SMAUG: Pushing Lattice-based Key Encapsulation Mechanisms to the Limits

Jung Hee Cheon[1,2], Hyeongmin Choe[1], Dongyeon Hong[2], and MinJune Yi[1]

[1] Seoul National University
{jhcheon, sixtail528, yiminjune}@snu.ac.kr
[2] CryptoLab Inc.
{decenthong93}@cryptolab.co.kr

Abstract. Recently, NIST has announced Kyber, a lattice-based key encapsulation mechanism (KEM), as a post-quantum standard. However, it is not the most efficient scheme among the NIST's KEM finalists. Saber enjoys more compact sizes and faster performance, and Mera et al. (TCHES '21) further pushed its efficiency, proposing a shorter KEM, Sable. As KEM are frequently used on the Internet, such as in TLS protocols, it is essential to achieve high efficiency while maintaining sufficient security.

In this paper, we further push the efficiency limit of lattice-based KEMs by proposing SMAUG, a new post-quantum KEM scheme submitted to the Korean Post-Quantum Cryptography (KPQC) competition, whose IND-CCA2 security is based on the combination of MLWE and MLWR problems. We adopt several recent developments in lattice-based cryptography, targeting the *smallest* and the *fastest* KEM while maintaining high enough security against various attacks, with a full-fledged use of sparse secrets. Our design choices allow SMAUG to balance the decryption failure probability and ciphertext sizes without utilizing error correction codes, whose side-channel resistance remains open.

With a constant-time C reference implementation, SMAUG achieves ciphertext sizes up to 12% and 9% smaller than Kyber and Saber, with much faster running time, up to 103% and 58%, respectively. Compared to Sable, SMAUG has the same ciphertext sizes but a larger public key, which gives a trade-off between the public key size versus performance; SMAUG has 39%-55% faster encapsulation and decapsulation speed in the parameter sets having comparable security.

Keywords: Key Encapsulation Mechanism, Public Key Encryption, Post-Quantum Cryptography, Module Learning With Errors, Module Learning With Roundings.

1 Introduction

Recent advances in quantum computers raise the demand for quantum-resistant cryptographic protocols, i.e., the Post-Quantum Cryptographic (PQC) schemes. As the demand for quantum-secure cryptographic protocols has increased, the

American National Institute of Standards and Technology (NIST) has established a standardization process focusing on Public Key Encryption (PKE), digital signature, and Key Encapsulation Mechanism (KEM). In particular, KEM is one of the most widely used algorithms over the Internet, such as in Transport Layer Security (TLS) protocols; however, the KEM currently used in the protocol is considered vulnerable to quantum attacks.

Various lattice-based KEMs [6,13,17,18,21,25,49,58] have been proposed and submitted to NIST standardization to secure the Internet in the quantum world. Also, diverse techniques related to the lattice-based KEMs have been suggested, significantly improving the efficiency or security. Recently NIST announced that the lattice-based KEM scheme Kyber [17] is selected as a future standard. Other candidates, such as Saber [58] or NTRU [21], have enough security and efficiency; however, the standardization restrictions forced NIST and the community to choose only one of them.

As of independent interest to NIST's standardization, the KEM's efficiency is crucial since it is executed and transmitted frequently on the Internet. In particular, the TLS protocols are also necessary for embedded devices, so the efficiency requirement has become even more pressing with the proliferation of the Internet of Things (IoT). To this end, some variants of Saber focusing on efficiency, Scabbard [53], have been recently proposed by Mera et al. Scabbard consists of three schemes based on the R/MLWR problem, Floreta, Espada, and Sable, each targeting HW/SW-efficient, parallelizable, and shorter KEM than Saber. In particular, Sable achieves the smallest public key and ciphertext sizes among the KEM schemes targetting the NIST's security level 1 and low enough decryption failure probability (DFP). Keep in mind that the public key and ciphertext sizes must be as small as possible, as they are transmitted frequently; this question comes naturally:

What is the "efficiency limit" of the lattice-based KEMs?

1.1 Our results

In this work, we answer the above question by proposing a new lattice-based KEM, SMAUG, constructed based on both MLWE and MLWR problems. By bringing the MLWE-based public key generation and the sparse secret to Sable, SMAUG exploits a remaining room for efficiency. SMAUG tends to have *the shortest* ciphertext among the LWE/LWR-based KEM schemes while maintaining the *high* security and *even better* performance.

The SMAUG. The design rationale of SMAUG aims to achieve small ciphertext and public key with low computational cost while maintaining security against a variety of 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 [35].
- 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 [27, 41].
- 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.

	Sizes (bytes)			**Security**			**Cycle**		
Scheme	**sk**	**pk**	**ct**	**Lvl.**	**Sec.**	**DFP**	KeyGen	Encap	Decap
SMAUG-128	176	672	672	1	120	2^{-120}	77k	77k	92k
SMAUG-192	236	1088	1024	3	181	2^{-136}	153k	136k	163k
SMAUG-256	218	1792	1472	5	264	2^{-167}	266k	270k	305k

Table 1: Parameter sets of SMAUG for NIST's security levels 1, 3, and 5. Security (Sec.) is given in classical core-SVP hardness. One core of an Intel Core i7-10700k is used for cycle counts.

To achieve this goal, we exploit the possible combination of the known techniques in lattice-based cryptography, such as underlying lattice assumptions, ciphertext compression, Fujisaki-Okamoto (FO) transforms, and the sparse secret.

Among the possibilities on the choice of lattice assumptions, we conclude to use LWE-based key generation and LWR-based encapsulation with sparse secrets, following the construction of Lizard [25] and RLizard [49], but adapted to module lattices. This choice allows SMAUG to enjoy the conservative secret key security based on the hardness of the un-rounded module learning-with-errors

(MLWE) problem while exploring more efficiency on encapsulation and decapsulation by using the module learning with roundings (MLWR)-based approach.

Note that other possibilities, such as Kyber or Saber using sparse secrets, are also considered, but they reported worse results than SMAUG. For Kyber with the sparse secret, it has an expensive Gaussian error sampling, making Saber with the sparse secret a more appealing candidate. However, the modules should be decreased to maintain security, which makes DFP higher. Bringing the MLWE-based key generation lowers the noise introduced in the key generation and significantly reduces the overall error in the ciphertext. As the noise is multiplied by the ephemeral secret during encapsulation, the one last combination– the KEM with MLWR-based key generation, MLWE-based encapsulation, and sparse secret–can not avoid suffering from a more significant overall error.

Sparse secret allows SMAUG to enjoy fast polynomial multiplications and small secret keys. The sparse secret is widely used in homomorphic encryption (HE) schemes to speed up the expensive homomorphic operations and to reduce the noise [23, 37], whose ability is attractive for efficient KEMs. By using the SampleInBall algorithm of Dilithium [30], we can efficiently sample the sparse ternary secrets. Regarding security, the hardness reductions for sparse LWE and LWR problems [24, 25] from LWE problem exists; however, the concrete security should be treated carefully[3].

We take the recent approaches in Fujisaki-Okamoto (FO) transform for key exchange in the quantum random oracle model (QROM) [40] and apply it to our IND-CPA PKE, SMAUG.PKE. Precisely, we use the FO transform with implicit rejection and no ciphertext contributions ($\mathsf{FO}_m^{\not\perp}$).

We delicately choose three parameter sets for SMAUG regarding NIST's security levels 1, 3, and 5 (classical core-SVP hardness of 120, 180, and 260, respectively) and having smaller DFP than Saber.

Comparison to other KEMs. We compare SMAUG with the NIST-selected Kyber, one of the round 3 finalists Saber, and its variant Sable in Table 2.

