Titanium: Proposal for a NIST Post-Quantum Public-key Encryption and KEM Standard
Specifications Document Version 1.1

Ron Steinfeld and Amin Sakzad and Raymond Kuo Zhao
Faculty of Information Technology
Monash University, Australia

December 27, 2018

Abstract

This document specifies Titanium, a proposed pair of algorithms for a NIST standard for post-quantum public-key encryption and Key Encapsulation Mechanism (KEM): Titanium-CPA a chosen-plaintext attack secure (IND-CPA) public-key encryption with 256 bit message space, and Titanium-CCA and a chosen ciphertext attack secure (IND-CCA) KEM with 256 bit key space[1]. In addition to complete algorithm specifications, this document contains the design rationale, recommended parameter sets, algorithm time and memory performance, Known Answer Test (KAT) values, expected security levels of the specified parameter sets, security analysis and known cryptanalytic attacks, and discussion of advantages and limitations.

[1] In some literature IND-CCA is refereed to IND-CCA2.

Contents

Chapter 1

Introduction

Background. Lattice-based cryptography relies in great parts on the assumed hardness of two well-studied and closely related problems: the Small Integer Solution problem (SIS) introduced in [Ajt96] and the Learning With Errors problem (LWE) introduced in [Reg09]. They lead to numerous cryptographic constructions, are conjectured exponentially hard to solve even for quantum algorithms, and enjoy reductions from standard worst-case lattice problems such as finding a short non-zero vector in a lattice (ApproxSVP). However, the resulting cryptographic constructions suffer from large keys and/or rather inefficient algorithms. This is because the problems themselves involve large-dimensional random matrices over a ring $\mathbb{Z}_q$ (for some $q \geq 2$).

To obtain more efficient SIS-based primitives, Lyubashevsky and Micciancio [LM06], and Peikert and Rosen [PR06] introduced the Polynomial SIS problem (PSIS), inspired from [Mic07, HPS98]. The problem was called Ideal-SIS in [LM06], Cyclotomic-SIS in [PR06], and is now commonly referred to as Ring-SIS. We prefer to call it PSIS as it is not defined in terms of number fields but polynomial rings (as opposed to RLWE), similarly to the Polynomial-LWE problem (PLWE) we consider in this work. It is possible to define a SIS variant of RLWE, i.e., involving number fields: in the common case of power-of-2 cyclotomics, PSIS and RSIS match (as do PLWE and RLWE). In this work, we are interested in larger classes of polynomials, making the distinction important. PSIS^f can be described in terms of elements of $\mathbb{Z}_q[x]/f$ for an integer $q \geq 2$ and a polynomial f that parametrizes the problem. Equivalently, it may be described as SIS where the uniform matrix is replaced by a structured matrix (the precise structure depends on f). PSIS allows the design of fast digital signatures, among other applications (see [Lyu09], for example).

This approach was extended to LWE by Stehlé *et al.* [SSTX09], who introduced and studied the (search version of) Polynomial-LWE problem (PLWE). It was originally called Ideal-LWE, by analogy to Ideal-SIS (the decision version of PLWE and the name Polynomial-LWE were defined in [BV11]). Lyubashevsky *et al.* [LPR13] introduced the RLWE problem, which involves number fields rather than polynomials, and proposed a reduction from its search to decision versions, in the case of cyclotomic polynomials. (See also [EHL14, CLS15] for extensions to larger classes of fields of the RLWE search to decision reduction.) Power-of-2 cyclotomic polynomials (for which PLWE and RLWE match) have been exploited to design fast encryption schemes, among others (see [ADPS16], for example). Cryptographic schemes based on PLWE/RLWE most often enjoy keys of $\widetilde{O}(\lambda)$ bit-sizes and algorithms with $\widetilde{O}(\lambda)$ runtime, where λ refers to the security parameter (i.e., all known attacks run in time $\geq 2^\lambda$) and the $\widetilde{O}(\cdot)$ notation hides poly-logarithmic factors.

Switching from unstructured SIS and LWE to their polynomial counterparts PSIS and PLWE has undeniable efficiency advantages. However, the security guarantees are severely degraded. PSIS and PLWE also enjoy reductions from worst-case lattice problems such as ApproxSVP, but these lattice problems, e.g., $\mathsf{ApproxSVP}^f$, are restricted to lattices that correspond to ideals of $\mathbb{Z}[x]/f$, where f is the polynomial that parametrizes PSIS and PLWE: under some conditions on f, there exists a reduction from $\mathsf{ApproxSVP}^f$ with small approximation factor, to PSIS^f and PLWE^f (see [LM06, PR06, SSTX09]). It is entirely possible that $\mathsf{PSIS}^{(f)}/\mathsf{PLWE}^{(f)}$ could be easy to solve for some polynomials

f, and hard for others. We note that the stability of the polynomial rings under multiplication by x can be exploited to accelerate some known lattice algorithms by small polynomial factors, but we are interested here in more drastic weaknesses. For instance, if f has a linear factor over the integers, then it is well-known that $\mathsf{PSIS}^{(f)}/\mathsf{PLWE}^{(f)}$ are computationally easy (we note that the reductions from $\mathsf{ApproxSVP}^{(f)}$ require f to be irreducible). Finding weak f's for PLWE has been investigated in [EHL14, ELOS15, CLS15, CLS16]. The attacks presented in this sequence of articles were used to identify such f's, but they only work for error distributions with small width relative to the geometry of the corresponding ring [CIV16b, CIV16a, Pei16a]. In another sequence of works, Cramer *et al.* [CDPR16, CDW16] showed that $\mathsf{ApproxSVP}^f$ is easier for f a cyclotomic polynomial of prime-power conductor than for general lattices. More concretely, the authors of [CDW16] give a quantum polynomial-time algorithm for $\mathsf{ApproxSVP}^f$ with approximation factor $2^{\widetilde{O}(\sqrt{n})}$, where n is the degree of f. As a comparison, for such approximation factors and arbitrary lattices, the best known algorithms run in time $2^{\widetilde{O}(\sqrt{n})}$ (see [Sch87]). Finally, we note that the choice of non-cyclotomic polynomials in [BCLvV16] was motivated by such weaknesses. Even though the results in [CDPR16, CDW16] impact $\mathsf{ApproxSVP}^f$, it may be argued that it could have implications for $\mathsf{PLWE}^{(f)}$ as well, possibly even for lower approximation factors. On the other hand, it could be that similar weaknesses exist for $\mathsf{ApproxSVP}^f$ considered in [BCLvV16], although none is known at the moment. This lack of understanding of which f's correspond to hard PLWE^f problems motivated researchers to take three different directions:

- **Approach 1, Security Oriented:** To achieve stronger security guarantees, the structured RLWE can be replaced by the unstructured LWE problem, whose security is related to more well understood unstructured lattices. However, this comes at the cost of a significant performance penalty, although encryption scheme optimisations for variants of Regev's LWE-based cryptosystem have been used to reduce this cost penalty in the Frodo proposal [BCD+16].

- **Approach 2, Performance Oriented:** To achieve better security guarantees without a significant performance penalty, RLWE can be replaced by Module-RLWE [LS15] (e.g. Kyber [BDK+17, DLL+17]). Although the performance has not been greatly sacrificed compared to RLWE-based schemes such as [ADPS16], the security of this approach still relies on the hardness of the Module-RLWE problem over a single fixed ring R. This means that any attacks exploiting RLWE weaknesses inherent to the ring R could translate to an attack on the Module-RLWE problems for R-modules too [AD17]. Overall, it is not entirely clear if this approach can buy much more security than RLWE over a single ring.

- **Approach 3, Intermediate:** This approach seeks to achieve an intermediate point in the 'security-guarantee vs. efficiency' trade-off curve, sitting in between Approach 1 and 2 above. To obtain better security guarantees than reliance on RLWE (or Module-RLWE) over a single fixed ring, this approach, initiated by Lyubashevsky [Lyu16] for the design of digital signatures and extended by Rosca et al. [RSSS17] for design of public-key encryption, aims at problems that are provably as hard as PLWE^f for the hardest f in a *large family of polynomials*, to hedge against the weakness of specific polynomial rings, while achieving a better efficiency than schemes based on unstructured LWE. Our proposal Titanium is based on this 'intermediate' approach, as a practical instantiation of the results in [RSSS17].

Lattice Problems as Hard as PLWE^f for the hardest f in a large family. Lyubashevsky [Lyu16] introduced a variant $R^{<n}$-SIS of SIS that is not parametrized by a polynomial f and which enjoys the following desirable properties. First, an efficient algorithm for $R^{<n}$-SIS with degree bound n leads to an efficient algorithm for PSIS^f for all f's in a family of polynomials of size exponential in n. Second, there exists a signature scheme which is secure under the assumption that $R^{<n}$-SIS is hard, involves keys of bit-size $\widetilde{O}(\lambda) = \widetilde{O}(n)$ and whose algorithms run in time $\widetilde{O}(\lambda)$. In this sense, $R^{<n}$-SIS can serve as an alternative cryptographic foundation that hedges against the risk that PSIS^f is easy to solve for some f (as long as it stays hard for some f in the family).

The main contribution of [RSSS17] is the introduction of an LWE counterpart to Lyubashevsky's $R^{<n}$-SIS problem, called *Middle-Product Learning With Errors* (MP-LWE). Let $n, q \geq 2$. We let $\mathbb{Z}_q^{<n}[x]$ denote the set of polynomials with coefficients in $\mathbb{Z}_q$ and degree $< n$. For $a \in \mathbb{Z}_q^{<n}[x]$ and $s \in \mathbb{Z}_q^{<2n-1}[x]$, we let $a \odot_n s = \lfloor (a \cdot s \bmod x^{2n-1})/x^{n-1} \rfloor \in \mathbb{Z}_q^{<n}[x]$ denote the polynomial obtained by multiplying a and s and keeping only the middle n coefficients. Middle-Product LWE (MP-LWE), with parameters $n, q \geq 2$ and $\alpha \in (0, 1)$, consists in distinguishing arbitrarily many samples (a_i, b_i) uniform in $\mathbb{Z}_q^{<n}[x] \times \mathbb{Z}_q^{<n}[x]$, from the same number of samples (a_i, b_i) with a_i uniform in $\mathbb{Z}_q^{<n}[x]$ and $b_i = a_i \odot_n s + e_i$, where each e_i is sampled from the binomial difference distribution with parameter η, $\mathsf{BinDiff}(\eta)$, and s is uniformly chosen in $\mathbb{Z}_q^{<2n-1}[x]$.

A reduction from (decision) PLWE^f to (decision) MP-LWE of parameter n, for every monic f of degree n whose constant coefficient is coprime with q is given in [RSSS17]. However, in the reduction presented in [RSSS17], the noise parameter is amplified by the reduction by a factor linear in the so-called 'Expansion Factor' of f, introduced in [LM06]. The noise parameter in MP-LWE can for example be set to handle all monic polynomials $f = x^n + g$ with constant coefficient coprime with q, $\deg g \leq n/2$ and $\|g\| = n^c$ for an arbitrary constant $c > 0$. For any $c \geq 1$, this set of f's has exponential size in n. However, in general the reduction is not tight due to the reduction's noise amplification factor, and does not give strong concrete security guarantees. We note that similar restrictions involving the expansion factor appeared before in [LM06, SSTX09].

Our Proposal: Titanium. In this proposal, we specify Titanium-CPA, a practical and optimised public-key encryption scheme based on the MP-LWE problem, building on the theoretical results reported in [RSSS17] that exploit the associativity property of the middle product. We optimise both the encryption scheme in [RSSS17], its parameters and implementation, and its security proof. By specializing the reduction of [RSSS17] to a suitable polynomial family $\mathcal{F}$, we give a *tight* security reduction to the IND-CPA security of Titanium-CPA from the PLWE^f hardness assumption with respect to any f in the family $\mathcal{F}$ of exponential size in the security parameter (the reduction also assumes the classical random oracle model for an underlying hash function). We use a variant of the generic Fujisako-Okamoto transformation [FO99, HHK17] to tightly convert Titanium-CPA to a Key Encapsulation Mechanism (KEM) called Titanium-CCA that is secure under adaptive chosen-ciphertext attacks (i.e. IND-CCA secure), assuming the classical random oracle model for the underlying hash functions. Our optimised constant-time implementations of Titanium-CPA and Titanium-CCA employ fast algorithms based on the Number Theoretic Transform (NTT) for computing the middle product operations, optimised randomness distribution parameters, and precomputation optimisation techniques to improve the scheme efficiency.

We emphasize several notable features of our proposal:

- Strong concrete security guarantees: Our security reduction is tight in terms of both reduction time and noise amplification, in fact it preserves the noise distribution exactly. This tightness makes it feasible to incorporate this security proof into our parameter selection procedure, and not only as asymptotical evidence for security. In particular, our choice of parameters together with our security reduction gives concrete *lower bounds* on the security of Titanium-CPA in terms of the security of the hardest PLWE^f problem over all exponentially many f's in the family $\mathcal{F}$. Our family $\mathcal{F}$ contains polynomials of the form $x^m + \sum_{i \leq \ell(m)} f_i x^i$ for some $\ell(m)$, including specific f's previously used in lattice-based cryptography, such as power-of-2 cyclotomics ($x^m + 1$ for m a power of 2) and 'NTRUPrime' polynomials [BCLvV16] ($x^m - x - 1$ for m prime), among many others. For example, for our Std128 parameter set, $\mathcal{F}$ contains polynomials of any degree in the interval $[896, 1024]$, including the cyclotomic power-of-2 polynomial $x^{1024} + 1$, as used in New Hope [ADPS16].

- Conservative choice of parameters: Our parameter selection makes conservative assumptions, to allow safety margins for future improvements in cryptanalysis techniques and computing architectures. In particular, we adopt a conservative variant of the 'Core SVP hardness' approach [ADPS16, BCD+16] in deriving complexity estimates for lattice reduction attacks based

on the BKZ algorithm [SE94]. This approach only counts the cost of a Shortest Vector Problem (SVP) call of BKZ, neglecting other BKZ costs. Other conservative choices include a random access machine (RAM) model neglecting the cost of each memory access, which is also made in the 'Core SVP hardness' approach [ADPS16]. Our approach is even more conservative than that of [ADPS16] by allowing for the possibility that a variant of sieve SVP algorithm might be devised to generate as many 'pseudorandom' short vectors as the time complexity rather than the memory complexity of the algorithm. We use attack time to advantage ratio as the main complexity cost measure, neglecting memory cost to allow for future cryptanalytic improvements that reduce the attack memory requirements. We also allow an 10% quantum security safety margin and 5% classical security safety margin in setting our parameters for a target security goal. Further, our parameter settings are based on our PLWE^f-based lower bounds derived from our security proof, but we do not know of attacks with that complexity: our best known attacks on the MP-LWE problem underlying our scheme have a higher complexity.

Further cryptanalysis may clarify whether some of our conservative assumptions might be relaxed. In the latter case, we may consider tweaking our parameter sets to achieve a better efficiency for a given security level/category. For example, for our Std128 parameter set for Titanium-CCA, the PLWE^f lower bound quantum complexity estimates range between 2^{136} and 2^{161} gates and classical complexity from 2^{149} to 2^{176} gates as f ranges over the family $\mathcal{F}$, while the best known attack on our scheme via the MP-LWE problem has quantum complexity estimate 2^{214} gates (resp. 2^{236} gates classically). For our Lite96 parameter set for Titanium-CCA, the best known attack on our scheme via the MP-LWE problem has quantum complexity estimate 2^{164} gates and classical complexity estimate 2^{181} gates, which are already above the $\lambda_Q = 130$ and $\lambda_C = 143$ goals for Std128 (refer to Table 6.10). This implies that we could potentially take parameter sets given for Lite96 to achieve goals for Std128. Our parameter choices for Titanium-CCA also provably ensure a negligible decryption error probability p_e to provably avoid decryption failure chosen-ciphertext attacks, for example, for parameter set Std128 of Titanium-CCA, we have a decryption error probability upper bounded approximately by 2^{-161}.

- Better Efficiency than LWE-based schemes: Our scheme implementations have better efficiency than existing proposals [BCD+16] based on the unstructured LWE problem in terms of both ciphertext length and algorithm run-time (see comparison below).

- Flexible choice of dimensions: In RLWE systems based on 'power-of-2' cyclotomics, such as New Hope [ADPS16], the main parameter for controlling security is the polynomial degree n, which must jump in increasingly large steps due to the power of 2 restriction, limiting flexibility of parameter choices for tuning security. In our Titanium schemes, the analogous LWE dimension parameter n is replaced by an interval $[m_{\min}, n]$ for the degrees of the PLWE^f polynomials in family $\mathcal{F}$ whose security reduces to MP-LWE. In our schemes, there is no 'power-of-2' restriction on the analogous dimension parameter n, allowing a flexible tunability of security versus efficiency (although for maximum efficiency of our optimised radix-2 based fast NTT implementations of middle product computations, n should be close to a multiple of 256). This allows us to generate a range of parameter sets with smoothly increasing dimension n. For example, our Titanium-CCA parameter sets Toy64, Lite96, Std128 have $n = 684, 800, 1024$, respectively.

Comparison of Titanium with other lattice-based schemes. Table 1.1 shows a brief comparison of Titanium-CPA and Titanium-CCA with other lattice-based proposals based on approaches 1 and 2 above (we refer to Tables 3.1-3.2 for the full benchmark results of our Titanium-CPA and Titanium-CCA implementations for all our six parameter sets), showing how our schemes offer an 'intermediate' point in terms of security guarantees versus efficiency. In particular, we point out the following:

- *Efficiency Aspects:* For our Std128 parameter set (corresponding to NIST category 1, or AES128 security level), our Titanium-CPA ciphertexts are significantly smaller in size (3.2 times factor)

Table 1.1: Comparison of Titanium-CPA and Titanium-CCA with Frodo and Kyber.

Scheme	Cl. Sec.	Sec. Guarantees		Eff. Aspects	
		Prob.	Family Size	Size (Bytes)	Cycles
Frodo	130	LWE	n/a	$\|\mathsf{pk}\| = 11296$ $\|\mathsf{sk}\| = 11280$ $\|\mathsf{ct}\| = 11288$	KeyGen : 2938000 Encrypt : 3484000 Decrypt : 338000
Titanium-CPA.Std128	155	MP-LWE	$\geq 3^{256}$	$\|\mathsf{pk}\| = 14720$ $\|\mathsf{sk}\| = 32$ $\|\mathsf{ct}\| = 3520$	KeyGen : 1619550 Encrypt : 1262047 Decrypt : 217612
Titanium-CPA.Std128 (AVX2 optimised)	155	MP-LWE	$\geq 3^{256}$	$\|\mathsf{pk}\| = 14720$ $\|\mathsf{sk}\| = 32$ $\|\mathsf{ct}\| = 3520$	KeyGen : 828542 Encrypt : 742541 Decrypt : 116311
Kyber	161	Module-LWE	1	$\|\mathsf{pk}\| = 1088$ $\|\mathsf{sk}\| = 2368$ $\|\mathsf{ct}\| = 1184$	KeyGen : 276720 Encaps. : 332800 Decaps. : 376104
Titanium-CCA.Std128	134	MP-LWE	$\geq 3^{256}$	$\|\mathsf{pk}\| = 16352$ $\|\mathsf{sk}\| = 16384$ $\|\mathsf{ct}\| = 3552$	KeyGen : 1806119 Encaps. : 1446751 Decaps. : 1671578
Kyber (AVX2 optimised)	161	Module-LWE	1	$\|\mathsf{pk}\| = 1088$ $\|\mathsf{sk}\| = 2368$ $\|\mathsf{ct}\| = 1184$	KeyGen : 77892 Encaps. : 119652 Decaps. : 125736
Titanium-CCA.Std128 (AVX2 optimised)	134	MP-LWE	$\geq 3^{256}$	$\|\mathsf{pk}\| = 16352$ $\|\mathsf{sk}\| = 16384$ $\|\mathsf{ct}\| = 3552$	KeyGen : 934051 Encaps. : 865352 Decaps. : 986905

compared to the LWE-based IND-CPA scheme Frodo [BCD+16] at a higher security level. In handshake protocols, the quantity $|\mathsf{pk}| + |\mathsf{ct}|$ is the main communication size, for which we could save ≈ 4.3 Kilo Bytes (KB). Our key generation, encryption, and decryption time are also faster by factors of 1.4, 2.3, and 1.3 compared to Frodo [BCD+16], respectively. Note that we could even save more in these efficiency aspects once AVX2 optimisation techniques are employed. The p_e for Frodo is set to be 2^{-30}, while ours is 2^{-33}.

We also compare the efficiency aspects of Titanium-CCA (and its AVX2 optimised version) to that of Kyber (and its AVX2 version, respectively). With a smaller quantum security claim, our ciphertexts, secret key, and public key are 3, 6.9, and 15 times larger than the corresponding quantities in Kyber. Our key generation, encapsulation, and decapsulation times are slower by factors of 7.6, 5.0, and 5.1 compared to Kyber [BDK+17], respectively. The AVX2 optimised versions can also be compared accordingly.

- *Security Guarantees:* We have qualitatively achieved/provided better security proof guarantees than other structured (RLWE-based) schemes; Titanium-CPA security is provably (and tightly) as hard as the hardest instance of PLWE in a family of polynomial rings of size at least 3^{256}, hedging against weakness of a few special (e.g. cyclotomic) rings, whereas Kyber [BDK+17] relies on Module-RLWE over a single specific power-of-2 cyclotomic ring in dimension 256.

Chapter 2

Algorithm Specifications

This Chapter contains the full specifications of our Titanium-CPA public-key encryption algorithm and Titanium-CCA Key Encapsulation Mechanism (KEM) algorithm.

2.1 Mathematical and Notation Preliminaries

2.1.1 Polynomials and operations

We use the following notations:

- For $k > 0$, and a ring R, we let $R^{<k}[x]$ denote the set of polynomials with coefficients in R of degree $< k$.
- Given a polynomial $a = a_0 + a_1 x + \cdots + a_{k-1}x^{k-1} \in R^{<k}[x]$, we let

  $$\mathsf{PolToVec}(a) := \mathbf{a} = (a_0, \ldots, a_{k-1})^T \in R^k,$$

  and

  $$\mathsf{Rev}(\mathbf{a}) = (a_{k-1}, \ldots, a_0)^T \in R^k.$$

  The latter notation is extended to the corresponding polynomial too. For two polynomials a and b (not necessarily with same degree), it is easy to check that:

  $$\mathsf{Rev}(a \cdot b) = \mathsf{Rev}(a) \cdot \mathsf{Rev}(b). \tag{2.1}$$

  On the other hand, for any $\mathbf{a} = (a_0, \ldots, a_{k-1})^T \in R^k$, we define:

  $$\mathsf{VecToPol}(\mathbf{a}) := a_0 + a_1 x + \cdots + a_{k-1}x^{k-1} \in R^{<k}[x].$$

- Given two polynomials $a = a_0 + a_1 x + \cdots + a_{k-1}x^{k-1} \in R^{<k}[x]$ and $b = b_0 + b_1 x + \cdots + b_{m-1}x^{m-1} \in R^{<m}[x]$, we denote by $a \cdot b \in R^{<k+m-1}[x]$ the ordinary polynomial product of a and b over $R[x]$.
- Let d_a, d_b, d, k be integers such that $d_a + d_b - 1 = d + 2k$. The middle-product $\odot_d : R[x]^{<d_a} \times R[x]^{<d_b} \to R[x]^{<d}$ is the map:

  $$(a, b) \mapsto a \odot_d b = \left\lfloor \frac{(a \cdot b) \bmod x^{k+d}}{x^k} \right\rfloor,$$

  in which the notation $\lfloor \cdot / x^k \rfloor$ means that we divide by x^k as a power series in x and drop the terms $c_j x^j$ with $j < 0$. We use the same notation $\odot_d$ for every d_a, d_b such that $d_a + d_b - 1 - d$ is non-negative and even.

- Let f be a polynomial of degree m with coefficients in ring R. We define M_f as the (Hankel) matrix in $R^{m\times m}$ such that for any $1 \le i, j \le m$, the coefficient $(\mathsf{M}_f)_{i,j}$ is the constant coefficient of $x^{i+j-2} \bmod f$.

The (reversed) coefficient vector of the middle-product of two polynomials is in fact equal to the product of the Toeplitz matrix associated to one polynomial by the (reversed) coefficient vector of the second polynomial.

Lemma 2.1.1 ([RSSS17]). *Let $d, k > 0$. Let $r \in R^{<k+1}[x]$ and $a \in R^{<k+d}[x]$ and $b = r \odot_d a$. Then $\mathsf{Rev}(\mathbf{b}) = \mathsf{Toep}^{d,k+1}(r) \cdot \mathsf{Rev}(\mathbf{a})$. In other words, we have $\mathbf{b} = \mathsf{Rev}\left(\mathsf{Toep}^{d,k+1}(r) \cdot \mathsf{Rev}(\mathbf{a})\right)$.*

The middle-product is an additive homomorphism when either of its inputs is fixed. As a consequence of the associativity of matrix multiplication and Lemma 2.1.1, the middle-product satisfies the following associativity property, which is crucial to the correctness of Titanium-CPA.

Lemma 2.1.2 ([RSSS17]). *Let $d, k, n > 0$. For all $r \in R[x]^{<k+1}$, $a \in R[x]^{<n+1}$, $s \in R[x]^{<n+d+k}$, we have $r \odot_d (a \odot_{d+k} s) = (r \cdot a) \odot_d s$.*

2.1.2 Number Theoretic Transform (NTT)

Let q be a prime such that $d|(q-1)$ and ω_d be the d-th primitive root of unity in $\mathbb{Z}_q$ for a positive integer d. For $\mathbf{y} = (y_0, \ldots, y_{d-1}) \in \mathbb{Z}_q^d$, we define:

$$\mathsf{NTT}_d(\omega_d, \mathbf{y}) := \mathbf{z} = (z_0, \ldots, z_{d-1}) \in \mathbb{Z}_q^d, \tag{2.2}$$

where

$$z_i = \sum_{j=0}^{d-1} \omega_d^{i\cdot j} y_j, \tag{2.3}$$

for $i = 0, \ldots, d-1$. On the other hand, for $\mathbf{z} \in \mathbb{Z}_q^d$, we also define

$$\mathsf{NTT}_d^{-1}(\omega_d, \mathbf{z}) := \mathbf{y} = (y_0, \ldots, y_{d-1}) \in \mathbb{Z}_q^d, \tag{2.4}$$

where

$$y_i = d^{-1} \cdot \sum_{j=0}^{d-1} \omega_d^{-i\cdot j} z_j, \tag{2.5}$$

for $i = 0, \ldots, d-1$.

2.1.3 Probability

We use the following notations:

- For a finite domain R, we denote by $U(R)$ the uniform probability distribution over R.
- For a positive even integer B, we denote by $\mathsf{ZeIntU}(B)$ the zero-excluded interval uniform distribution $U(\{-B/2, \ldots, -1\} \cup \{1, \ldots, B/2\})$ over $\mathbb{Z}$, i.e. the distribution of an integer with magnitude uniformly random in the interval $\{1, \ldots, B/2\}$ and a uniformly random sign.
- For a positive integer parameter η, we denote by $\mathsf{BinDiff}(\eta)$ the 'binomial difference' distribution over $\mathbb{Z}$ with parameter η, i.e. the distribution of the random variable $X - Y$ when random variables X, Y are independently sampled from the binomial distribution with number of trials parameter η and success probability in each trial parameter $1/2$.
- For a real-valued random variable r, we denote by $\mathbb{E}[r]$ the expected value of r.

2.2 Titanium-CPA: IND-CPA public-key encryption scheme

The design for the IND-CPA secure version of Titanium, to be called here Titanium-CPA, is based on the MP-LWE-based public-key cryptosystem described in Section 4 of [RSSS17].

2.2.1 Outline

The scheme of [RSSS17]

In this section, we outline the main high level ideas behind our scheme Titanium-CPA, and the main (mild) differences between Titanium-CPA and the encryption scheme in [RSSS17]. Following sections contain full reference specifications of Titanium-CPA and Titanium-CCA.

We first recall the main ideas in the scheme of [RSSS17]. The public key consists of t MP-LWE samples of the form

$$\mathsf{pk} = (a_i, b_i = a_i \odot_{d+k} s + e_i)_{1 \le i \le t},$$

where $a_i \in \mathbb{Z}_q^{<n}[x]$ are uniformly random polynomials, $s \in \mathbb{Z}_q^{<n+k+d-1}[x]$ is a uniformly random secret key polynomial, and $e_i \in \mathbb{Z}_q^{<d+k}[x]$ are error polynomials with 'small' coefficients sampled from an appropriate error distribution χ_e, which is a rounded continuous Gaussian distribution in [RSSS17]. The secret key is $\mathsf{sk} = s$. To encrypt a message $m \in \{0,1\}^{<d}[x]$, the encryption algorithm uses an analogue of Regev's encryption scheme [Reg05], computing

$$c_1 = \sum_{1 \le i \le t} r_i \cdot a_i \text{ and } c_2 = \sum_{1 \le i \le t} r_i \odot_d b_i + m \cdot \lfloor q/2 \rfloor,$$

using random polynomials r_i with 'small' coefficients sampled from an appropriate error distribution χ_r, which is uniform on binary coefficients in [RSSS17]. The decryption algorithm decrypts a ciphertext (c_1, c_2) by exploiting the associativity property of middle-product (Lemma 2.1.2):

$$r \odot_d (a \odot_{d+k} s) = (r \cdot a) \odot_d s,$$

which implies that the decryption algorithm can compute

$$c_1 - c_2 \odot_d s = \sum_{1 \le i \le t} r_i \odot_d e_i + m \cdot \lfloor q/2 \rfloor \approx m \cdot \lfloor q/2 \rfloor,$$

since $\sum_{1 \le i \le t} r_i \odot_d e_i$ is 'small' compared to $q/2$ with overwhelming probability for appropriate choice of parameters (see Chapter 7). Then m can be recovered (except with negligible error probability) in decryption by rounding $c_1 - c_2 \odot_d s$ to a multiple of $\lfloor q/2 \rfloor$.

The security of the scheme follows from the hardness of the MP-LWE problem, based on a statistical 'Leftover Hash Lemma' (LHL) argument (see Chapter 6). Namely, the hardness of MP-LWE implies that given the a_i's, the $b_i = a_i \odot_{d+k} s + e_i$ are indistinguishable from uniformly random polynomials in $\mathbb{Z}_q^{<d+k}[x]$. Then, if the r_i's have sufficient min-entropy, an LHL argument can be used to show that if b_i are uniformly random, then $\sum_{1 \le i \le t} r_i \odot_d b_i$ is statistically close to uniform given c_1 and the b_i's, which statistically hides the message m.

Differences between Titanium-CPA and [RSSS17]

We now summarize the main differences between Titanium-CPA and the scheme from [RSSS17]:

- *Reference and Optimised Algorithms and Implementations:* We present both a reference specification of our Titanium-CPA algorithms (in this Chapter) and optimised algorithms (in Chapter 3). The reference and optimised algorithms produce compute exactly the same input/output functionality (and hence their implementation produce identical KATs). However, the reference algorithms and implementation is written in terms of middle-product and polynomial multiplication operations (and NTTs) and use slow quadratic time (but easy-to-understand) classical

algorithms to compute those operations. Our optimised algorithms and implementations use fast (quasi-linear time) optimised algorithms for NTT operations and do not explicitly perform middle-product operations. Our reference algorithms/implementations are only provided to clarify understanding of the specification, and should not be used for performance benchmarking purposes.

- *Simplified Algorithms:* Our NTT precomputation optimisations (see below) modify the input-output functionality of our algorithms, and consequently, our reference specification also involves NTT operations. In addition, the reference specification specifies explicit randomness sampling algorithms and output encoding/packing algorithms. As a consequence, our reference specification is not mathematically concise. Accordingly, we also define simplified variants of our Titanium-CPA and Titanium-CCA schemes in Chapter 4, closely related to the scheme in [RSSS17] summarized above. We show that the simplified schemes are equivalent in security and correctness to the schemes in our reference specification, but are easier to understand and analyse.

- *Optimized r_i distribution χ_r:* In [RSSS17], the r_i's are chosen with uniformly random binary coefficients. In Titanium-CPA, we allow the r_i coefficients to be bigger and tune their variance to optimise the resulting key and ciphertext length, as well as the algorithm run-times, for a given security and decryption error probability level. In particular, although increasing the variance of the r_i's implies a corresponding increase in the decryption noise term $\sum_{1\leq i\leq t} r_i \odot_d e_i$ (which tends to increase the decryption error probability and a corresponding increase in q to compensate), on the other hand the larger entropy of higher variance r_i's reduces the number t of required MP-LWE samples in the public-key to satisfy the LHL entropy condition for the security proof, and a reduced t has a significant improvement on both computation and key length, even if q is increased up to some point. It turns out that optimal values for the variance of the r_i to minimise the public-key length are typically significantly larger than 1. For efficiency of sampling, we sample the r_i coefficients from a uniform distribution on a set $\{-B_1/2, \ldots, 1\} \cup \{1, \ldots, B_1/2\}$ of power-of-2 size $B_1 = 2^{b_1+1}$ (i.e. the distribution $\mathsf{ZeInt U}(B_1)$), but we also allow a fraction of the r_i coefficients to be sampled from $\mathsf{ZeInt U}(B_2)$ with a larger $B_2 = 2^{b_2+1} > B_1$ to allow a smooth tuning of the 'mean' variance of the r_i's, to have a fine control the trade-off between t and q.

- *Optimized e_i distribution χ_e:* In [RSSS17], the e_i error (noise) distribution χ_e is chosen as an integer-rounded continuous Gaussian distribution, but sampling from this distribution tends to be computationally expensive. Instead, Titanium-CPA uses a 'binomial difference' distribution BinDiff as also used in New Hope [ADPS16] and Kyber [BDK+17]. This distribution is efficiently sampleable, and approximates a Gaussian distribution. Importantly, our optimisation of the security reduction of [RSSS17] from PLWE^f for f in our ring polynomial family $\mathcal{F}$ to MP-LWE preserves the BinDiff distribution exactly (in shape and variance), so we are still able to provably lower bound the security of Titanium-CPA based on the assumed security of PLWE^f with the BinDiff distribution. As the distribution variance we use 2, which also matches previous choices [BDK+17].

- *Explicit constant-time algorithms:* Titanium-CPA specifies explicit algorithms for sampling from each of the distributions needed in key generation, encryption, and decryption. To resist timing side-channel attacks, the algorithms are (with one exception treated below) designed and implemented to run in constant-time, except with negligible probability that we explicitly upper bound in our correctness analysis in Chapter 7. Similarly, other operations in our encryption, decryption and key generation algorithms are also implemented with constant-time algorithms. We remark that one exception to the constant run-time implementation is our rejection-based sampling algorithm for uniformly random integers in $\mathbb{Z}_q$. The run-time of the latter algorithm

depends on the pattern of rejected/accepted integers generated by our SHA-3 based pseudorandom generator PRG; however, assuming the pseudorandomness of PRG, the accepted integers output by our rejection sampling algorithm are (indistinguishable from) uniformly random and independent integers in $\mathbb{Z}_q$, even conditioned on the rejection pattern leaked by the running-time of this sampling algorithm. Therefore, the non-constant run-time of this algorithm does not pose a timing side-channel risk to our scheme implementations.

- NTT-*based algorithm for middle-product:* We implement the middle-product computations in Titanium-CPA using a fast NTT-based algorithm optimised for our Titanium-CPA parameters, that improves on the efficiency of the naive approach of reducing middle product to NTT-based multiplication. This fast middle product algorithm, an optimised variant of the middle-product algorithm in [HQZ04] (see Chapter 3) to our setting, can still be viewed MP-LWE analogue of the standard NTT algorithm for multiplication in rings of the form $\mathbb{Z}_q[x]/(f(x))$. That is, to evaluate $c = a \odot b$, the algorithm essentially computes the forward NTT transforms $\widehat{\mathbf{a}}$ and $\widehat{\mathbf{b}}$ of a and b in appropriate dimensions, performs a coordinate-wise multiplication of $\widehat{\mathbf{a}}$ and $\widehat{\mathbf{b}}$ in the NTT domain, and performs an inverse NTT transform (and truncation) to recover c. As in PLWE-based lattice cryptosystems such as [ADPS16], the NTT computations make up the dominant computation cost in our algorithms. Accordingly, similarly to [ADPS16], we use presampling and precomputation optimisation techniques (see below) to either reduce the number of NTT computations, or trade-off a better encryption/decryption speed for a longer key generation time in our optimised algorithms for Titanium-CPA or Titanium-CCA.

- *Presampling s directly in the* NTT *domain*: Titanium-CPA uses the same distribution of the secret polynomial s as in [RSSS17], i.e. with coefficients uniformly random in $\mathbb{Z}_q$. However, since the secret key s is only used in its transformed NTT domain representation $\widehat{\mathbf{s}}$ in our NTT-based optimised algorithms for Titanium in Sec. 3.3, we save NTT computations for s by presampling s in key generation directly in its NTT domain representation $\widehat{\mathbf{s}}$. Thanks to the uniformity of s and the injectivity of NTT, the distribution of $\widehat{\mathbf{s}}$ is also uniform and easy to sample directly.

- NTT *precomputation for b_i and c_1*: To further speed-up Titanium-CPA encryption at the cost of a longer key generation time, we precompute during key generation the NTT of the public key polynomials b_i needed for the fast middle-product computation of c_2 in the Titanium-CPA encryption algorithm, and store the NTT domain representations $\widehat{\mathbf{b}}_i$ in the public key. Also, to speed up encryption of Titanium-CPA at the cost of a longer decryption, we send in the ciphertext c_1 in its NTT domain representation $\widehat{\mathbf{c}}_1$, moving the inverse NTT computation for recovering c_1 to the Titanium-CPA decryption. For our Titanium-CCA scheme built from Titanium-CPA using the Fujisaki-Okamoto transformation, this optimisation actually does not increase the Titanium-CCA decryption algorithm running time, since the latter algorithm performs both Titanium-CPA decryption and encryption (due to the Fujisaki-Okamoto re-encryption ciphertext validity check), but we still decrease the Titanium-CCA encryption run-time, as the latter only performs a Titanium-CPA encryption (but not a Titanium-CPA decryption).

- *Ciphertext length compression*: To reduce the length of our scheme ciphertext, we also apply a ciphertext compression optimisation technique (used also in previous lattice-based schemes) by chopping off cmp least-significant bits of the coefficients of c_2. This reduces ciphertext length at the cost of a larger decryption error probability. However, as the compression error term is added to the already existing decryption error term, a certain amount of compression can be achieved almost 'for free', i.e. with little effect on the overall decryption error term and hence decryption error probability. Note that we always choose cmp in a manner that while we still meet the probability of error goals, the number of remaining bits in c_2 be a multiple of 8 (for packing/unpacking purposes to one or two bytes).

- *Pseudorandom Generation of Randomness:* To save consumption of truly random bits that tend to be difficult to generate, Titanium-CPA generates all its randomness pseudorandomly from

256-bit seeds, using a fast pseudorandom bit generator based on SHA-3 in its KMAC256 PRF mode of operation in CTR mode [NISb].

2.2.2 Titanium-CPA parameters and building blocks

The scheme is an adaptation of Regev's cryptosystem from [Reg09] but adapted to give a security reduction from the MP-LWE problem introduced in [RSSS17].

Parameters

Our scheme relies on the following parameters and probability distributions:

- **Main Parameters**:
  - n - dim. of public key polynomials a_i,
  - k - deg. of enc. randomness polynomials r_i,
  - d - dim. of message polynomial $\mathsf{enc}(\mu)$,
  - t - no. of public key polynomials a_i,
  - q - ciphertext modulus,
  - p - plaintext modulus,
  - cmp - number of chopped ciphertext LS bits.
- **Error Distribution (χ_e) Sampling Parameters**:
  - η - no. of trials parameter of BinDiff error distribution χ_e.
- **Randomness Distribution (χ_r) Sampling Parameters**:
  - b_1 - log (base 2) of first χ_r interval half size $B_1/2$,
  - b_2 - log (base 2) second χ_r interval half size $B_2/2$,
  - N_{dec1} - no. of coefficients in χ_r in B_1 interval,
  - N_{dec} - total no. of coefficients.
- **Unif($\mathbb{Z}_q$) Distribution Sampling Parameters**:
  - bytpcs - secret key gen. no. of bytes in PRG call,
  - bytpca - public key gen. no. of bytes in PRG call,
  - bytpm - $U(\mathbb{Z}_q)$ rejection sampler no. of bytes per coordinate,
  - ZqRej - $U(\mathbb{Z}_q)$ rejection sampler multiple of q.
- NTT **over $\mathbb{Z}_q$ Parameters**:
  - d_1, d_2, d_3 - NTT dimensions,
  - ω_1, ω_2, ω_3 - NTT roots of unity.

Symmetric-Key building block

We use an extendable-output function (XOF) and its instantiations as our main symmetric-key building block. It is implemented using the KMAC256 Pseudorandom Function (PRF) specified in [NISb], in CTR mode and we model it as a random oracle here.

2.2.3 Titanium-CPA algorithms

Main algorithms

Algorithm 1 : Titanium-CPA.KeyGen

Input: $\mathsf{seedkg} = \mathsf{Randbytes}(32) \in \mathsf{byte}^{32}$.
Output: $\mathsf{pk} \in \mathsf{byte}^{\mathsf{lenpk}}$ and $\mathsf{sk} \in \mathsf{byte}^{\mathsf{lensk}}$.

function KeyGen($\mathsf{seedkg} = \mathsf{Randbytes}(32)$)
  Let $\mathsf{prgst} = \mathsf{PRG.Init}(\mathsf{seedkg})$.
  Let $(\mathsf{prgst}, \mathsf{seedsk}) = \mathsf{PRG.Out}(\mathsf{prgst}, 32) \in \mathsf{StSp}_{\mathsf{prg}} \times \mathsf{byte}^{32}$.
  Let $(\mathsf{prgst}, \mathsf{seedpk}) = \mathsf{PRG.Out}(\mathsf{prgst}, 32) \in \mathsf{StSp}_{\mathsf{prg}} \times \mathsf{byte}^{32}$.
  Let $(\mathsf{prgst}, \widehat{\mathbf{s}}) = \mathsf{Samps}(\mathsf{seedsk}) \in \mathsf{StSp}_{\mathsf{prg}} \times \mathbb{Z}_q^{\mathsf{d}_3}$.
  Let $(\bar{a}_1, \ldots, \bar{a}_t) = \mathsf{Sampa}(\mathsf{seedpk}) \in (\mathbb{Z}_q^{<n}[x])^t$.
  Let $(e_1, \ldots, e_t) = \mathsf{Sampe}(\mathsf{prgst}) \in (\mathbb{Z}_q^{<d+k}[x])^t$.
  Let $s = \mathsf{Trunc}(n+k+d-1, \mathsf{VecToPol}(\mathsf{d}_3^{-1} \cdot \mathsf{NTT}_{\mathsf{d}_3}(\omega_3, \mathsf{InvPerm}_{\mathsf{d}_3}(\widehat{\mathbf{s}})))) \in \mathbb{Z}_q^{<n+d+k-1}[x]$.
  for $i \leq t$ **do**
    Let $a_i = \mathsf{Rev}(\bar{a}_i)$.
    Let $b_i = a_i \odot_{d+k} s + e_i \in \mathbb{Z}_q^{<d+k}[x]$.
    Let $\widehat{\mathbf{b}}_i = \mathsf{Perm}_{\mathsf{d}_1}(\mathsf{d}_1 \cdot \mathsf{NTT}_{\mathsf{d}_1}^{-1}(\omega_1, \mathsf{Zpad}(\mathsf{d}_1, \mathsf{PolToVec}(b_i)))) \in \mathbb{Z}_q^{\mathsf{d}_1}$.
  end for
  Let $\mathsf{pk} = \mathsf{Encodepk}(\mathsf{seedpk} \in \mathsf{byte}^{32}, (\widehat{\mathbf{b}}_1, \ldots, \widehat{\mathbf{b}}_t)) \in \mathsf{byte}^{\mathsf{lenpk}}$.
  Let $\mathsf{sk} = \mathsf{seedsk} \in \mathsf{byte}^{\mathsf{lensk}}$.
end function

Algorithm 2 : Titanium-CPA.Encrypt

Input: $\mathsf{pk} \in \mathsf{byte}^{\mathsf{lenpk}}$, $\mathsf{m} \in \mathsf{byte}^{32}$, and $\mathsf{seedr} = \mathsf{Randbytes}(32) \in \mathsf{byte}^{32}$.
Output: $\mathsf{ct} \in \mathsf{byte}^{\mathsf{lenct}}$.

function Encrypt($\mathsf{pk}, \mathsf{m}, \mathsf{seedr} = \mathsf{Randbytes}(32)$)
  Let $(\mathsf{seedpk}, (\widehat{\mathbf{b}}_1, \ldots, \widehat{\mathbf{b}}_t)) = \mathsf{Decodepk}(\mathsf{pk}) \in \mathsf{byte}^{32} \times (\mathbb{Z}_q^{\mathsf{d}_1})^t$.
  Let $m = \mathsf{Decodem}(\mathsf{m}) \in \mathbb{Z}_q^{<d}[x]$.
  Let $(\bar{a}_1, \ldots, \bar{a}_t) = \mathsf{Sampa}(\mathsf{seedpk}) \in (\mathbb{Z}_q^{<n}[x])^t$.
  Let $(r_1, \ldots, r_t) = \mathsf{Sampr}(\mathsf{seedr}) \in (\mathbb{Z}_q^{<k+1}[x])^t$.
  Let $c_1' = \sum_{i=1}^t r_i \cdot \bar{a}_i \in \mathbb{Z}_q^{<n+k}[x]$.
  Let $\widehat{\mathbf{c}}_1 = \mathsf{Perm}_{\mathsf{d}_2}(\mathsf{NTT}_{\mathsf{d}_2}(\omega_2, \mathsf{PolToVec}(\mathsf{Zpad}(\mathsf{d}_2, c_1')))) \in \mathbb{Z}_q^{\mathsf{d}_2}$.
  $b_i = \mathsf{Trunc}(d+k, \mathsf{PolToVec}(\mathsf{d}_1^{-1} \cdot \mathsf{NTT}_{\mathsf{d}_1}(\omega_1, \mathsf{InvPerm}_{\mathsf{d}_1}(\widehat{\mathbf{b}}_i)))) \in \mathbb{Z}_q^{<d+k}[x]$.
  Let $c_2' = \sum_{i=1}^t \mathsf{Rev}(r_i) \odot_d b_i + \lfloor q/p \rfloor \cdot m \in \mathbb{Z}_q^{<d}[x]$.
  Let $c_2 = \mathsf{Chop}(\mathsf{cmp}, c_2') \in \mathbb{Z}_q^{<d}[x]$.
  Let $\mathsf{ct} = \mathsf{Encodect}(\widehat{\mathbf{c}}_1, c_2) \in \mathsf{byte}^{\mathsf{lenct}}$.
end function

Algorithm 3 : Titanium-CPA.Decrypt

Input: $\mathsf{sk} \in \mathsf{byte}^{\mathsf{lensk}}$ and $\mathsf{ct} \in \mathsf{byte}^{\mathsf{lenct}}$.
Output: $\mathsf{m} \in \mathsf{byte}^{32}$.

function Decrypt(sk, ct)
  Let $\mathsf{seedsk} = \mathsf{sk} \in \mathsf{byte}^{\mathsf{lensk}}$.
  Let $(\widehat{\mathbf{s}}, \mathsf{prgst}) = \mathsf{Samps}(\mathsf{seedsk}) \in \mathbb{Z}_q^{\mathsf{d}_3} \times \mathsf{StSp}_{\mathsf{prg}}$.
  Let $s = \mathsf{Trunc}(n+k+d-1, \mathsf{VecToPol}(\mathsf{d}_3^{-1} \cdot \mathsf{NTT}_{\mathsf{d}_3}(\omega_3, \mathsf{InvPerm}_{\mathsf{d}_3}(\widehat{\mathbf{s}})))) \in \mathbb{Z}_q^{<n+d+k-1}[x]$.
  Let $(\widehat{\mathbf{c}}_1, c_2) = \mathsf{Decodect}(\mathsf{ct}) \in \mathbb{Z}_q^{\mathsf{d}_2} \times \mathbb{Z}_q^{<d}[x]$.
  Let $c_1' = \mathsf{Trunc}(n+k, \mathsf{VecToPol}(\mathsf{NTT}_{\mathsf{d}_2}^{-1}(\omega_2, \mathsf{InvPerm}_{\mathsf{d}_2}(\widehat{\mathbf{c}}_1)))) \in \mathbb{Z}_q^{<n+k}[x]$.
  Let $c' = c_2 - \mathsf{Rev}(c_1') \odot_d s \in \mathbb{Z}_q^{<d}[x]$.
  Let $m' = \mathsf{Round}(\lfloor q/p \rfloor, c') \in \mathbb{Z}_p^{<d}[x]$.
  Let $\mathsf{m} = \mathsf{Encodem}(m') \in \mathsf{byte}^{32}$.
end function

Pseduorandom bit Generator (PRG)

This algorithm expands a 32-byte uniformly random seed into arbitrarily long pseudorandom bit sequences. It is implemented using an XOF. As mentioned earlier, we will use KMAC256 as our XOF and we model it as a random oracle.

