



A Novel CCA Attack Using Decryption Errors Against LAC

Qian Guo^{1,2(✉)}, Thomas Johansson^{2(✉)}, and Jing Yang²

¹ Department of Informatics, University of Bergen,
Box 7803, 5020 Bergen, Norway
qian.guo@uib.no

² Department of Electrical and Information Technology, Lund University,
P.O. Box 118, 221 00 Lund, Sweden
{qian.guo, thomas.johansson, jing.yang}@eit.lth.se

Abstract. Cryptosystems based on Learning with Errors or related problems are central topics in recent cryptographic research. One main witness to this is the NIST Post-Quantum Cryptography Standardization effort. Many submitted proposals rely on problems related to Learning with Errors. Such schemes often include the possibility of decryption errors with some very small probability. Some of them have a somewhat larger error probability in each coordinate, but use an error correcting code to get rid of errors. In this paper we propose and discuss an attack for secret key recovery based on generating decryption errors, for schemes using error correcting codes. In particular we show an attack on the scheme LAC, a proposal to the NIST Post-Quantum Cryptography Standardization that has advanced to round 2.

In a standard setting with CCA security, the attack first consists of a precomputation of special messages and their corresponding error vectors. This set of messages are submitted for decryption and a few decryption errors are observed. In a statistical analysis step, these vectors causing the decryption errors are processed and the result reveals the secret key. The attack only works for a fraction of the secret keys. To be specific, regarding LAC256, the version for achieving the 256-bit classical security level, we recover one key among approximately 2^{64} public keys with complexity 2^{79} , if the precomputation cost of 2^{162} is excluded. We also show the possibility to attack a more probable key (say with probability 2^{-16}). This attack is verified via extensive simulation.

We further apply this attack to LAC256-v2, a new version of LAC256 in round 2 of the NIST PQ-project and obtain a multi-target attack with slightly increased precomputation complexity (from 2^{162} to 2^{171}). One can also explain this attack in the single-key setting as an attack with precomputation complexity of 2^{171} and success probability of 2^{-64} .

Keywords: Chosen-ciphertext security · Decryption errors · Lattice-based cryptography · NIST post-quantum standardization · LAC · LWE · Reaction attack

1 Introduction

Lattice-based cryptography and the learning with errors problem (LWE) [24] is now one of the main research areas in cryptography. Factoring and the discrete logarithm problem have always been the fundamental basis in modern cryptography, but due to the threat of quantum computers, this will change. Lattice-based cryptography is the enabler for a rich collection of cryptographic primitives, ranging from key exchange, KEMs, encryption and digital signature to more advanced constructions like fully homomorphic encryption.

There are several reasons for using LWE or related problems as the underlying problem in cryptographic constructions. One is that constructions can be computationally very efficient compared to existing solutions. Another motivation is that LWE-based constructions may be resistant to quantum computers. It is also potentially the way how one can best provide constructions of fully homomorphic encryption [5, 7].

An important problem is to establish the difficulty of solving various LWE-like problems, as it directly determines an upper bound on the security for a construction. One can use reductions for LWE to worst-case lattice problems [6, 23, 24], but it may not always be applicable or it may not give useful help in choosing optimal parameters. As of today, the security of a primitive is often estimated from the computational complexity of lattice-basis reduction algorithms like BKZ and its different versions.

Recent developments in several areas where problems may potentially be difficult even for a quantum computer, motivated several standardization projects, and some time ago the NIST post-quantum standardization project [2] started. In the specification of the analysis of submitted proposals, the most important aspect was said to be their security. Typically, the computational complexity for solving problems like LWE through lattice basis reduction is the guide when explicitly suggesting parameters in the different constructions. Most proposals have some proof of security, relating to some well known and difficult problems in lattice theory, such as the shortest vector problem. Most lattice-based schemes include also the possibility of having decryption errors with some small probability. Making this probability zero has a price, as the parameters should be adjusted accordingly, resulting in a performance loss. So many schemes tolerate a very small probability of decryption error, say something of size 2^{-128} .

An approach used by some schemes to enhance the performance is to allow a larger error probability in each position and then use error-correcting codes to correct the errors that occurred. In essence, part of the message information are parity-check bits that enable correction of up to a fixed number of errors. Such schemes can thus have a larger error probability in each bit position, as it requires that a number of them are in error for a decryption error to occur. Still, the possibility of having decryption errors can be used in cryptanalysis and the motivation for this paper is to further examine such possibilities.

We specifically focus on the proposal LAC, a scheme that has now advanced to round 2 in the NIST project. LAC is perhaps the most extreme scheme among the LWE-based schemes in the NIST project. It has a very small modulus, $q = 251$,

which makes it very interesting. It leads to a rather large probability of error in a single position ($2^{-7.4}$), but then it uses a strong error correcting code to correct up to 55 errors, resulting in a small overall probability of decryption error (2^{-115}). LAC has excellent performance and is indeed an elegant design.

In our attack we consider CCA (chosen-ciphertext attacks) security for PKE (public-key encryption) schemes and use the algorithms as specified in the LAC design document.

1.1 Related Works

The use of decryption errors in cryptanalysis has been frequently used in all areas of cryptography, e.g., [4]. For lattice-based encryption systems and NTRU, some works in this direction are listed [12, 16–18].

More recently, Fluhrer [11] showed an attack on key-exchange protocols in a key reuse setting and [9] extended the attack. In [3] a chosen-ciphertext attack on the recent proposal HILA5 [25] was described, using decryption errors. These attacks can be described as CCA type attacks on proposals without CCA transforms.

Here we will only consider CCA attacks on schemes proposed for CCA security. For such a case, an attack model for LWE-based schemes and a specific attack on ss-ntru-pke, another NIST submission, was given in the recent paper [8]. We base the attack in this paper on the same model. For the specific case of LAC, there has also been some discussion on the NIST forum, on how to increase the probability of decryption errors [1].

For code-based schemes, Guo, Johansson and Stankovski [14] proposed a key-recovery attack against the CCA-secure version of QC-MDPC. They used a property that ‘colliding pairs’ in the noise and the secret can change the decryption failure rate. In the statistical analysis in this paper, we use some kind of similar idea, identifying similar patterns between a part of the secret key and error vectors.

1.2 Contributions

In this paper we describe an attack for secret key recovery based on generating decryption errors, where error correcting codes are used. It is applied on the CCA version of the proposal LAC and it is a chosen-ciphertext attack. The attack is described as a sequence of steps. The first step is a precomputation phase where messages generating special error vectors are found. In the second step we send these encrypted messages for decryption and some decryption errors are observed. Finally, the major part of the attack is the last step, in which a statistical analysis of the messages/errors causing the decryption errors are analyzed. In particular, we identify a correlation between consecutive positions in the secret key and consecutive positions in error vectors that can be used to restore the secret vector. The attack success is conditioned on a certain weight-property of the secret key, causing the decoding error probability to be significantly higher than that in the average case. In particular, we describe

the details of an attack to LAC256 with success probability¹ larger than 2^{-64} with complexity less than 2^{79} , assuming a single precomputation of complexity 2^{162} encryptions. The statistical analysis is supported by extensive simulation results².

We also extend our approach to attacking a new version of LAC256 in round 2 of the NIST PQ-project. We design a new desired noise pattern that can lead to a high decryption error probability. For instance, with the precomputation of about 2^{120} for one chosen message/error, the error probability is simulated to be $2^{-12.74}$, for a key with probability 2^{-64} . Using this error pattern, one could classically solve LAC256-v2 with complexity far less than that of the claimed security level by our estimation.

1.3 Organization

The remaining of the paper is organized as follows. In Sect. 2 we describe the LAC proposal from the NIST Post-Quantum standardization process. In Sect. 3, we present the main attack procedure, which is followed by a section elaborating the statistical analysis step, i.e., how to reconstruct the secret key from the decryption failures. Section 5 shows how to apply the proposed attack to the new LAC version in round 2 of the NIST PQ-project, and Sect. 6 includes related discussions. Finally, we present the conclusion in Sect. 7.

2 Description of LAC

LAC [19] is a proposal in the NIST Post-Quantum competition, including three versions for different security levels, i.e., LAC128, LAC192, and LAC256. We focus in this paper only on attacking LAC256. Also, we consider only CCA security as a CPA-version is almost trivially broken in a reaction attack model.

2.1 Some Basic Notation

Let \mathbb{Z}_q be the ring of integers modulo q represented in $(-q/2, q/2]$ and let \mathcal{R} denote the ring $\mathbb{Z}_q[X]/(X^n+1)$. Consider the one-to-one correspondence between polynomials in \mathcal{R} and vectors in \mathbb{Z}_q^n . Vectors will be represented with bold lower-case letters, while matrices are written in uppercase. For a vector \mathbf{a} , the transpose vector is written \mathbf{a}^T .

The Euclidean norm of a polynomial $a \in \mathcal{R}$ is written as $\|a\|_2$ and defined as $\sqrt{\sum_i a_i^2}$, which is extended to vectors as $\|\mathbf{a}\|_2 = \sqrt{\sum_i \|\mathbf{a}_i\|_2^2}$. The notation $a \stackrel{s}{\leftarrow} \chi(\mathcal{R})$ will be used to represent the sampling of $a \in \mathcal{R}$ according to the

¹ Assuming for 2^{64} users in the system is considered as a reasonable setting in the NIST PQC project discussion forum [1].

² The implementation is available at: <https://github.com/MelodyJuly/A-Novel-CCA-Attack-using-Decryption-Errors-against-LAC>.

distribution χ . Writing $\text{Samp}(\chi; \text{seed})$ means computing an output following the distribution χ using seed as the seed.

A distribution used in LAC is the centered binomial distribution, denoted Ψ_σ^n . In particular, in LAC256 one uses Ψ_1 , which is the distribution on $\{-1, 0, 1\}$, where $P(X = 0) = 1/2$ and $P(X = -1) = P(X = 1) = 1/4$ for $X \stackrel{\$}{\leftarrow} \Psi_1$. Note that the mean is 0 and the variance is $1/2$, so for Ψ_1^n the variance is $n/2$. We also denote $U(\mathcal{R})$ the uniform distribution on \mathcal{R} .

For cryptographic schemes of this type, the definition of security is to (at least) fulfill the concept of indistinguishability under adaptive chosen ciphertext attacks, denoted IND-CCA2. This is usually described through the *advantage* of a certain security game where the adversary may adaptively ask for decryptions of various ciphertexts, except the one that is given as the challenge. As our attack is more direct and simply tries to recover the secret key, we do not further introduce notions of security. We note however that all results given can be translated to corresponding results in the form of advantage of security games in the IND-CCA2 model.

