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, 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 that we submit to the 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 achieves ciphertext sizes up to 12% and 9% smaller than Kyber (NIST's 2022 selection) and Saber (NIST's finalist), with much faster running time, up to 103% and 58%, respectively.

Keywords: Lattice-based Cryptography · Post-Quantum Cryptography · Key Encapsulation Mechanism · Module Learning With Errors · Module Learning With Roundings.

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

Changelog

May 23, 2023 First, we update the python script for DFP computation as it was computing the decryption failure probability (DFP) wrongly. Note that the script was missing in the submission file, but included in our website. The parameter sets for NIST's security levels 3 and 5 were having higher DFPs thenthey were reported in the 1st round submission. As a result, the parameter sets are updated.

Second, we additionally compress the ciphertexts. As compression makes the error larger, we exploit the balance between the sizes and DFP.

Third, we put additional cost estimations on some algebraic and topological attacks: Arora-Ge [8], Coded-BKW [27], and Meet-LWE [41] attacks. We note that the previous parameter sets were all in a secure region against these attacks; however, for the new parameter sets, we aim to have more security margins. We put our code for estimating the cost of Meet-LWE attack in the python script.

Based on the above three updates, we changed our recommanded parameter sets. As $q = 1024$ is not available anymore for sufficient DFPs in the security levels 3 and 5, we move to $q = 2048$ for those levels, resulting in slightly larger public key and secret key sizes. The ciphertext sizes are decreased by at most 96 bytes.

We also update the reference implementation to have a constant running time with much faster speed. It is uploaded to our website: kpqc.cryptolab.co.kr/smaug.

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 [19] and RLizard [39]. SMAUG.KEM bases its security on the module variant lattice problems: the public key does not leak the secret key information by the hardness of MLWE problem, and the ciphertext protects sharing keys based on the hardness of MLWR problem. SMAUG consists of an underlying public key encryption (PKE) scheme SMAUG.PKE, which turns into SMAUG.KEM via Fujisaki-Okamoto transform.

1.1 Design rationale

The design rationale of SMAUG aims is to achieve small ciphertext and public key with low computational cost while maintaining security against various attacks. In more detail, we target the following practicality and security requirements considering real applications:

Practicality:

– Both the public key and ciphertext, especially the latter, which is transmitted more frequently, need to be short in order to minimize communication costs.
– As the key exchange protocol is frequently required on various personal devices, a KEM algorithm with low computational costs is more feasible than a high-cost one.
– A small secret key is desirable in restricted environments such as embedded or IoT devices since managing the secure zone is crucial to prevent physical attacks on secret key storage.

Security:

– The shared key should have a large enough entropy, at least ≥ 256 bits, to prevent Grover's search [26].
– Security should be concretely guaranteed concerning the attacks on the underlying assumptions, say lattice attacks.
– The low enough decryption failure probability (DFP) is essential to avoid the attacks boosting the failure and exploiting the decryption failures [33, 20].
– As KEMs are widely used in various devices and systems, countermeasures against implementation-specific attacks should also be considered. Especially combined with DFP, using error correction code (ECC) on the message to reduce decryption failures should be avoided since masking ECC against side-channel attacks is a very challenging problem.

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 [19] and Ring-Lizard (RLizard) [39]. Since LWE problem has been a well-studied problem for the last two decades, there are many LWE-based schemes (e.g., Frodo [15]). 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 [22]) 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 [13], we chose to use MLWR samples for the encryption. By basing the security of encryption to MLWR, we reduce the ciphertext size by $\log q / \log p$ than MLWE instances so that more efficient encryption and decryption are 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 [24, 25]. Line of works on FO transforms in the quantum random oracle model [12, 29, 35, 42] make it possible to analyze the quantum security, i.e., in the quantum random oracle model (QROM). In particular, we use the FO transform with implicit rejection and no ciphertext contributions ($\mathsf{FO}_m^{\not\perp}$) following [32].

Sparse secret key and ephemeral key. We design the key generation algorithm based on MLWE problem using sparse secret. Based on the hardness reduction on the LWEproblem using sparse secret [18], we use sparse ternary polynomials for the secret key and the ephemeral polynomial vectors. We take advantage of the sparsity, e.g., significantly smaller secret keys and faster multiplications. In particular, the small secret makes SMAUG more feasible in IoT devices having restricted resources.

Choice of moduli. All our moduli are powers of 2. This choice makes SMAUG enjoy faster encapsulation using simple bit shiftings, easy uniform sampling, and scailings. The power of 2 moduli makes it hard to apply Number Theoretic Transform (NTT) on the polynomial multiplications. However, small enough moduli and polynomial degrees enable SMAUG to achieve faster speed.

Negligible decapsulation failures. Since we base the security on the lattice problems, noise is inherent. The decryption result of a SMAUG.PKE ciphertext could be different from the original message but with negligible probability, say decryption failure probability (DFP). As other LWE/LWR-based KEMs, SMAUG has the message-independent DFP. We balance the sizes, DFP, and security of SMAUG by fine-tuning the parameters. Trade-offs between them could give additional parameter sets for specific purposes.

We give estimated security and sizes for our parameter sets in Table 1. The complete parameter sets are given in Section 3.3. The security is estimated via lattice estimator [3] and our code. It is shown in core-SVP hardness. The DFP is calculated via a python script modified from Lizard and RLizard [31] and is reported in logarithm base two. The sizes are given in bytes. We include the security estimator of SMAUG in the reference code package on the team SMAUG website: kpqc.cryptolab.co.kr/smaug.

1.2 Advantages and limitations

Advantages

- Our scheme relies on the hardness of the lattice problems MLWEand MLWR, which enable SMAUG to balance its security and efficiency.
- In terms of sizes, SMAUG has smaller ciphertext sizes compared to Kyber or Saber (tie in level 5).
- In terms of DFP, SMAUG achieves low enough DFP, which are similar to that of Saber. We do not use error correction code (ECC) to avoid side-channel attacks targetting decryption failures.

Parameters sets Target security	SMAUG 128 I	SMAUG 192 III	SMAUG 256 V
n	256	256	256
k	2	3	5
(q, p, p')	$(1024, 256, 32)$	$(2048, 256, 256)$	$(2048, 256, 64)$
Classical core-SVP hardness	120.0	181.7	264.5
Quantum core-SVP hardness	105.6	160.9	245.2
Decryption failure probability	-120	-136	-167
Secret key size	176	236	218
Public key size	672	1088	1792
Ciphertext size	672	1024	1472

Table 1. Security and sizes for our parameter sets.

