LAC: Practical Ring-LWE Based Public-Key Encryption with Byte-Level Modulus

Xianhui Lu[1,2], Yamin Liu[1,2], Zhenfei Zhang[3], Dingding Jia[1,2], Haiyang Xue[1,2], Jingnan He[1,2], Bao Li[1,2], Kunpeng Wang[1,2]

1. State Key Laboratory of Information Security,IIE, CAS.
2. School of Cyber Security, University of Chinese Academy of Sciences.
3. Algorand.
luxianhui@outlook.com, zhenfei@algorand.com, hejingnan@iie.ac.cn

Abstract. We propose an instantiation of public key encryption scheme based on the ring learning with error problem, where the modulus is at a byte level and the noise is at a bit level, achieving one of the most compact lattice based schemes in the literature. The main technical challenges are a) the decryption error rates increases and needs to be handled elegantly, and b) we cannot use the Number Theoretic Transform (NTT) technique to speed up the implementation. We overcome those limitations with some customized parameter sets and heavy error correction codes. We give a treatment of the concrete security of the proposed parameter set, with regards to the recent advance in lattice based cryptanalysis. We present an optimized implementation taking advantage of our byte level modulus and bit level noise. In addition, a byte level modulus allows for high parallelization and the bit level noise avoids the modulus reduction during multiplication. Our result shows that LAC is more compact than most of the existing (Ring-)LWE based solutions, while achieving a similar level of efficiency, compared with popular solutions in this domain, such as Kyber.

1 Introduction

Due to the rapid advances of quantum computing, the construction of cryptographic schemes secure against quantum attacks (a.k.a post-quantum cryptography) becomes an important mission in the field of cryptology. Lattice based cryptography is one of the most promising and mature candidates for the post-quantum migration plan of the National Institute of Standards and Technology (NIST) [21,55]. Currently, most of the practical lattice based encryption schemes are designed based on the Learning With Errors (LWE) problem, which was initially proposed by Regev [63], and became extremely versatile in constructing public key encryption schemes [61,57,45,53,14], identity based encryption schemes [37,20,1,2] and fully homomorphic encryption schemes [18,17,38]. Despite of all those ground breaking applications, the main drawback remains that they have key size at least quadratic in the main security parameter. Inspired by the NTRU cryptosystem [40] and the ring-based short integer solution problem

[52,48], Lyubashevsky, Peikert and Regev [49,50,60] resolved this problem by introducing an algebraic variant of LWE, namely, Ring-LWE; and showed that its security can be reduced to worst-case problems on ideal lattices. It is worth noting that in a concurrent and independent work, Stehlé *et al.* [69] also proposed a special case of Ring-LWE over power-of-two cyclotomic polynomials; in [60], it was shown that Ring-LWE problem is hard for any ring with appropriate error distribution.

For almost all LWE based constructions, there exists an instantiation with Ring-LWE where the size of the public key and the ciphertext can be reduced by a factor of n, where n is the dimension of the polynomial ring. Depending on the choice of the ring, one may also carry out the ring multiplications in $O(n \log n)$ by using the fast Fourier transform (FFT) or number theoretic transform (NTT). Due to its great security, utility and efficiency, Ring-LWE and its variants become the most popular building-blocks in the design of practical cryptosystems [15,8,7,16,43,66,9].

To date, the (Ring-)LWE based public key encryption schemes have arrived at a mature state that is almost ready for deployment, except for the aforementioned size problem. For the public key encryption schemes, they all follow a similar framework by Regev [63] and Lyubashevsky *et al.* [49]. For the key exchange protocols, one may use the reconciliation method, first put forth by Ding [30], and then refined by Peikert [58], to improve efficiency.

Subsequent works, such as [15,8,16,66,9], have contributed to a large portion of the NIST post-quantum cryptography standardization process (NIST-PQC) [55]. As one has seen from those work, further improvement of the bandwidth efficiency has become one of the main missions in the design of practical lattice based cryptographic schemes.

1.1 Our Contributions

Motivation. Before presenting our contributions, let us briefly present our motivation. For the sake of simplicity, we will use the ring $\mathcal{R} = \mathbb{Z}_q[x]/(x^n + 1)$ with a power-of-two n, a favorable choice by many Ring-LWE based schemes [32,8], to illustrate our idea, although we must remark that it is shown by Peikert *et al.* in [60] that Ring-LWE is hard for any ring of integers.

Intuitively, the hardness of Ring-LWE problem is mainly determined by the error rate α (the ratio of the noise magnitude to the modulus q) and the dimension n. According to the concrete hardness analysis [1] in [6,8,5], suitable choices of the dimension n are $2^9 = 512$ and $2^{10} = 1024$. For these choices, $q = 12289$ is the smallest prime for which $q \equiv 1 \bmod 2n$. In other words, to enable the super efficient NTT multiplications, we have a constraint that q is at least 12289. As a result, $q = 12289$ is one of the most widely used modulus for the Ring-LWE based schemes. Almost concurrent of this paper, in the design of Kyber [16], a smaller polynomial ring of degree $n = 256$ was used as the basic block of the

[1] As opposite to the provable security, this is a method to obtain the bit-complexity by looking at the cost of best known attacks, such as BKZ with quantum sieving.

Module-LWE problem. NTT over this ring is possible with a smaller modulus $q = 7681$. Recently, based on a new variant of the NTT technique [51], the modulus $q = 3329$ was used in the NIST-PQC second round version of Kyber,

From the view of cryptographic scheme design, 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, for the sake of space complexity, it makes sense to choose the modulus as small as possible, while keeping the ratio somewhat a constant to maintain a same security level.

In this paper, we investigate the above approach. We consider "byte" level modulus. Byte is the smallest data type that modern processors handle. It seems to be a sweet spot to balance performance, size and security. We also remark that for moduli that are significantly smaller than 256, the performance gain will be minimal (since processors will treat the data type as a byte anyway) while it becomes unfeasible to find error distributions that can maintain a same error/modulus ratio.

Parameter Derivation. There have been a sequence of work on the theoretical worst-case hardness of Ring-LWE problems [69,49,60,65]. However, they give no guidance on the choice of concrete parameters. Parameter derivation is an active research topic for lattice based cryptography, for both cryptography and cryptanalysis [22,59,4]. Arguably, most lattice based submissions to NIST-PQC follow a similar design [49,45,30,58], and a major differentiator among the schemes is the choices of parameters.

As mentioned earlier, we consider the family of "byte" level modulus that breaks the constraint of NTT modulus. Specifically, we consider three types of byte-level moduli, namely "power-of-two modulus", "max-split modulus" and "min-split modulus". We then select proper secret and error distribution to match the proposed modulus. Since the concrete security partially relies on the error rates, to be able to sample errors efficiently becomes crucial to the overall design. For provable security one requires discrete Gaussian samples; however, in practice it is sufficient to sample from distributions that are close enough to a Gaussian. We observe that centered binomial distribution with the standard deviation of $\sigma = 1/\sqrt{2}$ is a sweet spot for security, correctness and efficiency.

To show the concrete security of our scheme, similar to other works in this field, we perform a concrete analysis of best known attacks, using both the popular and generic analysis such as BKZ with (quantum) sieving [6,8,5] and hybrid attacks [41,39], as well as dedicated attacks, such as the subfield attacks, hamming weight attacks [54] and pattern attacks [62].

Versions. Since the debut of our first proposal realizing the above methodology, there have been two versions of parameters. The first version of our scheme was submitted to the first round of NIST-PQC. Parameter sets for this submission

is referred to as

$$\mathsf{LAC\text{-}v1} = \{\mathsf{LAC\text{-}128\text{-}v1}, \mathsf{LAC\text{-}192\text{-}v1}, \mathsf{LAC\text{-}256\text{-}v1}\}.$$

In the second round of NIST-PQC, as well as the previous version of this paper [47], we proposed a second version of the parameter sets, addressing several potential risks arised during the first round evaluation and the second round evaluation [54]. Those are referred to as

$$\mathsf{LAC\text{-}v2} = \{\mathsf{LAC\text{-}128\text{-}v2}, \mathsf{LAC\text{-}192\text{-}v2}, \mathsf{LAC\text{-}256\text{-}v2}\}.$$

The major change from LAC-v1 to LAC-v2 is that the secrets and the noises are sampled from a fixed hamming weight binomial distribution; and consequently a weaker error correction code is adopted. It was observed that LAC-v1 may be vulnerable to side channel attacks [27] and high hamming weight attacks [54]. We moved to LAC-v2 to seal those leakage. We give a detailed analysis of those attacks in section 5.

For completeness we list both LAC-v1/2 parameter sets in Appendix A.

In this paper/revision, we introduce a third version of the parameter set

$$\begin{aligned}\mathsf{LAC\text{-}v3} = \{&\mathsf{LAC\text{-}light\text{-}v3a}, \mathsf{LAC\text{-}128\text{-}v3a}, \mathsf{LAC\text{-}192\text{-}v3a}, \mathsf{LAC\text{-}256\text{-}v3a}\\ &\mathsf{LAC\text{-}light\text{-}v3b}, \mathsf{LAC\text{-}128\text{-}v3b}, \mathsf{LAC\text{-}192\text{-}v3b}, \mathsf{LAC\text{-}256\text{-}v3b}\},\end{aligned}$$

in accouting for new cryptanalysis results [29,28,27,25,62]. This revision sees two major changes. First, we present LAC-light-v3. This parameter is designed for embedded systems such as a Cortex M4 platform. It is a variant of LAC-128-v3, with smaller noise and weaker error corrections. Second, for each level, we have two parameter sets, for example, LAC-128-v3a and LAC-128-v3b, corresponding to modulus 251 and 256, respectively.

It is worth noting that the underlying algorithms remain unchanged for all three versions of LAC parameter sets.

