LAC

Lattice-based **C**ryptosystems

Principal Submitter: Xianhui Lu[1,2], +86-15010131069, luxianhui@iie.ac.cn

Original Submitters: Yamin Liu[1,2], liuyamin@iie.ac.cn

Dingding Jia[1,2], jiadingding@iie.ac.cn

Haiyang Xue[1,2], xuehaiyang@iie.ac.cn

Jingnan He[1,2], hejingnan@iie.ac.cn

Zhenfei Zhang[3], zhenfei@algorand.com

Round 2 Contributors: Zhe Liu[4], zhe.liu@nuaa.edu.cn

Hao Yang[4], anonymous@nuaa.edu.cn

Bao Li[1,2], libao@iie.ac.cn

Kunpeng Wang[1,2], wangkunpeng@iie.ac.cn

Organizations: 1. Data Assurance and Communications Security Center, CAS

2. State Key Laboratory of Information Security, IIE, CAS

3. Algorand

4. Nanjing University of Aeronautics and Astronautics

Owner: Xianhui Lu

Postal Address: No. 89A, Minzhuang Road, Beijing 100093, China

Alternative Contact: hejingnan@iie.ac.cn, +86-18519196307

Signature:

Table of Contents

1 Introduction 1
2 Preliminaries 2
2.1 Mathematical Notations 2
2.2 Lattices and Hard Problems 3
2.3 Learning with Errors (over Rings) 4
2.4 Distributions and Random Sampling 4
3 Design Rationale 6
4 Description of the Cryptosystems 7
4.1 LAC.CPA 7
4.2 LAC.CCA 8
4.3 LAC.KE 9
4.4 LAC.AKE 9
5 Parameter Choices 10
5.1 Modulus 10
5.2 The Errors and Secrets Distribution of Ring-LWE 11
5.3 Decryption Errors 12
5.4 The Error Correction Code 13
5.5 Recommended Parameter Categories 13
6 Security Analysis 14
6.1 Generic Attacks 14
Primal attack. 14
Dual attack. 15
Security Estimates. 15
6.2 Dedicated Attacks 15
Subfield Attacks. 15
High Hamming Weight Attack. 16
7 Implementation and Performance 18
7.1 Polynomial Multiplication 18
7.2 Error Correction 18
7.3 Computational Performance of the Optimized Version 19
7.4 Computational Performance of the AVX2 Based Version 19
7.5 Key and Ciphertext Size 20
7.6 Memory Cost of Error Correction Code 20
8 Known Answer Test Values 20
8.1 LAC.CPA 21
8.2 LAC.CCA 21
8.3 LAC.KE 21
8.4 LAC.AKE 22
9 The Advantages and Limitations 22
9.1 Advantages 22
9.2 Limitations 22

List of Figures

1 LAC: **La**ttice-based **C**ryptosystems 1
2 LAC.KE: the unauthenticated lattice-based key exchange protocol 9
3 LAC.AKE: the authenticated lattice-based key exchange protocol 10

List of Tables

1 Recommended parameter of LAC.CPA 13
2 Concrete security of LAC 14
3 Decryption error rate of high Hamming attack 18
4 Performance of LAC.CPA 19
5 Performance of LAC.CCA 19
6 Performance of LAC.CPA with AVX2 19
7 Performance of LAC.CCA with AVX2 20
8 Key and Ciphertext size of LAC.CPA and LAC.CCA 20
9 Message size of LAC.KE and LAC.AKE 20
10 Memory cost of error correction code 20

1 Introduction

In this document we introduce LAC (**La**ttice-based **C**ryptosystems). It consists of four public key cryptographic primitives based on the learning with errors assumption over rings:

- LAC.CPA: an IND-CPA secure public key encryption scheme;
- LAC.CCA: an IND-CCA secure key encapsulation mechanism, which is obtained by applying a variant of the FO transformation [27,30,32] to LAC.CPA;
- LAC.KE: a passively secure key exchange protocol which is directly converted from LAC.CPA;
- LAC.AKE: an authenticated key exchange protocol which is obtained by applying a generic FSXY transformation [25,26] to LAC.CCA and LAC.CPA.

Relations of the four public key cryptographic primitives are shown in Fig. 1.

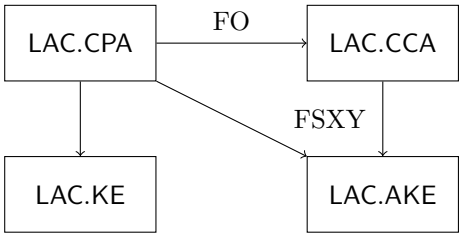

Fig. 1. LAC: **La**ttice-based **C**ryptosystems

Versions and modifications.

Compared to the version submitted to Round 1 of NIST PQC standardization process (hereafter referred to as Version 1), the main modifications in this version (referred to as Version 2) are:

1. **Message Space/Session Key Size:** In Version 1, the message spaces and session key sizes for security levels I, III and V are 256 bits, 384 bits and 512 bits. In Version 2, we set the message space and session key size to 256 bits for all security levels. This modification is to simplify the comparison of LAC with other schemes.
2. **Noise/Secret Distribution:** In Version 1, the secret and error vectors are chosen from standard centered binomial distributions. In Version 2, $\boldsymbol{s}, \boldsymbol{e}, \boldsymbol{r}, \boldsymbol{e}_1$ are chosen from fixed hamming weight centered binomial distributions. The hamming weight is fixed to the expectation of Version 1's binomial distribution. This modification is to make LAC immune to the high hamming weight CCA attacks, in which the adversary exploits high hamming weight secrets/errors through pre-computation, and causes higher-than-normal decryption error rates.
   Remark 1: Switch to a fixed hamming weight distribution insignificantly reduces the entropy. We use $n = 512$ as an example. The entropy of $\boldsymbol{s}$ from standard centered binomial distribution, with a standard deviation of $1/\sqrt{2}$, is 768. When the hamming weight of $\boldsymbol{s}$ is fixed to 256, with even number of ones and minus-ones, its entropy is reduced to approximately $\log\left(\binom{512}{128} \cdot \binom{384}{128}\right) \approx 758$. Later on we will show that such a reduction in entropy will not affect the security reduction or the concrete security evaluation of LAC.
   Remark 2: An alternative, and perhaps more straightforward way to make LAC robust against high hamming weight CCA attacks is to increase the capability of BCH codes, at a cost of reduced efficiency due to heavy error correction. Our analysis shows that, with this solution we will need to correct 30 errors. This incurs a cost of 27 microseconds. In comparison, by switching to fixed hamming weight, we only need to correct 16 errors, and the time cost is 12 microseconds.
3. **Error Correction Code:** In Version 1, only BCH error correction code was adopted. In round 2, for LAC-256, additional D2 code are used, while LAC-128/192 remain unchanged.

The parameter sets for BCH code are also updated. We use [511, 256, 33] for LAC-128/256, and [511, 256, 17] for LAC-192. This modification is to make the error correction capability to match the new message space/session key size and secret/error distribution of LAC.

4. **Ciphertext Compression:** In Version 2, we remove the lower four bits of $\boldsymbol{c}_2$ during the transmission. Omitting some bits is a popular compression technique in lattice cryptography. Note that compressions on $\boldsymbol{b}$ and $\boldsymbol{c}_1$ may affect the security reduction, therefore, in LAC we only compress $\boldsymbol{c}_2$. This will not affect the security reduction. The aim of this modification is to make LAC more compact.
5. **Message Length Counter:** In Version 1, we used a counter during the decryption to get the message length. This counter is removed in Version 2. The length of the message is implicitly obtained from the length of the ciphertext. Briefly, the dimension of $\boldsymbol{c}_2$ equals to the sum of the length of the message and the length of error correction bits of BCH. The aim of this modification is to make LAC simpler and more compact.
6. **Constant Time Decoding of BCH:** In Version 1, we adopted the code of BCH from https://github.com/jkent/python-bchlib/tree/master/bchlib. In Version 2, we implemented a constant time BCH based on this generic implementation. The aim of this modification is to make LAC immune to timing attacks.

Organization. The document is organized as follows. Firstly, in Section 2, we define the mathematical notations and operations, and introduce the background on lattices and error-correction codes. Then in Section 3, the overall design rationale is explained. Specifications and parameter are given in Section 4 and 5, respectively. In Section 6, the formal and concrete security analysis are presented. Section 7 describes implementation and performance analysis. Known answer test values (KATs) are given in Section 8. Finally, the pros and cons of the cryptosystems are discussed in Section 9.

2 Preliminaries

In this section we first define several mathematical notations, then introduce backgrounds on lattices and error correction codes.

2.1 Mathematical Notations

Vectors and Matrices. Vectors are denoted by bold lower-case characters, such as $\boldsymbol{a}$, and $\boldsymbol{a}^t$ denotes the transposition of $\boldsymbol{a}$.

An m-dimensional vector $\boldsymbol{a} = (a_0, \cdots, a_{m-1})$, where the a_i's are the components of $\boldsymbol{a}$ for $0 \leq i < m$.

For a scalar s and an m-dimensional vector $\boldsymbol{a}$, $s \cdot \boldsymbol{a}$ denotes that each component of $\boldsymbol{a}$ is multiplied by s, i.e., $s \cdot \boldsymbol{a} = (s \cdot a_0, \cdots, s \cdot a_{m-1})$.

For an m-dimensional vector $\boldsymbol{a} = (a_0, \cdots, a_{m-1})$ and a non-negative integer $l \leq m$, define $(\boldsymbol{a})_l = (a_0, \cdots, a_{l-1})$.

Vectors of the same dimension can add component-wise, e.g., for two m-dimensional vectors $\boldsymbol{a} = (a_0, \cdots, a_{m-1})$ and $\boldsymbol{b} = (b_0, \cdots, b_{m-1})$, $\boldsymbol{a} + \boldsymbol{b} = (a_0 + b_0, \cdots, a_{m-1} + b_{m-1})$.

Matrices are denoted by upper-case characters, such as $\boldsymbol{A}$, and $\boldsymbol{A}^t$ denotes the transposition of $\boldsymbol{A}$.

Norms. The length of vectors is measured with norms. For an m-dimensional vector $\boldsymbol{x} = (x_0, x_1, ..., x_{m-1})$, its l_1-norm is defined as $\|\boldsymbol{x}\|_1 = \sum_{i=0}^{m-1} |x_i|$; the l_2-norm, also known as the Euclidean norm, is defined as $\|\boldsymbol{x}\|_2 = \sqrt{\sum_{i=0}^{m-1} x_i^2}$, or solely denoted as $\|\boldsymbol{x}\|$; the infinity norm is defined as $\|\boldsymbol{x}\|_\infty = \max |x_i|$. The length of a matrix is the norm of its longest column vector, e.g., $\|\boldsymbol{X}\| = \max \|\boldsymbol{x}_i\|$.

Arrows. For a set S, $x \xleftarrow{\$} S$ denotes that an element x is chosen from S uniformly at random. For a distribution D, $x \xleftarrow{\$} D$ denotes that a random variable x is sampled according to the distribution D. For a randomized algorithm A, $y \xleftarrow{\$} \mathsf{A}(x)$ denotes that y is assigned the output of A on input x; if the algorithm A is deterministic, we just write $y \leftarrow \mathsf{A}(x)$.

Algebraic structures. Let $\mathbb{R}$ be real numbers, $\mathbb{Q}$ be rational numbers, and $\mathbb{Z}$ be integers. For an integer $q \geq 1$, let $\mathbb{Z}_q$ be the residue class ring modulo q and $\mathbb{Z}_q = \{0, \cdots, q-1\}$. Define the ring of integer polynomials modulo $x^n + 1$ as $R = \mathbb{Z}[x]/(x^n + 1)$ for an integer $n \geq 1$, and the ring $R_q = \mathbb{Z}_q[x]/(x^n + 1)$ denotes the polynomial ring modulo $x^n + 1$ where the coefficients are from $\mathbb{Z}_q$. The addition and multiplication of the elements in R_q are operated according to those of polynomials.

String operations. For two bit-strings $s_1, s_2 \in \{0,1\}^*$, denote their concatenation as $s_1 \| s_2$. The length of a bit-string s is denoted as $|s|$. Sometimes, we treat bit-strings of length m as m-dimensional vectors. We use ϵ to denote an empty string.

