LAC

Lattice-based **C**ryptosystems

Principal Submitter: Xianhui Lu[1]

Contact: luxianhui@iie.ac.cn, 0086-15010131069

Organizations: 1: Data Assurance and Communication Security Research Center,
Institute of Information Engineering, Chinese Academy of Sciences
2: OnBoard Security Inc.

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

Auxiliary Submitters: Yamin Liu[1], Dingding Jia[1], Haiyang Xue[1], Jingnan He[1], Zhenfei Zhang[2]

Inventors: Xianhui Lu[1], Yamin Liu[1], Dingding Jia[1], Haiyang Xue[1], Jingnan He[1], Zhenfei Zhang[2]

Owner: Xianhui Lu[1]

Alternative Contact: hejingnan@iie.ac.cn, 0086-18519196307

Signature:

Table of Contents

List of Figures

List of Tables

1 Introduction

In this document we introduce LAC (**La**ttice-based **C**ryptosystems), which comprises 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.KE: a passively secure key exchange protocol which is directly converted from LAC.CPA;
- LAC.CCA: an IND-CCA secure key encapsulation mechanism, which is obtained by applying a variant of the FO transformation [19] to LAC.CPA;
- LAC.AKE: an authenticated key exchange protocol which is obtained by applying a generic FSXY transformation [17] to LAC.CCA and LAC.CPA.

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

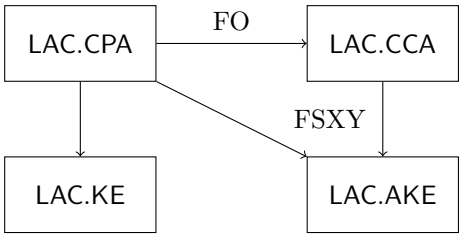

Fig. 1. LAC: **La**ttice-based **C**ryptosystems

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, which are the preliminaries needed for understanding the proposed cryptosystems. Then in Section 3, the overall design rationale is explained. In Section 4, the specifications of the algorithms are given. In Section 5, the parameter settings are given. In Section 6 and 7, the correctness analysis and the security analysis are given respectively. In Section 8, the implementation of the cryptosystems is described, followed by the performance in Section 9. Known answer test values (KATs) are described in Section 10. Finally, the pros and cons of the cryptosystems are discussed in Section 11.

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 $\mathbf{a}$, and $\mathbf{a}^t$ denotes the transposition of $\mathbf{a}$.

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

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

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

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

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

Norms. The length of vectors is measured with norms. For an m-dimensional vector $\mathbf{x} = (x_1, x_2, ..., x_m)$, its l_1-norm is defined as $\|\mathbf{x}\|_1 = \sum_{i=1}^{m} |x_i|$; the l_2-norm, also known as the Euclidean norm, is defined as $\|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^{m} x_i^2}$, or solely denoted as $\|\mathbf{x}\|$; the infinity norm is defined as $\|\mathbf{x}\|_\infty = \max |x_i|$. The length of a matrix is the norm of its longest column vector, e.g., $\|\mathbf{X}\| = \max \|\mathbf{x}_i\|$.

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 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 the ring of 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 polynomial rings are operated according to those of polynomials.

Definition 1. *The wrap-around multiplication rule in ring $R = \mathbb{Z}[x]/(x^n + 1)$ is defined as:*

$$\mathbf{h} = \mathbf{f}\mathbf{g} \Leftrightarrow h_i = \sum_{j=0}^{i} f_j g_{i-j} - \sum_{j=i+1}^{n-1} f_j g_{n+i-j}.$$

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 $\mathbf{B} = \{\mathbf{b}_1, ..., \mathbf{b}_m\} \in \mathbb{R}^{m \times m}$ can be defined as

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

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

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

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

***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 $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$, define the following full-rank m-dimensional integer q-ary lattices:

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

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

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).

Definition 2 (Search SVP). *Given a lattice basis $\mathbf{B} \in \mathbb{Z}^{m \times n}$, find $\mathbf{v} \in \mathcal{L}(\mathbf{B})$ s.t. $\|\mathbf{v}\| = \lambda_1(\mathbf{B})$.*

Definition 3 (Decisional SVP). *Given a lattice basis* $\mathbf{B} \in \mathbb{Z}^{m\times n}$*, and a rational* $r \in \mathbb{Q}$*, determine whether* $\lambda_1(\mathcal{L}(\mathbf{B})) \leq r$ *or not.*

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

Definition 4 (Search SVP$_\gamma$). *Given a lattice basis* $\mathbf{B} \in \mathbb{Z}^{m\times n}$*, and* $\gamma \geq 1$*, find* $\mathbf{v} \in \mathcal{L}(\mathbf{B})$ *s.t.* $\|\mathbf{v}\| \leq \gamma \cdot \lambda_1(\mathbf{B})$.

Definition 5 (GapSVP$_\gamma$). *For* $\gamma \geq 1$*, given a pair of input* $(\mathbf{B}, r)$*, where* $\mathbf{B} \in \mathbb{Z}^{m\times m}$ *is the basis of a full-rank* m*-dimensional lattice and* $r \in \mathbb{Q}$*, it is a* YES *instance if* $\lambda_1(\mathcal{L}(\mathbf{B})) \leq r$*, and a* NO *instance if* $\lambda_1(\mathcal{L}(\mathbf{B})) > \gamma \cdot r$.

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

Definition 7 (SIVP$_\gamma$). *For* $\gamma \geq 1$*, given* $\mathbf{B} \in \mathbb{Z}^{m\times m}$*, the basis of a full-rank* m*-dimensional lattice, output a set of* m *linearly independent lattice vectors* $\mathbf{S} \subset \mathcal{L}(\mathbb{B})$*, s.t.* $\|\mathbf{S}\| \leq \gamma \cdot \lambda_n(\mathcal{L}(\mathbf{B}))$.

The unique-SVP (uSVP) refers to solve SVP in lattices with a relatively short $\lambda_1(\Lambda)$.

Definition 8 (γ-uSVP). *Given a lattice basis* $\mathbf{B} \in \mathbb{Z}^{m\times n}$*, and* $\gamma > 1$*, s.t.* $\lambda_2(\mathcal{L}(\mathbf{B})) > \gamma\lambda_1(\mathcal{L}(\mathbf{B}))$*, find* $\mathbf{v} \in \mathcal{L}(\mathbf{B})$ *s.t.* $\|\mathbf{v}\| = \lambda_1(\mathbf{B})$.

