Saber: Module-LWR based key exchange, CPA-secure encryption and CCA-secure KEM

Jan-Pieter D'Anvers, Angshuman Karmakar
Sujoy Sinha Roy, and Frederik Vercauteren

imec-COSIC, KU Leuven
Kasteelpark Arenberg 10, Bus 2452, B-3001 Leuven-Heverlee, Belgium
firstname.lastname@esat.kuleuven.be

Abstract In this paper, we introduce Saber, a package of cryptographic primitives whose security relies on the hardness of the Module Learning With Rounding problem (Mod-LWR). We first describe a secure Diffie-Hellman type key exchange protocol, which is then transformed into an IND-CPA encryption scheme and finally into an IND-CCA secure key encapsulation mechanism using a post-quantum version of the Fujisaki-Okamoto transform. The design goals of this package were simplicity, efficiency and flexibility resulting in the following choices: all integer moduli are powers of 2 avoiding modular reduction and rejection sampling entirely; the use of LWR halves the amount of randomness required compared to LWE-based schemes and reduces bandwidth; the module structure provides flexibility by reusing one core component for multiple security levels. A constant-time AVX2 optimized software implementation of the KEM with parameters providing more than 128 bits of post-quantum security, requires only 101K, 125K and 129K cycles for key generation, encapsulation and decapsulation respectively on a Dell laptop with an Intel i7-Haswell processor.

1 Introduction

The threat of quantum computers, which break most widely used public key cryptographic primitives, has sparked a rising interest in post-quantum cryptography. This is emphasized by organizations such as ETSI and NIST that are looking towards standardization of post-quantum cryptography [19]. Lattice based cryptography is one of the most promising candidates that are resilient to all known quantum attacks. Examples include NTRU based schemes [29,45,11] and protocols based on the (ring)-Learning With Errors (LWE) problem: Alkim et al. [4] presented 'A New Hope', based on the ring-LWE problem; Bos et al. [17] introduced an alternative scheme called 'Frodo' based solely on LWE, but suffers from higher bandwidth and computational complexity; Bhattacharya et al. [12] improved upon the bandwidth of 'Frodo', by basing their protocol on LWR whilst still avoiding the use of rings; Bos et al. [16] presented a CCA-secure Mod-LWE based key exchange called 'Kyber' which takes the middle road between 'Frodo' and 'a New Hope' by using modules. Concurrently to our work, Jin et al. described a generic key exchange for Ring-LWE, Mod-LWE, LWE and LWR in [33], and Baan et al. [8] described a LWR, Ring-LWR key exchange.

In this paper, we introduce Saber, a suite of cryptographic primitives based on the Mod-LWR problem. The choices we made for the underlying hard problem and also the actual parameters of the scheme were motivated by three design principles: simplicity of the scheme and its implementation, efficiency and flexibility:

- Learning with Rounding (LWR) [10]: schemes based on (variants of) LWE require sampling from noise distributions, which needs randomness. Furthermore, the noise is included in public keys and ciphertexts resulting in higher bandwidth. LWR based schemes naturally reduce the bandwidth while avoiding additional randomness for the noise since it is deterministically obtained.
- Choice of moduli: we choose all integer moduli in the scheme to be powers of 2. This eliminates the need for explicit modular reduction and complicated sampling routines such as rejection sampling. We also prove that using powers of two, the keys are unbiased and that there is no need for steps such as uplifting and randomization or decoding of the exchanged information. These advantages contribute to the simplicity of our design, and facilitate constant time implementations. The main disadvantage of using such moduli is that it excludes the use of the number theoretic transform (NTT) to speed up polynomial multiplication. We propose the use of a combination of Toom-Cook and Karatsuba polynomial multiplication to mitigate this disadvantage.
- Modules [36,16]: the module versions of the problems (see Section 2) allow to interpolate between the original pure LWE/LWR problems and their ring versions, lowering computational complexity and bandwidth compared to LWE/LWR, while introducing protection against attacks on the ring structure of Ring-LWE/LWR and flexibility to move to higher security levels without any need to change the underlying arithmetic.

A high-level constant-time software implementation of Saber is provided and has been placed in the public domain[1] as part of the submission to the NIST competition. The implementation has been optimized using AVX2 instructions available in modern Intel processors and uses a combination of Toom-Cook and Karatsuba polynomial multiplication algorithms.

The remainder of the paper is organised as follows: in Section 2 we review the necessary background; we present a secure Diffie-Hellman type key exchange scheme in Section 3, a CPA secure encryption scheme in Section 4 and a CCA secure key encapsulation mechanism in Section 5. A security analysis of the hardness on the underlying mod-LWR problem is given in Section 6.1, based on which three parameter sets are chosen in Section 6.2. Finally, specific implementation choices that speed up our protocols are discussed in Section 7 and our implementation results are compared with the state of the art in Section 8.

[1] Source codes available at https://github.com/Angshumank/SABER

2 Preliminaries

2.1 Notation

We denote with $\mathbb{Z}_q$ the ring of integers modulo an integer q with representants in $[0, q)$ and for an integer z, we denote $z \bmod q$ the reduction of z in $[0, q)$. R_q is the quotient ring $\mathbb{Z}_q[X]/(X^n + 1)$ with n a fixed power of 2 (we only need $n = 256$). For any ring R, $R^{l \times k}$ denotes the ring of $l \times k$-matrices over R. For $p \mid q$, the $\bmod p$ operator is extended to (matrices over) R_q by applying it coefficient-wise. Single polynomials are written without markup, vectors are bold lower case and matrices are denoted with bold upper case. $\mathcal{U}$ denotes the uniform distribution and β_μ is a centered binomial distribution with parameter μ and corresponding standard deviation $\sigma = \sqrt{\mu/2}$. If χ is a probability distribution over a set S, then $x \leftarrow \chi$ denotes sampling $x \in S$ according to χ. If χ is defined on $\mathbb{Z}_q$, $\boldsymbol{X} \leftarrow \chi(R_q^{l \times k})$ denotes sampling the matrix $\boldsymbol{X} \in R_q^{l \times k}$, where all coefficients of the entries in $\boldsymbol{X}$ are sampled from χ.

We use the part selection function $\texttt{bits}(x, i, j)$ with $j \leq i$ to access j consecutive bits of a positive integer x *ending* at the i-th index (assuming least significant bit in the 0-th index), producing an integer in $\mathbb{Z}_{2^j}$; i.e., written in standard C code the function returns $(x \gg (i - j))\&(2^j - 1)$, where $\gg$ is the right-shift operator. This is explained in Fig. 1.The part selection function is extended to polynomials and matrices by applying it coefficient-wise. Finally let $\lfloor \rceil$ denote rounding to the nearest integer, which can be extended to polynomials and matrices coefficient-wise.

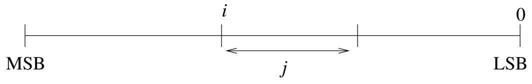

Figure 1: The $\texttt{bits}(x, i, j)$ operator.

2.2 Cryptographic definitions

Let KE be a Diffie-Hellman type key exchange protocol between two parties as illustrated in Protocol 1. KE is called $(1 - \delta)$-correct if after execution of the protocol $Pr[k' = k] \geqslant 1 - \delta$, where the probability is computed over the random coins used in Protocol 1. KE is called IND-RND secure if it is hard for an adversary to distinguish the real shared secret from random. More formally, we define the advantage of an adversary in distinguishing the key k from a uniformly random key $\hat{k} \leftarrow \mathcal{U}(\mathcal{K})$ as follows:

$$\mathrm{Adv}_{\mathrm{KE}}^{\text{ind-rnd}}(A) = \left| Pr\big[A(\mathbf{P}, \mathbf{A}, \mathbf{B}, k) = 1\big] - Pr\big[A(\mathbf{P}, \mathbf{A}, \mathbf{B}, \hat{k}) = 1\big] \right| .$$

A public key encryption scheme consists of a triple of functions PKE = ($\texttt{KeyGen}, \texttt{Enc}, \texttt{Dec}$), where $\texttt{KeyGen}$ returns a secret key sk and a public key pk;

Public parameters $\mathbf{P}$		
Alice		Bob
Choose secret a		
Compute $\mathbf{A}$ as function of $\mathbf{P}$ and a	$\xrightarrow{\mathbf{A}}$	Choose secret b
		Compute $\mathbf{B}$ as function of $\mathbf{P}$ and b
$k =$ Derive key from $\mathbf{P}, a, \mathbf{B}$	$\xleftarrow{\mathbf{B}}$	$k' =$ Derive key from $\mathbf{P}, b, \mathbf{A}$

Protocol 1: Diffie-Hellman type key exchange protocol

$\texttt{Enc}$ takes a public key pk and a message $m \in \mathcal{M}$ to produce a ciphertext $c \in \mathcal{C}$, and $\texttt{Dec}$ takes the secret key sk together with ciphertext c to output a message $m' \in \mathcal{M}$ or the symbol $\perp$ to denote rejection. The PKE is said to be $(1-\delta)$-correct if $Pr[\texttt{Dec}(sk, \texttt{Enc}(pk, m)) = m] \geqslant 1 - \delta$, where the probability is taken over $(pk, sk) \leftarrow \texttt{KeyGen}$ and the random coins of $\texttt{Enc}$. We use the notion of indistinguishability under chosen plaintext attacks (IND-CPA) and define the advantage of an adversary A by:

$$\mathrm{Adv}_{\mathrm{enc}}^{\text{ind-cpa}}(A) = \left| Pr\left[b' = b : \begin{array}{r} (pk, sk) \leftarrow \texttt{KeyGen}(); \\ (m_1, m_2) \leftarrow A^{\texttt{Enc}}(pk);\ b \leftarrow \mathcal{U}(\{0,1\}); \\ c \leftarrow \texttt{Enc}(pk, m_b);\ b' \leftarrow A^{\texttt{Enc}}(pk, c); \end{array} \right] - \frac{1}{2} \right| .$$

The weaker notion of one-wayness under chosen plaintext attacks (OW-CPA) is defined as:

$$\mathrm{Adv}_{\mathrm{enc}}^{\text{ow-cpa}}(A) = \left| Pr\left[m' = m : \begin{array}{r} (pk, sk) \leftarrow \texttt{KeyGen}(); \\ m \leftarrow \mathcal{M};\ c \leftarrow \texttt{Enc}(pk, m); \\ m' \leftarrow A^{\texttt{Enc}}(pk, c); \end{array} \right] - \frac{1}{2} \right| .$$