2.2 The LAC Scheme

LAC is a concrete instantiation of a general construction proposed in [22] where the novelty lies in the combination of a very small q together with a very strong error correcting procedure, which allows to have many errors in different positions and still be able to correctly decrypt to the message used in encryption with a very large probability.

The key generation algorithm of LAC is shown in Algorithm 1. The encapsulation algorithm Enc is shown in Algorithm 2, and the decapsulation algorithm Dec is shown in Algorithm 3. These algorithms call the CPA-secure schemes described in Algorithms 4–5. For more details we refer to the original design document [19].

Algorithm 1. LAC.KeyGen()

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

- 1) $\text{seed}_a \stackrel{\$}{\leftarrow} \mathcal{S}$;
 - 2) $\mathbf{a} \leftarrow \text{Samp}(U(\mathcal{R}); \text{seed}_a) \in \mathcal{R}$;
 - 3) $\mathbf{s} \stackrel{\$}{\leftarrow} \Psi_\sigma^n$;
 - 4) $\mathbf{e} \stackrel{\$}{\leftarrow} \Psi_\sigma^n$;
 - 5) $\mathbf{b} \leftarrow \mathbf{a}\mathbf{s} + \mathbf{e} \in \mathcal{R}$;
 - 6) **return** $(pk := (\text{seed}_a, \mathbf{b}), sk := \mathbf{s})$;
-

Recall that our prime target LAC256, uses Ψ_1 , a distribution that is 1 (or -1) with probability $1/4$ and 0 with probability $1/2$. We assume that $l_v = n$.

Algorithm 2. LAC.CCA.Enc($pk; \text{seed}_m$)

Output: A ciphertext and encapsulation key pair (\mathbf{c}, K) .

- 1) $\mathbf{m} \leftarrow \text{Samp}(U(\mathcal{M}); \text{seed}_m) \in \mathcal{M}$;
 - 2) $\text{seed} \leftarrow G(\mathbf{m}) \in \mathcal{S}$;
 - 3) $\mathbf{c} \leftarrow \text{LAC.CPA.Enc}(pk, \mathbf{m}; \text{seed})$;
 - 4) $K \leftarrow H(\mathbf{m}, \mathbf{c}) \in \{0, 1\}^{l_k}$;
 - 5) **return** (\mathbf{c}, K) ;
-

Algorithm 3. LAC.CCA.Dec($sk; \mathbf{c}$)

Output: An encapsulation key (K) .

- 1) $\mathbf{m} \leftarrow \text{LAC.CPA.Dec}(sk, \mathbf{c})$;
 - 2) $K \leftarrow H(\mathbf{m}, \mathbf{c})$;
 - 3) $\text{seed} \leftarrow G(\mathbf{m}) \in \mathcal{S}$;
 - 4) $\mathbf{c}' \leftarrow \text{LAC.CPA.Enc}(pk, \mathbf{m}; \text{seed})$;
 - 5) **if** $\mathbf{c}' \neq \mathbf{c}$ **then**
 | $K \leftarrow H(H(sk), \mathbf{c})$;
 - 6) **return** K ;
-

The underlying ring is of the form $\mathcal{R} = \mathbb{Z}_q[x]/(x^n + 1)$, where $n = 1024$ and $q = 251$. One important selling point of this scheme is its much smaller alphabetic size, compared with other lattice-based proposals; this, however, also leads to the main obstacle regarding to its decryption success probability. This scheme targets the highest NIST security level of V, corresponding roughly to 256-bit classical security.

An important part of the scheme is the use of the $\text{ECCEnc}(\mathbf{m})$ subroutine. This part uses a BCH code with length 1023 and dimension 520, which is capable of decoding up to 55 errors and is employed for correcting errors. We assume a decoder for the BCH code that will fail if the number of erroneous positions is 56 or more. All parameters are summarized in Table 1.

A characterizing property of the scheme (as for many other schemes) is the fact that decryption may fail. A main question is to examine the probability of such an event. This is done in the design document [19] and we briefly summarize the results. The error term in LAC, denoted \mathbf{W} , is of the form³

$$\mathbf{W} = \mathbf{e}_1 \mathbf{s} - \mathbf{e} \mathbf{r} + \mathbf{e}_2,$$

since the computation $\mathbf{c}'_m \leftarrow \mathbf{c}_2 - \mathbf{c}_1 \mathbf{s}$ gives

$$\mathbf{c}'_m = (\mathbf{b} \mathbf{r}) + \mathbf{e}_2 + \lfloor \frac{q}{2} \rfloor \cdot \mathbf{c}_m - (\mathbf{a} \mathbf{r} + \mathbf{e}_1) \mathbf{s} = \lfloor \frac{q}{2} \rfloor \cdot \mathbf{c}_m + \mathbf{W}.$$

³ The noise term is equivalent to the representation in [19] due to symmetry.

Algorithm 4. LAC.CPA.Enc($pk = (\text{seed}_a, \mathbf{b}), \mathbf{m} \in \mathcal{M}; \text{seed} \in S$)

Output: A ciphertext \mathbf{c} .

- 1) $\mathbf{a} \leftarrow \text{Samp}(U(\mathcal{R}); \text{seed}_a) \in \mathcal{R}$;
 - 2) $\mathbf{c}_m \leftarrow \text{ECCEnc}(\mathbf{m}) \in \{0, 1\}^{l_v}$;
 - 3) $(\mathbf{r}, \mathbf{e}_1, \mathbf{e}_2) \leftarrow \text{Samp}(\Psi_\sigma^n, \Psi_\sigma^n, \Psi_\sigma^{l_v}; \text{seed})$;
 - 4) $\mathbf{c}_1 \leftarrow \mathbf{a}\mathbf{r} + \mathbf{e}_1 \in \mathcal{R}$;
 - 5) $\mathbf{c}_2 \leftarrow (\mathbf{b}\mathbf{r})_{l_v} + \mathbf{e}_2 + \lfloor \frac{q}{2} \rfloor \cdot \mathbf{c}_m \in \mathbb{Z}_q^{l_v}$;
 - 6) **return** $\mathbf{c} := (\mathbf{c}_1, \mathbf{c}_2) \in \mathcal{R} \times \mathbb{Z}_q^{l_v}$;
-

Algorithm 5. LAC.CPA.Dec($sk = \mathbf{s}; \mathbf{c} = (\mathbf{c}_1, \mathbf{c}_2)$)

Output: A plaintext \mathbf{m} .

- 1) $\mathbf{u} \leftarrow \mathbf{c}_1\mathbf{s} \in \mathcal{R}$;
 - 2) $\mathbf{c}'_m \leftarrow \mathbf{c}_2 - (\mathbf{u})_{l_v} \in \mathbb{Z}_q^{l_v}$;
 - 3) **for** $i = 0$ to $l_v - 1$ **do**
 - if** $\frac{q}{4} \leq \mathbf{c}'_{mi} < \frac{3q}{4}$ **then**
 - $\mathbf{c}_{mi} \leftarrow 1$
 - else**
 - $\mathbf{c}_{mi} \leftarrow 0$
 - 4) $\mathbf{m} \leftarrow \text{ECCDec}(\mathbf{c}_m)$;
 - 5) **return** \mathbf{m} ;
-

Now a single position in \mathbf{W} is essentially a sum of $2n$ random variables, each drawn from a distribution obtained by multiplying two random variables from Ψ_1 . The variance for such a random variable is $1/4$. The sum is then approximated by a Gaussian distribution with mean 0 and variance $2n/4$. A single position is in error if the contribution from \mathbf{W} in that position is larger than $\lfloor \frac{q}{4} \rfloor$ in absolute value, so this gives an error probability in a single position which is roughly

$$\delta = 1 - \text{erf}(62/\sqrt{1024}) \approx 2^{-7.44}.$$

Now, since the error correction procedure corrects up to 55 errors, it is argued in [19] that one can then approximate the overall probability of a decryption error as

$$\sum_{i=56}^{1024} \binom{1024}{i} \delta^i (1 - \delta)^{1024-i} \approx 2^{-115}.$$

Table 1. Proposed parameters of LAC256.

| n | q | \mathcal{R} | Distribution | BCH $[n_e, l_e, d_e, t_e]$ | Security |
|------|-----|---|--------------|----------------------------|----------|
| 1024 | 251 | $\frac{\mathbb{Z}_q[x]}{\langle x^n + 1 \rangle}$ | Ψ_1 | [1023, 520, 111, 55] | V |

Since the stated decryption error probability is very small, the scheme does appear to be quite safe against attacks trying to use the possibility of having decryption errors.

3 The Attack

We first note that LAC is a scheme without protection against multi-target attacks, meaning that precomputed information can be used on any public key. This is because the public key is not included when the seed is computed for generating the noise vectors in encryption. In the code, this is visible in the step 2) $\text{seed} \leftarrow G(\mathbf{m}) \in \mathcal{S}$; of Algorithm 2. It is also a bit unclear how to consider the computational complexity of the precomputation part, as it is something that only needs to be performed once and then never again. At least, as long as the complexity is below 2^{256} encryptions (or 2^{128} in a quantum setting) it should not violate the limits of a successful attack.

We will now present the attack on LAC256 and it is described in three steps; a first step of precomputation; a second step of getting precomputed ciphertexts decrypted and checking the decryption error probability; and a last phase of performing a statistical analysis to recover the secret key.

3.1 Attack Step 1 - Precomputation

We construct a special set \mathcal{S} of messages/error vectors by precomputation. To be precise, we pick a random message \mathbf{m} ($\text{seed}_{\mathbf{m}}$) and compute the seed through the two steps from Algorithm 2:

- 1) $\mathbf{m} \leftarrow \text{Samp}(U(\mathcal{M}); \text{seed}_{\mathbf{m}}) \in \mathcal{M}$;
- 2) $\text{seed} \leftarrow G(\mathbf{m}) \in \mathcal{S}$;

Then compute the noise vectors according to step 3 of Algorithm 4:

- 3) $(\mathbf{r}, \mathbf{e}_1, \mathbf{e}_2) \leftarrow \text{Samp}(\Psi_{\sigma}^n, \Psi_{\sigma}^n, \Psi_{\sigma}^{l_v}; \text{seed})$;

We are now only interested in keeping messages that give rise to noise vectors of special form. In our attack we target only special properties of the \mathbf{e}_1 vector. Let us first consider messages/errors including any combination where the error vector \mathbf{e}_1 contains an interval of consecutive l_1 all positive or all negative entries. Assuming a randomly selected error vector, the probability of finding such an interval starting in the first position is then

$$p = 2 \times (1/4)^{l_1}.$$

As we can start from any position, the probability that a random message/noise vector fulfills the condition for \mathcal{S} can be lower-bounded by

$$p_0 = 2 \times (1/4)^{l_1} \times 3/4 \cdot n. \tag{1}$$

The reason is that if one searches for a desired pattern, an erroneous sequence will on average use $3/4 \cdot 1 + 1/4 \cdot 3/4 \cdot 2 + (1/4)^2 \cdot 3/4 \cdot 3 + \dots \approx 4/3$ positions until there is a possibility for a new desired sequence. We then know Eq. (1) by $p_0 = p \times 3/4 \cdot n$. The precomputation complexity is thus less than $|\mathcal{S}|/p_0$ runs of the steps above. We denote the type of noise vectors of the above kind as *TYPE 1* noise vectors. In particular, we will consider the length⁴ $l_1 \in \{65, 85\}$ when describing the attack by examples.

