# SMAUG: the Key Exchange Algorithm based on Module-LWE and Module-LWR[⋆]

Jung Hee Cheon[1,2], Hyeongmin Choe[1], Dongyeon Hong[2], Jeongdae Hong[3], Hyoeun Seong[2], Junbum Shin[2], and MinJune Yi[1,2]

[1] Seoul National University
{jhcheon, sixtail528, yiminjune}@snu.ac.kr
[2] CryptoLab Inc.
{decenthong93, jungwoo.kim, she000, junbum.shin}@cryptolab.co.kr
[3] Ministry of National Defense
ghjd2000@gmail.com

**Abstract.** In this documentation, we introduce a new lattice-based post-quantum key encapsulation mechanism (KEM), a type of key exchange algorithm, which we are submitting to Korean Post-Quantum Cryptography Competition. Based on the hardness of the MLWE and MLWR problems, defined in module lattices, we design an efficient public key encryption (PKE) scheme SMAUG.PKE and KEM scheme SMAUG.KEM With the choice of sparse secret, SMAUG.KEM has 16% smaller public key sizes and at most 18% smaller ciphertext sizes with 30-45% faster running-time compared to NIST's 2022 standard KEM Kyber. Compared to Saber, one of the NIST's finalist KEM, we have similar sizes but with a more conservative key generation. We base the key security to MLWE and use MLWR only for the faster encryption.

**Keywords:** Lattice-based Cryptography · Post-Quantum Cryptography · Key Encapsulation Mechanism.

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

## 1 Introduction

SMAUG is an efficient post-quantum key encapsulation mechanism, whose security is based on the hardness of the lattice problems. The IND-CPA security of SMAUG.PKE relies on the hardness of MLWE problem and MLWR problem, which implies the IND-CCA2 security of SMAUG.KEM.

Our SMAUG.KEM scheme follows the approaches in recent constructions of post-quantum KEMs such as Lizard [18] and RLizard [32], as the key security relies on the MLWE problem and the ciphertext security relies on the MLWR problem. SMAUG consists of an underlying public key encryption (PKE) scheme SMAUG.PKE, which can be turned into SMAUG.KEM via Fujisaki-Okamoto transform. We also use the tweaks for the quantum FO transforms [33], also used in Kyber [14] and Saber [21].

By using the module variants MLWE and MLWR problems, we reduce the size of the public key and the ciphertext sizes. Compared to Kyber, selected as NIST PQC standard in 2022, our SMAUG.KEM has $\approx 16\%$ smaller public key sizes and at most 18% smaller ciphertext sizes for the same security levels. The first reference implementation, included in our submission, has 30-45% faster running-time.

### 1.1 Design rationale

**MLWE and MLWR.** SMAUG is constructed on the hardness of MLWE (Module-Learning with Errors) and MLWR (Module-Learning with Rounding) problems and follow the key structure of Lizard [18] and Ring-Lizard (RLizard) [32]. Since LWE problem is a well-studied problem for last two decades, there are many LWE-based schemes (e.g Frodo [13]). Ring and module LWE problems are variants defined over structured lattices and regarded as hard as LWE. Many schemes base their security on RLWE/MLWE (e.g. NewHope [5], Kyber [14] and Saber [21]) for efficiency reasons. We also chose the module structure which enables us to fine-tune security and efficiency in a much more scalable way, unlike standard and ring versions. Since MLWR problem is regarded as hard as MLWE problem unless we overuse the same secret to generate the samples [12], we chose to use MLWR samples for the encryption. By basing the security of encryption to MLWR, we reduce the ciphertext size in factor of $\log q / \log p$ than MLWE instances so that more efficient encryption and decryption is possible.

**Quantum Fujisaki-Okamoto transform.** SMAUG consists of key encapsulation mechanism SMAUG. KEM and public key encryption scheme SMAUG.PKE. On top of the PKE scheme, we construct the KEM scheme using the Fujisaki-Okamoto (FO) transform [23, 24]. Line of works on FO transform in the quantum random oracle model [11, 27, 29, 33] make it possible to analyze the quantum security of our KEM scheme SMAUG constructed upon FO transform.

**Sparse secret key and ephemeral key.** We design the key generation algorithm based on MLWE problem using sparse secret, since MLWE with sparse secret is still hard to solve if the min-entropy of the secret distribution is sufficiently large [17]. In detail, we use sparse ternary secret and ephemeral polynomials. Based on the reduction from the general MLWE problem to the MLWE problem with sparse secret, we can take the advantage of the sparsity, that the secret and public keys have smaller sizes and that an easier estimation of the error bound is possible.

**Choice of moduli.** All our integer moduli are powers of 2. As described in Saber, this has some advantages: (1) simple modular reduction by bit shifting, (2) sampling uniformly modulo a power of 2 is easy. (3) The condition $p|q$ implies that the scaling operation maps the uniform distribution modulo $q$ to the uniform distribution modulo $p$.

**No Number Theoretic Transform.** By choosing moduli $q$ and $p$ power of 2 integers, our moduli are not suitable for Number Theoretic Transform (NTT) which is a useful technique for fast polynomial multiplications. It would be slower than using the primes suitable for NTT with similar sizes of moduli, but there is no noticeable slowdown since the degree of the polynomials is small and also the size of modulus we use is much smaller.

**Additional hashes.** (No Hash 어때요. No NTT 와 가은 형태로) (HM: 이 문단은 굳이 필요할까요?) In Lizard and RLizard, there is a hash value $d$ used in checking decryption failure and being one of the ciphertext components. This came from Dent [19], but Jiang et al. [29] showed that $d$ is not necessary. Hence, we construct the KEM scheme without the hash value $d$ in ciphertexts. This reduces ciphertext size and skips additional hash operations. And like Kyber, we use a hash of the public key and ciphertext during encapsulation and computation of the final key, respectively. These additional hashes are not necessary for security reduction, but make our scheme contributory and prevent certain kinds of multi-target attacks.

**Negligible decapsulation failure probability.** Since we base the security to MLWE and MLWR, which have errors inherently, the decryption result of a SMAUG.PKE ciphertext may different to the original message. Hence we make the probability of decryption failure to be negligible. For this, we fine-tuned the parameters to trade-off between reducing the size of errors (which reduces the decryption failure probability), increasing the security, and making the scheme more efficient. In case of the decryption failure in SMAUG.KEM, we return a pseudo-random value computed by the secret bytes and hash of ciphertext. By doing so, the security in the QROM with decryption failure can be guaranteed via a FO variant proposed in [27].

We give estimated security as well as sizes for our parameter sets in Table 1. The full parameters sets can be found in Section 3.5. The security is estimated via lattice estimator [2]. The decryption failure probability is calculated via a python script modified from Lizard and RLizard. The python script for the security and the decryption failure probability estimator is included in the submission file. The probability is given in logarithm with base two. The sizes are shown in bytes. We give the secret key sizes which is ready to use, not a seed, which is beneficial in some restricted devices. The sizes in parenthesis are the sizes of the secret key and the public key.

| Parameters sets | SMAUG128 | SMAUG192 | SMAUG256 |
|---|---|---|---|
| Target security | I | III | V |
| $n$ | 256 | 256 | 256 |
| $k$ | 2 | 3 | 5 |
| $(p, q)$ | $(1024, 256)$ | $(1024, 256)$ | $(1024, 256)$ |
| Classical core-SVP hardness | 120.0 | 180.2 | 260.3 |
| Quantum core-SVP hardness | 105.6 | 164.7 | 243.8 |
| Decryption failure probability | -177.2 | -160.4 | -168.0 |
| Secret key size | 174 (846) | 185 (1177) | 182 (1814) |
| Public key size | 672 | 992 | 1632 |
| Ciphertext size | 768 | 1024 | 1536 |

**Table 1.** Security and sizes for our parameter sets.

### 1.2 Advantages and limitations

**Advantages**

– Our scheme relies on the difficulty of hard lattice problem LWE, which has been well-studied for the last two decades.
– In terms of size, it presents the lower sizes compared to Kyber, and also similar to or less then to Saber.
– We follow the FO-transform approaches in the ROM/QROM setting with some tweaks used in Kyber, and also previously shown by many line of works.
– Implementation-wise, use of MLWR samples in encryption makes it much easier to simply implement the encryption, which will also make the secure implementation possible.
– We give the C reference code, which proves the completeness of the scheme.

**Limitations**

– We use MLWR problem which has been studied shorter than MLWE or LWE problems. However, it has a reduction to MLWE, so the security is guaranteed. We choose to use it for the encryption to reduce the sizes and achieve efficiency as in Saber, but in a more conservative way.

- MLWE problem with sparse secret has a similar issue, but has been studied more and is used in various applications for e.g. homomorphic encryptions.

## 2 Preliminaries

### 2.1 Notation

We denote matrices with bold type and upper case letters (e.g. $\mathbf{A}$) and vectors with bold type and lower case letters (e.g. $\mathbf{b}$). Unless otherwise stated, the vector is a column vector.

We define a polynomial ring $\mathcal{R} = \mathbb{Z}[x]/(x^n + 1)$ where $n$ is a power of 2 integer and denote a quotient ring by $\mathcal{R}_q = \mathbb{Z}[x]/(q, x^n + 1) = \mathbb{Z}_q[x]/(x^n + 1)$ for a positive integer $q$. For an integer $\eta$, we denote the set of polynomials of degree less than $n$ with coefficients in $[-\eta, \eta] \cap \mathbb{Z}$ as $S_\eta$. Let $\tilde{S}_\eta$ be a set of polynomials of degree less than $n$ with coefficients in $[-\eta, \eta) \cap \mathbb{Z}$.

### 2.2 Lattice assumptions

We first define some well-known lattice assumptions $\mathsf{MLWE}$ and $\mathsf{MLWR}$ on the structured Euclidean lattices.

**Definition 1 (Decision-$\mathsf{MLWE}_{n,q,k,\ell,\eta}$).** *For positive integers $q, k, \ell, \eta$ and the dimension $n$ of $\mathcal{R}$, we say that the advantage of an adversary $\mathcal{A}$ solving the decision-$\mathsf{MLWE}_{n,q,k,\ell,\eta}$ problem is*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{MLWE}}_{n,q,k,\ell,\eta}(\mathcal{A}) = \big| &\Pr\left[b = 1 \mid \mathbf{A} \leftarrow \mathcal{R}_q^{k\times\ell}; \mathbf{b} \leftarrow \mathcal{R}_q^k; b \leftarrow \mathcal{A}(\mathbf{A}, \mathbf{b})\right] \\ &- \Pr\left[b = 1 \mid \mathbf{A} \leftarrow \mathcal{R}_q^{k\times\ell}; (\mathbf{s}, \mathbf{e}) \leftarrow S_\eta^\ell \times S_\eta^k; b \leftarrow \mathcal{A}(\mathbf{A}, \mathbf{A}\cdot\mathbf{s} + \mathbf{e})\right] \big|\end{aligned}$$

**Definition 2 (Decision-$\mathsf{MLWR}_{n,p,q,k,\ell,\eta}$).** *For positive integers $p, q, k, \ell, \eta$ with $q \geq p \geq 2$ and the dimension $n$ of $\mathcal{R}$, we say that the advantage of an adversary $\mathcal{A}$ solving the decision-$\mathsf{MLWR}_{n,p,q,k,\ell,\eta}$ problem is*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{MLWE}}_{n,p,q,k,\ell,\eta}(\mathcal{A}) = \big| &\Pr\left[b = 1 \mid \mathbf{A} \leftarrow \mathcal{R}_p^{k\times\ell}; \mathbf{b} \leftarrow \mathcal{R}_q^k; b \leftarrow \mathcal{A}(\mathbf{A}, \mathbf{b})\right] \\ &- \Pr\left[b = 1 \mid \mathbf{A} \leftarrow \mathcal{R}_q^{k\times\ell}; \mathbf{s} \leftarrow S_\eta^\ell; b \leftarrow \mathcal{A}(\mathbf{A}, \lfloor p/q \cdot \mathbf{A}\cdot\mathbf{s}\rceil)\right] \big|\end{aligned}$$

## 3 Specification

SMAUG.CCAKEM is an IND-CCA2-secure key-encapsulation mechanism (KEM) based on the hardness of module-LWE and module-LWR. SMAUG.CCAKEM is consisted of two steps: (1) first introduce an IND-CPA public-key encryption scheme encrypting messages of a fixed length of 32 bytes, which we call SMAUG.CPAPKE. (2) Then we use a slightly tweaked Fujisaki-Okamoto (FO) transform suggested by Kyber to construct the IND-CCA2-secure KEM. SMAUG means SMAUG.CCAKEM unless otherwise stated.

### 3.1 Choice of symmetric primitives

We use two hash H : $\{0,1\}^{256} \longrightarrow \{0,1\}^{256}$, G : $\mathcal{M} \times \{0,1\}^{256} \longrightarrow \{0,1\}^{256}$, one extendable function XOF, and one key derivation function KDF : $\mathcal{M} \times \{0,1\}^{256} \longrightarrow \{0,1\}^{256}$. H is used for hashing a public key and ciphertext, G is used for generation of a seed of $\mathbf{r}$, XOF is used for generation of seeds of $\mathbf{A}$ and $\mathbf{s}$, and KDF is used for key derivation of a final shared key. We instantiate these primitives as follows:

- H is instantiated with sha3-256;
- G is instantiated with sha3-512;
- XOF is instantiated with shake-128;
- KDF is instantiated with shake-256.

### 3.2 Specification of IND-CPA PKE of Smaug

Smaug is similar to Lizard [18] and Ring-Lizard (RLizard) [32]. Lizard and RLizard are the first KEM schemes based on LWE and LWR, and their ring version, respectively. They have small secret key size and simple implementation. However, it is hard to scale between efficiency and sizes. The main difference from the Lizard and RLizard is to use module structure instead of LWE and LWR or their ring variant.

For generation of $\mathbf{A}$, we follow Saber [21]'s approach since they do not use NTT and their moduli are quite close with ours and expandA of line 3 of Algorithm 1 is denoted the function generating a uniformly random matrix $\mathbf{A}$ (see implementation details). We transpose the $\mathbf{A}$ while computing $\mathbf{b}$ and not transpose $\mathbf{A}$ in Smaug.CPAPKE.Encrypt as same reasons in Saber.

Smaug samples $\mathbf{s}$ from sparse ternary distribution with hamming weight $h_s$ and $\mathbf{e}$ from discrete Gaussian distribution. $\mathsf{HWT}_h$ is denoted a sampling function from a distribution of $\{-1,0,1\}^n$ with the number of -1 and 1 is $h$ and discreteGaussian is denoted a sampling function from discrete Gaussian distribution with standard deviation $\sigma = 1.0625$. The discrete Gaussian sampling could be one of threat points of side-channel attack, however, to avoid it we followed the constant-time implementation suggested by Karmakar et al. [30].

To become an deterministic INC-CPA PKE scheme, we can take a seed of $\mathbf{r}$ as an input of SMAUG.CPAPKE.Encrypt.

**Algorithm 1** SMAUG.CPAPKE.KeyGen: key generation

```
Output $pk = (\rho, \mathbf{b}) \in \{0,1\}^{256} \times R_q^k$
Output $sk = \mathbf{s} \in S_\eta^k$
1: $seed \leftarrow \{0,1\}^{256}$
2: $(\rho, \tau) \leftarrow \mathrm{XOF}(seed)$
3: $\mathbf{A} \leftarrow \mathsf{expandA}(\rho) \in R_q^{k \times k}$
4: $\mathbf{s} \leftarrow \mathsf{HWT}_{h_s}(\tau) \in S_\eta^k$
5: $\mathbf{e} \leftarrow \mathsf{discreteGaussian}(\tau) \in R^k$
6: $\mathbf{b} = -\mathbf{A}^T \cdot \mathbf{s} + \mathbf{e} \mod q$
7: $pk = (\rho, \mathbf{b})$
8: $sk = \mathbf{s}$
9: Return $(pk, sk)$
```

**Algorithm 2** SMAUG.CPAPKE.Encrypt: encryption

```
Input $pk = (\rho, \mathbf{b})$
Input message $\mu \in \{0,1\}^{256}$
Input(Optional) a seed $\rho' \in \{0,1\}^{256}$
Output $ctxt = (\mathbf{c_1}, c_2) \in R_p^k \times R_p$
1: if $\rho'$ is not given then
2:     $\rho' \leftarrow \{0,1\}^{256}$
3: end if
4: $\mathbf{r} \leftarrow \mathsf{HWT}_{h_r}(\rho') \in S_\eta^k$
5: $\mathbf{c_1} \coloneqq \lfloor p/q \cdot \mathbf{A} \cdot \mathbf{r} \rceil \in R_p^k$
6: $c_2 \coloneqq \lfloor p/q \cdot \left(\mathbf{b}^T \cdot \mathbf{r} + q/t \cdot \mu\right) \rceil \in R_p$
7: Return $ctxt = (\mathbf{c_1}, c_2)$
```

**Algorithm 3** SMAUG.CPAPKE.Decrypt: decryption

```
Input $sk = \mathbf{s} \in S_\eta^k$
Input $ctxt = (\mathbf{c_1}, c_2) \in R_p^k \times R_p$
Output message $\mu' \in \{0,1\}^{256}$
1: $\mu' \coloneqq \lfloor t/p \cdot (c_2 + \mathbf{c_1}^T \cdot \mathbf{s}) \rceil$
2: Return $\mu'$
```

## 3.3 Correctness

**Theorem 1.** *Let* $\mathbf{A}$, $\mathbf{b}$, $\mathbf{s}$, $\mathbf{e}$ *and* $\mathbf{r}$ *are defined as in* SMAUG.CPAPKE*. Let* $\mathbf{e}_1 \in \mathcal{R}_{\mathbb{Q}}^k$ *and* $e_2 \in \mathcal{R}_{\mathbb{Q}}$ *be the rounding errors introduced by scaling and rounding* $\mathbf{A} \cdot \mathbf{r}$ *and* $\mathbf{b}^T \cdot \mathbf{r}$*, i.e.* $\mathbf{e}_1 = \frac{q}{p} \cdot \lfloor \frac{p}{q} \cdot \mathbf{A} \cdot \mathbf{r} \mod p \rceil - (\mathbf{A} \cdot \mathbf{r} \mod q)$ *and* $e_2 = \frac{q}{p} \cdot \lfloor \frac{p}{q} \cdot \mathbf{b}^T \mathbf{r} \mod p \rceil - (\mathbf{b}^T \mathbf{r} \mod q)$*. If we set*

$$\delta = \Pr\left[\|\mathbf{r}^T \mathbf{e} + \mathbf{s}^T \mathbf{e}_1 + e_2\|_\infty > q/2t\right],$$

*then the decryption failure probability of* SMAUG.CPAPKE *scheme is less than* $\delta$*.*

*Proof.* See Appendix A

Recall that $\mathbf{e} \leftarrow \mathsf{discreteGaussian}(\cdot) \in R^k$ and $\mathbf{r} \leftarrow \mathsf{HWT}_{h_r}(\cdot) \in S_\eta^k$. We can also compute the distribution of the rounding errors as in [21].

In Table 2, we give the failure probabilities for the parameter sets in logarithm of base two. They are computed using the python script included in the submission file. The script is modified from the script of Lizard and RLizard [28].

| Parameters sets | SMAUG128 | SMAUG192 | SMAUG256 |
|---|---|---|---|
| Decryption failure probability | -177.2 | -160.4 | -168.0 |

**Table 2.** Decryption failure probability for our parameter sets.

### 3.4 Specification of IND-CCA2 KEM of Smaug

SMAUG.CCAKEM is constructed from SMAUG.CPAKEM using Fujisaki-Okamoto transformation [23]. We adopt the tweaked version of FO transformation proposed in Kyber.

During encryption, a hash of public key is taken as an input of SMAUG.CPAPKE .Encrypt along with message. This prevents multi-target attack.

The differences from Lizard and RLizard are (1) a hash of public key for generating $\mathbf{r}$ and (2) no additional hash in encryption.

**Algorithm 4** SMAUG.CCAKEM.KeyGen: key generation

```
Output a public key $pk = (\rho, \mathbf{b}) \in \{0,1\}^{256} \times R_q^k$
Output a secret key $sk = (\mathbf{s}, d) \in S_\eta^k \times \{0,1\}^{256}$
1: $(pk, sk') := \mathsf{CPAPKE.Keygen}()$
2: $d \leftarrow \{0,1\}^{256}$
3: $sk = (sk', d)$
4: Return $(pk, sk)$
```

**Algorithm 5** SMAUG.CCAKEM.Encap: encapsulation

```
Input a public key $pk = (\rho, \mathbf{b}) \in \{0,1\}^{256} \times R_q^k$
Output a ciphertext $ctxt = (\mathbf{c_1}, c_2) \in R_p^k \times R_p$
Output a shared key $K \in \{0,1\}^{256}$
1: $\mu \leftarrow \{0,1\}^{256}$
2: $ctxt := \mathsf{CPAPKE.Encrypt}\big(pk, \mu; \mathrm{G}(\mu, \mathrm{H}(pk))\big)$
3: $K := \mathrm{KDF}\big(\mu, \mathrm{H}(ctxt)\big)$
4: Return $(ctxt, K)$
```

### 3.5 Parameter sets

SMAUG is parameterized by integers $n, k, q, p, t, h_s$ and $h_r$, and the standard deviation $\sigma$ for the discrete Gaussian error in the key. We use same $n = 256$, $q = 1024$, $p = 256$, $t = 2$ and $\sigma = 1.0625$ for every parameter set.

### 3.6 Implementation details

**Coefficient in MSB.** For $x \in \mathbb{Z}_q$, rather than storing itself, we store the value ($x \ll$ `_16_LOG_Q`) in `uint16_t`, i.e. $x$ is stored in $\log q$ most significant bits of `uint16_t`. In other words, we identify $\mathbb{Z}_q$ with the subspace of 16-bit data space of which the components are all zero except the most significant $\log q$ bits. If vectors or matrices (resp. polynomials) are defined over $\mathbb{Z}_q$, then the above data storage strategy is applied to each of the components (resp. coefficient).

**Algorithm 6** SMAUG.CCAKEM.Decap: decapsulation

```
Input a public key $pk = (\rho, \mathbf{b}) \in \{0,1\}^{256} \times R_q^k$
Input a secret key $sk = (\mathbf{s}, d) \in S_\eta^k \times \{0,1\}^{256}$
Input a ciphertext $ctxt = (\mathbf{c}_1, c_2) \in R_p^k \times R_p$
Output a shared key $K' \in \{0,1\}^{256}$
1: $\mu' := \mathsf{CPAPKE.Decrypt}(sk.\mathbf{s}, ctxt)$
2: $ctxt' := \mathsf{CPAPKE.Encrypt}\big(pk, \mu'; \mathrm{G}(\mu', \mathrm{H}(pk))\big)$
3: if $ctxt = ctxt'$ then
4:     Return $K' = \mathrm{KDF}\big(\mu, \mathrm{H}(ctxt)\big)$
5: else
6:     Return $K' = \mathrm{KDF}\big(sk.d, \mathrm{H}(ctxt)\big)$
7: end if
```

| Parameters sets<br>Target security | SMAUG128<br>I | SMAUG192<br>III | SMAUG256<br>V |
|---|---|---|---|
| $n$ | 256 | 256 | 256 |
| $k$ | 2 | 3 | 5 |
| $q$ | 1024 | 1024 | 1024 |
| $p$ | 256 | 256 | 256 |
| $t$ | 2 | 2 | 2 |
| $h_s$ | 140 | 150 | 145 |
| $h_r$ | 132 | 147 | 140 |
| $\sigma$ | 1.0625 | 1.0625 | 1.0625 |
| Classical core-SVP hardness | 120.0 | 180.2 | 260.3 |
| Quantum core-SVP hardness | 105.6 | 164.7 | 243.8 |
| Decryption failure probability | -177.2 | -160.4 | -168.0 |

**Table 3.** Parameter choices for security levels I, III, V.

**Form of sparse polynomial.** As mentioned above, sparse polynomial space is $S_\eta$ which means that coefficient belongs to $-1, 0, 1$. In addition, our sparse polynomial $s(x)$ and $r(x)$ have $h_s$ and $h_r$ of non-zero coefficients, respectively, since they are sampled from HWT. It is wasting memory to store polynomial itself. Hence, we store degrees of non-zero coefficient. The degrees of coefficient 1 are stacked from the beginning of the array, and those of coefficient -1 are stacked backward from the end of the array. The smallest index indicating the degree of coeffients -1 is denoted by `neg_start`. Converting to degree arrays from sparse polynomial is done by `convToIdx`.

**Packing and Unpacking.** Packing means that conversion to `uint8_t` array from polynomial in $\mathcal{R}_q$ and $\mathcal{R}_p$, or new form of sparse polynomial described above. Assume that coefficients of a polynomial in $\mathcal{R}_q$ and $\mathcal{R}_p$ are shifted to the right by `_16_LOG_Q` and `_16_LOG_P`, respectively.

Rq_to_bytes is the function to pack a polynomial in $\mathcal{R}_q$ to `uint8_t` array. We pack 4 coefficients to 5 `uint8_t` elements of array since $q$ is 1024. First, pack 8 least significant bits of each coefficients to the corresponding `uint8_t` elements, and then left 2 bits store in the last 5-th `uint8_t` element.

Rp_to_bytes is the function to pack a polynomial in $\mathcal{R}_p$ to `uint8_t` array. Unlike $\mathcal{R}_q$, we pack each coefficient to `uint8_t` element directly as $p$ is 256. However, we cannot use memcpy on polynomial itself since data type of polynomial is `uint16_t`.

Sx_to_bytes is the function to pack a degree array of spare polynomial described in 3.6.2 to `uint8_t` array. We pack each degree to `uint8_t` element directly as our degree $n$ is 256.

Unpacking is to recover a polynomial or degree array from packed `uint8_t` array.

**Polynomial multiplication.** Polynomial multiplication in SMAUG operates with polynomial in $\mathcal{R}_q$ or $\mathcal{R}_p$ and $S_\eta$. This implies that we do not need NTT and toom-cook algorithm.

**Algorithm 7** poly_mult_add(): polynomial multiplication

```
Input polynomial $a(x) \in \mathcal{R}_q$
Input degree array $b$ of sparse polynomial and its length $len(b)$
Input integer neg_start
Output polynomial $b(x) \in \mathcal{R}_q$
1: for $i$ from 0 to $len(b)$ do
2:     $degree = b[i]$
3:     $branch = 2 \cdot ((i - \texttt{neg\_start}) \gg 7 \wedge \texttt{0x01}) - 1$
4:     for $j$ from 0 to $n - degree$ do
5:         $b[degree + j] = b[degree + j] + branch \cdot f[i]$;
6:     end for
7:     for $j$ from $n - degree$ to $n$ do
8:         $b[degree + j] = b[degree + j] - branch \cdot f[i]$;
9:     end for
10: end for
11: Return $b$
```

**Sampling a uniformly random matrix A, expandA.** Sample pseudorandom bytes of length the number of elements in **A** and covert each polynomial from `uint8_t` array by Unpacking function.

**Sampling a sparse polynomial, HWT$_h$ with hamming weight $h$.** This function is used for sampling sparse polynomial $s(x)$ and $r(x)$ having $h$ non-zero coefficients. The reason for `uint64_t` is most efficient data type for sampling compared with other data types, `uint8_t`, `uint16_t`, and `uint32_t`. We use 10 bits for each degree and ternary value and drop 4 bits.

**Algorithm 8** expandA(): sampling a uniformly random matrix $\mathbf{A}$

**Input** a seed $\rho \in \{0,1\}^{256}$

**Output** a uniformly random $\mathbf{A} \in \mathcal{R}_q^{k \times k}$

```
1: $buf \leftarrow \text{XOF}(\rho)$
2: for $i$ from 0 to $k-1$ do
3:     for $j$ from 0 to $k-1$ do
4:         $\mathbf{A}[i][j] = \texttt{bytes\_to\_Rq}(buf + offset1 \cdot i + offset2 \cdot j)$
5:     end for
6: end for
7: Return A
```

**Algorithm 9** $\mathsf{HWT}_h()$: sampling a sparse polynomial

**Input** a seed $\tau$

**Output** degree array $arr$ of sparse polynomial $s$

```
1: $count = 0$
2: $hash\_idx = 0$
3: $hash = \text{XOF}(\tau)$
4: while $count < h$ do
5:     for $i$ from 0 to 5 do
6:         $offset = 10 \cdot i$
7:         $degree = (\texttt{uint8\_t})(hash[hash\_idx] \gg offset)$
8:         if $s[degree] = 0$ then
9:             $s[degree] = ((\texttt{uint8\_t})(hash[hash\_idx] \gg offset + 8) \wedge \texttt{0x02}) - 1$
10:            $count = count + 1$
11:        end if
12:    end for
13:    $hash\_idx = hash\_idx + 1$
14: end while
15: Return $arr = \texttt{convToIdx(s)}$
```

## 4 Performance analysis

In this section, we report performance results of C reference implementation.

### 4.1 Description of platform

Table 4 and 5 report performance results of the reference implementation and the sizes. All benchmarks were obtained on one core of an AMD Ryzen 3,700x processor with clock speed 3,589 MHz (as reported by `/proc/cpuinfo`) with TurboBoost and hyperthreading disabled. The benchmarking machine has 64 GB of RAM and is running Debian GNU/Linux with Linux kernel version 5.4.0. Implementation were compiled with gcc version 9.4.0 and the compiler flags as indicated in the CMakeLists included in the submission package.

### 4.2 Performance of reference implementation

All cycle counts reported are the median of the cycle counts of 10,000 executions of the respective function. The implementation are not optimized for memory usage, but generally SMAUG has only very modest memory requirements. This means that in particular our implementations do not need to allocate any memory on the heap.

| **Scheme** | **Keygen** | **Encapsulation** | **Decapsulation** |
|---|---|---|---|
| SMAUG128 | 73584 | 81684 | 88920 |
| SMAUG192 | 106956 | 115128 | 124812 |
| SMAUG256 | 191268 | 200520 | 210240 |

**Table 4.** Cycle counts for all parameter sets of SMAUG. Cycle counts were obtained on one core of an AMD Ryzen 7 3700X. For each funtions includes cycles of packing to bytes array from each data structure such as `public key`, `secret key`, and `ciphertext` and randombytes sampling.

The secret key is consisted of a vector of secret polynomial of vector length $k$ and a random bytes $t$. The publc key is consisted of a seed of $\mathbf{A}$ and a vector of polynomial $b$ in $\mathcal{R}_q^k$. The ciphertext is consisted of a vector of polynomial $c_1$ in $\mathcal{R}_p^k$ and a polynomial $c_2$ in $\mathcal{R}_p$.

| **Scheme** | **sk** | **pk** | **ct** |
|---|---|---|---|
| SMAUG128 | 174 (846) | 672 | 768 |
| SMAUG192 | 185 (1177) | 992 | 1024 |
| SMAUG256 | 182 (1814) | 1632 | 1536 |

**Table 5.** Key and ciphertext sizes in bytes. The numbers in parenthesis are the sum of secret key sk sizes and public key pk sizes.

## 5 Security

*Indistinguishability against adaptive Chosen Ciphertext Attacks* (IND-CCA2) is regarded as a strong security notion for the key encapsulation mechanisms. In the IND-CCA2 security game, the adversary can access to the public key and the decapsulation oracle, with adaptively chosen ciphertexts. That is, it can query a sequence of ciphertexts $ctxt_i$ and receives $\mathsf{KEM.Decap}(sk, ctxt_i)$, adaptively. At some point during the run-time, the adversary may gets a pair $(K, ctxt)$, where $K$ be either a session key corresponds to a ciphertext $ctxt$ or a random key (with $ctxt$ output from $\mathsf{KEM.Encap}$). In the end, the adversary outputs its guess on whether the pair is a correct pair or not. It wins if the guess is correct.

Our key encapsulation mechanism $\mathsf{SMAUG.CCAKEM}$ has IND-CCA2 security. Since our $\mathsf{KEM}$ is constructed based on the Fujisaki-Okamoto transform [23, 24] upon a public key encryption scheme $\mathsf{SMAUG.CPAPKE}$, we first see the security notion for the underlying public key encryption scheme. If *Indistinguishability against Chosen Plaintext Attacks* (IND-CPA) security of the underlying $\mathsf{PKE}$ is guaranteed, then the IND-CCA2 security of our $\mathsf{SMAUG.CCAKEM}$ is also guaranteed due to FO transform.

IND-CPA is a security notion of public key encryption schemes. In the IND-CPA game, the adversary has accesses to the public key and the encryption oracle. At some point during the run-time, the adversary queries two messages to the challenger and receives a ciphertext of one of the messages. It wins if it guesses correctly on which message is used for the encryption.

### 5.1 Security definition

#### 5.1.1 Security definitions and reductions.

**Definition 3 (Indistinguishablity under Chosen Plaintext Attacks (IND-CPA)).** *For a (randomized) public key encryption scheme* $\mathsf{PKE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$*, an IND-CPA adversary* $\mathcal{A}$*, with a sub-algorithm* $\mathcal{A}_{\mathsf{sub}}$*, has an access to the public key* $\mathsf{pk}$ *(as a result, it has accesses to the encryption oracle* $\mathsf{Enc}(\mathsf{pk}, \cdot)$*). Then the advantage of the IND-CPA adversary* $\mathcal{A}$ *is*

$$\mathsf{Adv}^{\mathsf{IND-CPA}}_{\mathsf{PKE}}(\mathcal{A}) = \left| \Pr\left[ b = b' \;\middle|\; \begin{array}{c} (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen};\; (M_0, M_1) \leftarrow \mathcal{A}_{\mathsf{sub}}(\mathsf{pk}); \\ b \leftarrow \{0,1\};\; b' \leftarrow \mathcal{A}(\mathsf{pk}, \mathsf{Enc}(\mathsf{pk}, M_b)) \end{array} \right] - \frac{1}{2} \right|.$$

**Definition 4 (Indistinguishablity under adaptive Chosen Ciphertext Attacks (IND-CCA2)).** *For a (randomized) key encapsulation mechanism* $\mathsf{KEM} = (\mathsf{KeyGen}, \mathsf{Encap}, \mathsf{Decap})$*, an IND-CCA2 adversary* $\mathcal{A}$ *has accesses to the public key* $\mathsf{pk}$ *and the decapsulation oracle* $\mathsf{Decap}(\mathsf{sk}, \cdot)$*. It can adaptively query ciphertexts to the oracle. Then the advantage of the IND-CCA2 adversary* $\mathcal{A}$ *is*

$$\mathsf{Adv}^{\mathsf{IND-CCA2}}_{\mathsf{KEM}}(\mathcal{A}) = \left| \Pr\left[ b = b' \;\middle|\; \begin{array}{l} (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen};\; (K_0, \mathsf{ctxt}) \leftarrow \mathsf{Encap}(\mathsf{pk}); \\ K_1 \leftarrow \mathcal{K};\; b \leftarrow \{0,1\};\; b' \leftarrow \mathcal{A}(\mathsf{pk}, (K_b, \mathsf{ctxt})) \end{array} \right] - \frac{1}{2} \right|.$$

We first note that the underlying public key encryption scheme $\mathsf{SMAUG.PKE}$ is IND-CPA secure against any (possibly quantum) adversary, assuming $\mathsf{MLWE}$, $\mathsf{MLWR}$ and the pseudo-randomness of $\mathsf{PRF}$. We give the corresponding theorem with a sketch of the proof.

**Theorem 2 (IND-CPA security of $\mathsf{SMAUG.PKE}$).** *Assume* $\mathsf{expandA}$ *and* $\mathsf{HWT}$ *are random oracles. In the (quantum) random oracle model, for any IND-CPA adversary $\mathcal{A}$ for* $\mathsf{SMAUG.PKE}$*, there exist three adversaries $\mathcal{B}_0$, $\mathcal{B}_1$ and $\mathcal{B}_2$ of the* $\mathsf{PRF}$*,* $\mathsf{MLWE}$ *and* $\mathsf{MLWR}$ *with roughly the same running time as $\mathcal{A}$, such that*

$$\mathsf{Adv}^{\mathsf{IND-CPA}}_{\mathsf{SMAUG.PKE}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{PRF}}_{\mathsf{expandA}}(\mathcal{B}_0) + \mathsf{Adv}^{\mathsf{MLWE}}_{n,q,k,k,\eta,h_s}(\mathcal{B}_1) + \mathsf{Adv}^{\mathsf{MLWR}}_{n,p,q,k+1,k,\eta,h_r}(\mathcal{B}_2).$$

*Proof (sketch).* Assuming the pseudo-randomness of the $\mathsf{PRF}$and the hardness of the $\mathsf{MLWE}$ problem, the reconstructed public-key $(\mathbf{A}^\top \mid \mathbf{b}^\top)^\top \in \mathcal{R}_q^{(k+1)\times k}$ is pseudo-random. Hence the ciphertext $\mathbf{c} = (\mathbf{c}_1^\top, c_2) \in \mathcal{R}_p^{k+1}$ can be rewritten as

$$\mathbf{c} = \begin{bmatrix} \mathbf{c}_1 \\ c_2 \end{bmatrix} = \left\lfloor \frac{p}{q} \cdot \begin{pmatrix} \mathbf{A} \\ \mathbf{b}^\top \end{pmatrix} \cdot \mathbf{r} \right\rceil + \frac{p}{t} \cdot \begin{bmatrix} 0 \\ \mu \end{bmatrix},$$

since $t \mid p \mid q$. Assuming the hardness of $\mathsf{MLWR}$ problem, $\mathbf{c}$ is indistinguishable from the random sample from $\mathcal{R}_p^{k+1}$ added by the vector $\frac{p}{t} \cdot \begin{bmatrix} 0 \\ \mu \end{bmatrix}$, which is also a random sample in $\mathcal{R}_p^{k+1}$. □

**Tight reduction in ROM.** We first give the tight reduction from IND-CPA security to IND-CCA2 security in the random oracle model (ROM), which is basically FO transform [23, 24] with some decryption failure probabilities [27]. Since the reduction is tight, we can directly analyze the security of our key encapsulation scheme $\mathsf{SMAUG.KEM}$from the following theorem.

**Theorem 3 (IND-CPA-$\mathsf{PKE}$ $\Rightarrow$ IND-CCA2-$\mathsf{KEM}$, [27]).** *Assume $G$ and $H$ are random oracles. For any classical IND-CCA2 adversary $\mathcal{A}$ for* $\mathsf{SMAUG.KEM}$*, there exists an IND-CPA adversary $\mathcal{B}$ for* $\mathsf{SMAUG.PKE}$ *with roughly the same running time as $\mathcal{A}$, such that*

$$\mathsf{Adv}^{\mathsf{IND-CCA2}}_{\mathsf{SMAUG.KEM}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{IND-CPA}}_{\mathsf{SMAUG.PKE}}(\mathcal{B}) + 4q_{\mathrm{RO}}\delta,$$

*where $q_{\mathrm{RO}}$ is the upper bound of the number of the random oracle queries that the adversary can make and $\delta$ is a decryption-failure probability.*

**Non-tight reduction in QROM.** For the security reductions in the quantum random oracle model (QROM), the adversary now has accesses to quantum random oracles. We shall follow the proofs in Kyber [14] and Saber [21], based on the security reduction of the FO transforms in the QROM [27, 33].

**Theorem 4 ((Non-tight) IND-CPA-$\mathsf{PKE} \Rightarrow$ IND-CCA2-$\mathsf{KEM}$ in QROM).** *Assume* $\mathsf{expandA}$*,* $G$ *and* $H$ *are random oracles. For any quantum adversary* $\mathcal{A}$ *who can access to the quantum random oracles* $\mathsf{expandA}$*,* $G$ *and* $H$*, there exists an IND-CPA adversary* $\mathcal{B}$ *for* $\mathsf{SMAUG.PKE}$*, such that*

$$\mathsf{Adv}^{\mathsf{IND-CCA2}}_{\mathsf{SMAUG.KEM}} \leq 2q_{\mathrm{RO}} \cdot \sqrt{\mathsf{Adv}^{\mathsf{IND-CPA}}_{\mathsf{SMAUG.PKE}} + 1/|\mathcal{M}|} + 4q_{\mathrm{RO}}\sqrt{\delta},$$

*where* $q_{\mathrm{RO}}$ *is the upper bound of the number of the random oracle queries that the adversary can make and* $\delta$ *is a decryption-failure probability.*

**Tight reduction in QROM under non-standard security notion.** Since the Theorem 4 is a non-tight reduction, it only gives the asymptotic security of $\mathsf{SMAUG.KEM}$. We therefore give a tight reduction for the security analysis assuming a non-standard assumption, that the deterministic version of the $\mathsf{PKE}$ is pseudo-random in the QROM, following [27, 33]. We remark that we are not aware of any quantum attack on deterministic version of $\mathsf{SMAUG.PKE}$ (and even for other PQC $\mathsf{PKE}$ schemes) that performs better than breaking the $\mathsf{MLWE}$ problem. With this assumption we give the following theorem.

**Theorem 5 ((Tight) IND-CPA-$\mathsf{PKE} \Rightarrow$ IND-CCA2-$\mathsf{KEM}$ in QROM, [33]).** *Assume* $\mathsf{expandA}$*,* $G$ *and* $H$ *are random oracles and assume that the underlying deterministic* $\mathsf{SMAUG.DPKE}$ *is pseudo-random in QROM. For any quantum adversary* $\mathcal{A}$ *who can access to the quantum random oracles* $\mathsf{expandA}$*,* $G$ *and* $H$*, there exist adversaries* $\mathcal{B}_0$*,* $\mathcal{B}_1$*,* $\mathcal{B}_2$ *and* $\mathcal{B}_3$ *for* $\mathsf{PRF}$*,* $\mathsf{MLWE}$*,* $\mathsf{MLWR}$ *and the pseudo-randomness of* $\mathsf{SMAUG.DPKE}$*, such that*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{IND-CPA}}_{\mathsf{SMAUG.PKE}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{PRF}}(\mathcal{B}_0) + \mathsf{Adv}^{\mathsf{MLWE}}_{n,q,k,\ell,\eta}(\mathcal{B}_1) + \mathsf{Adv}^{\mathsf{MLWR}}_{n,p,q,k,\ell,\eta}(\mathcal{B}_2)&\\ + \mathsf{Adv}^{\mathsf{PR}}_{\mathsf{SMAUG.DPKE}}(\mathcal{B}_3) + 8q^2_{\mathrm{RO}}\delta,&\end{aligned}$$

*where* $q_{\mathrm{RO}}$ *is the upper bound of the number of the random oracle queries that the adversary can make and* $\delta$ *is a decryption-failure probability.*

## 5.2 Security strength categories

We target the security of our $\mathsf{SMAUG.KEM}$ to the NIST PQC security levels 1, 3 and 5, which is at least as secure as Kyber and Saber. Targeting such security levels we use the *Core-SVP* methodology, a conservative security estimation method in lattice-based cryptography (see section 5.3), and give the following parameter sets correspond to the security levels.

## 5.3 Cost of known attacks

For the concrete security analysis, we list the best known lattice attacks and the required cost upon attacking our key encapsulation mechanism $\mathsf{SMAUG.KEM}$. All the best known attacks are essentially finding a nonzero short vector in the

| Parameters sets<br>Target security | SMAUG128<br>I | SMAUG192<br>III | SMAUG256<br>V |
|---|---|---|---|
| Classical core-SVP hardness | 120.0 | 180.2 | 260.3 |
| Quantum core-SVP hardness | 105.6 | 164.7 | 243.8 |

**Table 6.** Parameter choices for security levels I, III, V.

Euclidean lattices, using the Block–Korkine–Zolotarev (BKZ) lattice reduction algorithm [16, 26, 34].

The BKZ algorithm is a lattice basis reduction algorithm that uses the Shortest Vector Problem (SVP) solver repeatedly to small-dimensional sub-lattices, which we call a block of size $\beta$, rather than in the entire high-dimensional lattice. $\beta$-BKZ is the BKZ algorithm using SVP solver in the block size $\beta$, and the parameter $b$ determines the quality of the resulting basis and the time complexity. Indeed there is a quality/time trade-off: If $\beta$ gets larger, the better quality will be guaranteed but also the time complexity for the SVP solver will be exponentially increased. The time complexity of the $\beta$-BKZ algorithm is same with the SVP solver for dimension $\beta$ with a polynomial factor. Hence the time complexity differs depending on the SVP solver used. The most efficient SVP algorithm is using the sieving method proposed by Becker et al. [10] which takes time $\approx 2^{0.292\beta+o(\beta)}$ with classical solver. The fastest known quantum variant was recently proposed by Chailloux and Loyer in [15], and takes time $\approx 2^{0.257b+o(b)}$.

Based on the BKZ algorithm, we will follow the *Core-SVP* methodology as in [4] and in the subsequent lattice-based post-quantum schemes [3, 14, 20–22], which is regarded as a conservative way to set the security parameters. We ignore the polynomial factors and the $o(\beta)$ terms in the exponent for the time complexity of the BKZ algorithm.

We give the best known attacks for MLWE, namely *primal attack*, *dual attack* and their hybrid variants with the Core-SVP hardness of the attacks. We remark that any $\mathsf{MLWE}_{n,q,k,\ell,\eta}$ instance can be viewed as an $\mathsf{LWE}_{q,nk,n\ell,\eta}$ instance. Even though MLWE problem has some extra algebraic structure compared to the LWE problem, we do not currently have any attack advantaged by this structure. Hence we analyze the hardness of the MLWE problem over the structured lattices as the hardness of the corresponding LWE problem over the unstructured lattices.

When dealing with the hardness of the MLWR problem, however, we treat it as an MLWE problem, since there are no known attacks that uses the of the deterministic error term in MLWR structure. Further more, the reduction from the (M)LWE problem to the (M)LWR problem were also given by Banerjee et al. [9] and the improvements [6, 7, 12]. Basically, an MLWR sample given by $(\mathbf{A}, \lfloor p/q \cdot \mathbf{A} \cdot \mathbf{s} \rceil \mod p)$ for uniformly chosen $\mathbf{A} \leftarrow \mathcal{R}_q^k$ and $\mathbf{s} \leftarrow \mathcal{R}_p^\ell$ can be rewritten as $(\mathbf{A}, p/q \cdot (\mathbf{A} \cdot \mathbf{s} \mod q) + \mathbf{e} \mod p)$. This sample can be transformed to an MLWE sample over $\mathcal{R}_q$ by multiplying $q/p$ as $(\mathbf{A}, \mathbf{b} = \mathbf{A} \cdot \mathbf{s} + q/p \cdot \mathbf{e} \mod q)$. We assume that the error term in the resulting MLWE sample is a random variable, uniform in the interval $(-q/2p, q/2p]$, so that we can estimate the hardness of the MLWR problem as the hardness of the corresponding MLWE problem.

We summarize the cost of the known attacks in Table 7. The security is estimated via lattice estimator [2] and is represented as core-SVP hardness.

| Parameters sets<br>Target security | SMAUG128<br>I | SMAUG192<br>III | SMAUG256<br>V |
|---|---|---|---|
| Classical core-SVP hardness for MLWE | | | |
| Primal attack | 120.0 | 187.2 | 317.1 |
| Primal attack (BDD) | 120.9 | 188.5 | 319.5 |
| Primal attack (hybrid) | 121.3 | 189.0 | 292.2 |
| Dual attack | 125.9 | 195.3 | 329.1 |
| Dual attack (hybrid) | 122.7 | 180.2 | 262.0 |
| Classical core-SVP hardness for MLWR | | | |
| Primal attack | **120.0** | 187.8 | 317.7 |
| Primal attack (BDD) | 121.6 | 189.2 | 320.7 |
| Primal attack (hybrid) | 121.6 | 189.8 | 290.0 |
| Dual attack | 126.1 | 195.9 | 330.0 |
| Dual attack (hybrid) | 122.4 | **180.2** | **260.3** |
| Quantum core-SVP hardness for MLWE | | | |
| Primal attack | 105.6 | **164.7** | 279.1 |
| Primal attack (BDD) | 106.5 | 166 | 281.3 |
| Primal attack (hybrid) | 106.9 | 166.4 | 269.2 |
| Dual attack | 110.8 | 171.9 | 289.6 |
| Dual attack (hybrid) | 111.5 | 165.6 | 245.1 |
| Quantum core-SVP hardness for MLWR | | | |
| Primal attack | **105.6** | 165.3 | 279.6 |
| Primal attack (BDD) | 107.1 | 166.6 | 282.3 |
| Primal attack (hybrid) | 107.1 | 167.2 | 267.1 |
| Dual attack | 111 | 172.4 | 290.4 |
| Dual attack (hybrid) | 111.5 | 165.7 | **243.8** |

**Table 7.** Cost of known attacks. The security is represented as core-SVP hardness.

#### 5.3.1 Description of Primal attack.

Given an LWE instance $(\mathbf{A}, \mathbf{b}) \in \mathbb{Z}_q^{k\times\ell} \times \mathbb{Z}_q^k$, we first define a lattice $\Lambda_m = \{\mathbf{v} \in \mathbb{Z}^{\ell+m+1} : \mathbf{B}\mathbf{v} = 0 \mod q\}$, where $\mathbf{B} = \left(\mathbf{A}_{[m]} \mid \mathbf{Id}_m \mid \mathbf{b}_{[m]}\right) \in \mathbb{Z}_q^{m\times(\ell+m+1)}$, where $\mathbf{A}_{[m]}$ is the uppermost $m \times \ell$ sub-matrix of $\mathbf{A}$ and $\mathbf{b}_{[m]}$ is the uppermost length $m$ sub-vector of $\mathbf{b}$ for $m \le k$. Then, short non-zero vector in the lattice $\Lambda_m$ can be transformed to a short non-trivial solution to the LWE equation. Primal attack solves the SVP problem in the lattice $\Lambda_m$ using $\beta$-BKZ, increasing the block size $\beta$, for all possible $m$.

#### 5.3.2 Description of Dual attack.

Given an LWE instance $(\mathbf{A}, \mathbf{b}) \in \mathbb{Z}_q^{k\times\ell} \times \mathbb{Z}_q^k$, we first define a lattice $\Lambda'_m = \{(\mathbf{u}, \mathbf{v}) \in \mathbb{Z}^m \times \mathbb{Z}^\ell : \mathbf{A}_{[m]}^\top\mathbf{u} + \mathbf{v} = 0 \mod q\}$, where $\mathbf{A}_{[m]}$ is the uppermost $m \times \ell$ sub-matrix of $\mathbf{A}$ for $m \le k$. Then, again, short non-zero vector in the lattice $\Lambda'_m$ induces a short non-trivial solution to the $\mathsf{LWE}$ problem. Dual attack solves the $\mathsf{SVP}$ problem in the lattice $\Lambda'_m$ using $\beta$-BKZ, increasing the block size $\beta$, for all possible $m$.

**5.3.3 Description of the hybrid variants.** For both Primal and Dual attacks, there are some variants combining the attack with the combinatorial attacks or the meet-in-the-middle (MITM) attack, which we call hybrid attacks. These variants are usually slower than the original attacks, however, the attacks may benefit by the special choice of the secret used in the $\mathsf{LWE}$ problem - small or sparse. By exploiting the secret as a preprocessing, or using the MITM approach with guessing the part of the sparse secret, the attacks may improved compared to the original attacks.

Since hybrid attacks are combinations of lattice reduction attacks and combinatorial/MITM attacks, it is not natural to apply the Core-SVP method directly to the hybrid attacks, focusing only on the BKZ block-size, since it may ignore the part and parcel of the attack. We, instead, naïvely extend the Core-SVP methodology to the case of the hybrid attacks by using the Core-SVP methodology on the lattice reduction parts, and then divide by the probability of success of the combinatorial/meet-in-the-middle attack parts. We will estimate the cost by joining the information-theory and Core-SVP methodology. That is, we find the best block-size $\beta$ and calculate $c \cdot \beta - \log_2((\mathsf{Pr}[\text{guess is correct}]))$ where $c$ is either $c_C = 0.292$ or $c_Q = 0.257$. Since the success probability of the guessing is independent of the BKZ algorithm, this can be viewed as a naïve extension of Core-SVP method to the hybrid attacks.

**5.3.4 Beyond Core-SVP methodology.** We also analyze the cost of the attacks other than the Primal and Dual attacks variants. Algebraic attacks like Arora-Ge attack and the variants [8, 1] using Gröbener's basis or Coded-BKW attacks [31, 25] are also considered, using the lattice estimator [2], but they have much larger attack costs compared to the previously introduced attacks.

## 6 Summary or Conclusion

Our key encapsulation scheme takes the approaches of FO transform in the QROM to a public key encryption scheme which is IND-CPA secure based on the MLWE and MLWR assumptions. Our goal was to make the underlying PKE more smaller and simpler, so that the secure implementation could be done easily. Keeping this in our mind, we found that it is much simpler to use MLWR assumption for the encryption, not just for the key generation like in Saber.

The resulting PKE and KEM are now have smaller sizes compared to NIST's standard PQC KEM Kyber, and more efficient encryption and encapsulation algorithm then NIST's round 3 finalist Saber. The implemented SMAUG.PKE and SMAUG.KEM are also following exactly the quantumly secure FO transforms.

*Future works and directions.* First, we will keep studying on the hardness of sparse LWE problem and sparse LWR problem. These variants are already studied for a while, but more effort is needed since the hybrid attacks are especially strong to the sparse secret. Secondly, we will implement more faster multiplication befit on the sparse polynomials, for the later release of the SMAUG. Third, we will give optimized implementation of SMAUG in various devices. Fourth, we will add more formal security analysis including side-channel attacks, and give secure implementation against it.

## References

1. Albrecht, M.R., Cid, C., Faugère, J.C., Perret, L.: Algebraic algorithms for lwe. Cryptology ePrint Archive, Paper 2014/1018 (2014), https://eprint.iacr.org/2014/1018, https://eprint.iacr.org/2014/1018
2. Albrecht, M.R., Player, R., Scott, S.: On the concrete hardness of learning with errors. Journal of Mathematical Cryptology **9**(3), 169–203 (2015)
3. Alkim, E., Barreto, P.S.L.M., Bindel, N., Kramer, J., Longa, P., Ricardini, J.E.: The lattice-based digital signature scheme qtesla. Cryptology ePrint Archive, Paper 2019/085 (2019), https://eprint.iacr.org/2019/085, https://eprint.iacr.org/2019/085
4. Alkim, E., Ducas, L., Pöppelmann, T., Schwabe, P.: Post-quantum key Exchange—A new hope. In: 25th USENIX Security Symposium (USENIX Security 16). pp. 327–343. USENIX Association, Austin, TX (Aug 2016), https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/alkim
5. Alkim, E., Ducas, L., Pöppelmann, T., Schwabe, P.: Post-quantum key {Exchange—A} new hope. In: 25th USENIX Security Symposium (USENIX Security 16). pp. 327–343 (2016)
6. Alperin-Sheriff, J., Apon, D.: Dimension-preserving reductions from lwe to lwr. Cryptology ePrint Archive, Paper 2016/589 (2016), https://eprint.iacr.org/2016/589, https://eprint.iacr.org/2016/589
7. Alwen, J., Krenn, S., Pietrzak, K., Wichs, D.: Learning with rounding, revisited. In: Canetti, R., Garay, J.A. (eds.) Advances in Cryptology – CRYPTO 2013. pp. 57–74. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
8. Arora, S., Ge, R.: New algorithms for learning in presence of errors. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) Automata, Languages and Programming. pp. 403–415. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
9. Banerjee, A., Peikert, C., Rosen, A.: Pseudorandom functions and lattices. In: Pointcheval, D., Johansson, T. (eds.) Advances in Cryptology – EUROCRYPT 2012. pp. 719–737. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
10. Becker, A., Ducas, L., Gama, N., Laarhoven, T.: New directions in nearest neighbor searching with applications to lattice sieving, pp. 10–24. Society for Industrial and Applied Mathematics (2016). https://doi.org/10.1137/1.9781611974331.ch2, https://epubs.siam.org/doi/abs/10.1137/1.9781611974331.ch2
11. Birkett, J., Dent, A.W.: Relations among notions of plaintext awareness. In: Cramer, R. (ed.) Public Key Cryptography – PKC 2008. pp. 47–64. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)
12. Bogdanov, A., Guo, S., Masny, D., Richelson, S., Rosen, A.: On the hardness of learning with rounding over small modulus. In: Kushilevitz, E., Malkin, T. (eds.) Theory of Cryptography. pp. 209–224. Springer Berlin Heidelberg, Berlin, Heidelberg (2016)
13. Bos, J., 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: Proceedings of the 2016 ACM SIGSAC conference on computer and communications security. pp. 1006–1018 (2016)
14. Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Schwabe, P., Seiler, G., Stehlé, D.: Crystals-kyber: a cca-secure module-lattice-based kem. In: 2018 IEEE European Symposium on Security and Privacy (EuroS&P). pp. 353–367. IEEE (2018)

15. Chailloux, A., Loyer, J.: Lattice sieving via quantum random walks. In: Tibouchi, M., Wang, H. (eds.) Advances in Cryptology - ASIACRYPT. pp. 63–91 (2021)
16. Chen, Y., Nguyen, P.Q.: Bkz 2.0: Better lattice security estimates. In: Lee, D.H., Wang, X. (eds.) Advances in Cryptology – ASIACRYPT 2011. pp. 1–20. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
17. Cheon, J.H., Han, K., Kim, J., Lee, C., Son, Y.: A practical post-quantum public-key cryptosystem based on spLWE. In: International Conference on Information Security and Cryptology. pp. 51–74. Springer (2016)
18. Cheon, J.H., Kim, D., Lee, J., Song, Y.: Lizard: Cut off the tail! a practical post-quantum public-key encryption from lwe and lwr. In: International Conference on Security and Cryptography for Networks. pp. 160–177. Springer (2018)
19. Dent, A.W.: A designer's guide to kems. In: IMA International Conference on Cryptography and Coding. pp. 133–151. Springer (2003)
20. Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schwabe, P., Seiler, G., Stehlé, D.: Crystals-dilithium: A lattice-based digital signature scheme. IACR Transactions on Cryptographic Hardware and Embedded Systems **2018**(1), 238–268 (Feb 2018). https://doi.org/10.13154/tches.v2018.i1.238-268, https://tches.iacr.org/index.php/TCHES/article/view/839
21. D'Anvers, J.P., Karmakar, A., Sinha Roy, S., Vercauteren, F.: Saber: Module-lwr based key exchange, cpa-secure encryption and cca-secure kem. In: International Conference on Cryptology in Africa. pp. 282–305. Springer (2018)
22. Fouque, P.A., Hoffstein, J., Kirchner, P., Lyubashevsky, V., Pornin, T., Prest, T., Ricosset, T., Seiler, G., Whyte, W., Zhang, Z.: Falcon: Fast-fourier lattice-based compact signatures over ntru. Submission to the NIST's post-quantum cryptography standardization process **36**(5) (2018)
23. Fujisaki, E., Okamoto, T.: Secure integration of asymmetric and symmetric encryption schemes. In: Wiener, M. (ed.) Advances in Cryptology — CRYPTO' 99. pp. 537–554. Springer Berlin Heidelberg, Berlin, Heidelberg (1999)
24. Fujisaki, E., Okamoto, T.: Secure integration of asymmetric and symmetric encryption schemes. Journal of cryptology **26**(1), 80–101 (2013)
25. Guo, Q., Johansson, T., Stankovski, P.: Coded-bkw: Solving lwe using lattice codes. In: Annual Cryptology Conference. pp. 23–42. Springer (2015)
26. Hanrot, G., Pujol, X., Stehlé, D.: Algorithms for the shortest and closest lattice vector problems. In: Chee, Y.M., Guo, Z., Ling, S., Shao, F., Tang, Y., Wang, H., Xing, C. (eds.) Coding and Cryptology. pp. 159–190. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
27. Hofheinz, D., Hövelmanns, K., Kiltz, E.: A modular analysis of the fujisaki-okamoto transformation. In: Kalai, Y., Reyzin, L. (eds.) Theory of Cryptography. pp. 341–371. Springer International Publishing, Cham (2017)
28. Hong, S.: Lizarderror. https://github.com/swanhong/LizardError (2018)
29. Jiang, H., Zhang, Z., Chen, L., Wang, H., Ma, Z.: Ind-cca-secure key encapsulation mechanism in the quantum random oracle model, revisited. In: Shacham, H., Boldyreva, A. (eds.) Advances in Cryptology – CRYPTO 2018. pp. 96–125. Springer International Publishing, Cham (2018)
30. Karmakar, A., Roy, S.S., Reparaz, O., Vercauteren, F., Verbauwhede, I.: Constant-time discrete gaussian sampling. IEEE Transactions on Computers **67**(11), 1561–1571 (2018)
31. Kirchner, P., Fouque, P.A.: An improved bkw algorithm for lwe with applications to cryptography and lattices. In: Annual Cryptology Conference. pp. 43–62. Springer (2015)

32. Lee, J., Kim, D., Lee, H., Lee, Y., Cheon, J.H.: Rlizard: Post-quantum key encapsulation mechanism for iot devices. IEEE Access **7**, 2080–2091 (2018)
33. Saito, T., Xagawa, K., Yamakawa, T.: Tightly-secure key-encapsulation mechanism in the quantum random oracle model. In: Nielsen, J.B., Rijmen, V. (eds.) Advances in Cryptology – EUROCRYPT 2018. pp. 520–551. Springer International Publishing, Cham (2018)
34. Schnorr, C.P., Euchner, M.: Lattice basis reduction: Improved practical algorithms and solving subset sum problems. Mathematical programming **66**(1), 181–199 (1994)

## A Correctness

We here give the proof of Theorem 1.

**Theorem 6.** *Let* $\mathbf{A}$*,* $\mathbf{b}$*,* $\mathbf{s}$*,* $\mathbf{e}$ *and* $\mathbf{r}$ *are defined as in* SMAUG.CPAPKE*. Let* $\mathbf{e}_1 \in \mathcal{R}_{\mathbb{Q}}^k$ *and* $e_2 \in \mathcal{R}_{\mathbb{Q}}$ *be the rounding errors introduced by scaling and rounding* $\mathbf{A} \cdot \mathbf{r}$ *and* $\mathbf{b}^T \cdot \mathbf{r}$*, i.e.* $\mathbf{e}_1 = \frac{q}{p} \cdot \lfloor \frac{p}{q} \cdot \mathbf{A} \cdot \mathbf{r} \mod p \rceil - (\mathbf{A} \cdot \mathbf{r} \mod q)$ *and* $e_2 = \frac{q}{p} \cdot \lfloor \frac{p}{q} \cdot \mathbf{b}^T\mathbf{r} \mod p \rceil - (\mathbf{b}^T\mathbf{r} \mod q)$*. If we set*

$$\delta = \Pr\left[\|\mathbf{r}^T\mathbf{e} + \mathbf{s}^T\mathbf{e}_1 + e_2\|_\infty > q/2t\right],$$

*then the decryption failure probability of* SMAUG.CPAPKE *scheme is less than* $\delta$*.*

*Proof.* Recall that $t \mid p \mid q$ and that the rounding errors satisfy

$$\mathbf{c}_1 = \frac{p}{q} \cdot (\mathbf{A} \cdot \mathbf{r} + \mathbf{e}_1) \mod p \quad \text{and} \quad c_2 = \frac{p}{q} \cdot \left(\mathbf{b}^T\mathbf{r} + e_2\right) + \frac{p}{t} \cdot \mu \mod p.$$

By the definition of $\mathbf{e}_1$ and $e_2$, each coefficient of $\mathbf{e}_1$ and $e_2$ is in $\mathbb{Z} \cap (-\frac{q}{2p}, \frac{q}{2p}]$. Thus, decryption of a ciphertext with respect to a message $\mu$ and a randomness $\mathbf{r}$ can be written as

$$\begin{aligned}\left\lfloor \frac{t}{p} \cdot (c_2 + \mathbf{c}_1^T \cdot \mathbf{s} \mod p)\right\rceil &= \left\lfloor \frac{t}{q} \cdot \mathbf{r}^T \cdot (\mathbf{A}^T\mathbf{s} + \mathbf{b}) + \mu + \frac{t}{q} \cdot (\mathbf{s}^T\mathbf{e}_1 + e_2)\right\rceil \mod t \\ &= \mu + \left\lfloor \frac{t}{q} \cdot \left(\mathbf{r}^T\mathbf{e} + \mathbf{s}^T\mathbf{e}_1 + e_2\right)\right\rceil \mod t\end{aligned}$$

and is equal to $\mu$ if and only if every coefficient of $\mathbf{r}^T\mathbf{e} + \mathbf{s}^T\mathbf{e}_1 + e_2$ is in the interval $[-\frac{q}{2t}, \frac{q}{2t})$. This concludes the proof of the theorem. □