- Implementation-wise, encapsulation and decapsulation of SMAUG can be done efficiently. This makes SMAUG much easier to implement and secure against physical attacks.
- We give the constant-time C reference code, which proves the completeness and shows the efficiency of SMAUG.

Limitations

- We use MLWR problem, which has been studied shorter than MLWE or LWE problems; however, with a security reduction to MLWE, it reduces the sizes and increases the efficiency. MLWE problem with a sparse secret has a similar issue but has been studied much more and is used in various applications, e.g., homomorphic encryptions.
- As we use MLWE problem for the secret key security, larger public key sizes than Saber are inherent. It can be seen as a trade-off between the public key size versus performance with a smaller secret key size.

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 integers 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 η, we denote the set of polynomials of degree less than n with coefficients in $[-\eta, \eta] \cap \mathbb{Z}$ as S_η. Let $\widetilde{S}_\eta$ 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 MLWE and MLWR on the structured Euclidean lattices.

Definition 1 (Decision-$\mathsf{MLWE}_{n,q,k,\ell,\eta}$). *For positive integers q, k, ℓ, η 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, ℓ, η 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 consists of the following two schemes:

- $\mathsf{SMAUG.PKE}$ is an IND-CPA public-key encryption (PKE) scheme, encrypting messages of a fixed length 32 bytes,
- $\mathsf{SMAUG.KEM}$ is an IND-CCA2-secure key encapsulation mechanism (KEM) scheme, constructed using Fujisaki-Okamoto (FO) transform on $\mathsf{SMAUG.PKE}$.

We first introduce $\mathsf{SMAUG.PKE}$ in Section 3.1, then construct $\mathsf{SMAUG.KEM}$ in Section 3.2. The implementation detqils are given in Section 3.4.

3.1 Specification of $\mathsf{SMAUG.PKE}$

We now describe the public key encryption scheme $\mathsf{SMAUG.PKE}$ in Algorithms 1, 2, and 3 with the following building blocks:

- Hash function H for generating the seeds $\mathsf{seed}_{\mathbf{A}}$ and $\mathsf{seed}_{\mathsf{sk}}$,
- Uniform random matrix sampler $\mathsf{expandA}$ for deriving $\mathbf{A}$ from $\mathsf{seed}_{\mathbf{A}}$,
- Discrete Gaussian sampler $\mathsf{dGaussian}_\sigma$ for deriving a MLWEerror $\mathbf{e}$ with standard deviation σ from $\mathsf{seed}_{\mathsf{sk}}$,
- Hamming weight sampler HWT_h for deriving a sparse ternary $\mathbf{s}$ (resp. $\mathbf{r}$) with hamming weight $h = h_s$ (resp. $h = h_r$) from $\mathsf{seed}_{\mathsf{sk}}$ (resp. $\mathsf{seed}_{\mathbf{r}}$).

Algorithm 1 SMAUG.PKE.KeyGen: key generation

```
KeyGen($1^\lambda$):
1: $\mathsf{seed} \leftarrow \{0,1\}^{256}$
2: $(\mathsf{seed}_{\mathbf{A}}, \mathsf{seed}_{\mathsf{sk}}) \leftarrow H(\mathsf{seed})$
3: $\mathbf{A} \leftarrow \mathsf{expandA}(\mathsf{seed}_{\mathbf{A}}) \in \mathcal{R}_q^{k \times k}$
4: $\mathbf{s} \leftarrow \mathsf{HWT}_{h_s}(\mathsf{seed}_{\mathsf{sk}}) \in S_\eta^k$
5: $\mathbf{e} \leftarrow \mathsf{dGaussian}_\sigma(\mathsf{seed}_{\mathsf{sk}}) \in \mathcal{R}^k$
6: $\mathbf{b} = -\mathbf{A}^\top \cdot \mathbf{s} + \mathbf{e} \in \mathcal{R}_q^k$
7: return $\mathsf{pk} = (\mathsf{seed}_{\mathbf{A}}, \mathbf{b})$, $\mathsf{sk} = \mathbf{s}$
```

Algorithm 2 SMAUG.PKE.Enc: encryption

```
Enc($\mathsf{pk}, \mu; \mathsf{seed}_{\mathbf{r}}$):                ▷ $\mathsf{pk} = (\mathsf{seed}_{\mathbf{A}}, \mathbf{b})$, $\mu \in \mathcal{R}_t$
1: $\mathbf{A} = \mathsf{expandA}(\mathsf{seed}_{\mathbf{A}})$
2: if $\mathsf{seed}_{\mathbf{r}}$ is not given then $\mathsf{seed}_{\mathbf{r}} \leftarrow \{0,1\}^{256}$
3: $\mathbf{r} \leftarrow \mathsf{HWT}_{h_r}(\mathsf{seed}_{\mathbf{r}}) \in S_\eta^k$
4: $\mathbf{c}_1 = \lfloor p/q \cdot \mathbf{A} \cdot \mathbf{r} \rceil \in \mathcal{R}_p^k$
5: $c_2 = \lfloor p'/q \cdot \langle \mathbf{b}, \mathbf{r} \rangle + p'/t \cdot \mu \rceil \in \mathcal{R}_{p'}$
6: return $\mathsf{ct} = (\mathbf{c}_1, c_2)$
```

Algorithm 3 SMAUG.PKE.Dec: decryption

```
Dec($\mathsf{sk}, \mathbf{c}$):                ▷ $\mathsf{sk} = \mathbf{s}$, $\mathbf{c} = (\mathbf{c}_1, c_2)$
1: $\mu' = \lfloor t/p \cdot \langle \mathbf{c}_1, \mathbf{s} \rangle + t/p' \cdot c_2 \rceil \in \mathcal{R}_t$
2: return $\mu'$
```

3.2 Specification of SMAUG.KEM

We now introduce the key encapsulation mechanism SMAUG.KEM in Algorithms 4, 5, and 6. SMAUG.KEM is designed following the Fujisaki-Okamoto transform with implicit rejection using the non-perfectly correct PKE, whose security in the QROM is well-studied in [30, 32, 11]. It is constructed using SMAUG.PKE as an underlying IND-CPA secure PKE with the following building blocks, which can be implemented with symmetric primitives:

- hash function H for hashing a public key,
- hash function G for deriving a sharing key and a seed.