We note that there are many other special forms of the noise that can be useful in an attack. We define one more such set of special noise vectors related to the \mathbf{e}_1 vector, being the case when \mathbf{e}_1 contains an interval of length $l_0 + l_1$ with at least l_1 either all positive or all negative entries and the remaining entries all-zero. The probability of finding such an interval in \mathbf{e}_1 starting in the first position is then

$$p' = 2 \sum_{i=l_1}^{l_0+l_1} \binom{l_0+l_1}{i} \times (1/4)^i \cdot (1/2)^{l_1+l_0-i}.$$

Determining the probability of having such a subsequence starting from any position is more complicated to compute, but would roughly result in a probability $c \cdot n \cdot p'$ for some not too small constant c . We denote the type of noise vectors of the above kind as *TYPE 2* noise vectors.

Basically, *TYPE 2* noise vectors are much more likely to appear compared to *TYPE 1* noise vectors, so the required precomputation complexity will be smaller, but at the same time it will give a smaller contribution to the correlation used in the later statistical analysis part of the attack, for the same length.

After finishing this step, we have a stored set \mathcal{S} of precomputed messages/error vectors with some special property for the \mathbf{e}_1 part.

3.2 Attack Step 2 - Submit Ciphertexts for Decryption

We now map the messages in \mathcal{S} to ciphertexts and give them to the decryption algorithm for each public key. We record the decryption error rate and keep track of the set of error vectors creating a decryption error, denoted \mathcal{S}' . We will attack and recover the secret key for keys⁵ where the decryption error rate is large.

As the enabling property for \mathbf{s} to have a large decryption error rate, we assume the property

$$\left| \sum_{i=0}^{n-1} s_i \right| \geq \delta_0,$$

⁴ We choose $l_1 \in \{65, 85\}$ to balance the complexity of precomputation and simulations. One can definitely choose a smaller l_1 to achieve a lower precomputation complexity at the cost of increasing the attack effort.

⁵ In the real case, one can start with the key with the largest decryption error rate, and then try different keys with error rates in the decreasing order.

for δ_0 a positive integer. One can approximate $\sum_{i=0}^{n-1} s_i$ by a Gaussian distribution with mean 0 and variance $n/2$. With this approximation, if we set $\delta_0 = 208$ as an example, the secret \mathbf{s} will have this property with probability about 2^{-64} .

We now need to examine the decryption error probability for such a condition on \mathbf{s} . The error term in LAC is of the form

$$\mathbf{W} = \mathbf{e}_1 \mathbf{s} - \mathbf{er} + \mathbf{e}_2.$$

The decryption error occurs if among all the coefficients of \mathbf{W} , at least 56 of them are with absolute value larger than $\lfloor q/4 \rfloor = 62$. In polynomial form, the error $w(x)$ is computed as

$$w(x) = e_1(x)s(x) - e(x)r(x) + e_2(x).$$

We only target the $e_1(x)s(x)$ term and consider the remaining as additional contributing noise in each position, denoted $\hat{N}(x)$, for the moment. For simplicity, we assume that all error vectors are of *TYPE 1* and have the assumed consecutive ones in their first positions, i.e., $e_1(x)$ is of the form $(e_0, e_1, \dots, e_{n-1}) = (1, 1, \dots, 1, e_{l_1}, \dots, e_{n-1})$. In vector form, the multiplication $e_1(x)s(x)$ can be written as

$$(s_0, s_1, \dots, s_{n-1}) \cdot \begin{bmatrix} e_0 & e_1 & e_2 & \dots & e_{n-1} \\ -e_{n-1} & e_0 & e_1 & \dots & e_{n-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -e_1 & -e_2 & -e_3 & \dots & e_0 \end{bmatrix}.$$

Since we assume $(e_0, e_1, \dots, e_{l_1-1}) = (1, 1, \dots, 1)$, the above is written

$$(s_0, s_1, \dots, s_{n-1}) \cdot \begin{bmatrix} 1 & 1 & \dots & 1 & e_{l_1} & e_{l_1+1} & \dots & e_{n-1} \\ -e_{n-1} & 1 & 1 & \dots & 1 & e_{l_1} & \dots & e_{n-2} \\ -e_{n-2} & -e_{n-1} & 1 & \dots & 1 & 1 & \dots & e_{n-3} \\ \vdots & \vdots & \ddots & \ddots & \vdots & \vdots & \ddots & \vdots \\ -e_{l_1} & -e_{l_1+1} & \dots & -e_{n-1} & 1 & 1 & \dots & 1 \\ -1 & -e_{l_1} & -e_{l_1+1} & \dots & -e_{n-1} & 1 & \dots & 1 \\ \vdots & \ddots & \ddots & \ddots & \dots & \ddots & \ddots & \vdots \\ -1 & \dots & -1 & -e_{l_1} & -e_{l_1+1} & \dots & -e_{n-1} & 1 \end{bmatrix}.$$

We model the $\hat{N}(x) = -\mathbf{er} + \mathbf{e}_2$ part of the noise as a sum of randomly generated variables (as also done in the LAC submission). Instead, we focus on the $\mathbf{e}_1 \mathbf{s}$ part, where we now have both \mathbf{e}_1 and \mathbf{s} of special forms. We see that for a particular key \mathbf{s} , the contribution from the fixed part $(e_0, e_1, \dots, e_{l_1-1}) = (1, 1, \dots, 1)$ to the multiplication $\mathbf{e}_1 \mathbf{s}$ is a vector defined as follows.

Definition 1 (Contribution vector). *The contribution vector $\mathbf{cv}(\mathbf{s})$ of $(e_0, e_1, \dots, e_{l_1-1}) = (1, 1, \dots, 1)$ for a secret key \mathbf{s} is defined as*

$$(cv_0, cv_1, \dots, cv_{n-1}),$$

where

$$cv_i = \begin{cases} \sum_{k=0}^i s_{i-k} - \sum_{k=n-l_1+1+i}^{n-1} s_k, & \text{if } 0 \leq i < l_1 - 1, \\ \sum_{k=0}^{l_1-1} s_{i-k}, & \text{if } l_1 - 1 \leq i \leq n - 1. \end{cases} \quad (2)$$

The basic idea in the attack is that the contribution vector is a fixed contribution that is the same for all \mathbf{e}_1 vectors of *TYPE 1*. Furthermore, assuming the secret vector \mathbf{s} of the form $|\sum_{i=0}^{n-1} s_i| \geq \delta_0$, it is easily verified that most coefficients in the contribution vector $\mathbf{cv}(\mathbf{s})$ are quite large. With a large fixed contribution in most coefficients, the probability of having a decryption error will drastically increase.

It seems difficult to derive an accurate estimation on the error probability due to the dependence between different positions in the error. One may try to use experiments to determine the variance and have a Gaussian approximation. But this distribution could be key-dependent and therefore somewhat unhelpful in a general sense.

Example 1: We have two choices for l_1 in implementation, i.e., $l_1 = 85$ or 65 . In the first case, if we collect errors with 85 consecutive 1's or -1 's, then this event happens with probability about

$$2^{-85 \times 2} \times n \times \frac{3}{2} \geq 2^{-160}.$$

In the latter case, the probability is about 2^{-120} . To collect $|\mathcal{S}|$ messages/error vectors, we need $2^{160}|\mathcal{S}|$ (or $2^{120}|\mathcal{S}|$) precomputation work. If we bound the number of decryption oracle calls by 2^{64} , as suggested by NIST, then the overall precomputation complexity is bounded by 2^{224} (or 2^{184}).

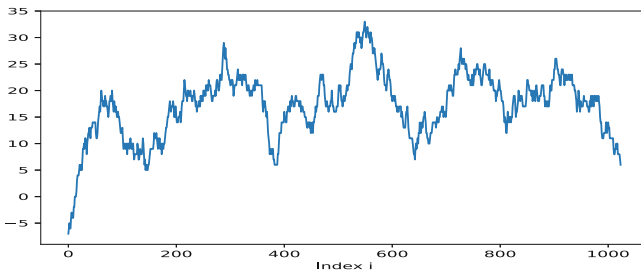


Fig. 1. The contribution vector $\mathbf{cv}(\mathbf{s})$ for a key \mathbf{s} with $\delta_0 = 208$ when $l_1 = 85$.

Assume we target a secret key \mathbf{s} having $\delta_0 = 208$ more 1 (-1) than -1 (1). For a randomly chosen secret key \mathbf{s} of this type we have plotted the contribution vector in Fig. 1. In Fig. 2 the corresponding histogram for $\mathbf{cv}(\mathbf{s})$ is given. We see that many coefficients in the contribution vector are quite large. Thus, the

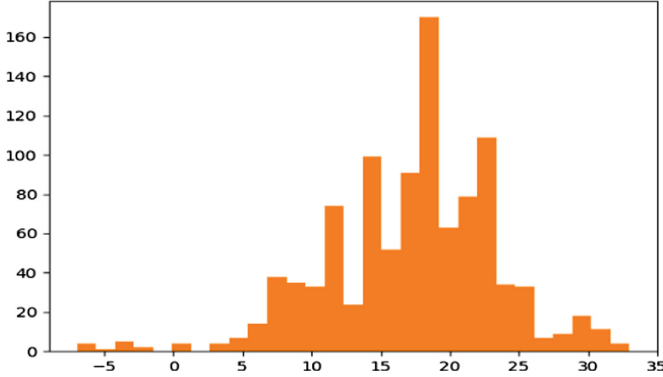


Fig. 2. The histogram of the contribution vector depicted in Fig. 1.

overall probability of having more than 56 coefficients with large absolute value can be much larger than that in the official analysis of decryption errors.