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 [28] and its variants. The proposed cryptosystems in LAC are based on a simplified version of the learning with errors assumption over rings (ring-LWE) [21], which admits more practical implementations and has been widely used.

Definition 9 (Search LWE). *Let* n, m, q *be positive integers, and* $\chi_{\mathbf{s}}, \chi_{\mathbf{e}}$ *be distributions over* $\mathbb{Z}$*. Given* $(\mathbf{A}, \mathbf{b} = \mathbf{As} + \mathbf{e})$*, recover the secret* $\mathbf{s}$*, where* $\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{m\times n}$*, the secret* $\mathbf{s} \xleftarrow{\$} \chi_{\mathbf{s}}^n$ *and the error* $\mathbf{e} \xleftarrow{\$} \chi_{\mathbf{e}}^m$.

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

Definition 10 (Decisional LWE). *Let* n, m, q *be positive integers, and* $\chi_{\mathbf{s}}, \chi_{\mathbf{e}}$ *be distributions over* $\mathbb{Z}$*. Distinguish the following two distributions:*

- $D_0 : (\mathbf{A}, \mathbf{b})$,
- $D_1 : (\mathbf{A}, \mathbf{u})$,

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

The decisional version is polynomially equivalent to the computational case [24].

In the case of ring-LWE, the noisy equations are $(\mathbf{a}, \mathbf{b} = \mathbf{as} + \mathbf{e})$, where $\mathbf{a}$, $\mathbf{s}$, $\mathbf{e}$ 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 [9], 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$_\gamma$ 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 11 (Search Ring-LWE). *Let n, q be positive integers, and $\chi_{\mathbf{s}}, \chi_{\mathbf{e}}$ be distributions over R. Given $(\mathbf{a}, \mathbf{b} = \mathbf{as} + \mathbf{e})$, recover the secret $\mathbf{s}$, where $\mathbf{a} \xleftarrow{\$} R_q$, the secret $\mathbf{s} \xleftarrow{\$} \chi_{\mathbf{s}}$ and the error $\mathbf{e} \xleftarrow{\$} \chi_{\mathbf{e}}$.*

Definition 12 (Decisional Ring-LWE). *Let n, q be positive integers, and $\chi_{\mathbf{s}}, \chi_{\mathbf{e}}$ be distributions over R. Distinguish the following two distributions:*

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

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

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 [1]. In the design of LAC we also use centered binomial distribution with parameter 1 and $\frac{1}{2}$ (denoted as Ψ_1 and $\Psi_{\frac{1}{2}}$ respectively) as follows:

Definition 13 (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 value of Ψ_1 is 0 and the variance is $\frac{1}{2}$.*

Definition 14 ($\Psi_{\frac{1}{2}}$). *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 value of $\Psi_{\frac{1}{2}}$ is 0 and the variance is $\frac{1}{4}$.*

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 $1 \leq i \leq t$.

2.5 BCH Code

In LAC we use the BCH (**B**ose-**C**haudhuri-**H**ocquenghem) code [20,4] to deal with the error correction problem. It is a well-studied and widely-used cyclic code, and can correct multiple random errors.

A binary BCH code $\mathsf{BCH}[n_e, l_e, d_e, t_e]$ is specified by integers n_e, l_e, d_e, t_e:

- n_e: length of the codeword;
- l_e: length of the encoded message;
- d_e: the minimum distance of the code;
- t_e: the maximum number of errors, usually $t \leq \frac{d_e-1}{2}$.

There are two algorithms related to a code $\mathsf{BCH}[n_e, l_e, d_e, t_e]$:

- The encoding algorithm $\mathsf{BCHE} : \{0,1\}^{l_e} \rightarrow \{0,1\}^{n_e}$: on input a message of length l_e, output a codeword of length n_e;
- The decoding algorithm $\mathsf{BCHD} : \{0,1\}^{n_e} \rightarrow \{0,1\}^{l_e}$: on input a codeword of length n_e, recover a message of length l_e.

3 Design Rationale

Criteria taken into account in the design of LAC:

- Security:
  - Provable security based on worst-case hardness assumption;
  - Resistance against all known attacks.
- Efficiency:
  - High speed;
  - Small key size and ciphertext size.
- Flexibility: easy to set parameters for different security strength categories.

To achieve provable security based on worst-case hardness assumption, LAC is designed based on the polynomial learning with errors (poly-LWE) problem over the ring $R_q = \mathbb{Z}_q[x]/(x^n + 1)$ for n being a power of 2, which is a simple form of the ring learning with errors problem (ring-LWE) [21,23]. According to the result in [21,23,12], for this specific ring, the hardness of the simple poly-LWE problem can be reduced to the worst-case hardness of lattice problems.

In most cryptographic schemes based the ring-LWE problem, to speed up the efficiency of polynomial multiplication by using the number theoretic transform (NTT), the modulus q is chosen to be 12289 or 7681 [13,1,7]. Our main design philosophy is to set the modulus q as small as possible to improve the bandwidth efficiency. Although NTT can not be used to speed up the computational efficiency when $q < 7681$, we can use the Intel Advanced Vector Extensions instructions (AVX, AVX2, AVX512) to improve the computational efficiency. Briefly, the AVX2 instructions can be used to process multiple multiplication operations in one instruction cycle.

For the simplicity of sampling secrets and errors, we use centered binomial distribution over $\{-1, 0, 1\}$. To provide enough security strength when using small secret and error over $\{-1, 0, 1\}$, we increase the value of $\alpha = \frac{\sigma}{q}\sqrt{2\pi}$, where σ is the standard variation of the secret and error distributions. When α becomes too large, the error rate of the decryption becomes unacceptable. To deal with this problem, we use error correction code with large blocks and large code distance such as the BCH code to correct the errors.

For cryptographic schemes based on the hard learning problem, using different dimension n is the most natural approach to achieving different security strength categories. However, when the ring $R_q = \mathbb{Z}_q[x]/(x^n + 1)$ for n being a power of 2 is used, only very scarce suitable n can be selected, such as $n = 512$ or $n = 1024$. To provide different security strength categories, we use different binomial distribution over $\{-1, 0, 1\}$ to trade-off between the security strength and the ciphertext size.