2.2 Lattices and Hard Problems

Lattices and Dual Lattices. A full-rank m-dimensional lattice Λ generated by a basis $\boldsymbol{B} = \{\boldsymbol{b}_1, ..., \boldsymbol{b}_m\} \in \mathbb{R}^{m \times m}$ can be defined as

$$\Lambda = \mathcal{L}(\boldsymbol{B}) = \{\boldsymbol{Bx} : \boldsymbol{x} \in \mathbb{Z}^m\},$$

where $\boldsymbol{b}_1, ..., \boldsymbol{b}_m$ are linearly independent vectors.

For a full-rank m-dimensional lattice Λ, its dual lattice is defined as

$$\Lambda^* = \{\boldsymbol{y} \in \mathbb{R}^m : \forall \boldsymbol{x} \in \Lambda, \langle \boldsymbol{x}, \boldsymbol{y} \rangle \in \mathbb{Z}\}.$$

***q*-ary Lattices.** As with most lattice-based cryptographic applications, we deal with two special full-rank integer lattices known as q-ary integer lattices. For positive integers n, m, q and a random integer matrix $\boldsymbol{A} \in \mathbb{Z}_q^{m \times n}$, define the following full-rank m-dimensional integer q-ary lattices:

$$\Lambda(\boldsymbol{A}) = \{\boldsymbol{z} \in \mathbb{Z}^m : \exists \boldsymbol{s} \in \mathbb{Z}_q^n \text{ s.t. } \boldsymbol{z} = \boldsymbol{As} \bmod q\};$$

$$\Lambda^{\perp}(\boldsymbol{A}^t) = \{\boldsymbol{z} \in \mathbb{Z}^m : \boldsymbol{A}^t \boldsymbol{z} = \boldsymbol{0} \bmod q\}.$$

Hard Lattice Problems. The length of the shortest nonzero vector in a lattice Λ is denoted by $\lambda_1(\Lambda)$. The most basic computational problem over lattices is the shortest vector problem (SVP).

Usually the approximation variants of SVP are used, with regard to a parameter $\gamma \geq 1$.

Definition 1 (Search SVP_γ). *Given a lattice basis $\boldsymbol{B} \in \mathbb{Z}^{m \times n}$, and $\gamma \geq 1$, find $\boldsymbol{v} \in \mathcal{L}(\boldsymbol{B})$ s.t. $\|\boldsymbol{v}\| \leq \gamma \cdot \lambda_1(\boldsymbol{B})$.*

Definition 2 (GapSVP_γ). *For $\gamma \geq 1$, given a pair of input $(\boldsymbol{B}, r)$, where $\boldsymbol{B} \in \mathbb{Z}^{m \times m}$ is the basis of a full-rank m-dimensional lattice and $r \in \mathbb{Q}$, it is a YES instance if $\lambda_1(\mathcal{L}(\boldsymbol{B})) \leq r$, and a NO instance if $\lambda_1(\mathcal{L}(\boldsymbol{B})) > \gamma \cdot r$.*

As the generalization to λ_1, the i-th successive minimum $\lambda_i(\Lambda)$ is the smallest radius r s.t. Λ contains i linearly independent vectors of norm at most r. The related shortest independent vector problem (SIVP) and its approximation version are defined as follows.

Definition 3 (SIVP). *Given $\boldsymbol{B} \in \mathbb{Z}^{m \times m}$, the basis of a full-rank m-dimensional lattice, output a set of m linearly independent lattice vectors $\boldsymbol{S} = \{\boldsymbol{s}_1, ..., \boldsymbol{s}_m\} \subset \mathcal{L}(\boldsymbol{B})$, s.t. $\|\boldsymbol{s}_i\| = \lambda_i(\mathcal{L}(\boldsymbol{B}))$.*

Definition 4 (SIVP_γ). *For $\gamma \geq 1$, given $\boldsymbol{B} \in \mathbb{Z}^{m\times m}$, the basis of a full-rank m-dimensional lattice, output a set of m linearly independent lattice vectors $\boldsymbol{S} \subset \mathcal{L}(\boldsymbol{B})$, s.t. $\|\boldsymbol{S}\| \leq \gamma \cdot \lambda_n(\mathcal{L}(\boldsymbol{B}))$.*

The $\gamma - \mathsf{uSVP}$ problem is defined as:

Definition 5 ($\gamma - \mathsf{uSVP}$). *For $\gamma \geq 1$, given the basis $\boldsymbol{B}$ of lattice Λ. If there is $\lambda_2(\Lambda) > \gamma\lambda_1(\Lambda)$ for Λ, find a non-zero vector $\boldsymbol{u} \in \Lambda$, s.t. $\|\boldsymbol{u}\| = \lambda_1(\Lambda)$.*

2.3 Learning with Errors (over Rings)

Currently, most lattice-based public key encryption schemes and key exchange protocols are based on the learning with errors (LWE) assumption [43] and its variants. The proposed cryptosystems in LAC are based on a simplified version of the learning with errors assumption over rings (ring-LWE) [37], which admits more practical implementations and has been widely used.

Definition 6 (Search LWE). *Let n, m, q be positive integers, and $\chi_{\boldsymbol{s}}, \chi_{\boldsymbol{e}}$ be distributions over $\mathbb{Z}$. Given $(\boldsymbol{A}, \boldsymbol{b} = \boldsymbol{A}\boldsymbol{s} + \boldsymbol{e})$, recover the secret $\boldsymbol{s}$, where $\boldsymbol{A} \xleftarrow{\$} \mathbb{Z}_q^{m\times n}$, the secret $\boldsymbol{s} \xleftarrow{\$} \chi_{\boldsymbol{s}}^n$ and the error $\boldsymbol{e} \xleftarrow{\$} \chi_{\boldsymbol{e}}^m$.*

The reasonability of the LWE assumption is based on hard problems over lattices, namely the aforementioned GapSVP_γ and the SIVP_γ problems, where the choice of γ is related to the parameters n, m, q, and the distributions of the secret and the error $\chi_{\boldsymbol{s}}, \chi_{\boldsymbol{e}}$.

Definition 7 (Decisional LWE). *Let n, m, q be positive integers, and $\chi_{\boldsymbol{s}}, \chi_{\boldsymbol{e}}$ be distributions over $\mathbb{Z}$. Distinguish the following two distributions:*

- *$D_0 : (\boldsymbol{A}, \boldsymbol{b})$, and*
- *$D_1 : (\boldsymbol{A}, \boldsymbol{u})$,*

where $\boldsymbol{b} = \boldsymbol{A}\boldsymbol{s} + \boldsymbol{e}$ for $\boldsymbol{A} \xleftarrow{\$} \mathbb{Z}_q^{m\times n}$, $\boldsymbol{s} \xleftarrow{\$} \chi_{\boldsymbol{s}}^n$ and $\boldsymbol{e} \xleftarrow{\$} \chi_{\boldsymbol{e}}^m$, and $\boldsymbol{u} \xleftarrow{\$} \mathbb{Z}_q^m$.

The decisional version is polynomially equivalent to the computational case [38].

In the case of ring-LWE, the noisy equations are $(\boldsymbol{a}, \boldsymbol{b} = \boldsymbol{a}\boldsymbol{s} + \boldsymbol{e})$, where $\boldsymbol{a}$, $\boldsymbol{s}$, $\boldsymbol{e}$ are chosen from a ring. Usually the integer polynomial ring $R_q = \mathbb{Z}_q[x]/(x^n + 1)$ for suitable ring dimension n is used. Sometimes the special case of ring-LWE over R_q is called poly-LWE [15], in which case we write $v \xleftarrow{\$} \chi$ to mean that $v \in R$ is generated from a distribution where each of its coefficients is generated according to χ. In LAC we also use poly-LWE. The reasonability of the ring-LWE assumption is based on the hardness of the SVP_γ problem over ideal lattices, rather than random lattices.

For simplicity, we use the most widely-used definition of ring-LWE, namely poly-LWE over the polynomial ring $R_q = \mathbb{Z}_q[x]/(x^n + 1)$.

Definition 8 (Search Ring-LWE). *Let n, q be positive integers, and $\chi_{\boldsymbol{s}}, \chi_{\boldsymbol{e}}$ be distributions over R. Given $(\boldsymbol{a}, \boldsymbol{b} = \boldsymbol{a}\boldsymbol{s} + \boldsymbol{e})$, recover the secret $\boldsymbol{s}$, where $\boldsymbol{a} \xleftarrow{\$} R_q$, the secret $\boldsymbol{s} \xleftarrow{\$} \chi_{\boldsymbol{s}}$ and the error $\boldsymbol{e} \xleftarrow{\$} \chi_{\boldsymbol{e}}$.*

Definition 9 (Decisional Ring-LWE). *Let n, q be positive integers, and $\chi_{\boldsymbol{s}}, \chi_{\boldsymbol{e}}$ be distributions over R. Distinguish the following two distributions:*

- *$D_0 : (\boldsymbol{a}, \boldsymbol{b})$, and*
- *$D_1 : (\boldsymbol{a}, \boldsymbol{u})$,*

where $\boldsymbol{b} = \boldsymbol{a}\boldsymbol{s} + \boldsymbol{e}$ for $\boldsymbol{a} \xleftarrow{\$} R_q$, $\boldsymbol{s} \xleftarrow{\$} \chi_{\boldsymbol{s}}$ and $\boldsymbol{e} \xleftarrow{\$} \chi_{\boldsymbol{e}}$, and $\boldsymbol{u} \xleftarrow{\$} R_q$.

2.4 Distributions and Random Sampling

The Uniform Distribution. The uniform distribution over a set X is defined as $U(X)$. For example, the uniform distribution over R_q is $U(R_q)$.

The Gaussian Distribution. The secrets and errors in (ring)-LWE are often sampled from Gaussian distributions. The normal Gaussian distribution is a continuous distribution with probability density function:

$$\rho_\sigma(x) = (\sqrt{2\pi}\sigma)^{-1} \exp(-\pi x^2/2\pi\sigma^2),$$

where σ is the standard deviation.

The discrete Gaussian distribution over $\mathbb{Z}$ is defined as:

$$\forall x \in \mathbb{Z}, \mathcal{D}_{\mathbb{Z},\sigma}(x) = \frac{\rho_\sigma(x)}{\rho_\sigma(\mathbb{Z})},$$

where $\rho_\sigma(\mathbb{Z}) = \sum_{y\in\mathbb{Z}} \rho_\sigma(y)$.

The discrete Gaussian distribution over $\mathbb{Z}_q$ is defined as:

$$\forall x \in \mathbb{Z}_q, \mathcal{D}_{\mathbb{Z}_q,\sigma}(x) = \sum_{w, w=x \bmod q} \mathcal{D}_{\mathbb{Z},\sigma}(w).$$

The Centered Binomial Distribution. Since the sampling of discrete Gaussian distribution is not an easy task, in the design of practical lattice-based cryptosystems, the centered binomial distribution is also used, as introduced in [8]. Let Ψ_σ be the centered binomial distribution with σ being the parameter of the distribution, where the corresponding standard variance is $\sqrt{\frac{\sigma}{2}}$. In the design of LAC we also use centered binomial distribution with parameters 1 and $\frac{1}{2}$ (denoted as Ψ_1 and $\Psi_{\frac{1}{2}}$ respectively) as follows:

Definition 10 (Ψ_1). *Sample $(a, b) \xleftarrow{\$} \{0, 1\}^2$, and output $a - b$. Obviously it picks 0 with probability $\frac{1}{2}$, and ± 1 with probability $\frac{1}{4}$ according to the distribution Ψ_1. The mean of Ψ_1 is 0 and the variance is $\frac{1}{2}$.*

Definition 11 ($\Psi_{\frac{1}{2}}$). *Sample $(a, b) \xleftarrow{\$} \Psi_1$, and output $a * b$. Obviously it picks 0 with probability $\frac{3}{4}$, and ± 1 with probability $\frac{1}{8}$ according to the distribution $\Psi_{\frac{1}{2}}$. The mean of $\Psi_{\frac{1}{2}}$ is 0 and the variance is $\frac{1}{4}$.*

For a positive integer n, Ψ_σ^n denotes the n independently identical distribution of Ψ_σ. When sampling according to Ψ_σ^n, the components of the random variable are all independently chosen.

Besides, we define n-ary centered binomial distribution with fixed Hamming weight, denoted as $\Psi_\sigma^{n,h}$, when $0 < h < n/2$ is even. For a random variable according to the distribution, its Hamming weight is fixed to the expectation h, and the numbers of both 1's and -1's are $h/2$, the number of 0 is $n - h$.