Compared to Kyber-512 [17], the NIST-selected standard targeting the security level 1, SMAUG-128, has 16% and 12% smaller public key and ciphertext, respectively. The secret key size of SMAUG is tiny and ready to use, which enable efficient management of secure zone in restricted IoT devices. With high enough security and low enough decryption failure probability, SMAUG further achieves 110% and 103% speed up in encapsulation and decapsulation.

Compared to LightSaber [58], one of the round 3 finalists with the security level 1, SMAUG-128, has 9% smaller ciphertext and the same public key size. The secret key is, again, significantly smaller than LightSaber, with a 58% and 44% speed up in encapsulation and decapsulation, respectively.

When compared to Sable [53], SMAUG-128 has the same ciphertext size but a larger public key size. It can be seen as a trade-off as SMAUG achieves 48% and

[3] We use the lattice-estimator [3], from `https://github.com/malb/lattice-estimator`, commit `9687562`. We also consider some attacks not implemented in the lattice-estimator.

Schemes	Sizes (bytes) sk	pk	ct	Security Classic.	DFP	Cycle (ratio) KeyGen	Encap	Decap
NIST's security level 1 (120)								
Kyber512 [17]	1632	800	768	118	2^{-139}	1.70	2.10	2.03
LightSaber [58]	832	672	736	118	2^{-120}	1.21	1.58	1.44
LightSable [53]	800	608	672	114	2^{-139}	1.10	1.48	1.39
SMAUG-128	176	672	672	120	2^{-120}	1	1	1
NIST's security level 3 (180)								
Kyber768 [17]	2400	1184	1088	183	2^{-164}	1.38	1.84	1.75
Saber [58]	1248	992	1088	189	2^{-136}	1.21	1.64	1.47
Sable [53]	1152	896	1024	185	2^{-143}	1.10	1.48	1.39
SMAUG-192	236	1088	1024	181	2^{-136}	1	1	1
NIST's security level 5 (260)								
Kyber1024 [17]	3168	1568	1568	256	2^{-174}	1.25	1.38	1.36
FireSaber [58]	1664	1312	1472	260	2^{-165}	1.21	1.58	1.44
FireSable [53]	1632	1312	1376	223	2^{-208}	1.03	1.25	1.22
SMAUG-256	218	1792	1472	264	2^{-167}	1	1	1

Table 2: Comparison of KEM schemes with comparable efficiency and security. Security is given in the classical core-SVP hardness with DFP. The cycle counts are given relative to that of SMAUG's, reported in the same machine.

39% faster encapsulation and decapsulation speed with a significantly smaller secret key and 6 bits higher security.

In NIST's security levels 3 and 5, SMAUG similarly outperforms Kyber and provides a trade-off with Saber and Sable. For instance, SMAUG-128 has the same ciphertext size as level-3 Sable, a larger public key but a smaller secret key, and is faster than Sable. Note that FireSable has a smaller ciphertext than SMAUG-256; however, it has a classical core-SVP hardness way lower than 260. We refer to Section 6.2 for detailed comparisons.

1.2 Related work: LWE/LWR-based PKEs

We focus on the LWE/LWR-based IND-CPA secure PKEs, which can be turned into IND-CCA[4] secure KEM by applying FO transforms. The original Regev's public key encryption [55], followed by most recent LWE-LWR-based PKE constructions, bases its security on the LWE assumption. It generates an LWE sample as a public key and returns a ciphertext of a binary message by scaling and adding a public key multiplied by a random vector. In more detail, the public key is $(\mathbf{A}, \mathbf{b} = \mathbf{A} \cdot \mathbf{s} + \mathbf{e} \bmod q) \in \mathbb{Z}_q^{k \times \ell} \times \mathbb{Z}_q^k$, where $\mathbf{s}$ is a secret key and $\mathbf{e}$ is an error term. The encryption of a binary message μ is a ciphertext $(\mathbf{r}^\top \cdot \mathbf{A}, \mathbf{r}^\top \cdot \mathbf{b} + \lfloor q/2 \cdot \mu \rceil)$, where $\mathbf{r}$ is a random vector, called an ephimeral secret. The ciphertext is then statistically uniform over the range due to the leftover

[4] In this paper, we only consider the adaptive IND-CCA attacks, i.e., often called as IND-CCA2 attacks.

hash lemma; however, it makes the size of the matrix $\mathbf{A}$ too massive for even a binary message to be used efficiently.

The approaches thereafter tried to gain more efficiency. Lindner and Peikert [50] introduce an encryption using an LWE sample as a ciphertext, i.e., $(\mathbf{r}^\top \cdot \mathbf{A} + \mathbf{e}_1, \mathbf{r}^\top \cdot \mathbf{b} + e_2 + \lfloor q/2 \cdot \mu \rceil) = \mathbf{r}^\top \cdot \mathsf{pk} + \mathbf{e} + (0, \lfloor q/2 \cdot \mu \rceil)$. Frodo [18], an instantiation of [50], introduces a narrower error for more efficiency. Lizard [25] uses LWR-based encryption, which can be viewed as an LWE sample with deterministic rounding error: $\left(\left\lfloor \frac{p}{q} \cdot \mathbf{r}^\top \cdot \mathbf{A} \right\rceil, \left\lfloor \frac{p}{q} \cdot \mathbf{r}^\top \cdot \mathbf{b} + \frac{p}{2} \cdot \mu \right\rceil \right) = \frac{p}{q} \cdot (\mathbf{r}^\top \cdot \mathbf{A} + \mathbf{e}, \mathbf{r}^\top \cdot \mathbf{b} + \frac{q}{2} \cdot \mu + e)$, where the coefficients of $\mathbf{e}$ and e are in $\mathbb{Z} \cap (-\frac{q}{2p}, \frac{q}{2p}]$. The scaling factor reduces the ciphertext size and the running time since the encryption process skips a complex error sampling procedure.

More efficient approaches are usually based on the assumptions over structured lattices such as NewHope [6] over RLWE, RLizard [49] over RLWR, and Kyber [17] over MLWE assumptions. Additionally, Kyber uses a centered binomial distribution (CBD) as an MLWE error instead of a complex discrete Gaussian error, which makes the encryption much faster. Saber [28] expands the use of LWR also to the public key generation as $(\mathbf{A}, \mathbf{b} = \lfloor p/q \cdot \mathbf{A} \cdot \mathbf{s} \rceil \bmod p) \in \mathbb{Z}_q^{k \times \ell} \times \mathbb{Z}_p^k$ over module lattices. The extensions to rings and modules provide a wider message space, smaller sizes, and faster implementation based on the ring structure but with larger decryption failure probabilities.

Paper organization. The rest of the paper is organized as follows. Section 2 defines the notations and summarizes the formal definitions of key encapsulation mechanisms with the relevant works. In Section 3, we introduce the design choices of SMAUG. In Section 4, we introduce SMAUG and its security proofs. We provide concrete security analysis and the recommended parameter sets in Section 5. Lastly, we give the performance result with comparisons to recent KEM schemes and the implementation details in Section 6.

Code availability. We place all software, consisting of the constant-time C reference code of SMAUG and the Python scripts used for security estimation, into the public domain. They are available on the team SMAUG website: `https://kpqc.cryptolab.co.kr/smaug`.

2 Preliminaries

We start with the notations, formal definitions of PKE and KEM, and relevant works.

2.1 Notation

We denote matrices with bold 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 $\tilde{S}_\eta$ be a set of polynomials of degree less than n with coefficients in $[-\eta, \eta) \cap \mathbb{Z}$.

2.2 Lattice assumptions

We recall the lattice assumptions MLWE and MLWR in the structured Euclidean lattices, which security of SMAUG underlies.

