l]\) such that \(x = y + \theta t + \delta\).

Proof of Theorem 1. Suppose \(|\sigma_1 - \sigma_2|_t \leq d\). Then by Lemma 1, there exist \(\theta \in \mathbb{Z}\) and \(\delta \in [-l, l]\) such that \(\sigma_2 = \sigma_1 + \theta q + \delta\). From Line 3 to 7 in Algorithm 1, we know that there exists \(\theta' \in \mathbb{Z}\) such that \(k\sigma_1 = (k_1 + k\theta')q + v\). Taking these into the formula of \(k_2\) in \(\text{Rec}\) (Line 12), we have

\[ k_2 = \lfloor (k\sigma_2 - v)/q \rfloor \mod k = \lfloor k(\sigma_1 + \theta q + \delta)/q - v/q \rfloor \mod k = \lfloor k_1 + k\delta/q \rfloor \mod k. \]

It follows from \(2kd < q\) that \(|k\delta/q| \leq kd/q < 1/2\), making \(k_2 = k_1\).

Based on DKCN, we then present several algorithms and some of their properties.
In this section, we propose our signature scheme SKCN, which is defined in the module lattice, and can be proven to be strongly existentially unforgeable under adaptive chosen-message attacks in the quantum random oracle model.

SKCN could be seen as a generalization and optimization of Dilithium with the aid of DKCN, and its correctness and security roughly follows from that of Dilithium as well.

---

**Proposition 1.** For every \( r, z \in \mathbb{Z}_q \) such that \( \|z\|_\infty < \lfloor q/(2k) \rfloor \), we have

\[
\text{UseHint}_{q,k}(\text{MakeHint}_{q,k}(z, r)) = \text{HighBits}_{q,k}(r + z).
\]

**Proof.** The outputs of \((r_1, r_0) \leftarrow \text{Con}(r), (r'_1, r'_0) \leftarrow \text{Con}(r + z)\) satisfy \(0 \leq r_1, r'_1 < k\), and \(\|r_0\|_\infty, \|r'_0\|_\infty \leq q/2\). Since \(\|z\|_\infty < \lfloor q/(2k) \rfloor\), by Theorem 1, we have \(\text{Rec}(r, r'_0) = r'_1 = \text{HighBits}_{q,k}(r+z)\). Let \(h = \text{MakeHint}_{q,k}(z, r)\). Since \(r'_1 = \text{Rec}(r, r'_0) = \lfloor (kr-r'_0)/q \rfloor \mod k = \lfloor r_1 + (r_0-r'_0)/q \rfloor \mod k \in \{r_1-1, r_1, r_1+1\}\). When \(r_0 > 0\), we have \(\text{Rec}(r, r'_0) \in \{r_1, r_1+1\}\). When \(r_0 < 0\), we have \(\text{Rec}(r, r'_0) \in \{r_1-1, r_1\}\). Recall that by definition, \(h = 0\) if and only if \(r_1 = r'_1\). The correctness of \(\text{HighBits}_{q,k}(r+z) = r'_1 = \text{Rec}(r, r'_0) = \text{UseHint}_{q,k}(h, r)\) is thus established.

---

**Proposition 2.** For \(r'_1 \in \mathbb{Z}_k\), \(r \in \mathbb{Z}_q\), \(h \in \{0, 1\}\), if \(r'_1 = \text{UseHint}_{q,k}(h, r)\), then \(\|r - [q \cdot r'_1/k]\|_\infty \leq q/k + 1/2\).

**Proof.** It is routine to see that for \((r_1, r_0) \leftarrow \text{Con}(r)\), we have \(r_1 \in \mathbb{Z}_k, r_0 \in (-q/2, q/2)\), and there exists \(\theta \in \{0, 1\}\) such that \(k \cdot r = (r_1 + k\theta) \cdot q + r_0\). If \(h = 0\), then \(r'_1 = r_1\), and hence \(\|r - [q \cdot r'_1/k]\|_\infty \leq q/(2k) + 1/2\). If \(h = 1\) and \(r_0 < 0\), then \(r'_1 = (r_1 + 1) \mod k\), and hence \(\|r - [q \cdot r'_1/k]\|_\infty \leq q/k + 1/2\). Finally, if \(h = 1\) and \(r_0 > 0\), then \(r'_1 = (r_1 - 1) \mod k\), and therefore \(\|r - [q \cdot r'_1/k]\|_\infty \leq q/k + 1/2\).

---

**Proposition 3.** For \(r, z \in \mathbb{Z}\) such that \(\|z\|_\infty \leq U\). If \(\|r'_0\|_\infty < q/2 - kU\) where \((r_1, r_0) \leftarrow \text{Con}(r), (r'_1, r'_0) \leftarrow \text{Con}(r+z)\), then \(r_1 = r'_1\).

**Proof.** Since \(k \cdot r = q \cdot (r_1 + k\theta) + r_0\), \(\|r_0\|_\infty < q/2\) and \(k \cdot (r+z) = q \cdot (r'_1 + k\theta') + r'_0\), it is easy to verify \(r_1 = [kr/q] \mod k = [k(r + z)/q] \mod k = [r'_1 + (r'_0 - k\varphi)/q] \mod k = r'_1\).
4.1 Description of SKCN

Our key generation, signing, and verification algorithms are fully described in Algorithms 2, 4 and 3, respectively.