PRG Initializer ($\mathsf{PRG.Init}(\mathsf{seed})$): This function initializes the PRG seed and returns a PRG state $\mathsf{prgst} \in \mathsf{StSp}_{\mathsf{prg}}$, where $\mathsf{StSp}_{\mathsf{prg}} = \mathsf{byte}^{32} \times \mathsf{byte}^4$ is the state space.

Algorithm 4 : PRG.Init

Input: $\mathsf{seed} \in \mathsf{byte}^{32}$.
Output: $\mathsf{prgst} \in \mathsf{byte}^{32} \times \mathsf{byte}^4$.

function PRG.Init(seed)
  Let $c = 0$, represented as a 32-bit unsigned integer in little endian form.
  Let $\mathsf{prgst} = (\mathsf{seed}, c)$.
end function

PRG Output ($\mathsf{PRG.Out}(\mathsf{prgst}, \ell)$): Calls to $\mathsf{PRG.Out}(\cdot, \cdot)$ returns ℓ pseudorandom bytes in byte^ℓ and an updated state prgst.

Algorithm 5 : PRG.Out

Input: $\mathsf{prgst} \in \mathsf{StSp}_{\mathsf{prg}}$ and $\ell \in \mathbb{N}$.
Output: $\mathsf{prgst} \in \mathsf{StSp}_{\mathsf{prg}}$ and $\mathsf{out} \in \mathsf{byte}^\ell$.

function PRG.Out(prgst, ℓ)
  Let $K \in \mathsf{byte}^{32}$ be the first 32 bytes of prgst.
  Let $c \in \mathbb{Z}_{2^{32}}$ be the last 4 bytes of prgst denoted as a 32-bit unsigned integer in little endian form.
  Let $L = 8 \cdot \ell \in \mathbb{Z}$.
  Let S denote the empty string.
  Let $\mathsf{out} = \mathsf{XOF}(K, c, L, S) \in \mathsf{byte}^\ell$.
  Let $c = c + 1$.
  Let $\mathsf{prgst} = (K, c) \in \mathsf{StSp}_{\mathsf{prg}}$.
end function

Probability distribution sampling functions

These functions are used to sample (using randomness derived with the pseudorandom generator PRG from an input seed) from relevant probability distributions.
Secret Key Sampling Algorithm ($\mathsf{Samps}(\mathsf{seedsk} \in \mathsf{byte}^{32})$): The secret key sampling algorithm samples a vector $\widehat{\mathbf{s}} \in \mathbb{Z}_q^{\mathsf{d}_3}$ with coordinates uniformly (pseudo) random in $\mathbb{Z}_q$.

Algorithm 6 : Samps

Input: $\mathsf{seedsk} \in \mathsf{byte}^{32}$.
Output: $(\mathsf{prgst}', \widehat{\mathbf{s}}) \in \mathsf{StSp}_{\mathsf{prg}} \times \mathbb{Z}_q^{n+d+k-1}$.
1: **function** Samps(seedsk)
2: Let $\mathsf{prgst} = \mathsf{PRG.Init}(\mathsf{seedsk}) \in \mathsf{StSp}_{\mathsf{prg}}$.
3: Let $(\mathsf{prgst}', \widehat{\mathbf{s}}) = \mathsf{SampUnifZq}(\mathsf{prgst}, \mathsf{d}_3, \mathsf{bytpcs})$.
4: **end function**

Public Key Sampling Algorithm ($\mathsf{Sampa}(\mathsf{seedpk} \in \mathsf{byte}^{32})$): The public key sampling algorithm samples the polynomials $(a_1, \ldots, a_t) \in (\mathbb{Z}_q^n)^t$ with coordinates uniformly random in $\mathbb{Z}_q$.

Algorithm 7 : Sampa

Input: $\mathsf{seedpk} \in \mathsf{byte}^{32}$.
Output: $(a_1, \ldots, a_t) \in (\mathbb{Z}_q^{<n}[x])^t$.
1: **function** Sampa(seedpk)
2: Let $\mathsf{prgst} = \mathsf{PRG.Init}(\mathsf{seedpk}) \in \mathsf{StSp}_{\mathsf{prg}}$.
3: **for** $i \leq t$ **do**
4: Let $a_i = \mathsf{VecToPol}(\mathbf{a}_i) \in \mathbb{Z}_q^{<n}[x]$.
5: Let $(\mathsf{prgst}', \mathbf{a}_i) = \mathsf{SampUnifZq}(n, \mathsf{bytpca}) \in \mathsf{StSp}_{\mathsf{prg}} \times \mathbb{Z}_q^n$.
6: **end for**
7: **end function**

Uniform Distribution on $\mathbb{Z}_q$ Sampling Algorithm ($\mathsf{SampUnifZq}(\mathsf{prgst} \in \mathsf{StSp}_{\mathsf{prg}}, \ell \in \mathbb{Z}, \mathsf{bytpc} \in \mathbb{Z})$): Uses rejection sampling to sample and return a vector of ℓ (pseudo)-uniformly random elements in $\mathbb{Z}_q$. Uses PRG with state prgst for pseudorandomness generation. Generates bytpc bytes at initial call to PRG.

Algorithm 8 : SampUnifZq

Input: $\mathsf{prgst} \in \mathsf{StSp}_{\mathsf{prg}}$, $\ell \in \mathbb{Z}$, and $\mathsf{bytpc} \in \mathbb{Z}$.
Output: $(\mathsf{prgst}, \mathbf{z}) \in \mathsf{StSp}_{\mathsf{prg}} \times \mathbb{Z}_q^{\ell}$.

function SampUnifZq($\mathsf{prgst}, \ell, \mathsf{bytpc}$))
  Let $(\mathsf{prgst}', \mathbf{r}) = \mathsf{PRG.Out}(\mathsf{prgst}, \mathsf{bytpc})$.
  Let $i = 0$ and $j = 0$.
  while $(j < \ell)$ **do**
    if $i = \mathsf{bytpc}/\mathsf{bytpm}$ **then**
      Let $(\mathsf{prgst}', \mathbf{r}) = \mathsf{PRG.Out}(\mathsf{prgst}, \mathsf{bytpc})$.
      Let $\mathsf{prgst} = \mathsf{prgst}'$.
      Let $i = 0$.
    end if
    Let $x \in \mathbb{Z}$ denote the i-th consecutive block of bytpm bytes from $\mathbf{r}$, interpreted as an unsigned integer in little-endian form.
    if $(x < \mathsf{ZqRej})$ **then**
      Let $z_j = x \bmod q$ and $j = j + 1$.
    end if
    Let $i = i + 1$.
  end while
  Let $\mathbf{z} = (z_0, \ldots, z_{\ell-1}) \in \mathbb{Z}_q^{\ell}$.
end function

Key Generation Error distribution χ_e Sampling Algorithm ($\mathsf{Sampe}(\mathsf{prgst})$): The error sampling algorithm samples the polynomials $(e_1, \ldots, e_t) \in (\mathbb{Z}_q^{<d+k}[x])^t$ with coefficients independently sampled (pseudorandomly) from the binomial difference distribution with parameter η:

$$\chi_e = (\mathsf{BinDiff}(\eta))^{t\cdot(d+k)}. \tag{2.6}$$

Algorithm 9 : Sampe

Input: $\mathsf{prgst} \in \mathsf{StSp}_{\mathsf{prg}}$.
Output: $(e_1, \ldots, e_t) \in (\mathbb{Z}_q^{<d+k}[x])^t$.

function Sampe(prgst)
  Let $\mathsf{Zebytes}$ be the minimal integer satisfies $8 \cdot \mathsf{Zebytes} \geq 2\eta$.
  Let $(\mathsf{prgst}', \mathbf{r}) = \mathsf{PRG.Out}(\mathsf{prgst}, t \cdot (d + k + 1) \cdot \mathsf{Zebytes})$.
  for $1 \leq \tau \leq t$ **do**
    Let $\mathbf{rr}$ be the τ-th block of $(d + k + 1) \cdot \mathsf{Zebytes}$ bytes in $\mathbf{r}$.
    Let $e_\tau = 0$.
    for $0 \leq i \leq d + k - 1$ **do**
      Let (x_1, x_2) be the i-th consecutive block of $\mathsf{Zebytes}$ from $\mathbf{rr}$, where x_1, x_2 be the high and low $8 \cdot \mathsf{Zebytes}/2$ bits, respectively.
      Let s_1 denote the number of bit 1s in the low η bits in x_1.
      Let s_2 denote the number of bit 1s in the low η bits in x_2.
      Let $e_\tau = e_\tau + (s_2 - s_1) \cdot x^i$.
    end for
  end for
end function

Encryption Randomness distribution χ_r Sampling Algorithm ($\mathsf{Sampr}(\mathsf{seedr} \in \mathsf{byte}^{32})$): The randomness sampling algorithm samples the polynomials $(r_1, \ldots, r_t) \in (\mathbb{Z}_q^{k+1})^t$ with first N_{dec1} coefficients (in

the concatenated vector of $N_{\mathsf{dec}} = t \cdot (k+1)$ polynomial coefficients) independently sampled (pseudorandomly) from the zero-excluded interval uniform distribution $\mathsf{ZeInt U}(B_1)$ with even parameter $B_1 = 2^{b_1+1}$, and the remaining $N_{\mathsf{dec}} - N_{\mathsf{dec1}}$ coefficients of $(r_1, \ldots, r_t)$ independently sampled (pseudorandomly) from the zero-excluded interval uniform distribution $\mathsf{ZeInt U}(B_2)$ with even parameter $B_2 = 2^{b_2+1}$:

$$\chi_r = \mathsf{ZeInt U}(B_1)^{N_{\mathsf{dec1}}} \times \mathsf{ZeInt U}(B_2)^{N_{\mathsf{dec}} - N_{\mathsf{dec1}}} \tag{2.7}$$

Algorithm 10 : Sampr

Input: $\mathsf{seedr} \in \mathsf{byte}^{32}$.
Output: $(r_1, \ldots, r_t) \in (\mathbb{Z}_q^{<k+1}[x])^t$.

function Sampr(seedr)
  Let Zbbytes be the minimal integer satisfies $8 \cdot \mathsf{Zbbytes} \geq \max(b_1, b_2)$.
  Let $\mathsf{Zbt} = \lfloor N_{\mathsf{dec1}} / (k+1) \rfloor$ and $\mathsf{ZbRem} = N_{\mathsf{dec1}} \bmod (k+1)$.
  Let $\mathsf{prgst} = \mathsf{PRG.Init}(\mathsf{seedr}) \in \mathsf{StSp}_{\mathsf{prg}}$.
  Let $(\mathsf{prgst}', \mathbf{r}) = \mathsf{PRG.Out}(\mathsf{prgst}, N_{\mathsf{dec}} \cdot \mathsf{Zbbytes} + N_{\mathsf{dec}}/8)$.
  for $1 \leq \tau \leq t$ **do**
    Let **rr** be the τ-th block of $(k+1) \cdot \mathsf{Zbbytes}$ bytes in **r** (counted from 1).
    Let x be the i-th consecutive block of Zbbytes from **rr**, interpreted as an integer.[1]
    if $\tau \leq \mathsf{Zbt}$ **then**
      for $i < k+1$ **do**
        Let $x' = \left(x \bmod 2^{b_1}\right) + 1$ and $r_{\tau,i} = x'$.
      end for
    else if $\tau = \mathsf{Zbt} + 1$ **then**
      for $0 \leq i \leq k$ **do**
        if $i \leq \mathsf{ZbRem}$ **then**
          Let $x' = \left(x \bmod 2^{b_1}\right) + 1$.
          Let $r_{\mathsf{Zbt}+1,i} = x'$.
        else $\mathsf{ZbRem} \leq i \leq k$
          Let $x' = \left(x \bmod 2^{b_2}\right) + 1$ and $r_{\mathsf{Zbt}+1,i} = x'$.
        end if
      end for
    else if $\mathsf{Zbt} + 2 \leq \tau \leq t$ **then**
      for $i < k+1$ **do**
        Let $x' = \left(x \bmod 2^{b_2}\right) + 1$ and $r_{\tau,i} = x'$.
      end for
    end if
    Let $\mathbf{r}'$ be the randomness in **r** starting from $\mathsf{Zbbytes} \cdot N_{\mathsf{dec}}$ (counted from 0).
    Let s_i be the i-th 8 sample blocks in r_τ (counted from 1).
    For each s_i, let x be the i-th byte in **rr** (counted from 1) and let x_j be the j-th bit in x counted from the least significant side.
    if $x_j = 1$ **then**,
      the j-th sample in s_i becomes negative.
    end if
  end for
end function

Utility functions

Ciphertext Compression Algorithm ($\mathsf{Chop}(\mathsf{cmp}, c)$): The ciphertext compression algorithm Chop chops off (clears) the cmp Least Significant (LS) bits of each coefficient of its polynomial argument c.

Algorithm 11 : Chop

Input: $\mathsf{cmp} \in \mathbb{Z}$ and a polynomial c with coefficients in $\mathbb{Z}_q$.
Output: $c' \in \mathbb{Z}_q^{<d}[x]$.
1: **function** $\mathsf{Chop}(\mathsf{cmp}, c)$
2: Let $c' = c - (c \bmod 2^{\mathsf{cmp}}) \in \mathbb{Z}_q^{<d}[x]$.
3: **end function**

Rounding Algorithm ($\mathsf{Round}(\lfloor q/p \rfloor, c)$): The rounding algorithm Round divides each coefficient of its argument polynomial by $\lfloor q/p \rfloor$ and rounds the result to the nearest integer mod q, rounding up if the division is an odd integer multiple of $1/2$ (we assume that q is odd and the argument polynomial coefficients are reduced mod q into the interval $\{-\lfloor q/2 \rfloor, \ldots, \lfloor q/2 \rfloor\}$).

Algorithm 12 : Round

Input: $z \in \mathbb{Z}$ and a polynomial c with coefficients in $\mathbb{Z}_q$.
Output: $m \in \mathbb{Z}_p^{<d}[x]$.
1: **function** $\mathsf{Round}((z, c))$
2: Let $m = \lfloor \frac{c}{z} \rceil \in \mathbb{Z}_q^{<d}[x]$.
3: **end function**

Truncation Algorithm ($\mathsf{Trunc}(d \in \mathbb{Z}, \mathbf{v})$): The truncation algorithm Trunc returns the first d coordinates of its argument vector $\mathbf{v}$ (assumed to be of dimension $\geq d$).

Algorithm 13 : Trunc

Input: $d \in \mathbb{Z}$ and a vector $\mathbf{v}$.
Output: $\mathbf{v}' \in \mathbb{Z}_q^d$.
1: **function** $\mathsf{Trunc}(d, \mathbf{v})$
2: Let $\mathbf{v}' = (v_0, v_1, \ldots, v_{d-1})$.
3: **end function**

Zero padding Algorithm ($\mathsf{Zpad}(d \in \mathbb{Z}, \mathbf{v})$): The zero padding algorithm Zpad pads its argument vector $\mathbf{v}$ (assumed to be of dimension $\leq d$) with zeros to dimension d.

Algorithm 14 : Zpad

Input: $d \in \mathbb{Z}$ and a vector $\mathbf{v}$.
Output: $\mathbf{v}' \in \mathbb{Z}_q^d$.
1: **function** $\mathsf{Zpad}(d, \mathbf{v})$
2: Let l denote the dimension of $\mathbf{v}$.
3: Let $\mathbf{v}' = (v_0, v_1, \ldots, v_{l-1}, 0, \ldots, 0) \in \mathbb{Z}_q^d$.
4: **end function**

Permutation ($\mathsf{Perm}_{\mathrm{dim}}(\mathbf{a})$): For $\mathrm{dim} = \delta_1 \cdot \delta_2$, where δ_2 is a power of 2, we define the following NTT permutation. This function arises from the efficient NTT algorithms used in the optimised implementation, as described in the Chapter 3.

Algorithm 15 : Perm

Input: $\mathbf{a} \in \mathbb{Z}_q^\ell$.
Output: $\mathbf{a}' \in \mathbb{Z}_q^\ell$.
1: **function** $\mathsf{Perm}_{\dim}(\mathbf{a})$
2: **for** $i < \ell$ **do**
3: Let $\mathbf{a}'_i = \mathbf{a}_{(i \bmod \delta_2)\cdot\delta_1 + \lfloor i/\delta_2 \rfloor}$.
4: **end for**
5: **end function**

Inverse Permutation ($\mathsf{InvPerm}_{\dim}(\mathbf{a})$): For $\dim = \delta_1 \cdot \delta_2$, where δ_2 is a power of 2, we define the following NTT permutation. This function arises from the efficient NTT algorithms used in the optimised implementation, as described in the Chapter 3.

Algorithm 16 : InvPerm

Input: $\mathbf{a} \in \mathbb{Z}_q^\ell$.
Output: $\mathbf{a}' \in \mathbb{Z}_q^\ell$.
1: **function** $\mathsf{InvPerm}_{\dim}(\mathbf{a})$
2: **for** $i < \ell$ **do**
3: Let $\mathbf{a}'_i = \mathbf{a}_{(i \bmod \delta_1)\cdot\delta_2 + \lfloor i/\delta_1 \rfloor}$
4: **end for**
5: **end function**

Encoding/Decoding Functions: These functions are used to pack a vector of elements in $\mathbb{Z}_q$ to a vector of bytes and unpack them back. The main functions are Encodem, Decodem, Encodepk, Decodepk, Encodect, and Decodect.

Algorithm 17 : PackVecℓ

Input: $\mathbf{b} \in \mathbb{Z}_q^\ell$ and ℓ_0.
Output: $\mathsf{v} \in \mathsf{byte}^N$.
1: **function** $\mathsf{PackVec}\ell(\mathbf{b}, \ell_0)$
2: Let bitst denote a bitstream initialized to empty.
3: **for** $0 \le i \le \ell - 1$ **do**
4: Let $c_i \in \mathsf{bit}^{\ell_0}$ be the binary representation of $b_i \in \mathbb{Z}_q$ in little endian form.
5: Append c_i to the bitstream bitst.
6: **end for**
7: Let L denote the bit length of bitst and let $L' = -L \bmod 8$.
8: Let $\mathsf{bitst}' = \mathsf{Zpad}(L + L', \mathsf{bitst})$.
9: Let $N = (L + L')/8$.
10: **for** $1 \le i \le N$ **do**
11: Let $x \in \mathsf{byte}$ denote the i-th block of 8 bits from bitst′.
12: Let $v_i = x$.
13: **end for**
14: Let $\mathsf{v} = (v_1, \ldots, v_N) \in \mathsf{byte}^N$.
15: **end function**

Algorithm 18 : PackVec

Input: $\mathbf{b} \in \mathbb{Z}_q^\ell$.
Output: $\mathbf{v} \in \mathsf{byte}^N$.
function PackVec($\mathbf{b}$)
    Let $\ell_0 = \lfloor \log_2(q) \rfloor + 1$.
    Let PackVec$\ell(\mathbf{b} \in \mathbb{Z}_q^\ell, \ell_0) \in \mathsf{byte}^N$.
end function

Algorithm 19 : PackVecc

Input: $\mathbf{b} \in \mathbb{Z}_q^\ell$.
Output: $\mathbf{v} \in \mathsf{byte}^N$.
function PackVecc($\mathbf{b}$)
    for $i < \ell$ **do**
        $b'_i = b_i / 2^{\mathsf{cmp}}$
    end for
    Let $\ell_0 = \lfloor \log_2(q) \rfloor + 1 - \mathsf{cmp}$.
    Let PackVec$\ell(\mathbf{b}') \in \mathbb{Z}_q^\ell, \ell_0)$.
end function

Algorithm 20 : UnPackVecℓ

Input: $\mathbf{v} \in \mathsf{byte}^N$ and ℓ_0.
Output: $\mathbf{b} \in \mathbb{Z}_q^\ell$.
function UnPackVec$\ell(\mathbf{v}, \ell_0)$
    Let bitst denote a bitstream initialized to empty.
    for $1 \leq i \leq N$ **do**
        Let $x \in \mathsf{bit}^8$ be the bit representation of $\mathbf{v}_i \in \mathsf{byte}$, the i-th byte of $\mathbf{v}$.
        Append x to bitst.
    end for
    Let $\ell = \lfloor \frac{8 \cdot N}{\ell_0} \rfloor$.
    for $0 \leq i \leq \ell - 1$ **do**
        Let $b_i \in \mathbb{Z}_q$ denote the i-th block of ℓ_0 bits in bitst interpreted as the binary representation of an integer in little endian form.
    end for
    Let $\mathbf{b} = (b_0, \ldots, b_{\ell-1}) \in \mathbb{Z}_q^\ell$.
end function

Algorithm 21 : UnPackVec

Input: $\mathbf{v} \in \mathsf{byte}^N$.
Output: $\mathbf{b} \in \mathbb{Z}_q^\ell$.
function UnPackVec(v)
    Let $\ell_0 = \lfloor \log_2(q) \rfloor + 1$.
    Let UnPackVec$\ell(\mathbf{v} \in \mathbb{Z}_q^\ell, \ell_0)$
end function

Algorithm 22 : UnPackVecc

Input: $\mathbf{v} \in \mathsf{byte}^N$.
Output: $\mathbf{b} \in \mathbb{Z}_q^\ell$.
1: **function** UnPackVecc(v)
2: Let $\ell_0 = \lfloor \log_2(q) \rfloor + 1 - \mathsf{cmp}$.
3: $\mathbf{b}' = \mathsf{UnPackVec}\ell(\mathbf{v} \in \mathbb{Z}_q^\ell, \ell_0)$
4: **for** $i < \ell$ **do**
5: Let $b_i = b'_i \cdot 2^{\mathsf{cmp}}$
6: **end for**
7: **end function**

Algorithm 23 : Encodect

Input: $(\widehat{\mathbf{c}}_1, c_2) \in \mathbb{Z}_q^{\mathsf{d}_2} \times \mathbb{Z}_q^{<d}[x]$.
Output: $\mathsf{ct} \in \mathsf{byte}^{\mathsf{lenct}}$.
1: **function** Encodect($\widehat{\mathbf{c}}_1, c_2$)
2: Let $\mathsf{ct} = (\mathsf{PackVec}(\widehat{\mathbf{c}}_1), \mathsf{PackVecc}(\mathsf{PolToVec}(c_2))) \in \mathsf{byte}^{\mathsf{lenct}}$.
3: **end function**

Algorithm 24 : Decodect

Input: $\mathsf{ct} \in \mathsf{byte}^{\mathsf{lenct}}$.
Output: $(\widehat{\mathbf{c}}_1, c_2) \in \mathbb{Z}_q^{\mathsf{d}_2} \times \mathbb{Z}_q^{<d}[x]$.
1: **function** Decodect(ct)
2: Let ct_1 and ct_2 be the byte streams of the first and second parts in ct, respectively.
3: Let $(\widehat{\mathbf{c}}_1, \mathbf{c}_2) = (\mathsf{UnPackVec}(\mathsf{ct}_1), \mathsf{UnPackVecc}(\mathsf{ct}_2)) \in \mathbb{Z}_q^{\mathsf{d}_2} \times \mathbb{Z}_q^d$.
4: Let $c_2 = \mathsf{VecToPol}(\mathbf{c}_2) \in \mathbb{Z}_q^{<d}[x]$.
5: **end function**

Algorithm 25 : Encodepk

Input: $\mathsf{seedpk} \in \mathsf{byte}^{32}$ and $(\widehat{\mathbf{b}}_1, \ldots, \widehat{\mathbf{b}}_t) \in (\mathbb{Z}_q^{d+k})^t$.
Output: $\mathsf{pk} \in \mathsf{byte}^{\mathsf{lenpk}}$.
1: **function** Encodepk(seedpk, $(\widehat{\mathbf{b}}_1, \ldots, \widehat{\mathbf{b}}_t)$)
2: Let $\mathbf{b} = (\widehat{\mathbf{b}}_1, \ldots, \widehat{\mathbf{b}}_t) \in (\mathbb{Z}_q^{d+k})^t$.
3: Let $\mathsf{b}_p = \mathsf{PackVec}(\mathbf{b}) \in \mathsf{byte}^{\mathsf{lenpk}-32}$.
4: Let $\mathsf{pk} = (\mathsf{seedpk}, \mathsf{b}_p)$.
5: **end function**

Algorithm 26 : Decodepk

Input: $\mathsf{pk} \in \mathsf{byte}^{\mathsf{lenpk}}$.
Output: $(\mathsf{seedpk}, (\mathbf{b}_1, \ldots, \mathbf{b}_t)) \in \mathsf{byte}^{32} \times (\mathbb{Z}_q^{d+k})^t$.
1: **function** Decodepk(pk)
2: Let seedpk denote the first 32 bytes of pk.
3: Let b_p denote the last $\mathsf{lenpk} - 32$ bytes of pk.
4: Let $(\mathbf{b}_1, \ldots, \mathbf{b}_t) = \mathsf{UnPackVec}(\mathsf{b}_p) \in (\mathbb{Z}_q^{d+k})^t$.
5: **end function**

Algorithm 27 : Encodem

Input: $\mathsf{m} \in \mathsf{byte}^{32}$.
Output: $m \in \mathbb{Z}_2^{<d}[x]$.
1: **function** Encodem(m)
2: Let $m = 0 \in \mathbb{Z}_q^{<d}[x]$.
3: **for** $1 \leq i \leq 32$ **do**
4: Let $(x_0, \ldots, x_7) \in \mathsf{bit}^8$ denote the 8 bits of of the i-th byte of m.
5: Let $m = m + \sum_{j=0}^{7} x_j \cdot x^{(i-1)\cdot 8+j}$.
6: **end for**
7: **end function**

Algorithm 28 : Decodem

Input: $m \in \mathbb{Z}^{<d}[x]$.
Output: $\mathsf{m} = (\mathsf{m}_1, \ldots, \mathsf{m}_{32}) \in \mathsf{byte}^{32}$.
1: **function** Decodem(m)
2: Let $m = 0 \in \mathbb{Z}_q^{<d}[x]$.
3: **for** $1 \leq i \leq 32$ **do**
4: Let $(x_0, \ldots, x_7) \in \mathbb{Z}^8$ denote the i-th block of 8 coefficients from m.
5: Let $(x'_0, \ldots, x'_7) = (x'_0 \bmod 2, \ldots, x'_7 \bmod 2) \in \mathsf{bit}^8$.
6: Let $\mathsf{m}_i = (x'_0, \ldots, x'_7) \in \mathsf{byte}$.
7: **end for**
8: **end function**

2.2.4 Titanium-CPA recommended parameter sets and claimed security

This Section contains the recommended parameter sets for our Titanium-CPA encryption scheme. The claimed security property for Titanium-CPA is semantic security with respect to chosen plaintext attack, known as IND-CPA security.

Recommended parameter sets

We specify total of 6 different parameters sets Toy64, Lite96, Std128, Med160, Hi192, Super256, intended to correspond to the brute force key search security level of a symmetric key cipher with key bit lengths $64, 96, 128, 160, 192, 256$, respectively. This means that any attack that breaks the security of our scheme must require computational resources comparable to or greater than those required for key search on a block cipher with a $64, 96, 128, 160, 192, 256$-bit key, respectively. In particular, the parameter sets Std128, Hi192, and Super256 satisfy the security categories 1, 3, and 5 specified by NIST in the call for proposals [NISa] corresponding to security of AES128, AES192, and AES256 against brute force key search attack.

The classical attack gate complexity level goal for the six parameter sets / symmetric-key search security levels, is denoted by λ_C with $\lambda_C \in \{79, 111, 145, 175, 207, 272\}$ corresponding to $\approx 2^{15}$ gates cost for each symmetric-key cipher evaluation, consistent with the specified AES128, AES192, and AES256 key search complexity levels specified in [NISa].

Similarly, the quantum attack gate complexity level goal for the six parameter sets / symmetric-key search security levels, is denoted by λ_Q with $\lambda_Q \in \{106, 140, 170, 202, 233, 298\} - \log_2(\mathsf{MD})$, intended to estimate the circuit gate complexity of quantum key search attacks under the assumption that the quantum attack circuit depth is restricted to MD (denoted by MAXDEPTH in [NISa]). These goals are consistent with AES128, AES192, and AES256 quantum key search complexity levels specified in [NISa].

In the following Tables, we give recommended core and error distribution and randomness, and NTT parameters of the following 6 parameter sets: Toy64, Lite96, Std128, Med160, Hi192, Super256.

In Table 2.1, we specify the core parameters of our Titanium-CPA encryption schemes corresponding to each parameter set. For further details on how we chose these concrete parameters, we refer the reader to Chapters 7-6. In Table 2.2, we present the relevant NTT and fast middle-product NTT

Table 2.1: Determined Titanium-CPA core parameters.

Parameter	Toy64	Lite96	Std128	Med160	Hi192	Super256
n	684	800	1024	1280	1536	2048
k	255	479	511	511	767	1023
d	256	256	256	256	256	256
t	10	8	9	9	7	7
q	240641	84481	86017	301057	737281	1198081
p	2	2	2	2	2	2
cmp	10	9	9	11	12	13

parameters for each parameter set. For definition of each parameter, please refer to Sections 2.1.2 and 3.2. In Table 2.3, we present the relevant error sampling parameters for each parameter set. For

Table 2.2: The NTT parameters for Titanium-CPA.

Parameter	Toy64	Lite96	Std128	Med160	Hi192	Super256
d_1	512	768	768	768	1024	1280
d_2	1024	1280	1536	1792	2304	3072
d_3	1280	1536	1792	2048	2560	3328
ω_1	56644	32904	71157	146766	1691	1061167
ω_2	238	68911	30077	168263	5454	186516
ω_3	35772	5170	35007	174	332395	1169874

definition of each parameter and how we calculated each, please refer to Chapters 7-6.

Table 2.3: The error distribution χ_e sampling parameter for Titanium-CPA.

Parameter	Toy64	Lite96	Std128	Med160	Hi192	Super256
η	4	4	4	4	4	4

In Table 2.4, we present the relevant randomness sampling parameters for each parameter set. For definition of each parameter and how we calculated each, please refer to Chapters 7-6. In Table 2.5, we present the relevant sampling parameters for uniform sampler over $\mathbb{Z}_q$ for each parameter set. For definition of each parameter and how we calculated each, please refer to Chapters 7-6.

2.3 Titanium-CCA: IND-CCA key encapsulation mechanism (KEM) scheme

Our KEM Titanium-CCA applies a variant of the Fujisaki-Okamoto (FO) transform [FO99] from [HHK17] to our IND-CPA encryption scheme Titanium-CPA to turn the latter into an IND-CCA KEM.

Table 2.4: The randomness χ_r sampling parameters for Titanium-CPA.

Parameter	Toy64	Lite96	Std128	Med160	Hi192	Super256
b_1	7	5	5	7	8	8
b_2	8	6	6	8	9	9
N_{dec}	2560	3840	4608	4608	5376	7168
N_{dec1}	1488	1496	2568	3816	3384	3848

Table 2.5: The sampling parameters for uniform sampler over $\mathbb{Z}_q$ for Titanium-CPA.

Parameter	Toy64	Lite96	Std128	Med160	Hi192	Super256
bytpcs	4044	4764	5469	6579	8571	10158
bytpca	2205	2523	3153	4191	5280	6300
bytpm	3	3	3	3	3	3
ZqRej	16604229	16727238	16773315	16558135	16220182	16773134

2.3.1 Titanium-CCA Parameters and building blocks

Symmetric-Key Building Blocks

We use XOF as our symmetric-key building block. The pseudorandom bit generator PRG is exactly the same as specified for Titanium-CPA. We also use hash functions for our Titanium-CCA. Cryptographic Hash functions G and H are modeled as a 'random oracle' in the security analysis, and are instantiated using the SHAKE256 mode of SHA-3 [NISc].

2.3.2 Titanium-CCA algorithms

Algorithm 29 : Titanium-CCA.KeyGen

Input: $\mathsf{seedkg} = \mathsf{Randbytes}(32) \in \mathsf{byte}^{32}$.
Output: $\mathsf{sk} \in \mathsf{byte}^{32}$ and $\mathsf{pk} \in \mathsf{byte}^{\mathsf{lenpk}}$.

function KeyGen(seedkg)
  Let $\mathsf{prgst} = \mathsf{PRG.Init}(\mathsf{seedkg})$.
  Let $((\mathsf{seedkg.cpa}, \mathsf{rdec}), \mathsf{prgst}) = \mathsf{PRG.Out}(\mathsf{prgst}, 64) \in \mathsf{byte}^{32} \times \mathsf{byte}^{32} \times \mathsf{StSp}_{\mathsf{prg}}$.
  Sample $(\mathsf{sk.cpa}, \mathsf{pk.cpa}) = \mathsf{Titanium\text{-}CPA.KeyGen}(\mathsf{seedkg.cpa})$.
  Let $\mathsf{sk} = (\mathsf{seedkg}, \mathsf{pk.cpa})$.
  Let $\mathsf{pk} = \mathsf{pk.cpa}$.
end function

Algorithm 30 : Titanium-CCA.Encrypt

Input: $\mathsf{pk} \in \mathsf{byte}^{\mathsf{lenpk}}$ and $\mathsf{m} = \mathsf{Randbytes}(32) \in \mathsf{byte}^{32}$.
Output: $\mathsf{ct} \in \mathsf{byte}^{\mathsf{lenct}}$ and $\mathsf{ss} \in \mathsf{byte}^{32}$.

function Encrypt(pk, m)
  Let $(\mathsf{seedenc.cpa}, \mathsf{dcca}) = \mathsf{G}(\mathsf{m}) \in \mathsf{byte}^{32} \times \mathsf{byte}^{32}$.
  Let $\mathsf{ct.cpa} = \mathsf{Titanium\text{-}CPA.Encrypt}(\mathsf{pk}, \mathsf{m}, \mathsf{seedenc.cpa})$.
  Let $\mathsf{ct} = (\mathsf{ct.cpa}, \mathsf{dcca}) \in \mathsf{byte}^{\mathsf{lenct}}$
  Let $\mathsf{ss} = \mathsf{H}(\mathsf{m}, \mathsf{ct}) \in \mathsf{byte}^{32}$.
end function

Algorithm 31 : Titanium-CCA.Decrypt

Input: $\mathsf{sk} \in \mathsf{byte}^{32}$ and $\mathsf{ct} \in \mathsf{byte}^{\mathsf{lenct}}$.
Output: $\mathsf{ss} \in \mathsf{byte}^{32}$.

function Decrypt(sk, ct)
  Parse $\mathsf{ct} = (\mathsf{ct.cpa}, \mathsf{dcca}) \in \mathsf{byte}^{\mathsf{lenct.cpa}} \times \mathsf{byte}^{32}$.
  Parse $\mathsf{sk} = (\mathsf{seedkg}, \mathsf{pk.cpa}) \in \mathsf{byte}^{32} \times \mathsf{byte}^{\mathsf{lenpk}}$.
  Let $\mathsf{prgst} = \mathsf{PRG.Init}(\mathsf{seedkg})$.
  Let $(\mathsf{prgst}, (\mathsf{rdec}, \mathsf{seedkg.cpa})) = \mathsf{PRG.Out}(\mathsf{prgst}, 64) \in \mathsf{StSp}_{\mathsf{prg}} \times \mathsf{byte}^{32} \times \mathsf{byte}^{32}$.
  Let $\mathsf{prgst} = \mathsf{PRG.Init}(\mathsf{seedkg.cpa})$.
  Let $(\mathsf{prgst}, \mathsf{sk.cpa}) = \mathsf{PRG.Out}(\mathsf{prgst}, 32) \in \mathsf{StSp}_{\mathsf{prg}} \times \mathsf{byte}^{32}$.
  Let $\mathsf{m}' = \mathsf{Titanium\text{-}CPA.Decrypt}(\mathsf{sk.cpa}, \mathsf{ct.cpa}) \in \mathsf{byte}^{32}$.
  Let $(\mathsf{seedenc.cpa}', \mathsf{dcca}') = \mathsf{G}(\mathsf{m}') \in \mathsf{byte}^{32} \times \mathsf{byte}^{32}$.
  Let $\mathsf{ct.cpa}' = \mathsf{Titanium\text{-}CPA.Encrypt}(\mathsf{pk.cpa}, \mathsf{m}', \mathsf{seedenc.cpa}')$.
  if $(\mathsf{ct.cpa}', \mathsf{dcca}') = (\mathsf{ct.cpa}, \mathsf{dcca})$ **then**
    Let $\mathsf{ss} = \mathsf{H}(\mathsf{m}', \mathsf{ct}) \in \mathsf{byte}^{32}$
  else
    Let $\mathsf{ss} = \mathsf{H}(\mathsf{rdec}, \mathsf{ct}) \in \mathsf{byte}^{32}$.
  end if
end function

Cryptographic hash functions

In the sequel, we specify our hash functions G and H:

Algorithm 32 : G

Input: $\mathsf{m} \in \mathsf{byte}^{32}$.
Output: $(\mathsf{out}_1, \mathsf{out}_2) \in \mathsf{byte}^{32} \times \mathsf{byte}^{32}$.

function G(m)
  Let $L = 512$.
  Let $(\mathsf{out}_1, \mathsf{out}_2) = \mathsf{SHAKE256}(\mathsf{m}, L) \in \mathsf{byte}^{32} \times \mathsf{byte}^{32}$.
end function

Algorithm 33 : H

Input: $\mathsf{m} \in \mathsf{byte}^{32}$ and $\mathsf{ct} \in \mathsf{byte}^{\mathsf{lenct}}$.
Output: $\mathsf{out}_2 \in \mathsf{byte}^{32}$.

function H(m, ct)
  Let $L = 512$.
  Let $\mathsf{out}_1 = \mathsf{SHAKE256}(\mathsf{ct}, L) \in \mathsf{byte}^{64}$.
  Let $L = 256$.
  Let $\mathsf{out}_2 = \mathsf{SHAKE256}((\mathsf{m}, \mathsf{out}_1), L) \in \mathsf{byte}^{32}$.
end function

2.3.3 Titanium-CCA recommended parameter sets and claimed security

This Section contains the recommended parameter sets for our Titanium-CCA Key Encapsulation Mechanism (KEM). The claimed security property for Titanium-CCA is semantic security with respect to adaptive chosen ciphertext attack, known as IND-CCA security.

Recommended parameter sets

We again specify total of 6 different parameters sets Toy64, Lite96, Std128, Med160, Hi192, Super256, intended to correspond to the brute force key search security level of a symmetric key cipher with key bit lengths $64, 96, 128, 160, 192, 256$, respectively. This means that any attack that breaks the security of our scheme must require computational resources comparable to or greater than those required for key search on a block cipher with a $64, 96, 128, 160, 192, 256$-bit key, respectively. In particular, the parameter sets Std128, Hi192, and Super256 satisfy the security categories 1, 3, and 5 specified by NIST in the call for proposals [NISa] corresponding to security of AES128, AES192, and AES256 against brute force key search attack.

The classical and quantum attack gate complexity level goals for the six parameter sets / symmetric-key search security levels are taken to be exactly the same as the ones for Titanium-CPA scheme. In the following Tables, we give recommended core and error distribution and randomness, and NTT parameters of the following 6 parameter sets: Toy64, Lite96, Std128, Med160, Hi192, Super256. In Table 2.6, we specify the core parameters of our Titanium-CCA schemes corresponding to each level. For further details on how we chose these concrete parameters, we refer the reader to Chapters 7-6. In Table 2.7, we present the relevant NTT and fast middle-product NTT parameters for each level.

Table 2.6: Determined Titanium-CCA core parameters.

Parameter	Toy64	Lite96	Std128	Med160	Hi192	Super256
n	684	800	1024	1280	1536	2048
k	255	479	511	511	767	1023
d	256	256	256	256	256	256
t	10	9	10	10	8	8
q	471041	115201	118273	430081	783361	1198081
p	2	2	2	2	2	2
cmp	11	9	9	11	12	13

For definition of each parameter, please refer to Sections 2.1.2 and 3.2. In Table 2.8, we present the

Table 2.7: The NTT parameters for Titanium-CCA.

Parameter	Toy64	Lite96	Std128	Med160	Hi192	Super256
d_1	512	768	768	768	1024	1280
d_2	1024	1280	1536	1792	2304	3072
d_3	1280	1536	1792	2048	2560	3328
ω_1	69169	13192	4136	15791	874	1061167
ω_2	263	4959	75894	235435	596706	186516
ω_3	327393	5516	9282	92	13502	1169874

relevant error sampling parameters for each level. For definition of each parameter and how we calculated each, please refer to Chapters 7-6. In Table 2.9, we present the relevant randomness sampling

Table 2.8: The error distribution χ_e sampling parameter for Titanium-CCA.

Parameter	Toy64	Lite96	Std128	Med160	Hi192	Super256
η	4	4	4	4	4	4

parameters for each level. For definition of each parameter and how we calculated each, please refer to Chapters 7-6. In Table 2.10, we present the relevant sampling parameters for uniform sampler

Table 2.9: The randomness χ_r sampling parameters for Titanium-CCA.

Parameter	Toy64	Lite96	Std128	Med160	Hi192	Super256
b_1	7	5	4	6	7	7
b_2	8	6	5	7	8	8
N_{dec}	2560	4320	5120	5120	6144	8192
N_{dec1}	328	4168	208	2248	4704	5904

over $\mathbb{Z}_q$ for each level. For definition of each parameter and how we calculated each, please refer to Chapters 7-6.

Table 2.10: The sampling parameters for uniform sampler over $\mathbb{Z}_q$ for Titanium-CCA.

Parameter	Toy64	Lite96	Std128	Med160	Hi192	Super256
bytpcs	4110	4788	5634	6255	8334	10158
bytpca	2250	2541	3276	3942	5115	6300
bytpm	3	3	3	3	3	3
ZqRej	16486435	16704145	16676493	16773159	16450581	16773134

Target and minimum claimed security and probability of error

The target quantum security λ_Q and classical security λ_C along with the requested probability of decryption error p_e for each level are specified in Table 2.11 and Tables 2.12-2.13, respectively. For different MAXDEPTH (MD), we have different goals and minimum achieved security levels. Note that the p_e goal for Titanium-CPA schemes are set to 2^{-30}. In Table 2.13, we only show the goal and achieved p_e of Titanium-CCA schemes with MAXDEPTH, MD = 40. For further information on how these parameters are calculated, please refer to Chapters 7 and 6.

Some of the above achieved (claimed) security levels in Table 2.11 are matched with security categories asked by NIST [NISa]. We summarize these matched levels in Table 2.14.

2.4 Design Rationale Summary

In this Section, we summarize our main design rationale aspects. Detailed reasoning for our choice of parameters is explained in other sections of this document (in particular Chapters 3, 5, and 6). Here, we mostly focus on some of the high level aspects not discussed elsewhere.

- **Choice of MP-LWE as the underlying hard problem.** As explained in the Introduction, our choice of MP-LWE as the basis for the security of Titanium is motivated by a balance between the goal of hedging against future advances in cryptanalysis exploiting algebraic properties of specific polynomial rings on the one hand, and the goal of achieving good performance on the other. The security of Titanium remains valid as long as at least *one* of exponentially many polynomial rings resists attacks against the PLWE problem in the ring. Yet Titanium still enjoys significant performance advantages compared to schemes based on unstructured lattices.[3]

- **Parameter Selection Approach.** Our approach to selecting parameter sets (detailed in Chapter 6), is based on provable *lower bound* estimates for the complexity of (classical) attacks on our schemes, based on the presumed hardness of the PLWE^f problem over a large family of polynomials $f \in \mathcal{F}_1$. We chose this approach in preference to deriving parameters based on

[3]The name Titanium comes from the well known hardness/strength properties of Titanium metal, making it among the hardest metals used for jewelry rings, while still being practical.

Table 2.11: The goal and claimed security params. for Titanium-CCA and Titanium-CPA.

Scheme	Param.	MD = 40		MD = 64		MD = 96	
		Goal	Min. Cl.	Goal	Min. Cl.	Goal	Min. Cl.
Titanium-CCA-Toy64	λ_Q	66	73	53	61	53	61
Titanium-CCA-Toy64	λ_C	79	82	79	82	79	82
Titanium-CPA-Toy64	λ_Q	66	83	53	83	53	83
Titanium-CPA-Toy64	λ_C	79	90	79	90	79	90
Titanium-CCA-Lite96	λ_Q	98	110	74	110	69	110
Titanium-CCA-Lite96	λ_C	111	126	111	120	111	120
Titanium-CPA-Lite96	λ_Q	98	115	74	115	69	115
Titanium-CPA-Lite96	λ_C	111	126	111	126	111	126
Titanium-CCA-Std128	λ_Q	130	134	106	126	85	105
Titanium-CCA-Std128	λ_C	143	146	143	146	143	146
Titanium-CPA-Std128	λ_Q	130	155	106	159	85	159
Titanium-CPA-Std128	λ_C	143	164	143	164	143	164
Titanium-CCA-Med160	λ_Q	162	176	138	164	106	132
Titanium-CCA-Med160	λ_C	175	192	175	192	175	192
Titanium-CPA-Med160	λ_Q	162	183	138	187	106	186
Titanium-CPA-Med160	λ_C	175	199	175	200	175	200
Titanium-CCA-Hi192	λ_Q	193	207	169	183	137	151
Titanium-CCA-Hi192	λ_C	207	230	207	230	207	230
Titanium-CPA-Hi192	λ_Q	193	214	169	217	137	187
Titanium-CPA-Hi192	λ_C	207	234	207	237	207	237
Titanium-CCA-Super256[2]	λ_Q	258	240	234	216	202	184
Titanium-CCA-Super256	λ_C	272	266	272	266	272	266
Titanium-CPA-Super256	λ_Q	258	243	234	219	202	187
Titanium-CPA-Super256	λ_C	272	269	272	269	272	269

best known attacks on MP-LWE, since the MP-LWE has only been recently introduced, and its concrete hardness is not yet well understood, as opposed to PLWE^f which has been known since at least 2009 [SSTX09].