It is easy to get an ephemeral-only key establishment scheme based on the basic public key encryption scheme by using traditional framework. However, the transformations to adaptive

chosen ciphertext secure public key encryption scheme and authenticated key establishment scheme need to be very careful. In this case, we use the popular FO [15,16,19] framework to get the adaptive chosen ciphertext secure public key encryption scheme, and the FSXY [17,18] 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 $R_q = \mathbb{Z}_q[x]/(x^n + 1)$. In LAC we always use the modulus $q = 251$ (the largest prime less than 2^8). Define the half of q as $\bar{q} = \lfloor \frac{q}{2} \rfloor = 125$.

Let Ψ_σ be the centered binomial distribution with σ being the parameter of the distribution, where the corresponding standard variance is $\sqrt{\frac{\sigma}{2}}$. For a positive integer n, Ψ_σ^n denotes the n independently identical distribution of Ψ_σ.

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.

Subroutines. In the subroutines dealing with the encoding and decoding of the error correction codes, ECCEnc, ECCDec, the conversion between a message $\mathbf{m} \in \{0,1\}^{l_m}$ and its encoding $\mathbf{c_m} \in \{0,1\}^{l_v}$ 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 encoding and decoding algorithms BCHE, BCHD of a code BCH$[n_e, l_e, d_e, t_e]$ are invoked in ECCEnc, ECCDec respectively.

The subroutines are detailed following the description of the algorithms of LAC.CPA, where ECCEnc, ECCDec are in Algorithm 4 and 5.

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

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

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

Ensure: A ciphertext $\mathbf{c}$.
1: $\mathbf{a} \leftarrow \mathsf{Samp}(U(R_q); \mathsf{seed}_{\mathbf{a}}) \in R_q$
2: $\mathbf{c_m} \leftarrow \mathsf{ECCEnc}(\mathbf{m}) \in \{0,1\}^{l_v}$
3: $(\mathbf{r}, \mathbf{e}_1, \mathbf{e}_2) \leftarrow \mathsf{Samp}(\Psi_\sigma^n, \Psi_\sigma^n, \Psi_\sigma^{l_v}; \mathsf{seed})$
4: $\mathbf{c}_1 \leftarrow \mathbf{ar} + \mathbf{e}_1 \in R_q$
5: $\mathbf{c}_2 \leftarrow (\mathbf{br})_{l_v} + \mathbf{e}_2 + \bar{q} \cdot \mathbf{c_m} \in \mathbb{Z}_q^{l_v}$
6: **return** $\mathbf{c} := (\mathbf{c}_1, \mathbf{c}_2) \in R_q \times \mathbb{Z}_q^{l_v}$

The algorithm LAC.CPA.Dec on input sk and a ciphertext $\mathbf{c}$, recovers the corresponding message $\mathbf{m}$.

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

Ensure: A plaintext $\mathbf{m}$.
1: $\mathbf{u} \leftarrow \mathbf{c}_1\mathbf{s} \in R_q$
2: $\mathbf{c'_m} \leftarrow \mathbf{c}_2 - (\mathbf{u})_{l_v} \in \mathbb{Z}_q^{l_v}$
3: **for** $i = 0$ to $l_v - 1$ **do**
4: **if** $\frac{q}{4} \leq \mathbf{c'}_{\mathbf{m}i} < \frac{3q}{4}$ **then**
5: $\mathbf{c}_{\mathbf{m}i} \leftarrow 1$
6: **else**
7: $\mathbf{c}_{\mathbf{m}i} \leftarrow 0$
8: **end if**
9: **end for**
10: $\mathbf{m} \leftarrow \mathsf{ECCDec}(\mathbf{c_m})$
11: **return** $\mathbf{m}$

The subroutine ECCEnc convert the message $\mathbf{m}$ into a codeword $\hat{\mathbf{c}}$ and padding $\hat{\mathbf{c}}$ with 0's (usually one 0 is enough) to be $\mathbf{c_m}$, which is in the space of $\{0,1\}^{l_v}$.

Algorithm 4 ECCEnc($\mathbf{m} \in \mathcal{M}$)

Ensure: An encoding $\mathbf{c_m} \in \{0,1\}^{l_v}$.
1: $\hat{\mathbf{c}} \leftarrow \mathsf{BCHE}(\mathbf{m}) \in \{0,1\}^{n_e}$
2: $\mathbf{c_m} \leftarrow \hat{\mathbf{c}} \| 0^{l_v - n_e} \in \{0,1\}^{l_v}$
3: **return** $\mathbf{c_m}$

The subroutine ECCDec on input an encoding $\mathbf{c_m}$, decoding the codeword $\hat{\mathbf{c}}$ in it. Usually, a message $\mathbf{m}$ is recovered. In case of a decoding error, an error symbol $\perp$ is returned.

Algorithm 5 ECCDec($\mathbf{c_m} \in \{0,1\}^{l_v}$)

Ensure: A bit string $\mathbf{m}$.
1: $\hat{\mathbf{c}} \leftarrow (\mathbf{c_m})_{n_e}$
2: $\mathbf{m} \leftarrow \mathsf{BCHD}(\hat{\mathbf{c}}) \in \{0,1\}^{le}$
3: **return** $\mathbf{m}$

4.2 LAC.CCA

The IND-CCA secure **key encapsulation mechanism** LAC.CCA is obtained by applying the Fujisaki-Okamoto transformation [15,19] to the IND-CPA secure encryption scheme LAC.CPA. The method was suggested by Peikert in [27] and also used in [7].

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 6.
- The decapsulation algorithm Dec, as illustrated in Algorithm 7.

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 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 $l_k = l_m$. The concrete choice of the hash functions G and H will be specified in Section 8.

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

Algorithm 6 LAC.CCA.Enc($pk; \mathsf{seed}_{\mathbf{m}}$)

Ensure: A ciphertext and encapsulated key pair $(\mathbf{c}, K)$.
1: $\mathbf{m} \leftarrow \mathsf{Samp}(U(\mathcal{M}); \mathsf{seed}_{\mathbf{m}}) \in \mathcal{M}$
2: $\mathsf{seed} \leftarrow G(\mathbf{m}) \in \mathcal{S}$
3: $\mathbf{c} \leftarrow \mathsf{LAC.CPA.Enc}(pk, \mathbf{m}; \mathsf{seed})$
4: $K \leftarrow H(\mathbf{m}, \mathbf{c}) \in \{0,1\}^{l_k}$
5: **return** $(\mathbf{c}, K)$

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 7 LAC.CCA.Dec($sk, \mathbf{c}$)

Ensure: An encapsulated key K.
1: $\mathbf{m} \leftarrow \mathsf{LAC.CPA.Dec}(sk, \mathbf{c})$
2: $K \leftarrow H(\mathbf{m}, \mathbf{c})$
3: $\mathsf{seed} \leftarrow G(\mathbf{m}) \in \mathcal{S}$
4: $\mathbf{c}' \leftarrow \mathsf{LAC.CPA.Enc}(pk, \mathbf{m}; \mathsf{seed})$
5: **if** $\mathbf{c}' \neq \mathbf{c}$ **then**
6: $\quad K \leftarrow H(H(sk), \mathbf{c})$
7: **end if**
8: **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 8.