The Fujisaki-Okamoto transform in SMAUG has some differences from $\mathsf{FO}_m^{\not\perp}$ transform in [32], in its encapsulation and decapsulation methods. Encap of SMAUG uses the hashed public key when generating the sharing key and the randomness. This prevents some *multi-target attacks* on SMAUG. In Decap, the sharing key is alternatively re-generated if $\mathsf{ct} \neq \mathsf{ct}'$ holds for efficiency, and side-channel attacks (SCA) may leak the failure information. However, security can rely on the explicit FO transform $\mathsf{FO}_m^{\perp}$ even in the case of rejection leakages, which is treated in [34] with a competitive bound.

We remark that the randomly chosen message μ should be hashed additionally in the environments using a non-cryptographic system RNG. Using a true

Algorithm 4 $\mathsf{SMAUG.KEM.KeyGen}$: key generation

```
KeyGen(1^λ):
1: $(\mathsf{pk}, \mathsf{sk}') \leftarrow \mathsf{SMAUG.PKE.KeyGen}(1^\lambda)$
2: $d \leftarrow \{0,1\}^{256}$
3: return $\mathsf{pk}$, $\mathsf{sk} = (\mathsf{sk}', d)$
```

Algorithm 5 $\mathsf{SMAUG.KEM.Encap}$: encapsulation

```
Encap(pk):                                  ▷ $\mathsf{pk} = (\mathsf{seed}_{\mathbf{A}}, \mathbf{b})$
1: $\mu \leftarrow \{0,1\}^{256}$
2: $(K, \mathsf{seed}) \leftarrow G(\mu, H(\mathsf{pk}))$
3: $\mathsf{ct} \leftarrow \mathsf{SMAUG.PKE.Enc}(\mathsf{pk}, \mu; \mathsf{seed})$
4: return $\mathsf{ct}$, $K$
```

Algorithm 6 $\mathsf{SMAUG.KEM.Decap}$: decapsulation

```
Decap(sk, ct):                              ▷ $\mathsf{sk} = (\mathsf{sk}', d)$
1: $\mu' = \mathsf{SMAUG.PKE.Dec}(\mathsf{sk}', \mathsf{ct})$
2: $(K', \mathsf{seed}') \leftarrow G(\mu, H(\mathsf{pk}))$
3: $\mathsf{ct}' = \mathsf{SMAUG.PKE.Enc}(\mathsf{pk}, \mu'; \mathsf{seed}')$
4: if $\mathsf{ct} \neq \mathsf{ct}'$ then
5:     $(K', \cdot) \leftarrow G(d, H(\mathsf{ct}))$
6: return $K'$
```

random number generator (TRNG) is recommended for sampling the message μ.

3.3 Parameter sets

SMAUG is parameterized by integers n, k, q, p, p', t, h_s and h_r, and the standard deviation σ for the discrete Gaussian error in the key. We use the same ring dimension $n = 256$ and the message modulus $t = 2$ for every parameter set.

3.4 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 data storage strategy is applied to each component (resp. coefficient).

Sparse polynomial structure. As mentioned above, sparse polynomial space is S_η 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,

Parameters sets Security level	SMAUG-128 I	SMAUG-192 III	SMAUG-256 V
n	256	256	256
k	2	3	5
q	1024	2048	2048
p	256	256	256
p'	32	256	64
t	2	2	2
h_s	140	198	176
h_r	132	151	160
σ	1.0625	1.453713	1.0625
Classical core-SVP	120.0	181.7	264.5
Quantum core-SVP	105.6	160.9	245.2
DFP	-119.6	-136.1	-167.2
Secret key	176	236	218
Public key	672	1,088	1,792
Ciphertext	672	1,024	1,472

Table 2. The NIST security level, selected parameters, classical and quantum core-SVP hardness, decryption failure probability (in $\log_2$), and sizes (in bytes) of SMAUG.

since they are sampled from HWT. It is wasting memory to store polynomials 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 coefficients -1 is denoted by `neg_start`. Converting to degree arrays from sparse polynomials is done by `convToIdx`.

Packing and Unpacking. Packing means conversion to `uint8_t` array from polynomial in $\mathcal{R}_q$ and $\mathcal{R}_p$, or the structure of sparse polynomials 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 four coefficients to 5 `uint8_t` elements of the array since q is 1024. First, pack eight least significant bits of each coefficient to the corresponding `uint8_t` elements, then the left 2 bits are stored in the fifth from last `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 the polynomial since the 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.

Sampling algorithms. We use the following algorithms for sampling the randomnesses used in SMAUG:

– expandA for sampling a uniform random matrix $\mathbf{A}$,
– HWT_h for sampling sparse secrets, i.e., the secret key $\mathbf{s}$ and the ephimeral secret $\mathbf{r}$ with fixed hamming weights h_s and h_r,
– $\mathsf{dGaussian}_\sigma$ for sampling a discrete Gaussian error with standard deviation σ for MLWE sample.

We give the detailed algorithms of the sampling algorithms below.

Uniform random matrix sampler, expandA. We adopt the gen algorithm in Saber [44] for our uniform random matrix sampler expandA, given in Figure 7. This pseudorandom generator samples a public matrix $\mathbf{A}$ from uniformly random distribution over $\mathcal{R}_q^{k\times k}$.

Algorithm 7 expandA: uniform random matrix sampler

```
expandA(seed):                                      ▷ seed ∈ $\{0,1\}^{256}$
 1: buf ← XOF(seed)
 2: for $i$ from 0 to $k-1$ do
 3:     $\mathbf{A}[i]$ = bytes_to_Rq(buf + polybytes · $i$)    ▷ Convert to ring elements
 4: return $\mathbf{A}$
```

Hamming weight sampler, HWT_h*.* The hamming weight sampler, HWT_h in Figure 8, is adapted from the SampleInBall algorithm in Dilithium [21], having a secret-independent running time. It samples a ternary polynomial vector having a hamming weight of h.

Algorithm 8 HWT_h: hamming weight sampler

```
$\mathsf{HWT}_h$(seed):                                      ▷ seed ∈ $\{0,1\}^{256}$
 1: count = 0
 2: buf ← XOF(seed)
 3: for $i$ from $n-h$ to $n-1$ do
 4:     repeat
 5:         degree = buf[idx] ∧ mask
 6:     until degree < $i$
 7:     res[i] = res[degree]
 8:     res[degree] = ((buf[idx] ≫ 14) ∧ 0x02) − 1
 9: return convToIdx($\mathbf{s}$)                    ▷ Storing the indexes
```

Discrete Gaussian sampler, dGaussian. Karmakar et al. [36] suggested a constant-time discrete Gaussian sampling using the Knuth-Yao algorithm [38] and logic minimization. Motivated by this, we deployed the Quine-McCluskey method[4] and applied logic minimization technique on a cumulative distribution table (CDT). As a result, even though our dGaussian is constructed upon CDT tables, it is expressed with minimized bit operations and is constant-time. It is easily parallelizable and also suitable for IoT devices as its memory requirment is low. dGaussian samplers with $\sigma = 1.0625$ and $\sigma = 1.453713$ are given in Algorithm 9 and 10, respectively.