- **Use of NTT in reference specification.** The reference specification of our scheme performs a pre-NTT computation in key generation and encryption to reduce the computation cost. Although this means that efficient implementations of our scheme need to implement the middle-product operation in our system using an NTT-based algorithm, we believe the efficiency improvements gained are a worthwhile reason to adopt this variant of the scheme.

- **Choice of Uniform distribution for secret key.** As opposed to most other LWE and PLWE-based schemes, we chose the distribution of our secret key coordinates as uniform over $\mathbb{Z}_q$, rather than having 'small' coefficients. Although it is possible to prove the hardness of the MP-LWE problem with 'small' secret coordinates (from the hardness of PLWE^f with 'small' secret coordinates, which is known [ACPS09] to be equivalent to PLWE^f with uniform secret), this choice would not be compatible with our optimisation of sampling the secret key directly in the NTT domain to save an NTT operation. Moreover, using a secret key with small coordinates would not save on secret key length, since we generate the secret key using a pseudorandom generator from a short seed. The main advantage of using a secret key with small coordinates in previous PLWE-based schemes is the ability to use the 'ElGamal' variant of Regev's encryption

Table 2.12: The target and achieved probability of error p_e of Titanium-CPA schemes.

Parameter	Goal	Toy64	Lite96	Std128	Med160	Hi192	Super256
$-\log_2(p_e)$	30	30	31	33	41	37	72

Table 2.13: The target and achieved probability of error p_e of Titanium-CCA schemes.

Parameter	Toy64		Lite96		Std128		Med160		Hi192		Super256	
	Goal	Ach.	Goal	Ach.	Goal	Ach.	Goal	Ach.	Goal	Ach.	Goal	Ach.
$-\log_2(p_e)$	79	85	111	158	143	161	175	199	206	218	271	354

scheme [LPR10], where the security of the encryption operation is analysed using a computational indistinguishability argument, rather than a statistical Leftover Hash Lemma argument as used in Regev's original scheme [Reg09] and (with appropriate modifications) in our scheme. The use of a computational argument has efficiency benefits, but it is an open problem to construct such a system based on MP-LWE. Overall, then, our Titanium schemes benefit from our choice of uniform distribution for the key that is easier to generate and has efficiency benefits due to the pre-NTT optimisation.

- **Choice of a Binomial Difference distribution for the errors.** In common with previous PLWE^f based schemes such as New Hope [ADPS16] and Kyber [BDK+17], our scheme uses a binomial difference error distribution, as it is significantly easier to sample efficiently and in constant time, in comparison with other choices such as a discrete Gaussian distribution. Moreover, the exact shape of the distribution does not seem to make a significant difference to the security of the scheme (and provable arguments to show this security equivalence under appropriate conditions can be given using the Renyi divergence technique [BLL+15]).

- **Choice of a Uniform distribution for encryption randomness.** Our choice of the shape of the interval-uniform distribution for the encryption randomizers was motivated by the maximal min-entropy of such a distribution for a given variance, minimizing the size of the modulus needed in our scheme for a given decryption error probability, while satisfying the leftover hash condition needed in our security proof. The use of two interval sizes being powers-of-2 was chosen for efficient implementation, while still allowing us a 'fine control' (via the choice of the fraction of coefficients $\rho = N_{\mathsf{dec1}}/N_{\mathsf{dec}}$ allocated to each interval size) to tweak the 'effective' entropy and 'average variance' of the randomness coefficients to minimize the size of our parameters.

- **No reconciliation or error correction techniques.** We considered the use of 'reconciliation' [Pei14] and error-correction techniques, but those do not appear to lead to significant savings in our scheme due to its statistical leftover hash security argument. Moreover, due to the security implications of decryption errors for our Titanium-CCA scheme, we decided to avoid error correction techniques that require heuristic analysis, to ensure that we can compute a provable upper bound on the decryption error probability p_e for the IND-CCA security proof for Titanium-CCA.

- **Choice of decryption error probability p_e for Titanium-CPA and Titanium-CCA.** For Titanium-CPA, since decryption error probability has no effect on IND-CPA security but has an efficiency impact, we decided to fix a moderate decryption error probability goal $p_e = 2^{-30}$. We think this level of decryption failure rate (about 1 in a billion decryption failures) should be acceptable in practical applications (such as key exchange protocols) via standard protocol restart/retransmission techniques. The rare occurrence of such failures would likely pose a negligible performance overhead, yet we achieve significant performance improvements in Titanium-CPA over Titanium-CCA due to the higher value of p_e in the former. In the case of

Table 2.14: Determined Titanium-CCA core parameters.

NIST category	Claimed levels
AES128	Std128, Med160
AES192	Hi192
AES256	Super256

Titanium-CCA, we set the decryption error probability goal to a cryptographically negligible value, to prevent decryption-failure chosen-ciphertext attacks against the IND-CCA security of Titanium-CCA, and we also provide proven upper bounds on p_e in our correctness analysis, that are also used as part of our Titanium-CCA IND-CCA security proofs.

2.5 Advantages and Limitations

We summarize several advantages of our Titanium-CPA and Titanium-CCA schemes:

- Our schemes' security (in the classical random oracle model) is *provably and tightly* based on the hardness of the *hardest* Polynomial-LWE problem (PLWE^f), a variant of the Learning with Errors (LWE) problem over a polynomial ring $\mathbb{Z}[x]/(f(x))$, among all f in a large family of ring polynomials. In particular, unlike previous schemes based on PLWE^f with specific choices for f (that may turn out to be weak), the security guarantees of our scheme remain valid even if the PLWE^f problem is found to be weak for some specific choices of f in the family (as long as some f is still hard). Our schemes therefore enjoy a hedge against future advances in cryptanalysis.

- Our scheme implementations are significantly more efficient than implementations of existing schemes based on unstructured matrices, such as [BCD+16].

- Our schemes (like others based on the PLWE and related problems) are amenable to fast implementation using the Number Theoretic Transform (NTT), a standard algorithm already implemented in many existing software libraries, such NTL [Sho]. Such libraries could be easily adapted 'off the shelf' to build fast implementations of our schemes.

- Our schemes should be suitable for 'plug-in' deployment in existing protocols using public key encryption, such as TLS, although we have not tested our schemes in such settings.

We summarize limitations of our Titanium-CPA and Titanium-CCA schemes:

- Our schemes' security analysis currently assumes the *classical* random oracle model (although non-tight security proofs in the quantum random oracle model are applicable for its Fujisaki-Okamoto transform [HHK17]).

Chapter 3

Implementation and Performance

This Chapter contains a summary of the time and memory performance of our Titanium-CPA and Titanium-CCA schemes, discusses our efficient optimised implementation techniques, and a summary of the Known Answer Test (KAT) files provided in files as part of this submission package.

3.1 Performance Summary

Our benchmark script will generate 1000 random seeds and random messages as the input, then the script will run the whole process of the Keygen, the Encryption/Encapsulation, and the Decryption/Decapsulation for these 1000 inputs, and record the number of CPU cycles consumed by these 3 steps, respectively. Then we take the median among these recorded number of CPU cycles as our benchmark result for the corresponding step.

Our benchmark platform has an Intel i7-7700K CPU running the Linux operating system. The CPU frequency is fixed at 4.2 GHz with the Hyperthreading and the Turbo Boost disabled during the benchmark. The program is compiled with GCC 7.2.0, with the compiler optimisation `-O3 -march=native` enabled, which will select the `O3` optimisation level and let the GCC itself choose the other optimisation options based on the architecture of the benchmark platform.

The Table 3.1 shows the benchmark results and the size of the key and the ciphertext in byte for each parameter set we specified for Titanium-CPA scheme. Let ℓ_0 denote $\lfloor \log_2 q \rfloor + 1$. The public-key and ciphertext sizes (in bytes) of the Titanium-CPA schemes can also be calculated using the following formulas:

$$|\mathsf{pk}_{\mathsf{Titanium\text{-}CPA}}| = 32 + \left\lceil \frac{\mathsf{d}_1 \cdot t \cdot \ell_0}{8} \right\rceil,$$

and

$$|\mathsf{ct}_{\mathsf{Titanium\text{-}CPA}}| = \left\lceil \frac{\mathsf{d}_2 \cdot \ell_0}{8} \right\rceil + \left\lceil \frac{d \cdot (\ell_0 - \mathsf{cmp})}{8} \right\rceil.$$

Since t, q (and consequently ℓ_0), and cmp are different in Titanium-CCA and Titanium-CPA for a given parameter set, we obtain different sizes in Tables 3.2 and 3.1.

The Table 3.2 shows the benchmark results and the size of the key and the ciphertext in byte for each parameter set we specified for Titanium-CCA scheme. The public-key and ciphertext sizes (in bytes) of the Titanium-CCA schemes can also be calculated using the following formulas:

$$|\mathsf{pk}_{\mathsf{Titanium\text{-}CCA}}| = 32 + \left\lceil \frac{\mathsf{d}_1 \cdot t \cdot \ell_0}{8} \right\rceil,$$

and

$$|\mathsf{ct}_{\mathsf{Titanium\text{-}CCA}}| = 32 + \left\lceil \frac{\mathsf{d}_2 \cdot \ell_0}{8} \right\rceil + \left\lceil \frac{d \cdot (\ell_0 - \mathsf{cmp})}{8} \right\rceil.$$

We further denote the secret part of the secret key, seedkg by sk^*.

Table 3.1: Benchmark results of the Titanium-CPA.

Security Level	Number of cycles	Size (byte)
Toy64	KeyGen: 1264647	pk: 11552
	Encrypt: 900120	sk: 32
	Decrypt: 152705	ct: 2560
Lite96	KeyGen: 1269479	pk: 13088
	Encrypt: 1073167	sk: 32
	Decrypt: 183832	ct: 2976
Std128	KeyGen: 1619550	pk: 14720
	Encrypt: 1262047	sk: 32
	Decrypt: 217612	ct: 3520
Med160	KeyGen: 1877257	pk: 16448
	Encrypt: 1646486	sk: 32
	Decrypt: 253458	ct: 4512
Hi192	KeyGen: 1894719	pk: 17952
	Encrypt: 1763250	sk: 32
	Decrypt: 323977	ct: 6016
Super256	KeyGen: 2486436	pk: 23552
	Encrypt: 2450834	sk: 32
	Decrypt: 439522	ct: 8320

We observe that the number of cycles and sizes of Super256 had a big jump compared to Hi192 in both Titanium-CPA and Titanium-CCA, in Tables 3.1 and 3.2, respectively, since the gap between all the other levels is only 32 (for example the quantum security goals for Lite96 and Std128 are 98 and 130, respectively, which are 32 apart), while the jump between the (quantum or classical) security levels of Hi192 and Super256 is 64.

3.2 Fast Middle Product Algorithm and Optimisations

For a polynomial a, to simplify the description of the algorithm, we use the following notations. For a vector $\mathbf{a}$ of dimension dim, we let

$$\begin{aligned}\mathsf{NTT}'_{\mathrm{dim}}\left(\omega', \mathbf{a}\right)_k &= \sum_{i=0}^{\mathrm{dim}-1} a_i \cdot (\omega')^{i\cdot k},\\ \mathsf{NTT}'^{-1}_{\mathrm{dim}}\left(\omega', \mathbf{a}\right)_k &= \mathrm{dim}^{-1} \cdot \sum_{i=0}^{\mathrm{dim}-1} a_i \cdot (\omega')^{i\cdot k},\end{aligned}$$

where the ω'_{dim} can be either the dim-th root of unity ω_{dim} or its inverse $\omega^{-1}_{\mathrm{dim}}$. The following Lemma represents how the standard NTT-based algorithm can be employed to compute a polynomial multiplication.

Lemma 3.2.1. *For polynomial $a \in \mathbb{Z}_q^{<d_1}[x]$, $b \in \mathbb{Z}_q^{<d_2}[x]$ and integer $\mathrm{dim} \geq d_1 + d_2 - 1$, the (zero-padded) coefficient vector of the polynomial product $a \cdot b \in \mathbb{Z}_q^{<d_1+d_2-1}[x]$ can be computed as:*

$$\mathsf{NTT}'^{-1}_{\mathrm{dim}}\left(\omega^{-1}_{\mathrm{dim}}, \mathsf{NTT}'_{\mathrm{dim}}\left(\omega_{\mathrm{dim}}, \mathsf{Zpad}\left(\mathrm{dim}, \mathsf{PolToVec}\left(a\right)\right)\right) \circ \mathsf{NTT}'_{\mathrm{dim}}\left(\omega_{\mathrm{dim}}, \mathsf{Zpad}\left(\mathrm{dim}, \mathsf{PolToVec}\left(b\right)\right)\right)\right),$$

where $\circ$ is the point-wise multiplication of two vectors.

Table 3.2: Benchmark results of the Titanium-CCA.

Security Level	Number of cycles	Size (byte)
Toy64	KeyGen: 1269090	pk: 12192
	Encrypt: 947906	sk: 12224
	Decrypt: 1107424	ct: 2720
		sk*: 32
Lite96	KeyGen: 1426439	pk: 14720
	Encrypt: 1234901	sk: 14752
	Decrypt: 1425403	ct: 3008
		sk*: 32
Std128	KeyGen: 1806119	pk: 16352
	Encrypt: 1446751	sk: 16384
	Decrypt: 1671578	ct: 3552
		sk*: 32
Med160	KeyGen: 2035675	pk: 18272
	Encrypt: 1855415	sk: 18304
	Decrypt: 2109199	ct: 4544
		sk*: 32
Hi192	KeyGen: 2122547	pk: 20512
	Encrypt: 1986198	sk: 20544
	Decrypt: 2310815	ct: 6048
		sk*: 32
Super256	KeyGen: 2829289	pk: 26912
	Encrypt: 2799390	sk: 26944
	Decrypt: 3247542	ct: 8352
		sk*: 32

3.2.1 Middle product NTT

To compute $a \odot_d b$, for $a \in \mathbb{Z}_q^{<n}[x]$ and $b \in \mathbb{Z}_q^{<n+d-1}[x]$, a naive approach keeps the middle d coefficients of $a \cdot b \in \mathbb{Z}_q^{<2n+d-2}[x]$, and the computation of $a \cdot b$ can be implemented by the $\mathsf{NTT}'_{\dim}$ such that $\dim \geq 2n + d - 2$, according to Lemma 3.2.1.

Algorithm 34 Naive MP-NTT-N

Input: $a \in \mathbb{Z}_q^{<n}[x]$, $b \in \mathbb{Z}_q^{<n+d-1}[x]$.
Output: $\mathbf{c}' = \mathsf{PolToVec}(a \odot_d b) \in \mathbb{Z}_q^d$.

1: **function** MP-NTT-N(a, b)
2: Find an NTT dimension dim that $\dim \geq 2n + d - 2$.
3: Let $\omega_{\dim}$ denote the dim-th root of unity on $\mathbb{Z}_q$.
4: Compute $\mathbf{a}' = \mathsf{NTT}'_{\dim}(\omega_{\dim}, \mathsf{Zpad}(\dim, \mathsf{PolToVec}(a))) \in \mathbb{Z}_q^{\dim}$.
5: Compute $\mathbf{b}' = \mathsf{NTT}'_{\dim}(\omega_{\dim}, \mathsf{Zpad}(\dim, \mathsf{PolToVec}(b))) \in \mathbb{Z}_q^{\dim}$.
6: Compute $\mathbf{c} = \mathsf{NTT}'^{-1}_{\dim}\left(\omega_{\dim}^{-1}, \mathbf{a}' \circ \mathbf{b}'\right) \in \mathbb{Z}_q^{\dim}$.
7: Let $\mathbf{c}' = (c_{n-1}, \ldots, c_{n+d-2}) \in \mathbb{Z}_q^d$.
8: **end function**

The naive MP-NTT-N algorithm requires the computation of three NTTs of dimension dim, which is at least $2n + d - 2$. However, the authors of [HQZ04] proposed a fast MP algorithm for the special case when $d = n$ (i.e. computing $a \odot_n b$, for $a \in \mathbb{Z}_q^{<n}[x]$ and $b \in \mathbb{Z}_q^{<2n-1}[x]$), based on the $2n$-dimensional Fast Fourier Transform (FFT). In this case, this algorithm reduces the lower bound of

the NTT dimension from $3n-2$ to $2n$, which can save about 1/3 of the NTT computation time.

However, the constraint that the number of the coefficients in the MP result is equal to the dimension of a for $a \odot b$ becomes too restrictive for both the parameter choice and the efficient implementation of the Titanium. Here, we observe a generalization of algorithm given in [HQZ04] by removing this limitation. For some arbitrary integers d_1, d_2, d, and k such that $d_1 + d_2 - 1 = d + 2k$, the generalized MP-NTT in Algorithm 35 computes $a \odot_d b$, for $a \in \mathbb{Z}_q^{<d_1}[x]$ and $b \in \mathbb{Z}_q^{<d_2}[x]$ using the NTT'.

Algorithm 35 : Generalized $\mathsf{MP-NTT}$

Input: $a \in \mathbb{Z}_q^{<d_1}[x]$ and $b \in \mathbb{Z}_q^{<d_2}[x]$.
Output: $\mathbf{c}' = \mathsf{PolToVec}(a \odot_d b) \in \mathbb{Z}_q^d$.

1: **function** $\mathsf{MP-NTT}(a, b)$
2: Find such a NTT dimension dim that $\dim \geq d_2$.
3: Let $\omega_{\dim}$ denote the dim-th root of unity.
4: Compute $\mathbf{a}' = \mathsf{NTT}'_{\dim}(\omega_{\dim}, \mathsf{Zpad}(\dim, \mathsf{PolToVec}(\mathsf{Rev}(a)))) \in \mathbb{Z}_q^{\dim}$.
5: Compute $\mathbf{b}' = \mathsf{NTT}'_{\dim}\left(\omega_{\dim}^{-1}, \mathsf{Zpad}(\dim, \mathsf{PolToVec}(b))\right) \in \mathbb{Z}_q^{\dim}$.
6: Compute $\mathbf{c} = \mathsf{NTT}'^{-1}_{\dim}(\omega_{\dim}, \mathbf{a}' \circ \mathbf{b}') \in \mathbb{Z}_q^{\dim}$.
7: Let $\mathbf{c}' = (c_0, \ldots, c_{d-1}) \in \mathbb{Z}_q^d$.
8: **end function**

Theorem 3.2.1 (Adopted from Theorem 11 of [HQZ04])**.** *Let $M_{p,q,n} : R^{<p}[x] \times R^{<q}[x] \to R^{<n}[x]$ and $\Pi_{n,p,q} : R^{<n}[x] \times R^{<p}[x] \to R^{<q}[x]$ be the bilinear forms defined by:*

$$\begin{aligned} M_{p,q,n}(y,z) &= \left(\sum_{j+k=i+p-1} y_j z_k\right)_{0 \leq i < n}, \\ \Pi_{n,p,q}(x,y) &= \left(\sum_{i+j=k} x_i y_j\right)_{0 \leq k < q}. \end{aligned}$$

Then, for any $(X, Y, Z) \in R^{<n}[x] \times R^{<p}[x] \times R^{<q}[x]$, we have:

$$(X | M_{p,q,n}(Y, Z)) = (\Pi_{n,p,q}(X, \mathsf{Rev}(Y)) | Z),$$

where $|$ denotes the canonical inner product of two vectors of the same length. For $q = n+p-1$, $\Pi_{n,p,q}$ is $X \cdot Y$ and $M_{p,q,n}$ is $Y \odot_n Z$.

Lemma 3.2.2. *Let $a \in \mathbb{Z}_q^{<d_1}[x]$, $b \in \mathbb{Z}_q^{<d_2}[x]$, and $\mathbf{c} = \mathsf{MP-NTT}(a, b)$. Then*

$$\mathbf{c} = \mathsf{PolToVec}(a \odot_d b) \in \mathbb{Z}_q^d.$$

Proof. For d_1, d_2, and d be integers such that $d_2 = d_1 + d - 1$, $c \in \mathbb{Z}_q^{<d}[x]$, $a \in \mathbb{Z}_q^{<d_1}[x]$, and $b \in \mathbb{Z}_q^{<d_2}[x]$, we write $\Pi_{d,d_1,d_2}(c, \mathsf{Rev}(a))$ as:

$$\Pi_{d,d_1,d_2}(c, \mathsf{Rev}(a)) = \dim^{-1} \cdot \sum_{m=0}^{\dim -1} \left(\omega_{\dim}^{m \cdot i} | c\right) \left(\omega_{\dim}^{m \cdot j} | \mathsf{Rev}(a)\right) \omega_{\dim}^{-m \cdot k} \bmod q \quad 0 \leq i, j, k < \dim,$$

with the NTT dimension $\dim \geq d_2$. Then, by Theorem 3.2.1, we have:

$$\begin{aligned} (c | M_{d_1,d_2,d}(a, b)) &= (\Pi_{d,d_1,d_2}(c, \mathsf{Rev}(a)) | b) \\ &= \dim^{-1} \cdot \sum_{m=0}^{\dim -1} \left(\omega_{\dim}^{m \cdot i} | c\right) \left(\omega_{\dim}^{m \cdot j} | \mathsf{Rev}(a)\right) \left(\omega_{\dim}^{-m \cdot k} | b\right) \bmod q. \end{aligned}$$

Let $\mathbf{a}' = \mathsf{PolToVec}(\mathsf{Rev}(a))$, $\mathbf{b} = \mathsf{PolToVec}(b)$, $\mathbf{c} = \mathsf{PolToVec}(a \odot_d b)$. We have:

$$\begin{aligned}
\mathbf{c} = M_{d_1,d_2,d}(a,b) &= \mathrm{dim}^{-1} \cdot \sum_{m=0}^{\mathrm{dim}-1} \left(\omega_{\mathrm{dim}}^{m\cdot j} | \mathsf{Rev}(a)\right) \left(\omega_{\mathrm{dim}}^{-m\cdot k} | b\right) \omega_{\mathrm{dim}}^{m\cdot i} \bmod q \\
&= \mathrm{dim}^{-1} \cdot \sum_{m=0}^{\mathrm{dim}-1} \left(\omega_{\mathrm{dim}}^{m\cdot i}\right) \left(\mathsf{NTT}'_{\mathrm{dim}}(\omega_{\mathrm{dim}}, \mathbf{a}')_m\right) \left(\mathsf{NTT}'_{\mathrm{dim}}\left(\omega_{\mathrm{dim}}^{-1}, \mathbf{b}\right)_m\right) \\
&= \mathsf{NTT}'^{-1}_{\mathrm{dim}}\left(\omega_{\mathrm{dim}}, \mathsf{NTT}'_{\mathrm{dim}}(\omega_{\mathrm{dim}}, \mathbf{a}') \circ \mathsf{NTT}'_{\mathrm{dim}}\left(\omega_{\mathrm{dim}}^{-1}, \mathbf{b}\right)\right) \\
&= \mathsf{MP\text{-}NTT}(a,b),
\end{aligned}$$

for $0 \le i,j,k < \mathrm{dim}$ and $\mathsf{NTT}'_{\mathrm{dim}}(\omega_{\mathrm{dim}}, \mathbf{a})_m$ is the m-th coordinate of the NTT result. □

3.2.2 Partial MP-NTT

Let $\mathrm{dim} \ge d_2 > d_1$ in computation of $\mathsf{MP\text{-}NTT}(a,b)$, with $a \in \mathbb{Z}_q^{<d_1}[x]$ and $b \in \mathbb{Z}_q^{<d_2}[x]$. There are many padded zeros in $\mathsf{Zpad}(\mathrm{dim}, \mathsf{PolToVec}(a))$ and $\mathsf{Zpad}(\mathrm{dim}, \mathsf{PolToVec}(b))$. The naive implementation of NTT' is inefficient for such a polynomial with lots of zero coordinates, due to the unnecessary additions/subtractions and multiplications with zero. However, for the NTT dimension $\mathrm{dim} = \mathrm{dim}_1 \cdot \mathrm{dim}_2$ and some integer $m_a, m_b \le \mathrm{dim}_1$, if we assume that $a_i = 0$, $b_j = 0$ for all $i \ge m_a \cdot \mathrm{dim}_2$, $j \ge m_b \cdot \mathrm{dim}_2$, we can compute the partial NTT which only takes the first $m \cdot \mathrm{dim}_2$ coordinates as the input.

The partial NTT is similar to the FFT decomposition algorithm in [SB93]. Let us denote $\omega_{\mathrm{dim}_1} = \omega_{\mathrm{dim}}^{\mathrm{dim}_2}$ and $\omega_{\mathrm{dim}_2} = \omega_{\mathrm{dim}}^{\mathrm{dim}_1}$. For some integer $m \le \mathrm{dim}_1$, the $\mathsf{NTT}'_{\mathrm{dim}}(\omega_{\mathrm{dim}}, \mathbf{a})$ algorithm, which only takes the first $m \cdot \mathrm{dim}_2$ coordinates as the input can be written as:

$$\mathsf{NTT}'_{\mathrm{dim}}(\omega_{\mathrm{dim}}, \mathbf{a})_{k_1 + \mathrm{dim}_1 \cdot k_2} = \sum_{n_2=0}^{\mathrm{dim}_2 - 1} \left[\omega_{\mathrm{dim}}^{n_2 \cdot k_1} \cdot \left(\sum_{n_1=0}^{m-1} a_{\mathrm{dim}_2 \cdot n_1 + n_2} \cdot \omega_{\mathrm{dim}_1}^{n_1 \cdot k_1}\right)\right] \cdot \omega_{\mathrm{dim}_2}^{n_2 \cdot k_2},$$

for $0 \le k_1 < \mathrm{dim}_1$ and $0 \le k_2 < \mathrm{dim}_2$. Similarly, the $\mathsf{MP\text{-}NTT}$ only requires the first d coordinates output of NTT'^{-1}, and the naive implementation of NTT'^{-1} will waste lots of CPU time in computing the unnecessary coordinates. If we assume that we only needs the first $m \cdot \mathrm{dim}_2$, for $m \le \mathrm{dim}_1$ coordinates output, we can also re-write the NTT'^{-1} decomposition similar to [SB93]:

$$\mathsf{NTT}'^{-1}_{\mathrm{dim}}(\omega_{\mathrm{dim}}, \mathbf{a})_{k_2 + \mathrm{dim}_2 \cdot k_1} = \mathrm{dim}^{-1} \cdot \left\{\sum_{n_1=0}^{\mathrm{dim}_1 - 1} \left[\omega_{\mathrm{dim}}^{n_1 \cdot k_2} \cdot \left(\sum_{n_2=0}^{\mathrm{dim}_2 - 1} a_{\mathrm{dim}_1 \cdot n_2 + n_1} \cdot \omega_{\mathrm{dim}_2}^{n_2 \cdot k_2}\right)\right] \cdot \omega_{\mathrm{dim}_1}^{n_1 \cdot k_1}\right\},$$

for $0 \le k_1 < m$ and $0 \le k_2 < \mathrm{dim}_2$.

These decompositions can be viewed as the combination of two sub-dimension NTTs on dimension dim_1 and dim_2, respectively. For the efficient implementation of the Titanium, we select our NTT dimension $\mathsf{d}_1, \mathsf{d}_2, \mathsf{d}_3$ such that dim_1 is some small arbitrary integer (less than 13 in our defined parameter sets), and dim_2 is a multiple of 256. Therefore, we achieve an efficient implementation by combining an efficient radix-2 NTT algorithm for the large sub-dimension dim_2, which has the time complexity $O(\mathrm{dim}_2 \log \mathrm{dim}_2)$ operations in $\mathbb{Z}_q$, and a classical NTT algorithm with complexity $O((\mathrm{dim}_1)^2)$ operations in $\mathbb{Z}_q$ for the small sub-dimension dim_1. The radix-2 NTT subroutine can in fact be any $O(n \log n)$ NTT algorithm with both the input and the output in the natural order. However, for the efficiency of the AVX2 parallelized implementation discussed in Chapter 3.4.4, we choose the Gentleman-Sande NTT implementation used by [ADPS16].

We pre-compute the tables to store the powers of ω_{dim}, ω_{dim_1}, and ω_{dim_2} for each NTT dimension respectively. We denote the matrix $\Omega \in \mathbb{Z}_q^{\mathrm{dim}_1 \times \mathrm{dim}_1}$ with:

$$\Omega_{i,j} = \omega_{\mathrm{dim}_1}^{(i-1)(j-1)} \quad 1 \le i,j \le \mathrm{dim}_1.$$

To implement the NTT of sub-dimension $\dim_1$, for each $n_2 < \dim_2$, let $\mathbf{a}_{u,n_2} \in \mathbb{Z}_q^u$ be the decimation

$$\left(a_{n_2}, a_{\dim_2 + n_2}, \ldots, a_{(u-1)\cdot\dim_2 + n_2}\right)$$

of $\mathbf{a}$. Let $\mathsf{NTT}_{m\to\dim_1}(\mathbf{a}, n_2)$ denote the NTT which takes $\mathbf{a}_{m,n_2} \in \mathbb{Z}_q^m$ as the input and generates $\dim_1$ coordinates output. Similarly, let $\mathsf{NTT}_{\dim_1\to m}(\mathbf{a}, n_2)$ denote the NTT which takes $\mathbf{a}_{\dim_1,n_2} \in \mathbb{Z}_q^{\dim_1}$ as the input and generates the first m coordinates output. We define $\Omega_{u\times v} \in \mathbb{Z}_q^{u\times v}$ as the submatrix of Ω which takes the first u rows and v columns. Then, we have:

$$\begin{aligned}\mathsf{NTT}_{m\to\dim_1}(\mathbf{a}, n_2) &= \Omega_{\dim_1\times m}\cdot\mathbf{a}_{m,n_2},\\ \mathsf{NTT}_{\dim_1\to m}(\mathbf{a}, n_2) &= \Omega_{m\times\dim_1}\cdot\mathbf{a}_{\dim_1,n_2}.\end{aligned}$$

The pseudocode of our partial NTT implementations are as follows. Note that both the output of $\mathsf{NTT\text{-}P}_{\dim}$ and the input of $\mathsf{NTT\text{-}P}_{\dim}^{-1}$ are permuted by $\mathsf{Perm}_{\dim}$, the permutation defined in the previous Chapter, and $\mathsf{NTT\text{-}P}_{\dim}^{-1}$ is the step-by-step reversal of $\mathsf{NTT\text{-}P}_{\dim}$.

Algorithm 36 : Partial NTT Algorithm: $\mathsf{NTT\text{-}P}_{\dim}$

Input: m, $\omega'_{\dim}$, and $\mathbf{a}$ satisfying $\dim = \dim_1\cdot\dim_2$ and $a_i = 0$ for all $i \geq m\cdot\dim_2$.
Output: Partial $\mathsf{Perm}_{\dim}\left(\mathsf{NTT}'_{\dim}(\omega'_{\dim}, \mathbf{a})\right) \in \mathbb{Z}_q^{\dim}$ transformation.

1: **function** $\mathsf{NTT\text{-}P}_{\dim}(m, \omega', \mathbf{a})$
2: **for** $n_2 = 0, \ldots, \dim_2 - 1$ **do**
3: Let $\mathbf{a}'_1$ be the computation result of classical $\mathsf{NTT}_{m\to\dim_1}(\mathbf{a}, n_2)$.
4: Set $\left(a_{n_2}, a_{\dim_2+n_2}, \ldots, a_{(\dim_1-1)\cdot\dim_2+n_2}\right) = \mathbf{a}'_1$.
5: **end for**
6: **for** $n_1 = 0, \ldots, \dim_1 - 1$ **do**
7: **for** $n_2 = 0, \ldots, \dim_2 - 1$ **do**
8: Set $a_{\dim_2\cdot n_1+n_2} = a_{\dim_2\cdot n_1+n_2}\cdot\omega'^{n_1n_2}_{\dim}$.
9: **end for**
10: **end for**
11: **for** $n_1 = 0, \ldots, \dim_1 - 1$ **do**
12: Let $\mathbf{a}_2$ be the decimation $\left(a_{n_1\cdot\dim_2}, a_{n_1\cdot\dim_2+1}\ldots, a_{(n_1+1)\cdot\dim_2-1}\right)$ of $\mathbf{a}$.
13: Let $\mathbf{a}'_2$ be the computation result of radix-2 $\mathsf{NTT}_{\dim_2}\left(\omega'_{\dim_2}, \mathbf{a}_2\right)$.
14: Set $\left(a_{n_1\cdot\dim_2}, a_{n_1\cdot\dim_2+1}\ldots, a_{(n_1+1)\cdot\dim_2-1}\right) = \mathbf{a}'_2$.
15: **end for**
16: **end function**

Algorithm 37 : Partial NTT inverse algorithm: $\mathsf{NTT\text{-}P}_{\mathrm{dim}}^{-1}$

Input: m, ω'_{dim} and $\mathbf{a}$. Assume $\mathrm{dim} = \mathrm{dim}_1 \cdot \mathrm{dim}_2$ and $\mathbf{a}$ is permutated by $\mathsf{Perm}_{\mathrm{dim}}$.
Output: the first $m \cdot \mathrm{dim}_2$ coordinates of the $\mathsf{NTT}'^{-1}_{\mathrm{dim}}(\omega'_{\mathrm{dim}}, \mathsf{InvPerm}_{\mathrm{dim}}(\mathbf{a}))$.

function $\mathsf{NTT\text{-}P}_{\mathrm{dim}}^{-1}(m, \omega', \mathbf{a})$
  for $n_1 = 0, \ldots, \mathrm{dim}_1 - 1$ **do**
    Let $\mathbf{a}_2$ be the decimation $\left(a_{n_1 \cdot \mathrm{dim}_2}, a_{n_1 \cdot \mathrm{dim}_2 + 1} \ldots, a_{(n_1+1) \cdot \mathrm{dim}_2 - 1}\right)$ of $\mathbf{a}$.
    Let $\mathbf{a}'_2$ be the computation result of radix-2 $\mathsf{NTT}_{\mathrm{dim}_2}\left(\omega'_{\mathrm{dim}_2}, \mathbf{a}_2\right)$.
    Set $\left(a_{n_1 \cdot \mathrm{dim}_2}, a_{n_1 \cdot \mathrm{dim}_2 + 1} \ldots, a_{(n_1+1) \cdot \mathrm{dim}_2 - 1}\right) = \mathbf{a}'_2$.
  end for
  for $n_1 = 0, \ldots, \mathrm{dim}_1 - 1$ **do**
    for $n_2 = 0, \ldots, \mathrm{dim}_2 - 1$ **do**
      Set $a_{\mathrm{dim}_2 \cdot n_1 + n_2} = a_{\mathrm{dim}_2 \cdot n_1 + n_2} \cdot \omega'^{n_1 \cdot n_2}_{\mathrm{dim}}$.
    end for
  end for
  for $n_2 = 0, \ldots, \mathrm{dim}_2 - 1$ **do**
    Let $\mathbf{a}'_1$ be the computation result of classical $\mathsf{NTT}_{\mathrm{dim}_1 \to m}(\mathbf{a}, n_2)$.
    Set $\left(a_{n_2}, a_{\mathrm{dim}_2 + n_2}, \ldots, a_{(m-1) \cdot \mathrm{dim}_2 + n_2}\right) = \mathbf{a}'_1$.
  end for
  for $i = 0, \ldots, m \cdot \mathrm{dim}_2 - 1$ **do**
    Set $a_i = a_i \cdot \mathrm{dim}^{-1}$.
  end for
end function

We now state the relation of our partial NTT algorithm to the standard NTT algorithm.

Lemma 3.2.3. *Let* $\mathsf{Perm}_{\mathrm{dim}}$ *and* $\mathsf{InvPerm}_{\mathrm{dim}}$ *denote the permutation defined in Algorithms 15 and 16 respectively. Let* $\mathrm{dim} = \mathrm{dim}_1 \cdot \mathrm{dim}_2$ *and integer* $m \leq \mathrm{dim}_1$*, for a vector* $\mathbf{a}$ *such that* $a_i = 0$ *for all* $i \geq m \cdot \mathrm{dim}_2$*. The following equalities hold:*

- $\mathsf{NTT\text{-}P}_{\mathrm{dim}}(m, \omega_{\mathrm{dim}}, \mathbf{a}) = \mathsf{Perm}_{\mathrm{dim}}(\mathsf{NTT}_{\mathrm{dim}}(\omega_{\mathrm{dim}}, \mathbf{a}))$,
- $\mathsf{NTT\text{-}P}_{\mathrm{dim}}\left(m, \omega^{-1}_{\mathrm{dim}}, \mathbf{a}\right) = \mathsf{Perm}_{\mathrm{dim}}\left(\mathrm{dim} \cdot \mathsf{NTT}^{-1}_{\mathrm{dim}}(\omega_{\mathrm{dim}}, \mathbf{a})\right)$.

For a vector $\mathbf{a}$ *permuted by* $\mathsf{Perm}_{\mathrm{dim}}$*, we have:*

- $\mathsf{NTT\text{-}P}^{-1}_{\mathrm{dim}}(m, \omega_{\mathrm{dim}}, \mathbf{a})_i = \left(\mathrm{dim}^{-1} \cdot \mathsf{NTT}_{\mathrm{dim}}(\omega_{\mathrm{dim}}, \mathsf{InvPerm}_{\mathrm{dim}}(\mathbf{a}))\right)_i$,
- $\mathsf{NTT\text{-}P}^{-1}_{\mathrm{dim}}\left(m, \omega^{-1}_{\mathrm{dim}}, \mathbf{a}\right)_i = \mathsf{NTT}^{-1}_{\mathrm{dim}}(\omega_{\mathrm{dim}}, \mathsf{InvPerm}_{\mathrm{dim}}(\mathbf{a}))_i$,

for $0 \leq i < m \cdot \mathrm{dim}_2$*. The* $\mathsf{NTT}_{\mathrm{dim}}$ *and the* $\mathsf{NTT}^{-1}_{\mathrm{dim}}$ *are the notations used in the scheme specification.*

Given NTT dimension $\mathrm{dim} = \mathrm{dim}_1 \cdot \mathrm{dim}_2$, $a \in \mathbb{Z}_q^{<d_1}[x]$, and $b \in \mathbb{Z}_q^{<d_2}[x]$ such that $d_1 \leq m_a \cdot \mathrm{dim}_2$ and $d_2 \leq m_b \cdot \mathrm{dim}_2$ for some integer $m_a, m_b \leq \mathrm{dim}_1$, we provide the $\mathsf{MP\text{-}NTT\text{-}P}$ algorithm merged with the $\mathsf{NTT\text{-}P}$ as follows:

Algorithm 38 : Partial MP-NTT-P

Input: $a \in \mathbb{Z}_q^{<d_1}[x]$ and $b \in \mathbb{Z}_q^{<d_2}[x]$.
Output: $\mathbf{c}' = \mathsf{PolToVec}(a \odot_d b) \in \mathbb{Z}_q^d$.
1: **function** MP-NTT-P(a, b)
2: Find such a NTT dimension $\dim = \dim_1 \cdot \dim_2$ that $\dim \geq d_2$.
3: Let $\omega_{\dim}$ denote the dim-th root of unity.
4: Find an integer $m_a \leq \dim_1$ such that $d_1 \leq m_a \cdot \dim_2$.
5: Compute $\mathbf{a}' = \mathsf{NTT\text{-}P}_{\dim}\left(m_a, \omega_{\dim}, \mathsf{Zpad}\left(m_a \cdot \dim_2, \mathsf{PolToVec}\left(\mathsf{Rev}\left(a\right)\right)\right)\right) \in \mathbb{Z}_q^{\dim}$.
6: Find an integer $m_b \leq \dim_1$ such that $d_2 \leq m_b \cdot \dim_2$.
7: Compute $\mathbf{b}' = \mathsf{NTT\text{-}P}_{\dim}\left(m_b, \omega_{\dim}^{-1}, \mathsf{Zpad}\left(m_b \cdot \dim_2, \mathsf{PolToVec}\left(b\right)\right)\right) \in \mathbb{Z}_q^{\dim}$.
8: Find an integer $m_c \leq \dim_1$ such that $d \leq m_c \cdot \dim_2$.
9: Compute $\mathbf{c} = \mathsf{NTT\text{-}P}_{\dim}^{-1}\left(m_c, \omega_{\dim}, \mathbf{a}' \circ \mathbf{b}'\right) \in \mathbb{Z}_q^{m_c \cdot \dim_2}$.
10: Let $\mathbf{c}' = (c_0, \ldots, c_{d-1}) \in \mathbb{Z}_q^d$.
11: **end function**

Lemma 3.2.4. *Given* NTT *dimension* $\dim = \dim_1 \cdot \dim_2$, $a \in \mathbb{Z}_q^{<d_1}[x]$, *and* $b \in \mathbb{Z}_q^{<d_2}[x]$ *such that* $d_1 \leq m_a \cdot \dim_2$ *and* $d_2 \leq m_b \cdot \dim_2$ *for some integer* $m_a, m_b \leq \dim_1$, *we have*

$$\mathsf{MP\text{-}NTT\text{-}P}\left(a, b\right) = \mathsf{MP\text{-}NTT}\left(a, b\right).$$

Let $T_{\dim_2}$ denote the running time of the radix-2 NTT subroutine for dimension $\dim_2$, which is the dominate part of NTT-P's running time. Therefore, the MP-NTT-P algorithm has time complexity ($\mathbb{Z}_q$ operations) of about:

$$3 \cdot \left(\dim_1 \cdot T_{\dim_2}\right) + O\left(\dim\right).$$

We also provide a variant of the MP-NTT-P taking $\widehat{\mathbf{b}} \in \mathbb{Z}_q^{\dim}$ as the input, which we call Partial NTT-P pre-transformed $b \in \mathbb{Z}_q^{<d_2}[x]$ by the computation of Line 7 in MP-NTT-P, i.e.

$$\widehat{\mathbf{b}} = \mathsf{NTT\text{-}P}_{\dim}\left(m_b, \omega_{\dim}^{-1}, \mathsf{Zpad}\left(m_b \cdot \dim_2, \mathsf{PolToVec}\left(b\right)\right)\right).$$

Algorithm 39 : Partial MP-NTT-P$_{\mathsf{pre}}$

Input: d, $a \in \mathbb{Z}_q^{<d_1}[x]$, and $\widehat{\mathbf{b}} \in \mathbb{Z}_q^{\dim}$ satisfying $\dim = \dim_1 \cdot \dim_2$.
Output: $\mathbf{c}' = \mathsf{PolToVec}(a \odot_d b) \in \mathbb{Z}_q^d$.
1: **function** MP-NTT-P$_{\mathsf{pre}}(d, a, \widehat{\mathbf{b}})$
2: Let $\omega_{\dim}$ denote the dim-th root of unity.
3: Find an integer $m_a \leq \dim_1$ such that $d_1 \leq m_a \cdot \dim_2$.
4: Compute $\mathbf{a}' = \mathsf{NTT} - \mathsf{P}_{\dim}\left(m_a, \omega_{\dim}, \mathsf{Zpad}\left(m_a \cdot \dim_2, \mathsf{PolToVec}\left(\mathsf{Rev}\left(a\right)\right)\right)\right) \in \mathbb{Z}_q^{\dim}$.
5: Find an integer $m_c \leq \dim_1$ such that $d \leq m_c \cdot \dim_2$.
6: Compute $\mathbf{c} = \mathsf{NTT} - \mathsf{P}_{\dim}^{-1}\left(m_c, \omega_{\dim}, \mathbf{a}' \circ \widehat{\mathbf{b}}\right) \in \mathbb{Z}_q^{m_c \cdot \dim_2}$.
7: Let $\mathbf{c}' = (c_0, \ldots, c_{d-1}) \in \mathbb{Z}_q^d$.
8: **end function**

3.3 Optimised Titanium-CPA Algorithms

We now provide the optimised Titanium-CPA algorithms by using the MP-NTT-P$_{\mathsf{pre}}$ as follows:

Algorithm 40 : Optimized Titanium-CPA.KeyGen-Opt

Input: $\mathsf{seedkg} = \mathsf{Randbytes}(32) \in \mathsf{byte}^{32}$.
Output: $\mathsf{pk} \in \mathsf{byte}^{\mathsf{lenpk}}$ and $\mathsf{sk} \in \mathsf{byte}^{\mathsf{lensk}}$.

function KeyGen-Opt($\mathsf{seedkg} = \mathsf{Randbytes}(32)$)
  Let $\mathsf{prgst} = \mathsf{PRG.Init}(\mathsf{seedkg})$.
  Let $(\mathsf{prgst}, \mathsf{seedsk}) = \mathsf{PRG.Out}(\mathsf{prgst}, 32) \in \mathsf{StSp}_{\mathsf{prg}} \times \mathsf{byte}^{32}$.
  Let $(\mathsf{prgst}, \mathsf{seedpk}) = \mathsf{PRG.Out}(\mathsf{prgst}, 32) \in \mathsf{StSp}_{\mathsf{prg}} \times \mathsf{byte}^{32}$.
  Let $(\mathsf{prgst}, \widehat{\mathbf{s}}) = \mathsf{Samps}(\mathsf{seedsk}) \in \mathsf{StSp}_{\mathsf{prg}} \times \mathbb{Z}_q^{\mathsf{d}_3}$.
  Let $(\bar{a_1}, \ldots, \bar{a_t}) = \mathsf{Sampa}(\mathsf{seedpk}) \in (\mathbb{Z}_q^{<n}[x])^t$.
  Let $(e_1, \ldots, e_t) = \mathsf{Sampe}(\mathsf{prgst}) \in (\mathbb{Z}_q^{<d+k}[x])^t$.
  for $i \leq t$ **do**
    Let $b_i = \mathsf{VecToPol}\left(\mathsf{MP\text{-}NTT\text{-}P}_{\mathsf{pre}}\left(d+k, \mathsf{Rev}(\bar{a_i}), \widehat{\mathbf{s}}\right)\right) + e_i \in \mathbb{Z}_q^{<d+k}[x]$.
    Let $\widehat{\mathbf{b}}_i' = \mathsf{Zpad}\left(\mathsf{d}_1, \mathsf{PolToVec}(b_i)\right) \in \mathbb{Z}_q^{\mathsf{d}_1}$.
    Let $\widehat{\mathbf{b}}_i = \mathsf{Perm}_{\mathsf{d}_1}\left(\mathsf{NTT}'_{\mathsf{d}_1}\left(\omega_1^{-1}, \widehat{\mathbf{b}}_i\right)\right) \in \mathbb{Z}_q^{\mathsf{d}_1}$.
  end for
  Let $\mathsf{pk} = \mathsf{Encodepk}(\mathsf{seedpk} \in \mathsf{byte}^{32}, (\widehat{\mathbf{b}}_1, \ldots, \widehat{\mathbf{b}}_t)) \in \mathsf{byte}^{\mathsf{lenpk}}$.
  Let $\mathsf{sk} = \mathsf{seedsk} \in \mathsf{byte}^{\mathsf{lensk}}$.
end function

Algorithm 41 : Optimized Titanium-CPA.Encrypt-Opt

Input: $\mathsf{pk} \in \mathsf{byte}^{\mathsf{lenpk}}$, $\mathsf{m} \in \mathsf{byte}^{32}$, and $\mathsf{seedr} = \mathsf{Randbytes}(32) \in \mathsf{byte}^{32}$.
Output: $\mathsf{ct} \in \mathsf{byte}^{\mathsf{lenct}}$.

function Encrypt-Opt($\mathsf{pk}, \mathsf{m}, \mathsf{seedr} = \mathsf{Randbytes}(32)$)
  Let $(\mathsf{seedpk}, (\widehat{\mathbf{b}}_1, \ldots, \widehat{\mathbf{b}}_t)) = \mathsf{Decodepk}(\mathsf{pk}) \in \mathsf{byte}^{32} \times (\mathbb{Z}_q^{\mathsf{d}_1})^t$.
  Let $m = \mathsf{Decodem}(\mathsf{m}) \in \mathbb{Z}_q^{<d}[x]$.
  Let $(\bar{a}_1, \ldots, \bar{a}_t) = \mathsf{Sampa}(\mathsf{seedpk}) \in (\mathbb{Z}_q^{<n}[x])^t$.
  Let $(r_1, \ldots, r_t) = \mathsf{Sampr}(\mathsf{seedr}) \in (\mathbb{Z}_q^{<k+1}[x])^t$.
  For $\mathsf{d}_2 = \dim_1 \cdot \dim_2$, find the integer $m_r, m_a \leq \dim_1$ such that $k+1 \leq m_r \cdot \dim_2$, $n \leq m_a \cdot \dim_2$.
  for $i \leq t$ **do**
    Let $\widehat{\mathbf{r}}_i = \mathsf{NTT\text{-}P}_{\mathsf{d}_2}\left(m_r, \omega_2, \mathsf{Zpad}\left(m_r \cdot \dim_2, \mathsf{PolToVec}\left(r_i\right)\right)\right) \in \mathbb{Z}_q^{\mathsf{d}_2}$.
    Let $\widehat{\mathbf{a}}_i = \mathsf{NTT\text{-}P}_{\mathsf{d}_2}\left(m_a, \omega_2, \mathsf{Zpad}\left(m_a \cdot \dim_2, \mathsf{PolToVec}\left(a_i\right)\right)\right) \in \mathbb{Z}_q^{\mathsf{d}_2}$.
  end for
  Let $\widehat{\mathbf{c}}_1 = \sum_{i=1}^{t} \widehat{\mathbf{r}}_i \circ \widehat{\mathbf{a}}_i \in \mathbb{Z}_q^{\mathsf{d}_2}$.
  Let $c_2' = \sum_{i=1}^{t} \mathsf{VecToPol}\left(\mathsf{MP\text{-}NTT\text{-}P}_{\mathsf{pre}}\left(d, \mathsf{Rev}(r_i), \widehat{\mathbf{b}}_i\right)\right) + \lfloor q/p \rfloor \cdot m \in \mathbb{Z}_q^{<d}[x]$.
  Let $c_2 = \mathsf{Chop}(\mathsf{cmp}, c_2') \in \mathbb{Z}_q^{<d}[x]$.
  Let $\mathsf{ct} = \mathsf{Encodect}(\widehat{\mathbf{c}}_1, c_2) \in \mathsf{byte}^{\mathsf{lenct}}$.
end function

Algorithm 42 : Titanium-CPA.Decrypt-Opt

Input: $\mathsf{sk} \in \mathsf{byte}^{\mathsf{lensk}}$ and $\mathsf{ct} \in \mathsf{byte}^{\mathsf{lenct}}$.
Output: $\mathsf{m} \in \mathsf{byte}^{32}$.
1: **function** Decrypt-Opt(sk, ct)
2: Let $\mathsf{seedsk} = \mathsf{sk} \in \mathsf{byte}^{\mathsf{lensk}}$.
3: Let $(\widehat{\mathbf{s}}, \mathsf{prgst}) = \mathsf{Samps}(\mathsf{seedsk}) \in \mathbb{Z}_q^{\mathsf{d}_3} \times \mathsf{StSp}_{\mathsf{prg}}$.
4: Let $(\widehat{\mathbf{c}}_1, c_2) = \mathsf{Decodect}(\mathsf{ct}) \in \mathbb{Z}_q^{\mathsf{d}_2} \times \mathbb{Z}_q^{<d}[x]$.
5: Let $c_1' = \mathsf{Trunc}(n+k, \mathsf{VecToPol}(\mathsf{NTT}'^{-1}_{\mathsf{d}_2}(\omega_2^{-1}, \mathsf{InvPerm}_{\mathsf{d}_2}(\widehat{\mathbf{c}}_1)))) \in \mathbb{Z}_q^{<n+k}[x]$.
6: Let $c' = c_2 - \mathsf{VecToPol}\left(\mathsf{MP\text{-}NTT\text{-}P}_{\mathsf{pre}}\left(d, \mathsf{Rev}(c_1'), \widehat{\mathbf{s}}\right)\right) \in \mathbb{Z}_q^{<d}[x]$.
7: Let $m' = \mathsf{Round}(\lfloor q/p \rfloor, c') \in \mathbb{Z}_p^{<d}[x]$.
8: Let $\mathsf{m} = \mathsf{Encodem}(m') \in \mathsf{byte}^{32}$.
9: **end function**

3.4 Additional implementation aspects

3.4.1 Simplifying the middle product

The first step of the NTT middle product will reverse the polynomial on the left hand side. However, to compute $\mathsf{Rev}(a) \odot b$, the first step of the NTT middle product directly takes a without any reversal. In our Titanium-CPA specification, all of the left hand side polynomials are in the form $\mathsf{Rev}(a)$, so there is no reversal in our actual implementation using the NTT middle product.