Definition 1 (Decisional MLWE). *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-*$\mathsf{MLWE}_{n,q,k,\ell,\chi_s,\chi_e}$ *problem is*

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

Definition 2 (Decisional MLWR). *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-*$\mathsf{MLWR}_{n,p,q,k,\ell,\chi_r}$ *problem is*

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

2.3 Public key encryption and key encapsulation mechanism

Definition 3 (PKE). *A public key encryption scheme is a tuple of* PPT *algorithms* $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *with the following specifications:*

- KeyGen*: a probabilistic algorithm that outputs a public key* pk *and a secret key* sk*;*
- Enc*: a probabilistic algorithm that takes as input a public key* pk *and a message μ and outputs a ciphertext* ct*;*
- Dec*: a deterministic algorithm that takes as input a secret key* sk *and a ciphertext* ct *and outputs a message μ.*

Let $0 < \delta < 1$. We say that it is $(1-\delta)$-correct if for any $(\mathsf{pk}, \mathsf{sk})$ *generated from* KeyGen *and μ,*

$$Pr[\mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, \mu)) \neq \mu] \leq \delta,$$

where the probability is taken over the randomness of the encryption algorithm. We call the above probability decryption failure probability (DFP). In addition, we say that it is correct in the (Q)ROM if the probability is taken over the randomness of the (quantum) random oracle, modeling the hash function.

Definition 4 (KEM). *A key encapsulation mechanism scheme is a tuple of* PPT *algorithms* $(\mathsf{KeyGen}, \mathsf{Encap}, \mathsf{Decap})$ *with the following specifications:*

- KeyGen*: a probabilistic algorithm that outputs a public key* pk *and a secret key* sk*;*
- Encap*: a probabilistic algorithm that takes as input a public key* pk *and outputs a sharing key* K *and a ciphertext* ct*;*
- Decap*: a deterministic algorithm that takes input a secret key* sk *and a ciphertext* ct *and outputs a sharing key* K*.*

The correctness of KEM *is defined similarly to that of* PKE*.*

We give the advantage function with respect to the attacks against PKE, namely the indistinguishability under chosen plaintext attacks (IND-CPA).

Definition 5 (IND-CPA security of PKE). *For a (quantum) adversary* $\mathcal{A}$ *against a public key encryption scheme* $\mathsf{PKE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$*, we define the* IND-CPA *advantage of* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *as follows:*

$$\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{PKE}}(\mathcal{A}) = \left| \Pr_{(\mathsf{pk},\mathsf{sk})} \left[b = b' \;\middle|\; \begin{array}{c} (\mu_0, \mu_1, st) \leftarrow \mathcal{A}_1(\mathsf{pk});\ b \leftarrow \{0,1\}; \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, \mu_b);\ b' \leftarrow \mathcal{A}_2(\mathsf{pk}, \mathsf{ct}, st) \end{array} \right] - \frac{1}{2} \right|.$$

The probability is taken over the randomness of $\mathcal{A}$ *and* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$*.*

We then define two advantage functions with respect to the attacks against KEM, namely the indistinguishability under chosen plaintext attacks (IND-CPA) as in PKE and the indistinguishability under (adaptively) chosen ciphertext attacks (IND-CCA).

Definition 6 (IND-CPA and IND-CCA security of KEM). *For a (quantum) adversary* $\mathcal{A}$ *against a key encapsulation mechanism* $\mathsf{KEM} = (\mathsf{KeyGen}, \mathsf{Encap}, \mathsf{Decap})$*, we define the* IND-CPA *advantage of* $\mathcal{A}$ *as follows:*

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

The probability is taken over the randomness of $\mathcal{A}$ *and* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$*. The* IND-CCA *advantage of* $\mathcal{A}$ *is defined similarly except that the adversary can query* $\mathsf{Decap}(\mathsf{sk}, \cdot)$ *oracle on any ciphertext* $\mathsf{ct}' (\neq \mathsf{ct})$*.*

Finally, we define the (quantum) security of PKE and KEM.

Definition 7 ((Q)ROM security of PKE and KEM). *For* $T, \epsilon > 0$*, we say that a scheme* $\mathcal{S} \in \{\mathsf{PKE}, \mathsf{KEM}\}$ *is* (T, ϵ)-ATK *secure in the (Q)ROM if for any (quantum) adversary* $\mathcal{A}$ *with runtime* $\leq T$ *given classical access to* $\mathcal{O}$ *and (quantum) access to a random oracle* H*, it holds that* $\mathsf{Adv}^{\mathsf{ATK}}_{\mathcal{S}}(\mathcal{A}) < \epsilon$*, where*

$$\mathcal{O} = \begin{cases} \mathsf{Enc} & \text{if } \mathcal{S} = \mathsf{PKE} \text{ and } \mathsf{ATK} \in \{\mathsf{OW\text{-}CPA}, \mathsf{IND\text{-}CPA}\}, \\ \mathsf{Encap} & \text{if } \mathcal{S} = \mathsf{KEM} \text{ and } \mathsf{ATK} = \mathsf{IND\text{-}CPA}, \\ \mathsf{Encap}, \mathsf{Decap}(\mathsf{sk}, \cdot) & \text{if } \mathcal{S} = \mathsf{KEM} \text{ and } \mathsf{ATK} = \mathsf{IND\text{-}CCA}. \end{cases}$$

2.4 Fujisaki-Okamoto transform

Fujiskai and Okamoto proposed a novel generic transform [33, 34] that turns a weakly secure PKE scheme into a strongly secure PKE scheme in the random oracle model (ROM), and various variants have been proposed to deal with tightness, non-correct PKEs, and in the quantum setting, i.e., QROM. Here, we recall the FO transformation for KEM as introduced by Dent [29] and revisited by Hofheinz et al. [39] and Hövelmanns et al. [15, 40].

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. [15] allowing the KEMs constructed over non-perfect PKEs to have IND-CCA security:

Theorem 1 ([15], Theorem 1 & 2). *Let G and H be quantum-accessible random oracles, and the deterministic PKE is ϵ-injective. Then the advantage of* IND-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*

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

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

3 Design choices

In this section, we explain the design choices for SMAUG.

3.1 FO transform, $\mathsf{FO}_m^{\not\perp}$

We choose to construct SMAUG upon the FO transform with implicit rejection and without ciphertext contribution to the sharing key generation, say $\mathsf{FO}_m^{\not\perp}$. This choice makes the encapsulation and decapsulation algorithm efficient since the sharing key can be directly generated from a message. The public key is additionally fed into the hash function with the message to avoid multi-target decryption failure attacks.

3.2 MLWE public key and MLWR ciphertext

One of the core designs of SMAUG uses the MLWE hardness for its secret key security and MLWR hardness for its message security. This choice is adapted from Lizard and RLizard, which use LWE/LWR and RLWE/RLWR, respectively. The use of both LWE and LWR variant problems makes the conceptual security distinction between the secret key and the ephemeral sharing key: a more conservative secret key with more efficient en/decapsulations. This can be viewed as a trade-off between the "conservativity" and "efficiency." Combined with the sparse secret, bringing the LWE-based key generation to the LWR-based scheme enables balancing the speed and the DFP.

Public key. Public key of SMAUG consists of a vector $\mathbf{b}$ over a polynomial ring $\mathcal{R}_q$ and a matrix $\mathbf{A}$, which can be viewed as an MLWE sample,

$$(\mathbf{A}, \mathbf{b} = -\mathbf{A}^\top \mathbf{s} + \mathbf{e}) \in \mathcal{R}_q^{k \times k} \times \mathcal{R}_q^k,$$

where $\mathbf{s}$ is a ternary secret polynomial with hamming weight h_s and $\mathbf{e}$ is an error sampled from discrete Gaussian distribution with standard deviation σ. Since the matrix $\mathbf{A}$ is sampled uniformly, it can be stored and transmitted as a seed of an extendable output function (XOF).