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

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

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

Note that LAC.KE constructed from LAC.CPA is only passively secure when there is no key caching. Besides, we can also construct a passively secure key exchange protocol directly from LAC.CCA. In this case, the key caching in the server end (Alice) is allowed, as in TLS.

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 [18]. The LAC.AKE is secure in Canetti-Krawczyk mode with weak perfect forward secrecy [11], resistance to key compromise impersonation (KCI) attack [17,18], and maximal exposure attacks (MEX) [17,18]. 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
$(pk, sk) \xleftarrow{\$} \mathsf{LAC.CPA.KG}()$		
$\mathbf{r}_1 \xleftarrow{\$} \{0,1\}^{l_s}$		
$\mathsf{seed}_1 \leftarrow G(\mathbf{r}_1, sk_A)$		
$(\mathbf{c}_1, K_1) \leftarrow \mathsf{LAC.CCA.Enc}(pk_B; \mathsf{seed}_1)$	$\xrightarrow{pk, \mathbf{c}_1}$	
		$\mathbf{r}_2 \xleftarrow{\$} \{0,1\}^{l_s}$
		$\mathsf{seed}_2 \leftarrow G(\mathbf{r}_2, sk_B)$
		$(\mathbf{c}_2, K_2) \leftarrow \mathsf{LAC.CCA.Enc}(pk_A; \mathsf{seed}_2)$
		$K_3 \xleftarrow{\$} \{0,1\}^{l_m}$
	$\xleftarrow{\mathbf{c}_2, \mathbf{c}_3}$	$\mathbf{c}_3 \xleftarrow{\$} \mathsf{LAC.CPA.Enc}(pk, K_3; \epsilon)$
$K_2 \leftarrow \mathsf{LAC.CCA.Dec}(sk_A, \mathbf{c}_2)$		
$K_3 \leftarrow \mathsf{LAC.CPA.Dec}(sk, \mathbf{c}_3)$		$K_1 \leftarrow \mathsf{LAC.CCA.Dec}(sk_B, \mathbf{c}_1)$
$K \leftarrow H(pk_A, pk_B, pk, \mathbf{c}_3, K_1, K_2, K_3)$		$K \leftarrow H(pk_A, pk_B, pk, \mathbf{c}_3, K_1, K_2, K_3)$

5 Parameter Choices

In this section we will motivate the choice of the dimension, modulus, distributions of the errors and secrets of the poly-LWE problem, and the code length, message length, and error correcting capability of the error correction codes. Main motivations of parameter choice including security, decryption error rate, bandwidth and computational efficiency.

5.1 The Dimension of Poly-LWE

The most important motivation for the choice of the dimension n of the poly-LWE problem is to guarantee its worst-case hardness and the resistance against concrete attacks. According to the result in [21,23,12], the hardness of the poly-LWE problem over the ring of $\mathbb{Z}_q[x]/(x^n+1)$ for n being a power of 2 can be reduced to the worst-case hardness of lattice problems. For concrete attacks, according to the result in [10,3,1], the complexity of the most efficient algorithm to solve the SVP problem is about $2^{0.292\times n}$. For these reasons, it is suitable to choose $n = 2^9$ and $n = 2^{10}$ for different categories of security strength.

5.2 The Modulus of Poly-LWE

The main criteria to choose the modulus q is to reduce the size of the key and ciphertext. In previous schemes based on the poly-LWE problem, to speed up the computation based on NTT, the modulus q is chosen to be 12289 or 7681 [13,1,7]. To improve the bandwidth efficiency, we use $q = 251$ in LAC, which is the largest prime that less than 256. Note that, to use the NTT algorithm, the modulus $7681 = 3 \times 2^8 + 1$ is the smallest prime for dimensions bigger than 2^8. Hence, we can not use NTT directly in LAC. Fortunately, when $q < 256$, we can gain a speed improvement for the polynomial multiplications of nearly 30× with AVX2 or 60 × with AVX512.

5.3 The Errors and Secrets Distribution of Poly-LWE

There are two rules for the choice of the distribution for the errors 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. According to these reasons, we use the centered binomial distributions in LAC:

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

See following sections for detail effects of the distribution on the security strength and the error rate of LAC.

5.4 Parameters of the Error Correction Codes

We choose the BCH codes as the error correction codes in LAC. The parameters of the BCH codes, namely, the code length n_e, the message length l_e, the minimum distance d_e, and the maximum error number t_e, are chosen according to the desired security strength and the corresponding message length of LAC.CPA.

Let the security parameter of the desired post-quantum security strength be λ, then the message length of LAC.CPA should satisfy $l_m = 2\lambda$, the message length of the BCH code should satisfy $l_e \geq l_m$, and the code length $n_e > l_e$. To achieve the best error-correcting capability, we choose the code length to be the smallest integer that satisfies $n_e > l_e$ and $n_e = 2^a - 1$ for some positive integer a.

5.5 Parameters for Different Security Strength Categories

Concrete parameters for three categories of LAC are described in Table 1:

Categories	n	q	Distribution	BCH$[n_e, l_e, d_e, t_e]$	Message Length l_m
LAC128	512	251	Ψ_1	$[511, 264, 59, 29]$	256
LAC192	1024	251	$\Psi_{\frac{1}{2}}$	$[511, 392, 27, 13]$	384
LAC256	1024	251	Ψ_1	$[1023, 520, 111, 55]$	512

Table 1. Parameter settings of LAC.CPA

We remark that, sometimes the implementations of the algorithm may encrypt the length of the plaintext together with the plaintext message itself, so the message length of BCH l_e is larger than the length of plaintext message l_m. In the BCH parameters above, we set $l_e = l_m + 8$ to store the length of the plaintext message.

