GCKSign: Simple and Efficient Signatures from Generalized Compact Knapsacks*

Joo Woo[1], Jong Hwan Park[2], Kwangsu Lee[3],
Dong-Guk Han[4], and Hwajeong Seo[5]

[1] Graduate School of Information Security, Korea University
woojoo0121@korea.ac.kr
[2] Department of Computer Science, Sangmyung University
jhpark@smu.ac.kr
[3] Department of Computer and Information Security, Sejong University
kwangsu@sejong.ac.kr
[4] Department of Information Security, Cryptology, and Mathematics, Kookmin University
christa@kookmin.ac.kr
[5] Department of IT convergence, Hansung University
hwajeong84@gmail.com

Abstract. In 2009, Lyubashevsky proposed a lattice-based signature scheme by applying the Fiat-Shamir transformation and proved its security under the generalized compact knapsack (GCK) problem. This scheme has a simple structure but has large signature and key sizes due to the security requirement of their security reduction. Dilithium, which was submitted to the NIST Post-Quantum Cryptography standardization and selected as one of the final candidates, is an improvement of the Lyubashevsky's signature scheme and decreases key and signature sizes by modifying the form of a public key and including additional steps in key generation, signing, and verification algorithms. Thus, Dilithium has a more complex structure to implement compared to the Lyubashevsky's scheme. To combine the strength of both signature schemes, we modify the Lyubashevsky's signature scheme and present a new security proof eliminating the security requirement, which made their signature inefficient. As a result, we propose a simple and practical signature scheme based on the hardness of a new GCK assumption, called target-modified one-wayness of GCK function. The signature size of our scheme decreases 40 percent, the sum of signature and public key sizes decreases 25 percent, and the secret key size decreases 90 percent for the NIST security level III, compared to Dilithium. Furthermore, by virtue of the simplicity of our construction, the key generation, signing, and verification algorithms of our scheme run 2.4×, 1.7×, and 2.0× faster than those of Dilithium, respectively.

Keywords: Post-quantum cryptography · Lattice-based signature · Generalized compact knapsack problem.

* This work was submitted to 'Korean Post-Quantum Cryptography Competition' (www.kpqc.or.kr).

1 Introduction

Lattice-based cryptography is seen as a very promising alternative to traditional cryptography with the arrival of quantum computers. Traditional cryptography is mostly based on the hardness of the number-theoretic problems such as integer factorization and discrete logarithm for their security. As it was shown that those problems can be solved in polynomial-time by quantum algorithms, cryptography based on the hardness of lattice problems, which are known to have resistance to quantum algorithms, has attracted a lot of attention. The National Institute of Standard and Technology (NIST) launched the Post-Quantum Cryptography (PQC) Standardization for digital signature, encryption, and key establishment protocols resistant to quantum algorithms. In the fourth round NIST-PQC announcement of 2022, two lattice-based signatures Dilithium [13] and Falcon [16] were selected as the final three signature candidates.

Falcon is a compact signature scheme over NTRU lattices based on the GPV framework [17] and its security relies on the hardness of the NTRU problem that finds a short vector in NTRU lattices [16]. Although cryptography based on the NTRU problem has a long and established history, it lacks a formal reduction from worst-case to average-case hardness. While NTRU lattices are very attractive because of their performance and storage efficiency, there are some concerns that the assumed difficulty of the underlying NTRU problem may be weaker than expected. In that respect, it is desired to avoid using the NTRU assumption as long as the efficiency penalty is not too high. Also, Falcon samples elements from the discrete Gaussian distribution which has a potential side-channel vulnerabilities [10, 27]. Additionally, Falcon suffers from a very complex signing algorithm which makes it hard to implement and parallelize.

On the other hand, Dilithium is a signature scheme based on the Fiat-Shamir transformation [15] and its security relies on the hardness of the Module Learning with Errors (MLWE) and Module Short Integer Solutions (MSIS) problems [13]. The design of Dilithium is based on the "Fiat-Shamir with Aborts" framework [20] and other signature schemes proposed in [5,8,18]. Dilithium has a simpler structure to implement compared to Falcon and samples elements from the uniform distribution, not from the discrete Gaussian distribution. To decrease the size of a public key, Dilithium uses only high-order bits of a public key, composed of the LWE instances, since the low-order bits of the LWE instances do not affect the verification phase too much. However, to make up for the correctness error resulted from this, Dilithium adds extra algorithms during the signing and verification algorithms and this point leads to the increase of complexity for the scheme.

1.1 Our Contributions

The main results of this paper is to give a simple and efficient signature scheme, requiring no sampling with the discrete Gaussian distribution and extra algorithms such as a hint algorithm in [13]. We modify the construction of [20] and propose a simpler lattice-based signature scheme than Dilithium and Falcon. Our

scheme is based on the hardness of a new GCK problem, called target-modified one-wayness (TMO) problem, to get rid of the security requirement of [20], which made the parameter of their signature scheme inefficient. Due to the absence of the requirement, our parameter sets get a smaller-sized signature and secret key, compared to Dilithium. The simplicity of our signature scheme allows for a fast and efficient implementation secure against side-channel attacks and makes it easy for a developer to detect subtle implementation mistakes that could leak a secret key. The main features of our signature scheme can be summarized as follows:

Simplicity and ease of implementation. Our signature scheme has a very simple and compact structure to implement. The design of our scheme is based on the Lyubashevsky's framework [20], which is one of the simplest lattice-based signature construction following the design of the Schnorr signature. Also, our scheme only uses the uniform sampling, not the Gaussian sampling, which has a potential side-channel vulnerabilities. Moreover, all other operations including the polynomial multiplication and the rounding can be implemented in constant time. This simplicity allows for an implementation secure against side-channel attacks.

Minimization of public key and signature size. The goal of our signature scheme is to minimize the sum of public key and signature sizes because many applications require the transmission of these parameters. By eliminating the security requirement of [20] and analyzing a bounding parameter rigorously using the Central Limit Theorem (CLT) and the error function analysis, our signature scheme can have smaller signature size and secret key compared to Dilithium. That is, our scheme has the smallest value of the sum of the public key and the signature size of any lattice-based scheme with the same security levels under the restriction that a scheme does not use Gaussian sampling.

Fast and efficient implementation. The simplicity of our signature scheme results in the fast and efficient implementation. Also, we minimize the dimension of the public key to get not only a smaller key size and also the fast implementation. A polynomial multiplication and a sampling are the most expensive operations in implementing a cryptosystem. The main operations performed in our scheme are the multiplication and the sampling of a public key. Hence, the smaller the dimension of the public key we have, the more efficient our scheme is. Additionally, to get the small dimension of the public key, we analyze a bound parameter rigorously and set the form of the public key based on the GCK one-wayness problem, not the LWE problem, which is required to increase the dimension for the security.

1.2 Our Technique

In this paper, we define a relaxed notion of one-wayness problem of GCK function called the TMO problem, which is to solve the one-wayness problem of GCK function *approximately* instead of exactly. We now provide an overview of the TMO problem of GCK function. Given a public polynomials $\boldsymbol{a} = (a_1, \ldots, a_m) \in$

R_q^m and $t \in R_q$, we call a polynomial vector $\boldsymbol{x} = (x_1, \ldots, x_m) \in R^m$ and a polynomial $c \in R$ a solution of the TMO problem if $\sum_{i=1}^{m}(a_i \cdot x_i) = c \cdot t \mod q$ where both $\boldsymbol{x}$ and c are short. Under proper settings of parameters, we show the TMO problem is as hard as the collision-resistance problem of GCK function, or no easier than the one-wayness problem of GCK function. The TMO problem of GCK function is a natural generalization of the one-wayness problem of GCK function. The primary motivation is to improve the efficiency of lattice-based Fiat-Shamir signatures. With a relaxation of the GCK problem, it is possible to prove the security of [20] without the security requirement, which made the signature scheme inefficient. Because of the absence of the requirement, our parameter sets get a smaller-sized public key, secret key, and signature.

Another technique that we use to get the smaller signature size in this paper is to analyze the bounding parameter L_s rigorously and set the value tightly based on the analysis. In our signature scheme, one component of the signature $\boldsymbol{z}$ is computed as $\boldsymbol{z} = \boldsymbol{y} + c \cdot \boldsymbol{s}$ where $\boldsymbol{s}$ is a small secret polynomial vector and c is also short. The range which the value $\boldsymbol{y}$ is sampled from needs to be large enough to hide the secret $\boldsymbol{s}$ from the signature $(\boldsymbol{z}, c)$. Also, to make the distribution of the signature independent of the secret key $\boldsymbol{s}$, we apply the rejection sampling [20], which is that the signing algorithm outputs $\boldsymbol{z}$ only if all coefficients of $\boldsymbol{z}$ are in $[-B + L_s, B - L_s]$ where the value B is the range of all coefficients of $\boldsymbol{y}$ and the value L_s is the bound parameter of $c \cdot \boldsymbol{s}$, i.e. $\|c \cdot \boldsymbol{s}\|_\infty \leq L_s$. The value L_s is determined to ensure both the correctness and security of the signature scheme. In [13] and [20], the bound parameter L_s is calculated assuming the worst-case scenario. For the improvement of the efficiency, we study the bound parameter selection and present a rigorous analysis of setting more tight bound by analyzing the multiplication of two polynomials over the ring $\mathbb{Z}_q[x]/(x^n + 1)$ using the CLT and the error function analysis. This point leads to a decrease in the signature size.

1.3 Related Work

In 2009, Lyubashevsky [20] proposed a signature scheme whose security is based on the GCK problem adapted from the Fiat-Shamir transformation in the random oracle model. The signature scheme has a very simple structure because it is based on the Schnorr signature. However, the public key and the signature sizes of [20] are considerably large. This is mainly due to the security requirement of the security proof under the collision resistance problem of GCK function (or the MSIS problem). We briefly introduce the Lyubashevsky's identification protocol and the security proof in Chapter 2.3 and Appendix B. The signature scheme has a public key of the form $t = \sum_{i=1}^{m}(a_i \cdot s_i) \mod q$ where $\boldsymbol{a}$ is sampled from R_q^m and $\boldsymbol{s}$ is sampled from the small uniform distribution, which is based on the Module Inhomogeneous SIS (MISIS) problem. In order to prove the unforgeability of the signature scheme, there should be another valid secret key corresponding a public key. This is achieved by taking the range which the secret key is sampled from to be sufficiently large, which is the security requirement of [20]. This approach would increase the signature size and the public key size.