Algorithm 2 Key Generation Algorithm

Input: \( t^A \)
Output: \( (pk = (\rho, t_1), sk = (\rho, s, e, t)) \)
1: \( \rho, t_1^A \leftarrow \{0, 1\}^{256} \)
2: \( A \in \mathcal{R}_{q \times t} := \text{Sam}(\rho) \)
3: \( \{s, e\} \in S_{t}^{\rho} \times S_{h}^{\rho} := \text{Sam}(\rho') \)
4: \( t_1 := \text{Power2Round}_{q,d}(t) \)
5: \( \text{return} (pk = (\rho, t_1), sk = (\rho, s, e, t)) \)

Algorithm 3 Verification Algorithm

Input: \( b \in \{0, 1\} \)
Output: \( \sigma \)
1: \( \text{Input:} \)
2: \( \text{Output:} \)
3: \( \text{Algorithm 4 The Signing Algorithm} \)

Input: \( \mu \in \{0, 1\}^* \)
Output: \( \sigma = (z, c, h) \)
1: \( A \in \mathcal{R}_{q \times t} := \text{Sam}(\rho) \)
2: \( t_1 := \text{Power2Round}_{q,d}(t) \)
3: \( t_0 := -t_1 \times 2^d \)
4: \( r \leftarrow \{0, 1\}^{256} \)
5: \( y \in S_{\ell}^{\rho} := \text{Sam}(r) \)
6: \( w \triangleq A^y \)
7: \( w_1 := \text{HighBits}_{q,k}(w) \)
8: \( c := H(\rho, t_1, [q \cdot w_1/k], \mu) \)
9: \( z := y + ce \)
10: \( (r_1, r_0) := \text{Con}(w - ce) \)
11: \( \text{Restart if} \ |z|_\infty \geq |q/k| \cdot U \text{ or } |r_0|_\infty \geq q/2 - kU \) or \( r_1 \neq w_1 \)
12: \( h := \text{MakeHint}_{q,k}(-ct_0, Az - ct + c) \)
13: \( \text{Restart if} \ |ct_0|_\infty \geq |q/2k| \text{ or the number of } 1's \text{ in } h \text{ is greater than } \omega \)
14: \( \text{return} (z, c, h) \)

Algorithm 5 The Simulator

Input: \( \mu \in \{0, 1\}^* \)
Output: \( (\rho, t_1, t_0) \)
1: \( A \in \mathcal{R}_{q \times t} := \text{Sam}(\rho) \)
2: \( (z, c) \leftarrow S_{\ell}^{(q,k)-U} \times B_{60} \)
3: \( (r_1, r_0) := \text{Con}(Az - ct) \)
4: \( \text{Restart if} \ |r_0|_\infty \geq q/2 - kU \)
5: \( \text{if} \ H \text{ has already been defined on } (\rho, t_1, [q \cdot r_1/k], \mu) \text{ then} \)
6: \( \text{Abort} \)
7: \( \text{else} \)
8: \( \text{Program } H(\rho, t_1, [q \cdot r_1/k], \mu) = c \)
9: \( \text{end if} \)
10: \( h := \text{MakeHint}_{q,k}(-ct_0, Az - ct + c) \)
11: \( \text{Restart if} \ |ct_0|_\infty \geq |q/2k| \text{ or the number of } 1's \text{ in } h \text{ is greater than } \omega \)
12: \( \text{return} (z, c, h) \)

4.1.1 Practical Implementation

When we implement SKCN with our recommended parameter set (cf. Table 1), several improvements that are similar to [DLL\textsuperscript{+} 17, LDK\textsuperscript{+} 17] are made, so as to improve its efficiency. Specifically, the sign algorithm in our implementation is deterministic in nature which is similar to that of Dilithium [LDK\textsuperscript{+} 17]. This is achieved by adding some new seeds (tr, key) into the secret key sk; thus, the random nonce y in the sign algorithm could be obtained via a pseudorandom string, which is obtained by extending the hash value of (tr, key), the message to be signed, and a counter. As a result, the \( t_1 \) in sk is no longer necessary, making \( sk = (\rho, tr, key, s, e, t_0) \). This minor modification can improve the efficiency of the sign algorithm significantly, and shorten the size of sk. Our implementation code is available at https://github.com/mulansig.
4.2 Correctness Analysis

In SKCN, the key generation algorithm first chooses a random 256-bit seed $\rho$ and expands it into a matrix $A \leftarrow \mathcal{R}_q^{h \times t}$ by an extendable output function $\text{Sam}(\cdot)$. The crucial component in the secret key is $(s, e) \in \mathcal{R}_q^h \times \mathcal{R}_q^t$, and each coefficient of $s$ (resp., $e$) is drawn uniformly at random from the set $[-\eta, \eta]$ (resp., $[-\eta', \eta']$). Finally, we compute $t = A \cdot s + e \in \mathcal{R}_q^t$. The public key is $pk = (\rho, t_1)$ where $t_1 = \text{Power2Round}_{q,a}(t)$, and the associated secret key is $sk = (\rho, s, e, t)$.