Algorithm 9 dGaussian$_{\sigma=1.0625}$: Discrete Gaussian sampler, $\sigma = 1.0625$

```
dGaussian_σ(x):
Require: $x = x_0x_1x_2x_3x_4x_5x_6x_7x_8x_9 \in \{0,1\}^{10}$
 1: $s = s_1s_0 = 00 \in \{0,1\}^2$
 2: $s_0 = x_0x_1x_2x_3x_4x_5x_7\overline{x_8}$
 3: $s_0 \mathrel{+}= (x_0x_3x_4x_5x_6x_8) + (x_1x_3x_4x_5x_6x_8) + (x_2x_3x_4x_5x_6x_8)$
 4: $s_0 \mathrel{+}= (\overline{x_2x_3x_6}x_8) + (\overline{x_1x_3x_6}x_8)$
 5: $s_0 \mathrel{+}= (x_6x_7\overline{x_8}) + (\overline{x_5x_6}x_8) + (\overline{x_4x_6}x_8) + (\overline{x_7}x_8)$
 6: $s_1 = (x_1x_2x_4x_5x_7x_8) + (x_3x_4x_5x_7x_8) + (x_6x_7x_8)$
 7: $s = (-1)^{x_9} \cdot s$                    ▷ · is the arithmetic multiplication
 8: return $s$
```

Algorithm 10 dGaussian$_{\sigma=1.453713}$: Discrete Gaussian sampler, $\sigma = 1.453713$

```
dGaussian_σ(x):
Require: $x = x_0x_1x_2x_3x_4x_5x_6x_7x_8x_9x_{10} \in \{0,1\}^{11}$
 1: $s = s_2s_1s_0 = 000 \in \{0,1\}^3$
 2: $s_0 = (x_0x_1x_2x_3x_5x_7x_8) + (x_1x_2x_3x_5\overline{x_6}x_7x_9) + (\overline{x_1x_2x_3}x_6x_7x_8)$
 3: $s_0 \mathrel{+}= (\overline{x_1x_2x_3x_5x_8}x_9) + (\overline{x_0x_2x_3x_5x_8}x_9)$
 4: $s_0 \mathrel{+}= (x_4x_5\overline{x_6}x_7x_9) + (x_3x_4x_8\overline{x_9}) + (\overline{x_5}x_6x_7x_8) + (\overline{x_4}x_6x_7x_8) + (\overline{x_4x_5x_8}x_9)$
 5: $s_0 \mathrel{+}= (x_5x_8\overline{x_9}) + (x_6x_8\overline{x_9}) + (x_7x_8\overline{x_9}) + (\overline{x_7x_8}x_9) + (\overline{x_6x_8}x_9)$
 6: $s_1 = (x_0x_1x_4\overline{x_5}x_6x_7x_9) + (x_2x_4\overline{x_5}x_6x_7x_9) + (x_3x_4\overline{x_5}x_6x_7x_9) + (x_5x_6x_7\overline{x_8}x_9)$
 7: $s_1 \mathrel{+}= (\overline{x_1x_2x_3}x_8x_9) + (\overline{x_7}x_8x_9) + (\overline{x_6}x_8x_9) + (\overline{x_5}x_8x_9) + (\overline{x_4}x_8x_9)$
 8: $s_2 = (x_1x_4x_5x_6x_7x_8x_9) + (x_2x_4x_5x_6x_7x_8x_9) + (x_3x_4x_5x_6x_7x_8x_9)$
 9: $s = (-1)^{x_{10}} \cdot s$                    ▷ · is the arithmetic multiplication
10: return $s$
```

Polynomial multiplication.

SMAUG uses the power-of-two moduli to ease the correct scaling and roundings. However, this makes the polynomial multiplications hard to benefit from

[4] We use the python package, from https://github.com/dreylago/logicmin.

Number Theoretic Transform (NTT). As a result, we propose a new polynomial multiplication benefit from the sparsity, adapted from [1, 39]. Our new multiplication, given in Figure 11, is constant-time and is faster than the original ones. Our secret storing method is similar to that of RLizard. The secret key stores only the degrees of non-zero coefficients, and the degrees are directly used in the polynomial multiplications.

Algorithm 11 poly_mult_add: polynomial multiplication using sparsity

```
poly_mult_add($a, b$, neg_start):                    ▷ $a \in \mathcal{R}_q$, $b \in S_\eta$
 1: for $i$ from 0 to neg_start - 1 do
 2:     degree = $b[i]$
 3:     for $j$ from 0 to $n$ do
 4:         $a[\text{degree} + j] = a[\text{degree} + j] + a[j]$;
 5: for $i$ from neg_start to len($b$) do
 6:     degree = $b[i]$
 7:     for $j$ from 0 to $n$ do
 8:         $a[\text{degree} + j] = a[\text{degree} + j] - a[j]$;
 9: for $j$ from 0 to $n$ do
10:     $a[j] = a[j] - a[n + j]$;
11: return $a$
```

4 Performance analysis

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

4.1 Description of platform

All benchmarks are obtained on one core of an Intel(R) Core(TM) i7-10700K CPU processor with clock speed 3.80GHz The benchmarking machine has 64 GB of RAM and runs Debian GNU/Linux with Linux kernel version 5.4.0. The implementation is 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

We instantiate the hash functions G, H and the extendable function XOF with the following symmetric primitives:

- G is instantiated with SHAKE-256,
- H is instantiated with SHA3-256,
- XOF is instantiated with SHAKE-128.

Table 3 reports the performance results of SMAUG, based on the constant-time implementation we provide on our wecite: kpqc.cryptolab.co.kr.

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

Schemes		KeyGen	Encap	Decap
SMAUG-128	*med*	77,220	77,370	92,916
	ave	77,940	77,063	93,046
SMAUG-192	*med*	154,862	136,616	163,354
	ave	155,311	136,702	164,782
SMAUG-256	*med*	266,704	270,123	305,452
	ave	270,123	270,672	304,292
	ave	262,580	240,545	275,035

Table 3. Median and average cycle counts of 1000 executions for SMAUG. Cycle counts are obtained on one core of an Intel Core i7-10700k, with TurboBoost and hyperthreading disabled.