In 2012, Lyubashevsky gave an improved signature scheme [21] based on the construction of [20]. The signature scheme has a public key based on the standard ISIS problem and its security proof is based on the standard SIS problem. Similar to [20], there should be another valid secret key corresponding a public key. Instead of taking the range of the secret key to be sufficiently large, they employ an alternative proof technique based on the SIS decision problem. The SIS decision problem is to distinguish between the uniform distribution and the SIS distribution, which is $\boldsymbol{t} = \boldsymbol{A}\boldsymbol{s} \mod q$ where $\boldsymbol{A}$ and $\boldsymbol{s}$ are sampled from $\mathbb{Z}_q^{n\times m}$ and $[-\eta, \eta]^m$, respectively. It is shown that the decisional version of the random subset sum problem is as hard as the search version [19] and Micciancio and Mol have generalized the relationship to the standard SIS problem [25]. In their security proof, using the SIS decision problem, the public key $\boldsymbol{t} = \boldsymbol{A}\boldsymbol{s}$ is replaced to $\boldsymbol{t} = \boldsymbol{A}\boldsymbol{s'}$ where $\boldsymbol{s'}$ has much larger entries than $\boldsymbol{s}$ does. Thus, the public key $\boldsymbol{t} = \boldsymbol{A}\boldsymbol{s'}$ has non-unique solutions with overwhelming probability. Since the SIS decision problem is not transferred easily to ring setting, the construction of [21] can not be extended to the ring version of the construction, which has an efficient multiplication operation and supports the reduction of the public key size by the algebraic structure.

Bai and Galbraith presented a provably secure efficient signature following the idea of the security proof of [21] and they changed the form of the public key based on the standard LWE and SIS problems [5] in 2014. Since then, several follow-up works based on the design of [5] have been proposed [1,3,6,12–14]. The decisional LWE problem can be extended to ring version and module version so their schemes have small size of the secret key, the public key, and signature and have an efficient performance by the algebraic structure. The signature scheme of [5] has a public key based on the LWE problem over standard lattices. The signature scheme of Dilithium [13] has a public key of the form $\boldsymbol{t} = \boldsymbol{A}\boldsymbol{s} + \boldsymbol{e} \mod q$ where $\boldsymbol{A}$ is sampled from $R_q^{k\times m}$ and $\boldsymbol{s}$ and $\boldsymbol{e}$ are sampled from the small uniform distribution, which is based on the MLWE problem over ideal lattices. Also, Ducas et al. [13] points out that the low-order bits of a public key, composed of LWE instances, do not affect the verification phase too much. Thus, Dilithium uses only high-order bits of the LWE instances as the public key. Because of the form of the public key and the public key compression technique, Dilithium has a shorter key sizes than previous lattice-based signatures. However, to make up for the correctness error resulted from this compression technique, Dilithium adds an extra algorithms and this point leads to an increase of the complexity for the scheme.

2 Preliminaries

2.1 Digital Signatures

A public-key signature (PKS) scheme for a message space $\mathcal{M}$ consists of three algorithms: KeyGen, Sign, and Verify which are defined as follows:

- $\mathsf{KeyGen}(\lambda)$: The key generation algorithm takes as input a security parameter λ and outputs public key and secret key (pk, sk).

- $\mathsf{Sign}(sk, \mu)$: The signing algorithm takes as input the secret key sk and a message $\mu \in \mathcal{M}$, and then outputs a signature σ.
- $\mathsf{Verify}(pk, \mu, \sigma)$: The verification algorithm takes as input the public key pk, a message μ and a signature σ, and then outputs 1 if the signature is valid or 0 otherwise.

Correctness. We say that a signature scheme is $(1-\gamma)$-*correct* if the following condition holds: for all $(pk, sk) \in \mathsf{KeyGen}(\lambda)$ and all messages $\mu \in \mathcal{M}$,

$$\Pr\left[(pk, sk) \leftarrow \mathsf{KeyGen}(\lambda);\ \sigma \leftarrow \mathsf{Sign}(sk, \mu) : \mathsf{Verify}(pk, \mu, \sigma) = 1\right] > 1 - \gamma(\lambda),$$

where γ is a negligible function for the security parameter λ.

Let $\mathsf{PKS} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ be a signature scheme. The unforgeability against chosen-message attack (UF-CMA) is defined via the following experiment $\mathbf{UF\text{-}CMA}^{\mathcal{A}}_{PKS}(\lambda)$ between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$:

1. $\mathcal{C}$ runs $(pk, sk) \leftarrow \mathsf{KeyGen}(\lambda)$ and gives pk to $\mathcal{A}$.
2. $\mathcal{A}$ queries signing oracle $\mathsf{Sign}(sk, \mu)$ with a message μ.
3. Finally, $\mathcal{A}$ outputs a signature σ^* and a message μ^* which was not previously queried to the signing oracle. $\mathcal{C}$ returns 1 if $\mathsf{Verify}(pk, \mu^*, \sigma^*) = 1$ and otherwise returns 0 as the output of the game.

The advantage of $\mathcal{A}$ for breaking the UF-CMA security of PKS is defined as $\mathsf{Adv}^{\text{UF-CMA}}_{PKS}(\mathcal{A}) = \Pr[\mathsf{UF\text{-}CMA}^{\mathcal{A}}_{PKS} \Rightarrow 1]$. We say that a signature scheme is UF-CMA secure if for any polynomial-time adversary $\mathcal{A}$, we have $\mathsf{Adv}^{\text{UF-CMA}}_{PKS}(\mathcal{A}) \leq \epsilon(\lambda)$, where ϵ is a negligible function for the security parameter λ.

2.2 Lattice Problems

Before we present lattice problems, we define the GCK function, introduced by Micciancio [24]

Definition 1 (GCK Function). *For a ring R, subset $S \subset R$, integer $m \geq 1$, and a randomly and independently chosen element $\boldsymbol{a} = (a_1, \ldots, a_m) \in R^m$, the generalized knapsack function $F_{\boldsymbol{a}} : R^m \to R$ is defined as follow:*

$$F_{\boldsymbol{a}}(\boldsymbol{x}) = \textstyle\sum_{i=1}^{m}(a_i \cdot x_i),$$

for $\boldsymbol{x} \in S^m$, where $\sum_{i=1}^{m}(a_i \cdot x_i)$ is computed using the ring multiplication and addition operation.

In this paper we consider the ring $R_q = \mathbb{Z}_q/(x^n + 1)$ and a subset $S \subset R = R_{[-\beta,\beta]}$ for some positive integer β.

Definition 2 (One-Wayness of GCK function [24]). *A GCK function is one-way (OW) if for any probabilistic polynomial-time algorithm $\mathcal{A}$, it is easy to compute, but computationally hard to invert the GCK function: given a pair $(\boldsymbol{a}, t = F_{\boldsymbol{a}}(\boldsymbol{x}))$ for randomly chosen $\boldsymbol{a} \in R^m$ and $\boldsymbol{x} \in R_{[-\beta,\beta]}$, find an $\boldsymbol{x}$ in the domain such that $F_{\boldsymbol{a}}(\boldsymbol{x})) = t$. For integers $m, q \in \mathbb{N}$ and a real number $\beta \in \mathbb{R}^+$, we define $\mathsf{Adv}^{OW}_{m,q,\beta}$ to be the advantage of an algorithm $\mathcal{A}$ in solving the OW problem of GCK function over the ring R_q.*

Based on the analysis in [24], solving the OW problem of GCK function with non-negligible probability is as hard as approximating the shortest independent vector problem (SIVP) on cyclic lattices in the worst case.

Definition 3 (Collision-Resistance of GCK function [22, 26]). *A GCK function is collision-resistant (CR) if for any probabilistic polynomial-time algorithm $\mathcal{A}$, it is computationally hard to find the collision of GCK function: given a random $\boldsymbol{a} \in R^m$, find a pair $(\boldsymbol{x}, \boldsymbol{x}')$ in the domain such that $F_{\boldsymbol{a}}(\boldsymbol{x}) = F_{\boldsymbol{a}}(\boldsymbol{x}')$. For integers $m, q \in \mathbb{N}$ and a real number $\beta \in \mathbb{R}^+$, we define $\mathsf{Adv}^{CR}_{m,q,\beta}$ to be the advantage of an algorithm $\mathcal{A}$ in solving the CR problem of GCK function over the ring R_q.*

Based on the analysis of [22, 26], finding the collision of GCK function with non-negligible probability is as hard as the shortest vector problem (SVP) over cyclic lattices in the worst case.

It is easy to see that there is a reduction from the CR problem of GCK function to the OW problem of GCK function. In other words, if there is a polynomial-time algorithm $\mathcal{A}$ that can solve the OW problem of GCK function, then there is another algorithm $\mathcal{B}$ that can break the collision resistance of GCK function, i.e. $\mathsf{Adv}^{\mathrm{OW}}_{m,q,\beta} \leq \mathsf{Adv}^{\mathrm{CR}}_{m,q,\beta}$. We briefly describe how $\mathcal{B}$ solves the CR problem using $\mathcal{A}$. Initially, $\mathcal{B}$ takes $\boldsymbol{a} \in R_q^m$ as input and needs to find a pair $(\boldsymbol{x}, \boldsymbol{x}')$ such that $F_{\boldsymbol{a}}(\boldsymbol{x}) = F_{\boldsymbol{a}}(\boldsymbol{x}')$, $\|\boldsymbol{x}\|_\infty \leq \beta$, and $\|\boldsymbol{x}'\|_\infty \leq \beta$. It samples $x \leftarrow R^m_{[-\beta,\beta]}$ and computes $t = F_{\boldsymbol{a}}(\boldsymbol{x})$. Then it runs $\mathcal{A}$ on input $(\boldsymbol{a}, t)$. Finally, $\mathcal{A}$ outputs $\boldsymbol{x}' \in R_q^m$ such that $\|\boldsymbol{x}'\|_\infty \leq \beta$ and $F_{\boldsymbol{a}}(\boldsymbol{x}') = t$. We need to show that $\boldsymbol{x} \neq \boldsymbol{x}'$. For a randomly picked $\boldsymbol{x} \in R^m_{[-\beta,\beta]}$, there is another value $\boldsymbol{x}' \in R^m_{[-\beta,\beta]}$ which produces the same value t with high probability since $(2\beta+1)^{nm} \gg q^n$. Also, $\mathcal{A}$ does not know which value is used to compute t. Thus, it holds that $\boldsymbol{x} \neq \boldsymbol{x}'$ with high probability. Thus $\mathcal{B}$ solves the CR problem by outputting a pair $(\boldsymbol{x}, \boldsymbol{x}')$.