Ciphertext. The ciphertext of SMAUG is a tuple of a vector $\mathbf{c}_1 \in \mathcal{R}_p^k$ and a polynomial $c_2 \in \mathcal{R}_{p'}$. The ciphertext is generated by multiplying a random vector $\mathbf{r}$ to the public key; then it is scaled and rounded as,

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

Along with the public key, it can be treated as an MLWR sample added by a scaled message as $(\mathbf{A}', \lfloor p/q \cdot \mathbf{A}' \cdot \mathbf{r} \rceil) + (0, \mu')$, where $\mathbf{A}'$ is a concatenated matrix of $\mathbf{A}$ and $\mathbf{b}^\top$.

The ciphertext can be further compressed by scaling the second component c_2 by p'/p, resulting in a shorter ciphertext but a larger error. We note that the public key can be compressed with the same technique. However, it introduces a more significant error, so we do not compress the public key in SMAUG.

3.3 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, respectively; and $\mathsf{dGaussian}_\sigma$ for sampling a discrete Gaussian error with standard deviation σ for MLWE sample.

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

Fig. 1: Uniform random matrix sampler, expandA.

Uniform random matrix sampler, expandA. We adopt the gen algorithm in Saber [58] for our uniform random matrix sampler expandA, given in Figure 1. This pseudorandom generator samples a public matrix $\mathbf{A}$ from uniformly random distribution over $\mathcal{R}_q^{k\times k}$.

Hamming weight sampler, HWT_h. The hamming weight sampler, HWT_h in Figure 2, is adapted from the SampleInBall algorithm in Dilithium [30], having a secret-independent running time. It samples a ternary polynomial vector having a hamming weight of h.

```
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(s)                                     ▷ Storing the indexes
```

Fig. 2: Hamming weight sampler, HWT_h.

Discrete Gaussian sampler, dGaussian. Karmakar et al. [45] suggested a constant-time discrete Gaussian sampling using the Knuth-Yao algorithm [47] and logic minimization. Motivated by this, we deployed the Quine-McCluskey method[5] 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. We give the algorithm with $\sigma = 1.0625$ in Figure 3, with $\sigma = 1.453713$ in Figure 4. It is easily parallelizable and also suitable for IoT devices as its memory requirment is low.

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

$\mathsf{dGaussian}_\sigma(x)$:

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: $s_0 \mathrel{+}= (\overline{x_2}x_3\overline{x_6}x_8) + (\overline{x_1}x_3\overline{x_6}x_8)$
5: $s_0 \mathrel{+}= (x_6x_7\overline{x_8}) + (\overline{x_5}x_6x_8) + (\overline{x_4}\overline{x_6}x_8) + (\overline{x_7}x_8)$
6: $s_1 = (x_1x_2x_4x_5x_7x_8) + (x_3x_4x_5x_7x_8) + (x_6x_7x_8)$
7: $s = (-1)^{x_9} \cdot s$ ▷ $\cdot$ is the arithmetic multiplication
8: **return** s

Fig. 3: Discrete Gaussian sampler with $\sigma = 1.0625$, $\mathsf{dGaussian}_\sigma$.

$\mathsf{dGaussian}_\sigma(x)$:

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: $s = (-1)^{x_{10}} \cdot s$ ▷ $\cdot$ is the arithmetic multiplication
10: **return** s

Fig. 4: Discrete Gaussian sampler with $\sigma = 1.453713$, $\mathsf{dGaussian}_\sigma$.

3.4 Polynomial multiplication using sparsity

SMAUG uses the power-of-two moduli to ease the correct scaling and roundings. However, this makes the polynomial multiplications hard to benefit from Number Theoretic Transform (NTT). As a result, we propose a new polynomial multiplication benefit from the sparsity, adapted from [1, 49]. Our new multiplication, given in Figure 5, 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.

4 The SMAUG

4.1 Specification of SMAUG.PKE

We now describe the public key encryption scheme SMAUG.PKE in Figure 6 with the following building blocks:

- Hash function H for generating the seeds $\mathsf{seed}_\mathbf{A}$ and $\mathsf{seed}_\mathsf{sk}$,

poly_mult_add$(a, b,$ neg_start$)$: $\triangleright\ a \in \mathcal{R}_q,\ b \in \mathcal{S}_\eta$

for i from 0 to neg_start - 1 **do**
    degree $= b[i]$
    for j from 0 to n **do**
        $a[\text{degree} + j] = a[\text{degree} + j] + a[j]$;
for i from neg_start to len(b) **do**
    degree $= b[i]$
    for j from 0 to n **do**
        $a[\text{degree} + j] = a[\text{degree} + j] - a[j]$;
for j from 0 to n **do**
    $a[j] = a[j] - a[n + j]$;
return a

Fig. 5: Polynomial multiplication using sparsity.

- Uniform random matrix sampler expandA for deriving $\mathbf{A}$ from $\mathsf{seed}_{\mathbf{A}}$,
- Discrete Gaussian sampler $\mathsf{dGaussian}_\sigma$ for deriving a MLWE error $\mathbf{e}$ 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. $\mathsf{seed}_{\mathbf{r}}$).

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 Figure 6. Let the moduli* t, p, p', *and* q *satisfy* $t \mid p \mid q$ *and* $t \mid p' \mid q$. *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 Figure 6 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

$\underline{\mathsf{KeyGen}(1^\lambda)}$:

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** $\mathsf{pk} = (\mathsf{seed}_{\mathbf{A}}, \mathbf{b})$, $\mathsf{sk} = \mathbf{s}$

$\underline{\mathsf{Enc}(\mathsf{pk}, \mu; \mathsf{seed}_{\mathbf{r}})}$: ▷ $\mathsf{pk} = (\mathsf{seed}_{\mathbf{A}}, \mathbf{b})$, $\mu \in \mathcal{R}_t$

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** $\mathsf{ct} = (\mathbf{c}_1, c_2)$

$\underline{\mathsf{Dec}(\mathsf{sk}, \mathbf{c})}$: ▷ $\mathsf{sk} = \mathbf{s}$, $\mathbf{c} = (\mathbf{c}_1, c_2)$

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** μ'

Fig. 6: Description of SMAUG.PKE

and the randomness $\mathbf{r}$ can be written as

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

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. □

4.2 Specification of SMAUG.KEM

We now introduce the key encapsulation mechanism SMAUG.KEM in Figure 7. 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 [15, 39, 40]. 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,

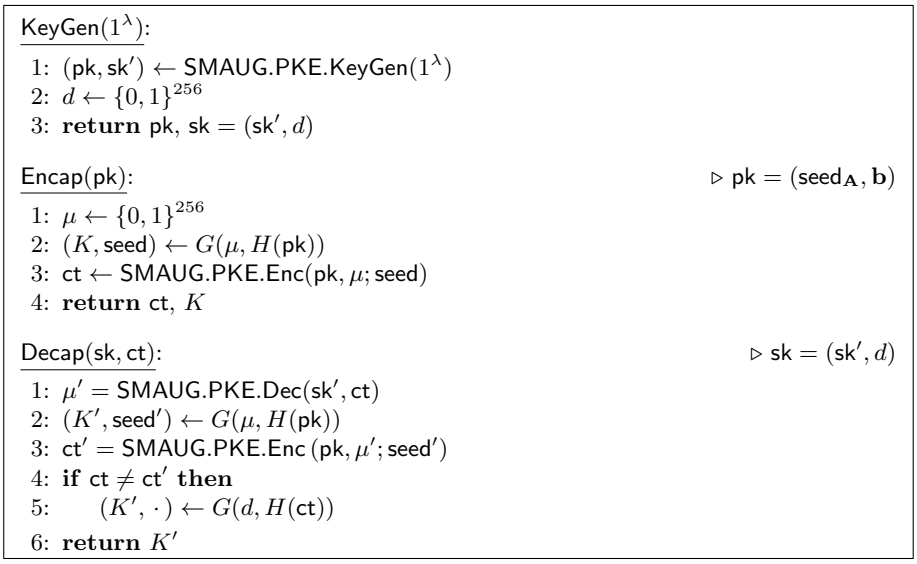

Fig. 7: Description of SMAUG.KEM

- hash function G for deriving a sharing key and a seed.