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 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 get a pair $(K, ctxt)$, where K be either a session key corresponds to a ciphertext $ctxt$ or a random key (with $ctxt$ output from KEM.Encap). In the end, the adversary outputs its guess on whether the pair is correct. It wins if the guess is correct.

Our key encapsulation mechanism SMAUG.CCAKEM has IND-CCA2 security. Since our KEM is constructed based on the Fujisaki-Okamoto transform [24, 25] upon a public key encryption scheme 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 PKE is guaranteed, then the IND-CCA2 security of our 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 access 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 correctly guesses which message is used for the encryption.

5.1 Security definition

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 access to the public key* 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*

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

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* 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*

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

The original FO transforms $\mathsf{FO}_m^{\perp}$ constructs a KEM from a deterministic PKE, i.e., a de-randomized version. The encapsulation randomly samples a message m and uses the message's hash value $G(m)$ as randomness for encryption, generating a ciphertext. The sharing key $K = H(m)$ is generated by hashing (with different hash functions) the message. In the decapsulation, it first decrypts the ciphertext and recovers the message, m'. If it fails to decrypt or fails to"re-encrypt" the ciphertext equals the received one, and it outputs $\perp$. The sharing key can be generated by hashing the recovered message.

In the quantum setting, however, the FO transform with "implicit rejection" ($\mathsf{FO}_m^{\not\perp}$) is proven more secure than the original version, which implicitly outputs a pseudo-random sharing key if the re-encryption fails.

We recap the QROM proof of Hövelmanns et al. [11] allowing the KEMs constructed over non-perfect PKEs to have $\mathsf{IND\text{-}CCA}$ security:

Theorem 1 ([11], Theorem 1 & 2). *Let* G *and* H *be quantum-accessible random oracles, and the deterministic PKE is* ϵ*-injective. Then the advantage of* $\mathsf{IND\text{-}CCA}$ *attacker* $\mathcal{A}$ *with at most* Q_{Dec} *decryption queries and* Q_G *and* Q_H *hash queries at depth at most* d_G *and* d_H*, respectively, is*

$$\mathsf{Adv}_{\mathsf{KEM}}^{\mathsf{IND-CCA}}(\mathcal{A}) \leq 2\sqrt{(d_G + 2)\left(\mathsf{Adv}_{\mathsf{PKE}}^{\mathsf{IND-CPA}}(\mathcal{B}_1) + 8(Q_G + 1)/|\mathcal{M}|\right)} + \mathsf{Adv}_{\mathsf{PKE}}^{DF}(\mathcal{B}_2) + 4\sqrt{d_H Q/|\mathcal{M}|} + \epsilon,$$

where $\mathcal{B}_1$ *is an* $\mathsf{IND\text{-}CPA}$ *adversary on PKE and* $\mathcal{B}_2$ *is an adversary against finding a decryption failing ciphertext.*

5.2 Security proof

We now prove the completeness of SMAUG.PKE.

Theorem 2 (Completeness of SMAUG.PKE). *Let* $\mathbf{A}$*,* $\mathbf{b}$*,* $\mathbf{s}$*,* $\mathbf{e}$*, and* $\mathbf{r}$ *are defined as in Algorithms 1, 2, and 3. Let the moduli* t*,* p*,* p'*, and* q *satisfy* $t \,|\, p \,|\, q$ *and* $t \,|\, p' \,|\, q$*. Let* $\mathbf{e}_1 \in \mathcal{R}_{\mathbb{Q}}^k$ *and* $e_2 \in \mathcal{R}_{\mathbb{Q}}$ *be the rounding errors introduced from the scalings and roundings of* $\mathbf{A} \cdot \mathbf{r}$ *and* $\mathbf{b}^T \cdot \mathbf{r}$*. That is,* $\mathbf{e}_1 = \frac{q}{p}(\lfloor \frac{p}{q} \cdot \mathbf{A} \cdot \mathbf{r} \rceil \mod p) - (\mathbf{A} \cdot \mathbf{r} \mod q)$ *and* $e_2 = \frac{q}{p'}(\lfloor \frac{p'}{q} \cdot \langle \mathbf{b}, \mathbf{r} \rangle \rceil \mod p') - (\langle \mathbf{b}, \mathbf{r} \rangle \mod q)$*. Let*

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

where the probability is taken over the randomness of the encryption. Then SMAUG.PKE *in Algorithms 1, 2, and 3 is* $(1-\delta)$*-correct. That is, for every message* μ *and every key-pair* $(\mathsf{pk}, \mathsf{sk})$ *returned by* $\mathsf{KeyGen}(1^\lambda)$*, the decryption fails with a probability less than* δ*.*

Proof. By the definition of $\mathbf{e}_1$ and e_2, it holds that

$$\mathbf{c}_1 = \frac{p}{q} \cdot (\mathbf{A} \cdot \mathbf{r} + \mathbf{e}_1) \mod p \quad \text{and} \quad c_2 = \frac{p'}{q} \cdot (\langle \mathbf{b}, \mathbf{r} \rangle + e_2) + \frac{p'}{t} \cdot \mu \mod p',$$

where the coefficients of $\mathbf{e}_1$ and e_2 are in $\mathbb{Z} \cap (-\frac{q}{2p}, \frac{q}{2p}]$ and $\mathbb{Z} \cap (-\frac{q}{2p'}, \frac{q}{2p'}]$, respectively. Thus, the decryption of ciphertext with respect to the message μ and the randomness $\mathbf{r}$ can be written as

$$\begin{aligned}
\left\lfloor \frac{t}{p} \cdot \langle \mathbf{c}_1, \mathbf{s} \rangle + \frac{t}{p'} \cdot c_2 \right\rceil \mod t &= \left\lfloor \frac{t}{q} \left(\langle \mathbf{A} \cdot \mathbf{r}, \mathbf{s} \rangle + \langle \mathbf{e}_1, \mathbf{s} \rangle + \langle \mathbf{b}, \mathbf{r} \rangle + e_2 \right) + \mu \right\rceil \mod t \\
&= \left\lfloor \frac{t}{q} \left(\langle \mathbf{A}^\top \cdot \mathbf{s} + \mathbf{b}, \mathbf{r} \rangle + \langle \mathbf{e}_1, \mathbf{s} \rangle + e_2 \right) + \mu \right\rceil \mod t \\
&= \mu + \left\lfloor \frac{t}{q} \left(\langle \mathbf{e}, \mathbf{r} \rangle + \langle \mathbf{e}_1, \mathbf{s} \rangle + e_2 \right) \right\rceil \mod t.
\end{aligned}$$

Thus, the decryption result is equal to μ if and only if every coefficient of $\langle \mathbf{e}, \mathbf{r} \rangle + \langle \mathbf{e}_1, \mathbf{s} \rangle + e_2$ is in the interval $[-\frac{q}{2t}, \frac{q}{2t})$. This concludes the proof of completeness of SMAUG.PKE. □

We then give the IND-CPA security of SMAUG.PKE.