Random Sampling. Define an abstract algorithm Samp as the procedure of sampling a random variable according to a distribution with a given seed:

$$x \leftarrow \mathsf{Samp}(D; \mathsf{seed}),$$

where D is a distribution, and seed is the random seed used to sample x. For an empty $\mathsf{seed} = \epsilon$, the process is the same as $x \xleftarrow{\$} D$, and is totally randomized. Otherwise, the sampling of x is always deterministic for the same seed.

We use

$$(x_1, x_2, \cdots, x_t) \leftarrow \mathsf{Samp}(D_1, D_2, \cdots, D_t; \mathsf{seed})$$

to denote the process of sampling random variables x_i according to distribution D_i where $1 \le i \le t$.

3 Design Rationale

Criteria taken into account in the design of LAC as listed as below.

- Security:
  - A provable secure design from Ring-LWE;
  - Concrete parameters resistant against all known attacks (with considerable margin).
- Efficiency:
  - Small key size and ciphertext size;
  - High speed.
- Correctness: low decryption error rate;
- Flexibility: easy to set parameters for different security strength categories.

We stress that, while achieving the claimed security, our focus is on the size. This appears to be the main bottleneck in practice. The most standing out aspect of LAC, from this point of view, is its byte size modulus.

Our focus on reducing the modulus. Intuitively, the hardness of Ring-LWE problem is mainly determined by the dimension n and the error rate α, where $\alpha = \frac{\sigma}{q}\sqrt{2\pi}$ is the ratio of the noise magnitude (measured with the standard variation σ) to the modulus q. Thus the choices of the noise distribution and the modulus are critical. According to the concrete hardness analysis in [6,8,5], suitable choices of the dimension n are $2^9 = 512$ and $2^{10} = 1024$, and in Module-LWE based schemes $2^8 = 256$ is also used. For these choices, $q = 12289$ or 7681 are chosen to enable the super efficient NTT multiplications.

On the other hand, this constraint is a bit artificial, in that it is purely decided by NTT, and not regulated by any security requirement. To be more specific, the security level grows with the error rate, which is the ratio between the error and the modulus, rather than the modulus itself. Therefore, to achieve a great compactness with an acceptable level of security, it makes sense to choose the modulus as small as possible, while keeping the ratio somewhat a constant.

Our main design philosophy is to set the modulus q as small as possible to improve the bandwidth efficiency. Thus, in LAC we use byte-level modulus, such as $q = 251$. Although NTT can not be used to speed up the computational efficiency when $q < 7681$, for a byte-level modulus, we can use the Intel Advanced Vector Extensions2 (AVX2) instructions to improve the computational efficiency. Briefly, the AVX2 instructions can be used to process multiple multiplication operations in one instruction cycle.

Secrets and noises. One caveat arises along with byte size modulus is that we need precise control of the noises and secrets. When α becomes too large, the decryption error rate becomes unacceptable. We limit ourself to trinary secrets and noises, i.e., polynomials with coefficients over $\{-1, 0, 1\}$. In addition, we adopt error correction code with large blocks and large code distance, such as the BCH code to correct the errors. In principle, any code with enough error correcting capability can be used in our scheme, such as Goppa, LDPC. For the sake of simplicity and efficiency we choose BCH code for implementation and benchmarking.

In LAC, we choose to use a centered binomial distribution over $\{-1, 0, 1\}$ with fixed hamming weight. This distribution, with appropriate parametrization, is sufficient for our purpose. It also defeats high hamming weight CCA attacks.

Transformations. It is easy to get an ephemeral key establishment scheme based on a basic public key encryption scheme using traditional frameworks. The transformations to adaptive chosen ciphertext secure public key encryption scheme and authenticated key establishment scheme are more subtle. We use the popular FO [27,28,30,32] framework to get the adaptive chosen ciphertext secure public key encryption scheme, and the FSXY [25,26] framework to get the authenticated key establishment scheme.

4 Description of the Cryptosystems

4.1 LAC.CPA

The IND-CPA secure **public key encryption** scheme LAC.CPA lays the foundation of the entire LAC. It comprises three algorithms:

- The key generation algorithm KG, as illustrated in Algorithm 1.
- The encryption algorithm Enc, as illustrated in Algorithm 2.
- The decryption algorithm Dec, as illustrated in Algorithm 3.

Notations. Let q be the modulus, and define the polynomial ring $R_q = \mathbb{Z}_q/(x^n + 1)$.

Define the message space $\mathcal{M}$ be $\{0,1\}^{l_m}$ for a positive integer l_m, and the space of random seeds $\mathcal{S}$ be $\{0,1\}^{l_s}$ for a positive integer l_s. The integers l_m and l_s will be specified afterwards.

We use n independently identical distribution of Ψ_σ, namely Ψ_σ^n. Beside, we use the n-ary centered binomial distribution $\Psi_\sigma^{n,h}$, where the concrete choices of parameters will be given later.

Subroutines. In the subroutines dealing with the encoding and decoding of the error correction codes, ECCEnc, ECCDec, the conversion between a message $\boldsymbol{m} \in \{0,1\}^{l_m}$ and its encoding $\widehat{\boldsymbol{m}} \in \{0,1\}^{l_v}$ is provided, wherein l_v is a positive integer denoting the length of the encoding and depending on the specific choice of the parameter settings.

The algorithms. The algorithm LAC.CPA.KG randomly generates a pair of public key and secret key (pk, sk).

Algorithm 1 LAC.CPA.KG()

Ensure: A pair of public key and secret key (pk, sk).

1: $\mathsf{seed}_{\boldsymbol{a}} \xleftarrow{\$} \mathcal{S}$
2: $\boldsymbol{a} \leftarrow \mathsf{Samp}(U(R_q); \mathsf{seed}_{\boldsymbol{a}}) \in R_q$
3: $\boldsymbol{s} \xleftarrow{\$} \Psi_\sigma^{n,h}$
4: $\boldsymbol{e} \xleftarrow{\$} \Psi_\sigma^{n,h}$
5: $\boldsymbol{b} \leftarrow \boldsymbol{a}\boldsymbol{s} + \boldsymbol{e} \in R_q$
6: **return** $(pk := (\mathsf{seed}_{\boldsymbol{a}}, \boldsymbol{b}), sk := \boldsymbol{s})$

The algorithm LAC.CPA.Enc on input pk and a message $\boldsymbol{m}$, encrypts $\boldsymbol{m}$ with the randomness seed. In case that seed is not given, the process is randomized. Otherwise, the encryption is deterministic for the same seed. The subroutine ECCEnc converts the message $\boldsymbol{m}$ into a codeword $\widehat{\boldsymbol{m}}$.

Algorithm 2 LAC.CPA.Enc$(pk = (\mathsf{seed}_{\boldsymbol{a}}, \boldsymbol{b}), \boldsymbol{m} \in \mathcal{M}; \mathsf{seed} \in \mathcal{S})$

Ensure: A ciphertext $\boldsymbol{c}$.

1: $\boldsymbol{a} \leftarrow \mathsf{Samp}(U(R_q); \mathsf{seed}_{\boldsymbol{a}}) \in R_q$
2: $\widehat{\boldsymbol{m}} \leftarrow \mathsf{ECCEnc}(\boldsymbol{m}) \in \{0,1\}^{l_v}$
3: $(\boldsymbol{r}, \boldsymbol{e}_1, \boldsymbol{e}_2) \leftarrow \mathsf{Samp}(\Psi_\sigma^{n,h}, \Psi_\sigma^{n,h}, \Psi_\sigma^{l_v}; \mathsf{seed})$
4: $\boldsymbol{c}_1 \leftarrow \boldsymbol{a}\boldsymbol{r} + \boldsymbol{e}_1 \in R_q$
5: $\boldsymbol{c}_2 \leftarrow (\boldsymbol{b}\boldsymbol{r})_{l_v} + \boldsymbol{e}_2 + \lfloor \frac{q}{2} \rceil \cdot \widehat{\boldsymbol{m}} \in \mathbb{Z}_q^{l_v}$
6: **return** $\boldsymbol{c} := (\boldsymbol{c}_1, \boldsymbol{c}_2) \in R_q \times \mathbb{Z}_q^{l_v}$

The algorithm LAC.CPA.Dec on input sk and a ciphertext $\boldsymbol{c}$, recovers the corresponding message $\boldsymbol{m}$. The subroutine ECCDec on input an encoding $\widehat{\boldsymbol{m}}$, decoding the codeword in it.

Usually, a message $\boldsymbol{m} \in \mathcal{M}$ is recovered. When there is a decryption error, the returned message $\boldsymbol{m} \notin \mathcal{M}$.

Algorithm 3 LAC.CPA.Dec($sk = \boldsymbol{s}, \boldsymbol{c} = (\boldsymbol{c}_1, \boldsymbol{c}_2)$)

Ensure: A plaintext $\boldsymbol{m}$.

$\boldsymbol{u} \leftarrow \boldsymbol{c}_1 \boldsymbol{s} \in R_q$
$\widetilde{\boldsymbol{m}} \leftarrow \boldsymbol{c}_2 - (\boldsymbol{u})_{l_v} \in \mathbb{Z}_q^{l_v}$
for $i = 0$ to $l_v - 1$ **do**
  if $\frac{q}{4} \leq \widetilde{\boldsymbol{m}}_i < \frac{3q}{4}$ **then**
    $\widehat{\boldsymbol{m}}_i \leftarrow 1$
  else
    $\widehat{\boldsymbol{m}}_i \leftarrow 0$
  end if
end for
$\boldsymbol{m} \leftarrow \mathsf{ECCDec}(\widehat{\boldsymbol{m}})$
return $\boldsymbol{m}$

4.2 LAC.CCA

The IND-CCA secure **key encapsulation mechanism** LAC.CCA is obtained by applying the Fujisaki-Okamoto transformation [27,30] to the IND-CPA secure encryption scheme LAC.CPA. The method was suggested by Peikert in [40] and also used in [13].

LAC.CCA comprises the following three algorithms:

- The key generation algorithm KG, which is the same with the key generation algorithm of LAC.CPA, as illustrated in Algorithm 1.
- The encapsulation algorithm Enc, as illustrated in Algorithm 4.
- The decapsulation algorithm Dec, as illustrated in Algorithm 5.

Notations. The notations for the description of LAC.CCA are the same with those of LAC.CPA. Additionally, we use a hash function $G : \{0,1\}^{l_m} \rightarrow \mathcal{S} \in \{0,1\}^{l_s}$, and a hash function $H : \{0,1\}^* \rightarrow \{0,1\}^{l_k}$ for the FO transformation and generating the encapsulated key, where l_k denotes the length of the session key, and will vary depending on different security levels. In LAC we always set $l_k = l_m$. Currently, the hash functions G and H are implemented with SHA256, and can be replaced with any other hash functions such as SHAKE256.

The algorithm LAC.CCA.Enc on input pk and a seed $\mathsf{seed}_{\boldsymbol{m}}$, generates a message $\boldsymbol{m}$, and encrypts $\boldsymbol{m}$ by invoking LAC.CPA.Enc with $pk, \boldsymbol{m}$ and the randomness seed, which is generated from $\boldsymbol{m}$.

Algorithm 4 LAC.CCA.Enc($pk; \mathsf{seed}_{\boldsymbol{m}}$)

Ensure: A ciphertext and encapsulated key pair $(\boldsymbol{c}, K)$.

$\boldsymbol{m} \leftarrow \mathsf{Samp}(U(\mathcal{M}); \mathsf{seed}_{\boldsymbol{m}}) \in \mathcal{M}$
$\mathsf{seed} \leftarrow G(\boldsymbol{m}) \in \mathcal{S}$
$\boldsymbol{c} \leftarrow \mathsf{LAC.CPA.Enc}(pk, \boldsymbol{m}; \mathsf{seed})$
$K \leftarrow H(\boldsymbol{m}, \boldsymbol{c}) \in \{0,1\}^{l_k}$
return $(\boldsymbol{c}, K)$

The decapsulation algorithm LAC.CCA.Dec on input sk and a ciphertext, recovers a message by invoking LAC.CPA.Dec. Then it verifies the correctness of the decryption by a re-encryption process. In case that the verification is passed, it returns the encapsulated key. Otherwise, it generates a pseudorandom key from the secret key and the ciphertext.

Algorithm 5 LAC.CCA.Dec$(sk, \boldsymbol{c})$

Ensure: An encapsulated key K.