Also, from Theorem 3.2.1, for $a \in \mathbb{Z}_q^{<d_1}[x], b \in \mathbb{Z}_q^{<d_2}[x]$, we have:

$$a \odot_d b = M_{d_1,d_2,d}(a,b) = \left(\sum_{j+k=i+d_1-1} a_j b_k \right)_{0 \le i < d}.$$

Since $j+k = i+d_1-1, 0 \le i < d$, we get $k < d+d_1-1$. Therefore, only the coefficients corresponding to monomials with degree less than $d+d_1-1$ in polynomial b contribute to $a \odot_d b$, so those coefficients corresponding to monomials with degree higher than d_1+d-1 in polynomial b do not affect the MP result. Therefore, we can directly use the secret key $\widehat{\mathbf{s}}$ sampled from the NTT domain in dimension $\mathsf{d}_3 \ge n+d+k-1$ without truncating the coefficients with the degree higher than $n+d+k-1$ in its pre-image.

3.4.2 Fast modulo reduction

Since the generated assembly code of the "%" operator in C programming language depends on the compiler and therefore this operator should be avoided in constant time implementation [Sei18], we need an efficient constant time modulo reduction algorithm. For the multiplication between the coordinate of a vector and the root of unity ω in the NTT, we apply the lazy Montgomery reduction with the pre-computed ω table in the Montgomery form [Har14]. The following algorithm computes $x' = x \cdot \rho^{-1} \bmod q, x' \in [0, 2q-1]$, where ρ is power of 2 satisfying $x < q\rho$ and $q < \rho$:

Algorithm 43 : Montgomery

Input: Integers x and q.
Output: $x' \in \mathbb{Z}_q$.
1: **function** Montgomery(x, q)
2: Let $b = -q^{-1} \bmod \rho$.
3: Let $m = ((x \bmod \rho) \cdot b) \bmod \rho$.
4: Let $x' = (x + mq)/\rho$.
5: **end function**

If we pre-transform the roots ω^i to their Montgomery form $\omega^i \cdot \rho \bmod q$, for an x in the natural form, the Montgomery reduction of $x \cdot (\omega^i \cdot \rho)$ will become $x \cdot \omega^i \bmod q$ in the natural form. Also, we define the word width as 32 bits (i.e. $\rho = 2^{32}$ in Algorithm 43), then the Line 3 in Algorithm 43 is simplified as the low product between x and b, which is faster than the full product [Sei18].

For the pointwise multiplication reduction and addition/subtraction reduction, to avoid the conversion between the natural form and the Montgomery form, we employ the lazy Barrett reduction [Har14, ADPS16]. The following algorithm computes $x' = x \bmod q, x' \in [0, 2q-1]$:

Algorithm 44 : Barrett

Input: Integers x and q.
Output: $x' \in [0, 2q-1]$.
1: **function** Barrett(x, q)
2: Find an integer k such that $x < 2^k q$.
3: Let $b = \left\lfloor 2^k / q \right\rfloor$.
4: Let $m = \left\lfloor (x \cdot b) / 2^k \right\rfloor$.
5: Let $x' = x - mq$.
6: **end function**

To reduce the number of reductions, we can omit the addition/subtraction Barrett reductions on some intermediate levels of the NTT [ADPS16]. This requires the factor ρ and k in Algorithm 43 and Algorithm 44 to be large enough for larger input, respectively. For our radix-2 NTT subroutine, we select $\rho = 2^{32}$ and $k = 32$, in which case we can omit all the Barrett reductions in the intermediate levels except the last level for all specified parameter sets. This reduces the number of reductions by about half in the NTT.

3.4.3 Optimised uniform sampler

We apply the optimisation in [GOPS13] to reduce the rejection rate of the uniform sampler in Algorithm 8. We compute $\mathsf{ZqRej} = 2^{8 \cdot \mathsf{bytpm}} - (2^{8 \cdot \mathsf{bytpm}} \bmod q)$, which is the multiple of q closest to $2^{8 \cdot \mathsf{bytpm}}$. Then, we perform the rejection sampling with the success rate $r = \mathsf{ZqRej}/2^{8 \cdot \mathsf{bytpm}}$, and reduce the samples to $\mathbb{Z}_q$. We refer to Chapter 7 for an in-depth analysis of the run-time of this sampler.

3.4.4 AVX2 instruction set

The AVX2 instruction set will significantly improve the performance of the Gentleman-Sande radix-2 NTT on newer Intel CPUs, as suggested by [ADPS16]. Therefore, we also provide the AVX2 parallelised version as one of our additional implementations in the submission. The AVX2 instruction can pack four 64-bit integers into a vector and perform the arithmetic operations simultaneously. Since all of our defined NTT dimensions are divisible by 4, the implementation is straightforward.

During the benchmark, the Keccak library is compiled toward the target architecture `Haswell`, which is also implemented by the AVX2 instruction set. We use Clang 5.0.1 to compile our AVX2 implementations, with the compiler switch `-O3 -march=native` enabled.

The Table 3.3 compares the AVX2 benchmark results with the implementations without the AVX2 instructions for each parameter set of the Titanium-CPA schemes. The Table 3.4 compares the AVX2 benchmark results with the implementations without the AVX2 instructions for each parameter set of the Titanium-CCA schemes.

3.4.5 AES variant of Titanium

The PRG costs a large amount of running time for the lattice-based crypto schemes requiring lots of randomness. Recent evidence [NAB+17] shows that by applying the Advanced Encryption Standard

Table 3.3: Comparison of the Titanium-CPA benchmark results with and without the AVX2.

Par. Set	Number of cycles (non-AVX2)	Number of cycles (AVX2)
Toy64	KeyGen: 1264647	KeyGen: 692916
	Encrypt: 900120	Encrypt: 525429
	Decrypt: 152705	Decrypt: 85511
Lite96	KeyGen: 1269479	KeyGen: 689425
	Encrypt: 1073167	Encrypt: 583561
	Decrypt: 183832	Decrypt: 102190
Std128	KeyGen: 1619550	KeyGen: 828542
	Encrypt: 1262047	Encrypt: 742541
	Decrypt: 217612	Decrypt: 116311
Med160	KeyGen: 1877257	KeyGen: 1036984
	Encrypt: 1646486	Encrypt: 919323
	Decrypt: 253458	Decrypt: 140187
Hi192	KeyGen: 1894719	KeyGen: 1062465
	Encrypt: 1763250	Encrypt: 993939
	Decrypt: 323977	Decrypt: 183561
Super256	KeyGen: 2486436	KeyGen: 1262301
	Encrypt: 2450834	Encrypt: 1376773
	Decrypt: 439522	Decrypt: 229425

New Instructions (AES-NI), the efficiency of the PRG can be significantly improved for such schemes on Intel CPUs. However, for those platforms without the AES hardware instructions, the AES software implementation is usually vulnerable to timing attacks [Ber05, BDK+17]. Therefore, we provide the AES variant of Titanium only as one of the additional implementations.

We use the following counter mode of the AES as our PRG: the high 32 bits of the 128-bit AES block is a 32-bit unsigned integer counter in little endian form, and the other bits in the block are zero (i.e. extend the counter c in PRG.Init and PRG.Out to 128 bits with zero). The counter starts from 0. For each block, we use the $\mathsf{AES}-256-\mathsf{ECB}$ to generate the randomness (i.e. replace the XOF in PRG.Out with $\mathsf{AES}-256-\mathsf{ECB}(K, c||0\ldots0 \in \mathsf{byte}^{16})$). For the non-AVX2 implementations, we employ the AES from the OpenSSL library. For the AVX2 implementations, we directly use the Intel AES-NI. The $\mathsf{AES}-\mathsf{PRG.Init}$ is the same as PRG.Init in Algorithm 4, and Algorithm 45 shows the $\mathsf{AES}-\mathsf{PRG.Out}$.

Table 3.4: Comparison of the Titanium-CCA benchmark results with and without the AVX2.

Par. Set	Number of cycles (non-AVX2)	Number of cycles (AVX2)
Toy64	KeyGen: 1269090	KeyGen: 703451
	Encrypt: 947906	Encrypt: 564866
	Decrypt: 1107424	Decrypt: 656566
Lite96	KeyGen: 1426439	KeyGen: 773685
	Encrypt: 1234901	Encrypt: 682140
	Decrypt: 1425403	Decrypt: 790026
Std128	KeyGen: 1806119	KeyGen: 934051
	Encrypt: 1446751	Encrypt: 865352
	Decrypt: 1671578	Decrypt: 986905
Med160	KeyGen: 2035675	KeyGen: 1122462
	Encrypt: 1855415	Encrypt: 1042861
	Decrypt: 2109199	Decrypt: 1182880
Hi192	KeyGen: 2122547	KeyGen: 1189978
	Encrypt: 1986198	Encrypt: 1118572
	Decrypt: 2310815	Decrypt: 1303825
Super256	KeyGen: 2829289	KeyGen: 1439023
	Encrypt: 2799390	Encrypt: 1590001
	Decrypt: 3247542	Decrypt: 1811888

Algorithm 45 : $\mathsf{AES-PRG.Out}$

Input: $\mathsf{prgst} \in \mathsf{StSp_{prg}}$ and $\ell \in \mathbb{N}$.
Output: $\mathsf{prgst} \in \mathsf{StSp_{prg}}$ and $\mathsf{out} \in \mathsf{byte}^{\ell}$.

function $\mathsf{AES-PRG.Out}(\mathsf{prgst}, \ell)$
  Let $K \in \mathsf{byte}^{32}$ be the first 32 bytes of prgst.
  Let $c \in \mathbb{Z}_{2^{32}}$ be the last 4 bytes of prgst denoted as a 32-bit unsigned integer in little endian form.
  Let S denote the empty binary string.
  for $i \leq \lceil \ell/16 \rceil$ **do**
    Let $S = S || \mathsf{AES-256-ECB}\left(K, c || 0 \ldots 0 \in \mathsf{byte}^{16}\right)$.
    Let $c = c + 1$.
  end for
  Let out be the first ℓ bytes of S.
  Let $\mathsf{prgst} = (K, c) \in \mathsf{StSp_{prg}}$.
end function

The Table 3.5 shows both the non-AVX2 and the AVX2 benchmark results for each parameter set of the Titanium-CPA schemes. The Table 3.6 shows both the non-AVX2 and the AVX2 benchmark results for each parameter set of the Titanium-CCA schemes. From both tables, we can find that the AES variants improve the efficiency by 10-15% for the non-AVX2 implementations, and 30-40% for the AVX2 implementations compared to the original KMAC variants.

3.4.6 Open Quantum Safe Integration

The Open Quantum Safe library (liboqs) contains several quantum-resistant key exchange protocols or KEMs, and it can be integrated with the popular OpenSSL library for a wide range of applications [MS]. Our Titanium-CCA optimised implementations (non-AVX2) are already included in the latest

Table 3.5: Comparison of the Titanium-CPA benchmark results with and without the AVX2 using the AES PRG.

Par. Set	Number of cycles (non-AVX2)	Number of cycles (AVX2)
Toy64	KeyGen: 1080756	KeyGen: 426869
	Encrypt: 751720	Encrypt: 315424
	Decrypt: 132875	Decrypt: 54228
Lite96	KeyGen: 1099505	KeyGen: 436896
	Encrypt: 935097	Encrypt: 383282
	Decrypt: 159962	Decrypt: 65568
Std128	KeyGen: 1396315	KeyGen: 501206
	Encrypt: 1079362	Encrypt: 473929
	Decrypt: 193412	Decrypt: 73888
Med160	KeyGen: 1612734	KeyGen: 639902
	Encrypt: 1436597	Encrypt: 590207
	Decrypt: 221925	Decrypt: 91578
Hi192	KeyGen: 1631230	KeyGen: 655928
	Encrypt: 1530741	Encrypt: 632895
	Decrypt: 286266	Decrypt: 119864
Super256	KeyGen: 2185642	KeyGen: 788307
	Encrypt: 2182793	Encrypt: 934890
	Decrypt: 395959	Decrypt: 155303

liboqs implementation (the `nist-branch` on official upstream's github[1]). Thus, our scheme has the flexibility and potential for common use.

The liboqs provides a benchmark script `speed_kem` to measure the average running time and CPU cycles among all KEMs. The Table 3.7 compares the running time of KeyGen, Encrypt and Decrypt between the Titanium-CCA with other lattice-based KEMs in the library on our benchmark platform [2]. The Table 3.8 compares the CPU cycles of KeyGen, Encrypt and Decrypt between the Titanium-CCA with other lattice-based KEMs in the library. Note that the `-DAVX2` switch is globally enabled by default during compiling, which might enable the AVX2 optimisations for some schemes.

3.5 Constant Time Implementation

The lazy reduction algorithm [Har14] discussed in Sec. 3.4.2 also contributes to the constant time implementation, since it removes the conditional adjustment of the result from the original Montgomery or Barrett reduction. In addition, we adapt the constant time implementation techniques discussed by [ALM+16], particularly the constant-time comparison and the constant-time branching. We apply these techniques to the following steps in our implementation:

- Constant time comparison [Sei18]: Given unsigned 32-bit integers $x, y < 2^{31}$, to check whether $x < y$, let c be the computation result of:

$$c = (x - y) \ \textbf{shr} \ 31.$$

  where **shr** is the bit right shift operation. We get $c = 1$ if $x < y$, and $c = 0$ otherwise.

- SampUnifZq in Algorithm 8: We generate more randomness at the beginning of SampUnifZq to make sure the possibility of rerunning the PRG is negligible, as discussed in Chapter 7.

[1] `https://github.com/open-quantum-safe/liboqs/tree/nist-branch`

[2] The compiler is gcc 7.4.1 in this benchmark

Table 3.6: Comparison of the Titanium-CCA benchmark results with and without the AVX2 using the AES PRG.

Par. Set	Number of cycles (non-AVX2)	Number of cycles (AVX2)
Toy64	KeyGen: 1085654	KeyGen: 430751
	Encrypt: 801691	Encrypt: 347802
	Decrypt: 937381	Decrypt: 400762
Lite96	KeyGen: 1233880	KeyGen: 488751
	Encrypt: 1085114	Encrypt: 458299
	Decrypt: 1248694	Decrypt: 522409
Std128	KeyGen: 1553925	KeyGen: 556385
	Encrypt: 1248256	Encrypt: 555743
	Decrypt: 1439221	Decrypt: 627455
Med160	KeyGen: 1775307	KeyGen: 704882
	Encrypt: 1638219	Encrypt: 695653
	Decrypt: 1863957	Decrypt: 785317
Hi192	KeyGen: 1843318	KeyGen: 740083
	Encrypt: 1760607	Encrypt: 755662
	Decrypt: 2041340	Decrypt: 876249
Super256	KeyGen: 2493796	KeyGen: 895571
	Encrypt: 2521849	Encrypt: 1123224
	Decrypt: 2919891	Decrypt: 1276374

We remark that the rejection sampling loop of SampUnifZq does not run in constant time due to the variable rejection pattern of x's. However, as discussed in the previous Chapter, the accepted x's output by this algorithm are uniform in $\mathbb{Z}_q$ and independent, even conditioned on the rejection pattern of x's (assuming the indistinguishability from uniform of the x's supplied as input to SampUnifZq by the PRG). Thus, this variability in the run-time of SampUnifZq does not constitute a timing side-channel attack issue.

- Sampr in Algorithm 10: Since we only get a random bit $b \in \{0, 1\}$ when applying the sign bit to the sample s, to map b to a signed byte $b' \in \{1, -1\}$ and compute $s' = s \cdot b'$, we apply the constant-time branching technique to compute:

$$b' = ((-b) \textbf{ and } 0\text{x}\mathsf{FE}) \textbf{ xor } 0\text{x}01,$$

where **and** is the bitwise and operation. Note that we store b as a single byte and $-b = -1 =$ 0xFF when $b = 1$.

- Titanium-CCA.Decrypt in Algorithm 31: The following algorithm 46 makes the comparison between $\mathsf{ct}' = (\mathsf{ct.cpa}', \mathsf{dcca}')$ and $\mathsf{ct} = (\mathsf{ct.cpa}, \mathsf{dcca})$, and set $h = \mathsf{m}'$ if $\mathsf{ct}' = \mathsf{ct}$, or $h = \mathsf{rdec}$ with $h \in \mathsf{byte}^{32}$ otherwise:

Table 3.7: Comparison of the Titanium-CCA running time (μs) with other lattice-based KEMs by liboqs

Scheme	KeyGen	Encrypt	Decrypt
FrodoKEM-640-AES	11010.355	11158.576	11177.721
FrodoKEM-640-cSHAKE	2005.305	2199.089	2219.091
FrodoKEM-976-AES	25487.475	25768.214	25828.094
FrodoKEM-976-cSHAKE	4295.913	4668.652	4697.009
Kyber512	37.154	52.704	63.600
Kyber768	57.939	76.178	89.680
Kyber1024	84.988	107.512	124.744
Lima-2p-1024-CCA-KEM	186.823	127.755	158.594
Lima-2p-2048-CCA-KEM	380.583	264.318	327.173
Lima-sp-1018-CCA-KEM	389.483	325.818	421.370
Lima-sp-1306-CCA-KEM	707.170	622.606	810.820
Lima-sp-1822-CCA-KEM	778.417	659.875	852.131
Lima-sp-2062-CCA-KEM	1378.093	1241.754	1621.791
NewHope-512-CCA-KEM	31.460	46.843	53.841
NewHope-1024-CCA-KEM	63.173	94.966	111.290
LightSaber-KEM	91.401	182.446	270.815
Saber-KEM	198.051	335.426	468.913
FireSaber-KEM	345.574	528.095	704.756
Titanium-CCA Std128	471.134	364.518	422.615
Titanium-CCA Med160	525.844	467.058	534.019
Titanium-CCA Hi192	551.760	502.875	588.879
Titanium-CCA Super256	757.930	703.959	821.646

Algorithm 46 : Compare ct' and ct and set h in constant time

Input: $\mathsf{ct}', \mathsf{ct}$.
Output: $h \in \{\mathsf{m}', \mathsf{rdec}\}$.

function $\mathsf{Compct}(\mathsf{ct}', \mathsf{ct})$
  Let unsigned 32-bit integer $c = 0$.
  for $i <$ lenct **do**
    Let $c = c$ **or** $(\mathsf{ct}'_i$ **xor** $\mathsf{ct}_i)$.
  end for
  Let $c = -((-c)$ **shr** $31)$.
  Let $h = \mathsf{m}'$ and $h \in \mathsf{byte}^{32}$.
  for $i < 32$ **do**
    Let $h_i = h_i$ **xor** $(c$ **and** $(\mathsf{rdec}_i$ **xor** $h_i))$.
  end for
end function

3.6 Known Answer Test (KAT) values

In Table 3.9, we provide the KAT folders of the Titanium-CPA and Titanium-CCA for the corresponding parameter sets.

Table 3.8: Comparison of the Titanium-CCA's number of CPU cycles with other lattice-based KEMs by liboqs

Scheme	KeyGen	Encrypt	Decrypt
FrodoKEM-640-AES	46243407	46865825	46946249
FrodoKEM-640-cSHAKE	8422190	9236035	9320081
FrodoKEM-976-AES	107047253	108226283	108477770
FrodoKEM-976-cSHAKE	18042738	19608210	19727287
Kyber512	155959	221257	267033
Kyber768	243256	319859	376572
Kyber1024	356866	451434	523831
Lima-2p-1024-CCA-KEM	784550	536430	666012
Lima-2p-2048-CCA-KEM	1598275	1109973	1374022
Lima-sp-1018-CCA-KEM	1635689	1368263	1769662
Lima-sp-1306-CCA-KEM	2969957	2614779	3405345
Lima-sp-1822-CCA-KEM	3269172	2771270	3578857
Lima-sp-2062-CCA-KEM	5787777	5215205	6811435
NewHope-512-CCA-KEM	132040	196627	226031
NewHope-1024-CCA-KEM	265240	398728	467292
LightSaber-KEM	383796	766207	1137351
Saber-KEM	831724	1408705	1969347
FireSaber-KEM	1451334	2217878	2959844
Titanium-CCA Std128	1978670	1530891	1774880
Titanium-CCA Med160	2208445	1961495	2242735
Titanium-CCA Hi192	2317277	2111889	2473171
Titanium-CCA Super256	3183179	2956480	3450786

Table 3.9: KAT folders for the Titanium-CPA and Titanium-CPA.

Par. Set	Folder for Titanium-CPA	Folder for Titanium-CCA
Toy64	`Titanium_CPA_toy`	`Titanium_CCA_toy`
Lite96	`Titanium_CPA_lite`	`Titanium_CCA_lite`
Std128	`Titanium_CPA_std`	`Titanium_CCA_std`
Med160	`Titanium_CPA_med`	`Titanium_CCA_med`
Hi192	`Titanium_CPA_hi`	`Titanium_CCA_hi`
Super256	`Titanium_CPA_super`	`Titanium_CCA_super`

Chapter 4

Simplified Algorithms

In this Chapter, we simplify our Titanium-CPA and Titanium-CCA algorithms into more algebraic formats suitable for correctness and security analysis.

4.1 Simplified Titanium-CPA Algorithms: Titanium-CPA-S

In this section the Titanium-CPA algorithms are simplified to have only work with polynomials. Let $\chi_e = (\mathsf{BinDiff}(\eta)^{d+k})^t$ as in (2.6) and $\chi_r = \mathsf{ZeIntU}(B_1)^{N_{\mathsf{dec1}}} \times \mathsf{ZeIntU}(B_2)^{N_{\mathsf{dec}}-N_{\mathsf{dec1}}}$ similar to (2.7). Let we denote the sampling of a random element x from distribution $\mathcal{X}$ by $x \hookleftarrow \mathcal{X}$. Then to encrypt a message $m \in \mathbb{Z}_p^{<d}[x]$, we have:

Algorithm 47 : Titanium-CPA-S.KeyGen

Input: 1^λ.
Output: pk and sk.
1: **function** KeyGen(1^λ)
2: Let $s \hookleftarrow U(\mathbb{Z}_q^{<n+d+k-1}[x])$.
3: Let $(\bar{a_1}, \ldots, \bar{a_t}) \hookleftarrow U(\mathbb{Z}_q^{<n}[x])^t$.
4: Let $(e_1, \ldots, e_t) \hookleftarrow \chi_e \in (\mathbb{Z}_q^{<d+k}[x])^t$.
5: **for** $i \leq t$ **do**
6: Let $b_i = \mathsf{Rev}(\bar{a_i}) \odot_{d+k} s + e_i \in \mathbb{Z}_q^{<d+k}[x]$.
7: **end for**
8: Let $\mathsf{pk} = ((\bar{a_1}, \ldots, \bar{a_t}), (b_1, \ldots, b_t))$ and $\mathsf{sk} = s$.
9: **end function**

Algorithm 48 : Titanium-CPA-S.Encrypt

Input: pk and m.
Output: $\mathsf{ct} = (c_1', c_2')$.
1: **function** Encrypt(pk, m)
2: Let $(r_1, \ldots, r_t) \hookleftarrow \chi_r \in (\mathbb{Z}_q^{<k+1}[x])^t$.
3: Let $c_1' = \sum_{i=1}^t r_i \cdot \bar{a_i}$
4: Let $c_2' = \sum_{i=1}^t \mathsf{Rev}(r_i) \odot_d b_i + \lfloor q/p \rfloor \cdot m \in \mathbb{Z}_q^{<d}[x]$.
5: **end function**

Algorithm 49 : Titanium-CPA-S.Decrypt

Input: sk and ct.
Output: m'.
1: **function** Decrypt(sk, ct)
2: Let $c' = c'_2 - \mathsf{Rev}(c'_1) \odot_d s \in \mathbb{Z}_q^{<d}[x]$.
3: Let $m' = \mathsf{Round}(\lfloor q/p \rfloor, c') \in \mathbb{Z}_p^{<d}[x]$.
4: **end function**

4.2 Simplified Titanium-CCA Algorithms: Titanium-CCA-S

Algorithm 50 : Titanium-CCA-S.KeyGen

Input: 1^λ.
Output: pk and sk.
1: **function** KeyGen(1^λ)
2: Sample $(\mathsf{sk.cpa}, \mathsf{pk.cpa}) = \mathsf{Titanium\text{-}CPA.KeyGen}(1^\lambda)$.
3: Sample $\mathsf{rdec} \hookleftarrow U(\mathsf{byte}^{32})$.
4: let $\mathsf{sk} = (\mathsf{sk.cpa}, \mathsf{rdec}, \mathsf{pk.cpa})$ and $\mathsf{pk} = \mathsf{pk.cpa}$.
5: **end function**

Algorithm 51 : Titanium-CCA-S.Encrypt

Input: pk.
Output: ct and ss.
1: **function** Encrypt(pk)
2: Sample $\mathsf{m} \hookleftarrow U(\mathsf{byte}^{32})$.
3: Let $(\mathsf{seedenc.cpa}, \mathsf{dcca}) = \mathsf{G}(\mathsf{m}) \in \mathsf{byte}^{32} \times \mathsf{byte}^{32}$.
4: Let $\mathsf{ct.cpa} = \mathsf{Titanium\text{-}CPA.Encrypt}(\mathsf{pk}, \mathsf{m})$.
5: Let $\mathsf{ct} = (\mathsf{ct.cpa}, \mathsf{dcca})$
6: Let $\mathsf{ss} = \mathsf{H}(\mathsf{m}, \mathsf{ct}) \in \mathsf{byte}^{32}$.
7: **end function**

Algorithm 52 : Titanium-CCA-S.Decrypt

Input: sk and ct.
Output: ss.
1: **function** Decrypt(sk, ct)
2: Let $\mathsf{m}' = \mathsf{Titanium\text{-}CPA.Decrypt}(\mathsf{sk.cpa}, \mathsf{ct.cpa})$.
3: Let $(\mathsf{seedenc.cpa}', \mathsf{dcca}') = \mathsf{G}(\mathsf{m}') \in \mathsf{byte}^{32} \times \mathsf{byte}^{32}$.
4: Let $\mathsf{ct.cpa}' = \mathsf{Titanium\text{-}CPA.Encrypt}(\mathsf{pk}, \mathsf{m}')$.
5: **if** $(\mathsf{ct.cpa}', \mathsf{dcca}') = (\mathsf{ct.cpa}, \mathsf{dcca})$ **then**
6: Let $\mathsf{ss} = \mathsf{H}(\mathsf{m}', \mathsf{ct})$.
7: **else**
8: Let $\mathsf{ss} = \mathsf{H}(\mathsf{rdec}, \mathsf{ct})$.
9: **end if**
10: **end function**

Chapter 5

Correctness Analysis

This Chapter contains the proofs of correctness for our utility functions, distribution sampling functions, and explains our method of computing a numerical provable upper bound on the error probability of decryption, that is also used in our IND-CCA security proof.

5.1 Utility and sampling functions correctness

5.1.1 Utility functions

We first note the following Lemma:

Lemma 5.1.1. *With appropriate domains and ranges, the following equalities are outstanding:*

- $\mathsf{Perm}_{\mathsf{d}_j}(\mathsf{InvPerm}_{\mathsf{d}_j}(\mathbf{a})) = \mathsf{InvPerm}_{\mathsf{d}_j}(\mathsf{Perm}_{\mathsf{d}_j}(\mathbf{a})) = \mathbf{a}$, *for* $j = 1, 2, 3$, *and* $\mathbf{a} \in \mathbb{Z}_q^\ell$,
- $\mathsf{NTT}_{\mathsf{d}_j}(\omega_j, \mathsf{NTT}^{-1}_{\mathsf{d}_j}(\omega_j, \mathbf{z})) = \mathsf{NTT}^{-1}_{\mathsf{d}_j}(\omega_j, \mathsf{NTT}_{\mathsf{d}_j}(\omega_j, \mathbf{z}))$, *for* $j = 1, 2, 3$, *and* $\mathbf{z} \in \mathbb{Z}_q^d$,
- $\mathsf{Trunc}(n_0, \mathsf{Zpad}(n_0, \mathbf{v})) = \mathbf{v}$, *for an integer* n_0, *and a vector* $\mathbf{v}$ *of length* $\ell < n_0$,
- $\mathsf{Decodem}(\mathsf{Encodem}(\mathsf{m})) = \mathsf{m}$, *for* $\mathsf{m} \in \mathsf{byte}^{32}$
- $\mathsf{Decodpk}(\mathsf{Encodepk}(\mathsf{pk})) = \mathsf{pk}$,
- $\mathsf{Decodect}(\mathsf{Encodect}(\mathsf{ct}) = \mathsf{ct}$.

Proof. We prove each item independently:

- Let $k_i = (i \bmod \delta_2) \cdot \delta_1 + \lfloor i/\delta_2 \rfloor$, and $\mathsf{d}_j = \delta_1 \cdot \delta_2$, then we have

$$\begin{aligned} \mathsf{InvPerm}_{\mathsf{d}_j}(\mathsf{Perm}_{\mathsf{d}_j}(\mathbf{a})) &= \mathsf{InvPerm}_{\mathsf{d}_j}(\mathbf{a}_{k_i}) \\ &= \mathbf{a}_{(k_i \bmod \delta_1)\cdot\delta_2 + \lfloor k_i/\delta_1 \rfloor}. \end{aligned}$$

  On the other hand, if $i \leq \delta_2$, it is easy to verify that $i' := (k_i \bmod \delta_1) \cdot \delta_2 + \lfloor k_i/\delta_1 \rfloor = i$. If $i > \delta_2$, we write $i = t \cdot \delta_2 + s$, for integers t and $s < D_2$. Rewriting i' in terms of s and t, one gets $i' = (t \bmod \delta_1) \cdot \delta_2 + s + \lfloor t/\delta_1 \rfloor$. Note that $t > \delta_1$ contradicts $i < \delta_1 \cdot \delta_2$, and hence $t \leq \delta_1$, which means that $i' = i$. The inverse is also true, which completes the proof.

- Let us calculate $\mathsf{NTT}_{\mathsf{d}_j}(\omega_{\mathsf{d}_j}, \mathsf{NTT}^{-1}_{\mathsf{d}_j}(\omega_{\mathsf{d}_j}, \mathbf{z}))$. Based on the definition of NTT^{-1}, we have that $\mathsf{NTT}^{-1}_{\mathsf{d}_j}(\omega_{\mathsf{d}_j}, \mathbf{z}) = \mathbf{y}$, with $y_i = \mathsf{d}_j^{-1} \cdot \sum_{\ell_0=0}^{d-1} \omega_{\mathsf{d}_j}^{-i\cdot\ell_0} z_{\ell_0}$. The i-th component of $\mathsf{NTT}_{\mathsf{d}_j}(\omega_{\mathsf{d}_j}, \mathbf{y})$ is

$$\sum_{\ell=0}^{d-1} \omega_{\mathsf{d}_j}^{i\cdot\ell} \left(\mathsf{d}_j^{-1} \cdot \sum_{\ell_0=0}^{d-1} \omega_{\mathsf{d}_j}^{-\ell\cdot\ell_0} z_{\ell_0} \right) = \mathsf{d}_j^{-1} \cdot \sum_{\ell_0=0}^{d-1} z_{\ell_0} \cdot \left(\sum_{\ell=0}^{d-1} \omega_{\mathsf{d}_j}^{\ell\cdot(i-\ell_0)} \right). \tag{5.1}$$

  When $i - \ell_0 \neq 0$, we have that $\sum_{\ell=0}^{d-1} \omega_{\mathsf{d}_j}^{\ell\cdot(i-\ell_0)} = 0$ and with $i - \ell_0 = 0$, the latter summation is equal to d_j, which means that (5.1) equals z_i.

- The proof is straightforward by looking at Algorithms 13 and 14.
- Note that the result of $\mathsf{Encodem}(\mathsf{m})$ in Algorithm 27 is a polynomial in $\mathbb{Z}_2^{<d}[x]$, in which the coefficients of $x^{(i-1)\cdot 8}$ up to $x^{(i-1)\cdot 8+j}$, for $1 \leq j \leq 7$, are the bits of the i-th byte of m, for $1 \leq i \leq 32$. The $\mathsf{Decodem}$ function in Algorithm 28 takes this polynomial and re-construct blocks of 8-bits one by one.
- Looking at Algorithms 25 and 26, it is enough to show that $\mathsf{PackVec}$ and $\mathsf{UnPackVec}$ in Algorithms 18 and 21 and hence $\mathsf{PackVec}\ell$ and $\mathsf{UnPackVec}\ell$ in Algorithms 17 and 20 are the compositional inverses of each other. Following the steps of Algorithm 17, we note that bitst is just the component-wise bit representation of $\mathbf{b}$ and bitst' is its zero-padded version to dimension $L+L'$, for $L' = -L \bmod 8$. Dividing this vector into chunks of 8 bits and putting each into a byte would give us the vector $\mathbf{v}$. On the other hand, to compute $\mathsf{UnPackVec}\ell(\mathsf{PackVec}\ell(\mathbf{b})) = \mathsf{UnPackVec}\ell(\mathbf{v})$, we first unpack $\mathbf{v}$ into bytes and then bits. Finally, we remove the padded zeros to get to $\mathbf{b}$. This completes the proof.
- Looking at Algorithms 23 and 24, it is enough to show that $\mathsf{UnPackVec}(\mathsf{PackVec}(\hat{\mathbf{c}}_1)) = \hat{\mathbf{c}}_1$ for $\hat{\mathbf{c}}_1 \in \mathbb{Z}_q^{\mathsf{d}_2}$, $\mathsf{UnPackVecc}(\mathsf{PackVecc}(c_2)) = c_2$, and $\mathsf{VecToPol}(\mathsf{PolToVec}(c_2)) = c_2$, for $c_2 \in \mathbb{Z}_q^{<d}[x]$. The first and the third statements are proved in previous parts, we prove the second one here. Note that in Algorithms 19 and 22, we use division and multiplication by 2^{cmp} and $\mathsf{PackVec}\ell$ and $\mathsf{UnPackVec}\ell$ with appropriate sizes, respectively. Hence, the correctness of these algorithms can easily be followed from previous proofs.

□

Now we describe the reason why Titanium-CPA and Titanium-CPA-S are basically the same. The decryption algorithm for Titanium-CPA computes

$$c_1 = \mathsf{Trunc}(n+k, \mathsf{VecToPol}(\mathsf{NTT}_{\mathsf{d}_2}^{-1}(\omega_2, \mathsf{InvPerm}_{\mathsf{d}_2}(\widehat{\mathbf{c}}_1))))$$

with $\widehat{\mathbf{c}}_1$ being the first part of $\mathsf{Decodect}(\mathsf{ct})$. On the other hand, $\mathsf{ct} = \mathsf{Encodect}(\widehat{\mathbf{c}}_1, c_2)$ with $\widehat{\mathbf{c}}_1 = \mathsf{Perm}_{\mathsf{d}_2}(\mathsf{NTT}_{\mathsf{d}_2}(\omega_2, \mathsf{PolToVec}(\mathsf{Zpad}(\mathsf{d}_2, c_1'))))$ from the $\mathsf{Encrypt}$ algorithm. Using Lemma 5.1.1, c_1' computed in $\mathsf{Decrypt}$ algorithm is equal to c_1' from the $\mathsf{Encrypt}$ algorithm. On the other hand s computed in $\mathsf{Decrypt}$ is equal to s derived in KeyGen, since both are calculated as

$$s = \mathsf{Trunc}(n+k+d-1, \mathsf{VecToPol}(\mathsf{d}_3^{-1} \cdot \mathsf{NTT}_{\mathsf{d}_3}(\omega_3, \mathsf{InvPerm}_{\mathsf{d}_3}(\widehat{\mathbf{s}})))),$$

with $\widehat{\mathbf{s}}$ being the output of Samps with the same input seedsk.

5.1.2 Sampler algorithms output correct distributions

In this Subsection, we show that algorithms Samps, Sampa, Sampe, and Sampr (with overwhelming probability) produce distributions $U(\mathbb{Z}_q^{<n+d+k-1}[x])$, $U(\mathbb{Z}_q^{<n}[x])$, χ_e (as in (2.6)), and χ_r (as in (2.7)), respectively.

Lemma 5.1.2. *Let* $\mathsf{prgst} \in \mathsf{byte}^{32}$ *and* ℓ *be an integer, and* $(\mathsf{prgst}, \mathsf{out}) \leftarrow \mathsf{PRG.Out}(\mathsf{prgst}, \ell)$. *If* out *is truly random, then*

- *the second part of the output of* $\mathsf{SampUniZq}$ *in Algorithm 8 is distributed as* $U(\mathbb{Z}_q)^\ell$,
- *the output of* Sampe *in Algorithm 9 is distributed as* $(\chi_e)^t$,
- *the output of* Sampr *is in Algorithm 10 distributed as* χ_r.

Proof. We provide the proof of each part independently:

- Note that we generate bytpm bytes per coordinate for uniform sampler over $\mathbb{Z}_q$ and let $\mathsf{ZqRej} = 2^{8\cdot\mathsf{bytpm}} - (2^{8\cdot\mathsf{bytpm}} \bmod q)$, which is the closest multiple of q to $2^{8\cdot\mathsf{bytpm}}$. We employ rejection sampling with the success rate $r = \mathsf{ZqRej}/2^{8\cdot\mathsf{bytpm}}$, and reduce the samples modulo q. On the other hand, to generate ℓ samples, the sampler requires more random bytes than $\mathsf{bytpm}\cdot\ell$ when $r < 1$. We generate slightly more randomness than $\mathsf{bytpm}\cdot\ell$ to avoid the overhead of rerunning the PRG. Let Y be the random variable denoting the number of bytpm groups of bytes tested to generate one $\mathbb{Z}_q$ element. The rejection sampling follows a geometric distribution with the bias r, that is $\mathsf{Geom}(r)$, with mean $1/r$, standard deviation $(1-r)/r^2$, and probability density function $f_Y(y) = (1-r)^y r$. As we want to generate ℓ independent $\mathbb{Z}_q$ elements, we have to choose $\mathsf{bytpc}_1 = \mathsf{bytpc}/\mathsf{bytpm}$ such that

$$\Pr\left[\mathsf{PRG} \text{ is called more than once in Algorithm 8}\right] = \Pr\left[\sum_{i=1}^{\ell} y_i \geq \mathsf{bytpc}_1\right] \leq 2^{-\lambda_C}, \tag{5.2}$$

with y_i distributed as $\mathsf{Geom}(r)$, for $1 \leq i \leq \ell$, and λ_C being the classical security parameter of the relevant category of interest, to ensure correctness of our sampler and claim constant time encryption independent of randomness (useful for side-channel attacks). This probability is equal to make the possibility of running out of the randomness from the first PRG call in Algorithm 8 negligible, say about $2^{-\lambda_C}$. It is easy to see that the mean of $\sum_{i=1}^{\ell} y_i$ is ℓ/r, we now begin with the expansion of the LHS of (5.2) based on Hoeffding approach [Rig15]:

$$\begin{aligned}
\Pr\left[\sum_{i=1}^{\ell} y_i \geq \mathsf{bytpc}\right] &= \Pr\left[\exp\left(s\cdot\sum_{i=1}^{\ell} y_i\right) \geq \exp\left(\mathsf{bytpc}_1\right)\right] \\
&\leq \frac{\mathbb{E}\left(\exp\left(s\cdot\sum_{i=1}^{\ell} y_i\right)\right)}{\exp(s\cdot\mathsf{bytpc}_1)} \qquad (5.3)\\
&= \frac{\mathbb{E}\left(\prod_{i=1}^{\ell}\exp(s\cdot y_i)\right)}{\exp(s\cdot\mathsf{bytpc}_1)} \\
&= \frac{\prod_{i=1}^{\ell}\mathbb{E}(\exp(s\cdot y_i))}{\exp(s\cdot\mathsf{bytpc}_1)}, \qquad (5.4)
\end{aligned}$$

where (5.3) is obtained using Markov inequality [BLM13]. The moment generating function of a geometric distribution $\mathsf{Geom}(r)$ can be derived as $\frac{r\exp(s)}{1-(1-r)\exp(s)}$. Plugging this formula into (5.4) and defining

$$g(s) := \frac{r^{\ell}\exp(s\cdot\ell)}{(1-(1-r)\exp(s))^{\ell}\exp(s\cdot\mathsf{bytpc}_1)}.$$

The defined function g is minimized at s^*, with

$$\exp(s^*) = \frac{\mathsf{bytpc}_1 - \ell}{\mathsf{bytpc}_1\cdot(1-r)}. \tag{5.5}$$

We now choose bytpc_1^* such that

$$g(s^*) \leq 2^{-\lambda_C}. \tag{5.6}$$

We also estimate (5.2) using central limit theorem (CLT). Using CLT, $\sum_{i=1}^{\ell} y_i$ can be approximated by a Gaussian distribution with the same mean and variance as $\sum_{i=1}^{\ell} y_i$. It is easy to see that the mean of $\sum_{i=1}^{\ell} y_i$ is $\mu = \ell/r$ and its variance is $\sigma^2 = \ell(1-r)/r^2$. Therefore,

$$\begin{aligned}
\Pr\left[\sum_{i=1}^{\ell} y_i \geq \mathsf{bytpc}_1\right] &\approx \Pr_{z\hookleftarrow\mathsf{Gauss}(\mu,\sigma^2)}\left[z \geq \mathsf{bytpc}_1\right] \qquad (5.7)\\
&= \Pr_{z\hookleftarrow\mathsf{Gauss}(0,1)}\left[z \geq (\mathsf{bytpc}_1 - \mu)/\sigma\right], \qquad (5.8)
\end{aligned}$$

where (5.7) is obtained using CLT and (5.8) is the Q-function of a standard Gaussian distribution that can be computed numerically. Letting ℓ be equal to d_3 and n, we choose $\mathsf{bytpcs}_1^{\mathsf{clt}}$ and $\mathsf{bytpca}_1^{\mathsf{clt}}$ such that the RHS of (5.8) is less that $2^{-\lambda_C}$, for different classical security requirements.