A key-encapsulation mechanism KEM $= (\texttt{KeyGen}, \texttt{Encaps}, \texttt{Decaps})$ is a triple of probabilistic algorithms, where $\texttt{KeyGen}$ returns a secret key sk and a public key pk, where $\texttt{Encaps}$ takes a public key pk and produces a ciphertext c and a key $k \in \mathcal{K}$, and where $\texttt{Decaps}$ takes the secret key sk, the public key pk and ciphertext c to return a key $k \in \mathcal{K}$ or the symbol $\perp$ to denote rejection. The KEM is said to be $(1-\delta)$-correct if $Pr[\texttt{Decaps}(sk, c) = k : (c, k) \leftarrow \texttt{Encaps}(pk)] \geqslant 1-\delta$, where the probability is taken over $(pk, sk) \leftarrow \texttt{KeyGen}$ and the random coins of $\texttt{Encaps}$. We use the notion of indistinguishability under chosen ciphertext attacks (IND-CCA) to define the advantage of an adversary A by:

$$\mathrm{Adv}_{\mathrm{KEM}}^{\text{ind-cca}}(A) = \left| Pr\left[b' = b : \begin{array}{c} (pk, sk) \leftarrow \texttt{KeyGen}();\ b \leftarrow \mathcal{U}(\{0,1\}); \\ (c, d, k_0) \leftarrow \texttt{Encaps}(pk); \\ k_1 \leftarrow \mathcal{K};\ b' \leftarrow A^{\texttt{Decaps}}(pk, c, d, k_b); \end{array} \right] - \frac{1}{2} \right| .$$

The advantage of an adversary A in distinguishing a pseudorandom generator $\texttt{gen}()$ with seed $seed_{\boldsymbol{A}} \leftarrow \mathcal{U}(\{0,1\}^{256})$ from a uniformly random distribution is defined as follows:

$$\mathrm{Adv}^{\mathrm{prg}}_{\mathtt{gen}()}(A) = \left| \begin{array}{c} Pr\left[b' = 1 : \begin{array}{c} seed_{\boldsymbol{A}} \leftarrow \mathcal{U}(\{0,1\}^{256}) \\ \boldsymbol{A} \leftarrow \mathtt{gen}(\mathrm{seed}_{\boldsymbol{A}}) \in R_q^{l\times l}; b' = A(\boldsymbol{A}); \end{array} \right] \cdot \\ \\ -Pr\left[b' = 1 : \boldsymbol{A} \leftarrow \mathcal{U}(R_q^{m\times l}); b' = A(\boldsymbol{A}); \right] \end{array} \right| \tag{1}$$

2.3 LWE, LWR and Mod-LWR problems

The learning with errors (LWE) problem was introduced by Regev [41] and its decisional version states that it is hard to distinguish uniform random samples $(\boldsymbol{a}, u) \leftarrow \mathcal{U}(\mathbb{Z}_q^{l\times 1} \times \mathbb{Z}_q)$ from LWE-samples of the form

$$\Big(\boldsymbol{a}, b = \boldsymbol{a}^T\boldsymbol{s} + e\Big) \ \in \ \mathbb{Z}_q^{l\times 1} \times \mathbb{Z}_q\,, \tag{2}$$

where the secret vector $\boldsymbol{s} \leftarrow \beta_\mu(\mathbb{Z}_q^{l\times 1})$ is fixed for all samples, $\boldsymbol{a} \leftarrow \mathcal{U}(\mathbb{Z}_q^{l\times 1})$ and $e \leftarrow \beta_\mu(\mathbb{Z}_q)$ is a small error. A module version of LWE, called Mod-LWE, was analyzed by Langlois and Stehlé [36] and essentially replaces the ring $\mathbb{Z}_q$ in the above samples by a quotient ring of the form R_q with corresponding error distribution $\beta_\mu(R_q^{l\times 1})$. The rank of the module is l and the dimension of the ring R_q is n. The case $l = 1$ corresponds to the ring-LWE problem introduced in [37].

The LWR problem was introduced by Banerjee et al. [10] and is a derandomized version of the LWE problem. In contrast to the LWE problem, the "noise" in the LWR problem is generated deterministically by scaling and rounding coefficients modulo q to modulo p (with $p < q$). In detail, an LWR sample is given by

$$\Big(\boldsymbol{a}, b = \Big\lfloor \frac{p}{q}(\boldsymbol{a}^T\boldsymbol{s}) \Big\rceil\Big) \ \in \ \mathbb{Z}_q^{l\times 1} \times \mathbb{Z}_p \tag{3}$$

for a fixed $\boldsymbol{s} \leftarrow \beta_\mu(\mathbb{Z}_q^{l\times 1})$ and uniform random $\boldsymbol{a} \leftarrow \mathcal{U}(\mathbb{Z}_q^{l\times 1})$. The decisional LWR problem states that is it hard to distinguish samples from the LWR distribution from that of the uniform distribution. A reduction from the LWE problem to the LWR problem was given by Banerjee et al. [10], and further improved by Alwen et al. [6], Bogdanov et al. [15] and, Alperin-Sheriff and Daniel Apon [5].

The security of our protocol relies on the hardness of the module version of LWR (Mod-LWR), which is a straightforward generalization of Mod-LWE. A Mod-LWR sample is given by

$$\Big(\boldsymbol{a}, b = \Big\lfloor \frac{p}{q}(\boldsymbol{a}^T\boldsymbol{s}) \Big\rceil\Big) \ \in \ R_q^{l\times 1} \times R_p \tag{4}$$

where the secret $\boldsymbol{s} \leftarrow \beta_\mu(R_q^{l\times 1})$ is fixed for all samples and $\boldsymbol{a} \leftarrow \mathcal{U}(R_q^{l\times 1})$.

The advantage of an adversary A in distinguishing m samples from a Mod-LWR distribution from that of a uniform distribution is defined as follows, where

m, k, μ, q and p are positive integers with $q > p$:

$$\mathrm{Adv}^{\text{Mod-LWR}}_{m,l,\mu,q,p}(A) = \left| \begin{array}{l} Pr\left(b' = 1 : \begin{array}{c} \boldsymbol{A} \leftarrow \mathcal{U}(R_q^{m\times l});\ \boldsymbol{s} \leftarrow \beta_\mu(R_q^{l\times 1}); \\ b' = A(\boldsymbol{A}, \lfloor (p/q)\boldsymbol{A}\boldsymbol{s} \rceil); \end{array}\right) \\ \\ -Pr\left(b' = 1 : \begin{array}{c} \boldsymbol{A} \leftarrow \mathcal{U}(R_q^{m\times l});\ \boldsymbol{u} \leftarrow \mathcal{U}(R_p^{l\times 1}); \\ b' = A(\boldsymbol{A}, \boldsymbol{u}); \end{array}\right) \end{array} \right| . \quad (5)$$

3 Key Exchange

In Protocol 2 we describe a Diffie-Hellman type key exchange scheme Saber.KE based on the hardness of Mod-LWR problem. Unlike the Diffie-Hellman key exchange [23], in our scheme the two communicating parties sometimes fail to agree on the same key. As in previous works [24,40,12], we can make this failure probability negligibly small by sending some additional reconciliation data c.

	Alice		Bob
1	$seed_{\boldsymbol{A}} \leftarrow \mathcal{U}(\{0,1\}^{256})$		
2	$\boldsymbol{A} \leftarrow \texttt{gen}(\text{seed}_{\boldsymbol{A}}) \in R_q^{l\times l}$		
3	$\boldsymbol{s} \leftarrow \beta_\mu(R_q^{l\times 1})$		$\boldsymbol{s}' \leftarrow \beta_\mu(R_q^{l\times 1})$
4	$\boldsymbol{b} = \texttt{bits}(\boldsymbol{A}\boldsymbol{s} + \boldsymbol{h}, \epsilon_q, \epsilon_p) \in R_p^{l\times 1}$	$\xrightarrow{\boldsymbol{b}, \text{seed}_{\boldsymbol{A}}}$	$\boldsymbol{A} \leftarrow \texttt{gen}(\text{seed}_{\boldsymbol{A}}) \in R_q^{l\times l}$
5			$\boldsymbol{b}' = \texttt{bits}(\boldsymbol{A}^T\boldsymbol{s}' + \boldsymbol{h}, \epsilon_q, \epsilon_p) \in R_p^{l\times 1}$
6			$v' = \boldsymbol{b}^T\texttt{bits}(\boldsymbol{s}', \epsilon_p, \epsilon_p) + h_1 \in R_p$
7	$v = \boldsymbol{b}'^T\texttt{bits}(\boldsymbol{s}, \epsilon_p, \epsilon_p) + h_1 \in R_p$	$\xleftarrow{\boldsymbol{b}', c}$	$c = \texttt{bits}(v', \epsilon_p - 1, \epsilon_t) \in R_t$
8	$k = \texttt{bits}(v - 2^{\epsilon_p-\epsilon_t-1}c + h_2, \epsilon_p, 1)$		$k' = \texttt{bits}(v', \epsilon_p, 1)$
9	$key_{\text{Alice}} = \texttt{kdf}(k)$		$key_{\text{Bob}} = \texttt{kdf}(k')$

Protocol 2: Saber.KE key exchange

All moduli involved in the scheme are chosen to be powers of 2, in particular we choose $q = 2^{\epsilon_q}$, $p = 2^{\epsilon_p}$ and $t = 2^{\epsilon_t}$ with $\epsilon_q > \epsilon_p > (\epsilon_t+1)$, so we have $2t \mid p \mid q$. In practice, our main parameter set will correspond to the case $\epsilon_q = 13$, $\epsilon_p = 10$ and $\epsilon_t = 3$. The secret vectors $\boldsymbol{s}$ and $\boldsymbol{s}'$ are sampled from $\beta_\mu(R_q^{l\times 1})$, with $\mu < p$, while the matrix $\boldsymbol{A} \in R_q^{l\times l}$ is sampled using a pseudorandom generator $\texttt{gen}()$ initialized with seed_A. The session key is obtained by feeding the common secret $k = k' \in R_2$ into a key derivation function $\texttt{kdf}()$. The algorithm also uses three constants: a constant vector $\boldsymbol{h} \in R_q^{l\times 1}$ consisting of polynomials all coefficients of which are set to the constant $2^{\epsilon_q-\epsilon_p-1}$, a constant polynomial $h_1 \in R_q$ with all coefficients equal to $2^{\epsilon_q-\epsilon_p-1}$, and a constant polynomial $h_2 \in R_q$ with all coefficients set equal to $(2^{\epsilon_p-2} - 2^{\epsilon_p-\epsilon_t-2})$. These constants are used to mimic rounding operations, which are necessary to reduce failure probability, while retaining the reduction to the underlying decisional Mod-LWR problem.

Note that the operations $\texttt{bits}(\boldsymbol{s}, \epsilon_p, \epsilon_p)$ in line **6** and $\texttt{bits}(\boldsymbol{s'}, \epsilon_p, \epsilon_p)$ in line **7** simply mean we are considering $\boldsymbol{s}$ mod p and $\boldsymbol{s'}$ mod p as elements in R_p which is well defined since $p \mid q$.