6 Correctness Analysis

In the following we show correctness of LAC.CPA described above. Parameters will be selected according to Section 5.

Theorem 1. *Let* $\mathbf{s}, \mathbf{e}, \mathbf{r}, \mathbf{e_1}, \mathbf{e_2}$ *be random variables that are distributed according to algorithms 1 and 2. Denote* $\mathbf{w} \triangleq (\mathbf{se_1} - \mathbf{re})_{l_v} + \mathbf{e_2}$*, and* $\delta = 1 - \Pr[-q/4 < w_i < q/4]$ *be the single bit failure probability, and we use a BCH code with parameters* (n_e, l_e, d_e, t_e)*,* LAC.CPA *can be decrypted correctly except with probability* $\Delta = \sum_{i=t_e+1}^{n_e} \binom{n_e}{i} (1-\delta)^{n_e - i} \delta^i$*.*

To formally prove Theorem 1, we will use $\mathsf{erfc}(x)$, central limit theorem and independence assumption, let us first review them.

Gaussian Error Function. The error function $\mathsf{erf}(x)$ is defined as $\mathsf{erf}(x) = \frac{1}{\sqrt{\pi}} \int_{-x}^{-x} e^{-t^2} dt$. And the error complementary function $\mathsf{erfc}(x) = 1 - \mathsf{erf}(x)$. In statistics, for nonnegative values of x, the error function $\mathsf{erf}(x)$ is interpreted as the probability of a variable Y which is normally distributed with mean 0 and variance $1/2$ falling in the range $[-x, x]$. While $\mathsf{erfc}(x)$ denotes the probability Y falling out of the range $[-x, x]$.

Note that for a normally distributed variable Y with mean 0 and variance σ^2, the probability of Y falling in range $[-x, x]$ is $\mathsf{erf}(\frac{x}{\sigma\sqrt{2}})$.

Central Limit Theorem. When random variables $\{X_i\}_{i \in [n]}$ have the same distribution with mean μ and variation s^2, and n is large enough, then $\sum_{i \in [n]} X_i$ is approaching Gaussian distribution with mean $n\mu$ and variation ns^2.

Independence Assumption. In [29] the authors proposed the independence assumption, which assumes that coefficients of $\mathbf{se}$ are independent, where $\mathbf{s}$ and $\mathbf{e}$ follow a strictly bounded error distribution.

During the computation process, we will use the following basic facts:

Fact 1. If variables x, y are distributed discretely, then $\Pr[xy = k] = \sum_a (\Pr[x = a] \Pr[y = k/a])$. Assume x, y are the same distribution that takes ± 1 with probability α and 0 with probability $1 - 2\alpha$, then the multiplication of x and y takes ± 1 with probability $2\alpha^2$ and 0 with probability $1 - 4\alpha^2$. And the variable xy is with mean 0 and variance $4\alpha^2$.

Fact 2. For variables x, y that are discrete distributed, then $\Pr[x + y = k] = \sum_a (\Pr[x = a] \Pr[y = k - a])$.

Fact 3. If variable x is distributed discretely, variable y follows a continuous distribution, then $\Pr[x + y \leq k] = \sum_a (\Pr[x = a] \Pr[y \leq k - a])$.

Proof. According to the algorithm 3,

$$\begin{aligned}\mathbf{c}'_{\mathbf{m}} &= \mathbf{c}_2 - (\mathbf{u})_{l_v} \in \mathbb{Z}_q^{l_v} \\ &= ((\mathbf{as}+\mathbf{e})\mathbf{r})_{l_v} + \mathbf{e}_2 + \bar{q}\cdot\mathbf{c}_{\mathbf{m}} - ((\mathbf{ar}+\mathbf{e_1})\mathbf{s})_{l_v} \\ &= \mathbf{w} + \bar{q}\cdot\mathbf{c}_{\mathbf{m}}\end{aligned}$$

Then if $-q/4 < w_i < q/4$, then $-q/4 + \bar{q}\cdot\mathbf{c}_{\mathbf{m}i} < \mathbf{c}'_{\mathbf{m}i} = w_i + \bar{q}\cdot\mathbf{c}_{\mathbf{m}i} < q/4 + \bar{q}\cdot\mathbf{c}_{\mathbf{m}i}$, obviously in this case $\mathbf{c}_{\mathbf{m}i}$ can be recovered correctly.

According to the wrap-around multiplication rule, one can get that, for $i \in [0, l_v]$, $w_i = \sum_{j=0}^{i}(s_j e_{1(i-j)} - r_j e_{i-j}) - \sum_{j=i+1}^{n-1}(s_j e_{1(n+i-j)} - r_j e_{n+i-j}) + e_{2i}$. For concrete correctness probability, we first compute the distribution of $X = s_j e_{1(i-j)}$, then $\sum_{j=0}^{i}(s_j e_{1(i-j)} - r_j e_{i-j}) - \sum_{j=i+1}^{n-1}(s_j e_{1(n+i-j)}$ is the distribution of 2n independent variables from distribution X, using the convolution summing rule we can quickly compute it, then sum the distribution with e_{2i} one can compute the distribution of w_i directly, so the single bit failure probability is the sum of the probability of those values less than $-q/4$ and greater than $q/4$, i.e. δ.

For a quick estimation, we give an easier and approaching computation with Gaussian error function. Applying the central limit theorem to our case, we can see that the item $\sum_{j=0}^{i}(s_j e_{1(i-j)} - r_j e_{i-j}) - \sum_{j=i+1}^{n-1}(s_j e_{1(n+i-j)})$ can be seen as a Gaussian distribution with mean 0 and variance $\bar{\sigma}^2 = 2n/(2^2)$ for LAC128 and LAC256, (while with variance $\bar{\sigma}^2 = 2n/(4^2)$ for LAC192), since every coordinate of $\mathbf{se_1}$ (i.e.$s_j e_{1(i-j)}$) is with mean 0 and variance $s^2 = (\sigma/2)^2$. Thus the error probability can be easily estimated with the complementary error function $\mathsf{erfc}(\frac{\lfloor q/4 \rfloor}{\sqrt{2}\bar{\sigma}})$. We denote the bit fail probability by this method as δ_e.