Definition 4 (Target-Modified One-wayness of GCK function). *For any $n, m, q \in \mathbb{N}$ and $\alpha, \beta \in \mathbb{R}$, the target-modified one-wayness (TMO) problem is defined as follows: Given $\boldsymbol{a} \in R_q^m, t \in R_q$, find $\boldsymbol{x} \in R_q^m$ and $c \in R_q$ such that $\|c\|_\infty \leq \alpha$, $\|\boldsymbol{x}\|_\infty \leq \beta$ satisfying*

$$F_{\boldsymbol{a}}(\boldsymbol{x}) = c \cdot t \mod q.$$

The TMO problem is to solve the OW problem approximately instead of exactly, which is a relaxed notion of the OW problem. Recall that the OW problem is to find a short $\boldsymbol{x} \in R_q^m$, which is a preimage of t for the GCK function $F_{\boldsymbol{a}}(\cdot)$. Now, the TMO problem is to find $\boldsymbol{x} \in R_q^m$, which is an *approximate* short preimage of t such that $F_{\boldsymbol{a}}(\boldsymbol{x}) = c \cdot t \mod q$ for some short $c \in R_q$. The TMO problem is non-trivial when the bound α are relatively small compared to the bound β. For integers $m, q \in \mathbb{N}$ and real numbers $\alpha, \beta \in \mathbb{R}^+$, we define $\mathsf{Adv}^{\mathrm{TMO}}_{m,q,\alpha,\beta}$ to be the advantage of an algorithm $\mathcal{A}$ solving the TMO problem of GCK function over the ring R_q. We present the hardness of the TMO problem based on previous GCK problems in Appendix A.

2.3 GCK-based Identification Protocol

A lattice-based identification protocol (Figure 1), introduced by Lyubashevsky in 2009 [20], is based on the CR problem of GCK function. In this protocol, a public key does not reveal a secret key because of the OW problem of GCK function and its security is proved under the stronger assumption that the underlying GCK function is CR. At a high level, the protocol of Lyubashevsky has a public key $(\boldsymbol{a}, t = F_{\boldsymbol{a}}(\boldsymbol{s}) \mod q)$ and a response is a proof of knowledge of a secret key $\boldsymbol{s}$ where $\boldsymbol{s} \in R^m_{[-\eta,\eta]}$ for some small integer η. In the first round, a prover picks $\boldsymbol{y} \in R^m_{[-B,B]}$ for some relatively large integer B and sends $F_{\boldsymbol{a}}(\boldsymbol{y})$ to a verifier. Next, the verifier picks a random challenge $c \in R_{n,h}$, and send it to the prover. After that, the prover computes $\boldsymbol{z} = \boldsymbol{y} + c \cdot \boldsymbol{s}$. If $\boldsymbol{z}$ and $c \cdot \boldsymbol{s}$ are in a proper range, then the prover sends $\boldsymbol{z}$ to the verifier to prevent the leakage of the secret key. Otherwise, the prover has to abort and the protocol has to be repeated. This identification protocol can be converted into a signature scheme by using the Fiat-Shamir transformation. In Appendix B, we sketch the security proof of Lyubashevsky's signature scheme under the CR problem of GCK function.

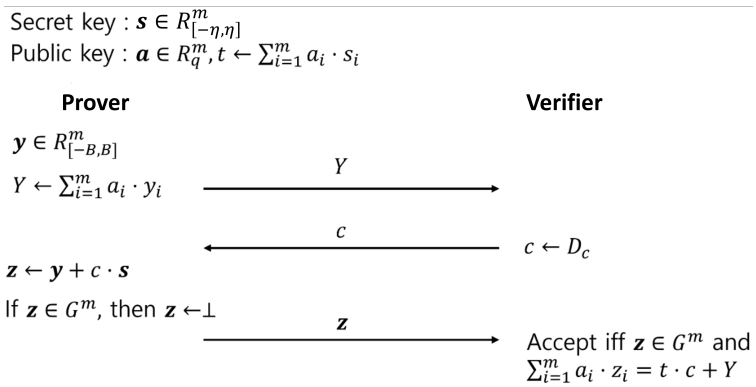

Fig. 1. Lyubashevsky's identification protocol

3 GCKSign Specification

3.1 Notation

For an even (odd) modulus $q \in \mathbb{Z}_{\geq 0}$ and an integer $k \in \mathbb{Z}$, define $k' = k \mod^{\pm} q$ as the unique element k' such that $-q/2 < k' \leq q/2$ (resp. $-\lfloor q/2 \rfloor \leq k' \leq \lfloor q/2 \rfloor$) and $k' = k \mod q$. $\mathbb{Z}_q = \mathbb{Z}/q\mathbb{Z}$ denotes the quotient ring of integers modulo q. Let $\mathcal{R}$ and $\mathcal{R}_q$ respectively denote the rings $\mathbb{Z}[x]/(x^n + 1)$ and $\mathbb{Z}_q[x]/(x^n + 1)$, where n is a power of two, and q is a prime such that $q \equiv 1 \mod 2n$. Vectors with entries in R_q are denoted with bold lowercase letters, for example,

$\boldsymbol{a} = (a_1, \ldots, a_m) \in R_q^m$ where $a_1, \ldots, a_m \in R_q$ for some positive integer m. $R_{n,h}$ denotes a set of an element in $\mathcal{R}_q$ that has all zeros coefficients except h out of n coefficients are in $\{1, -1\}$. We have $\|R_{n,h}\| = 2^h \cdot \binom{n}{h}$. Let $\mathcal{R}_{[-x,x]}$ denotes a set of an element in $\mathcal{R}_q$ satisfying that all coefficients are between $[-x, x]$ for a positive integer x. We define the infinity norm for a polynomial $f = f_0 + f_1 x + \ldots + f_{n-1} x^{n-1} \in \mathcal{R}$ as $\|f\|_\infty = \max_{0 \le i \le n-1} |f_i \mathrm{mod}^{\pm} q|$. Similarly, for $\boldsymbol{f} = (f_1, \ldots, f_m) \in R^m$, we define $\|\boldsymbol{f}\|_\infty = \max_i(\|f_i\|_\infty)$.

3.2 Construction

In this section we describe our signature scheme based on the GCK problem. This scheme requires a hash function H to binary strings of fixed length ℓ and an encoding function encode that maps binary strings of length ℓ to an element in the set $R_{n,h}$. For the security parameter λ, the system parameter *params* is generated as follows: choose an integer n such that $n = 2^a$ for a positive integer $a \in \mathbb{N}$ and we set $n = 256$ for the NIST security level II and III and $n = 512$ for the NIST security level V. Also, choose a modulus q as a prime and positive integers m, B, h, L_s to fulfill the security requirements. Then, *params* is given by (n, q, m, B, h, L_s). It is assumed that *params* is used for all algorithms in our signature construction. The key generation, signing, and verification algorithms of our signature scheme are described as follows:

KeyGen. This algorithm first chooses random 256-bit seeds $\mathrm{seed}_{\boldsymbol{a}}$ and $\mathrm{seed}_{\boldsymbol{s}}$. It samples public polynomials $a_1, \ldots, a_m$ uniformly at random over R_q by expanding $\mathrm{seed}_{\boldsymbol{a}}$ and secret polynomials $s_1, \ldots, s_m$ uniformly at random over $R_{[-\eta,\eta]}$ by expanding $\mathrm{seed}_{\boldsymbol{s}}$. Next, it computes a polynomial $t = F_{\boldsymbol{a}}(\boldsymbol{s}) = \sum_{i=1}^m (a_i \cdot s_i)$. Finally, it outputs a public key $pk = (t, \mathrm{seed}_{\boldsymbol{a}})$ and a secret key $sk = (\boldsymbol{s} = (s_1, \ldots, s_m), \mathrm{seed}_{\boldsymbol{a}})$.

Algorithm 1: KeyGen

Input : -

Output: public key $pk = (t, \mathrm{seed}_{\boldsymbol{a}})$, and secret key $sk = (\boldsymbol{s}, \mathrm{seed}_{\boldsymbol{a}})$

$\mathrm{seed}_{\boldsymbol{a}}, \mathrm{seed}_{\boldsymbol{s}} \leftarrow \{0,1\}^{256}$;
$\boldsymbol{a} = (a_1, \ldots, a_m) \leftarrow \mathrm{sample}_a(\mathrm{seed}_{\boldsymbol{a}}) \in R_q^m$;
$\boldsymbol{s} = (s_1, \ldots, s_m) \leftarrow \mathrm{sample}_s(\mathrm{seed}_{\boldsymbol{s}}) \in R_{[-\eta,\eta]}^m$;
$t \leftarrow F_{\boldsymbol{a}}(\boldsymbol{s}) = \sum_{i=1}^m (a_i \cdot s_i)$;
$pk \leftarrow (t, \mathrm{seed}_{\boldsymbol{a}})$;
$sk \leftarrow (\boldsymbol{s}, \mathrm{seed}_{\boldsymbol{a}})$;
return (pk, sk);

Sign. This algorithms first regenerates the public polynomials $a_1, \ldots, a_m$ from $\mathrm{seed}_{\boldsymbol{a}}$. It chooses a random 256-bit seed $\mathrm{seed}_{\boldsymbol{y}}$ and initializes a counter at 1. It samples polynomials $y_1, \ldots, y_m$ uniformly at random over $R_{[-B,B]}$ by using $\mathrm{seed}_{\boldsymbol{y}}$ and the counter. Next, it computes a polynomial $v = F_{\boldsymbol{a}}(\boldsymbol{y}) = \sum_{i=1}^m (a_i \cdot y_i)$. It

obtains $\hat{c}$ by computing the hash function $H(\boldsymbol{v}, \mu)$ together with the message μ. It obtains a sparse polynomial $c \in R_{n,h}$ by running encode($\hat{c}$) and computes $\boldsymbol{z} = \boldsymbol{y} + c \cdot \boldsymbol{s}$. If $\boldsymbol{z} \notin R_{[-B+L_s, B-L_s]}$, then it increase the counter and goes to the step that samples y and repeats the subsequent steps. Finally, it outputs a signature $\sigma = (\boldsymbol{z}, \hat{c})$.

Algorithm 2: Sign

Input : message μ, and secret key $sk = (\boldsymbol{s}, \text{seed}_{\boldsymbol{a}})$
Output: signature $(\boldsymbol{z}, \hat{c})$

counter $\leftarrow 1$;
$\boldsymbol{a} = (a_1, \ldots, a_m) \leftarrow \text{sample}_a(\text{seed}_{\boldsymbol{a}}) \in R_q^m$;
$\text{seed}_{\boldsymbol{y}} \leftarrow \{0,1\}^{256}$;
$\boldsymbol{y} = (y_1, \ldots, y_m) \leftarrow \text{sample}_y(\text{seed}_{\boldsymbol{y}}, \text{counter}) \in R^m_{[-B,B]}$;
$v \leftarrow F_{\boldsymbol{a}}(\boldsymbol{y}) = \sum_{i=1}^{m}(a_i \cdot y_i)$;
$\hat{c} \leftarrow H(v, \mu) \in \{0,1\}^{256}$;
$c \leftarrow \text{encode}(\hat{c}) \in R_{n,h}$;
$\boldsymbol{z} \leftarrow \boldsymbol{y} + \boldsymbol{s} \cdot c$;
if $\boldsymbol{z} \notin R^m_{[-B+L_s, B-L_s]}$ **then**
  counter $\leftarrow$ counter $+ 1$;
  goto step 4;
end
return $\sigma = (\boldsymbol{z}, \hat{c})$;

Verify. This algorithm first derives a polynomial c from $\hat{c}$ in the signature. It regenerates the public polynomials $a_1, \ldots, a_m$ from the seed $\text{seed}_{\boldsymbol{a}}$. Next, it computes $w = F_{\boldsymbol{a}}(\boldsymbol{z}) - c \cdot t = \sum_{i=1}^{m} a_i \cdot z_i - c \cdot t$. This polynomial w is used to compute the hash value $H(\boldsymbol{w}, \mu)$ together with the message μ. It accepts the signature if the hash value matches the signature $\hat{c}$ and $\boldsymbol{z} \in R_{[-B+L_s, B-L_s]}$.