The Fujisaki-Okamoto transform depicted in Figure 7 has some differences from $\mathsf{FO}_m^{\not\perp}$ transform in [40] in 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 [42] 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 random number generator (TRNG) is recommended for sampling the message μ.

We now show the completeness of SMAUG.KEM based on the completeness of the underlying public key encryption scheme, SMAUG.PKE.

Theorem 3 (Completeness of SMAUG.KEM). *We borrow the notations and assumptions from Theorem 2 and Figure 7. Then* SMAUG.KEM *in Figure 7 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* $1-\delta$*.*

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. □

4.3 Security proof

When proving the security of the KEMs constructed using FO transform in the (Q)ROM, on typically relies on the generic reductions from one-wayness or IND-CPA security of the underlying PKE. In the ROM, SMAUG has a tight reduction from the IND-CPA security of the underlying PKE, SMAUG.KEM. However, as with other lattice-based constructions, the underlying PKE has a positive probability of decryption failures, which makes the generic reduction unapplicable [56] or non-tight [15,39,40] in the QROM. We, therefore, prove the IND-CCA2 security of SMAUG based on the non-tight QROM reduction of [15] as explained in Section 2, by proving the IND-CPA security of SMAUG.KEM.

Theorem 4 (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,*

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

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

- G_0: the genuine IND-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 $\mathsf{Adv}_0 = \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{SMAUG.PKE}}(\mathcal{A})$ 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 $\mathsf{Adv}^{\mathsf{MLWE}}_{n,q,k,k,\mathsf{HWT}_{h_{\hat{s}}},\mathsf{dGaussian}_\sigma}(\mathcal{B}_2)$, 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. □

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

5 Parameter selection and concrete security

In this section we first give a concret security analysis of SMAUG and the recommanded parameter sets.

5.1 Concrete security estimation

To estimate the concrete security of SMAUG, we exploit the best-known lattice attacks.

Core-SVP methodology. Most of the known attacks are essentially finding a nonzero short vector in Euclidean lattices, using the Block–Korkine–Zolotarev (BKZ) lattice reduction algorithm [22, 38, 57]. The time complexity is generally estimated as a core-SVP hardness, which was first introduced in [6] and used in the subsequent lattice-based schemes [5,17,30,32,58]. It is based on the best BKZ block size β which will find a good-quality lattice basis. The β-BKZ algorithm takes $\approx 2^{0.292\beta+o(\beta)}$ as estimated in [11] time with a classical solver, and $\approx 2^{0.257\beta+o(\beta)}$ as given in [20], in the quantum setting. We ignore the polynomial factors and the $o(\beta)$ terms in the exponent. We use the lattice estimator [3] to concretely estimate security of SMAUG.

Beyond Core-SVP methodology. We also analyze the cost of the attacks other than the lattice reduction attacks. Algebraic attacks like Arora-Ge attack and the variants [2, 9] using Gröbener's basis or Coded-BKW attacks [36, 46] should be also considered. Though, they have much higher attack costs than the previously introduced attacks, with significantly higher memory requirements. We use the lattice estimator [3] for estimating such attacks.

We also focus on the attacks that are not considered in the lattice-estimator [3], targeting sparse secrets, such as Meet-LWE [52] attack. Motivated from the Meet-in-the-Middle approach of Odlyzko's, it uses the representations of the ternary secret in additive shares. We use a python script to estimate the cost of Meet-LWE attack, following the original description in [52].

MLWE hardness. We estimated the cost of 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.

MLWR hardness. For 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 MLWE problem to the MLWR problem were given by Banerjee et al. [10] and were improved in [7, 8, 16]. 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.

5.2 Parameter sets

Table 3 shows the three parameter sets of SMAUG, with NIST's security levels 1, 3, and 5. SMAUG is parameterized by integers n, k, q, p, p', t, h_s and h_r, and the standard deviation $\sigma > 0$ 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.

The core-SVP hardness is estimated[6] via the lattice estimator [3] using the cost model "ADPS16" introduced in [6] and "MATZOV" [51], the smaller one

[6] There are some suspect on the unsubstantiated dual-sieve attacks which are assumming the flawed heuristic [31]. However, we hereby estimate security of SMAUG following the methods in Kyber, Saber, and Sable for a fair comparison.

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
Beyond core-SVP	144.7	202.0	274.6
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 3: The NIST security level, selected parameters, classical and quantum core-SVP hardness and security beyond core-SVP (see Section 5.1), decryption failure probability (in $\log_2$), and sizes (in bytes) of SMAUG.

among them. We assumed that the number of 1 is equal to the number of -1 for simplicity, which conservatively underestimates security.

The security beyond core-SVP is estimated via the lattice estimator [3] and the python script implementing the Meet-LWE attack cost estimation. It shows the lowest attack costs among the variants of coded-BKW, Arora-Ge, and Meet-LWE attacks, having memory requirement of 2^{130} to 2^{260}, at least.

We also give three additional parameter sets SMAUG-128*, SMAUG-192*, and SMAUG-256* having higher DFP and smaller ciphertexts. Most of the parameters are shared with the original ones, but with smaller $p' = 16$, 64, and 32, respectively. The DFP gap is not significant, hence these sets could be a good alternatives, as a size-DFP trad-off. We summarize their DFP and sizes in Table 4.

Parameters sets Security level	SMAUG-128* I	SMAUG-192* III	SMAUG-256* V
p'	16	64	32
DFP	-104.5	-131.1	-156.4
Secret key	176	236	218
Public key	672	1,088	1,792
Ciphertext	640	960	1,440

Table 4: Additional parameter sets with higher DFP and shorter ciphertexts.

6 Implementation

In this section, we give the parameter sets of SMAUG implementation the performance. We compare the sizes and the reported performance with prior work such as Kyber, Saber, and Sable. The constant-time reference implementation of SMAUG and the supporting scripts can be found on the team SMAUG website: `https://kpqc.cryptolab.co.kr/smaug`.

6.1 Performance

We instantiate the hash functions G, H and the extendable function XOF with the following symmetric primitives: G is instantiated with SHAKE256, H is instantiated with SHA3-256, and XOF is instantiated with SHAKE128.

The resulting performance of SMAUG is reported in Table 5. For a fair comparison, we also performed measurements on the same system with identical settings of the reference implementation of Kyber[7], Saber[8], and Sable[9].

6.2 Comparison to prior/related work

In this section, we compare the sizes, security, and performance of the recent lattice-based KEM schemes.

Using unstructured lattices. Most recent KEM schemes are designed over the ring lattices or module lattices. The schemes based on unstructured lattices such as Lizard [26] or FrodoKEM [18] have much larger sizes; e.g., Lizard has a 10,896-byte ciphertext, and FrodoKEM has a 9,752-byte ciphertext for the lowest level ($\approx$ 150 classical core-SVP hardness) whereas the security level-2 KEM RLizard [26] has a 4,144-byte ciphertext.

Ring-based KEMs and decryption failures. However, a straightforward adaptation of KEMs over the unstructured lattices to the ring variants introduces a much higher probability of decryption failure. To ensure security against the decryption failure attacks [41], a high enough decryption failure probability (DFP) is necessary; otherwise, it is vulnerable to the attacks such as failure boosting in [27]. As a result, the ring-based schemes should take large parameters like RLizard or additionally use an error correction code (ECC) to reduce the failure rate. However, using ECC makes it vulnerable to side-channel attacks (SCA) since masking ECC against probing the decryption failure is yet an open problem.