Given the secret key $sk = (\rho, s, e, t)$ as well as the message $\mu \in \{0, 1\}^*$ to be signed, the signing algorithm first recovers the public matrix $A \in \mathcal{R}_q^{h \times t}$ via the random seed $\rho$ in the secret key. After that, the signing algorithm picks a “short” $y$ from the set $S_{\frac{q}{k}}^{c} \subseteq \mathcal{R}_q^h$ uniformly at random, and computes $w_1 := \text{HighBits}_{q,k}(w)$, where $w := Ay$. Upon input $(\rho, t_1, [q \cdot w_1/k], \mu)$, the random oracle $H(\cdot)$ returns a uniform $c \leftarrow B_q^{\delta}$. After obtaining $c$, the signing algorithm conducts a rejection sampling process to check if every coefficient of $z := y + cs$ is “small” enough, if every coefficient of $r_0$ is “small” enough, and if $c$ is “correct”. If this is the case, the algorithm restarts, until all the requirements are satisfied. We should point out that if $\|ce\|_{\infty} < \omega^U$, then by Proposition 3, the requirement $\|r_0\|_{\infty} < q/2 - kU^U$ forces $r_1 = w_1$. We hope $\|ce\|_{\infty} > \omega^U$ occurs with negligible probability, such that the probability that the check $r_1 = w_1$ fails is negligible as well. In addition, $U$ is chosen such that $\|cs\|_{\infty} \leq U$ holds with overwhelming probability. Furthermore, the function $\text{MakeHint}_{q,k}(\cdot)$ is invoked on input $(-ct_0, w - ce + ct_0)$ to generate the hint $h$, i.e., a binary vector in $\{0, 1\}^h$. The signing algorithm concludes by conducting the remaining two checks, i.e., if $\|ct_0\|_{\infty} < q/2k$ and the number of nonzero elements in $h \in \{0, 1\}^{h \cdot h}$ does not exceed the pre-defined threshold $\omega^C$; otherwise restart is carried out again. Here, the hint $h$ corresponds to the fact that it is $t_1$, not the whole $t = t_1 \cdot 2^d + t_0$ that is contained in the public key. With the hint $h$, we can still carry out the verification, even without $t_0$.

Given the public key $pk = (\rho, t_1)$, the message $\mu \in \{0, 1\}^*$ and the claimed signature $(z, c, h)$, the verifying algorithm first recovers $A \in \mathcal{R}_q^{h \times t}$ via the random seed $\rho$. After that, it computes $w'_1 := \text{UseHint}_{q,k}(h, Az - ct_1 \cdot 2^d)$. If the given $(z, c, h)$ is indeed a honestly generated signature of the incoming message $\mu$, then it is routine to see that every coefficient of $z$ is “small” enough, and the number of 1’s in $h$ is no greater than $\omega^C$; more importantly, we have $\text{HighBits}_{q,k}(Az - ce) = w'_1$ and therefore $c = c'$, where $c' := H(\rho, t_1, [qw'_1/k], \mu)$. The verifying algorithm would accept the input tuple if and only if the foregoing conditions are all satisfied.

Next, we show that our SKCN signature scheme is always correct, provided that the involving parameters are appropriately set. Roughly speaking, the correctness relies heavily on Proposition 1. When the public/private key pair $(pk, sk)$ is fixed, for a valid message/signature pair $(\mu, (z, c, h))$, it suffices to show that $c = c'$. Since $\|ct_0\|_{\infty} < q/2k$ and $Az - ct_1 \cdot 2^d = Ay - ce + ct_0$, it follows directly from Proposition 1 that

$$\text{UseHint}_{q,k}(h, Az - ct_1 \cdot 2^d) = \text{HighBits}_{q,k}(Ay - ce).$$

Given that the signing algorithm forces $\text{HighBits}_{q,k}(Ay - ce) = \text{HighBits}_{q,k}(Ay)$ by rejection sampling, it follows from the following equality that $c = c'$:

$$\text{UseHint}_{q,k}(h, Az - ct_1 \cdot 2^d) = \text{HighBits}_{q,k}(Ay - ce) = \text{HighBits}_{q,k}(Ay).$$

4.3 Recommended Parameters, and Comparison

To improve the time/space efficiency, our SKCN signature scheme could be set asymmetrically, in the sense that as long as the resulting scheme can resist the key-recovery attack, $\eta$ may not equal to $\eta'$. Moreover, the parameter $U$ and $U'$ are carefully chosen such that $\Pr(\|c\|_{\infty} \geq U)$ and $\Pr(\|c\|_{\infty} \geq U')$ are sufficiently small, say they are both small than $2^{-128}$. By default we have $\eta = \eta'$ (and hence $U = U'$).
Table 2: Comparison between SKCN/Dilithium.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>SKCN</th>
<th>Dilithium</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q$</td>
<td>1952257</td>
<td>8380417</td>
<td></td>
</tr>
<tr>
<td>$n$</td>
<td>256</td>
<td>256</td>
<td></td>
</tr>
<tr>
<td>$(h, \ell)$</td>
<td>$(5, 4)$</td>
<td>$(5, 4)$</td>
<td></td>
</tr>
<tr>
<td>$(\eta, \eta')$</td>
<td>$(2, 2)$</td>
<td>$(5, 5)$</td>
<td></td>
</tr>
<tr>
<td>pk size (in byte)</td>
<td>1312</td>
<td>1472</td>
<td></td>
</tr>
<tr>
<td>sk size (in byte)</td>
<td>3056</td>
<td>3504</td>
<td></td>
</tr>
<tr>
<td>sig. size (in byte)</td>
<td>2573</td>
<td>2701</td>
<td></td>
</tr>
<tr>
<td>expected # of repetitions</td>
<td>5.67</td>
<td>6.6</td>
<td></td>
</tr>
<tr>
<td>quantum bit-cost against</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>key recovery attack</td>
<td>128</td>
<td>128</td>
<td></td>
</tr>
<tr>
<td>quantum bit-cost against</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>forgery attack</td>
<td>125</td>
<td>125</td>
<td></td>
</tr>
</tbody>
</table>