Algorithm 3: Verify

Input : message μ, signature $\sigma = (\boldsymbol{z}, \hat{c})$, and public key $pk = (t, \text{seed}_{\boldsymbol{a}})$
Output: {1, 0} // accept or reject signature

$c \leftarrow \text{encode}(\hat{c}) \in R_{n,h}$;
$\boldsymbol{a} = (a_1, \ldots, a_m) \leftarrow \text{sample}_a(\text{seed}_{\boldsymbol{a}}) \in R_q^m$;
$w \leftarrow F_{\boldsymbol{a}}(\boldsymbol{z}) - t \cdot c = \sum_{i=1}^{m} a_i \cdot z_i - t \cdot c$;
if $\boldsymbol{z} \notin R^m_{[-B+L_s, B-L_s]} \vee \hat{c} \neq H(w, \mu)$ **then**
  return 0;
end
return 1;

3.3 Correctness

We show the correctness of our signature scheme. To guarantee the correctness of this signature scheme, it has to hold two requirements for a signature $\sigma = (z, \hat{c})$: The first one is that $\|z\|_\infty < B - L_s$ and the second one is that the output of the hash function in the signing algorithm and the output of the hash function in the verification algorithm are same. The first one is guaranteed since this condition is checked in the signing algorithm. The second one is also guaranteed by the following equations

$$\begin{aligned}\sum_{i=1}^{m}(a_i \cdot z_i) - c \cdot t &= \sum_{i=1}^{m}(a_i \cdot (y_i + c \cdot s_i) - c \cdot \sum_{i=1}^{m}(a_i \cdot s_i) \\ &= \sum_{i=1}^{m}(a_i \cdot y_i) + c \cdot \sum_{i=1}^{m}(a_i \cdot s_i - a_i \cdot s_i) = \sum_{i=1}^{m}(a_i \cdot y_i)\end{aligned}$$

3.4 Implementation Details

Generation of $\boldsymbol{a}$. The function sample_a maps a uniform seed $\text{seed}_{\boldsymbol{a}} \in \{0,1\}^{256}$ to a vector $\boldsymbol{a} \in R_q^m$ in NTT (number theoretic transform) domain representation during the key generation, signing, and verification algorithms. It computes each coefficient of $a_i \in R_q$ of $\boldsymbol{a}$ separately. The seed is expanded using SHAKE-128. The output stream is considered as a sequence of integers between 0 and $2^{\lceil \log q \rceil} - 1$. Then it proceeds to do rejection sampling to obtain a value less than the modulus q. It repeats SHAKE-128 until all the mn coefficients are filled out in case that the output stream is exhausted.

Generation of $\boldsymbol{s}$. The function sample_s maps a uniform seed $\text{seed}_{\boldsymbol{s}} \in \{0,1\}^{256}$ to a vector $\boldsymbol{s} \in R_{[-\eta,\eta]}^m$ during the key generation and signing algorithms. The process is similar to the way to sample $\boldsymbol{a}$. The seed is expanded using SHAKE-128 and the output stream is considered as a sequence of integers between 0 and $2^{\lceil \log(2\eta+1) \rceil} - 1$. Then it proceeds to do rejection sampling to generate a value in the range $\{0, 2\eta - 1\}$. Afterwards, the integers are obtained by subtracting $(\eta - 1)$ from the value.

Generation of $\boldsymbol{y}$. The function sample_y maps a uniform seed $\text{seed}_{\boldsymbol{y}} \in \{0,1\}^{256}$ to a vector $\boldsymbol{s} \in R_{[-B,B]}^m$ during the signing algorithm. The process is similar to the way to sample $\boldsymbol{s}$. The seed is expanded using SHAKE-128 and the output stream is considered as a sequence of integers between 0 and $2^{\lceil \log(2B+1) \rceil} - 1$. Then it proceeds to do rejection sampling to generate a value in the range $\{0, 2B - 1\}$. Afterwards, the integers are obtained by subtracting $(B - 1)$ from the value.

Encoding function. The encoding function encode maps a bit string $\hat{c}$ to a polynomial $c \in R_{n,h}$ of elements of R which have h coefficients that are either 1 or -1 and the rest are 0. This function absorbs the bit string $\hat{c}$ into SHAKE-256 and outputs a stream of random bytes that are interpreted as the positions and signs of the h nonzero entries of $\hat{c}$. It repeats SHAKE-256 until all the h coefficients are filled out in case that the output stream is exhausted.

Table 1. Security requirements for our signature scheme

Parameters	Definitions	Security Requirements
λ	security parameter	-
n		2^k where $k \in \mathbb{N}$
h	number of ± 1 in c	$2^h \cdot \binom{n}{h} \geq 2^\lambda$
η	secret polynomial range	-
m	dimension of pk	-
q	modulus	$q \equiv 1 \mod 2n$
L_s	bound of $\|c \cdot \boldsymbol{s}\|_\infty$	$\eta_s \cdot \sqrt{\frac{\tau}{3\eta^2}}$
δ_z	$1-$ rejection probability	≥ 0.3
B	y coefficient range	$\geq \frac{\sqrt[n]{\delta_z}+2L_s-1}{2(1-\sqrt[n]{\delta_z})} \geq \frac{1}{2} \cdot (q^{1/m}-1)$
pk size (bytes)	$(n \cdot \log_2 q + 256)/8$	
σ size (bytes)	$(nm \cdot (\lceil 2\log_2(B-L_s) \rceil + 1) + 256)/8$	
sk size (bytes)	$(nm \cdot \lceil \log_2(2\eta+1) \rceil) + n)/8$	

3.5 Number of Iterations

We analyze the probability of $\boldsymbol{z} \in R^m_{[-B+L_s, B-L_s]}$ that stops the iterations of the signing steps. The probability that $\|\boldsymbol{z}\|_\infty < B - L_s$ can be (heuristically) computed as

$$\delta_{\boldsymbol{z}} = \Big(\frac{2B-2L_s+1}{2B+1}\Big)^{nm}.$$

The probability that $\|\boldsymbol{s} \cdot c\|_\infty \leq L_s$ is approximately $1 - 1/2^{10}$ in our parameter sets by setting L_s in the way in Chapter 5. The expected number of iterations during the signing algorithm is the inverse of the product of the probability δ_z and $(1 - 1/2^{10})$. In Table 2, the expected number of iterations of our scheme varies from approximately 2.97 to 3.08.

3.6 Parameter Sets

We provide bounds of all the system parameters and concrete parameters to fulfill the security requirements. In Table 1, we summarize the bounds and requirements of all the system parameters. All parameters satisfy the requirements in Table 1 to ensure both correctness and security of our signature scheme.

Let λ be the security parameter, which is the bit security of our signature scheme. Let $\mathcal{R}$ and $\mathcal{R}_q$ denote the rings $\mathbb{Z}[x]/(x^n+1)$ and $\mathbb{Z}_q[x]/(x^n+1)$ respectively where q is a prime modulus such that $q \equiv 1 \mod 2n$ and n is the dimension of the ring which is a power of 2. Let η be the value indicating the range used to sample the coefficients of secret polynomials in $[-\eta, \eta]$. Let m be the number of public polynomials and h be the number of non-zero coefficients in the output of the hash function. A public key consists of seed_a and

Table 2. Parameters for our signature scheme

NIST Security Level	II	III	V
n	256	256	512
q	$2^{54} - 10751$	$2^{60} - 2559$	$2^{44} - 7167$
m	4	4	3
h	39	45	44
challenge space	192	212	256
B	$2^{14} - 1$	$2^{14} + 2^9$	$2^{15} - 1$
η	1	1	1
L_s	18	18	19
Repetitions	3.08	2.97	2.43
SIS Hardness (Core-SVP)			
BKZ block-size b	430	628	917
Classical Core-SVP	125	183	268
Quantum Core-SVP	114	166	243
Output Size			
pk size (bytes)	1,760	1,952	3,040
σ size (bytes)	1,952	2,080	3,104
sk size (bytes)	288	288	544
Performance (Reference Code)			
KeyGen (K cycles)	184	202	265
Sign (K cycles)	1,062	1,240	1,421
Verify (K cycles)	237	253	373

t, which is $(256 + n \cdot \log_2 q)/8$ bytes. A signature consists of $\boldsymbol{z}$ and $\hat{c}$, which is $(n \cdot (2\lceil \log_2(B - L_s) \rceil + 1) + 256)/8$ bytes. A secret key consists of seed_a and $\boldsymbol{s}$, which is $(nm \cdot \lceil \log_2(2\eta + 1) \rceil) + 256)/8$ bytes.

We propose three sets of parameters targeting the NIST security level II, III, and V in Table 2. We set $n = 256$ (or $n = 512$) for the dimension of the polynomial ring and $\eta = 1$ for the coefficient bound of the secret polynomials. For the NIST security level II, we set $n = 256$ and $q \approx 2^{54}$. For the NIST security level III, we set $n = 256$ and $q \approx 2^{60}$. For the NIST security level V, we set $n = 512$ and $q \approx 2^{44}$.

In Table 3, we compare our signature's parameter to those of Lyubashevsky's scheme [20] and Dilithium [13]. The table shows the parameters of our scheme including the sizes of public key, signature, and secret key compared to those of [13, 20]. Whereas our scheme is based on the TMO problem of GCK function, the scheme of [20] is based on the collision-resistance of GCK function, which is a stronger assumption. Also, in [20], the coefficient range of the secret polynomials η is set to 2047 to satisfy their additional security requirement $(2\eta + 1)^{nm} \geq q^n \cdot 2^{128}$ cased by the witness indistinguishability. Because large η causes larger

Table 3. Comparison of parameters in signature schemes

	n	m	q	η	Sig (bytes)	PK (bytes)	SK (bytes)	PK + Sig (bytes)	Classical Hardness	Hard Problem
[20]	512	5	$\approx 2^{60}$	2047	9,000	3,875	3,875	12,875	71	GCK-CR
	512	8	$\approx 2^{96}$	2047	14,875	6,125	6,125	21,000	127	
[13]	256	(4, 4)	$\approx 2^{23}$	2	2,420	1,312	2,544	3,732	123	MLWE, MSIS
	256	(6, 5)	$\approx 2^{23}$	4	3,293	1,952	4,016	5,245	182	
	256	(8, 7)	$\approx 2^{23}$	2	4,595	2,592	4,880	7,187	252	
Ours	256	4	$\approx 2^{54}$	1	1,952	1,760	288	3,712	125	GCK-TMO
	256	4	$\approx 2^{60}$	1	2,080	1,952	288	4,032	183	
	512	3	$\approx 2^{44}$	1	3,104	3,040	544	6,144	268	

bound parameter L_s and large range of $\boldsymbol{y}$, these values directly affect the size of signature. By changing the lattice problem and the security analysis of our scheme, we can remove their additional security requirement. Thus we can set smaller $\eta = 1$. As a result, the public key, secret key, and signature sizes of our scheme decrease considerably compared to those of [20].