Since we use a binomial centered distribution on {-1,0,1}, according to the independence lemma, we suppose that w_i for different i's are independent. As we use a BCH code with parameter (n_e, l_e, d_e, t_e), which can correct t_e errors, LAC.CPA can be correctly recovered as long as there exists no more than t_e errors for all the n_e length $\hat{\mathbf{c}}$. As for every single bit, the fail probability is δ, the failure probability for the whole message is $\Delta = \sum_{i=t_e+1}^{n_e}\binom{n_e}{i}(1-\delta)^{n_e-i}\delta^i$. With a python script we can compute the failure probability, listed as below. (Note that we always choose the same $q = 251$.)

Algorithms	n	σ	δ	δ_e	δ_t	Δ
LAC128	512	1	$2^{-13.35}$	$2^{-13.17}$	$2^{-13.41}$	$2^{-239.6}$
LAC192	1024	1/2	$2^{-24.51}$	$2^{-24.44}$	$2^{-24.74}$	$2^{-253.8}$
LAC256	1024	1	$2^{-7.44}$	$2^{-7.34}$	$2^{-7.48}$	$2^{-115.4}$

Table 2. Failure probability of LAC.CPA

In Table 2 the column "n" denotes the dimension of vectors $\mathbf{s}, \mathbf{r}, \mathbf{e}, \mathbf{e_1}, \mathbf{e_2}$, "$\sigma$" denotes the distribution of each coordinate is Ψ_σ, we use "δ" to denote the probability that $\mathbf{c}_{\mathbf{m}i}$ is not correctly recovered, "δ_e" to denote the bit failure probability estimated with gaussian error function, "δ_t" to give a tested result of bit failure probability by taking $l_m \times 10^7$ random message bits and testing how many of them will be decrypted falsely, and in the last column we use "Δ" to denote the probability that the whole message $\mathbf{m}$ is not correctly decrypted.

7 Security Analysis

7.1 Formal Security

Security of LAC.CPA follows the security result of [22], which states that LAC.CPA is IND-CPA secure under the poly-LWE assumption.

Security of the LAC.CCA scheme follows the Fujisaki-Okamoto transformation [15,19] (the version with implicit rejection), so IND-CCA security of LAC.CCA can be reduced to the IND-CPA security of LAC.CPA. Note that there is a slight difference from the transformation in [19].

In our scheme, we substitute $H(sk)$ for part of the secret key s, but this will not influence the security since $H(sk)$ can be changed back to a randomly chosen s indistinguishably.

7.2 Concrete Security

In this section we analyze the concrete security strength of LAC. We analyze the hardness of poly-LWE as a general LWE problem, since the existing best known attacks do not make use of the special structure of poly-LWE.

Hardness analysis. There are many general algorithms to solve the LWE problem. According to the survey in [3,30], lattice reduction attacks based on the BKZ algorithm are more powerful than exhaustive search, combinational and algebraic algorithms. For simplicity, just as in [1], we focus on the two embedding attacks usually named as primal attack and dual attack.

- Primal attack: In the primal attack, a unique-SVP instance from the LWE samples is constructed and then solved by using the BKZ algorithm. The security strength is evaluated by the block dimension b required for the BKZ algorithm to find the unique solution. Briefly, given the LWE instance $(\mathbf{A}, \mathbf{b} = \mathbf{As} + \mathbf{e})$, $\mathbf{A} \in \mathbb{Z}_q^{m\times n}$, the target lattice of dimension $d = m + n + 1$ is constructed as $\Lambda = \{\mathbf{x} \in \mathbb{Z}^{m+n+1} : (\mathbf{A}|\mathbf{I}_m| - \mathbf{b})\mathbf{x} = \mathbf{0} \mod q\}$. It is easy to verify that, $\mathbf{v} = (\mathbf{s}, \mathbf{e}, 1)$ is the unique-SVP solution. According to the result in [1], 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, $\delta = ((\pi b)^{1/b} b/2\pi e)^{1/(2(b-1))}$.
- Dual attack: In the dual attack, given the LWE instance $(\mathbf{A}, \mathbf{b} = \mathbf{As} + \mathbf{e})$, $\mathbf{A} \in \mathbb{Z}_q^{m\times n}$, the target lattice of dimension $d = m + n$ is constructed as $\Lambda' = \{(\mathbf{x}, \mathbf{y}) \in \mathbb{Z}^m \times \mathbb{Z}^n : \mathbf{A}^t\mathbf{x} = \mathbf{y} \mod q\}$. According to the result in [1], BKZ can be used to find a vector $\mathbf{v} = (\mathbf{x}, \mathbf{y})$ of length $l = \delta^{d-1} q^{n/d}$ and the distance between $\mathbf{v}^t\mathbf{b}$ and the uniform distribution is bounded by $\epsilon = 4\exp(-2\pi^2\tau^2)$, where $\tau = l\sigma/q$. This can be used to break the decisional LWE problem with advantage ϵ. Finally, the computational complexity is evaluated by the BKZ algorithm with the block size of b, which amplifies ϵ to $1/2$ by repeating the sieve algorithm $R = max(1, 1/(\gamma\epsilon^2))$ times to build $1/\epsilon^2$ short vectors, where $\gamma = 2^{0.2075b}$ is the number of vectors provided by the sieve algorithm.

Note that, the BKZ algorithm proceeds by reducing a lattice basis using an SVP oracle in a smaller dimension b polynomial times. As pointed out in [1], it is hard to exactly evaluate the number of calls to the SVP oracle. Thus, the polynomial factor is ignored and the computational complexity is evaluated by just considering the core SVP algorithm. According to the result in [1], the best complexity of the SVP oracle based on lattice sieve algorithm is $\sqrt{3/2}^{b+o(b)} \approx 2^{0.292b}$. The best complexity of the SVP oracle based on quantum sieve algorithm is $\sqrt{13/9}^{b+o(b)} \approx 2^{0.265b}$.

Security claims. According to the hardness analysis above, the computational complexity of LAC and the security claims are described as follows:

Algorithm	Claimed Security	Primal Attack		Dual Attack		Security Categories
		Classical	Quantum	Classical	Quantum	
LAC128	128	148	135	147	133	1,2
LAC192	192	288	261	286	259	3,4
LAC256	256	323	293	320	290	5

Table 3. Security claims of LAC.CPA

In the table above, the column "Security Categories" is according to the five categories described in Section 4.A.5 of the call for proposals document provided by NIST.

8 Implementation

Since the modulus $q = 251$ used in LAC is very small, it is suitable to be implemented on a wide range of processors. In this section, we provide the implementation detail on the Intel x64 processor.

8.1 Basic Blocks

Basic blocks used in the implementation of LAC include randomness generator, pseudo-randomness generator and hash functions. For simplicity, these blocks are constructed based on openssl as follows:

- Randomness generator: The randomness generation function "RAND_bytes()" is directly used in the implementation to generate random bytes.
- Pseudo-randomness generator: Pseudo-randomness generator is constructed based on the AES256 encryption algorithm in the ctr mode.
- Hash function: The hash functions used in LAC is constructed from SHA256, SHA384 and SHA512 from openssl for different security strength categories.

8.2 The Secrets and Errors Distribution

Two kinds of centered binomial distributions, $\Psi_1, \Psi_{1/2}$, are used in LAC for the generation of secret and error vectors. An element e from the distribution Ψ_1 is directly generated by $b - b'$, where b and b' are random bits. An element $\hat{e} \in \Psi_{1/2}$ is generated by $e_1 \times e_2$, where $e_1, e_2 \in \Psi_1$.

8.3 Polynomial Multiplication

Polynomial multiplication is the most time consuming operation in the implementation of LAC. Besides direct implementation in the reference version code, we provide two optimized versions as follows:

- **General Optimized Version:** 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. Thus we can package 4 items into one $uint64\_t$ type variable and implement the polynomial multiplication as $\mathbf{as} = \sum_{s_i=1} a_i - \sum_{s_i=-1} a_i$. We remark that, although we can package 8 items into one $uint64\_t$ type variable, it is more efficient to only package 4 items. The reason is that, when each item is represented as a 16 bits variable, we can remove the mod q operation during the sum operation.
- **AVX2 Based Version:** Since the modulus $q = 251$, based on the $\_mm256\_maddubs\_epi16$ instruction, we can compute 32 multiplications and adjacent addition operation with one instruction. Thus, AVX2 based polynomial multiplication is about 30 times faster than the reference version.

8.4 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. It is a third-party free software, and via an e-mail request, the use of of the codes has been approved by NIST. It can be compiled on both Linux and the msys2 environment on Windows. In the reference implementation version of LAC, the parameters of BCH are initiated by calling the $init\_bch()$ function dynamically. Since the operation $init\_bch()$ is very time consuming, in the optimized implementation of LAC, we avoid the execution of $init\_bch()$ by setting the BCH parameters as global variables.

9 Performance Analysis

9.1 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.

Categories	Key generation		Encryption		Decryption	
	CPU Cycles	Times(μs)	CPU Cycles	Times(μs)	CPU Cycles	Times(μs)
LAC128	90686	29.25	152575	49.22	68285	22.03
LAC192	309216	99.75	410469	132.41	238268	76.86
LAC256	269827	87.04	513753	165.73	336207	108.45

Table 4. Performance of LAC.CPA

Categories	Key generation		Encapsulation		Decapsulation	
	CPU Cycles	Times(μs)	CPU Cycles	Times(μs)	CPU Cycles	Times(μs)
LAC128	90411	29.16	160314	51.71	216957	69.99
LAC192	281324	36.92	421439	42.61	647030	73.89
LAC256	267831	90.75	526915	169.97	874742	282.17

Table 5. Performance of LAC.CCA

Categories	Alice0		Bob		Alice1	
	CPU Cycles	Times(μs)	CPU Cycles	Times(μs)	CPU Cycles	Times(μs)
LAC128	90200	29.10	160689	51.84	78071	25.18
LAC192	280615	90.52	416633	134.40	240988	77.74
LAC256	276207	89.10	543667	175.38	338270	109.12

Table 6. Performance of LAC.KE

Categories	Alice0		Bob		Alice1	
	CPU Cycles	Times(μs)	CPU Cycles	Times(μs)	CPU Cycles	Times(μs)
LAC128	253372	81.73	586217	189.10	332502	107.26
LAC192	706420	227.88	1513042	488.08	912086	294.22
LAC256	803437	259.17	1953638	630.21	1253351	404.31

Table 7. Performance of LAC.AKE

9.2 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.

Categories	Key generation		Encryption		Decryption	
	CPU Cycles	Times(μs)	CPU Cycles	Times(μs)	CPU Cycles	Times(μs)
LAC128	38957	12.56	53357	17.21	27259	8.79
LAC192	114753	37.02	117749	37.98	54313	17.52
LAC256	78678	25.38	137258	44.28	133379	43.03

Table 8. Performance of LAC.CPA with AVX2

Categories	Key generation		Encapsulation		Decapsulation	
	CPU Cycles	Times(μs)	CPU Cycles	Times(μs)	CPU Cycles	Times(μs)
LAC128	38847	12.53	60952	19.66	74812	24.13
LAC192	114461	36.92	132087	42.61	229054	73.89
LAC256	78214	25.23	160489	51.77	293128	94.56

Table 9. Performance of LAC.CCA with AVX2

Categories	Alice0		Bob		Alice1	
	CPU Cycles	Times(μs)	CPU Cycles	Times(μs)	CPU Cycles	Times(μs)
LAC128	38757	12.50	58502	18.87	28100	9.06
LAC192	115540	37.27	186310	60.10	89749	28.95
LAC256	78231	25.24	229757	74.12	157412	50.78

Table 10. Performance of LAC.KE with AVX2

Categories	Alice0		Bob		Alice1	
	CPU Cycles	Times(μs)	CPU Cycles	Times(μs)	CPU Cycles	Times(μs)
LAC128	122814	39.62	251954	81.28	120456	38.86
LAC192	241045	77.76	539718	174.10	371265	119.77
LAC256	273593	88.26	769274	248.15	536645	173.11

Table 11. Performance of LAC.AKE with AVX2

9.3 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	1024
LAC192	1056	2080	1536
LAC256	1056	2080	2048

Table 12. 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	1024	1568	2048
LAC192	1056	1536	2592	3072
LAC256	1056	2048	3104	4096

Table 13. Message size of LAC.KE and LAC.AKE

9.4 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	30	40932
LAC192	511	384	14	19452
LAC256	1023	512	57	81480

Table 14. Memory cost of error correction code

10 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.

10.1 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.

10.2 LAC.CCA

In this section, we test LAC.CCA.KG, LAC.CCA.ENC (Algorithm 6), and LAC.CCA.DEC (Algorithm7). 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 6.
– "ss" denotes "K" in LAC.CCA.Enc algorithm 6.

10.3 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.

10.4 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.

11 The Advantages and Limitations

11.1 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.

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.

11.2 Limitations

The limitations of LAC:

– Can not be sped up by using the NTT technique, which affects its computational performance on processors that do not support vector instructions.

References

1. E. Alkim, L. Ducas, T. Poppelmann, and P. Schwabe. Post-quantum Key Exchange - a New Hope. In USENIX Security 2016, USENIX Association, 2016. https://eprint.iacr.org/2015/1092.
2. E. Alkim, L. Ducas, T. Poppelmann, and P. Schwabe. NewHope without reconciliation. https://eprint.iacr.org/2016/1157.
3. M. R. Albrecht, R. Player, and S. Scott. On the concrete hardness of Learning with Errors. In J. Mathematical Cryptology, vol. 9(3), 169-203, De Gruyter, 2015.
4. R. C. Bose, and D. K. Ray-Chaudhuri. On a Class of Error Correcting Binary Group Codes. Information and Control, 3(1): 68-79, 1960.
5. J. Bos, C. Costello, L. Ducas, I. Mironov, M. Naehrig, V. Nikolaenko, A. Raghunathan, and D. Stebila. Frodo: Take off the ring! Practical, Quantum-Secure Key Exchange from LWE. In ACM CCS 2016, pp. 1006-1018, ACM, 2016.
6. J. W. Bos, C. Costello, M. Naehrig, and D. Stebila. Post-quantum Key Exchange for the TLS Protocol from the Ring Learning with Errors Problem. In 2015 IEEE Symposium on Security and Privacy, pp. 553-570, IEEE Computer Society, 2015.
7. J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, and D. Stehlé. CRYSTALS - Kyber: a CCA-secure Module-lattice-based KEM. In Cryptology ePrint Archive: Report 2017/634. https://eprint.iacr.org/2017/634.
8. M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated Key Exchange Secure Against Dictionary Attcks. In Eurocrypt 2000, LNCS 1807, pp. 139-155, Springer, 2000.
9. Z. Brakerski, and V. Vaikuntanathan. Fully Homomorphic Encryption from Ring-LWE and Security for Key Dependent Messages. In CRYPTO 2011, LNCS 6841, pp. 505-524, Springer, 2011.
10. Y. Chen, and P. Q. Nguyen. BKZ 2.0: Better Lattice Security Estimates. In ASIACRYPT 2011, LNCS 7073, pp. 1-20, Springer, 2011.
11. R. Canetti, and H. Krawczyk. Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels. In EUROCRYPT 2001, LCNS 2045, pp.453-474, springer, 2001.
12. L. Ducas, and A. Durmus. Ring-LWE in Polynomial Rings. In PKC 2012, LNCS 7293, pp. 34-51, Springer, 2012.
13. L. Ducas, A. Durmus, T. Lepoint, and V. Lyubashevsky. Lattice Signatures and Bimodal Gaussians. In CRYPTO 2013, LNCS 8042, pp. 40-56, Springer, 2012.
14. J. Ding, X. Xie, and X. Lin. A Simple Provably Secure Key Exchange Scheme based on the Learning with errors problem. In Cryptology ePrint Archive 2012/688, 2012.

15. E. Fujisaki, and T. Okamoto. Secure Integration of Asymmetric and Symmetric Encryption Schemes. In CRYPTO 1999, LNCS 1666, pp. 537-554, Springer, 1999.
16. E. Fujisaki, and T. Okamoto. Secure Integration of Asymmetric and Symmetric Encryption Schemes. In J. Cryptology, Vol. 26(1): 80-101, 2013.
17. A. Fujioka, K. Suzuki, K. Xagawa, and K. Yoneyama. Strongly Secure Authenticated Key Exchange from Factoring, Codes, and Lattices. In PKC 2012, LNCS 7293, pp. 467-484, Springer, 2012.
18. A. Fujioka, K. Suzuki, K. Xagawa, and K. Yoneyama. Practical and post-quantum authenticated key exchange from one-way secure key encapsulation mechanism. In Asia CCS 2013, pp. 83-94, 2013.
19. D. Hofheinz, K. Hövelmanns, and E. Kiltz. A Modular Analysis of the Fujisaki-Okamoto Transformation. In TCC 2017.
20. A. Hocquenghem. Codes correcteurs d'erreurs, Chiffres (in French), 2: 147-156, 1959.
21. V. Lyubashevsky, C. Peikert, and O. Regev. On Ideal Lattices and Learning with Errors over Rings. In Eurocrypt 2010, LNCS 6110, pp. 1-23, Springer, 2010.
22. R. Lindner and C. Peikert. Better Key Sizes (and Attacks) for LWE-Based Encryption. In CT-RSA 2011, LNCS 6558, pp. 319–339, Springer, 2011.
23. V. Lyubashevsky, C. Peikert, and O. Regev. A Toolkit for Ring-LWE Cryptography. In Eurocrypt 2013, LNCS 7881, pp. 35-54, Springer, 2013.
24. D. Micciancio, P. Mol. Pseudorandom Knapsacks and the Sample Complexity of LWE Search-to-decision Reductions. In CRYPTO 2011, LNCS 6841, pp. 465-484, Springer, 2011.
25. D. Micciancia and O. Regev. Worst-case to Average-case Reductions based on Gaussian Measures. In SIAM Journal on Computing, 37(1): 267-302, 2007.
26. C. Peikert. Public-Key Cryptosystems from the Worst-Case Shortest Vector Problem: Extended Abstract. In STOC 2009, pp. 333-342, ACM, 2009.
27. C. Peikert. Lattice Cryptography for the Internet. In PQCrypto 2014, LNCS 8772, pp. 197-219, Springer, 2014.
28. O. Regev. On Lattices, Learning with Errors, Random Linear Codes, and Cryptography. In STOC 2005, pp. 84-93, ACM, 2005.
29. M.-J. O. Saarinen. HILA5: On Reliability, Reconciliation, and Error Correction for Ring-LWE Encryption. In SAC 2017. IACR Cryptology ePrint Archive, report 2017/424, http://eprint.iacr.org/2017/424.
30. M. Schmidt, and N. Bindel. Estimation of the Hardness of the Learning with Errors Problem with a Restricted Number of Samples. In IACR Cryptology ePrint Archive, report 2017/140, http://eprint.iacr.org/2017/140.