$\boldsymbol{m} \leftarrow$ LAC.CPA.Dec$(sk, \boldsymbol{c})$
$K \leftarrow H(\boldsymbol{m}, \boldsymbol{c})$
seed $\leftarrow G(\boldsymbol{m}) \in \mathcal{S}$
$\boldsymbol{c}' \leftarrow$ LAC.CPA.Enc$(pk, \boldsymbol{m}$; seed$)$
if $\boldsymbol{c}' \neq \boldsymbol{c}$ **then**
  $K \leftarrow H(H(sk), c)$
end if
return K

4.3 LAC.KE

The passively secure unauthenticated key exchange protocol LAC.KE is directly obtained from the IND-CPA secure encryption scheme LAC.CPA, as described in Figure 2.

Notations. The notations for the description of LAC.KE are the same as those of LAC.CPA. Additionally, we use a hash function $H : \{0,1\}^* \rightarrow \{0,1\}^{l_k}$ for generating the session key, where l_k denotes the length of the session key, and will vary depending on different security levels. The concrete choice of H will be specified in Section 7.

Fig. 2. LAC.KE: the unauthenticated lattice-based key exchange protocol

Parameters: the specification of LAC.CPA
$H : \{0,1\}^* \rightarrow \{0,1\}^{l_k}$

Alice		Bob
$(pk, sk) \xleftarrow{\$}$ LAC.CPA.KG()	$\xrightarrow{pk}$	
		$r \xleftarrow{\$} \{0,1\}^{l_m}$
	$\xleftarrow{c}$	$c \xleftarrow{\$}$ LAC.CPA.Enc(pk, r)
$r \leftarrow$ LAC.CPA.Dec(sk, c)		$K \leftarrow H(pk, r) \in \{0,1\}^{l_k}$
$K \leftarrow H(pk, r) \in \{0,1\}^{l_k}$		

Besides, we can also construct a passively secure key exchange protocol directly from LAC.CCA.

4.4 LAC.AKE

The authenticated key exchange protocol LAC.AKE is built from the chosen-plaintext secure public key encryption LAC.CPA and the chosen-ciphertext secure key encapsulation mechanism LAC.CCA, by following the framework of [25,26]. The LAC.AKE is secure in Canetti-Krawczyk mode with weak perfect forward secrecy [17], resistance to key compromise impersonation (KCI) attack [25,26], and maximal exposure attacks (MEX) [25,26]. The protocol is described in Figure 3.

Fig. 3. LAC.AKE: the authenticated lattice-based key exchange protocol

Parameters: the specification of LAC.CCA and LAC.CPA
$G : \{0,1\}^* \to \{0,1\}^{l_s}$, $H : \{0,1\}^* \to \{0,1\}^{l_k}$

Alice		Bob
$(pk_A, sk_A) \xleftarrow{\$} \mathsf{LAC.CCA.KG}()$		$(pk_B, sk_B) \xleftarrow{\$} \mathsf{LAC.CCA.KG}()$
static public key: pk_A		static public key: pk_B
static secret key: sk_A		static secret key: sk_B
$(pk, sk) \xleftarrow{\$} \mathsf{LAC.CPA.KG}()$		
$r_1 \xleftarrow{\$} \{0,1\}^{l_s}$		
$\mathsf{seed}_1 \leftarrow G(r_1, sk_A)$		
$(\boldsymbol{c}_1, K_1) \leftarrow \mathsf{LAC.CCA.Enc}(pk_B; \mathsf{seed}_1)$	$\xrightarrow{pk, \boldsymbol{c}_1}$	
		$K_1 \leftarrow \mathsf{LAC.CCA.Dec}(sk_B, \boldsymbol{c}_1)$
		$r_2 \xleftarrow{\$} \{0,1\}^{l_s}$
		$\mathsf{seed}_2 \leftarrow G(r_2, sk_B)$
		$(\boldsymbol{c}_2, K_2) \leftarrow \mathsf{LAC.CCA.Enc}(pk_A; \mathsf{seed}_2)$
		$K_3 \xleftarrow{\$} \{0,1\}^{l_m}$
	$\xleftarrow{\boldsymbol{c}_2, \boldsymbol{c}_3}$	$\boldsymbol{c}_3 \xleftarrow{\$} \mathsf{LAC.CPA.Enc}(pk, K_3; \epsilon)$
$K_2 \leftarrow \mathsf{LAC.CCA.Dec}(sk_A, \boldsymbol{c}_2)$		
$K_3 \leftarrow \mathsf{LAC.CPA.Dec}(sk, \boldsymbol{c}_3)$		
$K \leftarrow H(pk_A, pk_B, pk, \boldsymbol{c}_3, K_1, K_2, K_3)$		$K \leftarrow H(pk_A, pk_B, pk, \boldsymbol{c}_3, K_1, K_2, K_3)$

5 Parameter Choices

Almost all lattice based key exchanges and public key encryptions, except for NTRU based ones, follow a similar framework from [22,40,12,7]. There are a set of theoretical results on the choice of rings, moduli, errors, etc [42,41,44] that ensure the framework stems from a provable secure design. However, those theoretical results do not give any guidance on selecting concrete parameters. Choosing parameters for (Ring-)LWE based schemes becomes an important research direction in subsequent works [12,7,41,14,33,45], and a main differentiator in most NIST-PQC submissions [1]. In this section, we present our choice of parameters, and give our design rational over common choices.

5.1 Modulus

Our first and foremost priority is to reduce the modulus. As mentioned earlier, the payload sizes are governed mainly by the dimension and the modulus. The choice of power-of-2 cyclotomic polynomial does not allow much freedom in the choice of n. Hence we focus on a small modulus to reduce the payload size. Note that the modulus cannot be too small; it needs to be large enough to tolerant the errors during decryption which will be scaled by a factor of $\sqrt{2n}$. A common choice was $q = 12289$. We take a more aggressive approach by using "byte level modulus".

A byte is the basic operating unit for most processors. Such a choice makes the public keys and ciphertexts compact, and is also optimal for implementations. The downside is that decryption errors increase when modulus is smaller. We will give more details in Section 5.3.

Depending on the structure of the polynomial ring, we consider three types of byte-level modulus.

- **Power of Two Modulus:** From the view of implementation, the most suitable byte-level modulus is $q = 256$, for which the modulus operation can be efficiently realized by ignoring

the carrier data. However, since $q = 256$ is not a prime, $\mathbb{Z}_{256}[x]/(x^n+1)$ does not yield a field for our choice of n. For conservative purpose we do not use this ring to avoid any potential weakness of the underlying structure.

- **Max-Split Modulus:** The reason to choose $q \equiv 1 \bmod 2n$ is that $x^n + 1 \in \mathbb{Z}_q[x]$ can be completely factorized. For byte-level modulus, this is no longer the case. However, we notice that when $q = 257$, $x^n + 1 \in \mathbb{Z}_{257}[x]$ has maximum number of factors:

$$x^{512} + 1 = \prod_{i=1}^{128}(x^4 + \tau_i), \qquad x^{1024} + 1 = \prod_{i=1}^{128}(x^8 + \tau_i),$$

  where $\tau_i \in \mathbb{Z}_q$. We call this type of modulus "Max-Split Modulus", for which $x^n + 1$ can be maximally factorized into polynomials with very small degrees.
- **Min-Split Modulus:** Unlike $q = 257$, for some other modulus, $x^n + 1 \in \mathbb{Z}_q[x]$ may have minimum number of factors. Concretely, we notice that for $q = 251$, which is the largest prime smaller than 2^8, $x^n + 1 \in \mathbb{Z}_{251}[x]$ can be minimally factorized as:

$$x^n + 1 = (x^{n/2} + 91x^{n/4} + 250)(x^{n/2} + 160x^{n/4} + 250).$$

  We call this type of modulus "Min-Split Modulus", for which $x^n + 1$ can only be factorized into two polynomials with the degree of $n/2$.

It has been argued that less algebraic structure reduces the attacking surface [10]. In that spirit, and also for the sake of simplicity, we choose the min-split modulus $q = 251$ for our scheme.

Remark 1. Our selection principle is simply to minimize algebraic structures. Nonetheless, we do not see any weakness of the power of two modulus or the max-split modulus. In fact, it has been shown in theory [42] that Ring-LWE is hard for any ring of integers, which implies that $\mathbb{Z}_{2^\ell}/(x^n + 1)$ is as hard as any other choices, asymptotically speaking. Further, one can convert instances over one ring to another via modulus switching [4,6], at a cost of increased secrets and/or errors. In the meantime, from the implementation point of view, the modulus 257 and 256 may deliver better efficiency. We leave the concrete security of those types of modulus to future research.

5.2 The Errors and Secrets Distribution of Ring-LWE

There are two rules for the choice of the distribution for the error and secret vector of the poly-LWE problem. Firstly, the errors and the secrets must be large enough to guarantee the hardness of the poly-LWE problem. Secondly, the errors and the secrets must be small enough to guarantee the correctness of the decryption algorithm. In literatures, there are mainly two families of distributions that satisfy the average/worst case reduction theorem [43,37], namely, discrete Gaussian distribution [37,8] and centered binomial distribution [14]. Gaussian distribution consumes lots of entropy, is hard to implement (in constant time), and is also vulnerable to memory based side channel attacks [16] when implemented with look-up tables [23]. Therefore, we opt to use the centered binomial distribution for our scheme.

In the implementation, as described in [8], a centered binomial distribution with the standard deviation of $\sqrt{\lambda/2}$ can be generated as $\sum_{i=1}^{\lambda}(b_i - \hat{b}_i)$, where $b_i, \hat{b}_i \in \{0, 1\}$ are uniformly random bits. When a byte-level modulus is used, the error-modulus-ratio becomes large enough even for small error distributions. This allow us to use the simplest centered binomial distribution with $\lambda = 1$ as our basic error distribution. That is, in order to get a centered binomial distribution with $\lambda = 1$, each element of the error vector is generated by $b - \hat{b}$, where $b, \hat{b}$ are uniformly random bits. Then we can get the distribution Ψ_1. Besides, we also use a narrower distribution $\Psi_{\frac{1}{2}}$.

1. Ψ_1: $\Pr[x = 0] = 1/2$, $\Pr[x = \pm 1] = 1/4$.
2. $\Psi_{\frac{1}{2}}$: $\Pr[x = 0] = 3/4$, $\Pr[x = \pm 1] = 1/8$.

As pointed out by Alperin-Sheriff in the comments to LAC [1], when centered binomial distribution is used, the adversary can increase the decryption error rate by finding high hamming weight random vectors through pre-computation. The direct approach to resist this attack is to decrease the error rate by using a more powerful error correction code. However, correcting more errors will affect the efficiency of the error correction code.

To make LAC immune high hamming weight attack in a more efficient manner, we use n-ary centered binomial distribution $\Psi_\sigma^{n,h}$ with fixed Hamming weight h for the error and secret vectors $\boldsymbol{s}, \boldsymbol{e}, \boldsymbol{r}, \boldsymbol{e}_1$. Since $\boldsymbol{e}_2$ only has slight influence on the decryption error rate, we still use the more efficient standard centered binomial distribution for $\boldsymbol{e}_2$. Concretely, we use the following distributions:

- LAC-128: Sample $\boldsymbol{s}, \boldsymbol{e}, \boldsymbol{r}, \boldsymbol{e}_1$ from $\Psi_1^{n,h}$, $\boldsymbol{e}_2$ from Ψ_1^n, with $n = 512, h = 256$.
- LAC-192: Sample $\boldsymbol{s}, \boldsymbol{e}, \boldsymbol{r}, \boldsymbol{e}_1$ from $\Psi_{\frac{1}{2}}^{n,h}$, $\boldsymbol{e}_2$ from $\Psi_{\frac{1}{2}}^n$, with $n = 1024, h = 256$.
- LAC-256: Sample $\boldsymbol{s}, \boldsymbol{e}, \boldsymbol{r}, \boldsymbol{e}_1$ from $\Psi_1^{n,h}$, $\boldsymbol{e}_2$ from Ψ_1^n, with $n = 1024, h = 512$.