The error probability is difficult to predict and requires further investigation. In simulation, as shown in Table 2, we obtained decryption error probabilities of $2^{-6.6}$ for $l_1 = 85$ and $2^{-12.2}$ for $l_1 = 65$. This should be compared to the general decryption error probability of 2^{-115} !

Table 2. Decryption error probability P_e for a key \mathbf{s} with $\delta_0 = 208$.

| l_1 | P_e |
|-------|-------------|
| 85 | $2^{-6.6}$ |
| 65 | $2^{-12.2}$ |

To conclude this part of the attack, we submit a limited number of ciphertexts of *TYPE 1* (say with $l_1 = 65$) for decryption (say 2^{15}) and if we detect several errors (say around $2^{15} \cdot 2^{-12.2}$) we assume that δ_0 is large. We then get many more decryption failures in \mathcal{S}' for this weak key and move to the statistical analysis part. If few errors are detected, we move on to another public key.

3.3 Attack Step 3 - Statistical Analysis

This step assumes that we have identified a weak public key in the previous step. After receiving the errors, caused by the vectors in \mathcal{S}' , we need to reconstruct the secret key \mathbf{s} . Since the reconstruction step is the most difficult task for attacking LAC, we write it in the next section for fully describing the details.

Last, suppose that we have a guessed secret vector denoted by $(\mathbf{s}', \mathbf{e}')$. If $(\Delta \mathbf{s}, \Delta \mathbf{e}) = (\mathbf{s}, \mathbf{e}) - (\mathbf{s}', \mathbf{e}')$ is small, we can recover it using lattice reduction efficiently. Thus, we obtain the correct value of (\mathbf{s}, \mathbf{e}) . To be more specific, we

want to recover (\mathbf{s}, \mathbf{e}) from the public key $(\mathbf{a}, \mathbf{b} = \mathbf{a}\mathbf{s} + \mathbf{e})$. Let $\mathbf{b}' = \mathbf{a}\mathbf{s}' + \mathbf{e}'$. We have that $\mathbf{b} - \mathbf{b}' = \mathbf{a}\Delta\mathbf{s} + \Delta\mathbf{e}$. This is a new lattice problem with same dimension but smaller noise, which can be solved with less computational effort. In conclusion, we can handle with some small errors from the statistical testing.

4 Statistical Analysis

Assume that we have determined a weak key. We now collect the vectors $(\mathbf{e}_1, \mathbf{r})$ in \mathcal{S}' that caused decryption errors and average all the collected vectors. Our observation is that the parts $\mathbf{e}_1\mathbf{s}$, $-\mathbf{er}$, and \mathbf{e}_2 are all highly correlated with the contribution vector. The intuition behind it is that for a larger absolute value in the contribution vector, the probability of the corresponding position in \mathbf{W} exceeding 62 is larger, thereby implying that the values of $\mathbf{e}_1\mathbf{s}$, $-\mathbf{er}$, and \mathbf{e}_2 should be all larger.

We next derive two different approaches for the statistical analysis. The first one is a theoretical approach that is easy to analyze, where we recover the secret \mathbf{s} by observing the correlation between \mathbf{e}_2 and the contribution vector. The second one is a heuristic approach exploiting the fact that $-\mathbf{er}$ has a positive contribution on almost all the coefficients. We then try to recover the \mathbf{e} vector. This heuristic approach shows stronger correlation in implementation.

4.1 Theoretical Arguments for Statistical Recovery of the Contribution Vector

This subsection contains a recovery procedure which uses more theoretical arguments. We first note that if we recover the contribution vector (or something close to the contribution vector) then we can also almost trivially recover the secret key. The procedure to be given uses the dependence between $\mathbf{cv}(\mathbf{s})$ and the given \mathbf{e}_2 vector. We know from before that the probability of error in a particular position i depends on the value of the contribution vector in this position, which is here simply denoted cv_i . The good thing for analysis is that \mathbf{e}_2 is independent of the other parts involved in the error \mathbf{W} . Now denote the value of \mathbf{e}_2 in position i as E_i for simplicity. The observation we will examine is that the larger the value of cv_i is, the more likely it is that $E_i = 1$.

We denote the event of decryption error by \mathcal{D} , meaning no less than 56 positions are in error. We know that the probability for an error in position i in the set of vectors causing decryption failure is $P(cv_i + N_i + E_i > 62|\mathcal{D})$, where N_i denotes the non-fixed part of \mathbf{W} excluding E_i , that can be numerically computed via the convolution of probability distributions.

Now let us examine $P(E_i = 1|\mathcal{D})$ through

$$\begin{aligned} P(E_i = 1|\mathcal{D}) &= P(\text{error in pos. } i|\mathcal{D})P(E_i = 1|\text{error in pos. } i, \mathcal{D}) \\ &\quad + P(\text{no error in pos. } i|\mathcal{D})P(E_i = 1|\text{no error in pos. } i, \mathcal{D}). \end{aligned}$$

We assume that $P(E_i = 1|\text{no error in pos. } i, \mathcal{D}) \approx P(E_i = 1|\text{no error in pos. } i)$. Also, $P(E_i = 1|\text{error in pos. } i, \mathcal{D}) \approx P(E_i = 1|\text{error in pos. } i)$. Then we can rewrite as

$$P(E_i = 1|\mathcal{D}) = P(\text{error in pos. } i|\mathcal{D})P(E_i = 1|\text{error in pos. } i) + P(\text{no error in pos. } i|\mathcal{D}) \cdot P(E_i = 1|\text{no error in pos. } i). \quad (3)$$

Finally, we note that $P(E_i = x|\text{error in pos. } i) = P(\text{error in pos. } i|E_i = x) \cdot P(E_i = x)/P(\text{error in pos. } i)$, and compute $P(E_i = 1|\text{error in pos. } i)$ by

$$\frac{P(N_i > 61 - cv_i)}{P(N_i > 61 - cv_i) + 2P(N_i > 62 - cv_i) + P(N_i > 63 - cv_i)}. \quad (4)$$

Similarly, $P(E_i = x|\text{no error in pos. } i) = P(\text{no error in pos. } i|E_i = x) \cdot P(E_i = x)/P(\text{no error in pos. } i)$, and we compute $P(E_i = 1|\text{no error in pos. } i)$ by

$$\frac{P(N_i \leq 61 - cv_i)}{P(N_i \leq 61 - cv_i) + 2P(N_i \leq 62 - cv_i) + P(N_i \leq 63 - cv_i)}.$$

We get

$$P(E_i = 1|\mathcal{D}) = P(\text{error in pos. } i|\mathcal{D})P(E_i = 1|\text{error in pos. } i) + (1 - P(\text{error in pos. } i|\mathcal{D})) \cdot P(E_i = 1|\text{no error in pos. } i).$$

The probability $P(\text{error in pos. } i|\mathcal{D})$ is difficult to derive analytically, as one has to consider all combinations of error patterns with ≥ 56 errors. However, it can be determined from simulation results quite efficiently, since its dependence on cv_i is strong. Figure 3 plots this correlation and shows that a bigger $|cv_i|$ leads to a larger $P(\text{error in pos. } i|\mathcal{D})$ in almost all the cases.

Let us now examine the difference between $P(E_i = 1|\mathcal{D})$ for two different positions where the cv_i values are close, say 10 and 11. Examining $P(\text{error in pos. } i|\mathcal{D})$ for $cv_i = 10$ in simulation gives $P(\text{error in pos. } i|\mathcal{D}) \approx 0.023587$ and the same for $cv_i = 11$ gives $P(\text{error in pos. } i|\mathcal{D}) \approx 0.027138$.

With these values, one can compute the absolute difference ϵ of $P(E_i = 1|\mathcal{D})$ for $cv_i = 10$ and $cv_i = 11$, respectively. It would then require no more than $4/\epsilon^2$ decryption failures to distinguish between different cv_i values counting only the frequency of $E_i = 1$, with high probability. For larger cv_i values the difference between probabilities for consecutive cv_i values is increasing, so almost all entries of $\mathbf{cv}(\mathbf{s})$ can be determined through the frequency of $E_i = 1$.

Note that we need to determine the cv_i values for multiple positions, so we conservatively choose the following formula to estimate the data complexity,

$$\frac{8 \ln(n_t)}{\epsilon^2}, \quad (5)$$

where n_t is the number of tests bounded by n . Setting $n_t = n$, we numerically compute that it requires about $2^{34.0}$ errors to distinguish all the cv_i values larger than 11 with probability close to 1, if $l_1 = 85$.

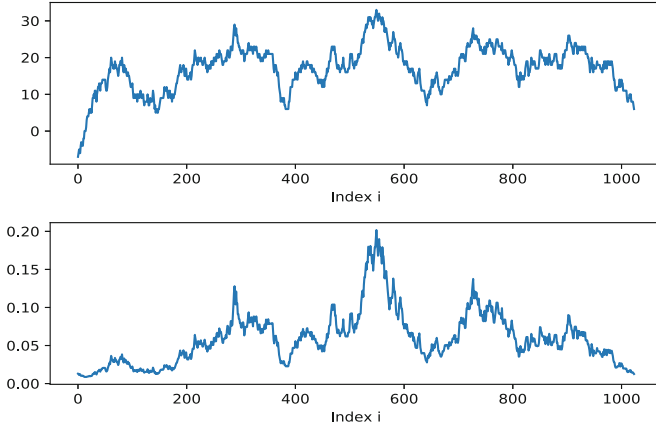


Fig. 3. The correlation of cv_i v.s. $P(\text{error in pos. } i | \mathcal{D})$. 700,000 *TYPE 1* errors are collected and $l_1 = 85$. The x axis represents the index i , for $0 \leq i < n$.

Now assume that we have recovered about 870 cv_i values⁶ for all $cv_i \geq 11$. We can then trivially recover the secret by using the inherent algebraic equations in the definition of cv_i , i.e., Eq. (2). We first notice that if two consecutive values of cv_i are with an absolute difference 2, then two positions in the secret \mathbf{s} are known. Based on these known positions, we then iteratively recover more positions in \mathbf{s} using the differences of known consecutive values of cv_i . We last fully recover the secret using a small number of guesses or other post-processing procedures like lattice reduction algorithms.

This recovery approach works well in our simulation⁷. In the simulation, we directly recover 594 positions using the algebraic structures discussed before. We then use the obtained 877 equations corresponding to the 877 positions with cv_i value no less than 11, and write them into a mixed integer linear programming model to maximize the value of $\left| \sum_{i=0}^{n-1} s_i \right|$. After running the optimization procedure for around 100 seconds using a desktop with an Intel(R) Core(TM) i7-7700 CPU, we successfully recover 995 positions among the 1024 unknown entries of \mathbf{s} . After guessing a few positions, it is easy to recover the key as most of the remaining errors are located in the first 100 positions.