Theorem 3 (IND-CPA security of SMAUG.PKE). *Assuming pseudorandomness of the underlying sampling algorithms, the* IND-CPA *security of* SMAUG.PKE *can be tightly reduced to the decisional MLWE and MLWR problems. Specifically, for any IND-CPA-adversary* $\mathcal{A}$ *of* SMAUG.PKE*, there exist adversaries* $\mathcal{B}_0$*,* $\mathcal{B}_1$*,* $\mathcal{B}_2$*, and* $\mathcal{B}_3$ *attacking the pseudorandomness of* H *and the sampling algorithms, MLWE, and MLWR problems, such that,*

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{IND-CPA}}_{\mathsf{SMAUG.PKE}}(\mathcal{A}) \leq\ & \mathsf{Adv}^{\mathsf{PR}}_{H}(\mathcal{B}_0) + \mathsf{Adv}^{\mathsf{PR}}_{\mathsf{expandA},\mathsf{HWT},\mathsf{dGaussian}}(\mathcal{B}_1) \\
&+ \mathsf{Adv}^{\mathsf{MLWE}}_{n,q,k,k,\mathsf{HWT}_{h_s},\mathsf{dGaussian}_\sigma}(\mathcal{B}_2) + \mathsf{Adv}^{\mathsf{MLWR}}_{n,p,q,k+1,k,\mathsf{HWT}_{h_r}}(\mathcal{B}_3).
\end{aligned}$$

Proof. The proof proceeds by a sequence of hybrid games from G_0 to G_4 defined as follows:

- G_0: the genuine $\mathsf{IND\text{-}CPA}$ game,
- G_1: identical to G_0, except that the public key is changed into $(\mathbf{A}, \mathbf{b})$,
- G_2: identical to G_1, except that the sampling algorithms are changed into truly random samplings,
- G_3: identical to G_2, except that $\mathbf{b}$ is randomly chosen from $\mathcal{R}_q^k$,
- G_4: identical to G_3, except that the ciphertext is randomly choosen from $\mathcal{R}_p^k \times \mathcal{R}_{p'}$. As a result, the public key and the ciphertexts are truly random.

We denote the advantage of the adversary on each game G_i as Adv_i, where $\mathsf{Adv}_0 = \mathsf{Adv}^{\mathsf{IND-CPA}}_{\mathsf{SMAUG.PKE}}(\mathcal{A})$ and $\mathsf{Adv}_4 = 0$. Then, it holds that

$$|\mathsf{Adv}_0 - \mathsf{Adv}_1| \leq \mathsf{Adv}^{\mathsf{PR}}_{H}(\mathcal{B}_0),$$

for some adversary $\mathcal{B}_0$ against the pseudorandomness of the hash function. Since the view of the transcripts in the hybrid games G_1 and G_2 are different only in the randomness sampling, it holds that

$$|\mathsf{Adv}_1 - \mathsf{Adv}_2| \leq \mathsf{Adv}^{\mathsf{PR}}\mathsf{expandA}, \mathsf{HWT}, \mathsf{dGaussian}(\mathcal{B}_1),$$

for some adversary, $\mathcal{B}_1$ attacking the pseudorandomness of at least one of the samplers. The difference in the games G_2 and G_3 is that the polynomial vector $\mathbf{b}$ is sampled as a part of an MLWE sample in G_2 or randomly in G_3. Thus, the difference between the advantages Adv_2 and Adv_3 can be bounded by $\mathsf{Adv}^{\mathsf{MLWE}}_{n,q,k,k,\mathsf{HWT}_{h_s},\mathsf{dGaussian}_\sigma}(\mathcal{B}_2)$, where $\mathcal{B}_2$ is an adversary against decisional MLWE problem, distinguishing the MLWE samples from random. Lastly, the only difference in the hybrids G_3 and G_4 is that the ciphertexts are generated in different ways: random over $\mathcal{R}_p^k \times \mathcal{R}_{p'}$ versus $(\mathbf{c}_1, \lfloor p'/p \cdot c_2 \rceil)$, where

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

If $\mathcal{A}$ distinguishes the two ciphertexts, then we can construct an adversary $\mathcal{B}_3$ distinguishing the MLWR sample from random, as follows: *for given a sample* $(\mathbf{A}, \mathbf{b}) \in \mathcal{R}_q^{(k+1)\times k} \times \mathcal{R}_p^{k+1}$*,* $\mathcal{B}_3$ *rewrites* $\mathbf{b}$ *as* $(\mathbf{b}_1, b_2) \in \mathcal{R}_p^k \times \mathcal{R}_p$*, computes* $(\mathbf{b}_1, \lfloor p'/p \cdot b_2 \rceil)$*, and use* $\mathcal{A}$ *to decide the ciphertext type, which will be the output of* $\mathcal{B}_3$*.* Thus, it holds that

$$|\mathsf{Adv}_3 - \mathsf{Adv}_4| \leq \mathsf{Adv}^{\mathsf{MLWR}}_{n,p,q,k+1,k,\mathsf{HWT}_{h_r}}(\mathcal{B}_3).$$

This concludes the proof. □

We now show the completeness of $\mathsf{SMAUG.KEM}$ based on the completeness of the underlying public key encryption scheme, $\mathsf{SMAUG.PKE}$.

Theorem 4 (Completeness of SMAUG.KEM). *We borrow the notations and assumptions from Theorem 2 and Algorithms 4, 5, and 6. Then* SMAUG.KEM *is also* $(1-\delta)$*-correct. That is, for every key-pair* $(\mathsf{pk}, \mathsf{sk})$ *generated by* $\mathsf{KeyGen}(1^\lambda)$*, the shared keys* K *and* K' *are identical with probability larger than* $1-\delta$*, if* SMAUG.PKE *is* $(1-\delta)$*-correct.*

Proof. The shared keys K and K' are identical if the decryption succeeds. Assuming the pseudorandomness of the hash function G, the probability of being $K \neq K'$ can be bounded by the decryption failure probability of SMAUG.PKE. The completeness of SMAUG.PKE (Theorem 2) concludes the proof. □

The IND-CPA security of SMAUG.PKE and Theorem 1 implies the IND-CCA security of SMAUG KEM scheme in the QROM.