We remark that, when the hamming weight of the secret or error vector is fixed as the expectation of the standard centered binomial distribution, it only brings a very small effect to its entropy. Concretely, we compute the entropy of the distributions we used as follows:

- LAC-128: The entropy of $\boldsymbol{s}$ from Ψ_1^n with $n = 512$ is 768, when the hamming weight is fixed as $h = 256$ with 128 ones and 128 minus-ones, its entropy is about $\log\left(\binom{512}{128} \cdot \binom{384}{128}\right) \approx 758$.
- LAC-192: The entropy of $\boldsymbol{s}$ from $\Psi_{\frac{1}{2}}^n$ with $n = 1024$ is 1086, when the hamming weight is fixed as $h = 256$ with 128 ones and 128 minus-ones, its entropy is about $\log\left(\binom{1024}{128} \cdot \binom{896}{128}\right) \approx 1077$.
- LAC-256: The entropy of $\boldsymbol{s}$ from Ψ_1^n with $n = 1024$ is 1536, when the hamming weight is fixed as $h = 512$ with 256 ones and 256 minus-ones, its entropy is about $\log\left(\binom{1024}{256} \cdot \binom{768}{256}\right) \approx 1525$.

So using fixed weight centered binomial distribution will not affect the security reduction and concrete security evaluation of LAC.

5.3 Decryption Errors

As shown in the decryption algorithm, the message is recovered via two steps. First, the error correction code word $\widehat{\boldsymbol{m}}$ is recovered from the ciphertext. Then, the message $\boldsymbol{m}$ is recovered from the code word. It is easy to verify that:

$$\begin{aligned}\widetilde{\boldsymbol{m}} &= \boldsymbol{c}_2 - (\boldsymbol{c}_1\boldsymbol{s})_{l_v} \\ &= (\boldsymbol{b}\boldsymbol{r})_{l_v} + \boldsymbol{e}_2 + \lfloor \tfrac{q}{2} \rceil \widehat{\boldsymbol{m}} - (\boldsymbol{c}_1\boldsymbol{s})_{l_v} \\ &= ((\boldsymbol{a}\boldsymbol{s} + \boldsymbol{e})\boldsymbol{r})_{l_v} + \boldsymbol{e}_2 + \lfloor \tfrac{q}{2} \rceil \widehat{\boldsymbol{m}} - ((\boldsymbol{a}\boldsymbol{r} + \boldsymbol{e}_1)\boldsymbol{s})_{l_v} \\ &= (\boldsymbol{e}\boldsymbol{r} - \boldsymbol{e}_1\boldsymbol{s})_{l_v} + \boldsymbol{e}_2 + \lfloor \tfrac{q}{2} \rceil \widehat{\boldsymbol{m}}\end{aligned} \tag{1}$$

Let $\boldsymbol{w} = (\boldsymbol{e}\boldsymbol{r} - \boldsymbol{e}_1\boldsymbol{s})_{l_v} + \boldsymbol{e}_2$, we have that the error rate of each $\widetilde{m}_i$ is $\delta = 1 - \Pr[-\lfloor \frac{q}{4} \rceil < w_i < \lfloor \frac{q}{4} \rceil]$. If $\boldsymbol{s}, \boldsymbol{e}, \boldsymbol{r}, \boldsymbol{e}_1, \boldsymbol{e}_2$ are all randomly chosen from a small distribution with a standard deviation of σ and an expectation of 0, then according to the central limit theory, w_i follows a distribution that is very close to a discrete Gaussian distribution with a standard deviation of $\sigma^2\sqrt{2n}$ and an expectation of 0. Thus, the error rate for each bit can be approximated by the Gaussian error function as $\delta \approx 1 - \mathsf{erf}\left(\frac{\lfloor q/4 \rceil}{\sqrt{2}(\sigma^2\sqrt{2n})}\right)$. For example, For $n = 512, q = 251$, and a distribution of Ψ_1 with a standard deviation $\sigma = 1/\sqrt{2}$, the error rate of each bit is estimated by:

$$\delta \approx 1 - \mathsf{erf}\left(\frac{\lfloor 251/4 \rceil}{\sqrt{2}((1/\sqrt{2})^2\sqrt{2 \times 512})}\right) \approx 2^{-13.195}.$$

Suppose that the BCH code can correct l_t errors at most and the code word length is $l_n = l_v$, and assume the coefficients of $\boldsymbol{w}$ are independent from each other, we have the decryption error rate for a message $\boldsymbol{m}$:

$$\Delta \approx \sum_{j=l_t+1}^{l_v} \left(\binom{l_v}{j} \delta^j (1-\delta)^{l_v - j}\right) \tag{2}$$

As pointed out by D'Anvers [20], when single bit error rate δ is too large, we can not assume that the coefficients of $\boldsymbol{w}$ are independent from each other. The theoretical dependence model and experiment results of D'Anvers show that the dependence mainly comes from the norm of $\boldsymbol{s}, \boldsymbol{e}, \boldsymbol{r}, \boldsymbol{e}_1$. When fix hamming weight distribution is used for $\boldsymbol{s}, \boldsymbol{e}, \boldsymbol{r}, \boldsymbol{e}_1$, their norms are also fixed and the main source of dependence is removed. So we can assume that the coefficients of $\boldsymbol{w}$ are independent from each other.

5.4 The Error Correction Code

Our byte level modulus incurs a very high decryption error rate by design. Trivial or light error correction methods such as D2 or D4 code [8] are not capable of handling such a situation. Heavy error correction methods ought to be used for our use case. In the field of code theory, there are many powerful codes such as BCH, Goppa, LDPC, Turbo and Polar. In principle, any code with enough error correcting capability can be used in our scheme. For the sake of simplicity and efficiency we choose BCH code for implementation and benchmarking.

5.5 Recommended Parameter Categories

We recommend the following parameter sets in Table 1, with respect to three categories of NIST post-quantum standardization project [1], namely, the equivalent security level of AES128, AES192 and AES256.

Categories	n	q	**dis**	**ecc**	$l_{\boldsymbol{m}}$	**pk**	**sk**	**ct**	**bit-er**	**dec-er**
LAC-128	512	251	$\Psi_1^n, \Psi_1^{n,\frac{n}{2}}$	$[511, 256, 33]$	256	544	512	712	$2^{-12.61}$	2^{-116}
LAC-192	1024	251	$\Psi_{\frac{1}{2}}^n, \Psi_{\frac{1}{2}}^{n,\frac{n}{4}}$	$[511, 256, 17]$	256	1056	1024	1188	$2^{-22.27}$	2^{-143}
LAC-256	1024	251	$\Psi_1^n, \Psi_1^{n,\frac{n}{2}}$	[511,256,33]+D2	256	1056	1024	1424	$2^{-12.96}$	2^{-122}

dis secret and noise distributions **ecc** error correction code
$l_{\boldsymbol{m}}$ message length **sk** secret key size (bytes)
pk public key size (bytes) **ct** ciphertext size (bytes)
bit-er single bit error rate without BCH **dec-er** decryption error rate

Table 1. Recommended parameter of LAC.CPA

Concretely, dimensions $n = 512$ and $n = 1024$ with a basic error distribution Ψ_1 discussed as above are for the low security level LAC-128 and the high security level LAC-256, respectively. To get the middle security level LAC-192, we use a smaller secret and noise distribution $\Psi_{1/2}$ and dimension 1024.

Note that it is sufficient to set the message size according to the security level, since in practice, public key encryption schemes are mainly used to encrypt session keys for symmetric encryption scheme. For the sake of simplicity, we set the message size to 256 for all security levels. In the previous version of LAC parameter sets [36], the message size was twice as the security level.

The parameters of BCH code are selected to achieve a suitable decryption error rate and a high efficiency while defeating the high Hamming weight attacks. We have $l_m = 256$ in our setting. Next, for $l_m = 256$, the minimum available BCH code length l_n is 511. Lastly, for the low security level LAC-128 and the high security level LAC-256, we choose $l_d = 33$ which allows us to correct 16 bits of errors at most. The redundant data due to this error correction code is 18 bytes. And for LAC-192, we use $l_d = 17$ which allows to correct up to 8 errors, and the redundancy is 9 bytes.

Note that the error rate for each coefficient is estimated by a convolution of all the error terms. In order to minimize the size of the ciphertext, in our implementation the lower 4 bits for each coefficient in $\boldsymbol{c}_2$ are discarded. This brings an additional uniformly random (under Ring-LWE assumption) error over $[-7, 7]$.

A public key consists of a 32 bytes seed $\mathsf{seed}_{\boldsymbol{a}}$, and an n bytes vector $\boldsymbol{b}$. A secret key is an n bytes vector. One may simply store a 32 bytes seed for the secret key to minimize storage, at a cost of slightly slower decryption. In the case where Fujisaki-Okamoto transformation is used to achieve chosen ciphertext security, a secret key also contains a copy of the corresponding public key, so that the decryption algorithm can re-encrypt to check the validity of the ciphertext. Thus the size of a secret key becomes $2n + 32$. Finally, a ciphertext contains both an n bytes vector $\boldsymbol{c}_1$, and l_v number of "half-byte" from $\boldsymbol{c}_2$ (since the lower 4 bits of each coefficient in $\boldsymbol{c}_2$ are discarded). For LAC-128 parameter set, $l_v = l_m + 18 \times 8$, where 18 is the size of the redundant data. For LAC-192, $l_v = l_m + 9 \times 8$, where 9 is the size of the redundant data. And for LAC-256, $l_v = (l_m + 18 \times 8) \times 2$ due to the use of D2 encoding.

6 Security Analysis

Formal Security: Following the result of [35], the chosen plaintext security of LAC can be easily reduced to the Ring-LWE assumption. Then, with Fujisaki-Okamoto transformation, we obtain the chosen ciphertext security version of LAC in both classical random oracle model [27,28] and quantum random oracle model [32]. It is easy to verify that the embedded error correction code will not affect the security reduction and these security proofs can be directly extended to the case of LAC. Therefore, we omit the details for both reductions.

Concrete Security: We consider the best known generic attacks against Ring-LWE with our parameters, which treat the Ring-LWE problems as plain LWE problems. Those attacks are well-known by the community; their costs are well understood.

We also consider dedicated attacks that target specific designs of our scheme, namely the subfield attacks and the high Hamming weight attacks. Those attacks are reported as comments to the Round 1 version of LAC submission to NIST-PQC. We will show that none of those dedicated attack works better than generic attacks for our (revised) parameter sets. Therefore, it is sufficient to use common methods (e.g. BKZ with (quantum) sieving) to evaluate the security of our scheme.

6.1 Generic Attacks

There are many generic algorithms to solve the LWE problem, see [6,46] for a survey of known techniques. It has been shown that lattice reduction attacks utilizing the BKZ algorithm [18] are more powerful than exhaustive search, combinational and algebraic algorithms. For simplicity, following the analysis of [7], we focus primly on two embedding attacks that are commonly referred to as primal attack and dual attack. We summarize the security estimates of both attacks in Table 2.

Algorithm	Primal Attack			Dual Attack		
	Classic	**Quantum**	**Block Size**	**Classic**	**Quantum**	**Block Size**
LAC-128	148	135	509	147	133	505
LAC-192	288	261	986	286	259	978
LAC-256	323	293	1105	320	290	1095

Classic: Classical complexity **Quantum:** Quantum complexity

Table 2. Concrete security of LAC

Primal attack. In a primal attack, one builds a lattice with a unique-SVP instance from the LWE samples; then, uses BKZ algorithm to recover this unique shortest vector. In a nutshell, given an LWE instance $(\boldsymbol{A}, \boldsymbol{b} = \boldsymbol{A}\boldsymbol{s}+\boldsymbol{e})$, $\boldsymbol{A} \in \mathbb{Z}_q^{m\times n}$, the target lattice of dimension $d = m+n+1$ is constructed as

$$\Lambda_{\boldsymbol{A}} = \{\boldsymbol{x} \in \mathbb{Z}^{m+n+1} : (\boldsymbol{A}|\boldsymbol{I}_m| - \boldsymbol{b})\boldsymbol{x} = \boldsymbol{0} \mod q\}.$$

It is easy to verify that, $\boldsymbol{v} = (\boldsymbol{s}, \boldsymbol{e}, 1)$ is the unique-SVP solution when both $\boldsymbol{s}$ and $\boldsymbol{e}$ are reasonably short. For exmaple, as shown in [7], the attack is successful if and only if $\sigma\sqrt{b} \leq \delta^{2b-d-1} \times q^{m/d}$, where σ is the standard deviation of the errors and secrets, $\delta = ((\pi b)^{1/b} b/2\pi e)^{1/(2(b-1))}$.

BKZ algorithm progressively processes the lattice basis by calling polynomial times a subroutine, such as the (quantum) sieving algorithm, to solve the exact shortest vector problem for sub-lattices with dimension (i.e. blocksize) b. This method is known as BKZ-core-(Q)Sieving, and its complexity depends solely on the block dimension b that is required for the BKZ algorithm to find the unique solution. According to [7], the best complexity of the SVP oracle is $\sqrt{3/2}^{b+o(b)} \approx 2^{0.292b}$ for classical sieving algorithms, and $\sqrt{13/9}^{b+o(b)} \approx 2^{0.265b}$ for quantum sieving algorithms.