The efficiency of the signing algorithm is firmly connected to the expected number of repetitions, which depends on the probabilities that the two rejection sampling steps occur. When some assumption is made with respect to the distribution of $w = Ay \in \mathcal{R}_q^n$, the probability that the first restart occurs is

$$p_1 = \left( \frac{2(\lfloor q/k \rfloor - U) - 1}{2 \lfloor q/k \rfloor - 1} \right)^{\ell n} \cdot \left( \frac{2(\lfloor q/2 \rfloor - kU') - 1}{q} \right)^{h n}.$$  

In regard to the second restart, experiments are carried to estimate the expected number of repetitions, and parameters are chosen such that in the experiments, the second restarts are carried out with probability no more than 1%. In sum, the average number of repetitions is dominated by the probability that the first restart occurs.

To choose the set of recommended parameters for SKCN, the following requirements or goals should be taken into account simultaneously: First, the parameters should be appropriately chosen so as to ensure the correctness of our signature scheme; Second, the involved parameters should be chosen with the goal of achieving 128-bit quantum security; Moreover, the parameters should be chosen such that the expected number of repetitions in the signing algorithm should be as small as possible, so as to ensure the efficiency of the signing algorithm; Finally, the parameters should be chosen such that the sum of the public key size and the signature size should be as minimal as possible.

Under such considerations, we choose the set of recommended parameters for SKCN which is depicted in Table 1. And the quantitative comparison between recommended-SKCN and recommended-Dilithium is summarized in Table 2. Note that for the security issue, the (quantum) security of our recommended parameter set is estimated by following exactly the methodology proposed in [DLL$^+$17, LDK$^+$17].

The strength of SKCN is best described by the foregoing quantitative measures. Roughly speaking, compared with Dilithium, SKCN is more efficient: while preserving the same (quantum) security level as Dilithium does, SKCN has shorter public/secret key, has shorter signature, and runs faster.

### 4.4 Implementation Details

In this subsection, we describe how our SKCN equipped with the set of recommended parameters (cf. Table 1) is implemented in practice. Our implementation can be best described by Algorithms 6 - 8 presented in Appendix A. Roughly speaking, compared with our theoretical design depicted...
in Algorithms 2 - 4, several changes similar to [LDK+17] are made so as to improve the efficiency of SKCN.

### 4.4.1 Deterministic signing procedure

First and foremost, the sign procedure in SKCN is made deterministic by adding a seed to the secret key and using this seed key as well as the message $\mu$ to be signed to produce the nonce $y \in \mathcal{R}_q^4$ via SHAKE-256. Given that repetition is almost necessary to generate a valid signature, we append a counter $\kappa$ into the secret key, which makes the SHAKE-256 output differ with each signing attempt of the same message.

### 4.4.2 NTT operation

As in most lattice-based schemes that are based on operations over algebraic lattice, the underlying ring in SKCN is well chosen so that the multiplication operation could be carried out efficiently via the Number Theoretic Transform (NTT) over the finite field $\mathbb{Z}_q$. To enable the NTT, we need to choose a prime $q$ so that the cyclic group $\mathbb{Z}_q^*$ has an element $\alpha \in \mathbb{Z}_q^*$ of order $2n = 512$, or equivalently, $q \equiv 1 \pmod{512}$. In our recommended parameter,

$$x^n + 1 = x^{256} + 1 \equiv (x - \alpha) \cdot (x - \alpha^3) \cdots (x - \alpha^{511}) \pmod{q},$$

and each polynomial $f \in \mathbb{Z}_q[x]/(x^n + 1)$ could be represented in its NTT form as

$$\hat{f} \overset{\text{def}}{=} (f(\alpha), f(\alpha^3), \ldots, f(\alpha^{511})).$$

The homomorphic property of this conversion enables us to carry out the polynomial multiplication efficiently.

### 4.4.3 Bit packing

Bit-packing is applied in the implementation of SKCN.

Take the binary representation of $t_0 \in \mathcal{R}_q^5$ as an example. Recall that $t_1$ contains 5 polynomials in $\mathbb{Z}_q[x]/(x^{256} + 1)$, and every coefficient belongs to $\{-2^{12} + 1, \ldots, 2^{12}\}$. Therefore, each polynomial coefficient could be represented by 13 bits via a simple translation, and $t_0$ could be represented by $5 \cdot 256 \cdot 13$ bits; in our implementation, they are bit-packed in little-endian byte-order. Similar techniques apply to the polynomial vectors $s, e, t_1, z$.

Also, the public key $(\rho, t_1)$ of SKCN is stored as the concatenation of the bit-packed representations of $\rho$ and $t_1$ in consecutive order. Similar considerations apply to the secret key and the signature.

### 4.4.4 Collision resistant hash

Similar to [LDK+17], the function CRH in Appendix A is a collision resistant hash function. In our implementation, it is instantiated by using the first 384 bits of the SHAKE-256 output.

### 4.4.5 Generation of $A, s, e$

To keep the public key as small as possible, the matrix $A$ is shared in terms of the seed $\rho$: in each procedure, the NTT representation of the matrix $A$, i.e., $\hat{A}$, is extracted from the seed $\rho$ by squeezing SHAKE-256 to obtain a stream of random bytes of arbitrary length. Similarly, part of the secret key $(s, e)$ is obtained from a random seed in the key generation procedure, via the use of SHAKE-256.
Table 3: The concrete hardware/software details for the implementation comparison.