[7] `https://github.com/pq-crystals/kyber`, commit `518de24`.
[8] `https://github.com/KULeuven-COSIC/SABER`, commit `f7f39e4`.
[9] `https://github.com/josebmera/scabbard`, commit `4b2b5de`.

Schemes		Cycles KeyGen	Encap	Decap	Cycles (ratio) KeyGen	Encap	Decap
Kyber512	*med*	131,560	162,472	189,030	1.70	2.10	2.03
	ave	131,857	162,773	189,377	1.69	2.11	2.04
LightSaber	*med*	93,752	122,176	133,764	1.21	1.58	1.44
	ave	94,062	122,294	133,811	1.21	1.59	1.44
LightSable	*med*	85,274	114,822	128,990	1.10	1.48	1.39
	ave	85,462	114,914	129,231	1.10	1.49	1.39
SMAUG-128	*med*	77,220	77,370	92,916	1	1	1
	ave	77,940	77,063	93,046	1	1	1
Kyber768	*med*	214,160	251,308	285,378	1.38	1.84	1.75
	ave	214,544	251,686	285,657	1.38	1.84	1.73
Saber	*med*	187,022	224,686	239,590	1.21	1.64	1.47
	ave	187,309	224,777	239,685	1.21	1.64	1.45
Sable	*med*	170,400	211,290	237,024	1.10	1.55	1.45
	ave	170,601	211,332	237,158	1.10	1.55	1.44
SMAUG-192	*med*	154,862	136,616	163,354	1	1	1
	ave	155,311	136,702	164,782	1	1	1
Kyber1024	*med*	332,470	371,854	415,498	1.25	1.38	1.36
	ave	333,144	372,404	416,007	1.23	1.38	1.37
FireSaber	*med*	289,278	347,900	382,326	1.08	1.29	1.25
	ave	289,631	348,010	382,462	1.07	1.29	1.26
FireSable	*med*	275,156	337,322	371,486	1.03	1.25	1.22
	ave	275,491	337,329	371,814	1.02	1.25	1.22
SMAUG-256	*med*	266,704	270,123	305,452	1	1	1
	ave	270,123	270,672	304,292	1	1	1

Table 5: Median and average cycle counts of 1000 executions for Kyber, Saber, Sable, and SMAUG. Their relative comparison to SMAUG's is also given. Cycle counts are obtained on one core of an Intel Core i7-10700k, with TurboBoost and hyperthreading disabled.

Shared key entropy. Moreover, for the ring-based KEMs with NIST's security levels 1 and 3, achieving a message space of 256 bits with a compact ciphertext size and low decryption failure probability is challenging. The ring of dimension ≈ 500 is commonly used for the security level 1, and it is hard to balance between DFP and the message space size in the underlying PKE. As a result, Round5 [14] or Tiger [54] has only a 128-bit message, which gives a low sharing key entropy for latter quantum-secure protocols. Round5 achieves a short ciphertext size of 859 bytes with NIST's security level 3, restricting the message space size to 192 bits.

Module-based KEMs. The above limitations for the KEMs constructed over unstructured lattices or the ring-variants make the module-based KEMs an attractive candidate in their practical usage. The module-based KEMs have more scalable options when considering their security, sizes, and DFP. The NIST's

selection, Kyber [17], and the finalist Saber [58] have the module structure and achieve ciphertext sizes of 768 and 736 bytes for the security level 1.

	Sizes (bytes)			**Sizes** (ratio)			**Security**	
Schemes	**sk**	**pk**	**ct**	**sk**	**pk**	**ct**	**Classic.**	**DFP**
Kyber512	1,632	800	768	9.4	1.2	1.1	118	-139
LightSaber	832	672	736	4.8	1	1.1	118	-120
LightSable	800	608	672	4.6	0.9	1	114	-139
SMAUG-128	176	672	672	1	1	1	120	-120
Kyber768	2,400	1,184	1,088	10.4	1.1	1.1	183	-164
Saber	1,248	992	1,088	5.4	0.9	1.1	189	-136
Sable	1,152	896	1,024	5	0.8	1	185	-143
SMAUG-192	236	1,088	1,024	1	1	1	181	-136
Kyber1024	3,168	1,568	1,568	15.2	0.9	1.1	256	-174
FireSaber	1,664	1,312	1,472	8	0.7	1	260	-165
FireSable	1,632	1,312	1,376	7.8	0.7	0.9	223	-208
SMAUG-256	218	1,792	1,472	1	1	1	264	-167

Table 6: Comparison of Kyber, Saber, Sable, and SMAUG. Sizes are given in bytes and the ratios are also given relative to the sizes of SMAUG. Security is provided in the classical core-SVP hardness with DFP (in logarithm base two).

Kyber and SMAUG. As shown in Tables 5 and 6, SMAUG outperforms Kyber in every aspect except for the DFP and the public key size in level 5. Compared to Kyber-512 [17], SMAUG-128 has 16% and 12% smaller public key and ciphertext, respectively. The secret key size of SMAUG is significantly smaller than Kyber's. It is tiny and ready to use[10], which enables efficient management of secure zone in restricted IoT devices. With high enough security and low enough decryption failure probability, SMAUG-128 further achieves 110% and 103% speed up in encapsulation and decapsulation.

Similar phenomena are observed in the security levels 3 and 5, except that the public key size of Kyber is shorter than SMAUG's in the level 5. The speed-ups also decrease in higher security parameters.

Saber, Sable, and SMAUG. Compared to Saber, one of NIST's round 3 finalists, SMAUG-128 has 9% smaller ciphertext and the same public key size than LightSaber. The secret key is significantly smaller, with a 58% and 44% speed up in encapsulation and decapsulation, respectively. compared to Sable, an efficient variant of Saber, SMAUG-128 has the same ciphertext size but a larger public key size. This is since we use small modulus and sparse secret, which allows

[10] Most of the KEMs can store a secret key as a seed, having 32 bytes, and Saber can also compress the secret to 256 to 384 bytes. However, this gives an additional chance to the side-channel attackers during the decapsulation process.

SMAUG-128 to achieve 48% and 39% faster encapsulation and decapsulation, samller secret key, and a bit higher security. Thus, it can be seen as a trade-off between smaller public key versus faster running time.

Again, similar phenomena are observed in NIST's security levels 3 and 5. SMAUG provides a trade-off with Saber and Sable, between the public key size versus secret key size and running time. In NIST's level 3, the size of ciphertext of SMAUG-192 is smaller than Saber, and is the same with Sable. The encapsulation and decapsulation speed outperforms by 44% to 64% with much smaller, ready-to-use secret key.

In the scurity level 5, we first note that FireSable has a classical core-SVP hardness of 223, which is much lower than 260. It achieves smaller public key and ciphertext than SMAUG-256, but still with slower speeds. In the strongest security level, SMAUG-256 has the same cihertext size with FireSaber, and a similar trade-off is observed.

6.3 Security against physical attacks

We consider the security against physical attacks based on the profiled Differential Power Analysis (DPA). In particular, the key generation and the encapsulation can be profiled with the Simple Power Analysis (SPA) since they are executed once or without a secret key. For the decapsulation, however, due to the existence of "re-encryption", multi-trace attacks are possible. As SMAUG and Kyber, or SMAUG and Saber share many aspects in design, we can follow the recent works masking Kyber and Saber [12, 19], adding SCA countermeasure. For our new sampler dGaussian, as it can be expressed with bit operations, it is easy to add the SCA countermeasures such as boolean masking. For the fixed hamming weight sampler, Krausz et al. [48] have recently proposed masking methods; however, the lack of efficiency lets us put it as a future work. The new multiplication method can be suffered from attacks focusing on the memory access patterns; however, it can be masked using coefficient-wise shuffling. We set the detailed analysis and the masked implementation as future works.

Acknowledgments. Part of this work was done while MinJune Yi was in CryptoLab Inc.

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`
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., Avanzi, R., Bos, J., Ducas, L., de la Piedra, A., Pöppelmann, T., Schwabe, P., Stebila, D.: Algorithm specifications and supporting documentation Version 1.1
5. 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
6. Alkim, E., Ducas, L., Pöppelmann, T., Schwabe, P.: Post-quantum key exchange - A new hope. In: Holz, T., Savage, S. (eds.) USENIX Security 2016. pp. 327–343. USENIX Association (Aug 2016)
7. 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
8. 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)
9. 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)
10. 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)
11. 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
12. Beirendonck, M.V., D'anvers, J.P., Karmakar, A., Balasch, J., Verbauwhede, I.: A side-channel-resistant implementation of saber. J. Emerg. Technol. Comput. Syst. **17**(2) (apr 2021). https://doi.org/10.1145/3429983, https://doi.org/10.1145/3429983
13. Bernstein, D.J., Chuengsatiansup, C., Lange, T., Van Vredendaal, C.: Ntru prime. IACR Cryptol. ePrint Arch. **2016**, 461 (2016)
14. Bhattacharya, S., Garcia-Morchon, O., Laarhoven, T., Rietman, R., Saarinen, M.J.O., Tolhuizen, L., Zhang, Z.: Round5: Kem and pke based on glwr. Cryptology ePrint Archive, Paper 2018/725 (2018), https://eprint.iacr.org/2018/725, https://eprint.iacr.org/2018/725
15. Bindel, N., Hamburg, M., Hövelmanns, K., Hülsing, A., Persichetti, E.: Tighter proofs of CCA security in the quantum random oracle model. In: Hofheinz, D., Rosen, A. (eds.) TCC 2019, Part II. LNCS, vol. 11892, pp. 61–90. Springer, Heidelberg (Dec 2019). https://doi.org/10.1007/978-3-030-36033-7_3
16. 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)
17. 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)
18. Bos, J.W., Costello, C., Ducas, L., Mironov, I., Naehrig, M., Nikolaenko, V., Raghunathan, A., Stebila, D.: Frodo: Take off the ring! Practical, quantum-secure key exchange from LWE. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers,

A.C., Halevi, S. (eds.) ACM CCS 2016. pp. 1006–1018. ACM Press (Oct 2016). https://doi.org/10.1145/2976749.2978425
19. Bos, J.W., Gourjon, M., Renes, J., Schneider, T., van Vredendaal, C.: Masking kyber: First- and higher-order implementations. IACR TCHES **2021**(4), 173–214 (2021). https://doi.org/10.46586/tches.v2021.i4.173-214, https://tches.iacr.org/index.php/TCHES/article/view/9064
20. Chailloux, A., Loyer, J.: Lattice sieving via quantum random walks. In: Tibouchi, M., Wang, H. (eds.) Advances in Cryptology - ASIACRYPT. pp. 63–91 (2021)
21. Chen, C., Danba, O., Hoffstein, J., Hülsing, A., Rijneveld, J., Schanck, J.M., Schwabe, P., Whyte, W., Zhang, Z.: Algorithm specifications and supporting documentation. Brown University and Onboard security company, Wilmington USA (2019)
22. Chen, Y., Nguyen, P.Q.: BKZ 2.0: Better lattice security estimates. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 1–20. Springer, Heidelberg (Dec 2011). https://doi.org/10.1007/978-3-642-25385-0_1
23. Cheon, J.H., Han, K., Kim, A., Kim, M., Song, Y.: Bootstrapping for approximate homomorphic encryption. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part I. LNCS, vol. 10820, pp. 360–384. Springer, Heidelberg (Apr / May 2018). https://doi.org/10.1007/978-3-319-78381-9_14
24. Cheon, J.H., Han, K., Kim, J., Lee, C., Son, Y.: A practical post-quantum public-key cryptosystem based on spLWE. In: Hong, S., Park, J.H. (eds.) ICISC 16. LNCS, vol. 10157, pp. 51–74. Springer, Heidelberg (Nov / Dec 2017). https://doi.org/10.1007/978-3-319-53177-9_3
25. 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: Catalano, D., De Prisco, R. (eds.) SCN 18. LNCS, vol. 11035, pp. 160–177. Springer, Heidelberg (Sep 2018). https://doi.org/10.1007/978-3-319-98113-0_9
26. Cheon, J.H., Park, S., Lee, J., Kim, D., Song, Y., Hong, S., Kim, D., Kim, J., Hong, S.M., Yun, A., et al.: Lizard public key encryption. NIST PQC Round **1**, 4 (2018)
27. 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)
28. D'Anvers, J.P., Karmakar, A., Roy, S.S., Vercauteren, F.: Saber: Module-LWR based key exchange, CPA-secure encryption and CCA-secure KEM. In: Joux, A., Nitaj, A., Rachidi, T. (eds.) AFRICACRYPT 18. LNCS, vol. 10831, pp. 282–305. Springer, Heidelberg (May 2018). https://doi.org/10.1007/978-3-319-89339-6_16
29. Dent, A.W.: A designer's guide to kems. In: Cryptography and Coding: 9th IMA International Conference, Cirencester, UK, December 16-18, 2003. Proceedings 9. pp. 133–151. Springer (2003)
30. Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schwabe, P., Seiler, G., Stehlé, D.: CRYSTALS-Dilithium: A lattice-based digital signature scheme. IACR TCHES **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
31. Ducas, L., Pulles, L.: Does the dual-sieve attack on learning with errors even work? Cryptology ePrint Archive, Paper 2023/302 (2023), https://eprint.iacr.org/2023/302, https://eprint.iacr.org/2023/302

32. 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)
33. Fujisaki, E., Okamoto, T.: Secure integration of asymmetric and symmetric encryption schemes. In: Wiener, M.J. (ed.) CRYPTO'99. LNCS, vol. 1666, pp. 537–554. Springer, Heidelberg (Aug 1999). https://doi.org/10.1007/3-540-48405-1_34
34. Fujisaki, E., Okamoto, T.: Secure integration of asymmetric and symmetric encryption schemes. Journal of Cryptology **26**(1), 80–101 (Jan 2013). https://doi.org/10.1007/s00145-011-9114-1
35. 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)
36. Guo, Q., Johansson, T., Stankovski, P.: Coded-bkw: Solving lwe using lattice codes. In: Annual Cryptology Conference. pp. 23–42. Springer (2015)
37. Halevi, S., Shoup, V.: Bootstrapping for HElib. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part I. LNCS, vol. 9056, pp. 641–670. Springer, Heidelberg (Apr 2015). https://doi.org/10.1007/978-3-662-46800-5_25
38. 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)
39. Hofheinz, D., Hövelmanns, K., Kiltz, E.: A modular analysis of the Fujisaki-Okamoto transformation. In: Kalai, Y., Reyzin, L. (eds.) TCC 2017, Part I. LNCS, vol. 10677, pp. 341–371. Springer, Heidelberg (Nov 2017). https://doi.org/10.1007/978-3-319-70500-2_12
40. Hövelmanns, K., Kiltz, E., Schäge, S., Unruh, D.: Generic authenticated key exchange in the quantum random oracle model. In: Kiayias, A., Kohlweiss, M., Wallden, P., Zikas, V. (eds.) PKC 2020, Part II. LNCS, vol. 12111, pp. 389–422. Springer, Heidelberg (May 2020). https://doi.org/10.1007/978-3-030-45388-6_14
41. 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)
42. 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
43. Jin, Z., Zhao, Y.: Optimal key consensus in presence of noise. Cryptology ePrint Archive, Paper 2017/1058 (2017), https://eprint.iacr.org/2017/1058, https://eprint.iacr.org/2017/1058
44. Jung, C.G., Lee, J., Ju, Y., Kwon, Y.B., Kim, S.W., Paek, Y.: Lizarmong: Excellent key encapsulation mechanism based on rlwe and rlwr. In: Seo, J.H. (ed.) Information Security and Cryptology – ICISC 2019. pp. 208–224. Springer International Publishing, Cham (2020)
45. 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)

46. 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)
47. Knuth, D., Yao, A.: Algorithms and Complexity: New Directions and Recent Results, chap. The complexity of nonuniform random number generation. Academic Press (1976)
48. Krausz, M., Land, G., Richter-Brockmann, J., Güneysu, T.: A holistic approach towards side-channel secure fixed-weight polynomial sampling. In: Public-Key Cryptography–PKC 2023: 26th IACR International Conference on Practice and Theory of Public-Key Cryptography, Atlanta, GA, USA, May 7–10, 2023, Proceedings, Part II. pp. 94–124. Springer (2023)
49. 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)
50. Lindner, R., Peikert, C.: Better key sizes (and attacks) for LWE-based encryption. In: Kiayias, A. (ed.) CT-RSA 2011. LNCS, vol. 6558, pp. 319–339. Springer, Heidelberg (Feb 2011). https://doi.org/10.1007/978-3-642-19074-2_21
51. 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
52. May, A.: How to meet ternary LWE keys. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part II. LNCS, vol. 12826, pp. 701–731. Springer, Heidelberg, Virtual Event (Aug 2021). https://doi.org/10.1007/978-3-030-84245-1_24
53. Mera, J.M.B., Karmakar, A., Kundu, S., Verbauwhede, I.: Scabbard: a suite of efficient learning with rounding key-encapsulation mechanisms. IACR TCHES **2021**(4), 474–509 (2021). https://doi.org/10.46586/tches.v2021.i4.474-509, https://tches.iacr.org/index.php/TCHES/article/view/9073
54. Park, S., Jung, C.G., Park, A., Choi, J., Kang, H.: Tiger: Tiny bandwidth key encapsulation mechanism for easy migration based on rlwe(r). Cryptology ePrint Archive, Paper 2022/1651 (2022), https://eprint.iacr.org/2022/1651, https://eprint.iacr.org/2022/1651
55. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: Gabow, H.N., Fagin, R. (eds.) 37th ACM STOC. pp. 84–93. ACM Press (May 2005). https://doi.org/10.1145/1060590.1060603
56. Saito, T., Xagawa, K., Yamakawa, T.: Tightly-secure key-encapsulation mechanism in the quantum random oracle model. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part III. LNCS, vol. 10822, pp. 520–551. Springer, Heidelberg (Apr / May 2018). https://doi.org/10.1007/978-3-319-78372-7_17
57. Schnorr, C.P., Euchner, M.: Lattice basis reduction: Improved practical algorithms and solving subset sum problems. Mathematical programming **66**(1), 181–199 (1994)
58. Vercauteren, I.F., Sinha Roy, S., D'Anvers, J.P., Karmakar, A.: Saber: Mod-lwr based kem (round 3 submission)

A Comparison result

Focusing on the sizes and the decryption failure probability, we compare the recent KEM schemes having ehough core-SVP hardness, i.e., roughly NIST's security levels 1, 3, and 5, in Table 7. Among the recent KEM schemes summarized in Table 7, SMAUG has the most petite ciphertext and secret key sizes among

the KEMs with a low enough DFP; without ECC; and enough entropy for the shared key.

The sizes are given in bytes and the decryption failure probability (DFP) is given in $-\log_2(\text{prob})$. Some of the schemes have not specified on their secret key sizes, so we left them as a blank. "∞" implies that it uses a perfectly correct PKE scheme as a building block. Security (simply, Sec.) is given in the classical core-SVP hardness reported/claimed in the speciification/paper of each scheme. The shared key entropy (simply, $|K|$) are given in bits if they are less than 256 bits. The required assumptions are given as "A+B" (A for key generation, B for ecapsulation) or "A", if A=B. The last column shows whether the scheme uses an error correction code (ECC) to pull down the DFP or not.

We note that in each security level, the parameter sets used for comparison are (if there are multiple options) chosen among the proposed parameter sets, having the most petite sizes and decryption failure probability. For Round5, all the parameter sets "R5ND_{1,3,5}KEM_{0,5}c" are used. For NTRU, the parameter sets "ntruhps 2048509" and "ntruhps4096821" are used. For RLizard, the parameter sets "RING _CATEGORY{1,5}" and "RING_CATEGORY3_N10 24" are used.

[11] As estimated in our script implementing the cost calculation of Meet-LWE attack. The claimed security was 200.

[12] See footnote 11. The claimed security was 256.

Scheme	sk	pk	ct ↑	DFP	Sec.	$\lvert K\rvert$	Assumption	no ECC
LizarMong [44]	640	544	544	179	133		RLWE+RLWR	✗
Round5 [14]		445	549	88	128	128	GLWR	✗
Sable [53]	800	608	672	139	114		MLWR	✓
SMAUG (ours)	176	672	672	120	120		MLWE+MLWR	✓
Round5 [14]		634	682	65	128	128	GLWR	✓
NTRU [21]	699	935	699	∞	106		NTRU	✓
Saber [58]	832	672	736	120	118		MLWR	✓
Kyber [17]	1632	800	768	139	118		MLWE	✓
Tiger [54]	528	480	768	128	130	128	RLWR+RLWE	✗
RLizard [26]	385	4096	2080	188	147		RLWE+RLWR	✓

(a) Lattice-based KEMs with ≈ NIST's security level I (120).

Scheme	sk	pk	ct ↑	DFP	Sec.	$\lvert K\rvert$	Assumption	no ECC
Round5 [14]		780	859	117	193	192	GLWR	✗
Round5 [14]		909	981	71	192	192	GLWR	✓
Sable [53]	1152	896	1024	143	185		MLWR	✓
SMAUG (ours)	236	1088	1024	136	181		MLWE+MLWR	✓
Tiger [54]	1056	800	1024	154 [11]	161		RLWR+RLWE	✗
Saber [58]	1248	992	1088	136	189		MLWR	✓
Kyber [17]	2400	1184	1088	164	183		MLWE	✓
NTRU [21]	1230	1590	1230	∞	178		NTRU	✓
RLizard [26]	641	4096	4144	245	195		RLWE+RLWR	✓

(b) Lattice-based KEMs with ≈ NIST's security level III (180).

Scheme	sk	pk	ct ↑	DFP	Sec.	$\lvert K\rvert$	Assumption	no ECC
Round5 [14]		972	1063	64	256		GLWR	✗
LizarMong [44]	1280	1056	1088	302 [12]	226		RLWE+RLWR	✗
Tiger [54]	1056	928	1152	200	263		RLWR+RLWE	✗
Round5 [14]		1178	1274	64	256		GLWR	✓
Saber [58]	1664	1312	1472	165	260		MLWR	✓
SMAUG (ours)	218	1792	1472	167	264		MLWE+MLWR	✓
Kyber [17]	3168	1568	1568	174	256		MLWE	✓
AKCN-SEC [43]		2240	1792	70	255		RLWE	✗
OKCN-SEC [43]		2336	1792	70	255		RLWE	✗
NewHope [4]	3680	1824	2208	216	257		RLWE	✓
RLizard [26]	769	8192	8256	305	318		RLWE+RLWR	✓

(c) Lattice-based KEMs with ≈ NIST's security level V (260).

Table 7: Comparison of lattice-based KEM schemes with comparable security and sizes.