Dual attack. In a dual attack, one firstly tries to build a dual lattice of the aforementioned primal lattice, and then uses the dual lattice to solve the decisional LWE problem. At a high level, given the LWE instance $(\boldsymbol{A}, \boldsymbol{b} = \boldsymbol{A}\boldsymbol{s} + \boldsymbol{e})$, $\boldsymbol{A} \in \mathbb{Z}_q^{m \times n}$, the target lattice of dimension $d = m + n$ is constructed as

$$\Lambda_{\boldsymbol{A}}^{\perp} = \{(\boldsymbol{x}, \boldsymbol{y}) \in \mathbb{Z}^m \times \mathbb{Z}^n : \boldsymbol{A}^t \boldsymbol{x} = \boldsymbol{y} \mod q\}.$$

Again, [7] showed that BKZ is capable of finding a vector $\boldsymbol{v} = (\boldsymbol{x}, \boldsymbol{y})$ of length $l = \delta^{d-1} q^{n/d}$, where the distance between $\boldsymbol{v}^t \boldsymbol{b}$ and the uniform distribution will be bounded by $\epsilon = 4\exp(-2\pi^2\tau^2)$ for $\tau = l\sigma/q$. This breaks the decisional LWE problem with an advantage ϵ.

Similar to primal attacks, the concrete security of dual attack also depends on the complexity of BKZ algorithm. There is a slight caveat when BKZ-core-QSieving is used: the attacker is able to amplify ϵ to $1/2$ by repeating the sieving algorithm for $R = max(1, 1/(\gamma\epsilon^2))$ times. This operation is almost free to the attacker, since sieving algorithm will produce $\gamma = 2^{0.2075b}$ vectors which far exceed the required number of short vectors $1/\epsilon^2$ for repeating.

Security Estimates. We use BKZ simulator with core-(Q)sieving model to estimate the security for our scheme. The required blocksize to achieve our target root Hermite factor is shown in Table 2. The corresponding security is then estimated for the obtained blocksize. Note that in [3], Albrecht *et al.* independently evaluated the security for all (Ring-)LWE candidates, and their estimation matches ours for LAC.

We remark that, for the parameters of LAC, hybrid attacks [31,47,29] that used to evaluate the security of NTRU or LWE problem with particularly small (e.g., binary, ternary) or sparse vectors will not outperform the core-(Q)sieving model. Briefly, the main idea of hybrid attack is to split the the lattice into two parts, one part is evaluated by using search algorithm (meet in the middle or quantum search), the other part is evaluated by solving the BDD problem. According to the result in [31,47,29] the number of necessary guesses can be reduced to the square root of the number of guesses needed in a naive brute-force approach by using the meet in the middle or quantum search algorithm. Thus, the cost of the search part can be roughly evaluated by half of the entropy of the target vector. It is easy to verify that, the secret and error vectors in LAC (see 5.2 for detail) have enough entropy to resist hybrid attacks.

6.2 Dedicated Attacks

We stress again that the two attacks we are about to discuss does not perform better than generic attacks. Specifically, although we revised the parameters partially due to the threat of high Hamming weight attack, such a revision is only for conservative purpose and the attack itself does not work on both the original parameter sets and the revised ones.

Subfield Attacks. The idea of exploiting subfields is known to the lattice community for years [9,2,11,34], and to use this idea to analyze LAC was firstly proposed by Alperin-Sheriff [39] during the first round evaluation of NIST-PQC. Recall that $x^n + 1$ has two factors modulo $q = 251$:

$$x^n + 1 = (x^{n/2} + 91x^{n/4} + 250)(x^{n/2} + 160x^{n/4} + 250).$$

In other words, there exist two subfields defined by two polynomials $\boldsymbol{g}$ and $\boldsymbol{h}$ where $\boldsymbol{g} = x^{n/2} + 91x^{n/4} + 250$ and $\boldsymbol{h} = x^{n/2} + 160x^{n/4} + 250$.

Given $(\boldsymbol{a}, \boldsymbol{b} = \boldsymbol{as} + \boldsymbol{e})$, one may recover $(\boldsymbol{s}, \boldsymbol{e})$ by looking at the samples over the subfields. It may be sufficient to recover $(\boldsymbol{s}_g := \boldsymbol{s} \bmod \boldsymbol{g}, \boldsymbol{e}_g := \boldsymbol{e} \bmod \boldsymbol{g})$ from $(\boldsymbol{a} \bmod \boldsymbol{g}, \boldsymbol{b} \bmod \boldsymbol{g})$, and $(\boldsymbol{s}_h, \boldsymbol{e}_h$, respectively). Next, it becomes straightforward to recover $(\boldsymbol{s}, \boldsymbol{e})$ via Chinese remainder theorem.

Analysis. In the rest, we give a full analysis of this attack. The key point of the attack is that by moving to the subfield, the lattice dimension is practically halved. Therefore, the BKZ complexity may be reduced for the new sub-lattices. Note that this is not necessarily always the case under core-(Q)Sieving model where only the cost of subroutine counts; and the cost of the subroutine depends only on the root Hermite factor. Nonetheless, to have a meaningful analysis, we assume that this is not an obstacle: the attacker may access an SVP oracle for BKZ subroutines solely for this attack.

Our analysis will show that the corresponding vectors in the subfields, $(\boldsymbol{s}_g, \boldsymbol{e}_g)$, will be larger than the Gaussian heuristic length. In other words, even if one was able to perform lattice reduction over the dimension-halved lattices, he will not be able to recover the desired vectors.

The attack reduces the dimension, in the meantime, the modulo operation increases the size of $(\boldsymbol{s}_g, \boldsymbol{e}_g)$ (similarly, $(\boldsymbol{s}_h, \boldsymbol{e}_h)$). To be precise, when $(\boldsymbol{s}, \boldsymbol{e})$ are small polynomials with the coefficients in $\{-1, 0, 1\}$, the coefficients of $(\boldsymbol{s}_g, \boldsymbol{e}_g)$ will lie in $\{0, \pm 1, \pm 2, \pm 91\}$. Coefficients of ± 91 will be too large. Alperin-Sheriff also pointed out that by multiplying $\boldsymbol{s}$ and $\boldsymbol{e}$ by 11, all the coefficients of $(\boldsymbol{s}_g, \boldsymbol{e}_g)$ will be within the interval of $[-25, 25]$.

Let $\boldsymbol{A} = [\boldsymbol{A}_g | \boldsymbol{I} | 11 \times \boldsymbol{b}_g]$, where $\boldsymbol{A_g}$ is the matrix generated by $\boldsymbol{a}_g$, if $\boldsymbol{z} = [11 \times \boldsymbol{s}_g | 11 \times \boldsymbol{e}_g | - 1]$ is the shortest solution of $\boldsymbol{Az} = 0 \bmod q$, we can recover $\boldsymbol{z}$ with the primal attack. Note that, the dimension of a primal attack is reduced from $d = 2n+1$ to $d = n+1$ via the subfield attack. Since $\boldsymbol{A}$ is a random matrix, the q-ary lattice $\Lambda_q^{\perp}(\boldsymbol{A})$ will behave as a random lattice [19], and therefore it is sufficient to use Gaussian heuristic to estimate the length of shortest vectors in this lattice:

$$\lambda_1(\Lambda_q^{\perp}) \approx q^{m/d} \sqrt{\frac{d}{2\pi e}}.$$

In the case of $n = 512$ and $n = 1024$, the lengths of the shortest vector is expected at 86.36 and 122.4, respectively.

On the other hand, we also need to estimate the length of $\boldsymbol{z}$. Central limit theory says that the length of $\boldsymbol{z}$ approximately follows a discrete Gaussian distribution. Our implementation shows that $\boldsymbol{z}$ closely follows a Gaussian distribution with a mean and deviation pair of $(253.59, 6.9)$ for LAC-128, $(253.26, 6.29)$ for LAC-192 and $(358.42, 6.86)$ for LAC-256[1].

It is easy to verify that, the length of $\boldsymbol{z}$ will be larger than the solution of $\boldsymbol{Az} = 0 \bmod q$ except for negligible probability. Hence $\boldsymbol{z}$ will not be a short vector in this lattice. In other words, if one were to use subfield attack, and assuming that they have free access to SVP oracles simply for the sub-lattices, they will not be able to locate the vector.

To sum up, the subfield attack described above will not affect the security of LAC for either original parameter sets or the revised version.

High Hamming Weight Attack. This is a chosen ciphertext attack that exploits the fact that the secrets and errors $(\boldsymbol{r}, \boldsymbol{e}_1)$ in some ciphertexts (with certain probability) may have higher-than-usual Hamming weight. It is feasible if $(\boldsymbol{r}, \boldsymbol{e}_1)$ are randomly selected from Ψ_1 or $\Psi_{\frac{1}{2}}$. It is easy to see that the decryption error rate is influenced by the Hamming weight. Therefore, with enough number of random samples, an attacker may obtain sufficient number of samples whose

[1] The data is obtained over 100,000 random samples for each parameter set using SageMath. The experiment does not mean to extensive to show any proof of statistical distances; the mean is obviously much higher than Gaussian heuristic length.

secrets and errors have very higher Hamming weight, and then invoke the decryption oracle to extract information of the private key.

In [21] D'Anvers et al. describe the attempt of high Hamming weight attacks against several NIST PQC candidates, and it turns out that the decryption error estimation of LAC-256 of the first version of LAC [36] is affected. To immune the attack, we use n-ary centered binomial distribution with fixed Hamming weight. Thus, the current parameters sets will not be affected by the attack.

Although, fixed hamming weight centered binomial distribution makes LAC immune high hamming attack, we also give a detailed analysis of the scenario when standard centered binomial distribution is used as in the Round 1 version of LAC. The analysis result shows that, with suitable BCH parameters, LAC is still immune high hamming attack even when standard centered binomial distribution is used.

Analysis. It has been shown that chosen plaintext secure version of (Ring-)LWE based schemes suffer from an reaction attack [24]. To address this vulnerability, most schemes rely on Fujisaki-Okamoto transformation [27,28,32] to achieve chosen ciphertext security. We also adopt the same approach. Via this transformation, the randomness vectors $(\boldsymbol{r}, \boldsymbol{e}_1)$ are generated from the plaintext message by a pseudorandom generator. Thus the vectors $(\boldsymbol{r}, \boldsymbol{e}_1)$ are randomly distributed from the view of the adversary.

In a comment to the Round 1 version of LAC [1], Alperin-Sheriff showed that, for the LAC-256 parameter set, the probability that a pair of valid $(\boldsymbol{r}, \boldsymbol{e}_1)$ with a Hamming weight of at least $1024 + 310 = 1334$ is greater than

$$\binom{2048}{1334} / 2^{2048} = 2^{-143}.$$

Therefore, with 2^{207} pre-computations (assuming each access to the pseudorandom generator incurs a cost of 1), the adversary will obtain 2^{64} messages for which the corresponding $(\boldsymbol{r}, \boldsymbol{e}_1)$ have Hamming weight exceeding 1334. It is worth noting that the adversary only needs to access the decryption oracle for 2^{64} times in this setting. Next, for samples with such high Hamming weights, the decryption error rate for each bit of $\widetilde{m}_i$ is expected at

$$\delta_{high} \approx 1 - \mathsf{erf}\left(\frac{\lfloor 251/4 \rceil}{\sqrt{2}((1/\sqrt{2})\sqrt{(1024+310)/2048}\sqrt{2 \times 1024})}\right) \approx 2^{-5.9},$$

This yields a decryption error rate for the message $\boldsymbol{m}$:

$$\Delta_{high} = \sum_{j=55+1}^{1023} \left(\binom{1023}{j} \delta_{high}^{j} (1 - \delta_{high})^{1023-j} \right) \approx 2^{-44.4}.$$

As a result, with 2^{207} pre-computations and 2^{64} decryption oracle queries, the adversary can get about $2^{19.6}$ decryption failures. We remark that, 1334 is a lower bound of the Hamming weight for decryption errors. Decryption errors may occur for any Hamming weight above 1334, and therefore the adversary may get (a little) more than $2^{19.6}$ decryption failures if they were to perform all above (pre-)computations.