Correctness: Using Saber.KE two communicating parties agree on a common random key with overwhelming probability. A tight bound on the failure probability can be obtained using following observations from Bos et al. [17]: the reconciliation between two integer values $v_i, v'_i \in \mathbb{Z}_p$ is correct if the distance between v_i and v'_i is smaller than $p/4(1 - 1/t)$, and fails if the distance is bigger than $p/4(1 + 1/t)$. In between these values, the probability of success decreases linearly from 1 to 0. Consequently, a tight bound on the failure probability given the distribution of $\Delta v_i = v'_i - v_i$ can be calculated by adding to Δv_i a discrete uniformly distributed error $e_r \in \mathbb{Z}_p$ with range $[-p/4t, p/4t]$. The success probability of the reconciliation between v_i and v'_i then equals $Pr[|\Delta v_i + e_r| < p/4]$. Using the above observation we can estimate a bound on the error probability:

Theorem 1. *Let $\boldsymbol{A}$ be a matrix in $R_q^{l \times l}$ and $\boldsymbol{s}, \boldsymbol{s}'$ two vectors in $R_q^{l \times 1}$ sampled as in Protocol 2. Define $\boldsymbol{e}$ and $\boldsymbol{e}'$ as the rounding errors introduced by scaling and rounding $\boldsymbol{A}\boldsymbol{s}$ and $\boldsymbol{A}^T\boldsymbol{s}'$, i.e. $\texttt{bits}(\boldsymbol{A}\boldsymbol{s} + \boldsymbol{h}, \epsilon_q, \epsilon_p) = \frac{p}{q}\boldsymbol{A}\boldsymbol{s} + \boldsymbol{e}$ and $\texttt{bits}(\boldsymbol{A}^T\boldsymbol{s'} + \boldsymbol{h}, \epsilon_q, \epsilon_p) = \frac{p}{q}\boldsymbol{A}^T\boldsymbol{s'} + \boldsymbol{e'}$. Let $e_r \in R_q$ be a polynomial with uniformly distributed coefficients with range $[-p/4t, p/4t]$. If we set*

$$\delta = Pr[||(\boldsymbol{s}'^T\boldsymbol{e} - \boldsymbol{e}'^T\boldsymbol{s} + e_r) \mod p||_\infty > p/4]$$

then after executing the Saber.KE protocol, both communicating parties agree on a n-bit key with probability $1 - \delta$.

Proof. The polynomials v' and v calculated by Bob and Alice respectively in Protocol 2 are given as: $v' = (\frac{p}{q}\boldsymbol{s}'^T\boldsymbol{A}\boldsymbol{s} + h_1 + \boldsymbol{s}'^T\boldsymbol{e} \mod p)$ and $v = (\frac{p}{q}\boldsymbol{s}'^T\boldsymbol{A}\boldsymbol{s} + h_1 + \boldsymbol{e}'^T\boldsymbol{s} \mod p)$. Here, the coefficients of $\boldsymbol{e}, \boldsymbol{e}'$ are the rounding errors and so are in $(-1/2, 1/2]$. It can be easily seen that the values calculated by the communicating parties differ by $\Delta v = ||(\boldsymbol{s}'^T\boldsymbol{e} - \boldsymbol{e}'^T\boldsymbol{s}) \mod p||$. Therefore, Bob and Alice agree on the same secret if $||\Delta v + e_r||_\infty \leq \frac{p}{4}$. Hence, for $\delta = Pr[||(\boldsymbol{s}'^T\boldsymbol{e} - \boldsymbol{e}'^T\boldsymbol{s} + e_r) \mod p||_\infty > p/4]$ the Saber.KE protocol is $(1 - \delta)$ correct. □

Similar to Bos et al. [16], a tight upper bound on the value of δ is calculated using a Python script. To be able to practically compute the distribution of $\Delta v = v' - v \in R_p$, Bos et al. assume independence between the terms $\boldsymbol{s}'^T\boldsymbol{e}$ and $\boldsymbol{e}'^T\boldsymbol{s}$, which is not necessarily the case. Analogous to Theorem 5.2 from Jin and Zhao [33], one could argue that they are independent if conditioned on $\boldsymbol{s}'^T\boldsymbol{A}\boldsymbol{s} \equiv a \mod q/p$, where $a \in R_{q/p}$. The recommended parameter set described in Section 6.2 yields $\delta < 2^{-136}$.

Unbiased keys: Since our moduli are powers of 2 and as such non-prime, there exists (negligibly small) exceptional sets for $\boldsymbol{s}$ and $\boldsymbol{s'}$ such that the common key is biased. The intuition is that if all coefficients of the polynomials in $\boldsymbol{s}$ or $\boldsymbol{s'}$ are divisible by a high power of 2, the same property will hold for $\boldsymbol{A}\boldsymbol{s}$ or $\boldsymbol{A}^T\boldsymbol{s'}$, and their scaled versions. The following theorem however shows that outside these sets, uniformity is attained.

Theorem 2. *Let S_{bad} denote the set of elements in $R_q^{l\times 1}$ for which none of the coefficients w satisfies $\texttt{gcd}(w,q)|(q/p)$ and let S'_{bad} denote the set of elements in $R_q^{l\times 1}$ for which none of the coefficients w satisfies $\texttt{gcd}(w,p)|(p/2)$. Let $\boldsymbol{s},\boldsymbol{s'} \leftarrow \beta_\mu(R_q^{l\times 1})$ and let $\boldsymbol{A} \leftarrow \mathcal{U}(R_q^{l\times l})$ and determine k as follows:*

1. $\boldsymbol{b} = \texttt{bits}(\boldsymbol{As}+\boldsymbol{h},\epsilon_q,\epsilon_p)$
2. $k = \texttt{bits}(\boldsymbol{b}^T(\boldsymbol{s'} \mod p)+h_1,\epsilon_p,1)$

For $\boldsymbol{s} \notin S_{bad}$ and $\boldsymbol{s'} \notin S'_{bad}$, k is distributed uniformly for $\boldsymbol{A} \leftarrow \mathcal{U}(R_q^{l\times l})$. This occurs with a probability $Pr[\boldsymbol{s} \notin S_{bad}]Pr[\boldsymbol{s'} \notin S'_{bad}]$.

Proof. Note that the multiplication of a uniformly distributed coefficient of $\boldsymbol{A}$, by a coefficient w of $\boldsymbol{s}$, is uniformly distributed in its ϵ_p most significant bits if $\texttt{gcd}(w,q)|(q/p)$, which is equivalent to stating that $\lfloor pw/q \rceil$ is invertible in $\mathbb{Z}_p$.

The distribution of the coefficients of $\boldsymbol{b} = \texttt{bits}(\boldsymbol{As}+\boldsymbol{h},\epsilon_q,\epsilon_p)$ is as follows: since convolution of any distribution with a uniform distribution in $\mathbb{Z}_p$ results again in a uniform distribution in $\mathbb{Z}_p$, we need only one term of the summation step to be uniform in its p most significant bits. Therefore, the coefficients of $\boldsymbol{b}$ will be uniformly distributed if $\boldsymbol{s} \notin S_{\text{bad}}$.

Finally note that the distribution of $k' = \texttt{bits}(\boldsymbol{b}^T(\boldsymbol{s'} \mod p)+h_1,\epsilon_p,1)$ is uniform if $\boldsymbol{b}$ has a uniform distribution and if $\boldsymbol{s'} \notin s'_{\text{bad}}$. As above, a multiplication of a uniformly distributed coefficient of $\boldsymbol{b}$, with a coefficient w' of $\boldsymbol{s}$ is uniformly distributed in its most significant bit if $\texttt{gcd}(w',p)|(p/2)$. Therefore, k will be uniform if the coefficients of $\boldsymbol{b}$ are uniformly distributed and if $\boldsymbol{s'} \notin S'_{\text{bad}}$. The probability of a sampling $\boldsymbol{s}$ and $\boldsymbol{s'}$ so that k has a uniform distribution is thus $Pr[\boldsymbol{s} \notin S_{\text{bad}}]Pr[\boldsymbol{s'} \notin S'_{\text{bad}}]$. □

Since in our setting $\boldsymbol{s},\boldsymbol{s'}$ are sampled from $\beta_\mu(R_q)$, the coefficients are small and thus the only sampleable vector in S_{bad} and S'_{bad} is the all zero vector which occurs with probability 2^{-1436}. In the rest of the paper, we assume that the secret vectors are not in the vector sets: $\boldsymbol{s} \notin S_{\text{bad}}$ and $\boldsymbol{s'} \notin S'_{\text{bad}}$.

Security: The security of Saber.KE can be reduced to the decisional Mod-LWR problem as shown by the following theorem.

Theorem 3. *For any adversary A, there exist three adversaries B_0, B_1 and B_2 such that $Adv^{ind\text{-}rnd}_{Saber.KE}(A) \leqslant Adv^{prg}_{\texttt{gen()}}(B_0) + Adv^{mod\text{-}lwr}_{l,l,\mu,q,p}(B_1) + Adv^{mod\text{-}lwr}_{l+1,l,\mu,q,p}(B_2)$, if $q/p \leqslant p/(2t)$.*

Proof. The IND-RND security of our key exchange can be expressed as the probability that an adversary A can distinguish between k and a uniformly random key $\hat{k} \leftarrow \mathcal{U}(\mathcal{K})$, given the public information $\boldsymbol{A}$,$\boldsymbol{b}$,$\boldsymbol{b'}$ and c. The proof proceeds by a sequence of games G_i, where $\text{Adv}_{G_i}(A) = |Pr[S_{A,i}] - 1/2|$, in which $S_{A,i}$ is the event that the adversary guesses correctly in game G_i. The sequence of games is depicted in Figure 2.

The first game G_0 is the original game. In game G_1, the public matrix is no longer generated using the pseudorandom generator $\texttt{gen()}$, but is sampled from a uniformly random distribution. An adversary that can distinguish these

Game G_0:

```
1.  $seed_{\boldsymbol{A}} \leftarrow \mathcal{U}(\{0,1\}^{256})$
2.  $\boldsymbol{A} \leftarrow \texttt{gen}(\text{seed}_{\boldsymbol{A}})$
3.  $\boldsymbol{s}, \boldsymbol{s}' \leftarrow \beta_\eta(\mathbb{R}_q^{l\times 1})$
4.  $\boldsymbol{b} = \texttt{bits}($
    $\boldsymbol{A} \cdot \boldsymbol{s} + \boldsymbol{h}, \epsilon_q, \epsilon_p)$
5.  $\boldsymbol{b}' = \texttt{bits}($
    $\boldsymbol{A}^T \cdot \boldsymbol{s}' + \boldsymbol{h}, \epsilon_q, \epsilon_p)$
6.  $v' = \boldsymbol{b}^T \cdot \texttt{bits}(\boldsymbol{s}', \epsilon_p, \epsilon_p)$
    $+ h_1$
7.  $c = \texttt{bits}(v', \epsilon_p - 1, \epsilon_t)$
8.  $k' = \texttt{bits}(v', \epsilon_p, 1)$
9.  $\hat{k} \leftarrow \mathcal{U}(\mathbb{R}_2)$
10. $u \leftarrow \mathcal{U}(\{0,1\})$
11. if $u = 0$:
    return$(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{b}', c, k')$
12. else:
    return$(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{b}', c, \hat{k})$
```

Game G_1:

```
1.
2.  $\boldsymbol{A} \leftarrow \mathcal{U}(\mathbb{R}_q^{l\times l})$
3.  $\boldsymbol{s}, \boldsymbol{s}' \leftarrow \beta_\eta(\mathbb{R}_q^{l\times 1})$
4.  $\boldsymbol{b} = \texttt{bits}($
    $\boldsymbol{A} \cdot \boldsymbol{s} + \boldsymbol{h}, \epsilon_q, \epsilon_p)$
5.  $\boldsymbol{b}' = \texttt{bits}($
    $\boldsymbol{A}^T \cdot \boldsymbol{s}' + \boldsymbol{h}, \epsilon_q, \epsilon_p)$
6.  $v' = \boldsymbol{b}^T \cdot \texttt{bits}(\boldsymbol{s}', \epsilon_p, \epsilon_p)$
    $+ h_1$
7.  $c = \texttt{bits}(v', \epsilon_p - 1, \epsilon_t)$
8.  $k' = \texttt{bits}(v', \epsilon_p, 1)$
9.  $\hat{k} \leftarrow \mathcal{U}(\mathbb{R}_2)$
10. $u \leftarrow \mathcal{U}(\{0,1\})$
11. if $u = 0$:
    return$(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{b}', c, k')$
12. else:
    return$(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{b}', c, \hat{k})$
```

Game G_2:

```
1.
2.  $\boldsymbol{A} \leftarrow \mathcal{U}(\mathbb{R}_q^{l\times l})$
3.  $\boldsymbol{s}' \leftarrow \beta_\eta(\mathbb{R}_q^{l\times 1})$
4.  $\boldsymbol{b} \leftarrow \mathcal{U}(\mathbb{R}_p^{l\times 1})$
5.  $\boldsymbol{b}' = \texttt{bits}($
    $\boldsymbol{A}^T \cdot \boldsymbol{s}' + \boldsymbol{h}, \epsilon_q, \epsilon_p)$
6.  $v' = \boldsymbol{b}^T \cdot \texttt{bits}(\boldsymbol{s}', \epsilon_p, \epsilon_p)$
    $+ h_1$
7.  $c = \texttt{bits}(v', \epsilon_p - 1, \epsilon_t)$
8.  $k' = \texttt{bits}(v', \epsilon_p, 1)$
9.  $\hat{k} \leftarrow \mathcal{U}(\mathbb{R}_2)$
10. $u \leftarrow \mathcal{U}(\{0,1\})$
11. if $u = 0$:
    return$(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{b}', c, k')$
12. else:
    return$(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{b}', c, \hat{k})$
```

Game G_3:

```
2.  $\boldsymbol{A} \leftarrow \mathcal{U}(\mathbb{R}_q^{l\times l})$
3.  $\boldsymbol{s}' \leftarrow \beta_\eta(\mathbb{R}_q^{l\times 1})$
4.  $\boldsymbol{b} \leftarrow \mathcal{U}(\mathbb{R}_p^{l\times 1})$
5.  $\boldsymbol{b}' = \texttt{bits}($
    $\boldsymbol{A}^T \cdot \boldsymbol{s}' + \boldsymbol{h}, \epsilon_q, \epsilon_p)$
6.  $v' = \boldsymbol{b}^T \cdot \texttt{bits}(\boldsymbol{s}', \epsilon_p, \epsilon_p)$
    $+ h_1$
7.  $c = \texttt{bits}($
    $v', \epsilon_p - 1, 2\epsilon_p - \epsilon_q - 1)$
8.  $k' = \texttt{bits}(v', \epsilon_p, 1)$
9.  $\hat{k} \leftarrow \mathcal{U}(\mathbb{R}_2)$
10. $u \leftarrow \mathcal{U}(\{0,1\})$
11. if $u = 0$:
    return$(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{b}', c, k')$
12. else:
    return$(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{b}', c, \hat{k})$
```

Game G_4:

```
2.  $\boldsymbol{A} \leftarrow \mathcal{U}(\mathbb{R}_q^{l\times l})$
3.  $\boldsymbol{s}' \leftarrow \beta_\eta(\mathbb{R}_q^{l\times 1})$
4.  $\boldsymbol{b} \leftarrow \mathcal{U}(\mathbb{R}_q^{l\times 1})$
5.  $\boldsymbol{b}' = \texttt{bits}($
    $\boldsymbol{A}^T \cdot \boldsymbol{s}' + \boldsymbol{h}, \epsilon_q, \epsilon_p)$
6.  $v' = \texttt{bits}($
    $\boldsymbol{b}^T \cdot \boldsymbol{s}' + h_1, \epsilon_q, \epsilon_p)$
7.  $c = \texttt{bits}($
    $v', \epsilon_q - 1, \epsilon_p - 1)$
8.  $k' = \texttt{bits}(v', \epsilon_q, 1)$
9.  $\hat{k} \leftarrow \mathcal{U}(\mathbb{R}_2)$
10. $u \leftarrow \mathcal{U}(\{0,1\})$
11. if $u = 0$:
    return$(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{b}', c, k')$
12. else:
    return$(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{b}', c, \hat{k})$
```

Game G_5:

```
2.  $\boldsymbol{A} \leftarrow \mathcal{U}(\mathbb{R}_q^{l\times l})$
4.  $\boldsymbol{b} \leftarrow \mathcal{U}(\mathbb{R}_q^{l\times 1})$
5.  $\boldsymbol{b}' \leftarrow \mathcal{U}(\mathbb{R}_p^{l\times 1})$
6.  $v' \leftarrow \mathcal{U}(\mathbb{R}_p^{l\times 1})$
7.  $c = \texttt{bits}($
    $v', \epsilon_p - 1, \epsilon_p - 1)$
8.  $k' = \texttt{bits}(v', \epsilon_p, 1)$
9.  $\hat{k} \leftarrow \mathcal{U}(\mathbb{R}_2)$
10. $u \leftarrow \mathcal{U}(\{0,1\})$
11. if $u = 0$:
    return$(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{b}', c, k')$
12. else:
    return$(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{b}', c, \hat{k})$
```

Figure 2: Sequence of games that are used in the proof of Theorem 3

two games, can also distinguish the matrix generated through the pseudorandom generator from a uniformly random matrix, and therefore $|Pr[S_{A,0}] - Pr[S_{A,1}]| \leqslant \text{Adv}^{\text{prg}}_{\texttt{gen()}}(B_0)$.

During the second game G_2, the vector $\boldsymbol{b}$ is generated uniformly random, so that $(\boldsymbol{A}, \boldsymbol{b})$ is a uniformly distributed sample, in contrast to the first game G_1, where $(\boldsymbol{A}, \boldsymbol{b})$ forms a Mod-LWR sample. An adversary that can distinguish between game G_1 and G_2 has also solved the decisional Mod-LWR problem on this sample, and therefore $|Pr[S_{A,1}] - Pr[S_{A,2}]| \leqslant \text{Adv}^{\text{mod-lwr}}_{l,l,\mu,q,p}(B_1)$.

In game G_2, the number of bits dropped in the calculation of $\boldsymbol{b}'$ and c is $\epsilon_q - \epsilon_p$ and $\epsilon_p - \epsilon_t - 1$ respectively, which is reduced to $\epsilon_q - \epsilon_p$ in game G_3. If we compare G_3 to G_2, since $(\epsilon_q - \epsilon_p) \leqslant (\epsilon_p - \epsilon_t - 1)$, the number of dropped bits is the same or less, and therefore the number of available bits to the adversary

Algorithm 1: `Saber.KeyGen()`

```
$seed_{\boldsymbol{A}} \leftarrow \mathcal{U}(\{0,1\}^{256})$
$\boldsymbol{A} \leftarrow \texttt{gen}(\mathrm{seed}_{\boldsymbol{A}}) \in R_q^{l\times l}$
$\boldsymbol{s} \leftarrow \beta_\mu(R_q^{l\times 1})$
$\boldsymbol{b} = \texttt{bits}(\boldsymbol{As}+\boldsymbol{h}, \epsilon_q, \epsilon_p) \in R_p^{l\times 1}$
return $(pk := (\boldsymbol{b}, seed_{\boldsymbol{A}}), sk := \boldsymbol{s})$
```

is at least the same. From this we conclude that G_2 is at least as hard as G_3: $\forall A, \exists A' : \mathrm{Adv}_{G_2}(A) \leqslant \mathrm{Adv}_{G_3}(A')$.

Up to game G_3, the coefficients of the inputs for the generation of $\boldsymbol{b'}$ and c are in $\mathbb{Z}_q$ and $\mathbb{Z}_p$ respectively. This is evened up to coefficients in $\mathbb{Z}_q$ for all of the calculations in game G_4. Using $\boldsymbol{s'}$ instead of $\texttt{bits}(\boldsymbol{s'}, \epsilon_p, \epsilon_p)$ does not change the result of the multiplication because $\mu < p$. Since $p \mid q$, generating $\boldsymbol{b}$ from $\mathcal{U}(\mathbb{R}_q^{l\times 1})$ instead of $\mathcal{U}(\mathbb{R}_p^{l\times 1})$ makes the advantage of the adversary in Game G_4 at least as big as in game G_3, as the adversary in Game G_4 can easily calculate the same value for c as in Game G_3. Cutting off the last $\epsilon_q - \epsilon_p$ bits of v' does not change the game since they are not used in the rest of the protocol. Thus we can state: $\forall A', \exists A'' : \mathrm{Adv}_{G_3}(A') \leqslant \mathrm{Adv}_{G_4}(A'')$.

Analogous to game G_2, $\boldsymbol{b'}$ and c are replaced by a uniform random value in game G_5, so that the Mod-LWR samples $(\boldsymbol{A}, \boldsymbol{b'})$ and $(\boldsymbol{b}, v')$, which share secret key $\boldsymbol{s'}$, are replaced by uniformly random variables. Therefore, an adversary that can distinguish between these two games, can solve the corresponding Mod-LWR decisional problem and thus $|Pr[S_{A'',4}] - Pr[S_{A'',5}]| \leqslant \mathrm{Adv}^{\text{mod-lwr}}_{l+1,l,\mu,q,p}(B_2)$.

In the resulting game G_5, the keys are independent of the values $\boldsymbol{b}$, $\boldsymbol{b'}$ and v'. Moreover, since v' is uniformly distributed in $\mathbb{R}_p^{l\times 1}$, where q is a power of two, and since k' is generated as the first bit of v', k' is also uniformly distributed, and therefore $Pr[S_{A'',5}] = 1/2$. Working backwards from the probability of success in game G_5 to that in game G_0, and using the fact that $\mathrm{Adv}_{G_i}(A) = |Pr[S_{A,i}] - 1/2|$, gives the desired result. □

4 CPA secure encryption

The key exchange scheme of the previous section can be transformed into a CPA secure public-key encryption scheme Saber.PKE by using a similar transformation from Diffie-Hellman key exchange to ElGamal encryption, i.e. the messages sent by Alice now define her public key, and the encryption simply consists of an XOR with the common (pre)key.

The message space is $\mathcal{M} \in \{0,1\}^n$ and a message $m \in \mathcal{M}$ is represented as an element in R_q with coefficients in $\{0,1\}$. Algorithms 1 to 3 describe the public-key encryption scheme Saber.PKE=(`KeyGen`,`Enc`,`Dec`), where the setup parameters are the same as in the key-exchange scheme described before. If the optional parameter r is specified while calling Saber.ENC, it is used as a seed to generate the secret vector $\boldsymbol{s'}$.

Algorithm 2: `Saber.Enc`$(pk = (\boldsymbol{b}, seed_{\boldsymbol{A}}), m \in \mathcal{M}; r)$

```
1 $\boldsymbol{A} \leftarrow \texttt{gen}(\text{seed}_{\boldsymbol{A}}) \in R_q^{l \times l}$
2 $\boldsymbol{s'} \leftarrow \beta_\mu(R_q^{l \times 1})$
3 $\boldsymbol{b}' = \texttt{bits}(\boldsymbol{A}^T \boldsymbol{s}' + \boldsymbol{h}, \epsilon_q, \epsilon_p) \in R_p^{l \times 1}$
4 $v' = \boldsymbol{b}^T \texttt{bits}(\boldsymbol{s}', \epsilon_p, \epsilon_p) + h_1 \in R_p$
5 $c_m = \texttt{bits}(v' + 2^{\epsilon_p - 1} m, \epsilon_p, \epsilon_t + 1) \in R_{2t}$
6 return $c := (c_m, \boldsymbol{b'})$
```

Algorithm 3: `Saber.Dec`$(sk = \boldsymbol{s}, c_m, \boldsymbol{b'})$

```
1 $v = \boldsymbol{b}'^T \texttt{bits}(\boldsymbol{s}, \epsilon_p, \epsilon_p) + h_1 \in R_p$
2 $m' = \texttt{bits}(v - 2^{\epsilon_p - \epsilon_t - 1} c_m + h_2, \epsilon_p, 1) \in R_2$
3 return $m'$
```

Security and Correctness: It is easily seen that the security and correctness of the encryption scheme are equivalent to that of the key exchange introduced in Section 3.

Theorem 4. *For any adversary A against Saber.PKE, there exists an adversary B against Saber.KE such that* $Adv^{ind\text{-}cpa}_{Saber.PKE}(A) = Adv^{ind\text{-}rnd}_{Saber.KE}(B)$*. Furthermore, Saber.PKE is* $(1 - \delta)$ *correct if and only if Saber.KE is* $(1 - \delta)$ *correct.*

Proof. The proof proceeds by showing the equivalence between Saber.PKE and the combination of Saber.KE with a one time pad of the message m with k'_{KE}. Note that the most significant bit of each coefficient of v' is equal to the corresponding (pre)key bits of k' in Saber.KE. Therefore, in line 5 of the Alg. 2, the addition is essentially a one time pad of the message bits m with the coefficients of the (pre)key k' in the key exchange scheme (Protocol. 2). We can therefore conclude that the security of our encryption equals the security of our key exchange scheme for the same parameters. Similarly, it can be seen that Saber.PKE is correct if the keys k and k' are equal. Hence, the correctness of the encryption scheme is equivalent to the correctness of the key exchange in Protocol. 2.

□

5 CCA secure KEM

The CPA secure encryption scheme can be turned into a CCA secure KEM Saber.KEM=(`Encaps, Decaps`) using an appropriate transformation. Recently, several post-quantum versions [30,46,42,32] of the Fujisaki-Okamoto transform with corresponding security reductions have been developed. At this point, the FO$^{\not\perp}$ transformation in [30] with post-quantum reduction from Jiang et al. [32] gives the tightest reduction for schemes with non-perfect correctness. However, other transformations could be used to turn Saber.PKE into a CCA secure KEM.

```
Algorithm 4: Saber.Encaps($pk = (\boldsymbol{b}, seed_{\boldsymbol{A}})$)
$m \leftarrow \mathcal{U}(\{0,1\}^{256})$
$(\hat{K}, r) = \mathcal{G}(pk, m)$
$c =$ Saber.Enc$(pk, m; r)$
$K = \mathcal{H}(\hat{K}, c)$
return $(c, K)$
```

```
Algorithm 5: Saber.Decaps($sk = (\boldsymbol{s}, z), pk = (\boldsymbol{b}, seed_{\boldsymbol{A}}), c$)
$m' =$ Saber.Dec$(\boldsymbol{s}, c)$
$(\hat{K}', r') = \mathcal{G}(pk, m')$
$c' =$ Saber.Enc$(pk, m'; r')$
if $c = c'$ then
    return $K = \mathcal{H}(\hat{K}', c)$
else
    return $K = \mathcal{H}(z, c)$
```

Saber.KEM is described in detail in Algorithm 4 and 5. The functions $\mathcal{G} : \{0,1\}^* \rightarrow \{0,1\}^{l \times n}$ and $\mathcal{H} : \{0,1\}^* \rightarrow \{0,1\}^n$ are hash functions,z is a secret random seed used to return a pseudorandom response when the re-encryption fails, and the `Saber.Enc` and `Saber.Dec` functions are from the CPA secure asymmetric encryption described in Section 4.

Correctness: Following Hofheinz et al. [30], Saber.KEM is $(1 - \delta)$ correct if and only if Saber.PKE is $(1 - \delta)$ correct, and thus also if and only if Saber.KE is $(1 - \delta)$ correct.

Security: By modeling the hash functions $\mathcal{G}$ and $\mathcal{H}$ as random oracles, a lower bound on the CCA security can be proven. We use the security bounds of Hofheinz et al. [30], which considers a KEM variant of the Fujisaki-Okamoto transform that can also handle a small failure probability δ of the encryption scheme. This failure probability should be cryptographically negligibly small for the security to hold. Using Theorem 3.2 and Theorem 3.4 from [30], we get the following theorems for the security and correctness of our KEM in the random oracle model:

Theorem 5 (ROM, Hofheinz et al. [30])**.** *For a IND-CCA adversary B, making at most $q_{\mathcal{H}}$ and $q_{\mathcal{G}}$ queries to respectively the random oracle $\mathcal{G}$ and $\mathcal{H}$, and q_D queries to the decryption oracle, there exists an IND-CPA adversary A such that:*

$$Adv^{ind\text{-}cca}_{Saber.KEM}(B) \leqslant 3Adv^{ind\text{-}cpa}_{Saber.PKE}(A) + q_{\mathcal{G}}\delta + \frac{2q_{\mathcal{G}} + q_{\mathcal{H}} + 1}{2^{256}} \, .$$

Jiang et al. [32] also provide a security reduction against a quantum adversary in the quantum random oracle model from IND-CCA security to OW-CPA security. IND-CPA with a sufficiently large message space implies OW-CPA [30,13]. Therefore, we can reduce the IND-CCA security of Saber.KEM to the IND.CPA security of the underlying public key encryption:

Theorem 6 (QROM, Jiang et al. [32])**.** *For any IND-CCA quantum adversary B, making at most $q_{\mathcal{H}}$ and $q_{\mathcal{G}}$ queries to respectively the random quantum oracle $\mathcal{G}$ and $\mathcal{H}$, and q_D many (classical) queries to the decryption oracle, there exists an adversary A such that:*

$$Adv^{ind\text{-}cca}_{Saber.KEM}(B) \leqslant 2q_{\mathcal{H}}\frac{1}{\sqrt{2^{256}}} + 4q_{\mathcal{G}}\sqrt{\delta} + 2(q_{\mathcal{G}} + q_{\mathcal{H}})\sqrt{Adv^{ind\text{-}cpa}_{Saber.PKE}(A)}$$

Multi target protection: As described in [16], hashing the public key into $\hat{K}$ has two beneficial effects: it makes sure that K depends on the input of both parties, and it offers multi-target protection. In this scenario, the adversary uses Grover's algorithm to precompute an m that has a relatively high failure probability. Hashing pk into $\hat{K}$ ensures that an attacker is not able to use precomputed 'weak' values of m.

6 Security analysis and parameter selection

6.1 Security analysis

Our security analysis is similar to the one in 'a New Hope' [4]. The hardness of Mod-LWR is analyzed as an LWE problem, since there are no known attacks that make use of the Module or LWR structure. A set of l LWR samples given by with $\boldsymbol{A} \leftarrow \mathcal{U}(R_q^{l\times l})$ and $\boldsymbol{s} \leftarrow \beta_\mu(R_q^{l\times 1})$, can be rewritten as an LWE problem in the following way:

$$\left(\boldsymbol{A}, \left\lfloor \frac{p}{q}(\boldsymbol{A}\boldsymbol{s} \mod q) \right\rceil \mod p\right) = \left(\boldsymbol{A}, \frac{p}{q}(\boldsymbol{A}\boldsymbol{s} \mod q) + \boldsymbol{e} \mod p\right).$$

We can lift this to a problem modulo q by multiplying by $\frac{q}{p}$:

$$\frac{q}{p}\boldsymbol{b} = \boldsymbol{A}\boldsymbol{s} + \frac{q}{p}\boldsymbol{e} \mod q\,,$$

where $q/p\boldsymbol{e}$ is the random variable containing the error introduced by the rounding operation, of which the coefficients are discrete and nearly uniformly distributed in $(-q/2p, q/2p]$.

BKW type of attacks [35] and linearization attacks [7] are not feasible, since the number of samples is at most double the dimension of the lattice. Moreover, the secret vectors $\boldsymbol{s}$ and $\boldsymbol{s}'$ are dense enough to avoid the sparse secret attack described by Albrecht [2]. As a result, we end up with two main type of attacks: the primal and the dual attack, that make use of BKZ lattice reduction [20,43].

Weighted Primal Attack The primal attack constructs a lattice that has a unique shortest vector that contains the noise $\boldsymbol{e}$ and the secret $\boldsymbol{s}$. BKZ, with block dimension b, can be used to find this unique solution. An LWE sample $(\boldsymbol{A}, \boldsymbol{b} = \boldsymbol{A}\boldsymbol{s} + \boldsymbol{e}) \in \mathbb{Z}_q^{m\times n} \times \mathbb{Z}_q^m$ can be transformed to the following lattice: $\Lambda = \{\boldsymbol{v} \in \mathbb{Z}^{m+n+1} : (\boldsymbol{A}|\boldsymbol{I}_m| - \boldsymbol{b})\boldsymbol{v} = 0 \mod q\}$, with dimension $d = m + n + 1$ and volume q^m. The unique shortest vector in this lattice is $\boldsymbol{v} = (\boldsymbol{s}, \boldsymbol{e}, 1)$, and it has norm $\lambda \approx \sqrt{n\sigma_s^2 + m\sigma_e^2}$. Using heuristic models, the primal attack succeeds if [4]:

$$\sqrt{n\sigma_s^2 + m\sigma_e^2} < \delta^{2b-d-1}\mathrm{Vol}(\Lambda)^{\frac{1}{d}}$$
$$\text{where: } \delta = ((\pi b)^{\frac{1}{d}} \frac{b}{2\pi e})^{\frac{1}{2(b-1)}}$$

However, the vector $\boldsymbol{v} = (\boldsymbol{s}, \boldsymbol{e}, 1)$ is unbalanced since $||\boldsymbol{s}_i||$ is not necessarily equal to $||\boldsymbol{e}_i||$. In our case, $||\boldsymbol{s}_i|| < ||\boldsymbol{e}_i||$, which can be exploited by the lattice rescaling method described by Bai et al. [9], and further analysed in [22]. Analogous to [4], the primal attack is successful if the projected norm of the unique shortest vector on the last b Gram-Schmidt vectors is shorter than the $(d-b)^{\text{th}}$ Gram-Schmidt vector, or:

$$\sigma_s\sqrt{b} \leqslant \delta^{2b-d-1}\left(\frac{q}{\alpha}\right)^{\frac{m}{d}} .$$

Weighted Dual Attack The dual attack tries to distinguish between an LWE sample $(\boldsymbol{A}, \boldsymbol{b} = \boldsymbol{A}\boldsymbol{s} + \boldsymbol{e}) \in \mathbb{Z}_q^{m\times n} \times \mathbb{Z}_q^m$ and a uniformly random sample by finding a short vector $(\boldsymbol{v}, \boldsymbol{w})$ in the lattice $\Lambda = \{(\boldsymbol{x}, \boldsymbol{y}) \in \mathbb{Z}^m \times \mathbb{Z}^n : \boldsymbol{A}^T\boldsymbol{x} = \boldsymbol{y} \mod q\}$. This short vector is used to compute a distinguisher $z = \boldsymbol{v}\boldsymbol{b}$. If $\boldsymbol{b} = \boldsymbol{A}\boldsymbol{s} + \boldsymbol{e}$, we can write $z = \boldsymbol{v}\boldsymbol{A}\boldsymbol{s} + \boldsymbol{v}\boldsymbol{e} = \boldsymbol{w}\boldsymbol{s} + \boldsymbol{v}\boldsymbol{e}$, which is small and approximately Gaussian distributed. If $\boldsymbol{b}$ is generated uniformly, z will also be uniform mod q. Since in our case, $||\boldsymbol{s}_i|| < ||\boldsymbol{e}_i||$, we observe that the $\boldsymbol{w}\boldsymbol{s}$ term will be smaller than the $\boldsymbol{v}\boldsymbol{e}$ term. The weighted attack [9,22] optimizes the shortest vector so that these terms have a similar variance, by considering the weighted lattice $\Lambda' = \{(\boldsymbol{x}, \boldsymbol{y}') \in \mathbb{Z}^m \times (\alpha^{-1}\mathbb{Z})^n : (\boldsymbol{x}, \alpha\boldsymbol{y}') \in \Lambda \mod q\}$.

Following the strategy of [4], we can calculate the cost of the dual attack. The statistical distance between a uniformly distributed z and a Gaussian distributed z is bounded by $\epsilon = 4\exp(-2\pi^2\tau^2)$, where $\tau = ||\boldsymbol{u}||\sigma_e/q$. Since the key is hashed, an advantage of ϵ is not sufficient and must be repeated at least $R = \max(1, 1/(2^{0.2075b}\epsilon^2))$ times. The cost of the dual attack is thus equal to:

$$\mathrm{Cost}_{\mathrm{dual}} = \mathrm{Cost}_{\mathrm{BKZ}}R = b2^{cb}R, .$$

6.2 Parameter selection

We use a python script to choose parameters q, p and t for optimum usage of communication bandwidth, while achieving a quantum security level of 128 and failure probability 2^{-128}. Additional parameter sets are generated as Light and Fire versions of the Saber.KEM, a light and paranoid version respectively.

We would like to remark that choosing p and q as primes facilitates the use of NTT based polynomial multiplications [16,3]. However, rounding from R_q to R_p introduces significant bias as $p \nmid q$. Bogdanov et al. [15] proved the pseudorandomness of the LWR problem for moduli p and q for general lattices but left it as open problem for the ring version. However by choosing p and q

as a power-of-two, we can be assured of the pseudorandomness, which we also showed in Subsection. 3.

Sec Cat	fail prob	attack	Classical	Quantum	pk (B)	sk (B)	ciphertext (B)
LightSaber-KEM: $k = 2$, $n = 256$, $q = 2^{13}$, $p = 2^{10}$, $t = 2^2$, $\mu = 10$							
1	2^{-120}	primal dual	126 126	115 115	672	1568	736
Saber-KEM: $k = 3$, $n = 256$, $q = 2^{13}$, $p = 2^{10}$, $t = 2^3$, $\mu = 8$							
3	2^{-136}	primal dual	199 198	181 180	992	2304	1088
FireSaber-KEM: $k = 4$, $n = 256$, $q = 2^{13}$, $p = 2^{10}$, $t = 2^5$, $\mu = 6$							
5	2^{-165}	primal dual	270 270	246 245	1312	3040	1472

Table 1: Security and correctness of Saber.KEM.

7 Implementation

In this section, we describe a constant-time software implementation of Saber. Our implementation is relatively simpler than several existing lattice-based post-quantum key exchange schemes [16,4,17]. This is primarily due to the underlying LWR problem and our choice of power-of-two moduli. As the LWR problem inherently introduces errors, Saber can bypass error sampling operations unlike other LWE-based schemes. Our choice of power-of-two moduli results in faster arithmetic operations and does not require rejection sampling [4,16] for generating the random matrix A. In the remaining part of this section we describe the building blocks that are used to realize an efficient implementation of Saber.

Symmetric primitives The hash functions $\mathcal{G}$ and $\mathcal{H}$ in the CCA-secure Saber-KEM are implemented using SHA3-512 and SHA3-256 respectively, standardized in FIPS 202 [1]. For pseudorandom number generation, we use the extendable output function SHAKE-128 [1]. On parallel platforms, such as Intel processors that support 'single instruction multiple data' (SIMD), one can speedup pseudorandom number generation by using a vectorized implementation of SHAKE-128 and multiple seed values [16]. We decided to use SHAKE-128 serially to generate pseudorandom byte string of a required length from a given seed. This is mainly because of the fact that on majority of resource-constrained platforms (e.g., billions of IoT devices) SIMD would not be feasible, and hence multiple execution of SHAKE-128 would worsen performance (time and energy) because of the costly initialization operation [1] performed in each execution of SHAKE-128. Note that, it is essential for the correctness of the KEM, that all parties generate pseudorandomness in the same way.

Secret polynomial generation Saber requires sampling of secret polynomials from an error distribution. In lattice-based public-key cryptography this error distribution is usually a discrete Gaussian distribution. A significant number of papers [27,44,39,18,25,26] improve the performance of discrete Gaussian sampling. However, the implementation of a constant-time Gaussian sampler turns out to be a challenging problem [38,34]. This motivated the authors of NewHope [4] to use a centered binomial distribution instead of a Gaussian distribution. Sampling from a centered binomial distribution can be performed easily [4] in constant time by comparing the Hamming weights of two random integers of same length. Hence we use a centered binomial distribution β_μ with the parameter $\mu = 8$ to sample the secret polynomials.

Matrix A generation Since A consists of 9 polynomials, each having 256 13-bit coefficients, we use SHAKE-128 to generate $9 \cdot 256 \cdot 13/8 = 3,744$ pseudorandom bytes. Next we pack these bytes into the 13-bit coefficients of A. Note that in our case no additional rejection sampling is required as in Kyber, due to their use of a prime moduli. The rejection sampling wastes a portion of the generated pseudorandom bytes.

Polynomial arithmetic Our protocols relies heavily on polynomial arithmetic in the ring R_q with modulus $q = 2^{13}$ and the irreducible polynomial $f(x) = x^{256} + 1$. While polynomial addition and subtraction are simple coefficient-wise addition and subtraction operations, polynomial multiplication is a costly operation. An optimized polynomial multiplication routine is crucial for an efficient implementation of Saber. Since q is not a prime, we cannot apply the Number Theoretic Transform (NTT) unlike the key exchange schemes such as 'New Hope' [4], Kyber [16] etc. The next best alternative is the Karatsuba method which does not require any special modulus. Hence we use the Karatsuba polynomial multiplication method in Saber. The Karatsuba polynomial multiplication has a higher asymptotic complexity of $O(n^{\log_2 3})$. Though we lose in asymptotic time complexity, we gain in modular arithmetic since modular reduction comes for free. Furthermore, we found that the Karatsuba polynomial multiplication method is relatively easier to vectorize in modern Intel processors that support AVX/AVX2 'single instruction multiple data' (SIMD) instructions.

The Karatsuba multiplication method follows a top-down recursive approach: a 256-coefficient polynomial multiplication is split into three 128-coefficient polynomial multiplications, next each 128-coefficient polynomial multiplication is split into three 64-coefficient polynomial multiplications, and so on. After several levels of recursive splitting, when the polynomial size becomes small enough, i.e., reaches a particular threshold, a quadratic-complexity polynomial multiplication such as the School-book method is used to compute the smallest polynomial multiplications. If we set the threshold value to 16, then a 256-coefficient Karatsuba polynomial multiplication calls the School-book polynomial multiplication routine 81 times.

However, we can improve this by using the Toom-Cook polynomial multiplication. The Toom-Cook method is a generalization of the Karatsuba method and can be used to split a 256-coefficient polynomial multiplication into seven 64-coefficient polynomial multiplications. This is called *four-way Toom-Cook* multiplication. The smaller multiplications can be computed using the Karatsuba method as described above. Thus using the four-way Toom-Cook multiplication, the total number of calls to the School-book multiplication routine reduces to only 63 for a 256-coefficient polynomial multiplication.

In the Toom-Cook multiplication the choice of the evaluation points affects the computation time. Following [14], we choose the set of evaluation points to be $\{0, \pm 1/2, \pm 1, 2, \infty\}$. In the interpolation phase multiplications and divisions by scalar constants are performed. Divisions by odd scalars are performed by computing multiplications by their respective inverses. However, the inverse of an even divisor does not exist when the modulus is a power of two, which is true for Saber. For an even divisor we compute the division in two steps: first, we multiply by the inverse of the odd factor, then we compute a true division (i.e. right shifting) by the power-of-two factor since we know beforehand the division has to be exact. In the four-way Toom-Cook multiplication, the maximum power-of-two factor we have is 8, which could result in a loss of precision of 3 bits. Hence, during the interpolation phase, we allow the intermediate coefficients to grow by 3 bits such that the extra bits can be used to calculate the divisions by 2, 4 and 8. Our choice of modulus $q = 2^{13}$ is especially helpful since we can use 16-bit data variables (short integers in C) to store the 13-bit coefficients. The steps are shown in Algorithm 6 in Appendix A.

AVX2 implementation of polynomial multiplication Starting from Sandy Bridge, Intel provides AVX/AVX2 SIMD instructions that support computation on 128/256-bit vectors. We utilize this feature to achieve fast polynomial multiplication inspired by the software implementations of NTRU Prime [11] and NTRU KEM [31]. In Algorithm 6 the interpolation phase is trivial to vectorize. However, the evaluation phase, where 64-coefficient polynomial multiplications are performed requires special care to take advantage of vectorized instructions. We explain this below.

Assume that we want to compute 16 polynomial multiplications $C_0 \cdot D_0$, $C_1 \cdot D_1$, to $C_{15} \cdot D_{15}$ where each polynomial has 16 coefficients. Also assume that the polynomials are stored in two AVX2-arrays C_{AVX} and D_{AVX} as shown in Figure 3. The i-th coefficients of all C_j (and D_j) polynomials reside in the same AVX2 vectors. With such an arrangement it is easy to compute the 16 polynomial multiplications in a batch by multiplying the elements of C_{AVX} and D_{AVX}. We design the polynomial multiplier routine with the aim to obtain such an arrangement of coefficients during the threshold School-book multiplications. This is explained below.

The seven 64-coefficient polynomial multiplications in Algorithm 6 require 63 School-book multiplications of 16-coefficient polynomials. Since a 16-coefficient

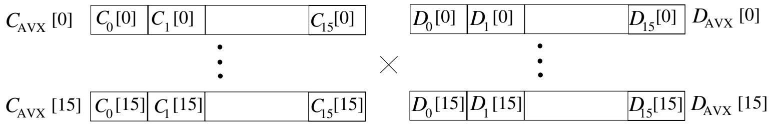

Figure 3: Arrangement of coefficients for batch polynomial multiplication

polynomial fits in an AVX2 vector, the 63 School-book multiplications can be computed in 4 batches using vectorized instructions. However, the batching is not trivial to implement. In the Karatsuba recursion, we do not immediately compute a School-book multiplication every time the recursion reaches the threshold condition. Instead, a *lazy approach* is adapted. We keep two ‘buckets’ each of which is an array of 16 AVX2 vectors. These buckets are gradually filled with the 16-coefficient polynomials that are the multiplicands of the School-book multiplications. Once the buckets are full, each of them can be viewed as a 16 × 16 matrix, containing 256 coefficients. Next we transpose the matrices using a sequence of AVX2 operations to reach the arrangement as shown in Figure 3. Now a batch multiplication is performed. The result is a collection of 31 vectors. This is again transposed to get the result of each 16-coefficient polynomial multiplication in two vectors. This lazy approach requires a bookkeeping which has a small overhead.

Table 2: Cycle count of the building blocks used in Saber and Kyber

Scheme	**Operation**	**Cycles**
Saber	Toom-Cook polynomial multiplication	3,439
`AVX2 optimized`	Sampling secret polynomial vector	13,656
	Generating random matrix $\boldsymbol{A}$ (serial SHAKE-128)	40,100
	Generating random matrix $\boldsymbol{A}$ (parallel SHAKE-128)[†]	25,300
Saber	Toom-Cook polynomial multiplication	20,520
`C`	Sampling secret polynomial vector	13,656
	Generating random matrix $\boldsymbol{A}$	54,707
Kyber	NTT	560
`AVX2 + assembly`	Inverse NTT	489
`optimized`	Sampling secret/error polynomial vector	10,545
	Generating random matrix $\boldsymbol{A}$ (parallel SHAKE-128)	32,601
Kyber	NTT	16,431
`C`	Inverse NTT	13,098
	Sampling secret/error polynomial vector	10,545
	Generating random matrix $\boldsymbol{A}$	69,620

[‡] Not used in Saber, see Sec. 7

8 Results

In Table 3, we compare our software implementation of Saber with software implementations of other lattice based post-quantum key exchange and encryption schemes. We compiled the Saber software using `gcc-7.1` with optimization flags `-O3` and measured computation time using a single core of a Intel(R) Core(TM) i7-6600U processor running at 2.60GHz with hyper-threading, Turbo-Boost, and multi-core support disabled on a Dell Latitude E7470 laptop with Ubuntu 16.04 operating system.

We remark that a totally fair comparison between the listed schemes and their software implementations is not possible since they are based on different hard problems, offer different levels of post-quantum security, implemented with different levels of optimizations and benchmarked on different platforms. Nevertheless, it is clear from the table that Saber is highly efficient both in terms of bandwidth and computation time.

The implementations of Saber and Kyber use similar building blocks namely polynomial multiplication, generation of random matrix $\boldsymbol{A}$, sampling of small secret (and error) polynomials and standard symmetric-key primitives for CCA transformations. In Table 2, we compare the performances of these building blocks excluding the symmetric-key primitives. Our Toom-Cook multiplication requires only 3,439 cycles. On the other hand, Kyber uses highly AVX-optimized NTT for polynomial multiplications. Furthermore, Kyber spends much less cycles in polynomial multiplications by generating the matrix A in the NTT domain directly and by keeping the secret polynomials in the NTT domain.

Saber does not require sampling of error polynomials, thus saving in computation time and entropy usage. As already described in Section 7 generating the random matrix A is faster in Saber (when same pseudorandom number generator is used) since rejection sampling is not performed, resulting in optimal usage of random numbers. Though in this paper we consider only software implementation on high-end Intel processors, we would like to remark that random number generation is very expensive on resource-constrained platforms. When we compare the high-level C implementations of Saber and Kyber, we see that Saber performs better than Kyber.

Finally note that at the expense of either using larger public keys, or caching the decompressed matrix $\boldsymbol{A}$, the implementation would run at least 25% faster.

Table 3: Performance and comparison of lattice-based KEMs and public-key encryption schemes. Cycles for key generation, encapsulation/encryption, and decapsulation/decryption are represented by **K**, **E**, and **D** respectively in the 5th column. Sizes of secret key (*sk*), public key (*pk*) and ciphertext (*c*) are reported in the last column. Constant-time implementations are marked with ✓in the column **ct?**. Performances are measured on the platform specified in the beginning of this section if not indicated otherwise.

Scheme	**Problem**	**Security**	**ct?**	**Cycles**	**Bytes**
		Passively secure KEMs			
NewHope [4] `AVX2 optimized`	Ring-LWE	255	✓	**K:** 88,920† **E:** 110,986† **D:** 19,422†	**sk:** 1,792 **pk:** 1,824 **c:** 2,048
Frodo [17]	LWE	130	✓	**K:** 2,938,000* **E:** 3,484,000* **D:** 338,000*	**sk:** 11,280 **pk:** 11,296 **c:** 11,288
		CCA-secure KEMs			
NTRU Prime [11]	NTRU	129	✓	**K:** 6,115,384$^{\otimes}$ **E:** 59,600$^{\otimes}$ **D:** 97,452$^{\otimes}$	**sk:** 1,600 **pk:** 1,218 **c:** 1,047
NTRU KEM [31] `AVX2 optimized`	NTRU	123	✓	**K:** 307,914$^{\perp}$ **E:** 48,646$^{\perp}$ **D:** 67,338$^{\perp}$	**sk:** 1,422 **pk:** 1,140 **c:** 1,281
spLWE-KEM [21]	spLWE	128	?	**K:** 336,700‡ **E:** 813,800‡ **D:** 785,200‡	**sk:** ? **pk:** ? **c:** 804
Kyber [16] `AVX2 + assembly optimized`	Module-LWE	161	✓	**K:** 92,461 **E:** 120,280 **D:** 113,718	**sk:** 2400 **pk:** 1088 **c:** 1152
Kyber [16] `C implementation`	Module-LWE	161	✓	**K:** 251,856 **E:** 336,112 **D:** 435,836	**sk:** 2400 **pk:** 1088 **c:** 1152
Saber `AVX2 optimized`	Module-LWR	180	✓	**K:** 101,138 **E:** 125,392 **D:** 129,138	**sk:** 2,304 **pk:** 992 **c:** 1,088
Saber `C implementation`	Module-LWR	180	✓	**K:** 190,420 **E:** 279,291 **D:** 306,346	**sk:** 2,304 **pk:** 992 **c:** 1,088
		CCA-secure public-key encryption schemes			
NTRUEncrypt [28]	NTRU	159	×	**K:** 1,194,816† **E:** 57,440† **D:** 110,604†	**sk:** 1120 **pk:** 1,027 **c:** 980
Lizard [22]	LWE,LWR	128	×	**K:** 97,573,000† **E:** 35,050† **D:** 80,840†	**sk:** 466,944$^{\bullet}$ **pk:** 2,031,616$^{\bullet}$ **c:** 1,072

† Compiled using `gcc-4.9.2` and benchmarked on Intel Core i7-4770K (Haswell) computer

* Benchmarked on a 2.6GHz Intel Xeon E5 (Sandy Bridge) with hyperthreading enabled.

$^{\otimes}$ Benchmarked on an Intel Haswell processor.

‡ Benchmarked on a Macbook Pro PC with 2.6GHz Intel Core i5.

$^{\bullet}$ Following the explanation provided in [16].

$^{\perp}$ Benchmarked on an Intel i7-Haswell, 3.5GHz processor.

9 Acknowledgements

This work was supported in part by the Research Council KU Leuven: C16/15/058. In addition, this work was supported by the European Commission through the Horizon 2020 research and innovation programme under grant agreement No H2020-ICT-2014-645622 PQCRYPTO, H2020-ICT-2014-644209 HEAT, Cathedral ERC Advanced Grant 695305 and in part by Flemish Government, by the Hercules Foundation AKUL/11/19.

References

1. 2015., National Institute of Standards and Technology.: SHA-3 standard: Permutation-Based Hash and Extendable-Output Functions. FIPS PUB 202 (2015)
2. Albrecht, M.R.: On dual lattice attacks against small-secret lwe and parameter choices in helib and seal. In: EUROCRYPT 2017. pp. 103–129 (2017), https://doi.org/10.1007/978-3-319-56614-6_4
3. Alkim, E., Ducas, L., Pöppelmann, T., Schwabe, P.: NewHope without reconciliation (2016), http://cryptojedi.org/papers/#newhopesimple
4. Alkim, E., Ducas, L., Pöppelmann, T., Schwabe, P.: Post-quantum key exchange – a new hope. In: USENIX Security 2016 (2016)
5. Alperin-Sheriff, J., Apon, D.: Dimension-preserving reductions from lwe to lwr. Cryptology ePrint Archive, Report 2016/589 (2016)
6. Alwen, J., Krenn, S., Pietrzak, K., Wichs, D.: Learning with rounding, revisited - new reduction, properties and applications. In: CRYPTO 2013. pp. 57–74 (2013), https://doi.org/10.1007/978-3-642-40041-4_4
7. Arora, S., Ge, R.: New algorithms for learning in presence of errors. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011. pp. 403–415 (2011), https://doi.org/10.1007/978-3-642-22006-7_34
8. Baan, H., Bhattacharaya, S., Garcia-Morchon, O., Rietman, R., Tolhuizen, L., Torre-Arce, J.L., Zhang, Z.: Round2: Kem and pke based on glwr. Cryptology ePrint Archive, Report 2017/1183 (2017), https://eprint.iacr.org/2017/1183
9. Bai, S., Galbraith, S.D.: Lattice Decoding Attacks on Binary LWE, pp. 322–337. Springer International Publishing, Cham (2014), https://doi.org/10.1007/978-3-319-08344-5_21
10. Banerjee, A., Peikert, C., Rosen, A.: Pseudorandom functions and lattices. In: EUROCRYPT 2012. pp. 719–737 (2012), https://doi.org/10.1007/978-3-642-29011-4_42
11. Bernstein, D.J., Chuengsatiansup, C., Lange, T., van Vredendaal, C.: Ntru prime: reducing attack surface at low cost. Cryptology ePrint Archive, Report 2016/461 (2016), http://eprint.iacr.org/2016/461
12. Bhattacharya, S., Garcia-Morchon, O., Rietman, R., Tolhuizen, L.: spkex: An optimized lattice-based key exchange. Cryptology ePrint Archive, Report 2017/709 (2017), http://eprint.iacr.org/2017/709
13. Birkett, J., Dent, A.W.: Relations among notions of plaintext awareness. In: PKC 2008. pp. 47–64 (2008), https://doi.org/10.1007/978-3-540-78440-1_4
14. Bodrato, M., Zanoni, A.: Integer and polynomial multiplication: Towards optimal toom-cook matrices. In: ISSAC '07. pp. 17–24. ACM (2007), http://doi.acm.org/10.1145/1277548.1277552

15. Bogdanov, A., Guo, S., Masny, D., Richelson, S., Rosen, A.: On the hardness of learning with rounding over small modulus. In: 13th International Conference on Theory of Cryptography. pp. 209–224 (2016), https://doi.org/10.1007/978-3-662-49096-9_9
16. Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Schwabe, P., Stehlé, D.: Crystals – kyber: a cca-secure module-lattice-based kem. Cryptology ePrint Archive, Report 2017/634 (2017), http://eprint.iacr.org/2017/634
17. Bos, J.W., Costello, C., Ducas, L., Mironov, I., Naehrig, M., Nikolaenko, V., Raghunathan, A., Stebila, D.: Frodo: Take off the ring! practical, quantum-secure key exchange from LWE. In: CCS 2016. pp. 1006–1018. ACM (2016), http://doi.acm.org/10.1145/2976749.2978425
18. Buchmann, J., Cabarcas, D., Göpfert, F., Hülsing, A., Weiden, P.: Discrete ziggurat: A time-memory trade-off for sampling from a gaussian distribution over the integers. In: Revised Selected Papers on Selected Areas in Cryptography – SAC 2013 - Volume 8282. pp. 402–417. Springer-Verlag New York, Inc., New York, NY, USA (2014), http://dx.doi.org/10.1007/978-3-662-43414-7_20
19. Chen, L., Jordan, S.P., Liu, Y.K., Moody, D., Peralta, R.C., Perlner, R.A., Smith-Tone, D.C.: Report on post-quantum cryptography. In: NIST Internal Report (NISTIR) - 8105 (2016), http://dx.doi.org/10.6028/NIST.IR.8105
20. Chen, Y., Nguyen, P.Q.: Bkz 2.0: Better lattice security estimates. In: ASIACRYPT 2011 (2011), https://doi.org/10.1007/978-3-642-25385-0_1
21. Cheon, J.H., Han, K., Kim, J., Lee, C., Son, Y.: A practical post-quantum public-key cryptosystem based on splwe. In: ICISC 2016. pp. 51–74 (2017), https://doi.org/10.1007/978-3-319-53177-9_3
22. Cheon, J.H., Kim, D., Lee, J., Song, Y.: Lizard: Cut off the tail! practical post-quantum public-key encryption from lwe and lwr. Cryptology ePrint Archive, Report 2016/1126 (2016), http://eprint.iacr.org/2016/1126
23. Diffie, W., Hellman, M.: New directions in cryptography. IEEE Transactions on Information Theory 22(6), 644–654 (Nov 1976)
24. Ding, J.: New cryptographic constructions using generalized learning with errors problem. Cryptology ePrint Archive, Report 2012/387 (2012), http://eprint.iacr.org/2012/387
25. Ducas, L., Durmus, A., Lepoint, T., Lyubashevsky, V.: Lattice signatures and bimodal gaussians. In: Canetti, R., Garay, J.A. (eds.) Advances in Cryptology – CRYPTO 2013: 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I, pp. 40–56. Springer Berlin Heidelberg, Berlin, Heidelberg (2013), http://dx.doi.org/10.1007/978-3-642-40041-4_3
26. Ducas, L., Nguyen, P.Q.: Faster gaussian lattice sampling using lazy floating-point arithmetic. In: Wang, X., Sako, K. (eds.) Advances in Cryptology – ASIACRYPT 2012: 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings, pp. 415–432. Springer Berlin Heidelberg, Berlin, Heidelberg (2012), http://dx.doi.org/10.1007/978-3-642-34961-4_26
27. Dwarakanath, N.C., Galbraith, S.D.: Sampling from discrete gaussians for lattice-based cryptography on a constrained device. Applicable Algebra in Engineering, Communication and Computing 25(3), 159–180 (2014), http://dx.doi.org/10.1007/s00200-014-0218-3
28. Hoffstein, J., Pipher, J., Schanck, J.M., Silverman, J.H., Whyte, W., Zhang, Z.: Choosing parameters for ntruencrypt. In: CT-RSA 2017. pp. 3–18 (2017), https://doi.org/10.1007/978-3-319-52153-4_1

29. Hoffstein, J., Pipher, J., Silverman, J.H.: Ntru: A ring-based public key cryptosystem. In: AANTS-III (1998), https://doi.org/10.1007/BFb0054868
30. Hofheinz, D., Hövelmanns, K., Kiltz, E.: A modular analysis of the fujisaki-okamoto transformation. Cryptology ePrint Archive, Report 2017/604 (2017), http://eprint.iacr.org/2017/604
31. H ulsing, A., Rijneveld, J., Schanck, J.M., Schwabe, P.: High-speed key encapsulation from ntru. Cryptology ePrint Archive, Report 2017/667 (2017), http://eprint.iacr.org/2017/667
32. Jiang, H., Zhang, Z., Chen, L., Wang, H., Ma, Z.: Post-quantum ind-cca-secure kem without additional hash. Cryptology ePrint Archive, Report 2017/1096 (2017), https://eprint.iacr.org/2017/1096
33. Jin, Z., Zhao, Y.: Optimal key consensus in presence of noise. Cryptology ePrint Archive, Report 2017/1058 (2017), https://eprint.iacr.org/2017/1058
34. Karmakar, A., Roy, S.S., Reparaz, O., Vercauteren, F., Verbauwhede, I.: Constant-time discrete gaussian sampling. IEEE Transactions on Computers pp. 1–1 (2018)
35. Kirchner, P., Fouque, P.: An improved BKW algorithm for LWE with applications to cryptography and lattices. In: CRYPTO 2015. pp. 43–62 (2015), https://doi.org/10.1007/978-3-662-47989-6_3
36. Langlois, A., Stehlé, D.: Worst-case to average-case reductions for module lattices. Designs, Codes and Cryptography 75(3), 565–599 (Jun 2015), https://doi.org/10.1007/s10623-014-9938-4
37. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: EUROCRYPT 2010 (2010), https://doi.org/10.1007/978-3-642-13190-5_1
38. Micciancio, D., Walter, M.: Gaussian sampling over the integers: Efficient, generic, constant-time. Cryptology ePrint Archive, Report 2017/259 (2017), http://eprint.iacr.org/2017/259
39. Peikert, C.: An efficient and parallel gaussian sampler for lattices. In: Rabin, T. (ed.) Advances in Cryptology – CRYPTO 2010: 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings, pp. 80–97. Springer Berlin Heidelberg, Berlin, Heidelberg (2010), http://dx.doi.org/10.1007/978-3-642-14623-7_5
40. Peikert, C.: Lattice cryptography for the internet. In: Mosca, M. (ed.) PQCrypto 2014 (2014), https://doi.org/10.1007/978-3-319-11659-4_12
41. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: STOC '05. pp. 84–93. ACM (2005), http://doi.acm.org/10.1145/1060590.1060603
42. Saito, T., Xagawa, K., Yamakawa, T.: Tightly-secure key-encapsulation mechanism in the quantum random oracle model. Cryptology ePrint Archive, Report 2017/1005 (2017), https://eprint.iacr.org/2017/1005
43. Schnorr, C.P., Euchner, M.: Lattice basis reduction: Improved practical algorithms and solving subset sum problems. Mathematical Programming (1994), https://doi.org/10.1007/BF01581144
44. Sinha Roy, S., Vercauteren, F., Verbauwhede, I.: High precision discrete gaussian sampling on fpgas. In: Lange, T., Lauter, K., Lisoněk, P. (eds.) Selected Areas in Cryptography – SAC 2013: 20th International Conference, Burnaby, BC, Canada, August 14-16, 2013, Revised Selected Papers. pp. 383–401. Springer Berlin Heidelberg, Berlin, Heidelberg (2014), https://doi.org/10.1007/978-3-662-43414-7_19

45. Stehlé, D., Steinfeld, R.: Making ntru as secure as worst-case problems over ideal lattices. In: EUROCRYPT 2011 (2011), https://doi.org/10.1007/978-3-642-20465-4_4

46. Targhi, E.E., Unruh, D.: Post-quantum security of the fujisaki-okamoto and oaep transforms. In: Theory of Cryptography: 14th International Conference (2016), https://doi.org/10.1007/978-3-662-53644-5_8

A Toom-Cook-4 polynomial multiplication.

Here we describe the Toom-Cook polynomial multiplication used in our implementation.

Algorithm 6: Toom-Cook Algorithm

```
Input: Two polynomials $A(x)$ and $B(x)$of degree $n = 256$
Output: $C(x) = A(x) * b(x)$
// Splitting $A(x)$ into four polynomials of size 64
$A(y) = A_3 \cdot y^3 + A_2 \cdot y^2 + A_1 \cdot y + A_0$ where $y = x^{64}$
// Splitting $B(x)$ into four polynomials of size 64
$B(y) = B_3 \cdot y^3 + B_2 \cdot y^2 + B_1 \cdot y + B_0$
// Evaluation of the polynomials at $y = \{0, \pm 1, \pm\frac{1}{2}, 2, \infty\}$. These
   multiplications are computed using Karatsuba
$w_1 = A(\infty) * B(\infty) = A_3 * B_3$
$w_2 = A(2) * B(2) = (A_0 + 2 \cdot A_1 + 4 \cdot A_2 + 8 \cdot A_3) * (B_0 + 2 \cdot B_1 + 4 \cdot B_2 + 8 \cdot B_3)$
$w_3 = A(1) * B(1) = (A_0 + A_1 + A_2 + A_3) * (B_0 + B_1 + B_2 + B_3)$
$w_4 = A(-1) * B(-1) = (A_0 - A_1 + A_2 - A_3) * (B_0 - B_1 + B_2 - B_3)$
$w_5 = A(\frac{1}{2}) * B(\frac{1}{2}) = (8 \cdot A_0 + 4 \cdot A_1 + 2 \cdot A_2 + A_3) * (8 \cdot B_0 + 4 \cdot B_1 + 2 \cdot B_2 + B_3)$
$w_6 = A(\frac{-1}{2}) * B(\frac{-1}{2}) = (8 \cdot A_0 - 4 \cdot A_1 + 2 \cdot A_2 - A_3) * (8 \cdot B_0 - 4 \cdot B_1 + 2 \cdot B_2 - B_3)$
$w_7 = A(0) * B(0) = A_0 * B_0$
// Interpolation
$w_2 = w_2 + w_5$
$w_6 = w_6 - w_5$
$w_4 = (w_4 - w_3)/2$
$w_5 = w_5 - w_1 - 64 \cdot w_7$
$w_3 = w_3 + w_4$
$w_5 = 2 \cdot w_5 + w_6$
$w_2 = w_2 - 65 \cdot w_3$
$w_3 = w_3 - w_7 - w_1$
$w_2 = w_2 + 45 \cdot w_3$
$w_5 = (w_5 - 8 \cdot w_3)/24$
$w_6 = w_6 + w_2$
$w_2 = (w_2 + 16 \cdot w_4)/18$
$w_3 = w_3 - w_5$
$w_4 = -(w_4 + w_2)$
$w_6 = (30 \cdot w_2 - w_6)/60$
$w_2 = w_2 - w_6$
return $w_1 \cdot y^6 + w_2 \cdot y^5 + w_3 \cdot y^4 + w_4 \cdot y^3 + w_5 \cdot y^2 + w_6 \cdot y + w_7$;
```