Error Corrections. In most lattice based schemes, dated back from one of the first lattice based encryption schemes, NTRU [40], there exists a (tunable) decryption error probability. One may choose a zero decryption error probability, at the cost of a larger modulus (and hence larger keys and ciphertexts); or a negligible one, with a moderate size modulus. See, for example [34], for a comparison of different error correction codes for lattice based cryptography. Our byte level modulus incurs a very high decryption error rate by design; and simple error correction techniques, such as D2 or D4 codes [8], do not work well in our use case.

To cope with this error growth, we encode the plaintext message with an error correction code that supports very large block size. Generally speaking, with the great power comes great cost: error correction code for large block sizes brings severe efficiency penalty. We propose to use binary BCH error correction code, which is particularly efficient, in both encoding and decoding. With BCH code we are able to decrease the decryption error rate to a desirable level.

The main drawback of a large block error correction code is the high computation cost. To balance the time and space complexity, we choose the combination of D2 and BCH. Briefly, the message is firstly encoded with the BCH code, then encoded with the D2 code. In the decryption algorithm, the code words will be corrected using D2-then-BCH method. In this approach, the error rate is sharply decreased by D2 already, and we only need to correct a small number of errors by using BCH. Since D2 is very efficient, the combination of D2 and a small number of BCH error correction is sufficient for our use case while remaining efficient.

We remark that our usage of heavy error correction mechanism has sparked fruitful discussions which lead to advancement of cryptanalysis in the field, for instance, see [29,28,27,25,62]. Looking ahead, our revised LAC-v3 parameter sets are robust against all those attacks. We also note that, the choice of error correction code will not affect the theoretical security of the scheme (see Section 4.4 for more details). Our scheme in principle supports any error correction code with required error correction ability.

Implementation. Recall that we have switched to a byte level modulus, we can no longer resort to NTT for efficient ring multiplications. Popular alternatives are Karatsuba/Toom-Cook algorithms, such as [11,23,26] and index based multiplications algorithms, such as [9]. We adopt the index based solutions, combined with the following customized optimization for our parameters.

- **Faster Multiplication.** Our secrets are sampled from a centered binomial distribution with a fixed hamming weight. No less than 50% of the secrets are 0s. So the total number of the additive operations can be reduced by half. Meanwhile, the polynomial multiplication is carried out without invoking integer multiplications. The secrets lie in $\{-1, 0, 1\}$ implies that the polynomial multiplication can be carried out with integer additions and subtractions.
- **Less Modulus Reduction.** Since the modulus and the noise are both very small, the intermediate value during the polynomial multiplication will always be smaller than 2^{16}. When the intermediate states are stored with `uint16_t` types, we only need to do a final mod operation, instead of for each addition.
- **High Parallelism.** We use `uint8_t` to store the coefficients of the polynomial. For a non-AVX computer, we use `uint16_t` to store intermediate variables during the execution. This achieves a 4-way parallelism with a 64 bits register. With AVX2, we achieve 16 and 32-way parallelism using `_mm256_madd_epi16` and `_mm256_madd_epi8` intrinsic, for LAC-v3a and LAC-v3b, respectively.

Our code is publicly available at [46]. It contains two types of optimized implementations over Intel64 platform, the first one is based on the general 64bits operations. The second solution is based on the AVX2 SIMD instructions. The code achieve constant time execution, while being significantly faster than previous implementations that are submitted to NIST-PQC process.

Constant time implementation. At a high level, for constant time implementations of polynomial multiplication, we used a new form of representation of the secret vectors. Instead of storing the exact values of $\{-1, 0, 1\}$, we store the positions of 1s, followed by -1s. Then, the polynomial multiplication can be carried out by executing a constant number of of accumulations. In addition, since the secret vector has a fixed hamming weight, the number of iterations is also constant.

Compared with previous implementation, we also achieved a constant time BCH code for LAC-v3. Previously, making BCH constant time incurs a very high cost, in that the decoder will need to iteration for l times, where l is the maximum number of errors. This is possible for LAC-v3 because the are significantly smaller than that of LAC-v1/v2.

Performance improvement. In terms of improvements, a brief comparison of the performance across three versions are listed in Table 12 (Appendix B). The main source of the improvement comes from:

- Accumulation based polynomial multiplication. Instead of using coefficients representation, we directly store the positions of 1 and -1 in the secret vector. Thus the polynomial multiplications can be carried out directly by accumulations.
- Higher parallelism and free modulus reduction with modulus $q = 256$. When $q = 256$ the intermediate value are stored in `uint8_t` instead of `uint16_t`, which supports 32-way parallelism based on AVX2 SIMD instructions.
- LAC-light without BCH. The decoding algorithm of BCH code is one of the most time consuming part in our scheme. LAC-light that achieves small decryption error rate, where D2 and parity check code is sufficient.
- An `AES-NI` and `SIMD` based pseudorandom generator. This was adopted from [16]. It improves the previous one (adopted from openssl) by 8 times.

1.2 Comparison.

To highlight the compactness and efficiency of our scheme, we briefly compare the performance of our scheme with Kyber [16] (detailed comparison with other lattice based schemes can be found in section 6). We compare the chosen ciphertext secure version of the schemes. All these schemes uses BKZ with (quantum) sieving to estimate their security. Note that our estimation is independently confirmed in [4].

In a nutshell, LAC outperforms Kyber at 128 and 256 bits security levels, in terms of key size, ciphertext size; while remaining adequately more generous on the security margin. It is also worth pointing out that our LAC-192-v3b may also be considered for NIST-PQC level 5, and has outperformed Kyber1024 for this category.

NIST-PQC category	Scheme	Size (in Bytes)			AVX2 Cycles			Security
		sk	pk	ct	gen	enc	dec	
N/A	LAC-light-v3b	1056	544	664	13,388	20,775	22,094	117
	LAC-light-v3a	1056	544	664	19,139	28,064	32,396	118
I	Kyber512-90s	1632	800	736	20,004	30,384	24,604	100
	Kyber512	1632	800	736	33,428	49,184	40,564	100
	LAC-128-v3b	1056	544	704	20,703	33,546	45,165	133
	LAC-128-v3a	1056	544	704	28,759	46,068	63,672	133
III	Kyber768-90s	2400	1184	1088	30,884	45,892	37,844	164
	Kyber768	2400	1184	1088	62,396	83,748	70,304	164
	LAC-192-v3b	2080	1056	1352	27,242	45,531	68,707	258
	LAC-192-v3a	2080	1056	1352	46,322	76,208	113,029	259
V	Kyber1024-90s	3168	1568	1568	44,040	64,352	54,448	230
	Kyber1024	3168	1568	1568	88,568	115,952	99,764	230
	LAC-256-v3b	2080	1056	1464	38,230	68,320	148,468	276
	LAC-256-v3a	2080	1056	1464	62,894	112,060	213,001	277

sk secret key **pk** public key
ct ciphertext **gen** key generation
enc encryption or encapsulation **dec** decryption or decapsulation

Table 1. Comparison of Kyber and LAC

1.3 Outline

In section 2 we recall basic definitions and notations. In section 3 we present our Ring-LWE based public key encryption scheme. In section 4 we describe the selection of parameters. In section 5 we give the security evaluation of our new scheme. In section 6 we discuss implementation issues of our scheme. Finally, we give the conclusion in section 7.

2 Preliminaries

In this section we first define several mathematical notations, the definitions of Ring-LWE and public key encryption schemes.

2.1 Basic Notations

Vectors are denoted by lower-case characters, such as $\vec{a}$. $\vec{a}^t$ denotes the transposition of $\vec{a}$. Matrices are denoted by upper-case characters, such as $\vec{A}$. $\vec{A}^t$ denotes the transposition of $\vec{A}$. For an m-dimensional vector $\vec{a} = (a_0, a_1, ..., a_{m-1})$, its l_1-norm is defined as $\|\vec{a}\|_1 = \sum_{i=0}^{m-1} |a_i|$; the l_2-norm, also known as the Euclidean norm, is defined as $\|\vec{a}\|_2 = \sqrt{\sum_{i=0}^{m-1} a_i^2}$, or solely denoted as $\|\vec{a}\|$. The length of a matrix is the norm of its longest column vector, e.g., $\|\vec{A}\| := \max \|\vec{a}_i\|$. For an m-dimensional vector $\vec{a} = (a_0, \cdots, a_{m-1})$ and a non-negative integer $l \leq m$, define $(\vec{a})_l := (a_0, \cdots, a_{l-1})$.

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 randomly from the set of output of A with input x; if the algorithm A is deterministic, we simplify it as $y \leftarrow \mathsf{A}(x)$.

For an integer $q \geq 1$, let $\mathbb{Z}_q$ be the residue class ring modulo q, define the ring of integer polynomials modulo $x^n + 1$ as $\mathcal{R} := \mathbb{Z}[x]/(x^n + 1)$ for an integer $n \geq 1$, and the ring $\mathcal{R}_q := \mathbb{Z}_q[x]/(x^n + 1)$ denotes the polynomial ring modulo $x^n + 1$ where the coefficients are from $\mathbb{Z}_q$.

2.2 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 $\mathcal{R}_q$ is $U(\mathcal{R}_q)$.

The Centered Binomial Distribution. The idea to simulate a Gaussian distribution with binomial distribution was firstly introduced in [8], in order to mitigate the heavy cost of Gaussian sampling. 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 1 (Ψ_1). *Sample $(a, b) \xleftarrow{\$} \{0, 1\}^2$, and output $a - b$. 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 2 ($\Psi_{\frac{1}{2}}$). *Sample $(a, b) \xleftarrow{\$} \Psi_1$, and output $a * b$. 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 $\frac{1}{4}$.*

We define n-ary centered binomial distribution with fixed Hamming weight, denoted as Ψ_n^h, where $0 < h < n/2$ 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. Denote by Samp an abstract algorithm that samples 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 $\mathsf{seed} = \perp$, the process is randomized, and equivalent to $x \xleftarrow{\$} D$. When a seed is present, the sampling of x will be deterministic.

We extend the definition to a multiple dimension setting. 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-s from distributions D_i-s for $1 \leq i \leq t$.

2.3 Learning with Errors (over Rings)

We refer the readers to [63,64,69,49,60] for a concrete background of the definitions and reductions.

Definition 3 (Search LWE). *Let n, m, q be positive integers, and $\chi_{\vec{s}}, \chi_{\vec{e}}$ be (bounded) distributions over $\mathbb{Z}$. Given $(\vec{A}, \vec{b} = \vec{A}\vec{s} + \vec{e})$, recover the secret $\vec{s}$, where $\vec{A} \xleftarrow{\$} \mathbb{Z}_q^{m \times n}$, the secret $\vec{s} \xleftarrow{\$} \chi_{\vec{s}}^n$ and the error $\vec{e} \xleftarrow{\$} \chi_{\vec{e}}^m$.*

Definition 4 (Decisional LWE). *Let n, m, q be positive integers, and $\chi_{\vec{s}}, \chi_{\vec{e}}$ be (bounded) distributions over $\mathbb{Z}$. Distinguish the two distributions of $(\vec{A}, \vec{b})$ and $(\vec{A}, \vec{u})$, where $\vec{b} = \vec{A}\vec{s} + \vec{e}$ for $\vec{A} \xleftarrow{\$} \mathbb{Z}_q^{m \times n}$, $\vec{s} \xleftarrow{\$} \chi_{\vec{s}}^n$, $\vec{e} \xleftarrow{\$} \chi_{\vec{e}}^m$, $\vec{u} \xleftarrow{\$} \mathbb{Z}_q^m$.*

Definition 5 (Search RLWE). *Let n, q be positive integers, and $\chi_{\vec{s}}, \chi_{\vec{e}}$ be (bounded) distributions over $\mathcal{R}$. Given $(\vec{a}, \vec{b} = \vec{a}\vec{s} + \vec{e})$, recover the secret $\vec{s}$, where $\vec{a} \xleftarrow{\$} \mathcal{R}_q$, the secret $\vec{s} \xleftarrow{\$} \chi_{\vec{s}}$ and the error $\vec{e} \xleftarrow{\$} \chi_{\vec{e}}$.*

Definition 6 (Decisional RLWE). *Let n, q be positive integers, and $\chi_{\vec{s}}, \chi_{\vec{e}}$ be (bounded) distributions over $\mathcal{R}$. Distinguish two distributions of $(\vec{a}, \vec{b})$ and $(\vec{a}, \vec{u})$, where $\vec{b} = \vec{a}\vec{s} + \vec{e}$ for $\vec{a} \xleftarrow{\$} \mathcal{R}_q$, $\vec{s} \xleftarrow{\$} \chi_{\vec{s}}$, $\vec{e} \xleftarrow{\$} \chi_{\vec{e}}$, $\vec{u} \xleftarrow{\$} \mathcal{R}_q$.*

2.4 Public Key Encryption

A *public key encryption scheme* PKE=(Gen,Enc,Dec) with message space $\mathcal{M}$ consists of three polynomial-time algorithms.

- **KG**(l): A probabilistic polynomial-time key generation algorithm takes as input the security parameter l and outputs a public key pk and a private key sk. We write $(pk, sk) \leftarrow \mathrm{KG}(l)$.
- **Enc**(pk, m): A probabilistic polynomial-time encryption algorithm takes as inputs a public key pk, a plaintext m and outputs a ciphertext c. We write $c \leftarrow \mathrm{E}_{pk}(m)$.
- **Dec**(sk, c): A decryption algorithm takes as inputs a ciphertext c and a private key sk, and outputs a plaintext m. We write $m \leftarrow \mathrm{D}_{sk}(c)$.

A public key encryption scheme is IND-CCA2 (indistinguishable against adaptive chosen ciphertexts attacks) secure if the advantage of any adversary

$\mathcal{A}$ defined in the following is negligible in the security parameter l:

$$\mathrm{Adv}^{\mathrm{cca}}_{\mathcal{A}}(l) = \left| \Pr \left[b' = b : \begin{array}{l} (pk, sk) \leftarrow \mathrm{Gen}(l), \\ (m_0, m_1) \leftarrow \mathcal{A}^{\mathrm{D}_{sk}(\cdot)}(pk), \\ b \xleftarrow{R} \{0,1\}, \\ c^* \leftarrow \mathrm{E}_{pk}(m_b), \\ b' \leftarrow \mathcal{A}^{\mathrm{D}_{sk}(\cdot)}(pk, c^*). \end{array} \right] - \frac{1}{2} \right|,$$

where $\mathcal{A}$ is restricted not to query $\mathrm{D}_{sk}(\cdot)$ with c^*.

3 The LAC Scheme

In this section we describe our Ring-LWE based public key encryption scheme "LAC". LAC is a concrete instantiation of the Ring-LWE based scheme proposed in [49]. The main deviation at an algorithmic level is that the plaintext message is encoded with a large-block error correction code.

3.1 The Scheme

Notations. Let q be the modulus. 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 the n-ary centered binomial distribution with fixed hamming weight Ψ_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 $\vec{m} \in \{0,1\}^{l_m}$ and its encoding $\widehat{\vec{m}} \in \{0,1\}^{l_v}$ 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 algorithm LAC.KG randomly generates a pair of public key and secret key (pk, sk).

Algorithm 1 LAC.KG()

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

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

The algorithm LAC.Enc on input pk and a message $\vec{m}$, encrypts $\vec{m}$ 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 $\vec{m}$ into a codeword $\widehat{\vec{m}}$.

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

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

The algorithm LAC.Dec on input sk and a ciphertext $\vec{c}$, recovers the corresponding message $\vec{m}$. The subroutine ECCDec on input an encoding $\widehat{\vec{m}}$, decoding the codeword in it. Usually, a message $\vec{m} \in \mathcal{M}$ is recovered. When there is a decryption error, the returned message $\vec{m} \notin \mathcal{M}$.

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

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

3.2 Formal Security

Following the result of [45], 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 [35,36] and quantum random oracle model [42]. 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.

4 Parameter Selection

Almost all lattice based key exchanges and public key encryptions, except for NTRU based ones, follow a similar framework from [30,58,15,7]. We have a set of theoretical results on the choice of rings, moduli, errors, etc [60,59,65] 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 one of a main research direction in subsequent works [15,7,59,16,43,66], and a main differentiator in most NIST-PQC submissions [55]. In this section, we present our choice of parameters, and give our design rational over common choices.

4.1 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 will work on reducing the modulus. Note that the modulus cannot be too small; it needs to be large enough to tolerant the errors during decryption. Prior to our work, 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 4.3.

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. Currently, it is unclear whether this algebraic structure will cause any weakness in the security.
- **Max-Split Modulus:** The reason for choosing $q \equiv 1 \bmod 2n$ 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 [11]. For conservative purpose, and also for the sake of simplicity, we choose the min-split modulus $q = 251$ for the main version of our scheme submitted to NIST-PQC. Nonetheless, we do not see any weakness of the power-of-two modulus or the max-split modulus. For different kinds of security and performance balance, in this paper, we also give an extended version of our scheme by using $q = 256$. Recently, Lyubashevsky et al. [51] proposed new tricks that partially enable NTT for max-split rings. We leave the implementation of $q = 257$ version of our scheme to future research.

4.2 The Errors and Secrets Distribution

There are two principles 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 [63,49], namely, discrete Gaussian distribution [49,8] and centered binomial distribution [16]. Gaussian distribution consumes lots of entropy, is hard to implement in constant time, and is also vulnerable to memory based side channel attacks [19] when implemented with look-up tables [32]. 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 allows 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: $\Pr[x = 0] = 1/2$, $\Pr[x = \pm 1] = 1/4$. This, and its variants, are used in LAC-v1.

As pointed out by Alperin-Sheriff in the comments to LAC-v1 [55], when a centered binomial distribution is used, the adversary can increase the decryption error rate by finding high hamming weight random vectors through precomputation. 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.

Looking ahead, to make LAC-v2/v3 immune to high hamming weight attack in a more efficient manner, we use n-ary centered binomial distribution Ψ_n^h with fixed Hamming weight h for the error and secret vectors ($h/2$ ones and $h/2$ minus ones). The implementation of fixed weight centered binomial distribution will be described in section 6.

4.3 Decryption Errors

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

$$\begin{aligned}\widetilde{\vec{m}} &= \vec{c}_2 - (\vec{c}_1\vec{s})_{l_v} \\ &= (\vec{br})_{l_v} + \vec{e}_2 + \lfloor \tfrac{q}{2} \rceil \widehat{\vec{m}} - (\vec{c}_1\vec{s})_{l_v} \\ &= ((\vec{as} + \vec{e})\vec{r})_{l_v} + \vec{e}_2 + \lfloor \tfrac{q}{2} \rceil \widehat{\vec{m}} - ((\vec{ar} + \vec{e}_1)\vec{s})_{l_v} \\ &= (\vec{er} - \vec{e}_1\vec{s})_{l_v} + \vec{e}_2 + \lfloor \tfrac{q}{2} \rceil \widehat{\vec{m}}\end{aligned} \tag{1}$$

Let $\vec{w} = (\vec{er} - \vec{e}_1\vec{s})_{l_v} + \vec{e}_2$, 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 $\vec{s}, \vec{e}, \vec{r}, \vec{e}_1, \vec{e}_2$ 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:

$$\delta \approx 1 - \mathsf{erf}\left(\frac{\lfloor 251/4 \rceil}{\sqrt{2}((1/\sqrt{2})^2\sqrt{2 \times 512})}\right) \approx 2^{-13.95}.$$

When D2 is used, one message bit will be encoded as two elements in $\widetilde{\vec{m}}$. As a result, the error rate of the message bit is $\delta = 1 - \Pr[|w_i| + |w_j| < \lfloor \frac{q}{2} \rceil]$. Thus the error rate of each message bit can be estimated as $\delta \approx 1 - \mathsf{erf}\left(\frac{\lfloor q/2 \rceil}{\sqrt{2(\sigma^2\sqrt{4n})}}\right)$. For the case of $n = 512, q = 251$, the error rate of each message bit is:

$$\delta \approx 1 - \mathsf{erf}\left(\frac{\lfloor 251/2 \rceil}{\sqrt{2}((1/\sqrt{2})^2\sqrt{4 \times 512})}\right) \approx 2^{-25.95}.$$

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

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

Note that, as pointed out by D'Anvers [28], when single bit error rate δ is too large, we can not assume that the coefficients of $\vec{w}$ 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 $\vec{s}, \vec{e}, \vec{r}, \vec{e}_1$. This leads to the use of fix hamming weight distribution in LAC-v2 and LAC-v3. When $\vec{s}, \vec{e}, \vec{r}, \vec{e}_1$ are sample from this distribution, their norms are also fixed and the main source of dependence is removed. So we can assume that the coefficients of $\vec{w}$ are independent from each other.

Based on the equations above, we can estimate the decryption error rate in the case of chosen plaintext attacks, in which the secrets and errors are all randomly selected according to the sampling algorithm. However, in the case of chosen ciphertext attacks, the adversary may choose noise vectors with special patterns to increase the correlation between the coefficients of $\vec{w}$. For example, Guo *et al.*[62] propose several patterns to increase the decryption error rate of LAC-v2. At the current stage, a comprehensive theoretical analysis of the dependence model seems impossible. Our solution in LAC-v3 is to decrease the decryption error rate further.

4.4 Error Correction Code

Our byte level modulus incurs a 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.

Although BCH code with large enough error correction capability can decrease the decryption error rate small enough, it will also bring high computational cost (see section 6 for detailed computational cost). To solve this problem, in this paper we choose the combination of D2 and BCH code. That is, the plaintext is firstly encoded with the BCH code, then the code word is encoded with the D2 code. With the help of the high performance D2 code, the BCH only need to correct a small number of errors. Concrete parameters of BCH are described in the next subsection.

4.5 Recommended Parameter Set

The main parameters of LAC-v3a are listed in Table 2, with respect to three categories of NIST-PQC standardization project [55], namely, the equivalent security level of AES128, AES192 and AES256. Since the decoding of BCH code is a time consuming operation, it may not suitable for small embedded processors such as Cortex M4. We also propose a light version of LAC, named as LAC-light, which can be seen as a noise reduced version of LAC-128. The error rate of LAC-light is small. We can achieve suitable decryption error rate based on simple parity check error correction technique combined with D2 code.

Categories	n	q	**dis**	**ecc**	$l_{\vec{m}}$	**pk**	**sk**	**ct**	**bit-er**	**dec-er**
light-v3a	512	251	Ψ_{512}^{128}	parity check+D2	128	544	512	664	$2^{-81.73}$	2^{-150}
128-v3a	512	251	Ψ_{512}^{256}	BCH[255,128,17]+D2	128	544	512	704	$2^{-22.26}$	2^{-151}
192-v3a	1024	251	Ψ_{1024}^{256}	BCH[511,256,17]+D2	256	1056	1024	1352	$2^{-42.24}$	2^{-324}
256-v3a	1024	251	Ψ_{1024}^{384}	BCH[511,256,41]+D2	256	1056	1024	1464	$2^{-20.01}$	2^{-302}

dis secret and noise distributions **ecc** error correction code
$l_{\vec{m}}$ 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 2. Main parameter of LAC

The dimension n, modulus q and the error distribution are selected to achieve high enough security levels according to the lattice reduction attacks and hybrid attacks (detailed analysis are in section 5).

The parameters of BCH "$[l_c, l_m, l_d]$" are selected to achieve a small enough decryption error rate to defeat the high Hamming weight attacks [55] and pattern attacks [62]. Concretely, for LAC-128 and LAC-192, we choose $l_d = 17$ which allows us to correct upto 8 bits of errors; for LAC-256 we choose $l_d = 41$ which allows to correct upto 20 bits of errors.

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 $\vec{c}_2$ are discarded. This brings an additional uniformly random (under Ring-LWE assumption) error over $[-7, 8]$.

A public key consists of a 32 bytes seed $\mathsf{seed}_{\vec{a}}$, and an n bytes vector $\vec{b}$. 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 $\vec{c}_1$, and l_v bytes of $\vec{c}_2$. For LAC-light $l_v = l_m + 3 \times 8$, where 3 is the byte size of the redundant data of the parity check error correction code. For LAC-128 and LAC-192 parameter set, $l_v = l_m + 8 \times 8$, where 8 is the size of the redundant data. For LAC-256, $l_v = l_m + 21 \times 8$, where 21 is the size of the redundant data.

In Table 2, the modulus of LAC-v3a is $q = 251$, which keeps the same as LAC-v1 and LAC-v2 . In addition, we also propose an alternative parameter set, named as LAC-v3b, with $q = 256$ instead of 251.The reminder of the parameters stays the same. Since the modulus is a little larger, the decryption error rate of LAC-v3b is a slightly smaller than the counterpart in LAC-v3a.

5 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 (e.g. BKZ with (quantum) sieving[2]) are well understood. Since our secrets and errors are sparse, we also evaluate the cost of hybrid attacks [41,70,39].

We also consider dedicated attacks that target specific designs of our scheme, namely the subfield attacks, the high Hamming weight attacks and the pattern attacks. Those attacks are reported as comments to the Round 1/2 version of LAC submission to NIST-PQC.

5.1 Generic Attacks

There are many generic algorithms to solve the LWE problem, see [6,68] for a survey of known techniques. It has been shown that lattice reduction attacks utilizing the BKZ algorithm [22] 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 also evaluate the cost for hybrid attacks. We summarize the security estimates in Table 3.

Algorithm	**Primal Attack**			**Dual Attack**			**Hybird Attack**	
	C	**Q**	**B**	**C**	**Q**	**B**	**C**	**Q**
LAC-LIGHT	131	118	448	130	118	445	124	119
LAC-128	148	135	509	147	133	505	148	141
LAC-192	288	261	986	286	259	978	278	267
LAC-256	308	279	1054	305	277	1044	316	301
LAC-LIGHT-q256	130	118	447	129	117	444	124	118
LAC-128-q256	148	134	508	147	133	503	148	140
LAC-192-q256	287	261	984	288	258	975	278	266
LAC-256-q256	307	278	1051	304	276	1042	315	300

C: Classical complexity **Q:** Quantum complexity
B: Block Size

Table 3. 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

[2] There has been some debate on the accuracy of the formula to calculate the concrete cost of sieving [54]. These discussions are irrelevant to our parameters since a) they do not change that fact that the generic attacks remain the best attacks for LAC, and b) the LAC-v3 parameters are derived from the more conservative side of the debates.

shortest vector. In a nutshell, given an LWE instance $(\vec{A}, \vec{b} = \vec{A}\vec{s}+\vec{e})$, $\vec{A} \in \mathbb{Z}_q^{m\times n}$, the target lattice of dimension $d = m + n + 1$ is constructed as

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

It is easy to verify that, $\vec{v} = (\vec{s}, \vec{e}, 1)$ is the unique-SVP solution when both $\vec{s}$ and $\vec{e}$ 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 $(\vec{A}, \vec{b} = \vec{A}\vec{s} + \vec{e})$, $\vec{A} \in \mathbb{Z}_q^{m\times n}$, the target lattice of dimension $d = m + n$ is constructed as

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

Again, [7] showed that BKZ is capable of finding a vector $\vec{v} = (\vec{x}, \vec{y})$ of length $l = \delta^{d-1}q^{n/d}$, where the distance between $\vec{v}^t\vec{b}$ 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.

Hybrid Attack. At a high level, the hybrid attack takes the following steps.

- Firstly, one interprets the lattice basis $\mathbf{B} \in \mathbb{Z}^{n,n}$ as a concatenation of two matrices $\mathbf{B}_1$, $\mathbf{B}_2 \in \mathbb{Z}^{\ell,n} \times \mathbb{Z}^{k,n}$, with $\ell > n/2$ and $\ell + k = n$. Note that $\mathcal{L}(B_1)$ spans a sublattice of dimension ℓ.
- Then, one performs lattice reductions over this sublattice, to an extend that it allows for solving bounded distance decoding problem effectively.
- Next, one guesses linear combinations of row vectors from $\mathbf{B}_2$. If one has guessed the correct combination as the unique shortest vector (or their rotations) in $\mathbf{B}$, this vector will be a close vector to $\mathcal{L}(\mathbf{B}_1)$.

– Lastly, one uses the reduced basis of $\mathcal{L}(B_1)$ to solve the bounded distance decoding problem.

It is easily to see that, to get the best performance, one usually assumes the cost of lattice reductions is on the same order of guessing. What remains to be estimated is the individual cost for each step. There are a few subtlety on estimating the cost of the above procedure. To be more conservative, we make the following assumptions/decisions.

– In the estimation, the cost of solving CVP/BDD is often neglected; a good basis yields a polynomial time algorithm (for example, Babai's algorithms), which may adds a few bits complexity in practice.
– We use BKZ with (quantum)-core sieving to estimate the cost of BKZ. This model counts the cost of a single SVP, and ignores the number of iterations. It is also a more conservative choice than BKZ with enumeration.
– We also assume that if basis is "good", and if we have successfully guessed the correct combination, then, the probability of solving the CVP is exactly 1.
– To be more conservative on the searching side, we estimate the entropy in the guessing phase, and taking its square root (in accounting for Grover's algorithm) as the cost of guessing. This results into a lower bound that isn't achievable through classical MITM attacks. This also assumes vector additions incur a cost of 1.

The details of the hybrid attack parameters can be found in Table 4. It was generated with the estimator [67].

Algorithm	**Classical**					**Quantum**				
	ℓ	Block	BKZ	Search	Total	ℓ	Block	BKZ	Search	Total
LAC-light-v3a	633	425	124	124	124	658	449	119	118	119
LAC-light-v3b	635	424	124	124	124	658	446	118	118	118
LAC-128-v3a	718	508	148	148	148	743	553	141	140	141
LAC-128-v3b	720	507	148	147	148	742	529	140	140	140
LAC-192-v3a	1141	953	278	278	278	1191	1009	267	266	267
LAC-192-v3b	1145	952	278	277	278	1190	1003	266	266	266
LAC-256-v3a	1255	1081	316	315	316	1303	1135	301	301	301
LAC-256-v3b	1257	1077	315	315	315	1307	1134	300	300	300

Table 4. Parameterizing the hybrid attacks

Security Estimates. We use BKZ simulator with core-(Q)sieving model to estimate the security for our scheme. The corresponding security is estimated for the obtained blocksize. Our script is available at [46]. Note that in [4], Albrecht *et al.* independently evaluated the security for all (Ring-)LWE candidates, and their estimation matches ours in this paper.

5.2 Dedicated Attacks

Subfield Attacks. The idea of exploiting subfields is known to the lattice community for years [10,3,13,44], and to use this idea to analyze LAC was firstly proposed by Alperin-Sheriff [54] 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 $\vec{g}$ and $\vec{h}$ where $\vec{g} = x^{n/2} + 91x^{n/4} + 250$ and $\vec{h} = x^{n/2} + 160x^{n/4} + 250$.

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

Analysis. In the rest, we give a high level 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 (while the number of iterations does not); 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.

We will show that the corresponding vectors in the subfields, $(\vec{s}_g, \vec{e}_g)$, 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 $(\vec{s}_g, \vec{e}_g)$ (similarly, $(\vec{s}_h, \vec{e}_h)$). To be precise, when $(\vec{s}, \vec{e})$ are small polynomials with the coefficients in $\{-1, 0, 1\}$, the coefficients of $(\vec{s}_g, \vec{e}_g)$ 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 $\vec{s}$ and $\vec{e}$ by 11, all the coefficients of $(\vec{s}_g, \vec{e}_g)$ will be within the interval of $[-25, 25]$.

Let $\vec{A} = \left[\vec{A}_g | \vec{I} | 11 \times \vec{b}_g\right]$, where $\vec{A}_g$ is the matrix generated by $\vec{a}_g$, if $\vec{z} = [11 \times \vec{s}_g | 11 \times \vec{e}_g | -1]$ is the shortest solution of $\vec{A}z = 0 \bmod q$, we can recover $\vec{z}$ 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 $\vec{A}$ is a random matrix, the q-ary lattice $\Lambda_q^{\perp}(\vec{A})$ will behave as a random lattice [24], 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 $\vec{z}$. Central limit theory says that the length of $\vec{z}$ approximately follows a discrete Gaussian distribution. Our implementation shows that $\vec{z}$ 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[3].

It is easy to verify that, the length of $\vec{z}$ will be larger than the solution of $\vec{Az} = 0 \bmod q$ except for negligible probability. Hence $\vec{z}$ 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 $(\vec{r}, \vec{e}_1)$ in some ciphertexts (with certain probability) may have higher-than-usual Hamming weight. It is feasible if $(\vec{r}, \vec{e}_1)$ 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 (and correspondingly, pre-computations), an attacker may obtain sufficient number of samples with higher Hamming weight secrets and errors. This may leak information on the secret key.

Analysis. It has been shown that chosen plaintext secure version of (Ring-)LWE based schemes suffer from an reaction attack [33]. To address this vulnerability, most schemes rely on Fujisaki-Okamoto transformation [35,36,42] to achieve chosen ciphertext security. We also adopt the same approach. Via this transformation, the randomness vectors $(\vec{r}, \vec{e}_1)$ are generated from the plaintext message by a pseudorandom generator. Thus the vectors $(\vec{r}, \vec{e}_1)$ are randomly distributed from the view of the adversary.

In a comment to LAC in the Round1 estimation of NIST-PQC [55], Alperin-Sheriff pointed out that, for the LAC-256 parameter set, the probability that a pair of valid $(\vec{r}, \vec{e}_1)$ with a Hamming weight of at least $1024 + 310 = 1334$ is greater than

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

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 $(\vec{r}, \vec{e}_1)$ have Hamming weight exceeding 1334. It is worth noting that the adversary can only access the decryption oracle for 2^{64} times in the security model of NIST-PQC. Next, for samples with such high

[3] 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.

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 $\vec{m}$:

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

As a result, with 2^{207} pre-computations and 2^{64} decryption oracle queries, the adversary can get about $2^{19.6}$ decryption failures.

Counter measures. To prevent high hamming weigh attacks, we switch to n-ary centered binomial distribution with fixed Hamming weight. This makes LAC completely immune from high Hamming weight attacks and their potential variants.

Pattern Attack. Recently, a new analysis result was proposed by Guo *et al.*. Its main idea is to find error vectors with special kind of patterns. In general, the adversary tries to find error vectors that contain consecutive "1"s or "−1"s. If this is the case, one can except a higher-then-normal decryption error rate. And as mentioned earlier, with enough decryption errors, the attacker will learn (partial) information about the secret key.

Concretely, the authors showed that, with the pre-computation cost of 2^{162} and online computation cost of 2^{79}, they can recover the private key of LAC-256-v1 with a success probability of 2^{-64}.

Analysis. There are mainly two reasons that LAC-256-v1 and LAC-256-v2 are vulnerable to pattern attacks. Firstly, we do not have multi-target protection in LAC-v1/2. That is, a pre-computation can be used to attack multiple public keys. This can be prevented easily by hashing the public key into the seed that instantiate the PRNG. Then, the adversary must compute errors for each public key; and cannot pre-process this task. The second reason is that the decryption error rate of LAC-256-v1 and LAC-256-v2 are non-negligible with regard to its security level.

Counter measures. To make LAC immune to decryption error based attacks, such as high Hamming attack and pattern attacks, we further decrease the average case decryption error rate of LAC-v3 to negligible. In other words, for a security level l, the decryption error rate will be smaller than 2^{-l}.

6 Implementations and Performance

The implementation of LAC mainly includes four subroutines:

– Random polynomial sampling;
– Secret polynomial sampling;
– Polynomial multiplication;
– Encoding/decoding of error correction code.

6.1 Random Polynomial Generation

For $q = 256$, we simply sample n random bytes, and cast it as $\vec{a} \in \mathbb{Z}_q[x]/(x^n+1)$. For $q = 251$, we use rejection sampling based technique, and keep reading from the sampler until we have gathered n integers that are small than q. Note that the data associated with this subroutine are all public. Rejection sampling here does not affect the fact that the execution of the code is independent from secret information.

The performance of this subroutine is summarized in Table 5.

n (dimension)	q (modulus)	prf (with aes256ctr)	performance (Cycles)
512	251	openssl	7253
1024	251	openssl	11774
512	251	`AES-NI, SIMD`	2817
1024	251	`AES-NI, SIMD`	4312
512	256	openssl	3981
1024	256	openssl	7626
512	256	`AES-NI, SIMD`	681
1024	256	`AES-NI, SIMD`	1099

Table 5. Performance of Random Polynomial Generation

"prf": the pseudo-random function used to generate the random bytes;
"openssl": the aes256ctr function from the openssl lib, used in LAC-v1/2;
"`AES-NI, SIMD`":the aes256ctr function from [56], used in LAC-v3.

6.2 Secret and Error Polynomial Generation

A trivial idea to generate a polynomial from Ψ_n^h is to randomly set $h/2$ positions to 1, $h/2$ positions to -1, and the rest $n-h$ positions to 0. This method is not constant time, and can only be used during encryption.

Under the CCA security, the decryptor will need to re-encrypt the message. Therefore, it is desirable to have a constant time sampler that can be used by both encryption and decryption. Sampling in constant time from Ψ_n^h has already been implemented in [9,12]. Here, we use a permutation based solution that is both efficient and constant time.

The algorithm "gen_r" outputs a polynomial $\vec{r} \in \Psi_n^h$ as follows:

1. **Init:** Set $r_i = i$ for $i \in [0, n-1]$.
2. **Permute:** For $i \in [0, n-1]$, generate a random $s_i \in [i, n-1]$, permute r_i and r_{s_i}.
3. **Output:** Output the first h positions of $\vec{r}$.

The intuition is that, gen_r is a map from a fixed element to another element in the support of Ψ_n^h. For each position i, an random index r_{s_i} will be selected. If we sample $s_i \in [i, n-1]$ randomly, then we get a shuffle of $i \in [0, n-1]$.

In the algorithm above, the trivial approach to generate $s_i \in [i, n-1]$ is sample rejection. However, this will cause non-constant executing time. In our implementation, for the sake of constant time, we use a sightly different form of the above procedure as follows:

1. **Init:** Set $r_i = i$ for $i \in [0, n-1]$, and $p = 0$.
2. **Permute:** For $i \in [0, repeat]$, generate a random $s_i \in [0, n-1]$, if $s_i \geq i$, then permute r_p and r_{s_i}, update $p = p + 1$, else permute r_p and r_p.
3. **Output:** Output the first h positions of $\vec{r}$.

The second step will repeat enough times to guarantee that the first h positions of $\vec{r}$ are permuted. For each security level l, we set the value "repeat" to make sure that $\Pr[p \geq h] < 2^{-l}$. The performance of this subroutine is described in Table 6.

n (distribution)	prf (with aes256ctr)	repeat times	performance (Cycles)
Ψ_{512}^{128}	openssl	245	4978
Ψ_{512}^{256}	openssl	590	10446
Ψ_{1024}^{256}	openssl	495	9566
Ψ_{1024}^{384}	openssl	815	14761
Ψ_{512}^{128}	`AES-NI, SIMD`	245	2040
Ψ_{512}^{256}	`AES-NI, SIMD`	590	4466
Ψ_{1024}^{256}	`AES-NI, SIMD`	495	3834
Ψ_{1024}^{384}	`AES-NI, SIMD`	815	6118

Follows the notations of Table 5.

Table 6. Performance of Secret and Error Polynomial Generation

6.3 Polynomial Multiplication

We provide three types of polynomial multiplication targeting different use cases:

- **ref:** The reference implementation is the most general version that only requires 32bits registers. It can be executed on most of the personal computers or servers, and can be easily translated to embedded platforms such as Cortex M4.
- **opt:** The optimized implementation uses a 64bits variant. It packs four 16bits intermediate values. This implementation can be used in any 64bits platforms.
- **avx:** The avx version use a 256bits register and the AVX2 instructions. It packs 16 or 32 intermediate values. The avx version relies on the hardware support of AVX2 instruction sets.

Now we are ready to describe our algorithm for polynomial multiplications, that is adopted in all three versions.

Given $\vec{a}$ and $\vec{r}$, let $\vec{v} = \langle -a_0, \cdots, -a_{n-1}, a_0, \cdots, a_{n-1} \rangle$, according to the definition of $\mathbb{Z}_q[x]/(x^n + 1)$, we can compute $\vec{b} = \vec{ar}$ as:

$$\vec{b} = \sum_{i=0}^{n-1} (\vec{v}[n-i : 2*n-i-1] \cdot r_i),$$

where $\vec{v}[x : y]$ denote the vector of $\{v_x, \cdots, v_y\}$. Note that, in our scheme $r_i \in \{-1, 0, 1\}$, so the computation can be simplified as:

$$\vec{b} = \sum_{r_i==1} \vec{v}[n-i : 2*n-i-1] - \sum_{r_i==-1} \vec{v}[n-i : 2*n-i-1].$$

Concretely, when $\vec{r} \in \Psi_n^h$ is in the form of position based representation (the first $h/2$ positions stores the indexes that r_i==1, the second $h/2$ positions stores the indexes that $r_i == -1$) the ref version of our implementation works as follows:

1. **Prepare $\vec{v}$:** Set $\vec{v} = \{-a_0, \cdots, -a_{n-1}, a_0, \cdots, a_{n-1}\}$.
2. **Init accumulator:** Initialize the intermediate accumulators:

   $$\vec{sum\_one} = 0, \vec{sum\_mone} = 0,$$

   where $\vec{sum\_one}$ is the accumulator vector for $r_i == 1$ and $\vec{sum\_mone}$ is the accumulator for vector $r_i == -1$.
3. **Addition:** For $i = 0$ to $i = h/2 - 1$, compute:

   $$\vec{sum\_one} = \vec{sum\_one} + \vec{v}[n - r_i : 2*n - r_i - 1],$$

   $$\vec{sum\_mone} = \vec{sum\_mone} + \vec{v}[n - r_{i+h/2} : 2*n - r_{i+h/2} - 1].$$

4. **Output:** Output $\vec{b} = \vec{sum\_one} - \vec{sum\_mone} \mod q$.

Note that, when the intermediate accumulators are stored with `uint16_t`, we only need execute the mod$q$ operation for once in the last step, instead for each iteration during the third step.

In addition, our opt and avx implementations use various parallalizations, depending on the computer architecture and the modulus q. We summarize the performance in Table 7.

distribution	q (modulus)	ref(cycles)	opt(cycles)	avx(cycles)
Ψ_{512}^{128}	251	23,792	15,553	7,008
Ψ_{512}^{256}	251	46,718	28,377	13,259
Ψ_{1024}^{256}	251	92,818	57,842	27,901
Ψ_{1024}^{384}	251	138,333	80,649	41,100
Ψ_{512}^{128}	256	23,424	14,057	3,661
Ψ_{512}^{256}	256	46,391	26,883	7,094
Ψ_{1024}^{256}	256	91,934	52,468	14,062
Ψ_{1024}^{384}	256	137,223	77,582	20,950

Table 7. Performance of Polynomial Multiplication

6.4 Error Correction Code

LAC-v1/v2 uses a generic binary BCH encoding/decoding library [31]. This implementation is not constant time, and may lead to timing and memory attacks [27]. In LAC-v3, we used an constant time implementation, modified from [31]. We highlight our modifications as follows.

- **Constant Looping Statements:** There are three loops to compute the syndromes, error location polynomials and roots. In [31], the number of iterations in each loop may be reduced based on the input data, in order to improve efficiency. In our modification, these loops will iterate for the maximum value regardless of the input.
- **Constant Branching Statements:** We removed the branching statements and used the masking technique to control whether a particular value gets updated or not. In [31], within error location polynomial computation, two branching structures ("if") are used to decide whether the intermediate status will be updated. We removed the branching statements and used masking a value to control the update of the status.

In [31], looking up tables are also frequently used. This may be vulnerable to cache attacks. Naive methods for constant time array access will decrease performance drastically. Our implementation unfortunately still uses a same lookup table. We leave the cache attacks resistant implementation to future research. The performance of BCH[511,128,17], BCH[511,256,17] and BCH[511,256,41] for LAC-192v3 is described in Table 8. We test the performance of different errors from 0 to 8. As one shall see from the table, there is an almost negligible deviation of the decoder.

Number of errors	BCH[511,128,17]	BCH[511,256,17]	BCH[511,256,41]
0	8521	13983	49041
1	8530	13994	49054
2	8533	13999	49058
3	8533	13994	49062
4	8534	13992	49053
5	8532	14004	49026
6	8537	14002	49044
7	8541	14005	49046
8	8524	14004	49045

Table 8. Performance of BCH, in Cycles

6.5 Performance

The performance of LAC-v3 is described in Table 9. where ref denotes the reference version, opt denotes the optimized version over 64bits instruction, avx denotes the optimized version over `AVX2 SIMD` instructions. There main difference among these three versions is their underlying polynomial multiplication. In the ref and opt version, openssl based pseudo-random function is used in the secret/noise polynomial generation. In the avx version, the `AES-NI` & `SIMD` based pseudo-random function is used the secret/noise polynomial generation. The performance results are tested on the ubuntu 18.04 operation system running on the Intel Xeon(R)W-2123 @ 3.6GHz, memory 15.3GB, with Turbo Boost and Hyperthreading disabled.

Scheme	**ref (Cycles)**			**opt (Cycles)**			**avx (Cycles)**		
	gen	**enc**	**dec**	**gen**	**enc**	**dec**	**gen**	**enc**	**dec**
light-v3a	51,049	74,328	90,024	39,712	57,111	66,017	19,139	28,064	32,396
128-v3a	88,765	143,740	193,355	65,663	104,660	137,344	28,759	46,068	63,672
192-v3a	149,212	234,636	320,457	110,839	175,077	238,633	46,322	76,208	113,029
256-v3a	231,417	410,739	618,689	151,312	261,869	403,563	62,894	112,060	213,001
light-v3b	50,465	74,963	90,839	35,311	51,694	59,897	13,388	20,775	22,094
128-v3b	87,696	145,173	194,307	60,293	98,334	120,958	20,703	33,546	45,165
192-v3b	141,331	227,948	313,432	101,010	163,770	222,955	27,242	45,531	68,707
256-v3b	213,731	381,269	574,613	133,323	240,771	381,833	38,230	68,320	148,468

Table 9. Performance of CCA Secure LAC

6.6 Comparison

We compare LAC with the other ring LWE based public key encryption schemes in Table 10. Performance results of LAC are test on ubuntu 18.04 operation system running on the Intel Xeon(R)W-2123 @ 3.6GHz, memory 15.3GB, with

Turbo Boost and Hyperthreading disabled. Parameters of the other 5 schemes are obtained from their Round-2 submission to NIST-PQC [56]. Note that ThreeBears stores seeds instead of the secret key to minimize the storage. As a trade-off it is computationally more costly decryption to regenerate the keys.

We also provide a performance comparison according to each security level in Fig 1, 2 and 3, for the ease of comparison. Note that, since the concrete security of LAC-192-v3a and LAC-192-v3b under the BKZ-core-(Q)Sieving modelmeets level 5, they are listed in both Fig 2 and 3.

NIST-PQC category	Scheme	Size (in Bytes)			AVX2 10^3 Cycles			Q-Security
		sk	pk	ct	gen	enc	dec	
N/A	LAC-light-v3b	1056	544	664	13	21	22	117
	LAC-light-v3a	1056	544	664	19	28	32	118
I	NewHope512	1888	928	1120	68	110	114	101
	LAC-128-v3b	1056	544	704	21	34	45	133
	LAC-128-v3a	1056	544	704	29	46	64	133
	Kyber512-90s	1632	800	736	20	30	25	100
	Kyber512	1632	800	736	33	49	41	100
	BabyBear	40	804	917	41	60	101	140
	R5ND_1PKE_0d	708	676	756	65	100	141	118
	R5ND_1PKE_5d	493	461	636	50	85	122	120
	LightSaber	1568	672	736	62	73	71	114
III	LAC-192-v3b	2080	1056	1352	27	46	69	258
	LAC-192-v3a	2080	1056	1352	46	76	113	259
	Kyber768-90s	2400	1184	1088	31	46	38	164
	Kyber768	2400	1184	1088	62	84	70	164
	MamaBear	40	1194	1307	79	96	156	213
	R5ND_3PKE_0d	1031	983	1119	88	138	192	180
	R5ND_3PKE_5d	828	780	950	91	159	237	176
	Saber	2304	992	1088	104	122	120	185
V	NewHope1024	3680	1824	2208	130	210	221	233
	LAC-256-v3b	2080	1056	1464	38	68	148	276
	LAC-256-v3a	2080	1056	1464	63	112	213	277
	Kyber1024-90s	3168	1568	1568	44	64	54	230
	Kyber1024	3168	1568	1568	89	116	100	230
	PapaBear	40	1584	1697	118	145	211	285
	R5ND_5PKE_0d	1413	1349	1525	104	169	235	246
	R5ND_5PKE_5d	1042	978	1301	143	241	365	232
	FireSaber	3040	1312	1472	161	185	187	257

Follows the notations of Table 1.

Table 10. Comparison of ring LWE based PKE in NIST-PQC Round2

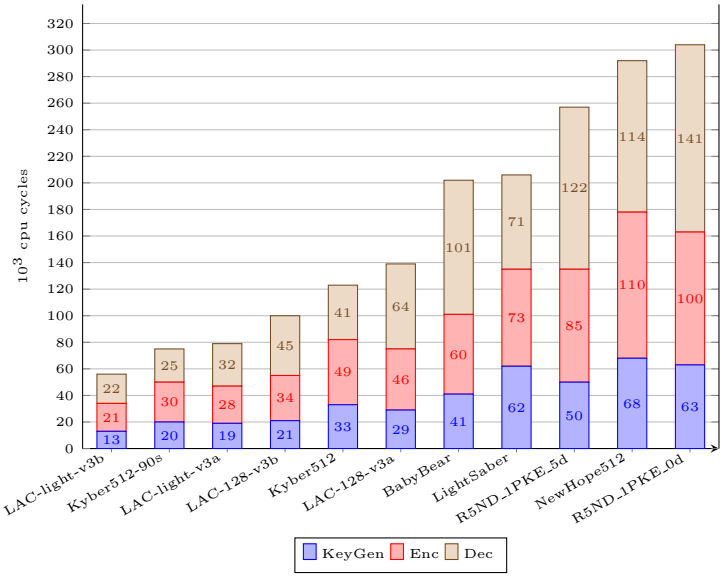

Fig. 1. Performance of 128-bits security level (AVX2 version)

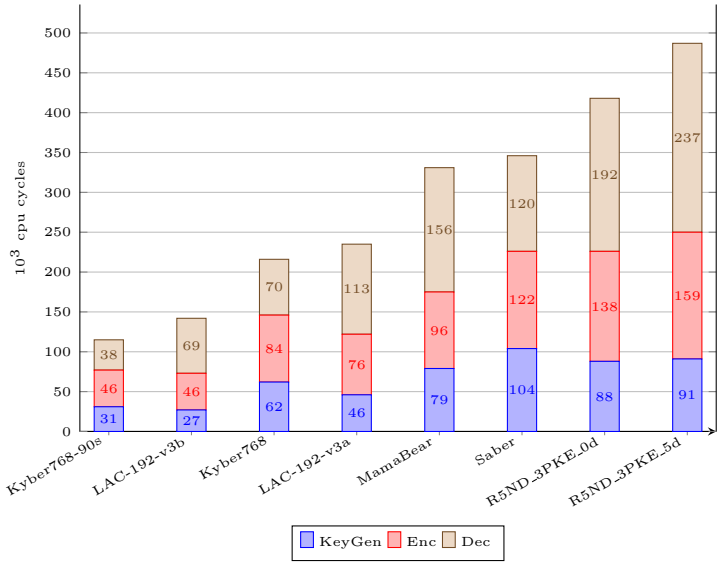

Fig. 2. Performance of 192-bits security level (AVX2 version)

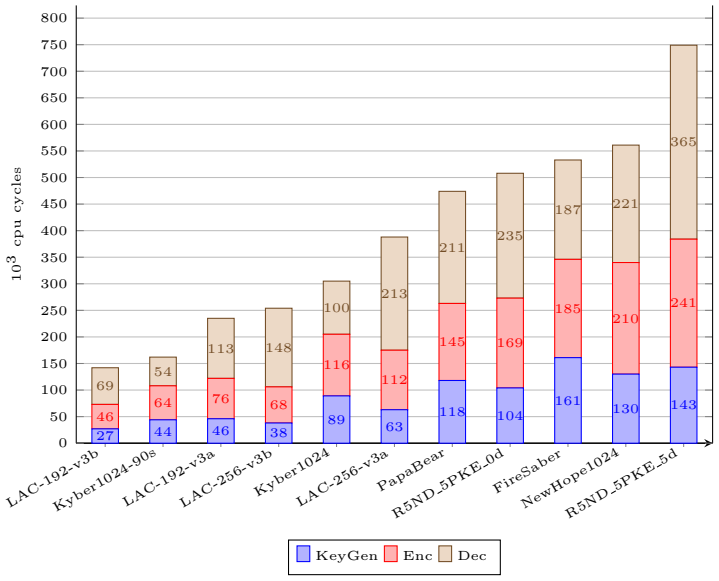

Fig. 3. Performance of 256-bits security level (AVX2 version)

7 Conclusion

Better instantiations and implementations of (Ring) LWE based PKE/KEX schemes have been a major topic for post-quantum cryptography community for the past years. Building on top of previous results, our work pushes the size limitation of post-quantum cryptography further. In parallel to rounding, modular lattices and other proposal, our idea of byte size modulus plus large block error corrections provides a new approach to get compact and efficient lattice based public key encryption.

References

1. Shweta Agrawal, Dan Boneh, and Xavier Boyen. Efficient lattice (H)IBE in the standard model. In *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Monaco / French Riviera, May 30 - June 3, 2010. Proceedings*, pages 553–572, 2010.
2. Shweta Agrawal, Dan Boneh, and Xavier Boyen. Lattice basis delegation in fixed dimension and shorter-ciphertext hierarchical IBE. In *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings*, pages 98–115, 2010.
3. Martin R. Albrecht, Shi Bai, and Léo Ducas. 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, pages 153–178, 2016.

4. Martin R. Albrecht, Benjamin R. Curtis, Amit Deo, Alex Davidson, Rachel Player, Eamonn W. Postlethwaite, Fernando Virdia, and Thomas Wunderer. Estimate all the {LWE, NTRU} schemes! In *Security and Cryptography for Networks - 11th International Conference, SCN 2018, Amalfi, Italy, September 5-7, 2018, Proceedings*, pages 351–367, 2018.
5. Martin R. Albrecht, Florian Göpfert, Fernando Virdia, and Thomas Wunderer. 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*, pages 297–322, 2017.
6. Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *J. Mathematical Cryptology*, 9(3):169–203, 2015.
7. Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Newhope without reconciliation. *IACR Cryptology ePrint Archive*, 2016:1157, 2016.
8. Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange - A new hope. In *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016.*, pages 327–343, 2016.
9. Hayo Baan, Sauvik Bhattacharya, Óscar García-Morchón, Ronald Rietman, Ludo Tolhuizen, Jose Luis Torre-Arce, and Zhenfei Zhang. Round2: KEM and PKE based on GLWR. *IACR Cryptology ePrint Archive*, 2017:1183, 2017.
10. Dan Bernstein. A subfield-logarithm attack against ideal lattices, 2014.
11. Daniel J. Bernstein, Chitchanok Chuengsatiansup, Tanja Lange, and Christine van Vredendaal. 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*, pages 235–260, 2017.
12. Sauvik Bhattacharya, Óscar García-Morchón, Thijs Laarhoven, Ronald Rietman, Markku-Juhani O. Saarinen, Ludo Tolhuizen, and Zhenfei Zhang. Round5: Compact and fast post-quantum public-key encryption. *IACR Cryptology ePrint Archive*, 2018:725, 2018.
13. Jean-François Biasse, Thomas Espitau, Pierre-Alain Fouque, Alexandre Gélin, and Paul Kirchner. 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*, pages 60–88, 2017.
14. Joppe W. Bos, Craig Costello, Léo Ducas, Ilya Mironov, Michael Naehrig, Valeria Nikolaenko, Ananth Raghunathan, and Douglas Stebila. Frodo: Take off the ring! practical, quantum-secure key exchange from LWE. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 1006–1018, 2016.
15. Joppe W. Bos, Craig Costello, Michael Naehrig, and Douglas Stebila. 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*, pages 553–570, 2015.
16. Joppe W. Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, and Damien Stehlé. Crystals-kyber: a cca-secure module-lattice-based kem. *IACR Cryptology ePrint Archive*, 2017:634, 2017.

17. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*, pages 309–325, 2012.
18. Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 97–106, 2011.
19. Leon Groot Bruinderink, Andreas Hülsing, Tanja Lange, and Yuval Yarom. 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*, pages 323–345, 2016.
20. David Cash, Dennis Hofheinz, Eike Kiltz, and Chris Peikert. Bonsai trees, or how to delegate a lattice basis. In *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Monaco / French Riviera, May 30 - June 3, 2010. Proceedings*, pages 523–552, 2010.
21. Lily Chen, Stephen Jordan, Yi-Kai Liu, Dustin Moody, Rene Peralta, Ray Perlner, and Daniel Smith-Tone. Report on post-quantum cryptography. Technical report, NIST, 2016.
22. Yuanmi Chen and Phong Q. Nguyen. 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*, pages 1–20, 2011.
23. Wei Dai, William Whyte, and Zhenfei Zhang. Optimizing polynomial convolution for ntruencrypt. *IACR Cryptology ePrint Archive*, 2018:229, 2018.
24. Oded Regev Daniele Micciancio. Lattice-based cryptography. Technical report, NYU, https://cims.nyu.edu/ regev/papers/pqc.pdf, 2008.
25. Jan-Pieter D'Anvers, Qian Guo, Thomas Johansson, Alexander Nilsson, Frederik Vercauteren, and Ingrid Verbauwhede. Decryption failure attacks on IND-CCA secure lattice-based schemes. In *Public-Key Cryptography - PKC 2019 - 22nd IACR International Conference on Practice and Theory of Public-Key Cryptography, Beijing, China, April 14-17, 2019, Proceedings, Part II*, pages 565–598, 2019.
26. Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, and Frederik Vercauteren. Saber: Module-lwr based key exchange, cpa-secure encryption and cca-secure KEM. In *Progress in Cryptology - AFRICACRYPT 2018 - 10th International Conference on Cryptology in Africa, Marrakesh, Morocco, May 7-9, 2018, Proceedings*, pages 282–305, 2018.
27. Jan-Pieter D'Anvers, Marcel Tiepelt, Frederik Vercauteren, and Ingrid Verbauwhede. Timing attacks on error correcting codes in post-quantum secure schemes. Cryptology ePrint Archive, Report 2019/292, 2019. `https://eprint.iacr.org/2019/292`.
28. Jan-Pieter D'Anvers, Frederik Vercauteren, and Ingrid Verbauwhede. The impact of error dependencies on ring/mod-lwe/lwr based schemes. *IACR Cryptology ePrint Archive*, 2018:1172, 2018.
29. Jan-Pieter D'Anvers, Frederik Vercauteren, and Ingrid Verbauwhede. On the impact of decryption failures on the security of LWE/LWR based schemes. *IACR Cryptology ePrint Archive*, 2018:1089, 2018.
30. Jintai Ding. A simple provably secure key exchange scheme based on the learning with errors problem. *IACR Cryptology ePrint Archive*, 2012:688, 2012.

31. Ivan Djelic. Bch source code. `https://github.com/jkent/python-bchlib/tree/master/bchlib`.
32. Léo Ducas, Alain Durmus, Tancrède Lepoint, and Vadim Lyubashevsky. 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*, pages 40–56, 2013.
33. Scott R. Fluhrer. Cryptanalysis of ring-lwe based key exchange with key share reuse. *IACR Cryptology ePrint Archive*, 2016:85, 2016.
34. Tim Fritzmann, Thomas Pöppelmann, and Johanna Sepúlveda. Analysis of error-correcting codes for lattice-based key exchange. *IACR Cryptology ePrint Archive*, 2018:150, 2018.
35. Eiichiro Fujisaki and Tatsuaki Okamoto. 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*, pages 53–68, 1999.
36. Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. *J. Cryptology*, 26(1):80–101, 2013.
37. Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, pages 197–206, 2008.
38. Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, pages 75–92, 2013.
39. Florian Göpfert, Christine van Vredendaal, and Thomas Wunderer. 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*, pages 184–202, 2017.
40. Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. NTRU: A ring-based public key cryptosystem. In *Algorithmic Number Theory, Third International Symposium, ANTS-III, Portland, Oregon, USA, June 21-25, 1998, Proceedings*, pages 267–288, 1998.
41. Nick Howgrave-Graham. 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*, pages 150–169, 2007.
42. Haodong Jiang, Zhenfeng Zhang, Long Chen, Hong Wang, and Zhi Ma. 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*, pages 96–125, 2018.
43. Zhengzhong Jin and Yunlei Zhao. Optimal key consensus in presence of noise. *CoRR*, abs/1611.06150, 2016.
44. Paul Kirchner and Pierre-Alain Fouque. 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*, pages 3–26, 2017.

45. Richard Lindner and Chris Peikert. 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*, pages 319–339, 2011.
46. Xiahui Lu. Lac source code. `https://github.com/pqc-lac/lac-intel64`.
47. Xianhui Lu, Yamin Liu, Zhenfei Zhang, Dingding Jia, Haiyang Xue, Jingnan He, and Bao Li. Lac: Practical ring-lwe based public-key encryption with byte-level modulus. Technical report, IIE, Chinese Academy of Sciences, https://eprint.iacr.org/2018/1009, 2018.
48. Vadim Lyubashevsky and Daniele Micciancio. Generalized compact knapsacks are collision resistant. In *Automata, Languages and Programming, 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part II*, pages 144–155, 2006.
49. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. 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*, pages 1–23, 2010.
50. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. A toolkit for ring-lwe cryptography. In *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, pages 35–54, 2013.
51. Vadim Lyubashevsky and Gregor Seiler. NTTRU: truly fast NTRU using NTT. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(3):180–201, 2019.
52. Daniele Micciancio. Generalized compact knapsacks, cyclic lattices, and efficient one-way functions from worst-case complexity assumptions. In *43rd Symposium on Foundations of Computer Science (FOCS 2002), 16-19 November 2002, Vancouver, BC, Canada, Proceedings*, pages 356–365, 2002.
53. Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, pages 700–718, 2012.
54. NIST. NIST PQC FORUM: LAC.
55. NIST. Nist post-quantum cryptography project. Technical report, NIST, 2017.
56. NIST. Nist post-quantum cryptography project round-2. Technical report, NIST, 2019.
57. Chris Peikert. Public-key cryptosystems from the worst-case shortest vector problem: extended abstract. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 333–342, 2009.
58. Chris Peikert. Lattice cryptography for the internet. In *Post-Quantum Cryptography - 6th International Workshop, PQCrypto 2014, Waterloo, ON, Canada, October 1-3, 2014. Proceedings*, pages 197–219, 2014.
59. Chris Peikert. 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*, pages 411–430, 2016.
60. Chris Peikert, Oded Regev, and Noah Stephens-Davidowitz. 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*, pages 461–473, 2017.

61. Chris Peikert and Brent Waters. Lossy trapdoor functions and their applications. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, pages 187–196, 2008.
62. Jing Yang Qian Guo, Thomas Johansson. A novel cca attack using decryption errors against lac. In *Asiacrypt2019 (accepted)*, 2019.
63. Oded Regev. 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*, pages 84–93, 2005.
64. Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6):34:1–34:40, 2009.
65. Miruna Rosca, Damien Stehlé, and Alexandre Wallet. 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*, pages 146–173, 2018.
66. Markku-Juhani O. Saarinen. On reliability, reconciliation, and error correction in ring-lwe encryption. *IACR Cryptology ePrint Archive*, 2017:424, 2017.
67. John Schanck. Estimator: Scripts for estimating the security of lattice based cryptosystems. `https://github.com/jschanck/estimator`.
68. Markus Schmidt and Nina Bindel. Estimation of the hardness of the learning with errors problem with a restricted number of samples. *IACR Cryptology ePrint Archive*, 2017:140, 2017.
69. Damien Stehlé, Ron Steinfeld, Keisuke Tanaka, and Keita Xagawa. Efficient public key encryption based on ideal lattices. In *Advances in Cryptology - ASIACRYPT 2009, 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Proceedings*, pages 617–635, 2009.
70. Thomas Wunderer. Revisiting the hybrid attack: Improved analysis and refined security estimates. *IACR Cryptology ePrint Archive*, 2016:733, 2016.

A Previous parameter sets

Categories	n	q	**dis**	**ecc**	$l_{\vec{m}}$	**pk**	**sk**	**ct**	**bit-er**	**dec-er**
LAC-128-v1	512	251	Ψ_1	BCH[511,264,59]	256	544	512	1024	$2^{-13.35}$	$2^{-239.6}$
LAC-192-v1	1024	251	$\Psi_{1/2}$	BCH[511,392,27]	384	1056	1024	1536	$2^{-24.51}$	$2^{-253.8}$
LAC-256-v1	1024	251	Ψ_1	BCH[1023,520,111]	512	1056	1024	2048	$2^{-7.48}$	$2^{-115.4}$
LAC-128-v2	512	251	Ψ_{512}^{256}	BCH[511,256,33]	256	544	512	712	$2^{-12.61}$	2^{-116}
LAC-192-v2	1024	251	Ψ_{1024}^{256}	BCH[511,256,17]	256	1056	1024	1188	$2^{-22.27}$	2^{-143}
LAC-256-v2	1024	251	Ψ_{1024}^{512}	BCH[511,256,33]+D2	256	1056	1024	1424	$2^{-12.96}$	2^{-122}

Follows the notations of Table 2.

Table 11. Parameters of LAC-v1 and LAC-v2

B Performance comparison of three versions of LAC

Scheme	AVX2 10^3 Cycles				Speed-up
	KeyGen	Encryption	Decryption	Total	
LAC-128-v1	36	71	90	197	
LAC-128-v2	63	98	156	317	
LAC-128-v3a	29	46	64	139	2.28x
LAC-128-v3b	20	34	45	99	3.20x
LAC-light-v3a	19	28	32	79	4.01x
LAC-light-v3b	13	21	22	56	5.66x
LAC-192-v1	112	167	238	517	
LAC-192-v2	147	204	340	691	
LAC-192-v3a	46	76	113	235	2.94x
LAC-192-v3b	27	46	69	142	4.87x
LAC-256-v1	100	199	363	662	
LAC-256-v2	185	295	433	913	
LAC-256-v3a	63	112	213	388	2.35x
LAC-256-v3b	38	68	148	254	3.59x

Table 12. Performance of avx version of LAC-v1, LAC-v2 and LAC-v3

Test on ubuntu 18.04, Intel Xeon(R)W-2123 @ 3.6GHz, memory 15.3GB, with TurboBoost and Hyperthreading disabled.