Remark 2. We argue that, as also pointed by D'Anvers [1], it is difficult to get any information about the private key from these decryption failures. All the information that an adversary may learn is whether there are more than l_t errors in the code word; they cannot determine which coefficients are failing as in a reaction attack [24].

Following the above example, with a message size of 256, the BCH code can correct up to $l_t = 100$ errors for the code length of 1023. Consequently, the decryption error rate for high Hamming weight random vectors $\boldsymbol{r}, \boldsymbol{e}_1$ is estimated as:

$$\Delta_{high} = \sum_{j=100+1}^{1023} \left(\binom{1023}{j} \delta_{high}^{j} (1 - \delta_{high})^{1023-j} \right) \approx 2^{-147}.$$

As a result, with 2^{64} message queries, the probability that the adversary gets one decryption failure is around 2^{-83}. In other words, it will take the adversary over 2^{256} operations to get a single decryption error.

However, we notice that, the decoding efficiency decreases drastically when large l_t is used. To resolve this problem, for LAC-256 we use the D2 error correction code [12,7] together with the BCH code. That is, the message is firstly encoded with BCH, then the code word is encoded with D2. As a result, the BCH code only need to correct 30 bits of errors.

The upper bound of the decryption error rate in the case of high Hamming weight attack is presented as follow. We give the upper bound of the Hamming weight that the adversary can obtain after 2^l operations of pre-computation, where l is the security level. Then we estimate the bit error rate and decryption error rate according to this upper bound of Hamming weight. It is clear that, for each parameter set, the decryption failure occurs with a negligible probability in the security parameter.

Categories	Ham$(\boldsymbol{r}, \boldsymbol{e}_1)$	Prob	Bit Error Rate	BCH	Decryption Error Rate
LAC-128	512+206	2^{-128}	$2^{-9.59}$	[511,256,61]	2^{-133}
LAC-192	512+333	2^{-192}	$2^{-14.75}$	[511,256,31]	2^{-142}
LAC-256	1024+416	2^{-256}	$2^{-9.77}$	[511,256,61]	2^{-138}

Ham$(\boldsymbol{r}, \boldsymbol{e}_1)$ denotes the Hamming weight of $(\boldsymbol{r}, \boldsymbol{e}_1)$, Prob denotes the probability that the adversary obtains $(\boldsymbol{r}, \boldsymbol{e}_1)$ with target Hamming weight in pre-computation.

Table 3. Decryption error rate of high Hamming attack

7 Implementation and Performance

As mentioned earlier, an important difference between LAC and previous Ring-LWE based public key encryption schemes is that our parameters do not support NTT. In this section, we present some highlights of our customized implementation, including a generalized polynomial multiplication method (as per NIST-PQC's request) and an optimized version based on AVX2 instructions.

7.1 Polynomial Multiplication

Polynomial multiplication is the most time consuming operation in the implementation of LAC. In addition to a reference implementation, we provide two optimized versions as follows:

- **General Optimized Version:** Our main observation is that, since $\mathbf{s}$ and $\mathbf{r}$ are selected from $\{-1, 0, 1\}$, the multiplication operation can be implemented by bitwise logical `AND` operation as $a_i \times 1 = a_i$&0xff and $a_i \times 0 = a_i$&0x00. Further more, we can pack 4 items into one `uint64_t` data type.
  With $q < 256$, it is possible to pack 8 coefficients into a single `uint64_t` unit, in theory. We choose to hold 4 coefficients at a time, and use the free space as a buffer for the carriers, so that we are not obliged to perform mod reductions after every arithmetic operation. This yields better performance in practice.
- **AVX2 Based Version:** AVX2 allows us to handle 256 bits data type. We are able to store 32 coefficients in a single `__mm256` data type, and utilize `_mm256_maddubs_epi16` instruction which does 32 multiplications and adjacent addition operation in a single operation. We obtain approximately 30x acceleration with this optimization.

7.2 Error Correction

We choose BCH as the error correction code for LAC. We use the generic implementation of bch algorithm by Ivan Djelic from https://github.com/jkent/python-bchlib/tree/master/bchlib. To compile it on both Linux and Windows, we remove the header file "endian.h" included in "bch.c",

and provide the implementation of the htobe32(x) function and the definition of "EBADMSG" directly in "bch.c".

Since the decoding time of the error correction code may vary and reflect the number of errors, it may cause a timing attack. To avoid the attack, we provide a constant time implementation of the BCH code, and can be used in situations that a side channel attack should be resisted. The constant time implementation of BCH can be enabled by defining "BCH_CONSTANT_TIME" in "lac-param.h". Generic more efficient implementation of BCH will be used if "BCH_CONSTANT_TIME" is not defined in "lac-param.h".

7.3 Computational Performance of the Optimized Version

LAC has been implemented in C language for the Intel x64 processor. In this section, we test the performance of Optimized version of LAC on the platform of ubuntu 16.04 operation system running on the Intel Core-i7-4770S (Haswell) @ 3.10GHz, memory 7.6GB, with Turbo Boost and Hyperthreading disabled. In each case we provide two values to describe the performance of the algorithm the CPU cycles and the microseconds.

Categories	Key generation		Encryption		Decryption		Decryption(Const-BCH)	
	CPUCycles	Times	CPUCycles	Times	CPUCycles	Times	CPUCycles	Times
LAC128	124915	40.28	194118	67.24	81187	26.28	122355	39.47
LAC192	335083	106.20	438204	144.63	292243	93.80	309896	93.80
LAC256	382627	124.23	636997	204.80	302890	95.18	338993	108.25

Table 4. Performance of LAC.CPA

Categories	Key generation		Encapsulation		Decapsulation		Decapsulation(Const-BCH)	
	CPUCycles	Times	CPUCycles	Times	CPUCycles	Times	CPUCycles	Times
LAC128	122691	39.67	209201	65.71	280125	88.07	323221	102.70
LAC192	333649	105.63	445696	145.48	731472	235.42	759871	244.49
LAC256	377123	123.59	643024	208.71	916835	297.01	934385	304.82

Table 5. Performance of LAC.CCA

7.4 Computational Performance of the AVX2 Based Version

LAC has been implemented in C language for the Intel x64 processor. In this section, we test the performance of AVX2 based version of LAC on the platform of ubuntu 16.04 operation system running on the Intel Core-i7-4770S (Haswell) @ 3.10GHz, memory 7.6GB. In each case we provide two values to describe the performance of the algorithm the cpucycles and the microseconds.

Categories	Key generation		Encryption		Decryption		Decryption(Const-BCH)	
	CPUCycles	Times	CPUCycles	Times	CPUCycles	Times	CPUCycles	Times
LAC128	61242	19.98	80173	25.91	25004	7.83	64238	20.77
LAC192	120528	38.87	130286	42.34	63266	26.41	134289	39.95
LAC256	136313	54.23	191543	63.14	72326	30.56	112654	48.99

Table 6. Performance of LAC.CPA with AVX2

Categories	Key generation		Encapsulation		Decapsulation		Decapsulation(Const-BCH)	
	CPUCycles	Times	CPUCycles	Times	CPUCycles	Times	CPUCycles	Times
LAC128	59584	19.59	89055	28.86	103229	39.26	140221	45.57
LAC192	119246	36.94	137653	65.14	224249	71.52	320135	77.32
LAC256	135780	53.95	207938	87.88	343335	84.21	359209	97.60

Table 7. Performance of LAC.CCA with AVX2

7.5 Key and Ciphertext Size

In this section the key size and ciphertext size of LAC are listed as following.

Categories	pk size(bytes)	sk size (bytes)	ciphertext size (bytes)
LAC128	544	1056	712
LAC192	1056	2080	1188
LAC256	1056	2080	1424

Table 8. Key and Ciphertext size of LAC.CPA and LAC.CCA

Categories	LAC.KE		LAC.AKE	
	Alice (bytes)	Bob (bytes)	Alice (bytes)	Bob (bytes)
LAC128	544	712	1256	1424
LAC192	1056	1188	2244	2376
LAC256	1056	1424	2480	2848

Table 9. Message size of LAC.KE and LAC.AKE

7.6 Memory Cost of Error Correction Code

The parameters bch_control defined for the encode and decode algorithm of the BCH error correction code is the most memory consuming part of LAC. In the optimized implementation and the AVX2 based implementation, these parameters are statistically defined in "bch128.h", "bch192.h" and "bch256.h" for the security strength categories LAC128, LAC192 and LAC256 respectively. Concrete memory cost of these parameters are listed as follows.

Categories	BCH Parameters			bch_control(bytes)
	code length	data length	maximum error	
LAC128	511	256	16	23736
LAC192	511	256	8	15048
LAC256	511	256	16	23736

Table 10. Memory cost of error correction code

8 Known Answer Test Values

There are four types of known answer tests (KATs):

- LAC.CPA
- LAC.CCA
- LAC.KE
- LAC.AKE

We test the three sets of parameters described as in Table 1 for each type. We use the source code offered by NIST at `https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Post-Quantum-Cryptography-Standardization/Example-Files` to generate KATs.

8.1 LAC.CPA

In this section, we test LAC.CPA.KG (Algorithm 1), LAC.CPA.ENC (Algorithm 2), and LAC.CPA.DEC (Algorithm 3). The name of the test file is "PQCencryptKAT_*", and '*' means the size of secret key. In the request file, inputs are as follows,

- "seed" denotes the seed which will be used to generate random bytes in every algorithm.
- "msg" denotes the message which will be encrypted by the encryption algorithm.
- "mlen" denotes the size of message.

In the response file, the outputs are as follows:

- "pk" denotes the public key.
- "sk" denotes the secret key.
- "c" denotes the result of LAC.CPA.ENC(PK, PLAINTEXT).
- "clen" denotes the size of ciphertext.

8.2 LAC.CCA

In this section, we test LAC.CCA.KG, LAC.CCA.ENC (Algorithm 4), and LAC.CCA.DEC (Algorithm5). The name of the test file is "PQCkemKAT_*", and '*' means the size of secret key. In the request file, the input is as follows,

- "seed" denotes the seed which will be used to generate random bytes in every algorithm.

In the response file, outputs are as follows,

- "pk" denotes the public key.
- "sk" denotes the secret key.
- "ct" denotes "$\mathbf{c}$" in LAC.CCA.Enc algorithm 4.
- "ss" denotes "K" in LAC.CCA.Enc algorithm 4.

8.3 LAC.KE

In this section, we test the processes of key exchange between Alice and Bob as described in Figure 2. The name of the test file is "PQCkeKAT_*", and '*' means the size of secret key. In the request file, the input is as follows,

- "seed" denotes the seed which will be used to generate random bytes in every algorithm.

In the response file, outputs are as follows,

- "pk" denotes the public key.
- "sk" denotes the secret key.
- "c" denotes the result sent from Bob to Alice.
- "k" denotes Bob's and Alice's session key.

8.4 LAC.AKE

In this section, we test the processes of authenticated key exchange between Alice and Bob as described in Figure 3. The name of the test file is "PQCakeKAT_*", and '*' means the size of secret key. In the request file, the input is as follows,

- "seed" denotes the seed which will be used to generate random bytes in every algorithm.

In the response file, outputs are as follows,

- "pk_a" denotes Alice's public key.
- "sk_a" denotes Alice's secret key.
- "pk_b" denotes Bob's public key.
- "sk_b" denotes Bob's secret key.
- "pk" denotes the public key.
- "sk" denotes the secret key.
- "c_a" denotes "$\mathbf{c}_1$" sent from Alice to Bob in Figure 3.
- "c_b" denotes "$(\mathbf{c}_2, \mathbf{c}_3)$" sent from Bob to Alice in Figure 3.
- "k" denotes Bob's and Alice's session key.

9 The Advantages and Limitations

9.1 Advantages

Implementation aspects:

- LAC can be implemented to run at high speeds on the Intel x64 processors that support the AVX2 instructions.
- LAC can be implemented to run at high speeds on ARM processors that support vector instructions such as NEON.
- The main operation of LAC is parallel by design, it is very suitable to be implemented on multi-core processors.

Simplicity of Design:

- The design rationale of LAC is very simple: use small modulus to cut down the size of the ciphertexts and keys.
- The distributions of the errors and secrets are very simple and easy to sample.
- The main operation of LAC is polynomial multiplication which is very easy to understand.

Variable security strength categories:

- Four security strength categories of LAC can be easily get by combining the dimensions and error distributions.
- All of the four security strength categories share the same modulus.

Flexibility:

- The error correction algorithm can be easily replaced for different requirement of the error rate.
- The security strength can be flexibly adjusted by setting the number of zeros in the errors and secrets distributions.