The overhead of such estimations are compared to the relevant means $\mu_s = \mathsf{d}_3/r$ and $\mu_a = n/r$ are called $\mathsf{ov}_s^{\mathsf{clt}}$ and $\mathsf{ov}_a^{\mathsf{clt}}$, respectively. The results of the above optimisation are given in Tables 5.1-5.2. Note that bytpcs_1 and bytpca_1 are calculated as $\mathsf{bytpcs}/\mathsf{bytpm}$ and $\mathsf{bytpca}/\mathsf{bytpm}$, respectively, with bytpca, bytpcs and bytpm given in Table 2.10, while bytpca_1^* and bytpcs_1^* are the smallest integers chosen to satisfy (5.5) and (5.6) simultaneously, for $\ell = n$ and $\ell = \mathsf{d}_3$, respectively. Furthermore, the overheads ov_a^* and ov_s^* are calculated (in %) as $(\mathsf{bytpca}_1^* - \mu_a)/\mu_a$ and $(\mathsf{bytpcs}_1^* - \mu_s)/\mu_s$, with $\mu_a = n/r$ and $\mu_s = \mathsf{d}_3/r$. Numerical evaluations given at Tables 5.1 and 5.2 justify the selections of bytpc for different security categories in Tables 2.5 and 2.10. In particular, we have that $\mathsf{bytpcs} = \mathsf{bytpcs}_1^* \cdot \mathsf{bytpm}$ and $\mathsf{bytpca} = \mathsf{bytpca}_1^* \cdot \mathsf{bytpm}$.

Table 5.1: The sampling parameters for uniform sampler over $\mathbb{Z}_q$ in Titanium-CPA. The parameters are divided into two groups. The top half are related to Samps in Algorithm 6 and the bottom half are related to sampa in Algorithm 7.

Parameter	Toy64	Lite96	Std128	Med160	Hi192	Super256
λ_C	79	111	143	175	207	272
d_3	1280	1536	1792	2048	2560	3328
μ_s	1293.33	1540.58	1792.41	2075.09	2647.91	3328.80
bytpcs_1^*	1348	1588	1823	2193	2857	3386
ov_s^* (in %)	4.2	3.0	1.7	5.6	7.8	1.7
$\mathsf{bytpcs}_1^{\mathsf{clt}}$	1330	1566	1801	2155	2807	3346
$\mathsf{ov}_s^{\mathsf{clt}}$ (in %)	2.8	1.6	0.4	3.8	6.0	0.5
n	684	800	1024	1280	1536	2048
bytpca_1^*	735	841	1051	1397	1760	2100
μ_a	691.12	802.39	1024.23	1296.93	1588.74	2048.49
ov_a^* (in %)	6.3	4.8	2.6	7.7	10.7	2.5
$\mathsf{bytpca}_1^{\mathsf{clt}}$	769	821	1030	1360	1712	2062
$\mathsf{ov}_a^{\mathsf{clt}}$ (in %)	11.2	2.3	0.5	4.8	7.7	0.6

- Since $\mathbf{r}$ is truly random, each $\mathbf{rr}$ is random too. Note that there are $(d + k - 1) \cdot \mathsf{Zebytes}$ bytes available in $\mathbf{rr}$. In Algorithm 9, we just divide each Zebytes bytes of $\mathbf{rr}$ into two chunks of $(8 \cdot \mathsf{Zebytes})/2$ bits and let the difference of these two halves as the coefficient of the corresponding polynomial. Since $8 \cdot \mathsf{Zebytes} \geq \eta$, each coefficient of each e_i, for $1 \leq i \leq t$, is distributed as a binomial difference distribution BinDiff with parameter η.

- First note that we take $(k + 1) \cdot t$ elements in $\mathbb{Z}_q$ to construct polynomials $r_j \in \mathbb{Z}_q^{<k+1}[x]$, for $1 \leq j \leq t$. Given N_{dec1} and N_{dec}, with $N_{\mathsf{dec1}} < N_{\mathsf{dec}}$, we write $N_{\mathsf{dec1}} = (k+1) \cdot \mathsf{Zbt} + \mathsf{ZbRem}$, for $\mathsf{ZbRem} \leq k$. Since $N_{\mathsf{dec}} = (k+1) \cdot t$, we have that $\mathsf{Zbt} < t$. Following the steps of Algorithm 10, each block of $(k + 1) \cdot \mathsf{Zbbytes}$ bytes are used to generate a random polynomial in $\mathbb{Z}_q^{<k+1}[x]$. To be more precise, each block of Zbbytes bytes will be employed to generate a coefficient in either $\mathsf{ZeIntU}(B_1)$ or $\mathsf{ZeIntU}(B_2)$ depending on the index of the consecutive blocks arranged in $\mathbf{r}$. The block with index $\mathsf{Zbt} + 1$ is divided into two sub-blocks, the first ZbRem blocks will be used to generate elements in $\mathsf{ZeIntU}(B_1)$ and the rest are used to generate random elements in $\mathsf{ZeIntU}(B_2)$. The final $N_{\mathsf{dec}}/8$ bytes are used to generate random signs for each coefficient in each random polynomial. Since $\mathbf{r}$ is random, all blocks in $\mathbf{rr}$, x, $x \bmod 2^{b_1}$, $x \bmod 2^{b_2}$, and s_j's are random too. Hence the output of Sampr is distributed as $\mathsf{ZeIntU}(B_1)^{N_{\mathsf{dec1}}} \times \mathsf{ZeIntU}(B_2)^{N_{dec} - N_{\mathsf{dec1}}}$.

□

Table 5.2: The sampling parameters for uniform sampler over $\mathbb{Z}_q$ in Titanium-CCA. The parameters are divided into two groups. The top half are related to Samps in Algorithm 6 and the bottom half are related to sampa in Algorithm 7.

Parameter	Toy64	Lite96	Std128	Med160	Hi192	Super256
λ_C	79	111	143	175	207	272
d_3	1280	1536	1792	2048	2560	3328
μ_s	1302.57	1542.71	1802.82	2048.49	2610.83	3328.80
bytpcs_1^*	1370	1596	1878	2085	2778	3386
ov_s^* (in %)	5.1	3.4	4.1	1.7	6.4	1.7
$\mathsf{bytpcs}_1^{\mathsf{clt}}$	1351	1574	1848	2059	2731	3346
$\mathsf{ov}_s^{\mathsf{clt}}$ (in %)	3.7	2.0	2.5	0.5	4.6	.05
n	684	800	1024	1280	1536	2048
bytpca_1^*	750	847	1092	1314	1705	2100
μ_a	696.06	803.49	1030.18	1280.30	1566.49	2048.49
ov_a^* (in %)	7.7	5.4	6.0	2.6	8.8	2.5
$\mathsf{bytpca}_1^{\mathsf{clt}}$	769	826	1064	1288	1659	2062
$\mathsf{ov}_a^{\mathsf{clt}}$ (in %)	10.4	2.8	3.2	0.6	5.9	0.6

Corollary 5.1.1. *The output of*

- Samps *in Algorithm 6 is distributed as* $U(\mathbb{Z}_q^{<n+d+k-1}[x])$,
- Sampa *Algorithm 7 is distributed as* $U(\mathbb{Z}_q^{<n}[x])^t$.

Proof. The proof of the corollary is trivial as we only use SampUnifZq as the uniform sampling algorithm over $\mathbb{Z}_q$ with appropriate dimensions $n+d+k-1$ and n for Samps and Sampa, respectively. Note that SampUnifZq is been used t times in Algorithm 7 and that is the reason why the output of this function belongs to $(\mathbb{Z}_q^{<n}[x])^t$. □

5.1.3 On equivalence of reference and simplified algorithms

With the above described correctness of utility functions, Lemma 5.1.2 and Corollary 5.1.1, it is clear that the correctness of Titanium-CPA is equivalent to the correctness of Titanium-CPA-S and also correctness of Titanium-CCA and Titanium-CCA-S are equivalent (under the random oracle model for XOF). It particularly means that it is enough to first prove/analyse the correctness of Titanium-CPA-S and Titanium-CCA-S. In the following two Sections we analyse each of these schemes independently.

5.2 Concrete correctness conditions of Titanium-CPA-S

We first define the concept of δ-correct Titanium-CPA-S.

Definition 5.2.1. *Our* Titanium-CPA-S *scheme is called* δ*-correct if for any functions* f*, we have*

$$\Pr[\mathsf{Decrypt}(\mathsf{sk},\mathsf{ct}) \neq m \mid (\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{KeyGen}; m = f(\mathsf{pk},\mathsf{sk}); \mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{pk},m)] \leq \delta. \tag{5.9}$$

We remark that the above definition of decryption error probability over the choice of both public key and encryption randomness (for any, even key-dependent, messages), matches the definition of δ-correctness in [HHK17], which allows us to apply the security analysis of [HHK17] to the Fujisaki-Okamoto transform applied to Titanium-CPA, which yields our Titanium-CCA scheme.

From now on, we let p_e denotes the LHS of (5.9). We now analyse the correctness of Titanium-CPA-S. Let us first expand the main operation in decryption of Titanium-CPA-S:

$$\begin{aligned}
c' &= c'_2 - \mathsf{Rev}(c'_1) \odot_d s \\
&= \sum_{i=1}^{t} \mathsf{Rev}(r_i) \odot_d b_i + \lfloor q/p \rfloor \cdot m - \mathsf{Rev}\left(\sum_{i=1}^{t} r_i \cdot \bar{a}_i\right) \odot_d s \\
&= \sum_{i=1}^{t} \mathsf{Rev}(r_i) \odot_d (\mathsf{Rev}(a_i) \odot_{d+k} s + e_i) + \lfloor q/p \rfloor \cdot m - \sum_{i=1}^{t} \mathsf{Rev}(r_i) \cdot \mathsf{Rev}(a_i) \odot_d s \qquad (5.10) \\
&= \sum_{i=1}^{t} \mathsf{Rev}(r_i) \cdot \mathsf{Rev}(a_i) \odot_{d+k} s + \sum_{i=1}^{t} \mathsf{Rev}(r_i) \cdot e_i + \lfloor q/p \rfloor \cdot m - \sum_{i=1}^{t} \mathsf{Rev}(r_i) \cdot \mathsf{Rev}(a_i) \odot_d s \qquad (5.11) \\
&= \lfloor q/p \rfloor \cdot m + \sum_{i=1}^{t} \mathsf{Rev}(r_i) \odot_d e_i \in \mathbb{Z}_q^d[x],
\end{aligned}$$

where (5.10) and (5.11) are obtained using (2.1) and Lemma 2.1.2, respectively. Therefore, in Decryption algorithm of Titanium-CPA-S we have

$$\begin{aligned}
m' &= \mathsf{Round}\left(\lfloor q/p \rfloor, c'\right) \\
&= \mathsf{Round}\left(\lfloor q/p \rfloor, \lfloor q/p \rfloor \cdot m + \sum_{i=1}^{t} \mathsf{Rev}(r_i) \odot_d e_i\right) \\
&= m,
\end{aligned}$$

if $\sum_{i=1}^{t} \mathsf{Rev}(r_i) \odot_d e_i$ computed over $\mathbb{Z}_q^d[x]$ (i.e. with reduction mod q) has coefficients smaller than $\lfloor q/p \rfloor/2$, i.e. if

$$\left\| \sum_{i=1}^{t} \mathsf{Rev}(r_i) \odot_d e_i \right\|_\infty < \lfloor q/p \rfloor/2, \qquad (5.12)$$

with the computations performed over $\mathbb{Z}^d[x]$. We upper bound the probability p_e that (5.12) does not hold, over the choice of the encryption randomness $(r_1, \ldots, r_t)$ from the distribution χ_r defined in (2.7) and the choice of key generation errors $(e_1, \ldots, e_t)$ from the distribution χ_e defined in (2.6).

We recall from (2.7) that χ_r has the form:

$$\chi_r = \mathsf{ZeInt U}(B_1)^{N_{\mathsf{dec1}}} \times \mathsf{ZeInt U}(B_2)^{N_{\mathsf{dec}} - N_{\mathsf{dec1}}},$$

i.e. the first N_{dec1} integer coefficients of the concatenated coefficient vectors of the r_i's are sampled from $\mathsf{ZeInt U}(B_1)$ and the remaining $N_{\mathsf{dec}} - N_{\mathsf{dec1}}$ coefficients sampled from $\mathsf{ZeInt U}(B_2)$. Also, χ_e samples each integer coefficient of $(e_1, \ldots, e_t)$ from the $\mathsf{BinDiff}(\eta)$ distribution. For $i = 1, 2$, let us define the distributions χ_i over $\mathbb{Z}$ as the distribution of the product (over $\mathbb{Z}$) of a sample from $\mathsf{ZeInt U}(B_i)$ and an independent sample from $\mathsf{BinDiff}(\eta)$. Let us define $\bar{r_i}$ as $\mathsf{Rev}(r_i)$. Then we observe that for each $1 \leq i \leq t$, each coefficient of $\bar{r_i} \odot_d e_i$ is an inner product between a row of $\mathsf{Toep}^{d,k}(\bar{r_i})$ and the coefficient vector $\mathbf{e}_i$ of e_i. Therefore, by the independence of the r_i and e_i coefficients, the distribution of each coefficient of $\sum_{i=1}^{t} \bar{r_i} \odot_d e_i$ is the distribution of a sum $\sum_{i=1}^{N_{\mathsf{dec}}} x_i$ of independent random variables x_i, where x_i is sampled from the distribution χ_i with

$$\chi_i := \begin{cases} \chi_1 & 1 \leq i \leq N_{\mathsf{dec1}}, \\ \chi_2 & N_{\mathsf{dec1}} < i \leq N_{\mathsf{dec}}. \end{cases} \qquad (5.13)$$

The probability of error $\bar{p_e}$ for any fixed coordinate of the message can therefore be upper bounded as follows:

$$\bar{p_e} = \Pr\left[\sum_{i=1}^{N} x_i \geq \lfloor q/p \rfloor/2\right],$$

with x_i distributed as in (5.13). Since the x_i's are independent with $\mathbb{E}[x_i] = 0$ for $1 \leq i \leq N_{\mathsf{dec}}$, we have

$$\begin{aligned}
\bar{p_e} &= \Pr\left[\sum_{i=1}^{N_{\mathsf{dec}}} x_i \geq \lfloor q/p \rfloor/2\right] && (5.14)\\
&= \Pr\left[\exp\left(s \cdot \sum_{i=1}^{N_{\mathsf{dec}}} x_i\right) \geq \exp\left(s \cdot \lfloor q/p \rfloor/2\right)\right] && (5.15)\\
&\leq \frac{\mathbb{E}\left[\exp\left(s \cdot \sum_{i=1}^{N_{\mathsf{dec}}} x_i\right)\right]}{\exp\left(s \cdot \lfloor q/p \rfloor/2\right)} && (5.16)\\
&= \frac{\mathbb{E}\left[\prod_{i=1}^{N_{\mathsf{dec}}} \exp\left(s \cdot x_i\right)\right]}{\exp\left(s \cdot \lfloor q/p \rfloor/2\right)} \\
&= \frac{\prod_{i=1}^{N_{\mathsf{dec}}} \mathbb{E}\left[\exp\left(s \cdot x_i\right)\right]}{\exp\left(s \cdot \lfloor q/p \rfloor/2\right)}, && (5.17)
\end{aligned}$$

where (5.15) is true because the mapping $x \mapsto \exp(s \cdot x)$ is monotonically increasing, (5.16) is obtained using Markov inequality [BLM13], and (5.17) is valid due to the fact that x_i's are independent of each other. Let us further define

$$M_{\chi_j}(s) := \mathbb{E}_{x \hookleftarrow \chi_j}[\exp(s \cdot x)],$$

for $j \in \{1, 2\}$. Therefore, (5.17) can be re-written as:

$$\bar{p_e} \leq \frac{\prod_{i=1}^{N_{\mathsf{dec}}} \mathbb{E}\left[\exp\left(s \cdot x_i\right)\right]}{\exp\left(s \cdot \lfloor q/p \rfloor/2\right)} = \frac{M_{\chi_1}^{N_{\mathsf{dec1}}}(s) M_{\chi_2}^{N_{\mathsf{dec}} - N_{\mathsf{dec1}}}(s)}{\exp\left(s \cdot \lfloor q/p \rfloor/2\right)}. \tag{5.18}$$

In order to minimize $\bar{p_e}$, one needs to find s that minimizes (5.18). Letting

$$f(s) := \frac{M_{\chi_1}^{N_{\mathsf{dec1}}}(s) M_{\chi_2}^{N_{\mathsf{dec}} - N_{\mathsf{dec1}}}(s)}{\exp\left(s \cdot \lfloor q/p \rfloor/2\right)},$$

one can differentiate f to find the critical point s^*, such that $f(s^*) = 0$ minimizing the right hand side of (5.18). The well-known bi-section method is now used to numerically evaluate s^* and hence $\bar{p_e}^{\mathsf{Hoeffding}}$ such that $\bar{p_e} \leq \bar{p_e}^{\mathsf{Hoeffding}}$. The above analysis and a union bound over the d coordinates of $\sum_{i=1}^{t} \bar{r_i} \odot_d e_i$ ensures that our Titanium-CPA-S is $p_e^{\mathsf{Hoeffding}} \leq d \cdot \bar{p_e}^{\mathsf{Hoeffding}}$-correct.

Instead of the above Hoeffding approach, one could use CLT heuristic analysis to upper bound (5.14). In particular, by the independence of the x_i's, we can approximate the distribution of $\sum_{i=1}^{N_{\mathsf{dec}}} x_i$ by a Gaussian distribution with mean μ and standard deviation σ that we can explicitly compute and then use standard Gaussian tail bounds to bound p_e. To be more precise, a straightforward computation using the independence of the x_i, and that the standard deviation of χ_e is $\sqrt{2\eta/4} = \sqrt{\eta/2}$ shows that the standard deviation of $\sum_{i=1}^{N_{\mathsf{dec}}} x_i$ is given by

$$\sigma = \sqrt{(B_{\mathsf{eff}}^2/12 + B_{\mathsf{eff}}/4 + 1/6) \cdot (\eta/2) \cdot N_{\mathsf{dec}}}, \tag{5.19}$$

where

$$B_{\mathsf{eff}} = \sqrt{\rho B_1^2 + (1-\rho) B_2^2},$$

and

$$\rho = N_{\mathsf{dec1}}/N_{\mathsf{dec}}.$$

Using a standard Gaussian tail bound along with union bound over the d coordinates as above, one gets

$$p_e^{\mathsf{clt}} \leq (2d) \cdot \exp(-z_{\mathsf{clt}}^2/2), \tag{5.20}$$

where

$$z_{\mathsf{clt}} = \lfloor q/p \rfloor/(2\sigma). \tag{5.21}$$

Furthermore, using union bound one can calculate $z_{\mathsf{Hoeffding}}$ such that the calculated $p_e^{\mathsf{Hoeffding}}$ satisfies the following inequality

$$p_e^{\mathsf{Hoeffding}} \leq (2d) \cdot \exp(-z^2_{\mathsf{Hoeffding}}/2). \tag{5.22}$$

In Tables 5.3-5.4, we compare our derived $z_{\mathsf{Hoeffding}}$ in (5.22) with that of z_{clt} in (5.21) for our different parameter sets. The results suggest that our provable Hoeffding bounds on the decryption error probability are close optimal, as they are not much higher than the bounds obtained from the CLT heuristic.

Table 5.3: The values of $z_{\mathsf{Hoeffding}}$ in (5.22) and z_{clt} defined in (5.21) for Titanium-CPA.

Parameter	Toy64	Lite96	Std128	Med160	Hi192	Super256
$z_{\mathsf{Hoeffding}}$	7.39	7.45	7.63	8.32	8.06	10.61
z_{clt}	7.55	7.64	7.83	8.58	8.26	10.93

Table 5.4: The values of $z_{\mathsf{Hoeffding}}$ in (5.22) and z_{clt} defined in (5.21) for Titanium-CCA.

Parameter	Toy64	Lite96	Std128	Med160	Hi192	Super256
$z_{\mathsf{Hoeffding}}$	11.42	15.25	15.36	17.00	17.74	22.45
z_{clt}	11.67	15.61	15.69	17.43	18.23	23.26

5.3 Concrete correctness condition of Titanium-CCA-S

We similarly define the following correctness for Titanium-CCA-S

Definition 5.3.1. *Our* Titanium-CCA *scheme is called δ-correct if*

$$\Pr[\mathsf{Decrypt}(\mathsf{sk}, \mathsf{ct}) \neq k | (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}; (k, \mathsf{ct}) \leftarrow \mathsf{Encrypt}(\mathsf{pk})] \leq \delta.$$

As we follow the KEM construction given in [HHK17], the following result is outstanding.

Lemma 5.3.1. *If* Titanium-CPA-S *is δ-correct and* G *and* H *are random oracles, then our* Titanium-CCA-S *is δ-correct.*

Chapter 6

Security Analysis

6.1 Introduction

This Chapter contains the Security analysis of our Titanium schemes. After introducing some preliminary notations and results, we present security proofs for our Titanium-CPA and Titanium-CCA schemes from the security of the MP-LWE problem, and then from the hardness of PLWE^f problem for f in a large family of ring polynomials $\mathcal{F}_1$ that we introduce. We then present and analyse the security of the PLWE^f problem against best known attacks, and explain how we select our scheme parameters based on the complexity of those attacks, combined with our security proofs, to give lower bounds on the complexity of attacks against Titanium-CPA and Titanium-CCA. Finally, we present our best known attacks on Titanium-CPA and Titanium-CCA and their complexity for our parameter sets.

6.2 Security analysis preliminaries

6.2.1 Probability

We use the following variant of the Leftover hash Lemma (LHL)[DORS08].

Lemma 6.2.1. *Let X, Y, Z denote finite sets. Let $\mathcal{H}$ be a universal family of hash functions $h : X \to Y$. Let $f : X \to Z$ be arbitrary. Then for any random variable T taking values in X, we have:*

$$\Delta\left((h, h(T), f(T)), (h, U(Y), f(T))\right) \;\leq\; \frac{1}{2} \cdot \sqrt{\gamma(T) \cdot |Y| \cdot |Z|},$$

where $\gamma(T) = \max_{t \in X} \Pr[T = t]$.

We will apply the LHL to the following universal hash family that arises in our construction.

Lemma 6.2.2 (Adapted from [RSSS17])**.** *Let $q, k, d \geq 2$, q prime, and $\mathsf{Supp}_r \subseteq \mathbb{Z}_q^{<k+1}[x]$. For $(b_i)_i \in (\mathbb{Z}_q^{<d+k}[x])^t$, we let $h_{(b_i)_i}$ denote the map that sends $(r_i)_{i \leq t} \in (\mathsf{Supp}_r)^t$ to $\sum_{i \leq t} r_i \odot_d b_i \in \mathbb{Z}_q^{<d}[x]$. Then the hash function family $(h_{(b_i)_i})_{(b_i)_i}$ is universal.*

Proof. Our aim is to show that for $r_1, \ldots, r_t$ not all 0 in Supp_r, we have

$$\Pr_{(b_i)_i, (b'_i)_i}\left[\sum_{i \leq t} r_i \odot_d b_i = \sum_{i \leq t} r_i \odot_d b'_i\right] = q^{-d}.$$

W.l.o.g. we may assume that $r_1 \neq 0$. By linearity, it suffices to prove that for all $y \in \mathbb{Z}_q^{<d}[x]$,

$$\Pr_{b_1}\left[r_1 \odot_d b_1 = y\right] = q^{-d}.$$

Let j be minimal such that the coefficient in x^j of r_1 is non-zero and hence co-prime to q. Then the equation $r_1 \odot_d b_1 = y$ restricted to entries $j+1$ to $j+d$ is a triangular linear system in the coefficients of b_1 with diagonal coefficients invertible mod q. The map $b_1 \mapsto r_1 \odot_d b_1$ restricted to these coefficients of b_1 is hence a bijection. This gives the equality above. □

6.2.2 Matrices

We use the following notations:

- For any $d, k > 0$ and $a \in R[x]^{<k}$ for a ring R, we let $\mathsf{Toep}^{d,k}(a)$ denote the (Toeplitz) matrix in $R^{d\times(k+d-1)}$ whose i-th row, for $i = 1, \ldots, d$, is given by the coefficients of $x^{i-1} \cdot a$.
- For a matrix M over $\mathbb{R}$, we denote by $\|\mathsf{M}\|$ its matrix norm (i.e. the magnitude of its largest singular value).
- For a Henkel matrix $\mathsf{M} \in R^{m\times n}$ for a ring R, and $1 \le j \le m + n - 1$, we denote by $\mathsf{ADiag}_j(\mathsf{M})$ the element (in R) repeated along the j-th anti-diagonal of M (i.e. the elements $\mathsf{M}_{i,k}$ with $i + k = j + 1$).
- For a polynomial f over R of degree m, we define M_f as the Hankel matrix with anti-diagonal element $\mathsf{ADiag}_j(\mathsf{M}_f)$ being the constant coefficient of the polynomial $x^{j-1} \bmod f$, for $j = 1, \ldots, 2m - 1$.

6.2.3 Lattices

We refer the reader to [MG02] for an introduction to lattices and their computational aspects.

6.2.4 Security models

We refer to the following definitions from [HHK17]. We denote the sampling of a random element x from distribution $\mathcal{X}$ by $x \hookleftarrow \mathcal{X}$.

Definition 6.2.1 (IND-CPA)**.** *Let* $\mathsf{PKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *be a public-key encryption scheme with message space* $\mathcal{M}$*. We define the* IND-CPA *game as in Algorithm 53, and the* IND-CPA *advantage function of an adversary* $\mathsf{A} = (\mathsf{A}_1, \mathsf{A}_2)$ *against* PKE *(such that* A_2 *has binary output) as*

$$\varepsilon^{(\mathsf{IND\text{-}CPA})}(\mathsf{A}) := \left|\Pr[\mathsf{IND\text{-}CPA}^{\mathsf{A}} \to 1] - 1/2\right|.$$

Algorithm 53 IND-CPA Game.

1: $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}$.
2: $b \hookleftarrow U(\{0, 1\})$.
3: $(m_0^*, m_1^*, \mathsf{st}) \leftarrow \mathsf{A}_1(\mathsf{pk})$.
4: $c^* \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b^*)$.
5: $b' = \mathsf{A}_2(\mathsf{pk}, c^*, \mathsf{st})$.
6: Return $b' =_? b$.

Definition 6.2.2 (IND-CCA)**.** *Let* $\mathsf{KEM} = (\mathsf{Gen}, \mathsf{Encaps}, \mathsf{Decaps})$ *be a key encapsulation mechanism with key space* $\mathcal{K}$*. We define the* IND-CCA *game as in Algorithm 54, and the* IND-CCA *advantage function of an adversary* A *(with binary output) against* KEM *as*

$$\varepsilon^{(\mathsf{IND\text{-}CCA})}(\mathsf{A}) := \left|\Pr[\mathsf{IND\text{-}CCA}^{\mathsf{A}} \to 1] - 1/2\right|.$$

Algorithm 54 IND-CCA Game.

1: $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}$.
2: $b \hookleftarrow U(\{0, 1\})$.
3: $(k_0^*, c^*) \leftarrow \mathsf{Encaps}(\mathsf{pk})$.
4: $k_1^* \hookleftarrow \mathcal{K}$.
5: $b' = \mathsf{A}^{\mathsf{Decaps}}(c^*, k_b^*)$.
6: Return $b' =_? b$.

6.2.5 Computational models

In both our concrete classical security proofs and attack complexity estimates, we model attackers as Random Access Machines (RAMs), with instructions that perform elementary two-input bit 'gate' operations (AND/OR/NOT) on arguments, and can access arbitrary memory locations in constant time (each access is counted as 1 elementary operation). We use as our 'time' complexity measure (denoted by T) for a RAM the sum of the actual run-time (number of elementary gate instructions executed) plus its program size (bits). The inclusion of program size in the 'time' measure models the cost of memory for storage of pre-computation look up tables. For an attack with run-time T that succeeds with advantage ε, we measure the overall 'gate complexity level' of the attack by the time to advantage ratio T/ε.

We chose the above model based on a conservative security viewpoint. Since a boolean circuit of N (AND/OR/NOT) two-input gates can be simulated by a RAM of run-time $T \approx N$ and memory $M \approx N$ (including program instructions), any lower bound L on the 'time' (run-time plus program size) of a RAM for some computational problem implies a similar lower bound $\approx L/2$ on the size of a circuit for solving the problem. Therefore, our RAM bounds also correspond to approximate circuit complexity lower bound estimates, neglecting for conservative measure the cost of the RAM's memory access instructions, which is highly dependent on the computing architecture used. (In general, the reverse simulation of a RAM of run-time T and memory M requires a circuit of size $O(T \cdot M)$, but as a conservative measure, in our claimed lower bound security estimates, we neglect the memory access cost in assessing the circuit complexity of attacks for setting our parameters.)

Similarly, for our quantum attack complexity estimates, we assume a RAM-like quantum machine, as also assumed in the quantum SVP sieve estimates of [LMvdP15], with instructions that perform elementary two-qubit gate operations (H/CNOT/T gates, i.e. the 'Clifford+T' elementary operations [AMM14, AMG$^+$16]) on arguments, and can quantumly address arbitrary memory locations in constant time (each access is counted as 1 elementary operation). Although the implementation prospects of a quantum RAM-like model is unclear, we chose to use it as a conservative measure for security level estimates. Similarly to the classical case, for an attack with run-time T that succeeds with advantage ε, we measure the overall 'gate complexity level' of the attack by the time to advantage ratio T/ε.

6.2.6 Grover-type bounds on quantum search problems

Grover [Gro96] gave a celebrated quantum algorithm for the quantum search problem. The basic variant of the problem is the following: given access to a quantum oracle evaluating a function $F : X \to Y$ where $F(x^*) = 1$ for some $x^* \in X$ and $F(x) = 0$ for $x \neq x^*$, find x^*. Grover's quantum algorithm [Gro96] solves this problem with probability $p_e^{\mathsf{Grover}} \approx Q^2 \cdot |X|^{-1}$ given Q quantum queries to the oracle, where $|X|$ denotes the number of elements in X (a quadratic improvement over the success probability $Q \cdot |X|^{-1}$ of the best classical algorithm for this problem). Here, we state an upper bound from [HHK17] on the success probability of any quantum algorithm for a variant of the quantum search problem called 'Generic Search Problem' (GSP). The upper bound is very close to Grover's algorithm success probability for this problem. In the GSP with parameter δ, $F : X \to Y$ denotes a function such that $F(x) \in \{0, 1\}$ is independently chosen (before giving oracle access to the attacker) for each $x \in X$ as a Bernoulli random variable with $\Pr[F(x) = 1] = \lambda(x) \leq \delta$, where $\lambda : X \to [0, 1]$ is a fixed function. The attacker is allowed Q quantum queries to a quantum implementation of F and the attack succeeds if it returns an x^* such that $F(x^*) = 1$.

Lemma 6.2.3 ([HHK17])**.** *For any (unbounded time) quantum GSP (with parameter δ) algorithm making Q quantum queries to its oracle, the success probability is at most $8 \cdot (Q + 1)^2 \cdot \delta$.*

6.3 Security proofs, hard problem attacks, and parameter selection approach

6.3.1 Security proof: IND-CPA of Titanium-CPA from IND-CPA of Titanium-CPA-S

To simplify the rest of our security analysis of Titanium-CPA, we first show that its IND-CPA security is as hard as that of the simplified algebraic scheme Titanium-CPA-S defined in Chapter 4. Our security reduction is in the Random Oracle Model [BR93] (ROM) and models the XOF cryptographic hash function underlying the PRG algorithm used in Titanium-CPA as a black box random function (a random oracle) that the Titanium-CPA attacker can query.

Lemma 6.3.1 (IND-CPA of Titanium-CPA from IND-CPA of Titanium-CPA-S)**.** . *Any* IND-CPA *attack against* Titanium-CPA *with run-time T and advantage ε in the Random Oracle Model for* XOF *with at most Q queries to the* XOF *random oracle, implies an* IND-CPA *attack against* Titanium-CPA-S *with run-time*

$$T' \approx T, \tag{6.1}$$

and distinguishing advantage

$$\varepsilon' \geq \varepsilon - 3 \cdot Q/2^{256}. \tag{6.2}$$

Proof. Let A denote an IND-CPA attack algorithm against Titanium-CPA. The proof consists of nine games (let p_i be the attacker's success probability in Game_i).

- Game_0 : The original IND-CPA attack game against Titanium-CPA with attacker A. In this game, each query $\mathsf{qu} = (K, c, L, S)$ to the random oracle XOF is answered by looking up qu in a table XOFTab of previous query-answer pairs. If an entry of the form $(\mathsf{qu}, \mathsf{out})$ exists in the table, the corresponding answer out is returned. Otherwise, an independent random answer out is sampled uniformly at random from $\{0,1\}^L$ and the entry $(\mathsf{qu}, \mathsf{out})$ is added to XOFTab.

- Game_1 : In this game, we change lines 3 and 4 of Titanium-CPA.KeyGen algorithm to sample $(\mathsf{prgst}, \mathsf{seedsk})$ and $(\mathsf{prgst}, \mathsf{seedpk})$ independently and uniformly at random from $\mathsf{StSp}_{\mathsf{prg}} \times \mathsf{byte}^{32}$ (instead of calling PRG.Out, which in Game_0 leads to a query to XOF of the form $\mathsf{qu} = (K = (\mathsf{seedkg}, 0), c, L, S)$).

  Let $\mathsf{Bad}_{\mathsf{kg}}$ denote the event that a query of the form $\mathsf{qu} = (K = (\mathsf{seedkg}, 0), c, L, S)$ (for some L, S) is made to XOF in Game_1. When $\mathsf{Bad}_{\mathsf{kg}}$ does not occur, A's view in Game_1 is identical to its view in Game_0, so $|p_1 - p_0| \leq \Pr[\mathsf{Bad}_{\mathsf{kg}}]$. Moreover, since seedkg is uniformly random in byte^{32} and is not used anywhere by the challenger in Game_1, we have $\Pr[\mathsf{Bad}_{\mathsf{kg}}] \leq Q/2^{256}$, so:

  $$|p_1 - p_0| \leq Q/2^{256}.$$

- Game_2 : In this game, we change the way we answer XOF queries of SampUnifZq when the latter is called by public key sampling algorithm sampa. Namely, for $i = 1, \ldots, t$, and $\tau \in \mathbb{Z}$ let $\mathbf{r}^{i,\tau}$ denote the τ-th (starting from 0) vector returned by PRG.Out in the execution of SampUnifZq when it is called by sampa to sample $\bar{a}_i$. Let $x_\theta^{i,\tau} \in \mathbb{Z}$ denote the θ-th block of bytpm bits from $\mathbf{r}^{i,\tau}$, and for $j \in \{0, \ldots, n-1\}$, let $x_{\theta(j)}^{i,\tau(j)}$ denote the j-th such block that passes the acceptance test $x_\theta^{i,\tau} < \mathsf{ZqRej}$ at line 12 of SampUnifZq in Algorithm 8.

  Let S_i denote the indices of accepted x's, i.e. $\{(\theta(j), \tau(j)) : j = 0, \ldots, n-1\}$. In Game_2, we replace the value of $x_{\theta(j)}^{(i,\tau(j))}$ (returned by XOF as answer to the relevant query and placed in XOFTab) with $\bar{a}_i[j] + \kappa_{i,j} \cdot q$ where $\bar{a}_i[j]$ and $\kappa_{i,j}$ sampled independently and uniformly at random from $\mathbb{Z}_q$ and $\{0, \ldots, \mathsf{ZqRej}/q - 1\}$, respectively. Note that this will result in $\bar{a}_i[j]$ being returned as the j-th coefficient of $\bar{a}_i$ in Game_2, so the distribution of $(a_1, \ldots, a_t)$ in Game_2 is $U(\mathbb{Z}_q^{<n})^t$.

  By the correctness of SampUnifZq in Algorithm 8 (see Chapter 7) and the uniformly random and independent distribution of the $\mathbf{r}^{i,\tau}$, we have that $\bar{a}_i$ is uniformly random in both Game_1 and

Game_2. The accepted indices S_i and the values of rejected blocks $x_\theta^{i,\tau}$ for $(\theta,\tau) \notin S_i$ are generated in exactly the same way in both games. Moreover, in Game_1, conditioned on the the distribution of S_i and the values of rejected blocks $x_\theta^{i,\tau}$ for $(\theta,\tau) \notin S_i$, the values of accepted blocks $x_{\theta(j)}^{i,\tau(j)}$ for $j = 0,\ldots,n-1$ are independent uniformly random integers in $\{0,\ldots,\mathsf{ZqRej}-1\}$. In Game_2, the same conditional distribution holds since $\bar{a}_i[j] + \kappa_{i,j} \cdot q$ is uniform in $\{0,\ldots,\mathsf{ZqRej}/q-1\}$ when $\bar{a}_i[j]$ and $\kappa_{i,j}$ are independently uniform in their domains. Therefore, since the distribution of the view of A remains the same in Game_1 and Game_2, we have

$$|p_2 - p_1| = 0.$$

- Game_3 : In this game, we define a new algorithm $\mathsf{SampUnifZqsk}$ that works exactly the same as $\mathsf{SampUnifZq}$, and change line 3 of Samps to call $\mathsf{SampUnifZqsk}$ (instead of calling $\mathsf{SampUnifZq}$ as in previous game). Since $\mathsf{SampUnifZqsk}$ and $\mathsf{SampUnifZq}$ are identical, this does not change A's view and we have $|p_3 - p_2| = 0$.

- Game_4 : In this game, we change lines 2 and 6 of $\mathsf{SampUnifZqsk}$ to sample $\mathbf{r}$ independently and uniformly at random from $\mathsf{byte}^{\mathsf{bytpc}}$ and $\mathsf{byte}^{\mathsf{bytpm}}$, respectively (instead of calling $\mathsf{PRG.Out}$, which in Game_3 leads to a query to XOF of the form $\mathsf{qu} = (K = (\mathsf{seedsk}, c'), C, L, S)$). We also change line 3 of Sampe to sample $\mathbf{r}$ independently and uniformly at random from $\mathsf{byte}^{t\cdot(d+k+1)\cdot\mathsf{Zebytes}}$ (instead of calling $\mathsf{PRG.Out}$, which in Game_3 leads to a query to XOF of the form $\mathsf{qu} = (K = (\mathsf{seedsk}, c'), C, L, S)$).

  Let $\mathsf{Bad_{sk}}$ denote the event that a query of the form $\mathsf{qu} = (K = (\mathsf{seedsk}, c'), C, L, S)$ (for some c', L, S) is made to XOF in Game_4. When Bad_{sk} does not occur, A's view in Game_4 is identical to its view in Game_3, so $|p_4 - p_3| \leq \Pr[\mathsf{Bad}_{sk}]$. Moreover, since seedsk is uniformly random in byte^{32} and is not used anywhere by the challenger in Game_4, we have $\Pr[\mathsf{Bad}_{sk}] \leq Q/2^{256}$, so:

$$|p_4 - p_3| \leq Q/2^{256}.$$

- Game_5 : In this game, we change line 3 of $\mathsf{Titanium\text{-}CPA}$'s Samps algorithm to sample $\widehat{\mathbf{s}}$ independently and uniformly at random from $\mathbb{Z}_q^{\dim_3}$ (instead of calling $\mathsf{SampUnifZqsk}$). By correctness of $\mathsf{SampUnifZq}$ with uniformly random $\mathbf{r}$'s (see Lemma 5.1.2), the distribution of $\widehat{\mathbf{s}}$ is identical in Game_5 to Game_4. Therefore, $|p_5 - p_4| = 0$.

- Game_6 : In this game, we change line 8 of $\mathsf{Titanium\text{-}CPA.KeyGen}$ algorithm to sample s independently and uniformly at random from $\mathbb{Z}_q^{<n+d+k-1}[x]$ (instead of letting $s = \mathsf{Trunc}(n+k+d-1, s')$, where $s' = \mathsf{VecToPol}(\mathsf{d}_3^{-1} \cdot \mathsf{NTT}_{\mathsf{d}_3}(\omega_3, \mathsf{InvPerm}_{\mathsf{d}_3}(\widehat{\mathbf{s}})))) \in \mathbb{Z}_q^{<n+d+k-1}[x]$).

  Since the mappings $\mathsf{InvPerm}_{\mathsf{d}_3}$, $\mathsf{NTT}_{\dim_3}$, $\mathbf{v} \mapsto \dim_3^{-1} \cdot \mathbf{v}$ and $\mathsf{VecToPol}$ are all injective, s' is independent and uniformly random in $\mathbb{Z}_q^{<\dim_3}[x]$ in Game_5, and therefore s is independent and uniformly random in $\mathbb{Z}_q^{<n+d+k-1}[x]$ in Game_5, exactly as in Game_6. Therefore, $|p_6 - p_5| = 0$.

- Game_7 : In this game, we change line 7 of $\mathsf{Titanium\text{-}CPA.KeyGen}$ algorithm to sample $(e_1,\ldots,e_t)$ from $\chi_e = \mathsf{BinDiff}(\eta)^t$ (instead of calling Sampe). By correctness of Sampe with uniformly random $\mathbf{r}$ (see Lemma 5.1.2), the distribution of $(e_1,\ldots,e_t)$ is identical in Game_7 and Game_6. Therefore, $|p_7 - p_6| = 0$.

- Game_8 : In this game, we change line 5 of Sampr to sample $\mathbf{r}$ independently and uniformly at random from $\mathsf{byte}^{N_{\mathsf{dec}}\cdot\mathsf{Zbbytes}+N_{\mathsf{dec}}/8}$ (instead of calling $\mathsf{PRG.Out}$, which in Game_7 leads to a query to XOF of the form $\mathsf{qu} = (K = (\mathsf{seedr}, c'), C, L, S)$).

  Let $\mathsf{Bad_r}$ denote the event that a query of the form $\mathsf{qu} = (K = (\mathsf{seedr}, c'), C, L, S)$ (for some c', L, S) is made to XOF in Game_8. When $\mathsf{Bad_r}$ does not occur, A's view in Game_8 is identical to its view in Game_7, so $|p_8 - p_7| \leq \Pr[\mathsf{Bad}_r]$. Moreover, since seedr is uniformly random in byte^{32} and is not used anywhere by the challenger in Game_8, we have $\Pr[\mathsf{Bad}_r] \leq Q/2^{256}$, so:

$$|p_8 - p_7| \leq Q/2^{256}.$$

- Game_9 : In this game, we change line 5 of Titanium-CPA.Encrypt algorithm to sample $(r_1, \ldots, r_t)$ from χ_r (instead of calling Sampr). By correctness of Sampr with uniformly random $\mathbf{r}$ (see Lemma 5.1.2), the distribution of $(r_1, \ldots, r_t)$ is identical in Game_9 and Game_8. Therefore, $|p_9 - p_8| = 0$.

We now modify the last game to construct an IND-CPA attacker A' against Titanium-CPA-S. Given the challenge public key $(\bar{a}_1, \ldots, \bar{a}_t), (b_1, \ldots, b_t)$ of Titanium-CPA, A' samples seedpk uniformly at random from byte^{32} and simulates the execution of Sampa(seedpk) to program the random oracle XOF using the coordinates $\bar{a}_i[j] \in \mathbb{Z}_q$ as in Game_9 (see Game_2 changes). A' then computes a Titanium-CPA public key pk from b_i's seedpk by simulating steps 12 and 14 of Titanium-CPA.KeyGen algorithm, and runs A on input pk, simulating A's XOF oracle queries as in Game_9. When A outputs a challenge message pair $(\mathsf{m}_0, \mathsf{m}_1)$, A' computes and outputs to its challenger the pair $(m_0 = \mathsf{Decodem}(\mathsf{m}_0), m_1 = \mathsf{Decodem}(\mathsf{m}_0))$, receiving back the challenge ciphertext (c'_1, c'_2). A' then simulates steps 6-7 and 10-11 of Titanium-CPA.Encrypt algorithm to compute a challenge ciphertext ct for Titanium-CPA from (c'_1, c'_2), which A' returns to A. Finally, A' returns the same bit that A outputs.

It is easy to verify that A' running with its challenger, simulates the same view to A as in Game_9. It follows from the above sequence of games that the success probability of A' is lower bounded as

$$\varepsilon' = p_9 \geq p_0 + 3 \cdot Q/2^{256}.$$

The stated run-time of A' follows from the NTT and MP computations in dimension $\leq n + k + d$ over $\mathbb{Z}_q$ that can be computed in time $O((n + d + k) \log(n + d + k))$ arithmetic operations (additions and multiplications) over $\mathbb{Z}_q$, as shown in Chapter 3. □

6.3.2 Security proof: IND-CPA of Titanium-CPA-S from MP-LWE hardness

We base IND-CPA security of Titanium-CPA-S on the Middle-Product LWE problem [RSSS17] (MP-LWE), a variant of LWE defined over $\mathbb{Z}_q[x]$ as follows.

Definition 6.3.1 (MP distribution [RSSS17])**.** *Let $n, d > 0$, $q \geq 2$, and χ a distribution over $\mathbb{Z}^{<d}[x]$. For $s \in \mathbb{Z}_q^{<n+d+1}[x]$, we define the distribution $\mathsf{MP}_{q,n,d,\chi}(s)$ over $\mathbb{Z}_q^{<n}[x] \times \mathbb{Z}_q^{<d}[x]$ as the one obtained by: sampling $a \hookleftarrow U(\mathbb{Z}_q^{<n}[x])$, $e \hookleftarrow \chi$ and returning $(a, b = a \odot_d s + e)$.*

Definition 6.3.2 (MP-LWE [RSSS17])**.** *Let $n, d > 0$, $q \geq 2$, and a distribution χ over $\mathbb{Z}^{<d}[x]$. The (decision) $\mathsf{MP\text{-}LWE}_{q,n,d,\chi}$ problem consists in distinguishing between arbitrarily many samples from $\mathsf{MP}_{q,n,d,\chi}(s)$ and the same number of samples from $U(\mathbb{Z}_q^{<n}[x] \times \mathbb{Z}_q^{<d}[x])$, with non-negligible probability over the choice of $s \hookleftarrow U(\mathbb{Z}_q^{<n+d+1}[x])$. If the number of $\mathsf{MP}_{q,n,d,\chi}(s)$ samples is restricted to t, we write $\mathsf{MP\text{-}LWE}_{q,n,d,t,\chi}$ to denote the corresponding restricted MP-LWE problem.*

We show that, under appropriate choice of parameters, the IND-CPA security of Titanium-CPA-S is as hard as the MP-LWE problem. The proof is based on adapting the Leftover hash Lemma (LHL) based argument from [RSSS17], with relatively mild changes and generalizations to account for the relatively mild differences between Titanium-CPA-S and the encryption scheme presented in [RSSS17].

Lemma 6.3.2 (IND-CPA of Titanium-CPA-S from MP-LWE, adapted from [RSSS17])**.** *Assume that*

$$q \text{ is prime}, \tag{6.3}$$

and the following Leftover Hash Lemma (LHL) condition holds:

$$t \geq \frac{2 \cdot (\log(\Delta_{\mathsf{LHL}}^{-1}) - 1) + (n + d + k) \cdot \log q}{(k + 1) \cdot b_{\mathsf{LHL}}}, \tag{6.4}$$

where

$$b_{\mathsf{LHL}} \stackrel{\text{def}}{=} \rho \cdot (b_1 + 1) + (1 - \rho) \cdot (b_2 + 1), \tag{6.5}$$

and

$$\rho \stackrel{\text{def}}{=} \frac{N_{\mathsf{dec1}}}{N_{\mathsf{dec}}},\ \textit{with}\ N_{\mathsf{dec}} \stackrel{\text{def}}{=} (k+1)\cdot t. \tag{6.6}$$

Then any IND-CPA *attack against* Titanium-CPA-S *with run-time* T *and advantage* ε*, implies an attack against the* $\mathsf{MP\text{-}LWE}_{q,n,d+k,D_{\alpha q}}$ *problem with run-time*

$$T_{\mathsf{MP\text{-}LWE}} \approx T \tag{6.7}$$

and distinguishing advantage

$$\varepsilon_{\mathsf{MP\text{-}LWE}} \geq \varepsilon/2 - \Delta_{\mathsf{LHL}}. \tag{6.8}$$

Proof. We summarize the modifications of the argument in [RSSS17] and the concrete reduction cost. The proof consists in three games (let p_i be the attacker A's success probability in Game_i).