<table>
<thead>
<tr>
<th>Hardware / Software</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating System</td>
<td>Ubuntu 18.04.3 LTS system</td>
</tr>
<tr>
<td>Computer</td>
<td>Lenovo ThinkPad T480S</td>
</tr>
<tr>
<td>CPU</td>
<td>Intel(R) Core(TM) i7-8550U</td>
</tr>
<tr>
<td>Memory</td>
<td>16G</td>
</tr>
<tr>
<td>Implementation of RO</td>
<td>SHA-3</td>
</tr>
<tr>
<td>Compiler</td>
<td>GCC</td>
</tr>
<tr>
<td>Hyperthreading option</td>
<td>On</td>
</tr>
</tbody>
</table>

Table 4: Comparison between SKCN and Dilithium.

<table>
<thead>
<tr>
<th></th>
<th>SKCN</th>
<th>Dilithium</th>
</tr>
</thead>
<tbody>
<tr>
<td>q</td>
<td>1952257</td>
<td>8380417</td>
</tr>
<tr>
<td>n</td>
<td>256</td>
<td>256</td>
</tr>
<tr>
<td>(h, ℓ)</td>
<td>(5, 4)</td>
<td>(5, 4)</td>
</tr>
<tr>
<td>(η, η')</td>
<td>(2, 2)</td>
<td>(5, 5)</td>
</tr>
<tr>
<td>pk size (in byte)</td>
<td>1312</td>
<td>1472</td>
</tr>
<tr>
<td>sk size (in byte)</td>
<td>3056</td>
<td>3504</td>
</tr>
<tr>
<td>sig. size (in byte)</td>
<td>2573</td>
<td>2701</td>
</tr>
<tr>
<td>average # of repetitions</td>
<td>5.7</td>
<td>6.6</td>
</tr>
<tr>
<td>KeyGen cycles</td>
<td>177707</td>
<td>198167</td>
</tr>
<tr>
<td>Sign cycles</td>
<td>859774</td>
<td>1056305</td>
</tr>
<tr>
<td>Verification cycles</td>
<td>191645</td>
<td>201511</td>
</tr>
</tbody>
</table>

4.4.6 Performance comparison

We test both the implementations of SKCN and that of Dilithium under the same software/hardware environment depicted in Table 3. And the quantitative comparison between recommended-SKCN and recommended-Dilithium is summarized in Table 4.

5 Security Analysis of SKCN

In this section, we analyze the security of the SKCN signature scheme. Roughly speaking, the security proof consists of two phases: In Phase I, the behavior of the signing oracle is proven to be statistically indistinguishable from that of an efficient simulator; In Phase II, we show that when the underlying hardness assumption holds, no efficient attacker can forge a valid message/signature pair with non-negligible probability, after interacting with the foregoing simulator polynomially many times.

In the following security proof, we will assume that the public key of SKCN is (ρ, t₁, t₀) instead of (ρ, t₁), which is similar to that of Dilithium [DLL¹⁷⁺, LDK¹⁷⁺].

5.1 Security Proof in Phase I: the Simulator

The simulation of the signature follows from that of [DLL¹⁷⁺]. The associated simulator for SKCN is depicted in Algorithm 5. It should be stressed that we assume the public key is t instead of t₁ as well. It suffices to show that the output of the signing oracle is indistinguishable from that of the simulator. The following two facts plays an essential role for the indistinguishability
proof. First, in the real signing algorithm, we have \( \Pr[z, c] = \Pr[c] \cdot \Pr[y = z - cs | c] \). Since \( \|z\|_\infty < \lfloor q/k \rfloor - U \) and \( \|cs\|_\infty \leq U \) (with overwhelming probability), we know that \( \|z - cs\|_\infty < \lfloor q/k \rfloor \), then \( \Pr[z, c] \) is exactly the same for every such tuple \((z, c)\). Second, when \( z \) does satisfy \( \|\text{LowBits}_{q,k}(w - c)\|_\infty < q/2 - kU' \), then as long as \( \|ce\|_\infty < U' \), we have

\[
\mathbf{r}_1 = \text{HighBits}_{q,k}(w - ce) = \text{HighBits}_{q,k}(w) = w_1
\]

by Proposition 3. Thus the simulator does not need to perform the check whether \( \mathbf{r}_1 = w_1 \) or not, and can always assume that it passes.

With the foregoing facts, it is routine to see that the distribution of the pair \((z, c)\) generated by the simulator is statistically indistinguishable from that of the pair \((z, c)\) generated by the signing oracle.

After that, the simulator computes \( \mathbf{r}_1 \) and programs \( H(\rho, \mathbf{t}_1, \lfloor q \cdot \mathbf{r}_1/k \rfloor, \mu) = c \). The resulting \((z, c)\) output by the simulator is indistinguishable from that of the real signing oracle in the security game, provided that collision occurs with negligible probability.

It remains to show that for each query \( \mu \), the probability that \( H(\rho, \mathbf{t}_1, \lfloor q \cdot \mathbf{r}_1/k \rfloor, \mu) \) was programmed previously is negligible. This follows directly from the following lemma, whose proof is similar to that of [KLS18].

**Lemma 2.** For every \( A \leftarrow R_q^{h \times \ell} \), we have

\[
\Pr[\forall \mathbf{w}_1^*: \exists r \in S_{\lfloor q/k \rfloor - 1} \left[ \text{HighBits}_{q,k}(A\mathbf{y}) = \mathbf{w}_1^* \right] \leq \left( \frac{q/k + 1}{2 \cdot \lfloor q/k \rfloor - 1} \right)^n > 1 - (n/q)^{h\ell}.
\]

**Proof.** Since the polynomial \( x^n + 1 \) splits into \( n \) linear factors modulo \( q \), the probability that for a uniform \( a \leftarrow R_q \), the probability that \( a \) is invertible in \( R_q = \mathbb{Z}_q[x]/(x^n + 1) \) is \((1 - 1/q)^n > 1 - n/q \). Thus the probability that at least one of \( h\ell \) polynomials in \( A \leftarrow R_q^{h \times \ell} \) is invertible is greater than \( 1 - (n/q)^{h\ell} \).

We shall now prove that for all \( A \) that contain at least one invertible polynomial, we will have that for all \( \mathbf{w}_1^* \),

\[
\Pr[\exists r \in S_{\lfloor q/k \rfloor - 1} \left[ \text{HighBits}_{q,k}(A\mathbf{y}) = \mathbf{w}_1^* \right] \leq \left( \frac{q/k + 1}{2 \cdot \lfloor q/k \rfloor - 1} \right)^n,
\]

which establishes the correctness of this lemma.

First, let us only consider the row of \( A \) which contains the invertible polynomial. Call the elements in this row \( [a_1, \ldots, a_\ell] \) and without loss of generality assume that \( a_1 \) is invertible. We want to prove that for all \( \mathbf{w}_1^* \),

\[
\Pr[\exists r \in S_{\lfloor q/k \rfloor - 1} \left[ \text{HighBits}_{q,k}(\sum a_i y_i) = \mathbf{w}_1^* \right] \leq \left( \frac{q/k + 1}{2 \cdot \lfloor q/k \rfloor - 1} \right)^n.
\]

Define \( T \) to be the set containing all the elements \( w \) such that \( \text{HighBits}_{q,k}(w) = w_1^* \). By the definition of \( \text{Con} \) in Algorithm 1, the size of \( T \) is upper-bounded by \((q/k + 1)^n\). Therefore, we can rewrite the above probability as

\[
\Pr[\sum_{i=1}^{\ell} a_i y_i \in T] = \Pr[\sum_{i=2}^{\ell} a_i y_i \leq (q/k + 1)(T - \sum_{i=2}^{\ell} a_i y_i)] \leq \left( \frac{q/k + 1}{2 \cdot \lfloor q/k \rfloor - 1} \right)^n,
\]

where the last inequality follows from the fact that the size of the set \( a_1^{-1}(T - \sum_{i=2}^{\ell} a_i y_i) \) is the same as that of \( T \), and the size of the set \( S_{\lfloor q/k \rfloor - 1}^{\ell} \) is exactly \((2 \cdot \lfloor q/k \rfloor - 1)^n \). \( \square \)

It should be stressed that, both inequalities \( \left( \frac{q/k + 1}{2 \cdot \lfloor q/k \rfloor - 1} \right)^n \ll 2^{-128} \) and \((n/q)^{h\ell} \ll 2^{-128}\) holds for our set of recommended parameters depicted in Table 1.
5.2 Security Proof in Phase II

By applying forking lemma, we can show SKCN is strongly existentially unforgeable under adaptive chosen-message attacks in the random oracle model, provided that the parameters are appropriately chosen and the underlying MLWE and MSIS (in $\ell_\infty$ norm) assumptions hold. However, this proof is not tight, and cannot be directly applied into the quantum setting. In contrast, in this section, we shall develop a quantum reduction of SKCN that is tight in nature by introducing another new underlying hardness assumption for SKCN, as is done in [DLL$^+17$, LDK$^+17$].

As is observed in [DLL$^+17$, LDK$^+17$], no counter-examples of schemes whose security is actually affected by the non-tightness of the reduction has been proposed. The main reason for this absence of counter-examples lies in that there is an intermediate problem which is tightly equivalent, to the UF-CMA security of the signature scheme. What is more, this equivalence still holds even under in quantum settings. Compared with classical hardness problems, this problem is essentially a convolution of the underlying mathematical problem with a cryptographic hash function $H(\cdot)$. As is justified in [LDK$^+17$], as long as there is no relationship between the structure of the math problem and the hash function $H(\cdot)$, solving this intermediate problem is not easier than solving the mathematical problem alone. In our setting, this intermediate problem is called the SelfTargetMSIS problem, which is to be defined later.

5.3 The SelfTargetMSIS Problem, and Quantum Security of SKCN

We follow the definition in [LDK$^+17$]. Assume $H : \{0,1\}^* \rightarrow B_{\ell_0}$ is a cryptographic hash function. For a given adversary $A$, it is given a random $A \leftarrow \mathcal{R}_{q \times \ell}^n$ and access to the quantum random oracle $H(\cdot)$, and is asked to output a pair $(y = (|r,c|_T, \mu))$ such that $0 \leq ||y||_\infty \leq \gamma$, $H(\mu, [I,A] \cdot y) = c$. In other words, the adversary $A$ is asked to solve the SelfTargetMSIS problem.

In this work, let $Adv_{H,h,\ell,\gamma}^{SelfTargetMSIS}(A)$ denote the probability that $A$ solves the given SelfTargetMSIS problem successfully.

Similar to results in [LDK$^+17$], given the similarity between SKCN and Dilithium, it follows from [KLS18] that when $H(\cdot)$ is modeled as quantum random oracle, the probability a given efficient adversary $A$ breaks the SEU-CMA security of SKCN is

$$Adv_{SKCN}^{SUFI-CMA}(A) \leq Adv_h^{MLWE}(B) + Adv_{H,h,\ell+1,\gamma}^{SelfTargetMSIS}(C) + Adv_{h,\ell,\gamma}^{MSIS}(D) + 2^{-254},$$

where $D$ denotes the uniform distribution over $S_q$, and

$$\zeta = \max(|q/k| - U, |q/k| + 1 + 60 \cdot 2^{d-1}), \zeta' = \max(2 \cdot (|q/k| - U), 2 |q/k| + 2).$$

Similar to Dilithium, SKCN is built upon three underlying hardness assumptions: intuitively, the MLWE assumption is needed to protect against key-recovery attack; the SelfTargetMSIS is the assumption upon which new message forgery is based, and the MSIS assumption is needed for strong unforgeability instead of standard unforgeability.

Note that the simulation proof in Section 5.1 holds even in quantum setting; equivalently, if an adversary having quantum access to $H(\cdot)$ and classical access to a signing oracle can produce a forgery of a new message, then there is also an adversary who can produce a forgery after interacting with the simulator defined in Section 5.1. When MLWE assumption holds with the distribution $D$, it remains for us to analyze the following experiment: for an efficient adversary $A$, it is given a random $(A,t)$, and is asked to output a valid message/signature pair $(\mu, (z, c, h))$ such that $||z||_\infty < |q/k| - U, H(\mu, UseHint_{q,k}(h, Az - ct_1 \cdot 2^d)) = c, ||h||_1 \leq \omega$.

It follows from the properties presented in Section 3 that

$$Az - ct_1 \cdot 2^d + u = [q/k] \cdot UseHint_{q,k}(h, Az - ct_1 \cdot 2^d) = Az - ct + u',$$

where $||u'||_\infty \leq ||u||_\infty + ||c \cdot t_1||_\infty \leq 60 \cdot 2^{d-1} + |q/k|$. In sum, to forge a valid message/signature pair means to find $z, c, u', \mu$ such that
6 Concrete Security Analysis

In this section, we analyze how to conduct the concrete security analysis of SKCN. As mentioned previously, we estimate the concrete hardness of SKCN by following the same methodology proposed in [LDK+17], and the result shows our recommended-SKCN achieves the same (quantum) security level as that of recommended-Dilithium.

For our signature schemes, two types of important attacks should be taken into consideration: the key-recovery attack which aims to recover the secret key, with the given associated public key; the forgery attack which tries to forge a signature in the security game of SEU-CMA. To our knowledge, the best known algorithms to implement these two attack both involve the lattice basis reduction as well as the Core-SVP problem.

6.1 Lattice Basis Reduction, BKZ Algorithm, and Core-SVP Problem

Given our recommended parameter, in practice the best known algorithm for finding a “short” nonzero vector in Euclidean lattices is the BKZ algorithm as well as its variants, which outperforms the combinatorial attacks (e.g., the BKW attack [BKW03]) and algebraic attacks (e.g., the Arora-Ge algorithm [AG11]).

In order to solve the SVP problem in \( \ell_2 \)-norm, the BKZ algorithm solves a related yet more generic problem, i.e., the lattice basis reduction problem. Generally speaking, BKZ solves the lattice basis reduction problem by making polynomial calls to a SVP oracle with block-size \( b \). The hardness of lattice basis reduction problem is implied by the following two facts: to obtain a “good” basis, the BKZ algorithm should be equipped with a SVP oracle with a large block-size \( b \); the cost
of implementing the underlying SVP oracle is exponential in the block-size $b$ (in fact, the best known quantum SVP solver [ADPS16] runs in time $\approx 2^{c_Q b}$, where $c_Q = \log_2 \sqrt{13/9} \approx 0.265$).

Since it is relatively difficult to analyze the upper-bound on the number of SVP calls, the CoreSVP model is thus introduced [ADPS16], which identifies the cost of BKZ algorithm with the cost of one single call to an SVP oracle with block-size $b$. This pessimistic estimation implies that similar to [DLL+17, LDK+17], our security analysis is pretty conservative.

### 6.2 Forgery Attack and Module-SIS Problem

As indicated in Section 5.2, the forgery attack could be boiled down to solving the SelfTargetMSIS problem. To solve the SelfTargetMSIS problem, we need either to break the security of $H(\cdot)$, or to solve the MSIS problem with input $(A, t', \beta)$. The hardness of breaking the security of $H(\cdot)$ is guaranteed by the assumption that $H(\cdot)$ is modeled as a random oracle in our security proof and by the fact that the range $B_{\|\cdot\|_6}$ of $H(\cdot)$ is roughly of size $2^{256}$. Hence, we concentrate our analysis on the hardness of the MSIS problem hereafter.

In essence, the MSIS problem could be seen as a variant of the SIS problem in the $\ell_\infty$-norm, which is in sharp contrast to the ordinary SIS problem endowed with the $\ell_2$-norm. Roughly speaking, a “short” vector in the $\ell_\infty$-norm is also a “short” solution in the $\ell_2$-norm, but the converse may not hold. (In fact, for our signature scheme with our recommended parameter, the solution to our SelfTargetMSIS problem has Euclidean length above $q$, whereas the trivial vector $(q, 0, \cdots, 0)^t$ has Euclidean length $q$, but its infinity norm is far from satisfactory.) This indicates that the problem we are confronting is intuitively much harder than the SIS problem in the $\ell_2$-norm.

Since BKZ algorithm works only on the Euclidean lattice, we cannot directly turn the MSIS instance into a Core-SVP instance. Nevertheless, we shall follow the general methodology proposed in [DLL+17, LDK+17] by sticking to using the BKZ algorithm to determine the solution in the infinity norm.

To be specific, we first narrow down the range of our analysis by choosing a subset of $w$ columns, and zeroing the other dimensions of the targeted vector. Then we still seek to choose a desired “short” vector by invoking the BKZ algorithm, but no longer the first one in the output lattice basis, e.g., the shortest one in the Euclidean norm. Instead, we aim to find a lattice vector whose projection in either direction is not very large, which is consistent with our purpose of finding a nonzero vector with small $\ell_\infty$-norm. When the block-size in BKZ is determined, heuristic assumptions are made so as to estimate the probability that such a desired lattice vector exists: we assume that for those dimensions that are not affected by the BKZ algorithm, the associated coordinates follow the uniform distribution modulo $q$, whereas for those dimensions that are affected by the BKZ algorithm, the associated coordinates follows the Gaussian distribution with appropriate standard deviation. Finally, the cost estimate is the inverse of that probability multiplied by the run-time of our $b$-dimensional SVP-solver [DLL+17, LDK+17].

### References


[CRYSTALS] Presentation of CRYSTALS (Kyber and Dilithium) at 2018 NIST PQC standardization conference. https://csrc.nist.gov/CSRC/media/Presentations/Crystals-Dilithium


A The Implementation Procedures

Algorithm 6 The key generation algorithm in our implementation

**Input:** \( \lambda \)

**Output:** \((pk, sk)\)

1: \( \rho \leftarrow \{0, 1\}^{256} \)
2: \( \text{key} \leftarrow \{0, 1\}^{256} \)
3: \( (s, e) \leftarrow S_{\ell}^{\eta} \times S_{h}^{\eta} \)
4: \( A \in \mathcal{R}_{q}^{h \times \ell} := \text{Sam}(\rho) \)
5: \( t := A + e \)
6: \( (t_1, t_0) := \text{Power2Round}_q(d)(t) \)
7: \( \text{tr} \in \{0, 1\}^{384} := \text{CRH}(\rho, t_1) \)
8: \( \text{pk} := (\rho, t_1) \)
9: \( \text{sk} := (\rho, \text{key}, \text{tr}, s, e, t_0) \)
10: \text{return} \((pk, sk)\)

Algorithm 7 The Verification algorithm in our implementation

**Input:** \( pk, \mu, (z, c, h) \)

**Output:** \( b \in \{0, 1\} \)

1: \( A \in \mathcal{R}_{q}^{h \times \ell} := \text{Sam}(\rho) \)
2: \( \text{tr} \in \{0, 1\}^{384} := \text{CRH}(\rho, t_1) \)
3: \( c \in \mathcal{B}_{60} := H(\mu, w_1) \)
4: \( w_1 := \text{UseHint}_{q,k}(h, Az - ct_1) \cdot 2^d \)
5: \( c' \leftarrow H(\rho, t_1, w_1, \mu) \)
6: if \( c = c' \) AND \( \|z\|_{\infty} < \lfloor q/k \rfloor - U \) AND \( \|h\|_{1} \geq \omega \) then
7: \( b := 1 \)
8: else
9: \( b := 0 \)
10: end if
11: \text{return} \( b \)

Algorithm 8 The Sign algorithm in our implementation

**Input:** \( \text{sk}, \mu \)

**Output:** \((z, c, h)\)

1: \( A \in \mathcal{R}_{q}^{h \times \ell} := \text{Sam}(\rho) \)
2: \( \mu \in \{0, 1\}^{384} := \text{CRH}(\text{tr}, \mu) \)
3: \( \kappa := 0 \)
4: \text{while} true \text{ do}
5: \( y \in S_{\ell}^{q/k} := \text{ExpandMask}(\text{key}, \mu, \kappa) \)
6: \( \kappa := \kappa + 1 \)
7: \( w := Ay \)
8: \( w_1 := \text{HighBits}_{q,k}(w) \)
9: \( c \in B_{60} := H(\mu, w_1) \)
10: \( z := y + cs \)
11: \( u := w - ce \)
12: \( (r_1, r_0) := \text{Con}(u) \)
13: if \( \|z\|_{\infty} \geq \lfloor q/k \rfloor - U \) OR \( \|r_0\|_{\infty} \geq \lfloor q/2 \rfloor - k \cdot U' \) OR \( r_1 \neq w_1 \) then
14: \text{continue}
15: \text{end if}
16: \( v := ct_0 \)
17: if \( \|v\|_{\infty} \geq \lfloor q/(2k) \rfloor \) then
18: \text{continue}
19: \text{end if}
20: \( h := \text{MakeHint}_{q,k}(-v, u + v) \)
21: if \( \|h\|_{1} \geq \omega \) then
22: \text{continue}
23: \text{end if}
24: \text{break}
25: \text{end while}
26: \text{return} \((z, c, h)\)