9.2 Limitations

The limitations of LAC:

- Can not be sped up by using the NTT technique, which effects its computational performance on processors that do not support vector instructions.
- The correctness of the decryption algorithm is guaranteed by powerful error correction code such as the BCH code.
- To avoid a timing attack, the implementation of the error correction code should be constant-time, which affects the computational efficiency.

References

1. Nist post-quantum cryptography project. https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Round-1-Submissions
2. Albrecht, M.R., Bai, S., Ducas, L.: A subfield lattice attack on overstretched NTRU assumptions - cryptanalysis of some FHE and graded encoding schemes. In: Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part I. pp. 153–178 (2016), https://doi.org/10.1007/978-3-662-53018-4_6
3. Albrecht, M.R., Curtis, B.R., Deo, A., Davidson, A., Player, R., Postlethwaite, E.W., Virdia, F., Wunderer, T.: Estimate all the {LWE, NTRU} schemes! In: Security and Cryptography for Networks - 11th International Conference, SCN 2018, Amalfi, Italy, September 5-7, 2018, Proceedings. pp. 351–367 (2018), https://doi.org/10.1007/978-3-319-98113-0_19
4. Albrecht, M.R., Faugère, J., Fitzpatrick, R., Perret, L.: Lazy modulus switching for the BKW algorithm on LWE. In: Public-Key Cryptography - PKC 2014 - 17th International Conference on Practice and Theory in Public-Key Cryptography, Buenos Aires, Argentina, March 26-28, 2014. Proceedings. pp. 429–445 (2014), https://doi.org/10.1007/978-3-642-54631-0_25
5. Albrecht, M.R., Göpfert, F., Virdia, F., Wunderer, T.: Revisiting the expected cost of solving usvp and applications to LWE. In: Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I. pp. 297–322 (2017), https://doi.org/10.1007/978-3-319-70694-8_11
6. Albrecht, M.R., Player, R., Scott, S.: On the concrete hardness of learning with errors. J. Mathematical Cryptology 9(3), 169–203 (2015), http://www.degruyter.com/view/j/jmc.2015.9.issue-3/jmc-2015-0016/jmc-2015-0016.xml
7. Alkim, E., Ducas, L., Pöppelmann, T., Schwabe, P.: Newhope without reconciliation. IACR Cryptology ePrint Archive 2016, 1157 (2016), http://eprint.iacr.org/2016/1157
8. Alkim, E., Ducas, L., Pöppelmann, T., Schwabe, P.: Post-quantum key exchange - A new hope. In: 25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016. pp. 327–343 (2016), https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/alkim
9. Bernstein, D.: A subfield-logarithm attack against ideal lattices (2014), https://blog.cr.yp.to/20140213-ideal.html
10. Bernstein, D.J., Chuengsatiansup, C., Lange, T., van Vredendaal, C.: NTRU prime: Reducing attack surface at low cost. In: Selected Areas in Cryptography - SAC 2017 - 24th International Conference, Ottawa, ON, Canada, August 16-18, 2017, Revised Selected Papers. pp. 235–260 (2017), https://doi.org/10.1007/978-3-319-72565-9_12
11. Biasse, J., Espitau, T., Fouque, P., Gélin, A., Kirchner, P.: Computing generator in cyclotomic integer rings - A subfield algorithm for the principal ideal problem in $l_{|\Delta K|}(\frac{1}{2})$ and application to the cryptanalysis of a FHE scheme. In: Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part I. pp. 60–88 (2017), https://doi.org/10.1007/978-3-319-56620-7_3
12. Bos, J.W., Costello, C., Naehrig, M., Stebila, D.: Post-quantum key exchange for the TLS protocol from the ring learning with errors problem. In: 2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015. pp. 553–570 (2015), https://doi.org/10.1109/SP.2015.40
13. Bos, J.W., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Schwabe, P., Stehlé, D.: CRYSTALS - kyber: a cca-secure module-lattice-based KEM. IACR Cryptology ePrint Archive 2017, 634 (2017), http://eprint.iacr.org/2017/634
14. Bos, J.W., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Schwabe, P., Stehlé, D.: Crystals-kyber: a cca-secure module-lattice-based kem. IACR Cryptology ePrint Archive 2017, 634 (2017)
15. Brakerski, Z., Vaikuntanathan, V.: Fully homomorphic encryption from ring-lwe and security for key dependent messages. In: Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings. pp. 505–524 (2011), https://doi.org/10.1007/978-3-642-22792-9_29
16. Bruinderink, L.G., Hülsing, A., Lange, T., Yarom, Y.: Flush, gauss, and reload - A cache attack on the BLISS lattice-based signature scheme. In: Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings. pp. 323–345 (2016), https://doi.org/10.1007/978-3-662-53140-2_16

17. Canetti, R., Krawczyk, H.: Analysis of key-exchange protocols and their use for building secure channels. In: Advances in Cryptology - EUROCRYPT 2001, International Conference on the Theory and Application of Cryptographic Techniques, Innsbruck, Austria, May 6-10, 2001, Proceeding. pp. 453–474 (2001), https://doi.org/10.1007/3-540-44987-6_28
18. Chen, Y., Nguyen, P.Q.: BKZ 2.0: Better lattice security estimates. In: Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings. pp. 1–20 (2011), https://doi.org/10.1007/978-3-642-25385-0_1
19. Daniele Micciancio, O.R.: Lattice-based cryptography. Tech. rep., http-s://cims.nyu.edu/ regev/papers/pqc.pdf (2008)
20. D'Anvers, J., Vercauteren, F., Verbauwhede, I.: The impact of error dependencies on ring/mod-lwe/lwr based schemes. IACR Cryptology ePrint Archive 2018, 1172 (2018), https://eprint.iacr.org/2018/1172
21. D'Anvers, J., Vercauteren, F., Verbauwhede, I.: On the impact of decryption failures on the security of LWE/LWR based schemes. IACR Cryptology ePrint Archive 2018, 1089 (2018), https://eprint.iacr.org/2018/1089
22. Ding, J.: A simple provably secure key exchange scheme based on the learning with errors problem. IACR Cryptology ePrint Archive 2012, 688 (2012), http://eprint.iacr.org/2012/688
23. Ducas, L., Durmus, A., Lepoint, T., Lyubashevsky, V.: Lattice signatures and bimodal gaussians. In: Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I. pp. 40–56 (2013), https://doi.org/10.1007/978-3-642-40041-4_3
24. Fluhrer, S.R.: Cryptanalysis of ring-lwe based key exchange with key share reuse. IACR Cryptology ePrint Archive 2016, 85 (2016), http://eprint.iacr.org/2016/085
25. Fujioka, A., Suzuki, K., Xagawa, K., Yoneyama, K.: Strongly secure authenticated key exchange from factoring, codes, and lattices. In: Public Key Cryptography - PKC 2012 - 15th International Conference on Practice and Theory in Public Key Cryptography, Darmstadt, Germany, May 21-23, 2012. Proceedings. pp. 467–484 (2012), https://doi.org/10.1007/978-3-642-30057-8_28
26. Fujioka, A., Suzuki, K., Xagawa, K., Yoneyama, K.: Practical and post-quantum authenticated key exchange from one-way secure key encapsulation mechanism. In: 8th ACM Symposium on Information, Computer and Communications Security, ASIA CCS '13, Hangzhou, China - May 08 - 10, 2013. pp. 83–94 (2013), http://doi.acm.org/10.1145/2484313.2484323
27. Fujisaki, E., Okamoto, T.: How to enhance the security of public-key encryption at minimum cost. In: Public Key Cryptography, Second International Workshop on Practice and Theory in Public Key Cryptography, PKC '99, Kamakura, Japan, March 1-3, 1999, Proceedings. pp. 53–68 (1999), https://doi.org/10.1007/3-540-49162-7_5
28. Fujisaki, E., Okamoto, T.: Secure integration of asymmetric and symmetric encryption schemes. J. Cryptology 26(1), 80–101 (2013), https://doi.org/10.1007/s00145-011-9114-1
29. Göpfert, F., van Vredendaal, C., Wunderer, T.: A hybrid lattice basis reduction and quantum search attack on LWE. In: Post-Quantum Cryptography - 8th International Workshop, PQCrypto 2017, Utrecht, The Netherlands, June 26-28, 2017, Proceedings. pp. 184–202 (2017), https://doi.org/10.1007/978-3-319-59879-6_11
30. Hofheinz, D., Hövelmanns, K., Kiltz, E.: A modular analysis of the fujisaki-okamoto transformation. In: Theory of Cryptography - 15th International Conference, TCC 2017, Baltimore, MD, USA, November 12-15, 2017, Proceedings, Part I. pp. 341–371 (2017), https://doi.org/10.1007/978-3-319-70500-2_12
31. Howgrave-Graham, N.: A hybrid lattice-reduction and meet-in-the-middle attack against NTRU. In: Advances in Cryptology - CRYPTO 2007, 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007, Proceedings. pp. 150–169 (2007), https://doi.org/10.1007/978-3-540-74143-5_9
32. Jiang, H., Zhang, Z., Chen, L., Wang, H., Ma, Z.: Ind-cca-secure key encapsulation mechanism in the quantum random oracle model, revisited. In: Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part III. pp. 96–125 (2018), https://doi.org/10.1007/978-3-319-96878-0_4
33. Jin, Z., Zhao, Y.: Optimal key consensus in presence of noise. CoRR abs/1611.06150 (2016), http://arxiv.org/abs/1611.06150
34. Kirchner, P., Fouque, P.: Revisiting lattice attacks on overstretched NTRU parameters. In: Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part I. pp. 3–26 (2017), https://doi.org/10.1007/978-3-319-56620-7_1

35. Lindner, R., Peikert, C.: Better key sizes (and attacks) for lwe-based encryption. In: Topics in Cryptology - CT-RSA 2011 - The Cryptographers' Track at the RSA Conference 2011, San Francisco, CA, USA, February 14-18, 2011. Proceedings. pp. 319–339 (2011), https://doi.org/10.1007/978-3-642-19074-2_21
36. Lu, X., Liu, Y., Jia, D., Xue, H., He, J., Zhang, Z.: Lac. Tech. rep., https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/round-1/submissions/LAC.zip (2017)
37. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30 - June 3, 2010. Proceedings. pp. 1–23 (2010), https://doi.org/10.1007/978-3-642-13190-5_1
38. Micciancio, D., Mol, P.: Pseudorandom knapsacks and the sample complexity of LWE search-to-decision reductions. In: Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings. pp. 465–484 (2011), https://doi.org/10.1007/978-3-642-22792-9_26
39. NIST: NIST PQC FORUM: LAC, https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/round-1/official-comments/LAC-official-comment.pdf
40. Peikert, C.: Lattice cryptography for the internet. In: Post-Quantum Cryptography - 6th International Workshop, PQCrypto 2014, Waterloo, ON, Canada, October 1-3, 2014. Proceedings. pp. 197–219 (2014), https://doi.org/10.1007/978-3-319-11659-4_12
41. Peikert, C.: How (not) to instantiate ring-lwe. In: Security and Cryptography for Networks - 10th International Conference, SCN 2016, Amalfi, Italy, August 31 - September 2, 2016, Proceedings. pp. 411–430 (2016), https://doi.org/10.1007/978-3-319-44618-9_22
42. Peikert, C., Regev, O., Stephens-Davidowitz, N.: Pseudorandomness of ring-lwe for any ring and modulus. In: Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017. pp. 461–473 (2017), http://doi.acm.org/10.1145/3055399.3055489
43. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005. pp. 84–93 (2005), http://doi.acm.org/10.1145/1060590.1060603
44. Rosca, M., Stehlé, D., Wallet, A.: On the ring-lwe and polynomial-lwe problems. In: Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part I. pp. 146–173 (2018), https://doi.org/10.1007/978-3-319-78381-9_6
45. Saarinen, M.J.O.: On reliability, reconciliation, and error correction in ring-lwe encryption. IACR Cryptology ePrint Archive 2017, 424 (2017), http://eprint.iacr.org/2012/688
46. Schmidt, M., Bindel, N.: Estimation of the hardness of the learning with errors problem with a restricted number of samples. IACR Cryptology ePrint Archive 2017, 140 (2017), http://eprint.iacr.org/2017/140
47. Wunderer, T.: Revisiting the hybrid attack: Improved analysis and refined security estimates. IACR Cryptology ePrint Archive 2016, 733 (2016), http://eprint.iacr.org/2016/733