For $l_1 = 65$, we similarly compute that it requires about $2^{29.8}$ errors to distinguish all the cv_i values larger than 9 with probability close to 1. We can then do a full key recovery similar to the approach discussed above.

⁶ For a key we chose in simulation, 877 positions are with a cv_i value no less than 11.

⁷ In simulation, we assume that cv_i is totally unknown for all $cv_i \leq 10$, which is pessimistic for an attacker. Actually, when the required decryption errors are obtained, we can have good knowledge of the values cv_i even if $cv_i \leq 10$.

Table 3. Success probability P_s of estimating $\mathbf{cv}(\mathbf{s})$ for a key \mathbf{s} with $\delta_0 = 208$.

| Number of <i>TYPE 1</i> errors | P_s |
|--------------------------------|-------|
| $2^{28.0}$ | 0.47 |
| $2^{29.5}$ | 0.58 |
| $2^{30.3}$ | 0.65 |

Experimental Verification. We have launched extensive simulation (of about 40,000 CPU core hours) to obtain $2^{30.3}$ *TYPE 1* errors when setting $l_1 = 85$. Firstly, we verify that the correlation between the probability $P(E_i = 1|\mathcal{D})$ and the value cv_i is very strong (see Fig. 4). Secondly, the experimental results match our theoretical prediction well. For instance, as shown in Table 3, one can recover 47% of the values cv_i with $2^{28.0}$ *TYPE 1* errors. The ratio increased to 65% (or 58%) when using $2^{30.3}$ (or $2^{29.5}$) *TYPE 1* errors.

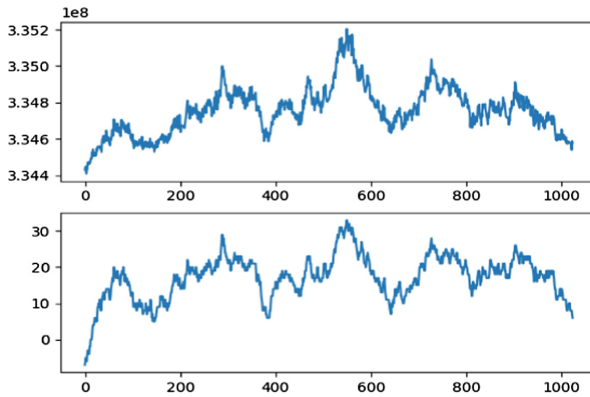


Fig. 4. The correlation of the frequency of $(E_i = 1|\mathcal{D})$ v.s. cv_i . $2^{30.3}$ *TYPE 1* errors are collected and $l_1 = 85$. The x axis represents the index i , for $0 \leq i < n$.

4.2 A Heuristic Approach

We have presented a simple key-recovery approach with theoretical arguments and also experimental validation. We next propose an alternative heuristic method, to demonstrate that better ways for key-recovery can probably be found.

We now look at the averaged values of $-\mathbf{er}$ for all the vectors in \mathcal{S}' causing errors. The strong correlation between this averaged vector and the contribution vector is shown in Fig. 5. Considering *TYPE 1* errors and $l_1 = 85$, we see that the correlation is much stronger if ten times more (i.e., 300000 v.s. 30000) decryption errors are provided, and the correlation is already very strong for 300000 error samples. Comparing Figs. 5 and 6, we also see that with a similar number of decryption failures the correlation between the contribution vector and the averaged vector is stronger if the error probability is smaller.

The remaining problem is to have an accurate estimation \mathbf{e}' of \mathbf{e} .

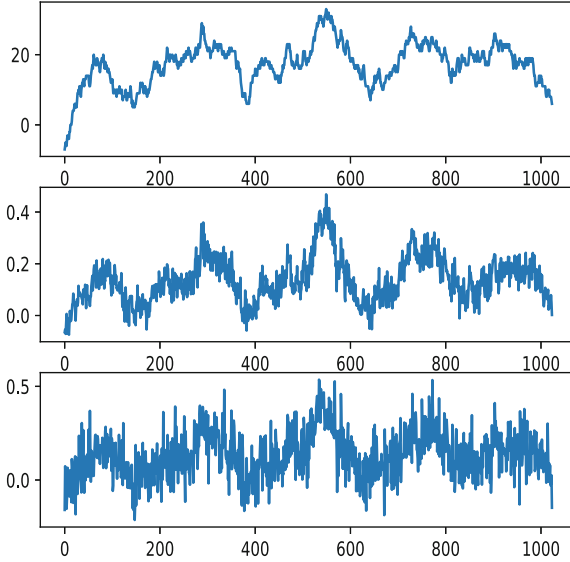


Fig. 5. The correlation of cv_i v.s. the averaged values of $-\mathbf{er}$ w.r.t. different positions for a key. The first subfigure plots the contribution vector and 300000 (30000) *TYPE 1* errors are collected in the second (third) subfigure. We set $l_1 = 85$. The x axis represents the index i , for $0 \leq i < n$.

One approach is to guess a small subvector \mathbf{e}_{sec} (say of length $l_s = 12$) of the \mathbf{e} vector and to check the decryption error probability when the occurrences of $-\mathbf{e}_{\text{sec}}$ (or near-collisions that are defined as a very close pattern) in part of the ciphertexts in \mathcal{S} are larger than a threshold th_0 . This probability should be higher for the correct guess, and vice versa for the wrong guesses, among all the guesses with the same numbers of ‘-1’, ‘1’, and ‘0’.

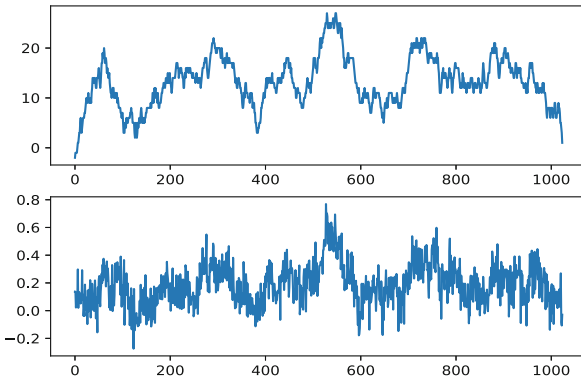


Fig. 6. The correlation of cv_i v.s. the averaged values of $-\mathbf{er}$ w.r.t. different positions for a key. 36690 *TYPE 1* errors are collected and $l_1 = 65$. The x axis represents the index i , for $0 \leq i < n$.

This idea is similar to that of colliding ‘ones’ from the key and the collected ciphertexts proposed in [14]. The intuition behind it is that it is more probable to show multiple (near-)collisions for a right guess because in a decryption error from \mathcal{S}' , more than 56 positions are of a large positive value.

Let us for instance guess the first $l_s = 12$ entries \mathbf{e}_{sec} of \mathbf{e} . We then record the number of the ciphertexts in \mathcal{S}' that more than th_0 (near-)collisions to its negative $-\mathbf{e}_{\text{sec}}$ in each chosen ciphertext \mathbf{r} with index from $l_1 - l_s$ to n are found. We next need to know the number for the ciphertexts in \mathcal{S} that the same condition holds and compute the likelihood by the ratio of the two numbers. In simulation, we can instead sample at random a large number (say 10,000,000) of ciphertexts in \mathcal{S} since the size of \mathcal{S} could be too large. We then rank all the guesses by these likelihood values. Note that the index interval should be adjusted if the guessed consecutive entries start from a different position. We select this index interval $[l_1 - l_s, n]$ since the starting entries of the contribution vector could be (with a high probability) negative.

We now have a list of subkeys near a certain position sorted according to their key-ranks. We then shift to a nearby starting position and produce a new list of subkeys with respect to the new position. Since the guessed subkeys with respect to nearby positions could have large overlaps and the correlation becomes stronger if the guessed length is longer, we can combine different subkeys to generate a list of longer subkeys. We iteratively combine the subkeys to have an estimation on the full \mathbf{e} vector.

If we accurately recovered the \mathbf{e} vector, then the secret \mathbf{s} can be solved from the public key ($\text{seed}_{\mathbf{a}}, \mathbf{b}$). Even if some errors occur in the statistical test step, they can be corrected by a further lattice reduction step.

Implementation Results. We present the implementation results with the test interval $[l_1 - l_s, n]$, where $l_s = 12$ and $l_1 = 65$. Thus, the starting positions of the guessed keys with a high rank should be close to the initial position. We checked all possible key patterns with length 12, five nonzero positions and nonzero starting and end entries. The threshold th_0 is set to be 18 and a near-collision is found if in a ciphertext a pattern of length 12 whose inner product with the guessed pattern is smaller than -4 occurs.

We use the collected 36690 decryption errors and test the ranks of 5 subvectors in \mathbf{e} of the select form and also of a starting position close to the initial position. From Table 4 we see, among the 5 tested ‘true’ keys, that four keys are with a relatively high rank. For example, the first row in the table states that a key with two minus ones and three ones is ranked fourth among all the 1200 possible key patterns.

Table 4. Key ranks of the ‘true’ keys.

| Type | Rank |
|--------|----------|
| [2, 3] | 4/1200 |
| [2, 3] | 109/1200 |
| [1, 4] | 34/600 |
| [1, 4] | 87/600 |

We notice that most of the guesses with a high rank are a subvector of \mathbf{e} with the starting position (left or right) shifted⁸ for less than 30 positions. Thus, we can reconstruct a longer (than length 12) subkey near the initial position with high confidence. The further combination of the subkey patterns is straightforward but requires much more implementation effort.

The correlation would be much stronger if more decryption failures are collected, as shown in Fig. 5. We conjecture that increasing the used number of errors in implementation by a reasonable constant factor (say 10 or 20) would lead to a full attack, and then the required number of data in the heuristic approach could be much smaller than that of the theoretical approach.

4.3 The Complexity Analysis

As claimed in Sect. 4.1, when setting $l_1 = 65$, the secret \mathbf{s} can be fully recovered using no more than $2^{29.8}$ decryption failures in the theoretical approach. Thus, the precomputation cost can be estimated as $2^{120+29.8}/2^{-12.2} \approx 2^{162}$. The heuristic approach may reduce the precomputation cost further. For both approaches, one needs to submit 2^{15} ciphertexts to 2^{64} decryption algorithms (public keys) to determine the weak key, with complexity about 2^{79} . He also needs to perform the statistical test whose complexity is negligible. It is common to exclude the precomputation effort in the complexity analysis since the precomputation will be done only once; therefore, we claim an overall attacking complexity of 2^{79} .

4.4 Discussions

In this part, we discuss possible extension of the new attack.