- Game_0 : The original IND-CPA game.
- Game_1 : Instead of generating $\mathsf{pk} = (\bar{a}_i, b_i)_{i\leq t}$ with $b_i = a_i \odot_{d+k} s + e_i \in \mathbb{Z}_q^{<d+k}[x]$ using Titanium-CPA-S.KeyGen, where we define $a_i = \mathsf{Rev}(\bar{a}_i)$ for $i = 1,\ldots,t$, the challenger sets $b_i \hookleftarrow U(\mathbb{Z}_q^{<d+k}[x])$ independently of a_i.

  We can construct a distinguishing attacker against $\mathsf{MP\text{-}LWE}_{q,n,d+k,D_{\alpha q}}$ given t samples, that has run-time $T_{\mathsf{MP\text{-}LWE}} = T + O(t\cdot(n+d+k)\cdot\log q)$ and distinguishing advantage $\varepsilon_{\mathsf{MP\text{-}LWE}} = |p_1 - p_0|$. Given t MP-LWE samples $(a'_i, b'_i)_{i\leq t}$, the MP-LWE attacker computes $\bar{a}_i = \mathsf{Rev}(a'_i)$ and $b_i = b'_i$ for $i = 1,\ldots,t$, and sets $\mathsf{pk} = (\bar{a}_i, b_i)_{i\leq t}$ as the public key. If (a'_i, b'_i) have the MP distribution (resp. uniform distribution), then $(\bar{a}_i, b_i)_{i\leq t}$ have the correct public key distribution as in Game_0 (resp. Game_1), using the fact that Rev is an injective mapping on $\mathbb{Z}_q^{<n}[x]$.
- Game_2 : Instead of generating the second challenge ciphertext component c_2 as $c'_2 = \sum_{i=1}^{t} \mathsf{Rev}(r_i) \odot_d b_i + \lfloor q/p \rfloor \cdot m \in \mathbb{Z}_q^{<d}[x]$, the challenger sets $c_2 \hookleftarrow U(\mathbb{Z}_q^{<d}[x])$, but leaves $c_1 = \sum_{i\leq t} r_i \cdot a_i$ as before. By the Leftover Hash Lemma 6.2.1 with $\gamma(T) = B_1^{N_{\mathsf{dec1}}} \cdot B_2^{N_{\mathsf{dec}} - N_{\mathsf{dec1}}}$ the (exponential of) the inverse min-entropy of the input $(\mathsf{Rev}(r_1),\ldots,\mathsf{Rev}(r_t))$ to the universal hash family in Lemma 6.2.2, $|Y| = q^d$ the hash output space size, and $|Z| = q^{n+k}$ the size of the leakage space due to c_1, the statistical distance between the distributions of the challenge ciphertext in Game_2 and Game_1 is at most Δ_{LHL} if the condition

  $$\frac{1}{2}\cdot\sqrt{B_1^{-N_{\mathsf{dec1}}} \cdot B_2^{-(N_{\mathsf{dec}} - N_{\mathsf{dec1}})} q^{n+d+k}} \leq \Delta_{\mathsf{LHL}} \tag{6.9}$$

  holds, which is equivalent to (6.4), using the definitions $N_{\mathsf{dec}} \stackrel{\text{def}}{=} (k+1)\cdot t$, $B_1 = 2^{b_1+1}$ and $B_2 = 2^{b_2+1}$.

In the last game, the attacker's view is independent of the encrypted challenge message, so $p_2 = 1/2$. It follows that $|p_0 - p_2| = |p_0 - 1/2| = \varepsilon/2 \leq |p_1 - p_0| + |p_2 - p_1| \leq \varepsilon_{\mathsf{MP\text{-}LWE}} + \Delta_{\mathsf{LHL}}$, which gives (6.8). □

6.3.3 Security proof: IND-CCA of Titanium-CCA from IND-CCA of Titanium-CCA-S

We simplify the rest of our security analysis of Titanium-CCA by first showing that its IND-CCA security is as hard as that of the slightly simpler scheme Titanium-CCA-S defined in Chapter 4. Our security reduction is in the Random Oracle Model [BR93] (ROM) and models the XOF cryptographic hash function underlying the PRG algorithm used as a black box random function (a random oracle) that the Titanium-CCA attacker can query.

Lemma 6.3.3 (IND-CCA of Titanium-CCA from IND-CCA of Titanium-CCA-S)**.** . *Any* IND-CCA *attack against* Titanium-CCA *with run-time* T *and advantage* ε *in the Random Oracle Model for* XOF *with at most* Q *queries to the* XOF *random oracle, implies an* IND-CCA *attack against* Titanium-CCA-S *with run-time*

$$T' \approx T, \tag{6.10}$$

making $\leq Q$ *queries to* XOF *and having distinguishing advantage*

$$\varepsilon' \geq \varepsilon - Q/2^{256}. \tag{6.11}$$

Proof. Let A denote an IND-CCA attack algorithm against Titanium-CCA. The proof consists of two games (let p_i be the attacker's success probability in Game_i).

- Game_0 : The original IND-CCA attack game against Titanium-CCA with attacker A. In this game, each query $\mathsf{qu} = (K, c, L, S)$ to the random oracle XOF is answered by looking up qu in a table XOFTab of previous query-answer pairs. If an entry of the form $(\mathsf{qu}, \mathsf{out})$ exists in the table, the corresponding answer out is returned. Otherwise, an independent random answer out is sampled uniformly at random from $\{0,1\}^L$ and the entry $(\mathsf{qu}, \mathsf{out})$ is added to XOFTab.

- Game_1 : In this game, we change lines 3 of Titanium-CCA.KeyGen algorithm to sample $((\mathsf{seedkg.cpa}, \mathsf{rdec}), \mathsf{prgst})$ independently and uniformly at random from $\mathsf{byte}^{32} \times \mathsf{byte}^{32} \times \mathsf{StSp}_{\mathsf{prg}}$ (instead of calling PRG.Out, which in Game_0 leads to a query to XOF of the form $\mathsf{qu} = (K = (\mathsf{seedkg}, 0), c, L, S)$).

  Let $\mathsf{Bad}_{\mathsf{kg}}$ denote the event that a query of the form $\mathsf{qu} = (K = (\mathsf{seedkg}, 0), c, L, S)$ (for some L, S) is made to XOF in Game_1. When $\mathsf{Bad}_{\mathsf{kg}}$ does not occur, A's view in Game_1 is identical to its view in Game_0, so $|p_1 - p_0| \leq \Pr[\mathsf{Bad}_{\mathsf{kg}}]$. Moreover, since seedkg is uniformly random in byte^{32} and is not used anywhere by the challenger in Game_1, we have $\Pr[\mathsf{Bad}_{\mathsf{kg}}] \leq Q/2^{256}$, so:

  $$|p_1 - p_0| \leq Q/2^{256}.$$

We observe that Game_1 simulates the view of A exactly as in an IND-CCA attack on Titanium-CCA-S. The result follows. □

6.3.4 Security proof: IND-CCA of Titanium-CCA-S from IND-CPA of Titanium-CPA

The Titanium-CCA-S scheme is the result of applying a variant of the Fujisaki-Okamoto (FO) transform [FO99, Den03] to the Titanium-CPA-S scheme. The variant of FO transform that we use is called $\mathsf{QFO}^{\not\perp}$ in [HHK17], where a tight reduction from the IND-CCA security of this scheme to the IND-CPA security of the Titanium-CPA-S is given (we remark that the random oracles G and H' in [HHK17] consist respectively of the left and right 32 bytes of the output of G in our scheme, and we also hash the ciphertext appended to the message in the shared secret derivation, rather than just the message). We restate the classical security reduction result from [HHK17] below (a non-tight reduction in the quantum random oracle model is also given for this transform in [HHK17]). We refer the reader to Chapter 7, where an upper bound on the correctness error probability p_e is proved.

Lemma 6.3.4 (IND-CCA of Titanium-CCA-S from IND-CPA of Titanium-CPA, Adapted from [HHK17], Th.3.2 and 3.4)**.** *If* Titanium-CPA *is* p_e*-correct, then any* IND-CCA *attack against* Titanium-CCA-S *with run-time* T *and advantage* ε *with at most* $Q_{\mathsf{XOF}}, Q_{\mathsf{G}}, Q_{\mathsf{H}}$ *queries in the Random Oracle Model for* XOF*,* G *and* H *respectively, implies an* IND-CPA *attack against* Titanium-CPA *with run-time*

$$T' \approx T, \tag{6.12}$$

making $\leq Q_{\mathsf{XOF}}$ *queries to* XOF *and having distinguishing advantage*

$$\varepsilon' \geq (\varepsilon - Q_{\mathsf{G}} \cdot p_e - (2 \cdot Q_{\mathsf{G}} + Q_{\mathsf{H}} + 1)/2^{256})/3. \tag{6.13}$$

6.3.5 Security proof: MP-LWE hardness from PLWE^f hardness over many f's

We base hardness of MP-LWE on the hardest PLWE^f among those with the ring modulus polynomial f in a large family $\mathcal{F}$. The PLWE^f is defined as follows. We first define the distribution the PLWE problem is based on. Note our definition here is slightly different to the corresponding one in [RSSS17], as we consider discrete error distributions over $\mathbb{Z}[x]/f$ rather than continuous distributions.

Definition 6.3.3 (P distribution)**.** *Let $q \geq 2$, $m > 0$, f a polynomial of degree m, χ a distribution over $\mathbb{Z}[x]/f$. Given $s \in \mathbb{Z}_q[x]/f$, we define the distribution $\mathsf{P}^f_{q,\chi}(s)$ over $\mathbb{Z}_q[x]/f \times \mathbb{Z}_q[x]/f$ obtained by sampling $a \hookleftarrow U(\mathbb{Z}_q[x]/f)$, $e \hookleftarrow \chi$ and returning $(a, b = a \cdot s + e)$.*

Definition 6.3.4 (PLWE)**.** *Let $q \geq 2$, $m > 0$, f a polynomial of degree m, χ a distribution over $\mathbb{Z}[x]/f$. The (decision) $\mathsf{PLWE}^f_{q,\chi}$ problem consists in distinguishing between arbitrarily many samples from $\mathsf{P}^f_{q,\chi}(s)$ and the same number of samples from $U(\mathbb{Z}_q[x]/f \times \mathbb{Z}_q[x]/f)$, with non-negligible probability over the choice of $s \hookleftarrow U(\mathbb{Z}_q[x]/f)$. If the number of $\mathsf{P}^f_{q,\chi}(s)$ samples is restricted to t, we write $\mathsf{PLWE}^f_{q,t,\chi}$ to denote the corresponding restricted PLWE problem.*

The important parameters of the family $\mathcal{F}$ for our reduction from MP-LWE to PLWE^f for all $f \in \mathcal{F}$, are what we call the *geometric matrix* and *geometric factor* of $\mathcal{F}$, which are closely related to the largest expansion factor [LM06] over the polynomials in $\mathcal{F}$. They are defined as follows.

Definition 6.3.5 (Geometric Matrix/Factor)**.** *Given a monic polynomial f of degree n, and an integer $d' \leq n$, its d'-geometric matrix $\mathsf{M}^{d'}_f$ is defined as the top d' rows of the Hankel matrix M_f having anti-diagonal element $\mathsf{ADiag}_j(\mathsf{M}_f)$ as the constant coefficient of the polynomial $x^{j-1} \bmod f$, for $j = 1, \ldots, 2m-1$. The geometric factor $G^{d'}(f)$ of f is defined as $G^{d'}(f) = \|\mathsf{M}^{d'}_f\|$. For a family $\mathcal{F}$ of polynomials of degree $\geq d'$, we define its geometric factor $G(\mathcal{F})$ as the maximum of $G^{d'}(f)$ over all f in $\mathcal{F}$.*

The geometric factor of f controls the tightness, in terms of error distribution variance amplification, of the following hardness result in [RSSS17] from PLWE^f to MP-LWE. The geometric matrix of f also describes how the shape of the error distribution is distorted by this security reduction.

Theorem 6.3.1 (adapted from [RSSS17], Le. 3.7)**.** *Let $n, d', t > 0$, $q \geq 2$, and let f denote a polynomial $f \in \mathbb{Z}[x]$ that is monic, has constant coefficient coprime with q, and has degree m in $[d', n]$. Let $\chi_{e,\mathsf{P}}$ denote a PLWE error distribution over $\mathbb{Z}[x]/f$ (i.e. over $\mathbb{Z}^m$ in the coefficient representation of $\mathbb{Z}[x]/f$), and let $\chi_{e,\mathsf{MP}}$ denote a MP-LWE error distribution over $\mathbb{Z}^{<d'}[x]$ (i.e. over $\mathbb{Z}^d$ in the coefficient representation of $\mathbb{Z}^{<d}[x]$) defined in the coefficient representation by*

$$\chi_{e,\mathsf{MP}} \stackrel{\text{def}}{=} \mathsf{J} \cdot \mathsf{M}^{d'}_f \cdot \chi_{e,\mathsf{P}}, \tag{6.14}$$

where J is the matrix for the coefficient reversal function Rev (with 1's on the anti-diagonal and 0's elsewhere).

Then any attack against $\mathsf{MP\text{-}LWE}_{q,n,d',t,\chi_{e,\mathsf{MP}}}$ with run-time $T_{\mathsf{MP\text{-}LWE}}$ and advantage $\varepsilon_{\mathsf{MP\text{-}LWE}}$, implies an attack against the $\mathsf{PLWE}^f_{q,t,\chi_{e,\mathsf{P}}}$ problem with run-time

$$T_{\mathsf{PLWE}} \approx T_{\mathsf{MP\text{-}LWE}} \tag{6.15}$$

and distinguishing advantage

$$\varepsilon_{\mathsf{PLWE}} \geq \varepsilon_{\mathsf{MP\text{-}LWE}}. \tag{6.16}$$

For general f, the geometric matrix $\mathsf{M}^{d'}_f$ of f is closely related to the structure of f, and this causes the distribution $\chi_{e,\mathsf{MP}}$ to also depend on the structure of f. Since our security hedging goal is to reduce from PLWE^f for a large family of f's to MP-LWE with single $\chi_{e,\mathsf{MP}}$ distribution as used in Titanium (namely χ_e), we need to remove from our security reduction such a dependence of $\chi_{e,\mathsf{MP}}$ on

f. In [RSSS17], this is achieved in by using for $\chi_{e,\mathsf{MP}}$ a spherical Gaussian distribution and adding a 'noise unskewing' step in the reduction to 'unskew' the covariance matrix back to a diagonal matrix, which in general tends to amplify the noise variance by a quantity related to the geometric factor of f.

In our Titanium setting, we use for $\chi_{e,\mathsf{MP}}$ the binomial difference distribution $\mathsf{BinDiff}(\eta)$, and we also would like a tight reduction in terms of error amplification, to allow a meaningful setting of our parameters based on the hardness of PLWE, as the latter's concrete hardness has already been studied for some time.

To this end, for our parameter selection procedure, we apply Theorem 6.3.1 to the following ring polynomial family $\mathcal{F}$:

Definition 6.3.6 (Ring polynomial family $\mathcal{F}$)**.** *For integers $n \geq m' \geq d'$, we denote by $\mathcal{F}(n, m', d')$ the set of ring polynomials f of the form*

$$f(x) = x^m + \sum_{i \leq \ell(m)} f_i \cdot x^i \tag{6.17}$$

with

$$m' \leq m \leq n, \tag{6.18}$$

and

$$\ell(m) = \min(m/2 + 1, m + 1 - d'), \tag{6.19}$$

and

$$f_0 \in \{-1, 1\}. \tag{6.20}$$

We choose $\mathcal{F}$ as a hardness basis for our security analysis of MP-LWE because:

- If $m' \leq (1 - \varepsilon')n$ for a constant $\varepsilon' > 0$, $\mathcal{F}$ contains an exponentially large (in n) number of polynomials (rings).
- It potentially (for a suitable choice of d', m' and n) contains both previously used cyclotomic polynomials $x^m + 1$ (for m a power of 2) and non-cyclotomic polynomials $x^m - x - 1$ (for m prime, as in NTRU Prime [BCLvV16]).
- MP-LWE enjoys a tight reduction from the hardest ring in the family, due to the family's optimal d'-geometric factor of 1, by the results summarized below, and the reduction preserves the shape of the distribution if the latter is balanced and has independent coordinates.

The following proposition and its corollary show that $\mathcal{F}$ has an optimal geometric factor.

Proposition 6.3.1. *Let $f(x) = x^m + \sum_{i \leq \ell} f_i \cdot x^i$, with $\ell \leq m/2 + 1$. Then,*

$$\mathsf{ADiag}_1(\mathsf{M}_f) = 1 \tag{6.21}$$

and

$$\mathsf{ADiag}_j(\mathsf{M}_f) = 0 \text{ if } 2 \leq j \leq m \text{ or } m + 2 \leq j \leq 2m - \ell, \tag{6.22}$$

and

$$\mathsf{ADiag}_j(\mathsf{M}_f) = -f_0 \text{ if } j = m + 1 \tag{6.23}$$

and

$$\mathsf{ADiag}_j(\mathsf{M}_f) = f_{2m+1-j} \cdot f_0 \text{ if } 2m - \ell + 1 \leq j \leq 2m - 1. \tag{6.24}$$

Proof. From the definition of M_f, we have $\mathsf{ADiag}_j(\mathsf{M}_f) = (x^{j-1} \bmod f(x)) \bmod x$ for $1 \le j \le 2n-1$. From this (6.21) and (6.22) for $j \le m$ follow immediately. For $j = m+1$, we have $x^m \bmod f(x) = -\sum_{i\le\ell} f_i \cdot x^i$ from the definition of f, which gives (6.23). For $m+2 \le j \le 2m-\ell$, we have $x^{j-1} \bmod f(x) = x^{j-m-1} \cdot (-\sum_{i\le\ell} f_i \cdot x^i) \bmod f(x) = 0$ since $j-m-1+\ell \le m-1$, giving (6.22). Finally, for $2m-\ell+1 \le j \le 2m-1$, we have $x^{j-1} \bmod f(x) = x^{j-m-1} \cdot (-\sum_{i\le\ell} f_i \cdot x^i) \bmod f(x) = -\sum_{i\le\ell} f_i \cdot x^{i+j-m-1} \bmod f(x)$. Using $(x^{i+j-m-1} \bmod f(x)) \bmod x = -f_0$ if $i+j-m-1 = m$ (i.e. $i = 2m+1-j$) and $(x^{i+j-m-1} \bmod f(x)) \bmod x = 0$ if $m+1 \le i+j-m-1 \le 2m-\ell$, we get (6.24). □

For f as in Prop. 6.3.1, the first d' rows of M_f contain elements from anti diagonals $1, \ldots, m+d'-1$. Therefore, if the condition $2m-\ell+1 > m+d'-1$ holds (or equivalently, $\ell \le m+1-d'$), the condition (6.24) is never satisfied in the first d' rows of M_f, so the non-zero columns of $\mathsf{M}_f^{d'}$ are orthogonal and (using $|f_0| = 1$) have unit norm. We therefore obtain the following corollary.

Corollary 6.3.1. *For integers n and $d' \le m' \le n$, the family $\mathcal{F}(n, m', d')$ in Def. 6.3.6 has geometric factor $G(\mathcal{F}) = 1$. Moreover, let $t > 0$, $q \ge 2$, and let f denote a polynomial $f \in \mathcal{F}(n, m', d')$ and has degree m in $[m', n]$. Let $\chi_{e,\mathsf{P}}$ denote a* PLWE *error distribution over $\mathbb{Z}[x]/f$ (i.e. over $\mathbb{Z}^m$ in the coefficient representation of $\mathbb{Z}[x]/f$) that has independent identically distributed coordinates, i.e. $\chi_{e,\mathsf{P}} = \chi_c^m$ for some distribution χ_c over $\mathbb{Z}$ which is balanced (i.e. $\chi_c(x) = \chi_c(-x)$ for all $x \in \mathbb{Z}$). Let $\chi_{e,\mathsf{MP}} = \chi_c^{d'}$.*

Then any attack against $\mathsf{MP\text{-}LWE}_{q,n,d',t,\chi_{e,\mathsf{MP}}}$ *with run-time $T_{\mathsf{MP\text{-}LWE}}$ and advantage $\varepsilon_{\mathsf{MP\text{-}LWE}}$, implies an attack against the* $\mathsf{PLWE}^f_{q,t,\chi_{e,\mathsf{P}}}$ *problem with run-time*

$$T_{\mathsf{PLWE}} \approx T_{\mathsf{MP\text{-}LWE}} \tag{6.25}$$

and distinguishing advantage

$$\varepsilon_{\mathsf{PLWE}} \ge \varepsilon_{\mathsf{MP\text{-}LWE}}. \tag{6.26}$$

In particular, this reduction holds for $\chi_c = \mathsf{BinDiff}(\eta)$ (i.e. $\chi_{e,\mathsf{MP}} = \mathsf{BinDiff}(\eta)^{d+k}$), the error distribution specified for Titanium-CPA-S.

It shows that for f in family $\mathcal{F}(n, m', d')$, the reduction from PLWE to MP-LWE is tight in terms of error variance, and moreover preserves the shape of the distribution, under mild conditions.

6.3.6 Underlying worst-case problems and different function families

In this subsection, we first state a result from [RWS17], which introduces $\mathsf{ApproxSVP}_{\mathcal{O}_K}$ to be the underlying worst case problem of PLWE. We then present the specific function families of interest in this submission and study their properties.

Underlying worst-case problem

Let K be a number field, $\mathcal{O}_K$ its ring of integers, and $q > 2$ be a prime. There are [LPR10, PRSD17] worst-case to average-case reductions from Approximate Shortest Vector Problem (ApproxSVP) restricted to the class of Euclidean lattices corresponding to ideals of $\mathcal{O}_K$ to decision RLWE, in which the secret is chosen from $\mathcal{O}_k^\vee/q\mathcal{O}_k^\vee$, where $\mathcal{O}_k^\vee$ denotes the dual of $\mathcal{O}_k$, denoted by $\mathsf{dRLWE}_{\mathcal{O}_k^\vee}$. If the secret is chosen from $\mathcal{O}_k/q\mathcal{O}_k$, the problem is called the primal dRLWE, and is denoted by $\mathsf{dRLWE}_{\mathcal{O}_k}$. A sequence of reductions as

$$\mathsf{ApproxSVP}_{\mathcal{O}_K} \longrightarrow \mathsf{dRLWE}_{\mathcal{O}_k^\vee} \longrightarrow \mathsf{dRLWE}_{\mathcal{O}_k} \longrightarrow \mathsf{PLWE}^f,$$

is given in [RWS17]. The first reduction is obtained from [PRSD17], the second reduction is a generalisation of [Pei16b] presented in [RWS17], which uses the so-called "polynomial t", and the last one is proven only in [RWS17] for three different function families. For the second reduction, the authors of [RWS17] have used the concept of conductor ideal for which its size is proportional to $f'(\alpha)$, the

first derivative of f, for f being the polynomial in $K = \mathbb{Q}[x]/f$, and α being a root of this polynomial modulo q. For the third reduction, the authors of [RWS17] made use of the associated Vandermonde matrix of $f = \prod_{j=1}^{n}(x - \alpha_j)$, V_f, as it corresponds to the linear map between the coefficient and embedding spaces. Thus a good approximation factor for $\mathsf{ApproxSVP}_{\mathcal{O}_K}$ can be obtained if the distortion given by the condition number of V_f, that is $\|V_f\| \cdot \|V_f^{-1}\|$, and $f'(\alpha)$ are both scaling polynomially with n. Note that, in addition to the above condition, we use smaller noises in our constructions/designs than what we have in the above mentioned reductions.

We remark that cyclotomic polynomials [LPR10], NTRUPrime polynomials [BCLvV16], and a class of functions named $g_{n,a}$ are shown to have quantities $\|V_f\|$ and $\|V_f^{-1}\|$ polynomially in n. However, we will introduce a new function family similar to that of $g_{n,a}$. In particular, the only difference between our function family and $g_{n,a}$ would be the constant coefficient a, for which we need to have $\{\pm 1\}$, while the authors of [RWS17] set it to be a large prime proportional to the degree n of $g_{n,a}$. In the sequel, we introduce our function family $\mathcal{F}_1$. Hence an interesting open problem is to derive similar corresponding properties (with respect to f', $\|V_f\|$, and $\|V_f^{-1}\|$) of class $g_{n,a}$ for $\mathcal{F}_1$.

We specifically study two sub-families of the above introduced $\mathcal{F}$ in Definition 6.3.6: namely $\mathcal{F}_1$ and $\mathcal{F}_2$. The first one is only used for our implementations in Toy64, Lite96, Std128, Med160, Hi192, and Super256. The latter one is only introduced and we study some properties of $\mathcal{F}_2$. The following definitions and notations are outstanding with $m' \leq m \leq n$: In both $\mathcal{F}_1$ and $\mathcal{F}_2$, we have that

Table 6.1: A list of notations used in function families $\mathcal{F}_1$ and $\mathcal{F}_2$.

notation	value
$m_{\max}$	n
m_{mid}	$(n + m')/2$
$m_{\min}$	m'
d'	$d + k$
$\ell(m)$	$\min(m/2 + 1, m + 1 - d')$
gap_1	$n - m'$
gap_2	$m' - d'$

$m'/2 \leq n/2 \leq d'$, which implies that $\ell(m) = m + 1 - d'$ for $m' \leq m \leq n$ in these cases. With this, $\ell(m_{\min}) = \mathsf{gap}_2 + 1$ and $\ell(m_{\max}) = \mathsf{gap}_1 + \mathsf{gap}_2 + 1$.

Function family $\mathcal{F}_1$

Let $d' < m'$. If we restrict the coefficients (except the leading and constant terms) of our function families to $\{-1, 0, 1\}$, then the total number of polynomials of degree m in $\mathcal{F}_1(n, m', d')$ is equal to $2 \cdot 3^{\ell(m)}$, in which the factor 2 is because of the constant coefficient being $\{\pm 1\}$. Since $\ell(m_{\min}) = \mathsf{gap}_2 + 1$, the total number of binary polynomials of degree m in $\mathcal{F}_1$ is at least $2 \cdot 3^{\mathsf{gap}_2+1}$. Hence, the total number of polynomials in $\mathcal{F}_1$ is at least

$$\sum_{m=\mathsf{gap}_2+1}^{\mathsf{gap}_1+\mathsf{gap}_2} 2 \cdot 3^m = 3^{\mathsf{gap}_2+1} \cdot (3^{\mathsf{gap}_1} - 1) \geq 3^{\mathsf{gap}_1+\mathsf{gap}_2}.$$

This is in fact the function family of our interest for our implementation purposes. We summarize quantities gap_1 and gap_2 in $\mathcal{F}_1$ for our implemented parameter sets in Tables 6.2-6.3. The last row of the table shows whether if the studied $\mathcal{F}_1$ includes a power-of-two cyclotomic polynomial or not. The inclusion of such polynomials are of particular interest as we can then compare the security of our schemes with those appeared in [ADPS16] and [BDK+17].

Note that all over our proposal, whenever we talk about $\mathcal{F}$, we actually mean $\mathcal{F}_1$.

Table 6.2: The parameters in $\mathcal{F}_1$ for Titanium-CPA.

Parameter	Toy64	Lite96	Std128	Med160	Hi192	Super256
gap_1	30	30	50	50	50	50
gap_2	142	35	178	462	462	718
lower bound on $\log_3(\lvert\mathcal{F}_1\rvert)$	172	65	256	512	512	768
power-of-two inclusion	✗	✗	✓	✗	✗	✓

Table 6.3: The parameters in $\mathcal{F}_1$ for Titanium-CCA.

Parameter	Toy64	Lite96	Std128	Med160	Hi192	Super256
gap_1	30	30	128	50	50	50
gap_2	142	35	128	462	462	718
lower bound on $\log_3(\lvert\mathcal{F}_1\rvert)$	172	65	256	512	512	768
power-of-two inclusion	✗	✗	✓	✗	✗	✓

Function family $\mathcal{F}_2$

Letting $m' = d'$ and having coefficients from $\{-1, 0, 1\}$, we obtain $\mathcal{F}_2$ that we are not concerned in this submission. However, we study some of the properties of $\mathcal{F}_2$ in Table 6.4. We emphasize that the family $\mathcal{F}_2$ may not achieve the security goals declared in Table 2.11. At one hand, we have that $\log_3(\lvert\mathcal{F}_1\rvert) = \log_3(\lvert\mathcal{F}_2\rvert)$ as $\mathsf{gap}_1 + \mathsf{gap}_2 = n - d'$ is fixed for both families. On the other hand, the new family $\mathcal{F}_2$ covers more power-of-2 cyclotomic polynomials in our specified parameter sets.

Table 6.4: The parameters in $\mathcal{F}_1$ for Titanium-CPA and Titanium-CCA.

Parameter	Toy64	Lite96	Std128	Med160	Hi192	Super256
lower bound on $\log_3(\lvert\mathcal{F}_2\rvert)$	172	65	256	512	512	768
power-of-two inclusion	✓	✗	✓	✓	✓	✓

6.3.7 Best known attacks: PLWE^f over any f in family

The security of PLWE^f has been studied over the last few years within the field of lattice-based cryptography since its introduction in [SSTX09] (where it was called 'Ideal-LWE') and the independent introduction in [LPR10] of the closely related 'Ring-LWE' problem. As special cases of the LWE problem [Reg05], the computational hardness of PLWE^f relies on the hardness of the ApproxSVP problem on a family of structured q-ary lattices. As explained in Section 6.1, weak f's for PLWE has been investigated in [EHL14, ELOS15, CLS15, CLS16], where attacks were described that work for error distributions with small width relative to the geometry of the corresponding ring [CIV16b, CIV16a, Pei16a]. In another sequence of works, Cramer *et al.* [CDPR16, CDW16] showed that ApproxSVP restricted to ideal lattices corresponding to ideals of the polynomial ring $\mathbb{Z}[x]/f$ is easier to solve quantumly for f a cyclotomic polynomial of prime-power conductor than for general lattices.

We recall that our security reductions show that the security of Titanium is as hard as the hardest PLWE^f over all f in the family $\mathcal{F}$ introduced in the previous Section. In this Section, we explain our security evaluation of best known attacks against PLWE^f under this assumption.

Security of PLWE^f against weak f

The security of PLWE has been investigated in [EHL14, ELOS15, CLS15, CLS16], where attacks were described that work for weak f's and error distributions with small width relative to the geometry of

the corresponding ring [CIV16b, CIV16a, Pei16a]. The polynomials f are reducible mod q if they are weak against the latter attacks. With this in mind, we show that an exponential proportion of functions in $\mathcal{F}$ are irreducible, meaning the explained attacks in [ELOS15] are not applicable to a lot of functions in our considered function family. In order to find a %95 confidence interval, we run an experiment by generating 10000 random polynomials of different degrees belonging to $\mathcal{F}_1$ corresponding to each parameter set. For example for Std128, we generate 200 random polynomials in $\mathcal{F}_1$ with degrees ranging between n to $n-49$ (in total of 10000 random polynomials). We then check the irreducibility of these polynomials and count the number of all irreducible polynomials. Table 6.5 summarizes the number c of irreducible polynomials we counted for each family corresponding to the parameter sets stated in the table. The evaluated probability for each level is also computed in the second row of the Table 6.5. Let $\hat{p} = c/10000$ denotes the observed probability of having irreducible polynomials, the third row of the below table gives the evaluated $\hat{p}$ for different parameter sets. Since our experiment follows a binomial distribution with parameters $(\hat{p}, 10000)$, the %95 binomial confidence interval can be computed as:

$$\hat{p} \pm z^* \sqrt{\hat{p}(1-\hat{p})/10000} \approx c/10000 \pm z^* \cdot (\sqrt{c}/10000),$$

which is justified by CLT by approximating the difference between correct probability p_c and $\hat{p}$ by a Gaussian distribution. Note that z^* for %95 assurance is 1.96.

Table 6.5: Number of irreducible polynomials counted in 10000 random polynomials from $\mathcal{F}_1$.

Parameter	Toy64	Lite96	Std128	Med160	Hi192	Super256
Counted irreducibles	16	12	9	8	4	7
Probability $\hat{p}$	$16/10^4$	$12/10^4$	$9/10^4$	$8/10^4$	$4/10^4$	$7/10^4$
%95 confidence interval ($\times 10^4$)	$[16 \pm 7.8]$	$[12 \pm 6.7]$	$[9 \pm 5.8]$	$[8 \pm 5.5]$	$[4 \pm 3.9]$	$[7 \pm 5.1]$

Since $\log_3(|\mathcal{F}_1|) \geq \mathsf{gap}_1 + \mathsf{gap}_2 = n - (d+k)$, we will be left by at least $3^{n-(d+k)-\log_3(n)}$ irreducible polynomials to which attacks in [EHL14, ELOS15, CLS15, CLS16] are not applicable.

Security of PLWE^f against 'dual' lattice attacks

A recent summary of known attacks against LWE is given in [APS15]. Except for unusual choices of parameters, one of the best known lattice-based attack against LWE is the 'Dual lattice attack', as described also in [ADPS16]. We describe this attack and its (conservative) analysis when applied to $\mathsf{PLWE}^f_{q,t,\chi_{e,\mathsf{P}}}$ with the goal of minimizing the ratio between the attack time to attack distinguishing advantage ratio. We then use those estimates to select parameters for our IND-CPA security of Titanium-CPA-S (see following section for our parameter selection procedure). Our analysis of this attack closely follows the conservative 'Core SVP Hardness' approach introduced in [ADPS16], with some modifications that we explain.

Before describing the details of our approach, we remark that our 'Core SVP hardness' approach for dual lattice attack security estimates for PLWE^f is conservative on several fronts, as follows. Namely, we do *not* put any restriction on the quantum attacker's circuit depth, and we assume a quantumly addressable random access machine computation model with unit time for memory access, as also assumed in previous analyses of quantum SVP sieve algorithms [LMvdP15].

Let $m_d = t \cdot m$. The 'dual lattice attack' consists in rewriting the $\mathsf{PLWE}^f_{q,t,\chi_{e,\mathsf{P}}}$ instance $(a_i, b_i = a_i \cdot s + e_i)_{i \leq t} \in (\mathbb{Z}_q[x]/f \times \mathbb{Z}_q[x]/f)^t$ with $f \in \mathcal{F}_1$ of degree $m \in [m', n]$ as an LWE instance $(\mathsf{A}, \mathbf{b}) \in \mathbb{Z}_q^{m_d \cdot m} \times \mathbb{Z}_q^{m_d}$ over $\mathbb{Z}_q$ (where $\mathbf{b} = \mathsf{A} \cdot \mathbf{s} + \mathbf{e}$ for an 'PLWE input' or $\mathbf{b}$ is uniform for a 'random input'), and using lattice reduction on the 'dual (perp) lattice'

$$L_q^{\perp}(\mathsf{A}) = \{\mathbf{w} \in \mathbb{Z}^{m_d} : \mathbf{w}^T \cdot \mathsf{A} = \mathbf{0} \bmod q.\}$$

to find a short non-zero lattice vector $\mathbf{v}$ of length ℓ in $L_q^{\perp}(\mathsf{A})$. The attack then tries to distinguish the distribution of $z = \mathbf{v}^t \cdot \mathbf{b} \bmod q \in \mathbb{Z}_q$ from uniform on $\mathbb{Z}_q$. In the 'PLWE input' case, we have

$z = \mathbf{v}^t \cdot \mathbf{e}$. Since the dimension of $\mathbf{e}$ is large and the coordinates of $\mathbf{e}$ are independent, we have by the Central Limit Theorem that the distribution of z tends (asymptotically) to a Gaussian with standard deviation $\|\mathbf{v}\| \cdot \alpha q$ and reduced mod q, whereas the in the "random input" case, z is uniformly random over $\mathbb{Z}_q$. Here, we denote by αq the standard deviation of the coordinates of $\mathbf{e}$ sampled from $\chi_{e,\mathsf{MP}}$. For our Titanium setting, we have $\chi_e = \mathsf{BinDiff}(\eta)^{N_{\mathsf{dec}}}$, so αq is the standard deviation of $\mathsf{BinDiff}(\eta)$, namely

$$\alpha q = \sqrt{\frac{\eta}{2}}. \tag{6.27}$$

The maximum distinguishing advantage achievable between the latter distributions of z in the 'PLWE input' and 'random input' cases is their statistical distance, which can be shown to be upper bounded as

$$\varepsilon'_Q \leq \frac{\exp(-2\pi^2\tau^2)}{1-\exp(-2\pi^2\tau^2)} \leq 4 \cdot \exp(-2\pi^2\tau^2), \tag{6.28}$$

where the rightmost inequality holds if

$$\tau \geq \sqrt{\frac{\ln(4/3)}{2\pi}} \approx 0.22, \tag{6.29}$$

with

$$\tau = \|\mathbf{v}\| \cdot \alpha. \tag{6.30}$$

We use the right-hand side as our advantage estimate ε'_Q for the best attack against $\mathsf{PLWE}^f_{q,t,D_{\alpha q}}$.

We assume that the BKZ algorithm [SE94] with block size b (where b is chosen to optimise the attack) is used for lattice reduction to compute the short non-zero vector $\mathbf{v}$ in lattice $L_q^\perp(\mathsf{A})$. The BKZ algorithm calls an exact SVP oracle in dimension b. Currently, the most efficient known exact SVP algorithm in large dimension b appears to be the 'Hypercone filtering' sieve based SVP algorithm of Laarhoven [LMvdP15] (see Chap. 14 therein), an SVP sieve algorithm accelerated by a quantum Grover-based search algorithm. As in standard 'Dual lattice attacks' [MR09], we let the attacker run BKZ on a sublattice of $L_q^\perp(\mathsf{A})$ of dimension $m^* \leq m_d$, where m^* is chosen to minimize the BKZ output vector length $\|\mathbf{v}\|$, which is estimated by

$$\|\mathbf{v}\| = \delta(b)^{m^*-1} \cdot q^{m/m^*}, \tag{6.31}$$

where $\delta(b)$ is the BKZ Hermite Factor (the ratio between the output vector norm and the root determinant of the lattice), which we estimate using the Geometric Series Assumption (GSA) model for the BKZ output Gram-Schmidt norms [Sch03a]. Under the GSA model, the BKZ Hermite Factor can be shown to be [Sch03a].

$$\delta(b) = \left(\frac{b}{2\pi e} \cdot (\pi b)^{1/b}\right)^{\frac{1}{2\cdot(b-1)}}, \tag{6.32}$$

The run-time of the attack is dominated by the time of BKZ with block size b. Following the conservative 'core-hardness' methodology used in [ADPS16, BCD+16], we estimate the BKZ running-time by only counting the time of a single SVP oracle call, which according to [LMvdP15] takes time

$$T_{\mathsf{QBKZ}}(b) = 2^{0.265\cdot b+o(b)} \text{ and } T_{\mathsf{CBKZ}}(b) = 2^{0.292\cdot b+o(b)}, \tag{6.33}$$

respectively, on quantum and classical computing model, respectively. For our attack complexity estimates, we neglect the $o(b)$ term (by setting it to zero). We remark again that the above estimates assume (conservatively) a quantum (resp. classical) Random Access Machine (RAM) model of computation with unlimited quantum (resp. classical) circuit depth and unit time for memory lookup. We also assume (conservatively) (following [ADPS16, LMvdP15]) that each such memory lookup operation can be implemented with one quantum (resp. classical) gate.

Running BKZ on the dual lattice yields a lattice vector of length given by (6.31), and therefore a distinguisher for $\mathsf{PLWE}^f_{q,t,D_{\alpha q}}$ with advantage ε'_Q given by (6.28). We call this basic distinguisher

a 'one-sample' distinguisher. It has a run-time $T_{\mathsf{QBKZ}}(b)$ quantumly (and $T_{\mathsf{CBKZ}}(b)$ classically), and therefore a time to advantage ratio $T_{\mathsf{QBKZ}}(b)/\varepsilon'_Q(b)$ (respectively $T_{\mathsf{CBKZ}}(b)/\varepsilon'_Q(b)$; from here onwards, we only describe the quantum case, but we use the same analysis with T_{QBKZ} replaced with T_{CBKZ} to estimate the classical cost of the attack). Our goal is to consider optimised versions of this attack that amplify the distinguishing advantage by performing the distinguishing test over multiple samples (i.e. multiple short lattice vectors $\mathbf{v}_i$ and corresponding z_i's) by taking a majority vote over the samples tested in order to minimize the the time to advantage ratio of the attack. Namely, using N independent samples, one can amplify the advantage in this way to $\approx \sqrt{N} \cdot \varepsilon'_Q(b)$. In fact, the last sieve SVP call in the BKZ algorithm gives, $M_{\mathsf{QBKZ}}(b)$ short vectors $\mathbf{v}$ (proportional to the memory of the algorithm) that could be used in the distinguishing attack. Following the conservative approach we first replace $M_{\mathsf{QBKZ}}(b)$ by the larger quantity $T_{\mathsf{QBKZ}}(b)$ in [ADPS16, BCD$^+$16], and we also assume heuristically (conservatively, i.e. optimistically for the attacker) that all those $T_{\mathsf{QBKZ}}(b)$ short vectors $\mathbf{v}$ obtained from a single sieve SVP call are 'independent' so they can be used in the distinguishing attack with the same $\mathsf{PLWE}^f_{q,t,D_{\alpha q}}$ instance to amplify the advantage. Let

$$R_{\max}(b, \varepsilon'_Q) = \max\left(1, \frac{1/(\varepsilon'_Q)^2}{T_{\mathsf{QBKZ}}(b)}\right). \tag{6.34}$$

For each BKZ block size b, if the attacker runs BKZ R' times (with $R' \in (1, R_{\max}(b, \varepsilon'_Q))$ he gets $R' \cdot T_{\mathsf{QBKZ}}(b)$ samples which leads to total running-time $T_{\mathsf{dual}} = R' \cdot T_{\mathsf{QBKZ}}(b)$ and distinguishing advantage $\varepsilon_{\mathsf{dual}} \approx \min(\sqrt{R' \cdot T_{\mathsf{QBKZ}}(b)} \cdot \varepsilon'_Q, 1)$. The attacker chooses optimum values of b and R' that minimizes $T_{\mathsf{dual}}/\varepsilon_{\mathsf{dual}}$ (we note that when $R' = R_{\max}$, the advantage is fully amplified to ≈ 1, and therefore there is no gain in using larger R').

There are two possible cases for b. If the case $R_{\max}(b, \varepsilon'_Q) = 1$ holds (equivalently, if $T_{\mathsf{QBKZ}}(b) \geq 1/(\varepsilon'_Q)^2$ or in other words $\varepsilon'_Q\sqrt{T_{\mathsf{QBKZ}}(b)} \geq 1$) then $R' = 1$ and $\varepsilon_{\mathsf{dual}} = 1$ and hence $T_{\mathsf{dual}}/\varepsilon_{\mathsf{dual}} \approx T_{\mathsf{QBKZ}}(b)$. In the other possible case $R_{\max}(b, \varepsilon'_Q) > 1$ (equivalently, if $T_{\mathsf{QBKZ}}(b) < 1/(\varepsilon'_Q)^2$), we have $T_{\mathsf{dual}}/\varepsilon_{\mathsf{dual}} \approx \sqrt{R' \cdot T_{\mathsf{QBKZ}}(b)}/\varepsilon'_Q(b)$ so in this case $R' = 1$ is also the optimal choice that minimizes $T_{\mathsf{dual}}/\varepsilon_{\mathsf{dual}}$ at $\approx \sqrt{T_{\mathsf{QBKZ}}(b)}/\varepsilon'_Q(b)$. The optimum value for b that minimizes $T_{\mathsf{dual}}/\varepsilon_{\mathsf{dual}}$ is therefore the value b^* that minimizes the attack time to advantage ratio function $g_{\mathsf{dual}}(b)$, which is given by:

$$g_{\mathsf{dual}}(b) = \begin{cases} \frac{\sqrt{T_{\mathsf{QBKZ}}(b)}}{\varepsilon'_Q(b)} \text{ for } b < b_{\mathsf{eq}}, \\ T_{\mathsf{QBKZ}}(b) \text{ for } b \geq b_{\mathsf{eq}}, \end{cases} \tag{6.35}$$

where b_{eq} is the value of b such that $\sqrt{T_{\mathsf{QBKZ}}(b)}/\varepsilon'_Q(b) = T_{\mathsf{QBKZ}}(b)$ (equivalently, $1/\varepsilon'_Q(b)^2 = T_{\mathsf{QBKZ}}(b)$). Note that since $T_{\mathsf{QBKZ}}(b)$ increases monotonically with b, we have $b^* \leq b_{\mathsf{eq}}$. Also, b_{eq} is the value of b assumed as the optimal value using a slightly different argument in [ADPS16], where it was assumed that the attacker aims to achieve a high advantage in the overall attack, whereas we do not make this assumption[1]. In our security estimates for parameter set selection, we compute the optimal value of b^* and m^* numerically, and return the PLWE^f attack time to advantage ratio log complexity

$$\lambda_{\mathsf{PLWE}} = \log_2(g_{\mathsf{dual}}(b^*)). \tag{6.36}$$

Note that if we would have taken the non-conservative $M_{\mathsf{QBKZ}}(b)$ short non-zero vectors rather than $T_{\mathsf{QBKZ}}(b)$, we would have a larger g_{dual} by a factor of $\sqrt{T_{\mathsf{QBKZ}}(b)/M_{\mathsf{QBKZ}}(b)}$.

We remark that, as shown in [MR09], for each fixed b, the optimal sublattice dimension m^* that minimizes $\|\mathbf{v}\|$ in (6.31) (and therefore maximizes the distinguishing advantage of the attack) and the corresponding minimum length $\|\mathbf{v}\|_{\mathsf{opt}}$ are given by:

$$m^* \approx \sqrt{\frac{m \log q}{\log \delta}} \text{ and } \|\mathbf{v}\|_{\mathsf{opt}} \approx 2^{2\sqrt{m \log q \log \delta}}. \tag{6.37}$$

[1] However, we remark that in numerical computations of b^* for security estimates of our Titanium parameter sets, we found that $b^* \approx b_{\mathsf{eq}}$ in all cases we considered.

We summarize in Table 6.6 the computed complexities (using the above 'core SVP hardness' methodology) of PLWE instances corresponding to our scheme parameter sets via the reduction of Corollary 6.3.1 (see Sec. 6.3.8). The table also includes, for comparison, the PLWE complexity goals for achieving our target scheme security levels based on our parameter selection approach described in Sec. 6.3.8. A Sagemath [Dev17] script used to compute those values and is also included in our submission package. In Table 6.6, the classical 'goal' column gives $\lambda_{\mathsf{PLWE},C,\mathsf{goal}}$ from (6.62) (for Titanium-CPA) and from (6.69) (for Titanium-CCA). The classical 'Min. Cl.', 'Med. Cl.' and 'Max. Cl.' columns give the corresponding claimed PLWE complexities $\lambda_{C,\mathsf{PLWE},m_{\min}}$, $\lambda_{C,\mathsf{PLWE},m_{\mathsf{med}}}$, and $\lambda_{C,\mathsf{PLWE},m_{\max}}$ for PLWE with dimensions $m_{\min}, m_{\mathsf{med}}, m_{\max}$ corresponding to the minimum, middle and maximum degrees of polynomials in our family $\mathcal{F}_1$ (see Sec. 6.3.8). Similarly, the quantum 'goal' column gives $\lambda_{\mathsf{PLWE},Q,\mathsf{goal}}$ from (6.66) (for Titanium-CPA) and from (6.73) (for Titanium-CCA), assuming $\mathsf{MD} = 2^{40}$ (this corresponds to the largest quantum security goals for MD in the range 2^{40} to 2^{96}). The quantum 'Min. Cl.', 'Med. Cl.' and 'Max. Cl.' columns give the corresponding claimed quantum PLWE complexities $\lambda_{Q,\mathsf{PLWE},m_{\min}}$, $\lambda_{Q,\mathsf{PLWE},m_{\mathsf{med}}}$, and $\lambda_{Q,\mathsf{PLWE},m_{\max}}$ for PLWE with dimensions $m_{\min}, m_{\mathsf{med}}, m_{\max}$ computed based on our dual attack 'core SVP hardness' methodology (which we recall, assumes conservatively, an unlimited quantum circuit depth).

Table 6.6: PLWE classical and quantum security goals and claimed classical and quantum complexities for each scheme parameter set. CCA (resp. CPA) rows refer to Titanium-CCA (resp. Titanium-CPA) scheme.

Par. Set	**Classical**				**Quantum**			
	Goal	Min. Cl.	Med. Cl.	Max. Cl.	Goal	Min. Cl.	Med. Cl.	Max. Cl.
CCA, Toy64	84	85	88	90	71	78	80	83
CPA, Toy64	82	91	94	97	69	84	86	89
CCA, Lite96	116	123	126	129	103	113	115	118
CPA, Lite96	114	127	130	133	101	116	119	122
CCA, Std128	148	149	162	176	135	136	149	161
CPA, Std128	145	171	176	182	133	156	161	166
CCA, Med160	180	195	200	205	167	178	183	187
CPA, Med160	178	201	206	211	165	184	189	194
CCA, Hi192	212	233	238	243	198	214	218	222
CPA, Hi192	210	235	239	244	196	215	219	224
CCA, Super256	277	323	328	333	263	296	300	305
CPA, Super256	275	327	330	333	261	299	302	305

Security of PLWE^f against primal 'embedding attack'

A primal embedding attack will be described in full details in subsubsection 6.4.2. There, we explain a generic LWE attack followed by an attack optimised for MP-LWE. Here, we apply the generic attack based on 'analysis approach 2' to PLWE^f rather than MP-LWE (the optimised attack can only be applied to MP-LWE taking advantage of sparse MP-LWE corresponding matrices). In Table 6.7, the classical 'goal' column gives $\lambda_{\mathsf{PLWE},C,\mathsf{goal}}$ from (6.62) (for Titanium-CPA) and from (6.69) (for Titanium-CCA).

The classical 'Min. At.', 'Med. At.' and 'Max. At.' columns give the corresponding estimated PLWE attack log. time complexities $\lambda_{C,\mathsf{emb},2,\mathsf{PLWE},m_{\min}}$, $\lambda_{C,\mathsf{emb},2,\mathsf{PLWE},m_{\mathsf{med}}}$, and $\lambda_{C,\mathsf{emb},2,\mathsf{PLWE},m_{\max}}$ for PLWE with dimensions $m_{\min}, m_{\mathsf{med}}, m_{\max}$ corresponding to the minimum, middle, and maximum degrees of polynomials in our family $\mathcal{F}_1$ (see Sec. 6.3.8). Similarly, the quantum 'goal' column gives $\lambda_{\mathsf{PLWE},Q,\mathsf{goal}}$ from (6.66) (for Titanium-CPA) and from (6.73) (for Titanium-CCA), assuming $\mathsf{MD} = 2^{40}$

(this corresponds to the largest quantum security goals for MD in the range 2^{40} to 2^{96}). The quantum 'Min. At.', 'Med. At.' and 'Max. At.' columns give the corresponding estimated quantum PLWE log. time complexities $\lambda_{Q,\mathsf{emb},2,\mathsf{PLWE},m_{\min}}$, $\lambda_{Q,\mathsf{emb},2,\mathsf{PLWE},m_{\mathsf{med}}}$, and $\lambda_{Q,\mathsf{emb},2,\mathsf{PLWE},m_{\max}}$ for PLWE with dimensions $m_{\min}$, m_{med}, and $m_{\max}$ computed based on our primal embedding attack.

Table 6.7: The generic PLWE primal embedding (analysis approach 2) key recovery lattice-based attack log expected classical (resp. quantum) complexity estimates for all our parameter sets. Here, attack complexity estimates on scheme parameter sets are denoted by 'Min. At.', 'Med. At.' and 'Max. At.' corresponding to claimed classical/quantum PLWE complexities with respect to with dimensions $m_{\min}, m_{\mathsf{med}}, m_{\max}$.

Par. Set	**Classical**				**Quantum**			
	Goal	Min. At.	Med. At.	Max. At.	Goal	Min. At.	Med. At.	Max. At.
CCA, Toy64	84	89	92	94	71	81	83	86
CPA, Toy64	82	96	99	102	69	87	89	92
CCA, Lite96	116	129	132	134	103	117	120	122
CPA, Lite96	114	133	136	140	101	121	124	127
CCA, Std128	148	155	169	183	135	141	154	166
CPA, Std128	145	178	184	190	133	162	167	172
CCA, Med160	180	202	208	213	167	184	188	193
CPA, Med160	178	209	215	220	165	190	195	200
CCA, Hi192	212	242	247	252	198	220	224	229
CPA, Hi192	210	243	248	253	196	221	225	230
CCA, Super256	277	334	339	344	263	303	308	312
CPA, Super256	275	334	339	344	261	303	308	312

We remark that a comparison of Tables 6.6-6.7 shows that our claimed security levels based on the dual attack are slightly lower than the corresponding estimated log. time complexities those of the primal embedding attack.

Security of PLWE^f against algebraic attacks

Security of the LWE problem (and also PLWE) against algebraic attacks [AG11, ACF+14] is not very well understood. The basic Arora-Ge attack against PLWE in dimension $m \geq m' \geq d + k$ with noise coordinates restricted to $[-\alpha q, \alpha q]$ (this is conservative, as Gaussian noise with s.d. αq will not satisfy this with significant probability) constructs a linearised system of $t \cdot m$ equations (one equation per LWE sample) over $\mathbb{Z}_q$ in $m^{2\alpha q+1}$ variables (originally monomials consisting of products of powers of the LWE secret coordinates in the non-linear algebraic equations). For security level λ_Q against this attack (taking into account Grover-based 'square-root' search acceleration), we can require the number of solutions to this system to be $\geq 2^{2\lambda_Q}$, giving the condition

$$q^{m^{2\alpha q+1}-t\cdot m} \geq 2^{2\lambda_Q}. \tag{6.38}$$

This condition is easy to satisfy even for very small αq (e.g. even with a very large $t = (m)$, the left hand side of (6.38) is $\geq q^{m^{2\alpha q-1}} \geq q^m$ for $\alpha q \geq 1$). However, the basic AG attack can be improved with Gröbner basis techniques[ACF+14] to require less equations at the cost of more computation to generate more equations. More investigation of the Gröbner based attack cost in [ACF+14] is needed. For our parameter estimates, we assume the following condition for security against algebraic attacks, which is consistent with similar choices in [ADPS16, BCD+16]:

$$\alpha q \geq 1.4. \tag{6.39}$$

6.3.8 Summary of parameter selection procedure

In this Section, we summarize our parameter selection procedure based on the security analysis in previous sections.

Definition of Titanium-CPA and Titanium-CCA attack circuit complexity level

We say that an IND-CPA attack A on Titanium-CPA has a quantum (resp. classical) circuit gate complexity 2^{λ_Q} (resp. 2^{λ_C}), if A has running time $1 \leq T_Q \leq 2^{\lambda_Q}$ (measured in number of elementary quantum, resp. classical gate operations) and advantage ε_Q such that $T_Q/\varepsilon_Q = 2^{\lambda_Q}$ (resp. 2^{λ_C}), and we call λ_Q and λ_C the log. complexity levels. Here, we count the cost of computing a response to each query of A to the random oracle XOF as T_{RO} classical gates (resp. T_{QRO} quantum gates). In the quantum (classical) case, we count the depth of the circuit for answering random oracle queries as D_{QRO} quantum (D_{RO} classical) gates.

Similarly, we say that IND-CCA attack A on Titanium-CCA has a quantum (resp. classical) complexity λ_Q (resp. λ_C), if A has running time $1 \leq T_Q \leq 2^{\lambda_Q}$ measured in number of quantum, (resp. classical gates) and advantage ε_Q such that $T_Q/\varepsilon_Q = 2^{\lambda_Q}$ (resp. 2^{λ_C}). In this case, similarly to the Titanium-CPA case, we count the cost of computing a response to each query of A to the random oracles XOF, G, H as taking T_{RO} classical gates (resp. T_{QRO} quantum gates). In the quantum case, we count the depth of the circuit for answering random oracle queries as D_{RO} quantum gates.

Choice of random oracle evaluation cost constants

Concretely, since our scheme instantiates the random oracles XOF, G, H using the SHA-3 SHAKE construction, we set $T_{\mathsf{RO}} \approx 2^{15}$ classical gates as a conservative estimate for the size of the circuit evaluating the random oracles, based on the 40k-50k gate complexity of fast classical hardware implementations of SHA-3 [SGH+13]. Similarly, we set $T_{\mathsf{QRO}} = 2^{19}$ and $D_{\mathsf{QRO}} = 2^{13}$ quantum gates as a conservative estimate for the size and depth, respectively, of the quantum circuit evaluating the random oracles, based on the optimised quantum SHA-3 implementation of [AMG+16].

Parameter sets and goal security levels

As specified in Chapter 2, to illustrate the efficiency-security scalability of Titanium, we generated six Titanium parameter sets called Toy64, Lite96, Std128, Med160, Hi192, Super256, intended to correspond to the brute force key search security level of a symmetric key cipher with key bit lengths $64, 96, 128, 160, 192, 256$, respectively. In particular, the parameter sets Std128, Hi192, Super256 satisfy the security categories 1,3,5 specified by NIST in the call for proposals [NISa] corresponding to security of AES128, AES192 and AES256 against brute force key search attack.

The classical attack gate complexity level goal for the six parameter sets / symmetric-key search security levels, is denoted by λ_C with

$$\lambda_C \in \{79, 111, 145, 175, 207, 272\} \tag{6.40}$$

for parameter sets Toy64, Lite96, Std128, Med160, Hi192, Super256 respectively, corresponding to $\approx 2^{15}$ gates cost for each symmetric-key cipher evaluation, consistent with the specified AES128/AES192/AES256 key search complexity levels specified in [NISa].

Similarly, the quantum attack gate complexity level goal for the six parameter sets / symmetric-key search security levels, is denoted by λ_Q with

$$\lambda_Q \in \{106, 140, 170, 202, 233, 298\} - \log_2(\mathsf{MD}), \tag{6.41}$$

for parameter sets Toy64, Lite96, Std128, Med160, Hi192, Super256 respectively, intended to estimate the circuit gate complexity of quantum key search attacks under the assumption that the quantum attack circuit depth is restricted to MD (the parameter MD denotes MAXDEPTH in [NISa]). These goals are consistent with AES128/AES192/AES256 quantum key search complexity levels specified in [NISa].

Approach: parameter set claimed security levels - classical attacks

In accordance with our aim of basing the security of Titanium on the security of the hardest PLWE^f problem over the large family $\mathcal{F}_1(n, m', d')$ of polynomial rings f, we computed our specified 'claimed classical security levels' for Titanium-CPA and Titanium-CCA as proven lower bounds on *classical* attack complexity on those schemes, based on our security proofs and the *minimum* complexity, over the choice of f from $\mathcal{F}_1(n, m', d')$ (corresponding to the lowest degree m' of polynomials in the family) of the best known attack on PLWE^f (the dual lattice attack described in the previous Section), and assuming the random oracle model for the underlying hash functions $\mathsf{XOF}, \mathsf{G}, \mathsf{H}$.

- *Conservative* PLWE^f *Attack Estimates*: Our parameter selection procedure is conservative, as the best attacks we know of on our Titanium schemes have a somewhat higher complexity than the above lower bounds implied by our security proofs and the best known attacks on PLWE^f (see following Section), even if our conservative 'core SVP hardness' and 'neglecting memory access gate cost' assumptions are taken into account. Moreover, Since the complexity of best known lattice attacks on PLWE^f increases monotonically with the degree m of the polynomial f in the degree interval $[m', n]$ of polynomials in $\mathcal{F}_1$, the hardness of PLWE^f for f of degree higher than m' potentially gives our schemes an extra security margin (assuming there are hard choices of f of degree larger than m'). To quantify this lattice attack security margin, we computed the log complexity levels $\lambda_{\mathsf{PLWE},C,m_{\min}}, \lambda_{\mathsf{PLWE},C,m_{\mathsf{mid}}}, \lambda_{\mathsf{PLWE},C,m_{\max}}$ of the best known lattice attacks on PLWE^f with f of minimum degree m', 'middle' degree $m_{mid} = (m' + n)/2$, and maximum degree n, for the family $\mathcal{F}_1(n, m', d')$ respectively (we explain our choice of n, m', and d' below).

- *Safety Margins*: As a safety margin against future advances in lattice cryptanalysis, in addition to requiring that $\lambda_{\mathsf{PLWE},C,m_{\min}}$ exceeds the desired log complexity goal $\lambda_{\mathsf{PLWE},C,\mathsf{goal}}$ (needed to achieve the desired lower bound λ_C for the scheme attack log complexity level), we imposed the additional constraint on our parameter selection that the middle PLWE complexity level $\lambda_{\mathsf{PLWE},C,m_{\mathsf{mid}}}$ exceeds the desired log complexity goal $\lambda_{\mathsf{PLWE},C,\mathsf{goal}}$ by at least 5%. This implies that our scheme has at least 5% log complexity safety margin, as long as the 'top half' subfamily of $\mathcal{F}_1$ (i.e the polynomials in $\mathcal{F}_1$ of degree $\geq m_{\mathsf{mid}} = (n + m')/2$) contains a polynomial for which the complexity of PLWE^f matches the best known attacks. The log complexity levels $\lambda_{\mathsf{PLWE},C,m_{\min}}, \lambda_{\mathsf{PLWE},C,m_{\mathsf{mid}}}, \lambda_{\mathsf{PLWE},C,m_{\max}}$ and achieved safety margins with respect to the desired log complexity goal $\lambda_{\mathsf{PLWE},C,\mathsf{goal}}$ for each parameter set are tabulated in Table 6.6.

- *Decryption Error Probability* p_e: Both our Titanium-CPA and Titanium-CCA schemes have a non-zero decryption failure probability p_e, over the choice of randomness in key generation *and* randomness in encryption (see Chapter 7 for proven upper bounds on this probability for our parameter sets). However, our choice of p_e is different in Titanium-CPA and Titanium-CCA.

  For Titanium-CCA, the choice of p_e is dictated by the IND-CCA security requirement. It is well known [HNP+03] that the decryption algorithm of lattice-based encryption schemes tends to be vulnerable to decryption failure attacks which extract information on the secret key by querying the decryption oracle on a set of ciphertexts that are likely to cause a decryption failure. Indeed, our Titanium-CCA scheme also potentially suffers from such attacks if parameters are set such that p_e is non-negligible. To thwart such decryption-failure CCA attacks, we choose our Titanium-CCA parameters to make p_e cryptographically small (we have $p_e < 2^{-85}$ for all our parameter sets; the exact goal for p_e is given in Tables 2.12–2.13, so that it is infeasible for the attacker to find an encryption randomness that will cause a decryption failure for the corresponding ciphertext encrypted with the encryption randomness.

  For Titanium-CPA, p_e has functional implications on applications of the encryption scheme (e.g. in key exchange, which would require a protocol restart in the rare case of a decryption failure), but p_e has no security implications in the IND-CPA attack model. Consequently, for Titanium-CPA, we chose an error probability that is extremely rare such that it is likely to

have a negligible performance effect in practice ($< 2^{-30}$, i.e. less than 1 in a Billion), but not negligible in the cryptographic security sense (a similar approach was taken in the design of other IND-CPA lattice-based encryption schemes designed for authenticated key exchange protocols, such as Frodo [BCD+16]). By allowing an extremely rare but non-negligible p_e for Titanium-CPA, we are able to improve the efficiency of our parameter sets for Titanium-CPA over those for Titanium-CCA at comparable security levels.

A summary of the claimed classical security level bounds $\lambda_{Q,\mathsf{CCA},Cl}$ and $\lambda_{Q,\mathsf{CPA},Cl}$ for Titanium-CCA and Titanium-CPA based on the above approach are given in (6.61) and (6.51), respectively, and their computed values for each parameter set is shown in the 'Min. Cl.' columns of Table 2.11 in Chapter 2, along with (for comparison) the scheme security goal λ_Q from (6.41). Computed values and comparison with corresponding goals of PLWE security levels are given in Table 6.6, and other terms in the security bounds for the parameter sets are given in Tables 2.12–2.13 in Chapter 2 and Table 6.8.

A summary of the claimed classical security level bounds $\lambda_{C,\mathsf{CCA},Cl}$ and $\lambda_{C,\mathsf{CPA},Cl}$ for Titanium-CCA and Titanium-CPA based on the above approach are given in (6.57) and (6.48), respectively, and their computed values for each parameter set is shown in the 'Min. Cl.' columns of Table 2.11 in Chapter 2, along with (for comparison) the scheme security goal λ_C from (6.40). Computed values and comparison with corresponding goals of PLWE security levels are given in Table 6.6, and other terms in the security bounds for the parameter sets are given in Tables 2.12–2.13 in Chapter 2 and Table 6.8.

Approach: parameter set claimed security levels - quantum attacks

Unlike the classical security proofs, existing security proofs for the IND-CCA security of the Fujisaki-Okamoto transform and its variants in the quantum random oracle model [HHK17, TU16] incur a significant reduction cost. The tightness of these quantum security proofs is currently not well understood. Some of the larger terms in the quantum reduction cost (compared to the classical reduction) are unavoidable and due to 'Grover-type' brute-force quantum search attacks against the scheme (see below), but other reduction costs seem to be an artifice of existing proof techniques, and we do not know of quantum attacks that exploit those costs.

Instead of relying on (probably) non-tight quantum security reductions, our estimated claimed *quantum* attack complexity is based on the additional (currently heuristic and unproven) assumption. Our assumption is that the classical security proof bounds of Lemma 6.3.1, Lemma 6.3.3 and Lemma 6.3.4 also apply in the quantum attack setting (a similar assumption is made in the parameter selection of [BDK+17]), with the following modification to account for the success probability of 'Grover-type' brute force quantum search attacks. Each probability term of the form $p_C = Q \cdot \delta$ in the classical security reduction bound, where here, Q denotes the number of queries Q_{XOF}, Q_{G}, Q_{H} to XOF, G or H, respectively, and δ denotes a probability (i.e. either $\delta = 1/2^{256}$, representing the event of querying a random oracle at some 'bad' point, or $\delta = p_e$, representing the event of querying the random oracle G of Titanium-CCA at a message that maps to randomness causing a decryption failure), is replaced by a term of the form

$$p_Q = 8 \cdot (Q/Q_D) \cdot (Q_D + 1)^2 \cdot \delta = (1 + 1/Q_D) \cdot Q \cdot (Q_D + 1) \cdot \delta \approx Q \cdot Q_D \cdot \delta \tag{6.42}$$

in the corresponding quantum bound, where

$$Q_D = \mathsf{MD}'/\mathsf{D}_{\mathsf{QRO}}, \tag{6.43}$$

$$\mathsf{MD}' = \min(\mathsf{MD}, 2^{\lambda_Q}), \tag{6.44}$$

is the maximal gate depth allowed for attacks of total gate complexity $\leq 2^{\lambda_Q}$ and gate depth $\leq \mathsf{MD}$, and $\mathsf{D}_{\mathsf{QRO}}$ is the quantum circuit depth of the hash function instantiating the random oracles (i.e. SHA-3 in SHAKE mode). This assumption requires further study, and we leave this and a possible proof for future work.

The heuristic assumption above is motivated by (an adaptation of) 'Grover-type' lower bounds on the query complexity of the quantum 'Generic Search Problem' [HHK17, HRS16]. In particular, by Lemma 6.2.3, if F denotes a function maps each $x \in X$ (where X denotes the domain of F) to $F(x)$, where $F(x) \in \{0,1\}$ is independently chosen for each x as a Bernoulli random variable with $\Pr[F(x) = 1] \leq \delta$, where $F(x) = 1$ represents a 'bad' x event, then any quantum circuit making Q queries to a quantum circuit implementing F, will return a 'bad' x such that $F(x) = 1$ with probability at most $p_Q \leq 8{\cdot}(Q{+}1)^2{\cdot}\delta$. In our setting, to obtain (6.42), we modified this estimate for p to account for the MD quantum circuit depth restriction. Namely, we assume that the quantum circuit C maximizing p_Q under the MD quantum circuit depth restriction (where evaluating the quantum oracle for each query is assumed to cost $\mathsf{D_{QRO}}$ gates of circuit depth), runs in parallel Q/Q_D sequential circuits C'_i $(i = 1, \ldots, Q/Q_D)$ each of depth MD making $Q_D = \mathsf{MD}/\mathsf{D_{QRO}}$ oracle queries (as Q_D is the maximal number of sequential queries C' can make under the MD restriction), and C returns a 'bad' x. By Lemma 6.2.3 each run of C'_i succeeds to find a 'bad' x with probability at most $8 \cdot (Q_D + 1)^2 \cdot \delta$, and hence, by the union bound over the Q/Q_D circuits C'_i, C succeeds to find a 'bad' x with probability at most $p_Q \leq (Q/Q_D) \cdot 8 \cdot (Q_D + 1)^2 \cdot \delta$, which gives our assumed estimate in (6.42).

We note that for the random oracles $\mathsf{XOF}, \mathsf{G}, \mathsf{H}$, the quantum brute-force query complexity quantum term p_Q corresponds approximately to an estimate for the success probability of the best known Grover brute force attacks involving Q queries to the quantum random oracle, under the restriction of at most MD quantum circuit depth, as summarized in the following Section.

Based on the above assumption, we compute the claimed quantum security estimates of our parameter sets similarly to the classical case, except that we evaluate the claimed security under three possible assumptions on the attacker's maximum allowed quantum circuit depth MD, namely $\mathsf{MD} \in \{2^{40}, 2^{64}, 2^{96}\}$, suggested in [NISa] as examples of moderate, large and extreme (potentially infeasible) estimates, respectively, on the power of practical quantum architectures.

We remark that, in line with our conservative assumptions for the complexity of lattice attacks, we do not actually make the MD depth restriction in estimating the quantum complexity of the best known attacks against the PLWE^f problem; we allow unrestricted depth quantum circuits for attacking PLWE^f, though we do make the MD restriction for estimating the complexity of Grover attacks on the scheme (see following section). Based on the BKZ attack with a quantum SVP sieve subroutine as analysed in the previous section, we compute the quantum complexities $\lambda_{\mathsf{PLWE},Q,m_{\min}}$, $\lambda_{\mathsf{PLWE},Q,m_{\mathsf{mid}}}$, $\lambda_{\mathsf{PLWE},Q,m_{\max}}$ of the best known lattice attacks on PLWE^f with f of minimum degree m', 'middle' degree $m_{mid} = (m' + n)/2$, and maximum degree n, for the family $\mathcal{F}_1(n, m', d')$ respectively.

For our safety margin against future advances in lattice cryptanalysis, in addition to requiring that $\lambda_{\mathsf{PLWE},Q,m_{\min}}$ exceeds the desired quantum log complexity goal $\lambda_{\mathsf{PLWE},Q,\mathsf{goal}}$ (needed to achieve the desired lower bound λ_Q for the scheme attack log complexity level), we imposed the additional constraint on our parameter selection that the middle PLWE complexity level $\lambda_{\mathsf{PLWE},Q,m_{\mathsf{mid}}}$ exceeds the desired log complexity goal $\lambda_{\mathsf{PLWE},Q,\mathsf{goal}}$ by at least 10%. We chose a larger safety margin for quantum attacks than for classical attacks because the quantum complexity of lattice problems appears less well understood than the classical complexity.

We also remark that, for the usages of the XOF as a pseudorandom generator with secret seeds (i.e. expanding seedkg to $(\mathsf{seedsk}, \mathsf{seedpk})$, seedsk to $\widehat{\mathbf{s}}$ and $(e_1, \ldots, e_t)$, and seedr to $(r_1, \ldots, r_t)$), the security can be easily proven with respect to the pseudorandomness of XOF in the standard model, and in that case, the terms p_Q corresponding to XOF queries can be replaced by the advantage of the best known quantum distinguishing attack on XOF as a pseudorandom generator. Our heuristic assumption then corresponds to assuming that the best quantum distinguishing attack is the Grover brute-force search for the seed under the MD quantum circuit depth restriction.

A summary of the claimed quantum security level bounds $\lambda_{Q,\mathsf{CCA},Cl}$ and $\lambda_{Q,\mathsf{CPA},Cl}$ for $\mathsf{Titanium\text{-}CCA}$ and $\mathsf{Titanium\text{-}CPA}$ based on the above approach are given in (6.61) and (6.51), respectively, and their computed values for each parameter set is shown in the 'Min. Cl.' columns of Table 2.11 in Chapter 2, along with (for comparison) the scheme security goal λ_Q from (6.41). Computed values and comparison with corresponding goals of PLWE security levels are given in Table 6.6, and other terms in the security

bounds for the parameter sets are given in Table 2.13 in Chapter 2 and Table 6.8.

Composing security proofs

As summarized above, we compose the security proof conditions from previous sections to derive conditions on the classical PLWE attack log circuit complexity $\lambda_{C,\mathsf{PLWE}}$ and on parameters Δ_{LHL} and p_e that are needed to achieve a lower bound $\lambda_{C,LB} \geq \lambda_C$ on the log circuit complexity of classical attacks against Titanium-CPA and Titanium-CCA. We then apply our heuristic assumption explained above to modify the classical lower bounds and obtain our claimed quantum complexity estimates.

Classical attack complexity lower bounds: Titanium-CPA

By composing the reductions in Lemma 6.3.1, Lemma 6.3.2 and Corollary 6.3.1, we conclude that any (T, ε) attack A against IND-CPA of Titanium-CPA making Q_{XOF} queries to the random oracle XOF implies a (T', ε') attack B against $\mathsf{PLWE}^{f}_{q,t,\chi_c^{d'}}$ for any $f \in \mathcal{F}_1(n, m', d')$ with

$$T \approx T' \text{ and } \varepsilon \leq 2 \cdot (\varepsilon' + \Delta_{\mathsf{LHL}}) + 3 \cdot Q_{\mathsf{XOF}}/2^{256}. \tag{6.45}$$

From (6.45), we obtain the following lower bound on the classical circuit complexity security level of Titanium-CPA:

$$2^{-\lambda_{C,\mathsf{CPA},Cl}} \stackrel{\text{def}}{=} \frac{\varepsilon}{T} \leq 2 \cdot \frac{\varepsilon^{'}}{T'} + 2 \cdot \frac{\Delta_{\mathsf{LHL}}}{T} + \frac{3 \cdot Q_{\mathsf{XOF}}}{T \cdot 2^{256}}. \tag{6.46}$$

To use (6.46) for our classical claimed security level estimates, we use the additional fact that

$$T \geq Q_{\mathsf{XOF}} \cdot T_{\mathsf{RO}}, \tag{6.47}$$

where T_{RO} is the total number of gates in a circuit for evaluating the random oracle XOF, so the third term in (6.46) is bounded as $\frac{3 \cdot Q_{\mathsf{XOF}}}{T \cdot 2^{256}} \leq \frac{3}{T_{\mathsf{RO}} \cdot 2^{256}}$. We also upper bound the first two terms in (6.46) as follows. The first term is upper bounded as $6 \cdot \varepsilon^{'}/T' = 6 \cdot 2^{-\lambda_{\mathsf{PLWE},C}}$, where $2^{\lambda_{\mathsf{PLWE},C}}$ is our conservative[2] circuit complexity (time to advantage ratio) lower bound estimate of the best known classical attack against $\mathsf{PLWE}^{f}_{q,t,\chi_c^{d'}}$ for any $f \in \mathcal{F}_1(n, m', d')$ (see previous Section). The second term is upper bounded as $6 \cdot \Delta_{\mathsf{LHL}}/T^{'} \leq 6 \cdot \Delta_{\mathsf{LHL}}$. This finally gives our classical claimed (and proven lower bound) security level estimate for Titanium-CPA:

$$2^{-\lambda_{C,\mathsf{CPA},Cl}} \leq 2 \cdot 2^{-\lambda_{\mathsf{PLWE},C}} + 2 \cdot \Delta_{\mathsf{LHL}} + \frac{3}{T_{\mathsf{RO}} \cdot 2^{256}}. \tag{6.48}$$

Quantum claimed attack complexity: Titanium-CPA

Applying our heuristic approach described above for deriving our claimed quantum attack complexity estimate from the classical lower bound (6.46), we get the following claimed quantum circuit complexity of Titanium-CPA:

$$2^{-\lambda_{Q,\mathsf{CPA},Cl}} \stackrel{\text{def}}{=} \frac{\varepsilon}{T} \leq 2 \cdot \frac{\varepsilon^{'}}{T'} + 2 \cdot \frac{\Delta_{\mathsf{LHL}}}{T} + \frac{3 \cdot 8 \cdot Q_{\mathsf{XOF}} \cdot Q_D}{T \cdot 2^{256}}, \tag{6.49}$$

where Q_D is given by (6.43). To use (6.49) for our quantum claimed security level estimates, we use the additional fact that

$$T \geq Q_{\mathsf{XOF}} \cdot T_{\mathsf{QRO}}, \tag{6.50}$$

where T_{QRO} is the total number of gates in a quantum circuit for evaluating the random oracle XOF, so the third term in (6.49) is bounded as $\frac{3 \cdot 8 \cdot Q_{\mathsf{XOF}}}{T \cdot 2^{256}} \leq \frac{24 \cdot Q_D}{T_{\mathsf{QRO}} \cdot 2^{256}}$. We also upper bound the first two terms on the right-hand side of (6.49) as follows. The first term is upper bounded as $2 \cdot \varepsilon^{'}/T^{'} = 2 \cdot 2^{-\lambda_{\mathsf{PLWE},Q}}$,

[2] We recall that we conservatively lower bound the circuit complexity of this attack by the time plus memory of the best known attack on a random access machine.

where $2^{\lambda_{\mathsf{PLWE},Q}}$ is our conservative circuit complexity (time to advantage ratio) lower bound estimate of the best known quantum attack against $\mathsf{PLWE}^{f}_{q,t,\chi_c^{d'}}$ for any $f \in \mathcal{F}_1(n, m', d')$ (see previous Section). The second term is upper bounded as $2 \cdot \Delta_{\mathsf{LHL}}/T^{'} \leq 2 \cdot \Delta_{\mathsf{LHL}}$. This finally gives our classical claimed (and proven lower bound) security level estimate for Titanium-CPA:

$$2^{-\lambda_{Q,\mathsf{CPA},Cl}} \leq 2 \cdot 2^{-\lambda_{\mathsf{PLWE},Q}} + 2 \cdot \Delta_{\mathsf{LHL}} + \frac{24 \cdot Q_D}{T_{\mathsf{QRO}} \cdot 2^{256}}. \tag{6.51}$$

Classical attack complexity lower bounds: Titanium-CCA.

By composing the reductions in Lemma 6.3.3 and Lemma 6.3.4, we conclude that any (T'', ε'') attack A'' against IND-CCA of Titanium-CCA making $Q_{\mathsf{XOF}}, Q_{\mathsf{G}}, Q_{\mathsf{H}}$ queries to $\mathsf{XOF}, \mathsf{G}, \mathsf{H}$ respectively, implies a (T, ε) attack A against IND-CPA of Titanium-CPA with

$$T'' \approx T \text{ and } \varepsilon'' \leq 3 \cdot \varepsilon + \frac{Q_{\mathsf{XOF}} + 2 \cdot Q_{\mathsf{G}} + Q_{\mathsf{H}} + 1}{2^{256}} + Q_{\mathsf{G}} \cdot p_e. \tag{6.52}$$

Composing (6.52) with (6.45) we get that any (T, ε) attack A against IND-CPA of Titanium-CPA making Q_{XOF} queries to the random oracle XOF implies a $(T^{(3)}, \varepsilon^{(3)})$ attack $\mathsf{B}^{(3)}$ against $\mathsf{PLWE}^{f}_{q,t,\chi_c^{d'}}$ for any $f \in \mathcal{F}_1(n, m', d')$ with

$$T \approx T^{(3)} \text{ and } \varepsilon \leq 6 \cdot (\varepsilon^{(3)} + \Delta_{\mathsf{LHL}}) + \frac{10 \cdot Q_{\mathsf{XOF}} + 2 \cdot Q_{\mathsf{G}} + Q_{\mathsf{H}} + 1}{2^{256}} + Q_{\mathsf{G}} \cdot p_e. \tag{6.53}$$

From (6.53), we obtain the following lower bound on the classical circuit complexity security level of Titanium-CCA:

$$2^{-\lambda_{C,\mathsf{CCA},Cl}} \stackrel{\text{def}}{=} \frac{\varepsilon}{T} \leq 6 \cdot \frac{\varepsilon^{(3)}}{T^{(3)}} + 6 \cdot \frac{\Delta_{\mathsf{LHL}}}{T^{(3)}} + (Q_{\mathsf{G}}/T) \cdot p_e + \frac{10 \cdot (Q_{\mathsf{XOF}} + Q_{\mathsf{G}} + Q_{\mathsf{H}}) + 1}{T \cdot 2^{256}}. \tag{6.54}$$

To use (6.54) for our classical claimed security level estimates, we use the additional fact that

$$T \geq (Q_{\mathsf{XOF}} + Q_{\mathsf{G}} + Q_{\mathsf{H}}) \cdot T_{\mathsf{RO}}, \tag{6.55}$$

where T_{RO} is the total number of gates in a circuit for evaluating each of the random oracles $\mathsf{XOF}, \mathsf{G}, \mathsf{H}$, and since may assume $Q_{\mathsf{XOF}} + \cdot Q_{\mathsf{G}} + Q_{\mathsf{H}} \geq 1$ (since otherwise the attacker cannot query the random oracles and has advantage zero), we conclude that the fourth term in (6.54) is bounded as

$$\frac{10 \cdot (Q_{\mathsf{XOF}} + Q_{\mathsf{G}} + Q_{\mathsf{H}}) + 1}{T \cdot 2^{256}} \leq \frac{20}{T_{\mathsf{RO}} \cdot 2^{256}}. \tag{6.56}$$

We also upper bound the first three terms in (6.54) as follows. The third term in (6.54) is upper bounded as $(Q_{\mathsf{G}}/T) \cdot p_e \leq p_e/T_{\mathsf{RO}}$ using (6.55). The first term is upper bounded as $6 \cdot \frac{\varepsilon^{(3)}}{T^{(3)}} = 6 \cdot 2^{-\lambda_{\mathsf{PLWE},C}}$, where $2^{\lambda_{\mathsf{PLWE},C}}$ is our conservative[3] circuit complexity (time to advantage ratio) lower bound estimate of the best known classical attack against $\mathsf{PLWE}^{f}_{q,t,\chi_c^{d'}}$ for any $f \in \mathcal{F}_1(n, m', d')$ (see previous Section). The second term is upper bounded as $6 \cdot \frac{\Delta_{\mathsf{LHL}}}{T^{(3)}} \leq 6 \cdot \Delta_{\mathsf{LHL}}$. This finally gives our classical claimed (and proven lower bound) security level estimate for Titanium-CCA:

$$2^{-\lambda_{C,\mathsf{CCA},Cl}} \leq 6 \cdot 2^{-\lambda_{\mathsf{PLWE},C}} + 6 \cdot \Delta_{\mathsf{LHL}} + \frac{p_e}{T_{\mathsf{RO}}} + \frac{20}{T_{\mathsf{RO}} \cdot 2^{256}}. \tag{6.57}$$

[3]We recall that we conservatively lower bound the circuit complexity of this attack by the time plus memory of the best known attack on a random access machine.

Quantum claimed attack complexity: Titanium-CCA

Applying our heuristic approach described above for deriving our claimed quantum attack complexity estimate from the classical lower bound (6.54), we get the following claimed quantum circuit complexity of Titanium-CCA:

$$2^{-\lambda_{C,\mathsf{CCA},Cl}} \stackrel{\text{def}}{=} \frac{\varepsilon}{T} \leq 6\cdot\frac{\varepsilon^{(3)}}{T^{(3)}} + 6\cdot\frac{\Delta_{\mathsf{LHL}}}{T^{(3)}} + (8\cdot Q_{\mathsf{G}}\cdot Q_D/T)\cdot p_e + \frac{10\cdot 8\cdot Q_D\cdot(Q_{\mathsf{XOF}}+Q_{\mathsf{G}}+Q_{\mathsf{H}})+1}{T\cdot 2^{256}}, \quad (6.58)$$

where Q_D is given by (6.43). To use (6.58) for our quantum claimed security level estimates, we use the additional fact that

$$T \geq (Q_{\mathsf{XOF}}+Q_{\mathsf{G}}+Q_{\mathsf{H}})\cdot T_{\mathsf{QRO}}, \quad (6.59)$$

where T_{QRO} is the total number of gates in a quantum circuit for evaluating the random oracles $\mathsf{XOF}, \mathsf{G}, \mathsf{H}$, and since may assume $Q_{\mathsf{XOF}} + \cdot Q_{\mathsf{G}} + Q_{\mathsf{H}} \geq 1$ (since otherwise the attacker cannot query the random oracles and has advantage zero), we conclude that the fourth term in (6.58) is bounded as

$$\frac{10\cdot 8\cdot Q_D\cdot(Q_{\mathsf{XOF}}+Q_{\mathsf{G}}+Q_{\mathsf{H}})+1}{T\cdot 2^{256}} \leq \frac{160\cdot Q_D}{T_{\mathsf{QRO}}\cdot 2^{256}}. \quad (6.60)$$

We also upper bound the first three terms in (6.58) as follows. The third term in (6.58) is upper bounded as $(8\cdot Q_{\mathsf{G}}\cdot Q_D/T)\cdot p_e \leq \frac{8\cdot Q_D}{T_{\mathsf{QRO}}}\cdot p_e$ using (6.59). The first term is upper bounded as $6\cdot\varepsilon^{(3)}/T^{(3)} = 6\cdot 2^{-\lambda_{\mathsf{PLWE},Q}}$, where $2^{\lambda_{\mathsf{PLWE},Q}}$ is our conservative lower bound estimate of the best known quantum attack against $\mathsf{PLWE}^f_{q,t,\chi_c^{d'}}$ for any $f \in \mathcal{F}_1(n,m',d')$ (see previous Section). The second term is upper bounded as $6\cdot\frac{\Delta_{\mathsf{LHL}}}{T^{(3)}} \leq 6\cdot\Delta_{\mathsf{LHL}}$. This finally gives our quantum claimed security level estimate for Titanium-CCA:

$$2^{-\lambda_{Q,\mathsf{CCA},Cl}} \leq 6\cdot 2^{-\lambda_{\mathsf{PLWE},Q}} + 6\cdot\Delta_{\mathsf{LHL}} + \frac{8\cdot Q_D}{T_{\mathsf{QRO}}}\cdot p_e + \frac{160\cdot Q_D}{T_{\mathsf{QRO}}\cdot 2^{256}}. \quad (6.61)$$

Titanium-CPA **parameter goals: classical attacks**

To satisfy our 2^{λ_C} classical attack complexity lower bound claims for Titanium-CPA, we set parameters such that the three terms on the right-hand side of (6.48) are each at most $\frac{1}{3}\cdot 2^{-\lambda_C}$, i.e. such that the following conditions hold:

$$\lambda_{\mathsf{PLWE},C} \geq \lambda_C + \log_2(6) \stackrel{\text{def}}{=} \lambda_{\mathsf{PLWE},C,\mathsf{goal}}, \quad (6.62)$$

$$\log_2(\Delta_{\mathsf{LHL}}^{-1}) \geq \lambda_C + \log_2(6) \stackrel{\text{def}}{=} \lambda_{\mathsf{LHL},C,\mathsf{goal}}, \quad (6.63)$$

$$\frac{3}{T_{\mathsf{RO}}\cdot 2^{256}} \leq \frac{1}{3}\cdot 2^{-\lambda_C}, \quad (6.64)$$

and satisfying the decryption error probability *functionality* requirement:

$$\log_2(p_e^{-1}) \geq 30. \quad (6.65)$$

Titanium-CPA **parameter goals: quantum attacks**

To satisfy our 2^{λ_Q} quantum attack complexity claims for Titanium-CPA, we also set parameters such that the three terms on the right-hand side of (6.51) are each at most $\frac{1}{3}\cdot 2^{-\lambda_Q}$, i.e. such that the following conditions hold:

$$\lambda_{\mathsf{PLWE},Q} \geq \lambda_Q + \log_2(6) \stackrel{\text{def}}{=} \lambda_{\mathsf{PLWE},Q,\mathsf{goal}}, \quad (6.66)$$

$$\log_2(\Delta_{\mathsf{LHL}}^{-1}) \geq \lambda_Q + \log_2(6) \stackrel{\text{def}}{=} \lambda_{\mathsf{LHL},Q,\mathsf{goal}}, \quad (6.67)$$

$$\frac{24\cdot Q_D}{T_{\mathsf{QRO}}\cdot 2^{256}} \leq \frac{1}{3}\cdot 2^{-\lambda_Q}. \quad (6.68)$$

If the goals (6.62) to (6.68) are satisfied, then (6.48) (resp. (6.51)) implies that any classical (resp. quantum) attack on Titanium-CPA will have circuit complexity (advantage to time ratio) less than $2^{-\lambda_C}$ (resp. $2^{-\lambda_Q}$), as required to achieve a λ_C (resp. λ_Q) attack complexity level. We achieve all these goals for all our parameter sets except the Super256 set. For the Super256, the circuit complexity goal corresponds to '256-bit' AES key search security. This is not achieved by our lower bound because goal (6.64) for classical attacks (resp. (6.68) for quantum attacks) is not satisfied, but the other goals are still satisfied. Informally, this is because we are using several 256-bit symmetric key primitives, so our provable security level for the scheme is always slightly lower than 256-bit key search security. However, we do not view this as a problem, since it is only a very small deviation from the goal that is (at least partially) due to looseness of our bounds, and to the use of several primitives in our scheme (a similar phenomenon occurs when evaluating the security of a symmetric key encryption scheme based on several 256-bit primitives; the security of the encryption scheme will generally be slightly worse than that of the primitives). Since all the other three security goals (related to PLWE, LHL and decryption errors) are satisfied, this issue with the Super256 parameter set could have been circumvented if required by substituting SHA-3-384 in place of SHA-3-256. However, we do not believe the slightly lower claimed security level for the Super256 parameter set warrants the overheads of this change.

Titanium-CCA parameter goals: classical attacks

To satisfy our 2^{λ_C} classical attack complexity lower bound claims for Titanium-CCA, we set parameters such that the four terms on the right-hand side of (6.57) are each at most $\frac{1}{4} \cdot 2^{-\lambda_C}$, i.e. such that the following conditions hold:

$$\lambda_{\mathsf{PLWE},C} \geq \lambda_C + \log_2(24) \stackrel{\text{def}}{=} \lambda_{\mathsf{PLWE},C,\mathsf{goal}}, \tag{6.69}$$

$$\log_2(\Delta_{\mathsf{LHL}}^{-1}) \geq \lambda_C + \log_2(24) \stackrel{\text{def}}{=} \lambda_{\mathsf{LHL},\mathsf{goal}}, \tag{6.70}$$

$$\log_2(p_e^{-1}) \geq \lambda_C - (\log_2(T_{\mathsf{RO}}) - 2), \tag{6.71}$$

and

$$\frac{20}{T_{\mathsf{RO}} \cdot 2^{256}} \leq \frac{1}{4} \cdot 2^{-\lambda_C}. \tag{6.72}$$

Titanium-CCA parameter goals: quantum attacks

To satisfy our 2^{λ_Q} quantum attack complexity claims for Titanium-CCA, we also set parameters such that the four terms on the right-hand side of (6.61) are each at most $\frac{1}{4} \cdot 2^{-\lambda_Q}$, i.e. such that the following conditions hold:

$$\lambda_{\mathsf{PLWE},Q} \geq \lambda_Q + \log_2(24) \stackrel{\text{def}}{=} \lambda_{\mathsf{PLWE},Q,\mathsf{goal}}, \tag{6.73}$$

$$\log_2(\Delta_{\mathsf{LHL}}^{-1}) \geq \lambda_Q + \log_2(24) \stackrel{\text{def}}{=} \lambda_{\mathsf{LHL},Q,\mathsf{goal}}, \tag{6.74}$$

$$\log_2(p_e^{-1}) \geq \lambda_Q - (\log_2(T_{\mathsf{QRO}}) - 5 - \log_2(Q_D)), \tag{6.75}$$

and

$$\frac{160 \cdot Q_D}{T_{\mathsf{QRO}} \cdot 2^{256}} \leq \frac{1}{4} \cdot 2^{-\lambda_Q}. \tag{6.76}$$

If the goals (6.62) to (6.68) are satisfied, then (6.48) (resp. (6.51)) implies that any classical (resp. quantum) attack on Titanium-CCA will have circuit complexity (advantage to time ratio) less than $2^{-\lambda_C}$ (resp. $2^{-\lambda_Q}$), as required to achieve a λ_C (resp. λ_Q) attack complexity level. We achieve all these goals for all our parameter sets except the Super256 set. The small deviation from the goal for the Super256 set is for the same reasons as for Titanium-CPA (see discussion above), and we also do not view it as a problem for the same reasons.

Table 6.8: LHL log inverse statistical distance ($\log_2(\Delta_{\mathsf{LHL}}^{-1})$) goal and achieved values for each scheme parameter set. 'Goal' value is $\lambda_{\mathsf{LHL},C,\mathsf{goal}}$ from (6.70), and achieved value 'Ach.' is the value on the left-hand side of (6.9). CCA (resp. CPA) rows refer to Titanium-CCA (resp. Titanium-CPA) scheme.

Par. Set	Goal	Ach.
CCA, Toy64	84	96
CPA, Toy64	82	95
CCA, Lite96	116	132
CPA, Lite96	114	131
CCA, Std128	148	166
CPA, Std128	146	165
CCA, Med160	180	202
CPA, Med160	178	201
CCA, Hi192	212	245
CPA, Hi192	210	249
CCA, Super256	277	323
CPA, Super256	275	327

Parameter selection procedure

We summarize the main aspects of our parameter selection procedure based on the goals and security evaluation above:

- Fix $p = 2$ and $d = 256$ to support 256 bit plaintexts.
- Fix n, and $k+1 < n$ at multiples of 256 (or slightly smaller), integer t and η satisfying algebraic attack constraint in Eq. (6.39) (revised if conditions below cannot be satisfied).
- Determine NTT dimensions $\mathsf{d}_1, \mathsf{d}_2, \mathsf{d}_3$ as a multiple of 256 (to support our Cooley-Tukey fast NTT implementation, see Chap. 3):
  - Let $\beta_1 = \lceil (d+k)/256 \rceil$ and $\mathsf{d}_1 = \beta_1 \cdot 256$.
  - Let $\beta_2 = \lceil (n+k)/256 \rceil$ and $\mathsf{d}_2 = \beta_2 \cdot 256$.
  - Let $\beta_3 = \lceil (n+d+k-1)/256 \rceil$ and $\mathsf{d}_3 = \beta_3 \cdot 256$.
- Pick the smallest q and a b_{LHL} satisfying (for $\mathsf{cmp} = 0$):
  - NTT constraint: $q = 1 \pmod{l \cdot 256}$, where $l = \mathrm{lcm}(\beta_1, \beta_2, \beta_3)$.
  - Leftover hash Lemma (LHL) constraint: $t \geq t_{\mathsf{LO}} + 0.01$, where t_{LO} is the LHL-based lower bound on t in right-hand side of (6.4), where we set Δ_{LHL} according to (6.70) above.
  - p_e constraint: Upper bound on decryption error probability p_e bounded using right-hand side of inequality (5.18) in Chapter 7 is less than the right-hand side of (6.71) above with a 5% safety margin.
- Let $b_1 = \lfloor b_{\mathsf{LHL}} \rfloor - 1$ and $b_2 = b_1 + 1$ and compute $N_{\mathsf{dec1}} \in \mathbb{Z}$ such that Eq. (6.5) is satisfied.
- Let $\lambda_{\mathsf{PLWE},C}$ and $\lambda_{\mathsf{PLWE},Q}$ quantum and classical attack log complexities against $\mathsf{PLWE}^{(f)}$ for $f \in \mathcal{F}_1(n, m', d')$ of minimum degree m' and maximum degree n, evaluated using (6.36).
- Choose ciphertext compression parameter $\mathsf{cmp} > 0$ subject to p_e constraint above.
- If $\lambda_{\mathsf{PLWE},C}$ and $\lambda_{\mathsf{PLWE},Q}$ satisfy the security goal (6.69), return parameter set. Else, restart with new n, k, t values.

6.4 Best known attacks: Titanium-CPA and Titanium-CCA

The previous Section contained analysis of our claimed security strength for Titanium-CPA and Titanium-CCA, based on our security proof relating the security of those schemes to the hardness of the PLWE problem, and the complexity of best known cryptanalytic attacks against the PLWE problem. However, we believe those security claims are somewhat conservative, as we do not know of attacks on our schemes with complexity that matches the claims. In this Section, we describe the several attacks we know of on Titanium-CPA and Titanium-CCA, and analyse their complexity for our Titanium parameter sets to identify our best known attacks.

6.4.1 Brute-force search attacks

Since our Titanium-CPA and Titanium-CCA schemes use a pseudorandom generator PRG based on the XOF hash function to generate all needed randomness 256-bit seeds, a brute-force search attack can be mounted against PRG, and is less costly than a brute-force attack on the MP-LWE problem. There are several ways to mount this attack due to the several uses of PRG in our scheme. The least costly is probably to invert the mapping g mapping $\mathsf{seedkg} \in \mathsf{byte32}$ to $\mathsf{seedpk} \in \mathsf{byte}^{32}$ that is stored in the public key. It is likely that this mapping has only a few preimages, one of which is the correct seedkg.

Classical attack

Given a run-time bound T, the attack can evaluate g on $Q \approx T/T_{\mathsf{RO}}$ input seeds x, and succeeds of $g(x) = \mathsf{seedpk}$. Here, we take $T_{\mathsf{RO}} = 2^{15}$ as the approximate classical gate cost for our SHA-3 based instantiation of XOF (see previous Section). The attack success probability is $p = Q/2^{256} = \frac{T}{T_{\mathsf{RO}} \cdot 2^{256}}$, giving am estimated time to success probability cost

$$\lambda_{\mathsf{brf},C} = T_{\mathsf{RO}} \cdot 2^{256} = 2^{272}. \tag{6.77}$$

Quantum attack

The quantum attack runs Grover's algorithm [Gro96] to find seedkg by making Q quantum queries to a quantum implementation of the Boolean function f mapping $x \in \mathsf{byte}^{32}$ to 1 if $g(x) = \mathsf{seedpk}$ and 0 else. Given a total run-time T quantum gates and the maximum quantum circuit depth restriction MD, the attack runs in parallel Q/Q_D Grover circuits C'_i $(i = 1, \ldots, Q/Q_D)$ each of depth MD, each making $Q_D = \mathsf{MD}/\mathsf{D}_{\mathsf{QRO}}$ oracle queries. Here, Q_D is the maximal number of sequential queries C' can make under the MD depth restriction, since we assume, as in the previous Section, that each call to f consumes a quantum circuit depth $D_{\mathsf{QRO}} \approx 2^{13}$ gates. We also assume that each call to f consumes a total number $T_{\mathsf{QRO}} \approx 2^{19}$ of quantum gates (we neglect the cost of the Grover 'diffusion' operator). The attack succeeds if one of the Grover circuits C'_i's succeeds. Since each run of C'_i succeeds with Q_D queries with probability $\approx Q_D^2/2^{256}$ and the C'_i are independent, the overall success probability of the attack over all Q/Q_D circuits C'_i is

$$1 - (1 - Q_D^2/2^{256})^{Q/Q_D} \approx Q \cdot Q_D/2^{256} \approx \frac{T \cdot \mathsf{MD}}{T_{\mathsf{QRO}} \cdot D_{\mathsf{QRO}} \cdot 2^{256}},$$

since $Q \approx T/T_{\mathsf{QRO}}$. This gives the following estimated time to success probability cost for this attack:

$$\lambda_{\mathsf{brf},Q} = \frac{\mathsf{D}_{\mathsf{QRO}} \cdot T_{\mathsf{QRO}} \cdot 2^{256}}{\mathsf{MD}} = \frac{2^{288}}{\mathsf{MD}}. \tag{6.78}$$

The above attack cost corresponds to security level $\lambda_{\mathsf{brf},Q} \in \{248, 224, 192\}$ for $\mathsf{MD} \in \{2^{40}, 2^{64}, 2^{96}\}$ correspond approximately to the NIST category 5 (AES256 key search) security levels [NISa]. We remark that the Grover attack complexity estimate in 6.78 is lower by a factor $\approx 2^{10}$ than the complexity estimated in [NISa] for Grover attack complexity against AES-256 key search attacks. We believe this

is due to the slightly higher circuit complexity and depth of AES-256 compared to SHA-3, and do not view it as a significant issue, as it only slightly affects complexity estimates at the highest security levels.

6.4.2 Lattice attacks on MP-LWE

Let $n' = n + d + k - 1$. We recall that the public-key of Titanium-CPA and Titanium-CCA contains a $\mathsf{MP\text{-}LWE}_{q,n,d',t,\chi_{e,\mathsf{MP}}}$ instance $(a_i, b_i = a_i \odot_{d+k} s + e_i)_{i \leq t} \in (\mathbb{Z}_q^{<n}[x] \times \mathbb{Z}_q^{d+k}[x])^t$ in the secret key $s \in \mathbb{Z}_q^{<n'}[x]$. The attack consists in recovering s from $(a_i, b_i)_{i \leq t}$. Viewing MP-LWE as a special case of an LWE instance in dimension $n' = n + d + k - 1$, any of the known algorithms for the search (rather than decision) variant of LWE could be used. In particular, we could use a search variant of the 'dual lattice' attack (described in Sec. 6.3.7) applied to MP-LWE. One such variant that seems to give the lowest complexity is based on Kannan's embedding method [Kan87] to convert the LWE instance to a 'unique SVP' instance, as analysed by Albrecht et al. in [AFG13] and Alkim et al. in [ADPS16]. We describe the attack below (we call it the 'primal embedding attack'), and then we explain how to optimise it to take advantage of the special Toeplitz structure of the MP-LWE matrix.

The generic primal 'embedding attack'

Let $t' = t \cdot d'$. Similar to the 'dual lattice' attack, the 'embedding attack' [Kan87, AFG13] consists in rewriting the $\mathsf{MP\text{-}LWE}_{q,n,d',t,\chi_{e,\mathsf{MP}}}$ instance $(a_i, b_i = a_i \odot_{d'} s + e_i)_{i \leq t} \in (\mathbb{Z}_q^{<n}[x] \times \mathbb{Z}_q^{d'}[x])^t$ as an LWE instance $(\mathsf{A}, \mathbf{b}) \in \mathbb{Z}_q^{t' \cdot n'} \times \mathbb{Z}_q^{t'}$ over $\mathbb{Z}_q$, where $\mathbf{b} = \mathsf{A} \cdot \mathbf{s} + \mathbf{e}$. Similar to the 'dual lattice' attack, we consider using a subset of $m^* \leq t' = t \cdot d'$ samples (rows) of the instance to give the sublattice LWE instance $(\mathsf{A}^*, \mathbf{b}^* = \mathsf{A}^* \cdot \mathbf{s} + \mathbf{e}^*) \in \mathbb{Z}_q^{m^* \cdot n'} \times \mathbb{Z}_q^{m^*}$, where the sublattice dimension m^* is chosen by the attacker to optimise the attack (see below). The attack constructs a (column) basis matrix $\bar{\mathsf{A}}^* \in \mathbb{Z}^{m^* \times m^*}$ for the m^*-dimensional LWE (primal) lattice $L_q(\mathsf{A}^*) = \{\mathsf{A}^* \cdot \mathbf{u} + q \cdot \mathbb{Z}^{m^*} : \mathbf{u} \in \mathbb{Z}_q^{n'}\}$ and builds a basis $\mathsf{B} \in \mathbb{Z}^{(m^*+1) \times (m^*+1)}$ for an *embedding* lattice $L(\mathsf{B})$ of the form

$$\mathsf{B} = \begin{pmatrix} \bar{\mathsf{A}}^* & \mathbf{b}^* \\ \mathbf{0}^T & c \end{pmatrix},$$

where c is a small constant (whose value depends on the variant of the attack; see below). The (m^*+1)-dimensional embedding lattice $L(\mathsf{B})$ generated by the columns of B therefore contains an embedding of the LWE primal lattice $L_q(\mathsf{A}^*)$ and the target LWE vector $\mathbf{b}^*$. In particular, the embedding lattice $L(\mathsf{B})$ contains the short vector $\mathbf{v} = (\mathbf{e}, c)^T$ of norm $\|\mathbf{v}\| \approx \|\mathbf{e}^*\|$. By running a lattice basis reduction algorithm on B, the attack aims at recovering $\mathbf{v}$ as one of the vectors in the reduced basis (which immediately reveals the error $\mathbf{e}^*$ and then the secret $\mathbf{s}$). There are several analyses/variants of the success condition (and complexity) of this attack in the literature. Here, we evaluate it using two analysis approaches: analysis approach 1 [AFG13]) is more rigorous, while analysis approach 2 [ADPS16] is heuristic, based on the block-structure of the BKZ lattice reduction algorithm and the Geometric Series Assumption for the output basis, and tends to give lower complexity estimates.

- *Primal Attack Analysis Approach 1 [AFG13]*: In this variant, the constant c in the basis B is set to approximate the error vector norm: $c \approx \sqrt{m^*} \cdot \alpha \cdot q$. Based on the Gaussian heuristic, $\mathbf{v}$ is typically the shortest vector in $L(\mathsf{B})$, and the gap ratio γ between $\|\mathbf{v}\|$ and the second minimum of the lattice is [AFG13]

$$\gamma \approx \frac{1}{\sqrt{4\pi \mathrm{e}} \cdot q^{n'/m^*} \varepsilon' \cdot \alpha} \tag{6.79}$$

  with success probability

$$p_{\mathsf{suc}} > 1 - \left(\varepsilon' \cdot \exp\left(\frac{1 - (\varepsilon')^2}{2} \right) \right)^{m^*}, \tag{6.80}$$

where $\varepsilon' > 1$ is an adjustable constant[4]. The attack then runs an approximate SVP algorithm, such as BKZ with block size b, to compute a short non-zero vector $\mathbf{v}'$ in lattice $L(\mathsf{B})$. Experimental practical evidence shows [GN08] that typically, the vector $\mathbf{v}'$ returned by BKZ will be the shortest vector $\mathbf{v}$ as long as the gap γ exceeds $\theta \cdot \delta(b)^{m^*}$, where $\delta(b)$ denotes the Hermite Factor of BKZ with block size b (see Eq. (6.32) for some parameter θ that experimentally is typically ≈ 0.3 for a high success probability. It follows that the 'embedding attack' is expected to succeed with high probability if

$$f_1(m^*, b) \stackrel{\text{def}}{=} \delta(b)^{m^*} \cdot q^{n'/m^*} < \frac{1}{\theta \cdot \sqrt{4\pi \mathrm{e}} \cdot \varepsilon' \cdot \alpha}, \tag{6.81}$$

and ε' is chosen so that the right-hand side in (6.80) is > 0.5 (choosing $\varepsilon' = 1.4$ suffices for the latter). The quantity $f_1(m^*, b)$ on the left-hand side of (6.81) is identical to the BKZ output vector length (6.31) in the 'dual lattice' attack described in the previous Sections (up to a constant factor δ^{-1}), except that here the LWE dimension is n' instead of $m \leq n$. For each fixed b, it can be minimised with respect to m^* similarly to (6.37) with the optimum choice of $m^* = m^*_{\mathsf{opt}}$:

$$m^*_{\mathsf{opt}}(b) \approx \sqrt{\frac{n' \log q}{\log \delta(b)}} \text{ and } f_1(m^*_{\mathsf{opt}}(b), b) \approx 2^{2\sqrt{n' \log q \log \delta(b)}}. \tag{6.82}$$

The expected classical (resp. quantum) log time complexity of the primal embedding attack (repeated until success), according to analysis approach 1, is

$$\lambda_{C,\mathsf{emb},1} = \log_2(T_{\mathsf{CBKZ}}(b)/p_{\mathsf{suc}}) \text{ and } \lambda_{Q,\mathsf{emb},1} = \log_2(T_{\mathsf{QBKZ}}(b)/p_{\mathsf{suc}}) \tag{6.83}$$

where the classical (resp. quantum) BKZ run-time $T_{\mathsf{CBKZ}}(b)$ (resp. $T_{\mathsf{QBKZ}}(b)$) is estimated by (6.33), at the smallest b such that success condition (6.85) is satisfied, for $m^* = m^*_{\mathsf{opt}}(b)$ chosen at the optimum value (6.82) and ε' is chosen to minimize the expected time complexity.

- *Primal Attack Analysis Approach 2 [ADPS16]*: In this primal attack variant [ADPS16], the constant c in the basis B is set to 1. The attack runs the BKZ lattice reduction algorithm with block size b on basis B, aiming to compute a short non-zero vector $\mathbf{v}'$ in lattice $L(\mathsf{B})$. Under the Geometric Series Assumption (GSA) model [Sch03b], the BKZ output basis Gram-Schmidt norms $\|\mathbf{b}^*_i\|$ $(i = 0, \ldots, m^*)$ follow a geometric series; namely, since the lattice determinant is with high probability $\det(L(\mathsf{B})) = \det(L_q(\mathsf{A}^*) = q^{m^*-n}$, we have:

$$\|\mathbf{b}^*_i\| = \delta(b)^{m^*-2i} \cdot q^{\frac{m^*-n}{m^*+1}}, i = 0, \ldots, m^*. \tag{6.84}$$

The heuristic in [ADPS16] is that if the attack fails to recover the short vector $\mathbf{v}$ from the BKZ reduced basis, the projection $\pi_S(\mathbf{v})$ of $\mathbf{v}$ onto the vector space S spanned by the last b BKZ GSO vectors $\mathbf{b}^*_{m^*+1-d}, \ldots, \mathbf{b}^*_{m^*+1}$ should behave as a random b-dimensional projection, with expected norm $\|\pi_S(\mathbf{v})\| \approx \sqrt{b} \cdot \alpha \cdot q$. On the other hand, BKZ reduction ensures that $\|\mathbf{b}^*_{m^*+1-d}\|$ is the norm of the shortest non-zero vector in the projection of $L(\mathsf{B})$ onto S. Therefore, under these heuristics, if the condition $\|\mathbf{b}^*_{m^*+1-d}\| > \sqrt{b} \cdot \alpha \cdot q$ holds, a failure of the BKZ reduced basis to contain $\mathbf{v}$ implies a contradiction, so we expect that the reduced basis will contain $\mathbf{v}$ when the latter condition holds. Based on the GSA (6.84), this gives the heuristic attack success condition (with high probability) if

$$f_2(m^*, b) \stackrel{\text{def}}{=} \delta(b)^{2b-1-(m^*+1)} \cdot q^{1-\frac{n}{m^*+1}} > \sqrt{b} \cdot \alpha \cdot q. \tag{6.85}$$

[4]We note that we define αq as the standard deviation of the error coordinates, which is smaller by a factor $1/\sqrt{2\pi}$ than the definition of αq in [AFG13].

For each fixed b, $f_2(m^*, b)$ can be minimized with respect to m^*, with the optimum choice of $m^* = m^*_{\mathsf{opt}}$ being

$$m^*_{\mathsf{opt}}(b) + 1 \approx \sqrt{\frac{n' \log q}{\log \delta(b)}} \text{ and } f_2(m^*_{\mathsf{opt}}(b), b) \approx \delta(b)^{2b-1-\sqrt{\frac{n' \log q}{\log \delta(b)}}} \cdot q^{1-\sqrt{\frac{n' \log \delta(b)}{\log q}}}. \tag{6.86}$$

The expected classical (resp. quantum) log time complexity of the primal embedding attack, according to analysis approach 2 is,

$$\lambda_{C,\mathsf{emb},2} = \log_2(T_{\mathsf{CBKZ}}(b)) \text{ and } \lambda_{Q,\mathsf{emb},2} = \log_2(T_{\mathsf{QBKZ}}(b)) \tag{6.87}$$

where the classical (resp. quantum) BKZ run-time $T_{\mathsf{CBKZ}}(b)$ (resp. $T_{\mathsf{QBKZ}}(b)$) is estimated by (6.33), at the smallest b such that success condition (6.85) is satisfied, for $m^* = m^*_{\mathsf{opt}}(b)$ chosen at the optimum value (6.86).

Optimised primal 'embedding attack' against MP-LWE

Our PLWE^f-based complexity lower bounds on MP-LWE in the previous sections reduce from PLWE with a secret polynomial of dimension $\leq n$, but the above 'generic' embedding attack against MP-LWE works on the MP-LWE secret in a larger dimension n'. Thus there is an apparent complexity gap of d' in the secret vector dimension between those lower and upper bounds. We now explain a simple optimisation of the generic 'embedding attack' against MP-LWE that takes advantage of the 'block Toeplitz' structure of the MP-LWE matrix to give a lower complexity attack on MP-LWE, closing some of this apparent complexity gap.

Indeed, the i-th block of d' rows of matrix A in the LWE representation of the MP-LWE instance has the Toeplitz form $\mathsf{Toep}^{d',n}(a_i)$, for $i = 1, \ldots, t$. Due to the triangle of zero coefficients on the right-hand side of $\mathsf{Toep}^{d',n}(a_i)$, the top m^*/t rows of each $\mathsf{Toep}^{d',n}(a_i)$ submatrix have non-zero coefficients only in the leftmost $n + m^*/t$ positions (equivalently, the polynomials $x^j \cdot a_i(x)$ are of degree $< n + m^*/t$ for $j = 0, \ldots, m^*/t - 1$). Therefore, if we choose in the 'embedding attack' on MP-LWE the m^* rows (samples) of $(\mathsf{A}, \mathbf{b})$ as our m^* of our sublattice LWE instance to consist of the top m^*/t rows of each of the $\mathsf{Toep}^{d',n}(a_i)$ blocks, those LWE samples in fact constitute an LWE instance with respect to a lower dimensional secret $\mathbf{s}^* \in \mathbb{Z}_q^{n+m^*/t}$ consisting of the first $n + m^*/t$ coefficients of the original n'-dimensional MP-LWE secret s. Therefore, our sublattice LWE instance in the optimised attack has the form $(\mathsf{A}^*, \mathbf{b}^* = \mathsf{A}^* \cdot \mathbf{s}^* + \mathbf{e}^*) \in \mathbb{Z}_q^{m^* \times (n+m^*/t)} \times \mathbb{Z}_q^{m^*}$, where we also remove the last $d' - m^*/t$ columns of A to form A^*.

- MP-LWE-*optimised primal embedding attack; Analysis Approach 1 [AFG13]*: The analysis of the attack then proceeds identically to approach 1 for analysing the generic attack above, replacing generic condition (6.81) with the MP-LWE optimised attack success condition

$$f_1(m^*, b) \stackrel{\text{def}}{=} q^{1/t} \cdot \delta(b)^{m^*} \cdot q^{n/m^*} < \frac{1}{\theta \cdot \sqrt{4\pi \mathrm{e}} \cdot \varepsilon' \cdot \alpha}, \tag{6.88}$$

and optimised sublattice dimension

$$m^*_{\mathsf{opt}}(b) \approx \sqrt{\frac{n \log q}{\log \delta(b)}} \text{ and } f_1(m^*_{\mathsf{opt}}(b), b) \approx q^{1/t} \cdot 2^{2\sqrt{n \log q \log \delta}}. \tag{6.89}$$

Note that the optimum sublattice dimension $m^*_{\mathsf{opt}}(b)$ is now exactly the same as it is in the attack in dimension n (rather than n'). The remaining overhead for this attack over the standard primal embedding attack on LWE in dimension n is the extra factor $q^{1/t}$ in the function $f(b)$, which is a constant between 3 and 4 for our parameter settings. We leave it as an open problem to find improved optimised attacks on MP-LWE that also close this remaining gap.

We remark that the 'optimised embedding attack' on MP-LWE as described above only recovers the first $n + m^*/t$ coefficients of the n'-dimensional MP-LWE secret s. But since $n > d'$ for our parameters, this constitutes more than half of the coefficients of s. The remaining coefficients of s can then be recovered by either repeating the attack using the last $n + m^*/t$ coefficients of s (which doubles the run-time), or (at even lower complexity) by solving the remaining $(d' - m^*/t)$-dimensional LWE instance in the last coefficients of s. In addition to the extra factor $q^{1/t}$ in the success condition, the attack also has a higher complexity than the distinguishing 'dual lattice' attack considered in our lower bound estimates due to the fact that it is solving the search LWE problem with high probability, rather than only distinguishing the samples from uniform.

- MP-LWE-*optimised primal embedding attack; Analysis Approach 2 [ADPS16]*: The approach 2 analysis proceeds identically to approach 2 for analysing the generic attack above, replacing generic condition (6.85) with the MP-LWE-optimised attack success condition

$$f_2(m^*, b) \stackrel{\text{def}}{=} q^{-1/t} \cdot \delta(b)^{2b-1-(m^*+1)} \cdot q^{1-\frac{n-1/t}{m^*+1}} > \sqrt{b} \cdot \alpha \cdot q. \tag{6.90}$$

and with optimised sublattice dimension

$$m^*_{\mathsf{opt}}(b)+1 \approx \sqrt{\frac{(n-1/t)\log q}{\log \delta(b)}} \text{ and } f_2(m^*_{\mathsf{opt}}(b), b) \approx q^{-1/t} \cdot \delta(b)^{2b-1-\sqrt{\frac{(n-1/t)\log q}{\log \delta(b)}}} \cdot q^{1-\sqrt{\frac{(n-1/t)\log \delta(b)}{\log q}}}. \tag{6.91}$$

Similarly to MP-LWE-optimised analysis approach 1, the optimum sublattice dimension $m^*_{\mathsf{opt}}(b)$ is now $n - 1/t \approx n$ (rather than n'), and the remaining overhead for this attack over the standard embedding attack on LWE in dimension n is the extra factor $q^{-1/t}$ in the function f_2.

Table 6.9: MP-LWE-optimised Primal Embedding (analysis approach 1) key recovery lattice-based attack log expected classical (resp. quantum) complexity estimates, for all our parameter sets. Here, attack complexity estimates on scheme parameter sets are denoted by $\lambda_{C,\mathsf{emb},1}$ (resp. $\lambda_{Q,\mathsf{emb},1}$), and (for comparison) we also show claimed lower bound estimates for PLWE security levels $\lambda_{\mathsf{PLWE},C,m_{\max}}$ (resp. $\lambda_{\mathsf{PLWE},Q,m_{\max}}$) and overall scheme goals λ_C (resp. λ_Q at $\mathsf{MD} = 2^{40}$). CCA (resp. CPA) rows refer to Titanium-CCA (resp. Titanium-CPA) scheme.

Par. Set	**Classical**			**Quantum**		
	$\lambda_{C,\mathsf{emb},1}$	$\lambda_{C,\mathsf{PLWE},m_{\max}}$	λ_C	$\lambda_{Q,\mathsf{emb},1}$	$\lambda_{Q,\mathsf{PLWE},m_{\max}}$	λ_Q
CCA, Toy64	154	90	79	139	83	66
CPA, Toy64	168	97	79	152	89	66
CCA, Lite96	238	129	111	216	118	98
CPA, Lite96	260	133	111	236	122	98
CCA, Std128	317	176	143	287	161	130
CPA, Std128	343	182	143	311	166	130
CCA, Med160	360	205	175	327	187	162
CPA, Med160	389	211	175	353	194	162
CCA, Hi192	460	243	207	418	222	193
CPA, Hi192	491	244	207	445	224	193
CCA, Super256	633	333	272	575	305	258
CPA, Super256	670	333	272	608	305	258

Other attacks. There exist other potential attack approaches on MP-LWE that could be applied to our schemes. In particular, one could try algebraic/linearisation attacks [AG11, AFG13] or combinatorial ('BKW'-type) attacks [BKW03] and their variants [KF15, GJS15]. However, we did not

Table 6.10: MP-LWE-optimised Primal Embedding (analysis approach 2) key recovery lattice-based attack log expected classical (resp. quantum) complexity estimates for all our parameter sets. Here, attack complexity estimates on scheme parameter sets are denoted by $\lambda_{C,\mathsf{emb}}$ (resp. $\lambda_{Q,\mathsf{emb}}$), and (for comparison) we also show claimed lower bound estimates for PLWE security levels $\lambda_{\mathsf{PLWE},C,m_{\max}}$ (resp. $\lambda_{\mathsf{PLWE},Q,m_{\max}}$) and overall scheme goals λ_C (resp. λ_Q at $\mathsf{MD} = 2^{40}$). CCA (resp. CPA) rows refer to Titanium-CCA (resp. Titanium-CPA) scheme.

Par. Set	**Classical**			**Quantum**		
	$\lambda_{C,\mathsf{emb},2}$	$\lambda_{C,\mathsf{PLWE},m_{\max}}$	λ_C	$\lambda_{Q,\mathsf{emb},2}$	$\lambda_{Q,\mathsf{PLWE},m_{\max}}$	λ_Q
CCA, Toy64	125	90	79	113	83	66
CPA, Toy64	134	97	79	121	89	66
CCA, Lite96	181	129	111	164	118	98
CPA, Lite96	194	133	111	176	122	98
CCA, Std128	236	176	143	214	161	130
CPA, Std128	251	182	143	228	166	130
CCA, Med160	274	205	175	248	187	162
CPA, Med160	291	211	175	264	194	162
CCA, Hi192	345	243	207	313	222	193
CPA, Hi192	363	244	207	330	224	193
CCA, Super256	467	333	272	424	305	258
CPA, Super256	489	333	272	444	305	258

give complexity estimates for those attacks, as both approaches (but in particular BKW style attacks) tend require a large number of given LWE samples with respect to the dimension of the secret ($\geq n$ in the case of MP-LWE). Due to the relatively low number of samples $t \cdot (d + k)$ in our public key, these approaches do not seem to yield competitive attacks against our schemes. We leave it to future work to see whether such attacks can be adapted to improve on the complexity of our best known attacks.

Summary of best known attack complexity estimates. We computed estimates for the classical (resp. quantum) expected complexity of our key recovery 'MP-LWE-optimised' primal embedding attack against Titanium-CPA and Titanium-CCA for all our parameter sets, based on the (conservative) time estimates for BKZ discussed in Sec. 6.3.7, using both approaches 1 and 2 described above. The resulting attack complexities for approach 1 (resp. approach 2) are summarized and compared to our overall scheme goals and maximum PLWE security claims in Table 6.9 (resp. Table 6.10).

Taking the minimum complexity over the attacks summarized above for each parameter set, Table 6.11 summarizes the best known attack complexities on our parameter sets, compared to our overall scheme goals and maximum PLWE security claims.

Table 6.11: Best known attack complexity estimates, for all our parameter sets. Here, best attack complexity estimates on scheme parameter sets are denoted by $\lambda_{C,\mathsf{bstatk}}$ (resp. $\lambda_{Q,\mathsf{bstatk}}$), and (for comparison) we also show claimed lower bound estimates for PLWE security levels $\lambda_{\mathsf{PLWE},C,m_{\max}}$ (resp. $\lambda_{\mathsf{PLWE},Q,m_{\max}}$) and overall scheme goals λ_C (resp. λ_Q at $\mathsf{MD} = 2^{40}$). CCA (resp. CPA) rows refer to Titanium-CCA (resp. Titanium-CPA) scheme.

Par. Set	**Classical**			**Quantum**		
	$\lambda_{C,\mathsf{bstatk}}$	$\lambda_{C,\mathsf{PLWE},m_{\max}}$	λ_C	$\lambda_{Q,\mathsf{bstatk}}$	$\lambda_{Q,\mathsf{PLWE},m_{\max}}$	λ_Q
CCA, Toy64	125	90	79	113	83	66
CPA, Toy64	134	97	79	121	89	66
CCA, Lite96	181	129	111	164	118	98
CPA, Lite96	194	133	111	176	122	98
CCA, Std128	236	176	143	214	161	130
CPA, Std128	251	182	143	228	166	130
CCA, Med160	272	205	175	245	187	162
CPA, Med160	272	211	175	245	194	162
CCA, Hi192	272	243	207	245	222	193
CPA, Hi192	272	244	207	245	224	193
CCA, Super256	272	333	272	245	305	258
CPA, Super256	272	333	272	245	305	258

Chapter 7

Version Update History

This Chapter summarises the updates made to the Titanium specification document since the original version (called version 1.0, dated 1 Dec 2017) submitted to the NIST PQC process.

7.1 Updates in version 1.1, dated Dec. 2018

This version contains the following updates (some of which were reported at the First NIST PQC workshop in April 2018):

- **Constant time implementation improvements:** The implementation submitted to NIST may not be constant time depending on the C compiler implementation of % mod reduction operations. To address this, we rewrote the mod reduction code to avoid % operations and ensure a compiler-independent constant-time implementation. At the same time, we also improved the efficiency of the NTT implementation by merging all the intermediate levels of the radix-2 NTT (except the last level), to reduce the number of mod reductions and perform the mod reduction at end. We chose $\rho = 2^{32}$ in Montgomery reduction, to make it more efficient by using the low product. The updated implementation code is available at `https://github.com/raykzhao/Titanium/`.
  - **Relevant document updates:**
    - Revised the modulo reduction documentation (Section 3.4.2).
    - Simplified the constant time comparison implementation (Section 3.5).
    - Updated Titanium benchmark timing results for the new more efficient constant-time implementations in Tables 1.1, 3.1, 3.2, 3.3, 3.4.
- **OpenQuantum integration:** Our updated implementation of Titanium was recently integrated into the Open Quantum Safe (liboqs) library. We have added our benchmark results for this integration and benchmark comparison with other liboqs schemes.
  - **Relevant document updates:** Added liboqs integration results in new Section 3.4.6.
- **New AES-based PRG Titanium variant Titanium – AES:** We added a new variant of Titanium (not in original NIST submission) which uses AES to replace the SHA-3 based PRG. This allows faster implementation on suitable hardware using Intel AES-NI instructions.
  - **Relevant document updates:** Added details of Titanium – AES and implementation benchmarks in new Section 3.4.5.
- **Minor documentation corrections/clarifications:**
  - Chap 1: delete reference to Kyber with respect to $x^{1024} + 1$.

- Chap 2, Sec. 2.1.1: clarify def. of middle product.
- Chap 3:
  - Fixed the typo: MP-NTT-P(a,b)=MP-NTT(a,b) in Lemma 3.2.4
  - Updated the version of compilers used in new benchmarks: gcc 7.2.0 and clang 5.0.1
  - Deleted some implementation remarks in chapter 3 which no longer apply to the revised modulo reduction implementation.
- Chap 6:
  - Sec. 6.3.2 and 6.3.4: corrected/clarified definition/notation of MP-LWE and PLWE problems
  - Sec. 6.3.7: clarified implication of weak f.
  - Sec. 6.3.8, quantum attack approach: "two possible assumptions" –> "three possible assumptions"
  - Sec. 6.4.2, "projection of det(L(B))" –> "projection of L(B)"

Acknowledgement. We thank Chitchanok Chuengsatiansup for helpful discussions and suggestions regarding our implementation of Titanium.

Bibliography

[ACF+14] M. R. Albrecht, C. Cid, J-Ch Faugère, R. Fitzpatrick, and L. Perret. Algebraic algorithms for LWE problems. *IACR Cryptology ePrint Archive*, 2014:1018, 2014.

[ACPS09] B. Applebaum, D. Cash, C. Peikert, and A. Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In *Proc. of CRYPTO*, volume 5677 of *LNCS*, pages 595–618. Springer, 2009.

[AD17] M. R. Albrecht and A. Deo. Large modulus ring-lwe >= module-lwe. *IACR Cryptology ePrint Archive*, 2017:612, 2017.

[ADPS16] E. Alkim, L. Ducas, T. Pöppelmann, and P. Schwabe. Post-quantum key exchange - A new hope. In *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016.*, pages 327–343, 2016.

[AFG13] M. R. Albrecht, R. Fitzpatrick, and F. Göpfert. On the efficacy of solving LWE by reduction to unique-svp. In *Information Security and Cryptology - ICISC 2013 - 16th International Conference, Seoul, Korea, November 27-29, 2013, Revised Selected Papers*, pages 293–310, 2013.

[AG11] S. Arora and R. Ge. New algorithms for learning in presence of errors. In *Proc. of ICALP*, pages 403–415. Springer, 2011.

[Ajt96] M. Ajtai. Generating hard instances of lattice problems (extended abstract). In *Proc. of STOC*, pages 99–108. ACM, 1996.

[ALM+16] J-P. Aumasson, T. Lange, N. Mathewson, S. Neves, and D. F. Aranha. Cryptography coding standard. `https://cryptocoding.net/index.php/Coding_rules`, 2016. Accessed: 2017-11-09.

[AMG+16] M. Amy, O. Di Matteo, V. Gheorghiu, M. Mosca, A. Parent, and J. M. Schanck. Estimating the cost of generic quantum pre-image attacks on SHA-2 and SHA-3. In *Selected Areas in Cryptography - SAC 2016 - 23rd International Conference, St. John's, NL, Canada, August 10-12, 2016, Revised Selected Papers*, pages 317–337, 2016.

[AMM14] M. Amy, D. Maslov, and M. Mosca. Polynomial-time t-depth optimization of clifford+t circuits via matroid partitioning. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 33(10):1476–1489, 2014.

[APS15] M. R. Albrecht, R. Player, and S. Scott. On the concrete hardness of learning with errors. *J. Mathematical Cryptology*, 9(3):169–203, 2015.

[BCD+16] J. W. Bos, C. Costello, L. Ducas, I. Mironov, M. Naehrig, V. Nikolaenko, A. Raghunathan, and D. Stebila. Frodo: Take off the ring! practical, quantum-secure key exchange from LWE. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 1006–1018, 2016.

[BCLvV16] D. J. Bernstein, C. Chuengsatiansup, T. Lange, and C. van Vredendaal. NTRU Prime. Cryptology ePrint Archive, 2016. `http://eprint.iacr.org/2016/461`.

[BDK+17] J. W. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, and D. Stehlé. CRYSTALS - kyber: a cca-secure module-lattice-based KEM. *IACR Cryptology ePrint Archive*, 2017:634, 2017.

[Ber05] Daniel J. Bernstein. Cache-timing attacks on AES. `https://cr.yp.to/antiforgery/cachetiming-20050414.pdf`, 2005.

[BKW03] A. Blum, A. Kalai, and H. Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. *Journal of the ACM*, 50(4):506–519, 2003.

[BLL+15] S. Bai, A. Langlois, T. Lepoint, D. Stehlé, and R. Steinfeld. Improved security proofs in lattice-based cryptography: Using the rényi divergence rather than the statistical distance. In *Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part I*, pages 3–24, 2015.

[BLM13] S. Boucheron, G. Lugosi, and P. Massart. *Concentration Inequalities: A Nonasymptotic Theory of Independence*. Oxford University Press, 2013.

[BR93] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *CCS '93, Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, November 3-5, 1993.*, pages 62–73, 1993.

[BV11] Z. Brakerski and V. Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 97–106, 2011.

[CDPR16] R. Cramer, L. Ducas, C. Peikert, and O. Regev. Recovering short generators of principal ideals in cyclotomic rings. In *Proc. of EUROCRYPT*. Springer, 2016.

[CDW16] R. Cramer, L. Ducas, and B. Wesolowski. Short Stickelberger class relations and application to Ideal-SVP. *Cryptology ePrint Archive*, 2016. `https://eprint.iacr.org/2016/885`.

[CIV16a] W. Castryck, I. Iliashenko, and F. Vercauteren. On the tightness of the error bound in Ring-LWE. *LMS J Comput Math*, 2016.

[CIV16b] W. Castryck, I. Iliashenko, and F. Vercauteren. Provably weak instances of Ring-LWE revisited. In *Proc. of EUROCRYPT*, pages 147–167. Springer, 2016.

[CLS15] H. Chen, K. Lauter, and K. E. Stange. Attacks on search RLWE. Cryptology ePrint Archive, 2015. `http://eprint.iacr.org/2015/971`.

[CLS16] H. Chen, K. Lauter, and K. E. Stange. Vulnerable galois RLWE families and improved attacks. Cryptology ePrint Archive, 2016. `http://eprint.iacr.org/2016/193`.

[Den03] A. W. Dent. *A Designer's Guide to KEMs*, pages 133–151. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[Dev17] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 8.0.0)*, 2017. `http://www.sagemath.org`.

[DLL+17] L. Ducas, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé. CRYSTALS - dilithium: Digital signatures from module lattices. *IACR Cryptology ePrint Archive*, 2017:633, 2017.

[DORS08] Y. Dodis, R. Ostrovsky, L. Reyzin, and A. D. Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. *SIAM J. Comput.*, 38(1):97–139, 2008.

[EHL14] K. Eisenträger, S. Hallgren, and K. Lauter. Weak instances of PLWE. In *Proc. of SAC*. Springer, 2014.

[ELOS15] Y. Elias, K. E. Lauter, E. Ozman, and K. E. Stange. Provably weak instances of Ring-LWE. In *Proc. of CRYPTO*. Springer, 2015.

[FO99] E. Fujisaki and T. Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, pages 537–554, 1999.

[GJS15] Q. Guo, Th. Johansson, and P. Stankovski. *Coded-BKW: Solving LWE Using Lattice Codes*, pages 23–42. Springer Berlin Heidelberg, 2015.

[GN08] N. Gama and P. Q. Nguyen. Predicting lattice reduction. In *Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings*, pages 31–51, 2008.

[GOPS13] T. Güneysu, T. Oder, Th. Pöppelmann, and P. Schwabe. Software speed records for lattice-based signatures. In Philippe Gaborit, editor, *Post-Quantum Cryptography*, volume 7932 of *Lecture Notes in Computer Science*, pages 67–82. Springer-Verlag Berlin Heidelberg, 2013. Document ID: d67aa537a6de60813845a45505c313, `http://cryptojedi.org/papers/\#lattisigns`.

[Gro96] L. K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, pages 212–219, 1996.

[Har14] D. Harvey. Faster arithmetic for number-theoretic transforms. *Journal of Symbolic Computation*, 60:113–119, 2014.

[HHK17] D. Hofheinz, K. Hövelmanns, and E. Kiltz. A modular analysis of the fujisaki-okamoto transformation. Cryptology ePrint Archive, Report 2017/604, 2017. `http://eprint.iacr.org/2017/604`.

[HNP+03] N. Howgrave-Graham, P. Q. Nguyen, D. Pointcheval, J. Proos, J. H. Silverman, A. Singer, and W. Whyte. The impact of decryption failures on the security of NTRU encryption. In *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, pages 226–246, 2003.

[HPS98] J. Hoffstein, J. Pipher, and J. H. Silverman. NTRU: A ring-based public key cryptosystem. In *Proc. of ANTS*, pages 267–288. Springer, 1998.

[HQZ04] G. Hanrot, M. Quercia, and P. Zimmermann. The middle product algorithm I. *Appl. Algebra Engrg. Comm. Comput.*, 14(6):415–438, 2004.

[HRS16] A. Hülsing, A. Rijneveld, and F. Song. Mitigating multi-target attacks in hash-based signatures. In *PKC 2016, Part I*, pages 387–416, 2016.

[Kan87] R. Kannan. Minkowski's convex body theorem and integer programming. *Math. Oper. Res.*, 12(3):415–440, 1987.

[KF15] P. Kirchner and P-A. Fouque. An improved BKW algorithm for LWE with applications to cryptography and lattices. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, pages 43–62, 2015.

[LM06] V. Lyubashevsky and D. Micciancio. Generalized compact knapsacks are collision resistant. In *Proc. of ICALP*, pages 144–155. Springer, 2006.

[LMvdP15] T. Laarhoven, M. Mosca, and J. van de Pol. Finding shortest lattice vectors faster using quantum search. *Designs, Codes and Cryptography*, 77(2):375–400, 2015.

[LPR10] V. Lyubashevsky, C. Peikert, and O. Regev. On ideal lattices and learning with errors over rings. In *Proc. of EUROCRYPT*, LNCS, pages 1–23. Springer, 2010.

[LPR13] V. Lyubashevsky, C. Peikert, and O. Regev. On ideal lattices and learning with errors over rings. *J. ACM*, 60(6):43, 2013.

[LS15] A. Langlois and D. Stehlé. Worst-case to average-case reductions for module lattices. *Des. Codes Cryptography*, 75(3):565–599, 2015.

[Lyu09] V. Lyubashevsky. Fiat-Shamir with aborts: Applications to lattice and factoring-based signatures. In *Proc. of ASIACRYPT*, pages 598–616. Springer, 2009.

[Lyu16] V. Lyubashevsky. Digital signatures based on the hardness of ideal lattice problems in all rings. In *Proc. of ASIACRYPT*, pages 196–214. Springer, 2016.

[MG02] D. Micciancio and S. Goldwasser. *Complexity of Lattice Problems: a cryptographic perspective*, volume 671. Kluwer, 2002.

[Mic07] D. Micciancio. Generalized compact knapsacks, cyclic lattices, and efficient one-way functions. *Comput. Complexity*, 16(4):365–411, 2007.

[MR09] D. Micciancio and O. Regev. Lattice-based cryptography. In *Post-Quantum Cryptography, D. J. Bernstein, J. Buchmann, E. Dahmen (Eds)*, pages 147–191. Springer, 2009.

[MS] Michele Mosca and Douglas Stebila. Open Quantum Safe. `https://openquantumsafe.org/`. Accessed: 2018-03-13.

[NAB+17] Michael Naehrig, Erdem Alkim, Joppe Bos, Leo Ducas, Karen Easterbrook, Brian LaMacchia, Patrick Longa, Ilya Mironov, Valeria Nikolaenko, Christopher Peikert, Ananth Raghunathan, and Douglas Stebila. FrodoKEM learning with errors key encapsulation. `https://frodokem.org/files/FrodoKEM-specification-20171130.pdf`, 2017.

[NISa] NIST. NIST post-quantum competition. `http://csrc.nist.gov/groups/ST/post-quantum-crypto/documents/call-for-proposals-final-dec-2016.pdf`. Accessed: 2017-06-13.

[NISb] NIST. SHA-3 derived functions: cSHAKE, KMAC, TupleHash and ParallelHash. `http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-185.pdf`. Accessed: 2017-09-29.

[NISc] NIST. SHA-3 standard: Permutation-based hash and extendable-output functions. `http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf`. Accessed: 2017-09-29.

[Pei14] C. Peikert. Lattice cryptography for the internet. In *Post-Quantum Cryptography - 6th International Workshop, PQCrypto 2014, Waterloo, ON, Canada, October 1-3, 2014. Proceedings*, pages 197–219, 2014.

[Pei16a] C. Peikert. How not to instantiate Ring-LWE. In *Proc. of SCN*, pages 411–430. Springer, 2016.

[Pei16b] C. Peikert. How not to instantiate Ring-LWE. In *Proc. of SCN*, volume 9841 of *LNCS*, pages 411–430. Springer, 2016.

[PR06] C. Peikert and A. Rosen. Efficient collision-resistant hashing from worst-case assumptions on cyclic lattices. In *Proc. of TCC*, pages 145–166. Springer, 2006.

[PRSD17] C. Peikert, O. Regev, and N. Stephens-Davidowitz. Pseudorandomness of Ring-LWE for any ring and modulus. To appear in the proceedings of STOC, 2017.

[Reg05] O. Regev. On lattices, learning with errors, random linear codes, and cryptography. In *Proc. of STOC*, pages 84–93, 2005.

[Reg09] O. Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6), 2009.

[Rig15] P. Rigollet. 18.s997 high-dimensional statistics, 2015. Massachusetts Institute of Technology: MIT OpenCourseWare, `https://ocw.mit.edu`. License: Creative Commons BY-NC-SA.

[RSSS17] M. Roşca, A. Sakzad, D. Stehlé, and R. Steinfeld. Middle-product learning with error. Cryptology ePrint Archive, 2017.

[RWS17] M. Roşca, A. Wallet, and D. Stehlé. On the ring-lwe and polynomial-lwe problems. Private Communication, 2017.

[SB93] H. V. Sorensen and C. S. Burrus. Efficient computation of the DFT with only a subset of input or output points. *IEEE Transactions on Signal Processing*, 41(3):1184–1200, 1993.

[Sch87] C. P. Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. *Theor. Comput. Sci.*, 53(2-3):201–224, 1987.

[Sch03a] C. P. Schnorr. *Lattice Reduction by Random Sampling and Birthday Methods*, pages 145–156. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[Sch03b] C. P. Schnorr. *Lattice Reduction by Random Sampling and Birthday Methods*, pages 145–156. Springer Berlin Heidelberg, 2003.

[SE94] C. P. Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Mathematical Programming*, 66(1):181–199, 1994.

[Sei18] Gregor Seiler. Faster AVX2 optimized NTT multiplication for Ring-LWE lattice cryptography. `https://eprint.iacr.org/2018/039.pdf`, 2018.

[SGH+13] M. Srivastav, X. Guo, S. Huang, D. Ganta, M. B. Henry, L. Nazhandali, and P. Schaumont. Design and benchmarking of an asic with five sha-3 finalist candidates. *Microprocessors and Microsystems*, 37(2):246–257, 2013.

[Sho] V. Shoup. NTL-number theory library. `http://www.shoup.net/ntl/`. Accessed: 2017-09-29.

[SSTX09] D. Stehlé, R. Steinfeld, K. Tanaka, and K. Xagawa. Efficient public key encryption based on ideal lattices. In *Proc. of ASIACRYPT*, pages 617–635. Springer, 2009.

[TU16] E. Ebrahimi Targhi and D. Unruh. Post-quantum security of the fujisaki-okamoto and OAEP transforms. In *Theory of Cryptography - 14th International Conference, TCC 2016-B, Beijing, China, October 31 - November 3, 2016, Proceedings, Part II*, pages 192–216, 2016.