Dilithium is an improvement of the signature scheme of [5]. Thus, the security of Dilithium is based on the hardness of not only the MSIS problem but also the MLWE problem. Briefly speaking, key-recovery attack, which attempts to recover a secret key from a public key, is to solve the MLWE problem and forgery attack, which attempts to forge a signature, is to solve the MSIS problem. To guarantee that the scheme is secure against those attacks, both the MSIS and MLWE problems have enough hardness against the best known lattice attacks such as primal attack and dual attack. The MLWE and MSIS problems involve finding a short vector in lattices. However, the analysis of two instances is slightly different. We now recall the MLWE and MSIS problems briefly. For integers k and m, given $(\boldsymbol{A}, t = \boldsymbol{A}\boldsymbol{s} + \boldsymbol{e})$, the MLWE problem is finding $(\boldsymbol{s}, \boldsymbol{e})$ where $\boldsymbol{A}$ is sampled from $R_q^{k \times m}$. The MSIS problem is finding $\boldsymbol{y}$ such that $\boldsymbol{A} \cdot \boldsymbol{y} = 0 \mod q$ where $\boldsymbol{A}$ is sampled from $R_q^{k \times m}$. The straightforward way of increasing the hardness of the MLWE problem is to increase the values (k, m). However, the hardness of the MSIS problem is increased when k is increased and m is decreased. In other words, if m is increased, the MLWE problem became harder, but the MSIS problem became easier. Thus, in Dilithium, m is set to a value providing similar hardness of the MLWE and MSIS problems after k is set.

Unlike Dilithium, our signature scheme is based on only SIS problem, so the security increases when the value m decreases (until the GCK function has a collision). Also, setting the bound parameter L_s more tightly, as described in Chapter 5, helps to reduce the value m and the bound parameter B while maintaining the number of iterations during the signing phase. Since the value m and the bound parameter B are directly related to the signature size ($= m \cdot \lceil \log_2 B \rceil$), our scheme provides smaller signature sizes compared to [13] for the same security level. In detail, the size of signature for our scheme decreases

Table 4. Performance comparison of signature schemes

	NIST Security Level	KeyGen (K cycle)	Sign (K cycle)	Verify (K cycle)	Classical Hardness
Dilithium [13]	II	272	1,323	298	123
	III	495	2,155	520	182
	V	728	2,592	779	252
Ours	II	184	1,062	237	125
	III	202	1,240	253	183
	V	265	1,421	373	268

20 percent for the NIST security level II and around 35 percent for the NIST security level III, compared to [13]. Moreover, we minimize the sum of the public key and the signature. The sum of these parameters decreases 25 percent for the NIST security level III.

4 Performance Analysis

In Table 4, we evaluate the performance of our implementations on a 3.7GHz Intel Core i7-8700k running Ubuntu 20.04 LTS. The table shows that the key generation, signing, and verification algorithms of our scheme are faster for every security level, compared to those of Dilithium. We choose the modulus q satisfying the condition $q \equiv 1 \mod 2n$, which enables the "fully-splitting" NTT algorithm for the multiplication operation. Even though our modulus q is much larger than the modulus of Dilithium ($q \approx 2^{23}$), our scheme is faster than Dilithium because of the simplicity of our scheme.

A polynomial multiplication and a sampling are the most costly operations in implementing a cryptosystem. Both Dilithium and our scheme apply the NTT to optimize polynomial multiplication. The main operation performed in both schemes is a multiplication of a public key ($\boldsymbol{A} \in R_q^{k \times m}$ in Dilithium and $\boldsymbol{a} \in R_q^m$ in our scheme). In the NIST security level III, the multiplication $\boldsymbol{A} \cdot \boldsymbol{y}$ involves 30 polynomial multiplications in Dilithium, whereas $\boldsymbol{a} \cdot \boldsymbol{y}$ involves only 4 polynomial multiplications in our scheme. Since one polynomial multiplication for each scheme costs similarly, our multiplication operation of a public key is approximately 7 times faster than that of Dilithium.

Secondly, Dilithium needs to sample $\lceil \log_2 q \rceil \cdot n \cdot k \cdot m \approx 22,080$ bytes to generate a public matrix $\boldsymbol{A} \leftarrow R_q^{k \times m}$. Meanwhile, our scheme needs to sample $\lceil \log_2 q \rceil \cdot n \cdot m \approx 7,680$ bytes to generate $\boldsymbol{a} \in R_q^m$. As a result, the sampling time of our scheme is approximately 3 times faster than that of Dilithium. These points make our scheme faster than Dilithium overall.

5 Security Analysis

5.1 Security Proof

The concrete security of our scheme is supported by a security reduction that gives a reduction from the TMO problem of GCK function to the unforgeability under chosen-message attack in the random oracle model.

Theorem 1. *If our signature scheme is insecure against chosen message attacks for the proposed parameters, then there is a polynomial-time algorithm that can solve the TMO problem of GCK function with success probability at least* $\epsilon'\left(\frac{\epsilon'}{q_s+q_h} - \frac{1}{2^\lambda}\right)$ *where* $\epsilon' = \mathsf{Adv}_{\mathsf{PKS}}^{UF\text{-}CMA}(\mathcal{A}) + \frac{q_s}{(2B-1)^n}$.

The security proof of this theorem is given in Appendix C.

5.2 Concrete Security Analysis

The BKZ lattice reduction algorithm [11] is the most important building block in most efficient SIS attacks. Thus, our estimator determines the overall hardness against the SIS solvers by estimating the cost of the BKZ algorithm. There are a variety of approaches to cost the running time of BKZ [2, 4, 11]. In general, an SVP solver is the main building block of the BKZ algorithm. Regarding the number of SVP oracle calls the BKZ algorithm makes, the Core-SVP model [4] is to assume that an SVP oracle is required to be called only once in a conservative model. This methodology is significantly more conservative than prior ones used in lattice-based cryptography. In particular, we assume that an adversary can run the asymptotically best algorithms known, with no overhead compared to the asymptotic run-times. In particular, we assume that the adversary can cheaply handle huge amounts of (possibly quantum) memory. This conservatism is in line with the goal of long-term post-quantum security. The best known classical SVP solver runs in time $\approx 2^{0.292 \cdot b}$ and best known quantum SVP solver runs in time $\approx 2^{0.265 \cdot b}$. Therefore, we determined to adopt the BKZ cost model of $0.292b$ for the classical model and the BKZ cost model of $0.265b$ for the quantum model where b is the BKZ block size. The security parameters in Table 2 are based on this conservative methodology.

As we mentioned above, it is assumed that the adversary can run the asymptotically best algorithms known, with no overhead compared to the asymptotic run-times in Core-SVP model. To estimate the cost of schemes more rigorously, Kyber [9] and Dilithium [13] refine the core-SVP methodology to count the number of gates required to solve lattice problems (e.g. MLWE) by relying on the concrete estimation for the cost of sieving in gates and by accounting for the number of calls to the SVP oracle. The required security level specified by NIST is based on the classical and quantum gate counts for the optimal key recovery and collision attacks on AES and SHA3, respectively. For example, security level 1, 3 and 5 are defined in terms of block ciphers AES128, AES192, AES256 and $2^{143}, 2^{207}$ and 2^{272} classical gates are required, respectively. Also, categories 2 and

4 are defined in terms of hash functions SHA-256 and SHA-384 and 2^{146} and 2^{210} classical gates are required for the corresponding security level, respectively.

With this refined estimates for the MLWE hardness, Dilithium present three parameters for the NIST security level II, III, and V with 158, 216 and 285 security (=$\log_2$(gates)) for MLWE hardness. Applying this analysis to the MSIS problem is not complete yet but it is strongly expected that the refined cost of SIS would be somewhat larger than the cost of LWE [13] when they have the same BKZ blocksize b. Based on this observation, we believe that our scheme satisfies their stated "NIST Security Level" designation as long as Kyber and Dilithium satisfy their security level since our parameter sets require larger BKZ block sizes than [13] for the same security categories.

5.3 Cost of Known Attacks

Forgery Attack: Solving the GCK-TMO problem. The attacker may attempt to forge a signature. By Theorem 1, this implies finding $(\boldsymbol{x}, \tilde{c})$ such that $F_{\boldsymbol{a}}(\boldsymbol{x}) = \tilde{c} \cdot t$, $\|\tilde{c}\|_\infty \leq \alpha$ where $\alpha = 2$ and $\|\boldsymbol{x}\|_\infty \leq \beta$ where $\beta = 2(B - L_s)$. As a result, it follows that $\Pr[\delta_2] \leq \mathsf{Adv}^{\mathrm{TMO}}_{m,q,2,2(B-L_s)}$.

To estimate the hardness of the TMO problem of GCK function upon which the security of our signature scheme is based, we follow the way explained in [13], which is a software to estimate the hardness of infinite norm SIS problem. This approach seems naturally since there are the reductions from MSIS problem to the CR problem of GCK function and from the CR problem to the TMO problem of GCK function. To our best knowledge, since there is no algorithm to solve TMO problem efficiently, we estimate the concrete security level using the algorithms to estimate the hardness of SIS problem instead. While the MSIS problem is defined over polynomial rings, the best attacks are applied by simply considering the problem as SIS problem since we do not currently have any attack for this ring structure. Note that the MSIS instance can be mapped to a SIS instance by considering the matrix $\boldsymbol{A} \in \mathbb{Z}_q^{(n \times n \cdot m)}$ with infinity norm bounds $\beta = 2(B - L_s)$.

Key-Recovery Attack: Solving the GCK-OW problem. The attacker may also attempt to recover the secret key $\boldsymbol{s}$ from the public key $\boldsymbol{a}, t = \sum_{i=1}^{m}(a_i \cdot s_i)$. As each of the m elements of the secret polynomials is sampled from $R^m_{[-\eta,\eta]}$, this amounts to solving the OW problem of GCK function, which is reduced to the MSIS problem. Note that the smaller the bound parameter in SIS problem, the harder the SIS problem is. Since η is much smaller than β in our parameter setting, key-recovery attack is concretely harder than the forgery attack. Therefore, we set our parameters to provide enough hardness for at least the forgery attack as described above.