Increase the Weak-Key Probability. In implementation, weak keys with probability of 2^{-64} are targeted. Based on the simulation results, we in Table 5 show the precomputation cost for generating a ciphertext regarding weak keys of various probability p . If the *TYPE 2* errors are chosen, then one can achieve an error rate of about 2^{-12} for weak keys with a fairly large probability, say 2^{-16} , having the precomputation effort below the claimed 256-bit (classical) security

⁸ This shift operation can be viewed as being multiplied by x^j for $j \in \mathbb{Z}$ over the ring $\frac{\mathbb{Z}_q[x]}{(x^n+1)}$.

level of LAC256. These keys can be risky, under the assumption that a similar level of key information (correlation) can be extracted for a fixed number of errors with similar error rates.

Table 5. Precomputation cost of generating a ciphertext for random keys with probability p to achieve an error rate of about 2^{-12} .

| p | $\log_2(C)$ | l_1 | l_0 | δ_0 |
|-----------|------------------|-------|-------|------------|
| 2^{-64} | 120 [†] | 65 | 0 | 208 |
| 2^{-32} | 140 | 145 | 135 | 143 |
| 2^{-16} | 220 | 225 | 215 | 98 |

[†]This complexity can be lower if $l_0 \neq 0$

5 Attacking the LAC Version in Round 2 of the NIST PQ-Project

The authors very recently revised the LAC proposal when it entered the round 2 of the NIST post-quantum standardization effort, see [20, 21]. The introduced modifications are a few:

- They employed a new coding scheme where the message is firstly encoded with BCH codes and the codeword is further encoded with the D2 error correcting codes. The BCH code now corrects much less number of errors (16 errors can be corrected), but the addition of the D2 encoder makes the overall error probability roughly the same as before. Also, the message/key size is decreased to 256 bits.
- The noise distribution is changed to have a constant weight and the same number of ones as minus ones. For instance, the vectors of $\mathbf{s}, \mathbf{e}, \mathbf{e}_1, \mathbf{r}$ are now sampled from $\Psi_1^{n,h}$, containing exactly $h/2$ ones and $h/2$ minus ones, and the distribution of \mathbf{e}_2 is unchanged. The authors⁹ made this change to resist against the high Hamming weight CCA attacks [1, 8], in which higher decryption error rates are achieved via choosing high Hamming weight messages/errors through precomputation.
- Some 4 bits of each position in the second part of the ciphertext are dropped, bringing some ciphertext compression. This however adds an additional noise term uniformly distributed in $[-7, 7]$.

Table 6 shows the concrete parameters of LAC256-v2 proposed in the NIST round 2 submission. In LAC256-v2, l_v is equal to 800; thus only the first 800 positions matter, and the D2 encoding/decoding combines two positions of distance 400 apart¹⁰. The decryption error probability is estimated to be 2^{-122} .

⁹ In the round 2 submission [20], they also argued that ‘it is difficult to get any information about the private key from these decryption failures’.

¹⁰ This part is verified from the reference implementation.

Table 6. Proposed parameters of LAC256-v2 in round 2.

| n | q | \mathcal{R} | h | l_v | Distribution | ecc | DER | Security |
|------|-----|---|-----|-------|------------------------|------------------------|------------|----------|
| 1024 | 251 | $\frac{\mathbb{Z}_q[x]}{\langle x^n+1 \rangle}$ | 512 | 800 | $\Psi_1, \Psi_1^{n,h}$ | BCH[511, 256, 33] + D2 | 2^{-122} | V |

We shortly present the steps of generalizing the previous attack to the new version. The basic idea is that we split the enabling error patterns in two parts, one part being the even positions having a positive contribution and the second part being the odd positions having a negative contribution. The same goes for the desired pattern in the secret vector \mathbf{s} . We expect the contribution of the two parts to have the same sign and the sum to be roughly doubled.

5.1 Attack Step 1 - Precomputation

We again construct a special set \mathcal{S} of messages/error vectors by precomputation. We are only interested in keeping messages that give rise to noise vectors of special form of the \mathbf{e}_1 vector. Let us include combinations where the error vector \mathbf{e}_1 contains an interval of consecutive l_1 entries, where every even position in the interval contains a 1 and every odd position contains a -1 (or vice versa). Thus, l_1 is even. The starting position of this interval is chosen from $[801, n - l_1]$, leading to the largest contribution to the first $l_v = 800$ positions¹¹ of $\mathbf{e}_1\mathbf{s}$.

Assuming a randomly selected error vector from the error distribution $\Psi_1^{n,h}$, the probability of finding such an interval starting from a fixed position in $[801, n - l_1]$ is then

$$p = 2 \times \frac{\binom{h/2}{l_1/2} \cdot \binom{h/2}{l_1/2}}{\binom{n}{l_1} \cdot \binom{l_1}{l_1/2}}.$$

The overall probability can be approximated by

$$p_0 \approx (n - 800 - l_1)p. \quad (6)$$

We denote the type of noise vectors of the above kind as *TYPE 1b* noise vectors.

Just as before, we can consider many other special forms of the noise that can be useful in an attack, for example noise vectors similar to what we previously defined as *TYPE 2* noise vectors but split in even and odd parts. We also define *TYPE 2b* errors that the error vector \mathbf{e}_1 contains an interval of consecutive $l_1 + l_0$ entries, where all the even positions include $l_1/2$ ones and $l_0/2$ zeros, and all the odd positions include $l_1/2$ minus-ones and $l_0/2$ zeros (or vice versa). The starting position of this interval is chosen from $[801, n - l_1 - l_0]$. With the same precomputation effort, this error pattern can lead to a much larger decryption error probability compared with *TYPE 1b* errors.

¹¹ Due to the mod $(x^n + 1)$ operation, the controlled interval is split into two consecutive parts in $(l_1 - 1)$ columns of the matrix generated by shifting \mathbf{e}_1 . These two parts will be multiplied by 1 or -1 , respectively, leading to a reduced absolute contribution.

After finishing this step, we assume that we have a stored set \mathcal{S} of pre-computed messages/error vectors with the *TYPE 1b* (or *TYPE 2b*) property for the \mathbf{e}_1 part. We could describe a *TYPE 1b* error as a *TYPE 2b error* with $l_0 = 0$.

5.2 Attack Step 2 - Submit Ciphertexts for Decryption

Similar to the procedure in Sect. 3.2, we map the messages in \mathcal{S} to ciphertexts and submit them to the decryption algorithm for each public key. We keep track of the set of error vectors causing a decryption error, denoted \mathcal{S}' , and record the decryption error rate. We then attempt to recover the public key for keys where the decryption error rate is large i.e., targeting users with error rates in a decreasing order.

The weak key property for \mathbf{s} that gives a large decryption error rate, is the property

$$\left| \sum_{i=0}^{n/2-1} s_{2i} \right| + \left| \sum_{i=0}^{n/2-1} s_{2i+1} \right| \geq \delta_0.$$

For a noise distribution with fixed $n/4$ ones and $n/4$ minus ones, it is easy to compute the probability for the above condition. For example, if we set $\delta_0 = 206$, the secret \mathbf{s} will have the above property with probability about 2^{-64} . Since $\sum_{i=0}^n s_i = 0$, we further have

$$\left| \sum_{i=0}^{n/2-1} s_{2i} \right| = \left| \sum_{i=0}^{n/2-1} s_{2i+1} \right| = \frac{\delta_0}{2} = 103.$$

We now examine the decryption error probability for such a weak secret \mathbf{s} . The error term in the new LAC is of the form

$$\mathbf{W} = \mathbf{e}_1 \mathbf{s} - \mathbf{e} \mathbf{r} + \mathbf{e}_2 + \mathbf{e}_3,$$

where \mathbf{e}_3 is a new error term due to the ciphertext compression.

We now get a decryption error occurring if among all 400 values after the D2 encoding/decoding of \mathbf{W} , at least 17 of them are in error.

The D2 encoding/decoding means that the same binary value (message/key bit) is encoded in two different code positions and the decoding means adding the two positions together and checking if the sum is around 0 or around q . For an error to occur, the sum of the noise in the two positions must have an absolute value larger than $\lfloor q/2 \rfloor = 125$.

In polynomial form, the error $w(x)$ is computed as

$$w(x) = e_1(x)s(x) - e(x)r(x) + e_2(x) + e_3(x).$$

We focus on $e_1(x)s(x)$ and consider the remaining as noise.

For simplicity, we assume that all error vectors are of *TYPE 1b* and have the assumed sequence of even ones and odd minus ones in their last positions,

i.e., $e_1(x)$ is of the form $(e_0, e_1, \dots, e_{n-1}) = (e_0, \dots, e_{n-l_1-1}, 1, -1, \dots, 1, -1)$. We examine the $\mathbf{e}_1\mathbf{s}$ part, where we again have both \mathbf{e}_1 and \mathbf{s} of special forms. For a particular key \mathbf{s} , the contribution from the fixed part $(e_{n-l_1}, \dots, e_{n-1}) = (1, -1, \dots, 1, -1)$ to the multiplication $\mathbf{e}_1\mathbf{s}$ is the contribution vector now defined as follows.

Definition 2 (Contribution vector for TYPE 1b errors). *The contribution vector $\mathbf{cv}(\mathbf{s})$ of $(e_{n-l_1}, \dots, e_{n-1}) = (1, -1, \dots, 1, -1)$ for a secret key \mathbf{s} is defined as*

$$(cv_0, cv_1, \dots, cv_{n-1}),$$

where

$$cv_i = \begin{cases} \sum_{k=i+1, k \text{ odd}}^{i+l_1} s_k - \sum_{k=i+1, k \text{ even}}^{i+l_1} s_k, & \text{if } 0 \leq i < n-l_1, i \text{ even,} \\ \sum_{k=i+1, k \text{ even}}^{i+l_1} s_k - \sum_{k=i+1, k \text{ odd}}^{i+l_1} s_k, & \text{if } 0 \leq i < n-l_1, i \text{ odd,} \\ \sum_{k=i+1, k \text{ odd}}^{n-1} s_k - \sum_{k=i+1, k \text{ even}}^{n-1} s_k \\ -(\sum_{k=0, k \text{ odd}}^{i+l_1-n} s_k - \sum_{k=0, k \text{ even}}^{i+l_1-n} s_k), & \text{if } n-l_1 \leq i \leq n-1, i \text{ even,} \\ \sum_{k=i+1, k \text{ even}}^{n-1} s_k - \sum_{k=i+1, k \text{ odd}}^{n-1} s_k \\ -(\sum_{k=0, k \text{ even}}^{i+l_1-n} s_k - \sum_{k=0, k \text{ odd}}^{i+l_1-n} s_k), & \text{if } n-l_1 \leq i \leq n-1, i \text{ odd.} \end{cases} \quad (7)$$

The new idea in this attack is that the contribution vector is a fixed contribution as before for all \mathbf{e}_1 vectors of *TYPE 1b*, but now it will shift in sign depending on whether i is even or odd.