5.3 Security strength categories

We target the security of our 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 5.4), and give the following parameter sets correspond to the security levels. We also give the security estimated by using another methodology, MATZOV [40], which reports much higher security.

Parameters sets	SMAUG 128	SMAUG 192	SMAUG 256
Target security	I	III	V
Classical core-SVP	120.0	181.7	264.5
Quantum core-SVP	105.6	160.9	245.2
MATZOV	140.9	199.1	274.6

Table 4. Core-SVP hardness for security levels I, III and V.

5.4 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 SMAUG.KEM. All the best-known attacks are essentially finding a nonzero short vector in the Euclidean lattices, using the Block–Korkine–Zolotarev (BKZ) lattice reduction algorithm [17, 28, 43].

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 β, rather than in the entire high-dimensional lattice. β-BKZ is the BKZ algorithm using SVP solver in the block size β, and the parameter b determines the quality of the resulting basis and the time complexity. The time complexity of the β-BKZ algorithm is the same as the SVP solver for dimension β with a polynomial factor. Indeed, there is a quality/time trade-off: If β gets larger, better quality will be guaranteed, and the time complexity for the SVP solver will exponentially increase. 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 the classical solver. The fastest known quantum variant was recently proposed by Chailloux and Loyer in [16] and takes time $\approx 2^{0.257\beta+o(\beta)}$.

Based on the BKZ algorithm, we will follow the *Core-SVP* methodology as in [5] and in the subsequent lattice-based post-quantum schemes [4, 14, 22, 23], 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.

However, when dealing with the hardness of the MLWR problem, we treat it as an MLWE problem since there are no known attacks that use the deterministic error term in MLWR structure. Further more, the reduction from the $\mathsf{(M)LWE}$ problem to the $\mathsf{(M)LWR}$ problem were also given by Banerjee et al. [9] and the improvements [6, 7, 13]. 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 5. The security is estimated via lattice estimator [3] and is represented as core-SVP hardness. The python script used for this estimation is attached in the submission files with this document. We assumed that the number of 1 is equal to the number of -1 for simplicity. The lowest value, which becomes the core-SVP hardness of each scheme, is indicated with bold type. We note that hybrid attacks require significant memory, for e.g., at least $2^{171.4}$ bits for SMAUG 192 dual hybrid attack or $2^{233.2}$ bits for SMAUG 256 dual hybrid attack.

5.4.1 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

Parameters sets Target security	SMAUG 128 I	SMAUG 192 III	SMAUG 256 V
Classical core-SVP hardness for MLWE			
Primal attack	120.0	182.8	300.5
Primal attack (BDD)	120.9	184.4	302.4
Primal attack (Hybrid)	121.3	185.0	297.2
Dual attack	125.9	190.4	311.0
Dual attack (Hybrid)	122.7	**181.7**	**264.5**
Classical core-SVP hardness for MLWR			
Primal attack	**120.0**	188.9	322.7
Primal attack (BDD)	121.5	191.9	329.5
Primal attack (Hybrid)	121.5	193.0	309.4
Dual attack	125.9	197.1	334.9
Dual attack (Hybrid)	122.1	181.8	274.3
Quantum core-SVP hardness for MLWE			
Primal attack	105.6	**160.9**	264.5
Primal attack (BDD)	106.5	162.4	265.9
Primal attack (Hybrid)	106.9	162.9	267.0
Dual attack	110.8	167.6	273.7
Dual attack (Hybrid)	111.5	165.5	**245.2**
Quantum core-SVP hardness for MLWR			
Primal attack	**105.6**	166.3	284.0
Primal attack (BDD)	107.0	168.9	290.0
Primal attack (Hybrid)	107.0	170.0	283.1
Dual attack	110.8	173.5	294.8
Dual attack (Hybrid)	111.3	167.1	255.8

Table 5. Cost of known lattice reduction attacks. The security is represented as core-SVP hardness.

$m \times \ell$ sub-matrix of $\mathbf{A}$ and $\mathbf{b}_{[m]}$ is the uppermost length m sub-vector of $\mathbf{b}$ for $m \leq k$. Then, a short non-zero vector in the lattice Λ_m can be transformed to a short non-trivial solution to the LWE equation. Primal attack solves the SVP problem in the lattice Λ_m using β-BKZ, increasing the block size β for all possible m.

5.4.2 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 $m \leq k$. Then, again, the short non-zero vector in the lattice Λ'_m induces a short non-trivial solution to the LWE problem. Dual attack solves the SVP problem in the lattice Λ'_m using β-BKZ, increasing the block size β, for all possible m.

5.4.3 Description of the hybrid variants. For both Primal and Dual attacks, some variants combine the attack with the combinatorial attacks, including 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 from the particular choice of the small or sparse secret used in the LWE problem. By exploiting the secret as a preprocessing or using the MITM approach to guess the part of the sparse secret, the attacks may be 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 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 dividing 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 β 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 the Core-SVP method to the hybrid attacks.

5.4.4 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, 2] using Gröbener's basis or Coded-BKW attacks [37, 27] are also considered, using the lattice estimator [3]. Still, they have much higher attack costs than the previously introduced attacks, with significantly higher memory requirements.

Recently, May [41] proposed a combinatorial attack called *Meet-LWE*, an improved version of Odlyzko's Meet-in-the-Middle approach. It brings down the asymptotic attack complexity from $\mathcal{S}^{0.5}$ to $\mathcal{S}^{0.25}$, which reaches to $\mathcal{S}^{0.3}$ in some lattice-based schemes with non-asymptotic instantiations. We note that the asymptotic complexities are far from the estimated attack costs in SMAUG parameter sets.

Parameters sets Target security		SMAUG 128 I	SMAUG 192 III	SMAUG 256 V
Classical core-SVP		120.0	181.7	264.5
Algebraic & Combinatorial attacks				
Arora-Ge	time (mem)	741.3 (598.0)	983.4 (636.5)	- -
BKW	time (mem)	144.7 (133.7)	202.0 (190.7)	274.6 (256.9)
Meet-LWE	time (mem)	164.3 (143.7)	213.8 (192.4)	283.2 (254.7)

Table 6. Attack costs beyond Core-SVP.

We summarize the costs of the algebraic and combinatorial attacks in Table 6. Attack costs for Arora-Ge and Coded-BKW are estimated with lattice estimator [3]. The estimated cost of Arora-Ge attack on SMAUG 256 is not determined by lattice-estimator, outputting ∞, which is at least a thousand bits of security. The costs for the Meet-LWE attack are estimated with a python script[5] based on May's analysis [41], best among Rep-1 and Rep-2.

Note that Rep-1 and Rep-2 always outperform Rep-0. It assumes the number of 1 is equal to the number of -1. We note that this is also assumed when estimating core-SVP hardness, which leads to several bits lower security. The time and memory costs are given in the base two logarithms, i.e., bit-cost; however, the units are not specified. We expect that the time and memory units are for generating and storing one element in a list. We just followed May's paper. For instance, Meet-LWE attack for SMAUG 128 requires $2^{164.3}$ runs and $2^{143.7}$ consumption of unit operation and memory, respectively.

We remark that, as in all the existing security estimate methods, including core-SVP hardness, there are gaps between the actual attack costs and the estimated costs, given in the tables.

6 Summary and future works

Our KEM scheme takes advantage of recent approaches in lattice-based cryptography. By bringing the MLWE-based public key generation and the sparse secret to Saber, SMAUG exploits the remaining room for efficiency. SMAUG has *the shortest*[6] ciphertext among the LWE/LWR-based KEM schemes while maintaining the *high* security and *even better* performance.

Future works and directions. First, we will keep studying the hardness of sparse secret LWEand LWR problems. These variants are already studied for a while, but more effort is required for long-term quantum security. Secondly, we will provide an optimized implementation of SMAUG in various devices, as SMAUG is an attractive candidate for resource-restricted devices. Lastly, we will include analysis against side-channel attacks and provide secure implementations.

References

1. Akleylek, S., Alkım, E., Tok, Z.Y.: Sparse polynomial multiplication for lattice-based cryptography with small complexity. The Journal of Supercomputing **72**, 438–450 (2016)
2. 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

[5] The script can be found on the team SMAUG website: http://kpqc.cryptolab.co.kr/

[6] Among the recent KEMs with comparable security, DFP, and SCA-resistance. In particular, we do consider KEMs using ECC on messages to reduce the DFP.

3. Albrecht, M.R., Player, R., Scott, S.: On the concrete hardness of learning with errors. Journal of Mathematical Cryptology **9**(3), 169–203 (2015)
4. 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
5. Alkim, E., Ducas, L., Pöppelmann, T., Schwabe, P.: Post-quantum key exchange - A new hope. 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. Bindel, N., Hamburg, M., Hövelmanns, K., Hülsing, A., Persichetti, E.: Tighter proofs of CCA security in the quantum random oracle model. pp. 61–90 (2019). https://doi.org/10.1007/978-3-030-36033-7˙3
12. 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)
13. 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)
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. 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. pp. 1006–1018 (2016). https://doi.org/10.1145/2976749.2978425
16. Chailloux, A., Loyer, J.: Lattice sieving via quantum random walks. In: Tibouchi, M., Wang, H. (eds.) Advances in Cryptology - ASIACRYPT. pp. 63–91 (2021)
17. 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)
18. 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)

19. 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. pp. 160–177 (2018). https://doi.org/10.1007/978-3-319-98113-0˙9
20. D'Anvers, J.P., Guo, Q., Johansson, T., Nilsson, A., Vercauteren, F., Verbauwhede, I.: Decryption failure attacks on ind-cca secure lattice-based schemes. In: Lin, D., Sako, K. (eds.) Public-Key Cryptography – PKC 2019. pp. 565–598. Springer International Publishing, Cham (2019)
21. Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schwabe, P., Seiler, G., Stehlé, D.: CRYSTALS-Dilithium: A lattice-based digital signature scheme **2018**(1), 238–268 (2018). https://doi.org/10.13154/tches.v2018.i1.238-268, https://tches.iacr.org/index.php/TCHES/article/view/839
22. 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)
23. 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)
24. 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)
25. Fujisaki, E., Okamoto, T.: Secure integration of asymmetric and symmetric encryption schemes. Journal of cryptology **26**(1), 80–101 (2013)
26. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: Proceedings of the twenty-eighth annual ACM symposium on Theory of computing. pp. 212–219 (1996)
27. Guo, Q., Johansson, T., Stankovski, P.: Coded-bkw: Solving lwe using lattice codes. In: Annual Cryptology Conference. pp. 23–42. Springer (2015)
28. 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)
29. 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)
30. Hofheinz, D., Hövelmanns, K., Kiltz, E.: A modular analysis of the Fujisaki-Okamoto transformation. pp. 341–371 (2017). https://doi.org/10.1007/978-3-319-70500-2˙12
31. Hong, S.: Lizarderror. https://github.com/swanhong/LizardError (2018)
32. Hövelmanns, K., Kiltz, E., Schäge, S., Unruh, D.: Generic authenticated key exchange in the quantum random oracle model. pp. 389–422 (2020). https://doi.org/10.1007/978-3-030-45388-6˙14
33. Howgrave-Graham, N., Nguyen, P.Q., Pointcheval, D., Proos, J., Silverman, J.H., Singer, A., Whyte, W.: The impact of decryption failures on the security of ntru encryption. In: Boneh, D. (ed.) Advances in Cryptology - CRYPTO 2003. pp. 226–246. Springer Berlin Heidelberg, Berlin, Heidelberg (2003)
34. Hövelmanns, K., Hülsing, A., Majenz, C.: Failing gracefully: Decryption failures and the fujisaki-okamoto transform. Cryptology ePrint Archive, Paper 2022/365 (2022), https://eprint.iacr.org/2022/365, https://eprint.iacr.org/2022/365

35. 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)
36. 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)
37. 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)
38. Knuth, D., Yao, A.: Algorithms and Complexity: New Directions and Recent Results, chap. The complexity of nonuniform random number generation. Academic Press (1976)
39. 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)
40. MATZOV: Report on the Security of LWE: Improved Dual Lattice Attack (Apr 2022). https://doi.org/10.5281/zenodo.6493704, https://doi.org/10.5281/zenodo.6493704
41. May, A.: How to meet ternary lwe keys. In: Malkin, T., Peikert, C. (eds.) Advances in Cryptology – CRYPTO 2021. pp. 701–731. Springer International Publishing, Cham (2021)
42. 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)
43. Schnorr, C.P., Euchner, M.: Lattice basis reduction: Improved practical algorithms and solving subset sum problems. Mathematical programming **66**(1), 181–199 (1994)
44. Vercauteren, I.F., Sinha Roy, S., D'Anvers, J.P., Karmakar, A.: Saber: Mod-lwr based kem (round 3 submission)