6 Conclusion

We proposed a simple and practical signature scheme based on the hardness of the TMO problem of GCK function in ideal lattices. Dilithium [13], which

is one of NIST finalist candidates for PQC signature schemes, has short public key and signature sizes. However, the scheme has a complicated structure with a public key compression technique. Meanwhile, Lyubashevsky's signature scheme has a much simpler structure to implement. However, the scheme has large public key and signature sizes resulted mainly due to the additional security requirement. To combine the strength of both schemes, we modified the scheme of Lyubashevsky slightly and presented a new security proof eliminating the security requirement. Because of the absence of this requirement, our parameter sets enable our scheme to have small public key, secret key, and signature sizes. Besides, we presented a tight analysis of bounding parameters L_s using CLT and error function analysis, which results in a decrease in the public key and secret key sizes as well. Furthermore, our scheme has a very simple and compact structure to implement, which results in the fast and efficient implementation. As a result, our signature scheme has small key and signature sizes and a simple and efficient structure to implement compared to that of Dilithium for the NIST security level II, III, and V.

References

1. Akleylek, S., Bindel, N., Buchmann, J.A., Krämer, J., Marson, G.A.: An efficient lattice-based signature scheme with provably secure instantiation. In: Progress in Cryptology - AFRICACRYPT 2016. Lecture Notes in Computer Science, vol. 9646, pp. 44–60. Springer (2016)
2. Albrecht, M.R., Player, R., Scott, S.: On the concrete hardness of learning with errors. J. Math. Cryptol. **9**(3), 169–203 (2015)
3. Alkim, E., Bindel, N., Buchmann, J.A., Dagdelen, Ö.: TESLA: tightly-secure efficient signatures from standard lattices. Cryptology ePrint Archive, Report 2015/755 (2015), `http://eprint.iacr.org/2015/755`
4. Alkim, E., Ducas, L., Pöppelmann, T., Schwabe, P.: Post-quantum key exchange - A new hope. In: 25th USENIX Security Symposium. pp. 327–343. USENIX Association (2016)
5. Bai, S., Galbraith, S.D.: An improved compression technique for signatures based on learning with errors. In: Topics in Cryptology - CT-RSA 2014. Lecture Notes in Computer Science, vol. 8366, pp. 28–47. Springer (2014)
6. Barreto, P.S.L.M., Longa, P., Naehrig, M., Ricardini, J.E., Zanon, G.: Sharper ring-lwe signatures. Cryptology ePrint Archive, Report 2016/1026 (2016), `http://eprint.iacr.org/2016/1026`
7. Bellare, M., Neven, G.: Multi-signatures in the plain public-key model and a general forking lemma. In: ACM Conference on Computer and Communications Security - CCS 2006. pp. 390–399. ACM (2006)
8. Bindel, N., Akleylek, S., Alkim, E., Barreto, P.S.L.M., Buchmann, J., Eaton, E., Gutoski, G., Kramer, J., Longa, P., Polat, H., Ricardini, J.E., Zanon, G.: Submission to NIST's post-quantum project: Lattice-based digital signature scheme qTESLA. In: [Online].Available: https://qtesla.org/, Accessed: Nov. 03, 2018 (2017)
9. Bos, J.W., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Schwabe, P., Seiler, G., Stehlé, D.: CRYSTALS - kyber: A CCA-secure module-lattice-based KEM. In: EuroS&P. pp. 353–367. IEEE (2018)

10. Bruinderink, L.G., Hülsing, A., Lange, T., Yarom, Y.: Flush, gauss, and reload - A cache attack on the BLISS lattice-based signature scheme. In: Cryptographic Hardware and Embedded Systems - CHES 2016. Lecture Notes in Computer Science, vol. 9813, pp. 323–345. Springer (2016)
11. Chen, Y., Nguyen, P.Q.: BKZ 2.0: Better lattice security estimates. In: Advances in Cryptology - ASIACRYPT 2011. Lecture Notes in Computer Science, vol. 7073, pp. 1–20. Springer (2011)
12. Dagdelen, Ö., Bansarkhani, R.E., Göpfert, F., Güneysu, T., Oder, T., Pöppelmann, T., Sánchez, A.H., Schwabe, P.: High-speed signatures from standard lattices. In: Progress in Cryptology - LATINCRYPT 2014. Lecture Notes in Computer Science, vol. 8895, pp. 84–103. Springer (2014)
13. Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schwabe, P., Seiler, G., Stehlé, D.: CRYSTALS-Dilithium algorithm specifications and supporting documentation. In: [Online].Available: https://pq-crystals.org/dilithium/resources.shtml, Submission to round 2 of the NIST post-quantum project (2018)
14. Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schwabe, P., Seiler, G., Stehlé, D.: CRYSTALS-Dilithium: A lattice-based digital signature scheme. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2018**(1), 238–268 (2018)
15. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Advances in Cryptology - CRYPTO '86. Lecture Notes in Computer Science, vol. 263, pp. 186–194. Springer (1986)
16. Fouque, P., Hoffstein, J., Kirchner, P., Lyubashevsky, V., Pornin, T., Prest, T., Ricosset, T., Seiler, G., Whyte, W., Zhang, Z.: Falcon: Fast-fourier lattice-based compact signatures over NTRU. In: [Online]. Available: https://falcon-sign.info/falcon.pdf (2017)
17. Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. In: ACM Symposium on Theory of Computing - STOC 2008. pp. 197–206. ACM (2008)
18. Güneysu, T., Lyubashevsky, V., Pöppelmann, T.: Practical lattice-based cryptography: A signature scheme for embedded systems. In: Cryptographic Hardware and Embedded Systems - CHES 2012. Lecture Notes in Computer Science, vol. 7428, pp. 530–547. Springer (2012)
19. Impagliazzo, R., Naor, M.: Efficient cryptographic schemes provably as secure as subset sum. J. Cryptol. **9**(4), 199–216 (1996)
20. Lyubashevsky, V.: Fiat-Shamir with aborts: Applications to lattice and factoring-based signatures. In: Advances in Cryptology - ASIACRYPT 2009. Lecture Notes in Computer Science, vol. 5912, pp. 598–616. Springer (2009)
21. Lyubashevsky, V.: Lattice signatures without trapdoors. In: Advances in Cryptology - EUROCRYPT 2012. Lecture Notes in Computer Science, vol. 7237, pp. 738–755. Springer (2012)
22. Lyubashevsky, V., Micciancio, D.: Generalized compact knapsacks are collision resistant. In: International Colloquium on Automata, Languages, and Programming - ICALP 2006. pp. 144–155. Springer (2006)
23. Lyubashevsky, V., Seiler, G.: Short, invertible elements in partially splitting cyclotomic rings and applications to lattice-based zero-knowledge proofs. In: Advances in Cryptology - EUROCRYPT 2018. pp. 204–224. Springer (2018)
24. Micciancio, D.: Generalized compact knapsacks, cyclic lattices, and efficient one-way functions. computational complexity **16**(4), 365–411 (2007)
25. Micciancio, D., Mol, P.: Pseudorandom knapsacks and the sample complexity of LWE search-to-decision reductions. In: Advances in Cryptology - CRYPTO 2011. Lecture Notes in Computer Science, vol. 6841, pp. 465–484. Springer (2011)

26. Peikert, C., Rosen, A.: Efficient collision-resistant hashing from worst-case assumptions on cyclic lattices. In: Theory of Cryptography Conference - TCC 2006. pp. 145–166. Springer (2006)
27. Pessl, P.: Analyzing the shuffling side-channel countermeasure for lattice-based signatures. In: Progress in Cryptology - INDOCRYPT 2016. Lecture Notes in Computer Science, vol. 10095, pp. 153–170. Springer (2016)
28. Rachid, E.B., Johannes, B.: High performance lattice-based CCA-secure encryption. Cryptology ePrint Archive, Report 2015/042 (2015), `http://eprint.iacr.org/2015/042`

A Hardness of the TMO problem

First, we observe a rather obvious reduction that bases the hardness of solving the TMO problem on the hardness of the OW problem. The following theorem shows that the $\mathsf{OW}_{n,m,q,\beta}$ problem is as hard as the $\mathsf{TMO}_{n,m,q,\alpha,\beta}$ problem for $\alpha \geq 1$.

Theorem 2. *Suppose there is a PPT algorithm $\mathcal{A}$ that solves the $\mathsf{OW}_{m,q,\beta}$ problem with advantage $\epsilon(k)$. Then there is a PPT algorithm $\mathcal{B}$ that solves the $\mathsf{TMO}_{m,q,\alpha,\beta}$ problem with the same advantage $\epsilon(k)$ for any $\alpha \geq 1$.*

Proof. Algorithm $\mathcal{B}$ takes $\boldsymbol{a} \in R_q^m$ and $t \in R_q$ as input and needs to find $\boldsymbol{x}$ and c such that $F_{\boldsymbol{a}}(\boldsymbol{x}) = c \cdot t$, $\|c\|_\infty \leq \alpha$, and $\|\boldsymbol{x}\|_\infty \leq \beta$. Algorithm $\mathcal{B}$ runs algorithm $\mathcal{A}$ on input $(\boldsymbol{a}, t)$. With advantage $\epsilon(k)$, algorithm $\mathcal{A}$ outputs $\boldsymbol{x} \in R_q^m$ such that $\|\boldsymbol{x}\|_\infty \leq \beta$ and $F_{\boldsymbol{a}}(\boldsymbol{x}) = t$. Algorithm $\mathcal{B}$ sets $c = 1 \in R_q$. Then it is easy to check that a pair $(\boldsymbol{x}, c)$ is a solution for the $\mathsf{TMO}_{m,q,\alpha,\beta}$ problem when $\alpha \geq 1$. That is, $\mathsf{Adv}^{\mathrm{OW}}_{m,q,\beta} \leq \mathsf{Adv}^{\mathrm{TMO}}_{m,q,\alpha,\beta}$ for any $\alpha \geq 1$. □

In the following theorem, we derive upper bound on the $\mathsf{Adv}^{\mathrm{TMO}}_{m,q,\alpha,\beta}$.

Theorem 3. *Let $n, m, q \in \mathbb{N}$, $\alpha, \beta, \gamma \in \mathbb{R}$ satisfying that $\gamma \ll \beta$ and $n\alpha\gamma \leq \beta$. For x and y are chosen uniformly at random from $R_{[-\alpha,\alpha]}$, if $(2\beta+1)^n \gg q^n$, then it holds that $\mathsf{Adv}^{TMO}_{m,q,\alpha,\beta} \leq 2\mathsf{Adv}^{CR}_{m,q,\beta}$.*

Proof. Suppose there is a PPT algorithm $\mathcal{A}$ that solves the $\mathrm{TMO}_{m,q,\alpha,\beta}$ problem with advantage $\mathsf{Adv}^{\mathrm{TMO}}_{m,q,\alpha,\beta}$ for any $\alpha \geq 1$. Recall that algorithm $\mathcal{A}$ takes $(\boldsymbol{a}, t)$ as input and find a pair $(\boldsymbol{x}, c)$ such that $F_{\boldsymbol{a}}(\boldsymbol{x}) = c \cdot t$. As described in [28, Section 3.3], the probability that an element chosen uniformly random in R_q is in the subset of the multiplicative invertible elements of R_q is given by $(1 - 1/q)^n$ [§]. This probability is overwhelming for our parameter setting. Thus we assume that c has an inverse element in R_q. With $(\boldsymbol{x}, c)$ that $\mathcal{A}$ outputs, we set $\boldsymbol{z} = \boldsymbol{x} \cdot c^{-1}$. Note that $F_{\boldsymbol{a}}(\boldsymbol{z}) = F_{\boldsymbol{a}}(\boldsymbol{x} \cdot c^{-1}) = F_{\boldsymbol{a}}(\boldsymbol{x}) \cdot c^{-1} = t$. Let Ω be the set of all possible $(\boldsymbol{x}, c)$ derived from $\mathcal{A}$ when the pair $(\boldsymbol{a}, t)$ is given. We partition the set Ω into two disjoint sets Ω_I and Ω_{II} which are defined as follows.

Case 1: $\Omega_I = \{ (\boldsymbol{x}, c) \in \Omega : \|\boldsymbol{z}\|_\infty > \gamma \}$,

Case 2: $\Omega_{II} = \{ (\boldsymbol{x}, c) \in \Omega : \|\boldsymbol{z}\|_\infty \leq \gamma \}$,
where $\boldsymbol{z} := \boldsymbol{x} \cdot c^{-1}$. Algorithm $\mathcal{A}$'s output $(\boldsymbol{x}, c)$ belongs to either case 1 or case 2. First, we will show that algorithm $\mathcal{B}$ in case 1 solves the $\mathsf{CR}_{m,q,\beta}$ problem using algorithm $\mathcal{A}$ in case 1 as follows:

[§] To make the probability 1, we can change the modulus q satisfying the condition presented in [23, Corollary 1.2.], i.e. $q \equiv 2k + 1 \mod 4k$ for some integer k such that $n \geq k > 1$ and k is a power of 2 (instead of satisfying the condition $q \equiv 1 \mod 2n$ for the fully-splitting NTT algorithm). For example, if we set $k = 16$, any $y \in R_q$ that satisfies $0 < \|y\|_\infty \leq 2$ will always have an inverse in R_q. This implies that the polynomial $x^n + 1$ splits into k factors and we can run "partially-splitting" NTT algorithm for the multiplication operation, which can lead to an increase in the multiplication cost slightly.

1. Choose a random $\boldsymbol{z'} \in R^m_{[-\gamma,\gamma]}$ and compute $t = F_{\boldsymbol{a}}(\boldsymbol{z'})$.
2. Run $\mathcal{A}$ on input $(\boldsymbol{a}, t)$ and get $(\boldsymbol{x}, c)$ from $\mathcal{A}$.
3. Compute $\boldsymbol{x'} = c \cdot \boldsymbol{z'}$ and output $(\boldsymbol{x}, \boldsymbol{x'})$ as a solution for $\mathsf{CR}_{m,q,\beta}$ problem.

For $(\boldsymbol{x}, \boldsymbol{x'})$ to be a solution of the $\mathsf{CR}_{m,q,\beta}$ problem, we need to show that $F_{\boldsymbol{a}}(\boldsymbol{x}) = F_{\boldsymbol{a}}(\boldsymbol{x'})$ first. Since $(\boldsymbol{x}, c)$ is a solution for the TMO problem, it holds that $F_{\boldsymbol{a}}(\boldsymbol{x}) = c \cdot t$. Also, we set $t = F_{\boldsymbol{a}}(\boldsymbol{z'})$ in step 1. By the homomorphic property of GCK function, it holds that $c \cdot t = c \cdot F_{\boldsymbol{a}}(\boldsymbol{z'}) = F_{\boldsymbol{a}}(c \cdot \boldsymbol{z'}) = F_{\boldsymbol{a}}(\boldsymbol{x'})$. Secondly, it holds that $\|\boldsymbol{x}\|_\infty \leq \beta$ by definition and $\|\boldsymbol{x'}\|_\infty \leq \beta$ since it holds that $\|\boldsymbol{x'}\|_\infty \leq n \cdot \|c\|_\infty \cdot \|\boldsymbol{z'}\|_\infty \leq n\alpha\gamma \leq \beta$. Lastly, we will show that $\boldsymbol{x}$ and $\boldsymbol{x'}$ are distinct. Note that $\boldsymbol{z} \neq \boldsymbol{z'}$ since $\|\boldsymbol{z}\|_\infty > \gamma$ and $\|\boldsymbol{z'}\|_\infty \leq \gamma$ by definition. Therefore, $\boldsymbol{x} - \boldsymbol{x'} = c \cdot (\boldsymbol{z} - \boldsymbol{z'}) \neq 0$.

Now, we consider algorithm $\mathcal{A}$ in case 2. We will show that there is algorithm $\mathcal{C}$ that solves the $\mathrm{OW}_{m,q,\gamma}$ problem using algorithm $\mathcal{A}$ in case 2. When algorithm $\mathcal{B}$ takes $(\boldsymbol{a}, t)$ as input, $\mathcal{B}$ runs algorithm $\mathcal{A}$ on input $(\boldsymbol{a}, t)$. Then algorithm $\mathcal{A}$ outputs $(\boldsymbol{x}, c)$. After then, $\mathcal{B}$ outputs $\boldsymbol{z} = \boldsymbol{x} \cdot c^{-1}$ as a solution for the $\mathrm{OW}_{m,q,\gamma}$ problem. The reason why $\boldsymbol{z}$ is a solution for the OW problem is because it holds that $F_{\boldsymbol{a}}(\boldsymbol{z}) = t$ and $\|\boldsymbol{z}\| \leq \gamma$ by definition.

Note that $\mathsf{Adv}^{\mathrm{TMO}}_{m,q,\alpha,\beta}$ is less than the sum of the advantage of algorithm $\mathcal{A}$ in solving the TMO problem in case 1 and the advantage of algorithm $\mathcal{A}$ in solving the TMO problem in case 2. Because we showed that the advantage of algorithm $\mathcal{A}$ in solving the TMO problem in case 1 is less than $\mathsf{Adv}^{\mathrm{CR}}_{m,q,\beta}$ and the advantage of algorithm $\mathcal{A}$ in solving the TMO problem in case 2 is less than $\mathsf{Adv}^{\mathrm{OW}}_{m,q,\gamma}$. Note that $\mathsf{Adv}^{\mathrm{OW}}_{m,q,\gamma} \leq \mathsf{Adv}^{\mathrm{OW}}_{m,q,\beta} \leq \mathsf{Adv}^{\mathrm{CR}}_{m,q,\beta}$. Therefore, we complete the proof. □

B Security Proof of Lyubashevsky's Scheme

We see how to prove that Lyubashevsky's signature scheme [20] is EU-CMA based on the CR problem of GCK function.

Theorem 4 ([20]). *If the signature scheme is insecure against chosen message attacks for the proposed parameters, then there is polynomial-time algorithm that can solve the CR problem of GCK function.*

We sketch the main idea here. Let $\mathcal{A}$ be a forger against the signature scheme. Then it is sufficient to build the polynomial-time algorithm $\mathcal{B}$ that solves the CR problem of GCK function using $\mathcal{A}$. In the first stage of the attack, the adversary $\mathcal{B}$ chooses a random secret key $\boldsymbol{s} \in R_{[-\eta,\eta]}$ and sends the public key pair $\boldsymbol{a}$ and $t = F_{\boldsymbol{a}}(\boldsymbol{s})$ to $\mathcal{A}$. Since $\mathcal{B}$ knows the secret key, $\mathcal{B}$ can perfectly respond $\mathcal{A}$'s signing oracles. In the second stage, $\mathcal{A}$ sends the forgery signature $(\boldsymbol{z}, c)$. A forking lemma argument shows that the reduction can then extract two signatures $(\boldsymbol{z}, c)$ and $(\boldsymbol{z'}, c')$ for $c \neq c'$ such that

$$\sum_{i=1}^{m}(a_i \cdot z_i) - c \cdot t = \sum_{i=1}^{m}(a_i \cdot z'_i) - c' \cdot t$$

Then $\mathcal{B}$ sets $\boldsymbol{x} = \boldsymbol{z} - c \cdot \boldsymbol{s}$ and $\boldsymbol{x'} = \boldsymbol{z'} - c' \cdot \boldsymbol{s}$. We can easily check that $F_{\boldsymbol{a}}(\boldsymbol{x}) = F_{\boldsymbol{a}}(\boldsymbol{x'})$ as follow:

$$\begin{aligned}\sum_{i=1}^{m}(a_i \cdot x_i) &= \sum_{i=1}^{m}(a_i \cdot (\boldsymbol{z} - c \cdot \boldsymbol{s})) = \sum_{i=1}^{m}(a_i \cdot z_i) - c \cdot \sum_{i=1}^{m}(a_i \cdot s_i) \\ &= \sum_{i=1}^{m}(a_i \cdot z_i) - c \cdot t = \sum_{i=1}^{m}(a_i \cdot z'_i) - c' \cdot t \\ &= \sum_{i=1}^{m}(a_i \cdot z'_i) - c' \cdot \sum_{i=1}^{m}(a_i \cdot s_i) = \sum_{i=1}^{m}(a_i \cdot x'_i)\end{aligned}$$

To guarantee that x and x' are distinct, the system parameters satisfy the condition that for a randomly-picked $\boldsymbol{s} \in R^m_{[-\eta,\eta]}$, there is another $\boldsymbol{s'} \in R^m_{[-\eta,\eta]}$ such that $F_{\boldsymbol{a}}(\boldsymbol{s}) = F_{\boldsymbol{a}}(\boldsymbol{s'})$ and the adversary $\mathcal{A}$ can not know the exact secret key the adversary $\mathcal{B}$ picked, which is called the witness-indistinguishability. So, the bound of the secret key η needs to be large enough to satisfy that $(2\eta{+}1)^{nm} \gg q^n$ for the existence of another secret $\boldsymbol{s'}$ and the witness-indistinguishability. The proof in detail is given in [20].

C Security Proof of Our Signature Scheme

Proof. In order to prove the security of our signature scheme under the TMO problem of GCK function, we define a sequence of hybrid games $\mathbf{Game}_0, \ldots, \mathbf{Game}_3$, where $\mathbf{Game}_0$ is an actual game defining the unforgeability of the signature scheme in Chapter 2 and $\mathbf{Game}_2$ is a game in which we can easily bound the success probability of the forger based on the hardness of the TMO problem of GCK function. For $i = 0, \ldots, 2$, we define an event δ_i which corresponds to the probability that the adversary $\mathcal{A}$ successfully outputs a valid forgery in experiment $\mathbf{Game_i}$.

$\mathbf{Game_0}$. In this game, the challenger runs $(pk, sk) \leftarrow \mathsf{KeyGen}(\lambda)$ and returns pk to the adversary $\mathcal{A}$.

Whenever $\mathcal{A}$ asks for a hash query (cmt, μ), the challenger checks if the query has already been asked and returns the same answer if this is the case. If it has not, then the challenger chooses a new value $\hat{c}$ in the challenge space and returns it to $\mathcal{A}$.

Whenever $\mathcal{A}$ asks for a sign query μ, the challenger computes a signature σ as in the signing algorithm, i.e, $\sigma \leftarrow \mathsf{Sign}(sk, \mu)$ and returns $\sigma = (\boldsymbol{z}, \hat{c})$. In doing so, the challenger checks if $H(cmt, \mu)$ has been defined. If not, then the challenger chooses a new value $\hat{c}$ from the challenge space, sets $H(cmt, \mu) = \hat{c}$, and computes $\boldsymbol{z}$ using this value.

Finally, when $\mathcal{A}$ outputs a forgery (μ, σ), where μ was not queried by the signing oracle, the challenger returns $\mathsf{Verify}(pk, \sigma, \mu)$ as the output of the experiment. Thus, we get $\Pr[\delta_0] = \mathsf{Adv}^{\text{UF-CMA}}_{PKS}(\mathcal{A})$ by definition.

Game$_1$. This game is the same as Game 0 except that the sign queries are replaced by a simulation in the random oracle model as Mid sign (see Algorithm 4 below) and hash queries are handled by answering with random values. Since Game 1 is the same as Game 0, except that the challenge value $\hat{c}$ is generated by the random oracle, the adversary can not tell if the sign oracle was answered by Sign algorithm or Mid sign algorithm.

Note that there is a possibility that the hash function responses are not consistent in Game 1. In other words, for the same input, the output of the hash function which was set for a hash oracle can be different from the output of the hash function which was reprogrammed for a sign oracle. The event occurs with the probability that the commitment value in R_q^m chosen by the adversary for the hash oracle is same to the value $\sum_{i=1}^{m}(a_i \cdot y_i)$ with the public key $\boldsymbol{a} = (a_1, \ldots, a_m)$ and randomly sampled $\boldsymbol{y} = (y_1, \ldots, y_m)$ in the Mid sign algorithm. First, we give proof for bounding the following probability.

Lemma 1. *For all w, given a randomly sampled $\boldsymbol{a} \leftarrow R_q^m$, $\Pr_{\boldsymbol{y} \leftarrow R_{[-B,B]}^m}[\sum_{i=1}^{m}(a_i \cdot y_i) = w] \leq (\frac{1}{2B-1})^n$*

Proof. Note that a random polynomial $x \leftarrow R_q$ is invertible in $R_q = \mathbb{Z}_q[X]/(x^n + 1)$ when the polynomial $x^n + 1$ splits into n linear factors with the probability at least $1 - n/q$. Hence the probability that at least one of m polynomials in $\boldsymbol{a} = (a_1, \ldots, a_m) \leftarrow R_q^m$ is invertible is greater than $1 - (n/q)^m$. This probability is overwhelming for our parameter setting. We assume that a_1 is invertible without loss of generality. Then, for all w, we can rewrite the above probability as

$$\Pr_{\boldsymbol{y} \leftarrow R_{[-B,B]}^m}[y_1 = a_1^{-1}(w - \textstyle\sum_{i=2}^{m}(a_i \cdot y_i))] \leq (\frac{1}{2B-1})^n.$$

The inequality follows due to the fact that a value $a_1^{-1}(w - \sum_{i=2}^{m}(a_i \cdot y_i))$ is an element in R_q and the size of the set $R_{[-B,B]}$ is $(2B-1)^n$. □

By summing up the probability over all q_s signing queries, we get $[\Pr[\delta_1] - \Pr[\delta_0]] \leq q_s/(2B-1)^n$.

Algorithm 4: Mid sign

Input : Message m, public key $(\boldsymbol{a}, t)$
Output: Signature $(\boldsymbol{z}, \hat{c})$

1 Choose $\boldsymbol{y}$ uniformly at random from $R_{[-B,B]}$;
2 Choose $\hat{c} \in \{0,1\}^\ell$ uniformly at random;
3 Compute $c = \text{encode}(\hat{c})(\hat{c})$;
4 Compute $\boldsymbol{z} = \boldsymbol{y} + c \cdot \boldsymbol{s}$. If $\|z_i\|_\infty \geq B - L_s$ for some i, then retry at step 1;
5 Compute $w = \sum_{i=1}^{m}(a_i \cdot y_i)$;
6 Re-program the hash oracle $\mathrm{H}(\cdot)$ so that $\mathrm{H}(w, \mu) = \hat{c}$;
7 Return $(\boldsymbol{z}, \hat{c})$;

Game$_2$. This game is the same as Game 1 except that the sign queries are replaced by a simulation in the random oracle model as Simulated sign (see

Algorithm 5 below) and hash queries are handled by answering with random values. Game 2 is the same as Game 1, except the way the value $\boldsymbol{z}$ is generated. In Game 1, $\boldsymbol{y}$ and $\hat{c}$ are sampled from $R^m_{[-B,B]}$, $\{0,1\}^\ell$, respectively. The value $\boldsymbol{z}$ is computed as $\boldsymbol{z} = \boldsymbol{y} + c \cdot \boldsymbol{s}$ and it checks if $\|\boldsymbol{z}\|_\infty \leq B - L_s$. Then, the hash oracle is reprogrammed as $\hat{c} = \mathrm{H}(\sum_{i=1}^m (a_i \cdot y_i), \mu)$ where $c = \mathrm{Enc}(\hat{c})$. On the other hand, in Game 2, $\boldsymbol{z}$ and $\hat{c}$ are sampled from $R^m_{[-B+L_s,B-L_s]}$, $\{0,1\}^\ell$, respectively. Then, the hash oracle is reprogrammed as $\hat{c} = \mathrm{H}(\cdot)$ so that $\mathrm{H}(\sum_{i=1}^m (a_i \cdot z_i) - c \cdot t, \mu)$ where $c = \mathrm{Enc}(\hat{c})$.

We need to check if the adversary can distinguish between the distributions of $(\boldsymbol{z}, c, w)$ in Game 1 and Game 2. In Game 1, the commitment w_{G1}, which is an input of the hash function, is computed as $w_{G1} = \sum_{i=1}^m (a_i \cdot y_i)$ where $y \in R^m_{[-B,B]}$. In Game 2, however, the commitment w_{G2} is computed as $w_{G2} = \sum_{i=1}^m (a_i \cdot z_i)$ - $c \cdot t$ where $z \in R^m_{[-B+L_s,B-L_s]}$. Even though the way how to compute the commitment value is changed, the commitment w_{G2} should be able to be expressed as an inner product of the public parameter $\boldsymbol{a}$ and $\boldsymbol{r}$ for some $\boldsymbol{r} \in R^m_{[-B,B]}$. To guarantee this, we can set $\boldsymbol{r} = \boldsymbol{z} - c \cdot \boldsymbol{s}$ for an unknown secret key $\boldsymbol{s} \in R_{[-\eta,\eta]}$ such that $t = \sum_{i=1}^m (a_i \cdot s_i)$. It is easy to see that $\sum_{i=1}^m (a_i \cdot r_i) = \sum_{i=1}^m (a_i \cdot (\boldsymbol{z} - c \cdot \boldsymbol{s})) = \sum_{i=1}^m (a_i \cdot z_i) - c \cdot t = w_{G2}$. Also, note that $\|c \cdot \boldsymbol{s}\|_\infty$ is less than or equal to L_s under a proper parameter set. Since $\boldsymbol{z}$ is sampled from $R_{[-B+L_s,B-L_s]}$, we get $\|\boldsymbol{r}\|_\infty \leq \|\boldsymbol{z}\|_\infty + \|c \cdot \boldsymbol{s}\|_\infty \leq B - L_s + L_s = B$. As a result, we get $\Pr[\delta_2] = \Pr[\delta_1]$.

Algorithm 5: Simulated sign

Input : Message m, public key $(\boldsymbol{a}, \boldsymbol{t})$
Output: Signature $(\boldsymbol{z}, \hat{c})$

1 Choose $\boldsymbol{z}$ uniformly at random from $R_{[-B+L_s,B-L_s]}$;
2 Choose $\hat{c} \in \{0,1\}^\ell$ uniformly at random;
3 Compute $c = \mathrm{encode}(\hat{c})(\hat{c})$;
4 Compute $w = \sum_{i=1}^m (a_i \cdot z_i) - c \cdot t$;
5 Re-program the hash oracle $\mathrm{H}(\cdot)$ so that $\mathrm{H}(w, \mu) = \hat{c}$;
6 Return $(\boldsymbol{z}, \hat{c})$;

In the next stage, we apply the forking lemma argument of Bellare and Neven [7]. With the probability $\epsilon'\left(\frac{\epsilon'}{q_s + q_h} - \frac{1}{2^\lambda}\right)$ where $\epsilon' = \Pr[\delta_2]$, we obtain two signatures $(\boldsymbol{z}, c)$ and $(\boldsymbol{z'}, c')$ for $c \neq c'$ such that

$$\textstyle\sum_{i=1}^m (a_i \cdot z_i) - c \cdot t = w,\ \sum_{i=1}^m (a_i \cdot z'_i) - c' \cdot t = w.$$

We set $\boldsymbol{x}$ as $\boldsymbol{x} = \boldsymbol{z} - \boldsymbol{z'}$ and $\tilde{c}$ as $\tilde{c} = c - c'$. We can easily check that $\|\boldsymbol{x}\|_\infty \leq 2(B - L_s)$ and $\|\tilde{c}\|_\infty \leq 2$. Also, we can check that $F_{\boldsymbol{a}}(\boldsymbol{x}) = \tilde{c} \cdot t$ as follow:

$$\begin{aligned} F_{\boldsymbol{a}}(\boldsymbol{x}) &= F_{\boldsymbol{a}}(\boldsymbol{z} - \boldsymbol{z'}) = (w + c \cdot t) - (w + c' \cdot t) \\ &= (c - c') \cdot t = \tilde{c} \cdot t \end{aligned}$$

As a result, we proved that forging a signature implies solving the TMO problem of GCK function, i.e. finding an x and $\tilde{c}$ such that $F_{\boldsymbol{a}}(\boldsymbol{x}) = \tilde{c} \cdot t$, $\|\tilde{c}\|_\infty \leq \alpha$ where $\alpha = 2$ and $\|\boldsymbol{x}\|_\infty \leq \beta$ where $\beta = 2(B - L_s)$. As a result, it follows that the probability $\epsilon'\big(\frac{\epsilon'}{q_s+q_h} - \frac{1}{2^\lambda}\big) \leq \mathsf{Adv}^{\mathrm{TMO}}_{m,q,2,2(B-L_s)}$. This completes the proof. □