Assuming the secret vector \mathbf{s} of the form $|\sum_{i=0}^{n/2-1} s_{2i}| + |\sum_{i=0}^{n/2-1} s_{2i+1}| \geq \delta_0$, we can verify that most coefficients in the contribution vector $\mathbf{cv}(\mathbf{s})$ are quite large, but with shifting signs (see Fig. 7 for an instance). Again, with a large fixed contribution in most coefficients, the probability of a decryption error will be much larger than expected.

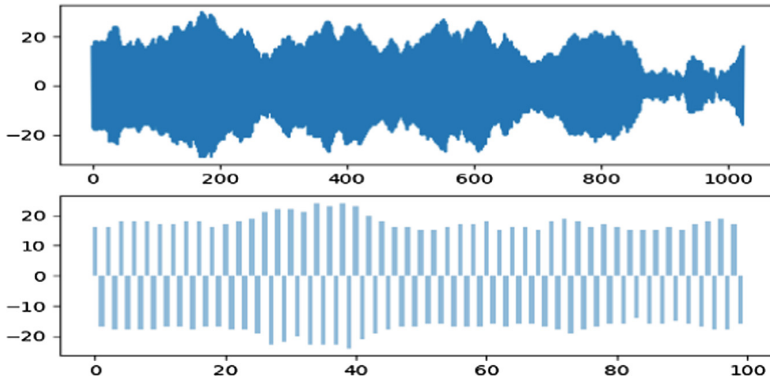


Fig. 7. The contribution vector $\mathbf{cv}(\mathbf{s})$ of a key \mathbf{s} with $\delta_0 = 206$ when $l_1 = 86$, for *TYPE 1b* errors. The first figure plots the whole vector and the second one plots its first 100 positions.

Table 7. Decryption error probability P_e for a key \mathbf{s} with $\delta_0 = 206$.

| $\log_2(C_{pre})$ | l_1 | l_0 | P_e |
|-------------------|-------|-------|--------------|
| 170 | 86 | 0 | $2^{-21.03}$ |
| 128 | 66 | 0 | $2^{-28.42}$ |
| 120 | 114 | 80 | $2^{-12.74}$ |

In LAC256-v2, the D2 encoding/decoding combines two positions of distance 400 apart, which means that the fixed contribution is of the same sign in both positions and we have a large fixed contribution when summing the two positions.

Example 2: For LAC256-v2 we tested three choices for l_1 and l_0 in implementation, i.e., for *TYPE 1b* errors with $l_1 = 86$ and 66 , and for *TYPE 2b* errors with $l_1 = 114$ and $l_0 = 80$. We targeted a secret key \mathbf{s} having $\delta_0 = 206$, and the key probability can be estimated as 2^{-64} . The simulated decryption error probabilities (no less than 17 positions in error) are shown in Table 7, where the first column describes the computational efforts (in $\log_2(\cdot)$) of finding one desired message/noise. We see that the decryption error probability can be higher than 2^{-13} with complexity of about 2^{120} for one message/noise.

We note that the error performance of the new LAC256-v2 in round 2 with respect to our attack is slightly better than that of the old version, but is still far from preventing this attack.

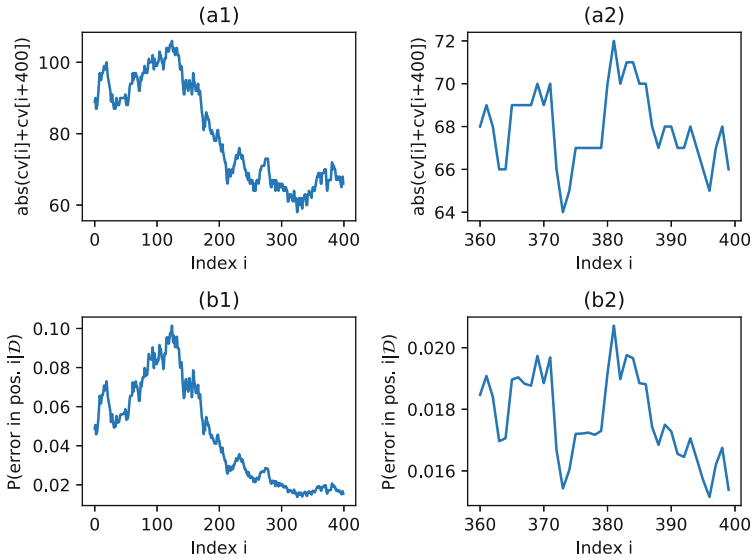


Fig. 8. The correlation of \mathbf{cv} w.r.t. the *TYPE 1b* errors with $l_1 = 194$ v.s. $P(\text{error in pos. } i | \mathcal{D})$. About 6,500,000 *TYPE 2b* errors are collected with $l_1 = 114$ and $l_0 = 80$.

5.3 Attack Step 3 - Recovering S

This final step assumes an identified weak public key in the previous step. After receiving the decryption errors, caused by the vectors in \mathcal{S}' , we reconstruct the secret key \mathbf{s} . The procedure is analogue to the procedure described in the previous section. Throughout the section, we focus on *TYPE 2b* errors as they can lead to a higher decryption error probability. Note that the distributions of the i -th and the $(i + 400)$ -th positions (of say \mathbf{e}_2) should be studied jointly due to the implementation of the D2 error correcting codes.

The interesting main observation is that for *TYPE 2b* errors with $l_1 = 114$ and $l_0 = 80$, the distribution of $P(E_i|\mathcal{D})$ is a function of the sum of the i -th and the $(i + 400)$ -th positions of the contribution vector for *TYPE 1b* errors with $l_1 = 194$, where E_i is the random variable representing the sum of the the i -th and the $(i + 400)$ -th positions of \mathbf{e}_2 .

We plot the simulation results for verifying this correlation in the first and the second subfigures of Fig. 9. Since about 2^{38} encryptions have been performed,

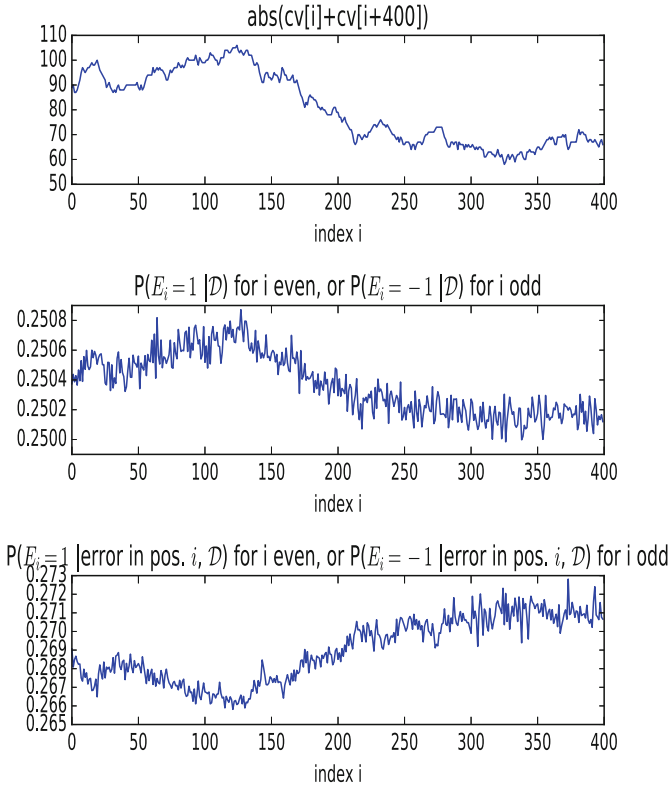


Fig. 9. The correlation of cv w.r.t. the *TYPE 1b* errors with $l_1 = 194$, $P(E_i|\mathcal{D})$ and $P(E_i|\text{error in pos. } i, \mathcal{D})$. About $2^{25.09}$ *TYPE 2b* errors are collected with $l_1 = 114$ and $l_0 = 80$.

it is beyond our computational capability to run an even larger experiment. However, the correlation between the sum of two positions in the contribution vector w.r.t. the *TYPE 1b* errors with $l_1 = 194$ and $P(\text{error in pos. } i | \mathcal{D})$ is much stronger and can be verified with fewer decryption errors. This strong correlation explains our main observation.

We show the latter strong correlation in Fig. 8 where the plots are obtained from about 6,500,000 *TYPE 2b* errors. The left two subfigures, (a1) and (a2), plot the correlation for the whole vector of length 400, and the right two, (b1) and (b2), plot the last 40 entries.

We can now use the theory from Sect. 4.1 to study the data complexity for this attack on LAC256-v2. We first obtain an equation with the same form as Eq. (3), and the difficulty is then to numerically compute $P(E_i = 1 | \text{error in pos. } i)$ (or $P(E_i = 1 | \text{no error in pos. } i)$). Since we include l_0 zeros in the controlled interval to form *TYPE 2b* errors, the contribution (denoted by a random variable CV) from the controlled intervals (of the i -th and $(i + 400)$ -th positions) is no longer a constant. We could approximate the distribution of CV by using a typical choice, i.e., assuming half of the positions of \mathbf{s} corresponding to the controlled intervals are 0, and then compute the distribution via convolution. Then, $P(E_i = 1 | \text{error in pos. } i)$ can be computed as

$$\sum_{cv} P(\text{CV} = cv) P(E_i = 1 | \text{error in pos. } i, \text{CV} = cv),$$

where similarly to Eq. (4), $P(E_i = 1 | \text{error in pos. } i, \text{CV} = cv)$ can be computed as $\frac{4P(N_i > 124 - cv)}{f(cv)}$, and $f(cv) = P(N_i > 123 - cv) + 4P(N_i > 124 - cv) + 6P(N_i > 125 - cv) + 4P(N_i > 126 - cv) + P(N_i > 127 - cv)$. Here N_i is defined, similar to that in Sect. 4.1, as the noise term after the D2 decoding excluding the CV part and the E_i part. We compute $P(E_i = 1 | \text{no error in pos. } i)$ similarly, and finally apply Eq. (5) to estimate the data complexity¹². With this analysis, the complexity to distinguish the sum of the i -th and the $(i + 400)$ -th positions of the contribution vector for *TYPE 1b* errors with $l_1 = 194$ to be 89 or 90 is $2^{34.73}$.

If we test $P(E_i | \text{error in pos. } i, \mathcal{D})$ empirically, as shown in the last subfigure of Fig. 9, we obtain a data complexity estimation of about $2^{33.42}$, to distinguish the sum to be 89 or 90. This empirical estimation verifies our theoretical estimation to which we will stick in our later analysis.

In simulation, we obtained 151 equations corresponding to the index set \mathcal{I}_{90} with size 151, where for $i \in \mathcal{I}_{90}$ the sum of the i -th and the $(i + 400)$ -th positions of the contribution vector for *TYPE 1b* errors with $l_1 = 194$ is no less than 90. We do more rounds to collect more linear equations using error patterns derived from *TYPE 2b* errors, i.e., replacing one position in the controlled interval by an carefully selected position outside the interval. We ask the two positions to be both even or odd and assign corresponding values of the same sign. Since only a single index of the controlled positions is changed, the contribution vector or the

¹² The analysis is conservative because we only consider one value of E_i conditioned on the collected errors. In fact, we can use the full distribution, i.e., the probabilities of $E_i | \mathcal{D}$ being 5 different possible values, to reduce the data complexity further.

error probability keeps (or has a small difference) but more linear equations of entries from \mathbf{s} are obtained. One then performs about $n/151 \approx 7$ times to collect enough linear equations for a full recovery. In summary, the required number of decryption errors is bounded by 2^{38} and the overall precomputation complexity is estimated to be 2^{171} .

6 Discussions

On Multi-target Protection. Many proposals to the NIST post-quantum standardization process have a multi-target protection technique, i.e., including the public key as an input to a hash function for generating the noise seeds. We see that for LAC256, the multi-target protection cannot thwart this attack because this attack can work in the single-key setting that is independent of the multi-target protection technique. Actually, this attack recovers the key with precomputation complexity 2^{162} and success probability 2^{-64} , when only a single key is assumed.

Using the adaptive attack model in [13], for LAC256 with the multi-target protection, one can prepare 2^{15} ciphertexts for 2^{64} users with complexity $2^{120+64+15} = 2^{199}$ if $l_1 = 65$, to determine the weak keys. The dominant part in the complexity analysis is a precomputation of 2^{199} , far below the 2^{256} bound. We can further reduce the complexity by balancing the computational efforts in different steps.

Similar analysis applies to LAC256-v2.

On Other LAC Parameter Settings. On LAC128-v2 having a key with probability 2^{-64} , with precomputation of 2^{58} for one chosen ciphertext, we simulated a decryption failure probability of around 2^{-28} . This experimental data demonstrate that LAC128-v2 could be vulnerable to the newly proposed attack in the multi-target setting. This multi-target attack can be prevented via including the multi-target protection technique discussed before. We believe that LAC192-v2 is more vulnerable to the multi-target attack than LAC128-v2 since one has a less restricted bound for precomputation.

7 Conclusions and Future Works

We have presented a CCA attack on the scheme LAC256, a proposal to the NIST Post-Quantum Cryptography Standardization project that has passed the first round selection. This attack exploits the decryption failures and recovers one weak key among about 2^{64} public keys. This attack has two versions due to the two different approaches to reconstruct the secret key from the decryption failures. With the first approach, we present an attack with complexity of 2^{79} , i.e., the cost of determining a weak key, if the required precomputation cost is bound by 2^{162} encryptions. One can definitely achieve different trade-offs between the

attacking complexity and the precomputation cost via choosing suitable parameters. The second approach is similar to the reaction attacks [10, 14, 15] on the MDPC and LDPC based cryptosystems, which could heuristically decrease the precomputation effort further. We also discuss the possibility of attacking a key with a much larger probability, say 2^{-16} . Last, we discuss the attack on a new version of LAC256 in round 2 of the NIST PQC project. We claim a multi-target attack with complexity bounded by 2^{80} and precomputation of about 2^{171} to recover a weak key among about 2^{64} public keys. This attack also means a CCA attack on LAC256-v2 with precomputation of 2^{171} , and success probability 2^{-64} , in the single-key setting.

This attack can also be applied to other schemes. It is interesting future work to check the performance of the attack with respect to different proposals in the NIST Post-Quantum Cryptography Standardization project.

Acknowledgments. The authors would like to thank Xianhui Lu and the anonymous reviewers from Asiacrypt 2019 for their helpful comments. This work was supported in part by the Swedish Research Council (Grant No. 2015-04528), by the Swedish Foundation for Strategic Research (Grant No. RIT17-0005), and by the Norwegian Research Council (Grant No. 247742/070). Jing Yang was also supported by the scholarship from the National Digital Switching System Engineering and Technological Research Center, China. The computations/simulations were performed on resources provided by UNINETT Sigma2 - the National Infrastructure for High Performance Computing and Data Storage in Norway, and by Swedish National Infrastructure for Computing.

References

1. NIST Post-Quantum Cryptography Forum. <https://groups.google.com/a/list.nist.gov/forum/#!forum/pqc-forum>. Accessed 11 Jan 2019
2. NIST Post-Quantum Cryptography Standardization. <https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Post-Quantum-Cryptography-Standardization>. Accessed 24 Sept 2018
3. Bernstein, D.J., Groot Bruinderink, L., Lange, T., Panny, L.: HILA5 pindakaas: on the CCA security of lattice-based encryption with error correction. In: Joux, A., Nitaj, A., Rachidi, T. (eds.) AFRICACRYPT 2018. LNCS, vol. 10831, pp. 203–216. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-89339-6_12
4. Boldyreva, A., Degabriele, J.P., Paterson, K.G., Stam, M.: On symmetric encryption with distinguishable decryption failures. In: Moriai, S. (ed.) FSE 2013. LNCS, vol. 8424, pp. 367–390. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-43933-3_19
5. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (Leveled) fully homomorphic encryption without bootstrapping. In: Goldwasser, S. (ed.) ITCS 2012: 3rd Innovations in Theoretical Computer Science, Cambridge, MA, USA, 8–10 January 2012, pp. 309–325. Association for Computing Machinery (2012)
6. Brakerski, Z., Langlois, A., Peikert, C., Regev, O., Stehlé, D.: Classical hardness of learning with errors. In: Boneh, D., Roughgarden, T., Feigenbaum, J. (eds.) 45th Annual ACM Symposium on Theory of Computing, Palo Alto, CA, USA, 1–4 June 2013, pp. 575–584. ACM Press (2013)

7. Brakerski, Z., Vaikuntanathan, V.: Efficient fully homomorphic encryption from (standard) LWE. In: Ostrovsky, R. (ed.) 52nd Annual Symposium on Foundations of Computer Science, Palm Springs, CA, USA, 22–25 October 2011, pp. 97–106. IEEE Computer Society Press (2011)
8. D’Anvers, J.-P., Guo, Q., Johansson, T., Nilsson, A., Vercauteren, F., Verbaauwhede, I.: Decryption failure attacks on IND-CCA secure lattice-based schemes. In: Lin, D., Sako, K. (eds.) PKC 2019. LNCS, vol. 11443, pp. 565–598. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17259-6_19
9. Ding, J., Alsayigh, S., Saraswathy, R.V., Fluhrer, S., Lin, X.: Leakage of signal function with reused keys in RLWE key exchange. Cryptology ePrint Archive, Report 2016/1176 (2016). <http://eprint.iacr.org/2016/1176>
10. Fabšič, T., Hromada, V., Stankovski, P., Zajac, P., Guo, Q., Johansson, T.: A reaction attack on the QC-LDPC McEliece cryptosystem. In: Lange, T., Takagi, T. (eds.) PQCrypto 2017. LNCS, vol. 10346, pp. 51–68. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59879-6_4
11. Fluhrer, S.: Cryptanalysis of ring-LWE based key exchange with key share reuse. Cryptology ePrint Archive, Report 2016/085 (2016). <http://eprint.iacr.org/2016/085>
12. Gama, N., Nguyen, P.Q.: New chosen-ciphertext attacks on NTRU. In: Okamoto, T., Wang, X. (eds.) PKC 2007. LNCS, vol. 4450, pp. 89–106. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-71677-8_7
13. Guo, Q., Johansson, T., Nilsson, A.: A generic attack on lattice-based schemes using decryption errors with application to ss-ntru-pke. IACR Cryptology ePrint Archive 2019, 43 (2019). <https://eprint.iacr.org/2019/043>
14. Guo, Q., Johansson, T., Stankovski, P.: A key recovery attack on MDPC with CCA security using decoding errors. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016, Part I. LNCS, vol. 10031, pp. 789–815. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53887-6_29
15. Guo, Q., Johansson, T., Wagner, P.S.: A key recovery reaction attack on QC-MDPC. IEEE Trans. Inf. Theory **65**(3), 1845–1861 (2019). <https://doi.org/10.1109/TIT.2018.2877458>
16. Hall, C., Goldberg, I., Schneier, B.: Reaction attacks against several public-key cryptosystem. In: Varadharajan, V., Mu, Y. (eds.) ICICS 1999. LNCS, vol. 1726, pp. 2–12. Springer, Heidelberg (1999). https://doi.org/10.1007/978-3-540-47942-0_2
17. Hoffstein, J., Silverman, J.H.: NTRU Cryptosystems Technical Report Report# 016, Version 1 Title: Protecting NTRU Against Chosen Ciphertext and Reaction Attacks (2000)
18. Howgrave-Graham, N., et al.: The impact of decryption failures on the security of NTRU encryption. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 226–246. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45146-4_14
19. Lu, X., Liu, Y., Jia, D., Xue, H., He, J., Zhang, Z.: LAC. Technical report, National Institute of Standards and Technology (2017). <https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions>
20. Lu, X., et al.: LAC. Technical report (2019). <https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Round-2-Submissions>
21. Lu, X., et al.: LAC: practical ring-LWE based public-key encryption with byte-level modulus. Cryptology ePrint Archive, Report 2018/1009 (2018). <https://eprint.iacr.org/2018/1009>

22. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 1–23. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13190-5_1
23. Peikert, C.: Public-key cryptosystems from the worst-case shortest vector problem: extended abstract. In: Mitzenmacher, M. (ed.) 41st Annual ACM Symposium on Theory of Computing, Bethesda, MD, USA, 31 May–2 June 2009, pp. 333–342. ACM Press (2009)
24. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: Gabow, H.N., Fagin, R. (eds.) 37th Annual ACM Symposium on Theory of Computing, Baltimore, MA, USA, 22–24 May 2005, pp. 84–93. ACM Press (2005)
25. Saarinen, M.J.O.: Hila5. Technical report, National Institute of Standards and Technology (2017). <https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions>