Attack on LAC Key Exchange in Misuse Situation

Aurélien Greuet [*,1], Simon Montoya [†,1,2], and Guénaël Renault [‡,2]

[1]IDEMIA France, Paris La Défense, France
[2]École Polytechnique, INRIA, Laboratoire Informatique de l'École Polytechnique, LIX, Équipe Grace, Palaiseau, France

Abstract

LAC is a Ring Learning With Error based cryptosystem that has been proposed to the NIST call for post-quantum standardization and passed the first round of the submission process. The particularity of LAC is to use an error-correction code ensuring a high security level with small key sizes and small ciphertext sizes. LAC team proposes a CPA secure cryptosystem, LAC.CPA, and a CCA secure one, LAC.CCA, obtained by applying the Fujisaki-Okamoto transformation on LAC.CPA. In this paper, we study the security of LAC Key Exchange (KE) mechanism, using LAC.CPA, in a misuse context: when the same secret key is reused for several key exchanges and an active adversary has access to a *mismatch oracle*. This oracle indicates information on the possible mismatch at the end of the KE protocol. In this context, we show that an attacker needs at most 8 queries to the oracle to retrieve one coefficient of a static secret key. This result has been experimentally confirmed using the reference and optimized implementations of LAC. Since our attack can break the CPA version in a misuse context, the Authenticated KE protocol, based on the CCA version, is not impacted. However, this research provides a tight estimation of LAC resilience against this type of attacks.

1 Introduction

The threat of a quantum computer that breaks most of the current public-key cryptosystems with Shor's Algorithm [1], led, in 2016, the National Institute of Standards and Technology (NIST) to begin a call for post-quantum safe

*Email address: aurelien.greuet@idemia.com
†Email address: simon.montoya@idemia.com
‡Email address: guenael.renault@lix.polytechnique.fr

public-key cryptography [2]. The NIST specifically asked for quantum safe Key Encapsulation Mechanisms (KEMs).

Among the different quantum resistant cryptosystems, those using ideal lattices based on a Ring instantiation of the Learning With Errors problem (RLWE) [3] are believed to be a promising direction to provide efficient and secure candidates. Indeed, 4 out of the 17 remaining KEMs of the round 2 of the NIST submissions are ideal lattices based on the RLWE problem [4, 5, 6, 7]. The interest of RLWE based KEM is confirmed by real life experiments. In 2016, Google started to experiment RLWE based KEM between Chrome and Google's services. Moreover, several RLWE-based KEMs are implemented by the Open Quantum Safe project in their OpenSSL and OpenSSH forks. This project involves academics, like University of Waterloo, and technology companies like Amazon Web Services or Microsoft Research. However, before a world-wide practical deployment of lattice-based KEMs, it is interesting to assess their security in different scenarios, for example in misuses conditions.

Motivation

In this paper we study LAC [4], a RLWE candidate to the NIST standardization process. It differs from other RLWE KEMs by its small key and ciphertext sizes, for an equivalent security level. Such small sizes can be an advantage, particularly in constrained environments and embedded systems. We focus on LAC.KE, a KEM based on the CPA secure public-key cryptosystem LAC.CPA. Our study is inspired by previous works in [8, 9, 10] which evaluate the resilience in a misuse context offered by 2 NIST KEM candidates. We want to pursue this evaluation with another NIST candidate to determine which one is the more resilient against this kind of attack.

Previous works

The seminal work of Diffie and Hellman [11] paved the way for active attacks on KE protocols. The idea of key mismatch attack on LWE based key exchange was first proposed by Fluhrer in [12, 13]. In a key mismatch attack, a participant's secret key is reused for several key establishments, and his private key can be recovered by comparing the shared secret key of the two participants.

Some lattice-based KEM of the NIST competition were analysed in the key reused context using a key mismatch oracle. In [14] Baetu *et al.* proposed a generic attack for several algorithms using the same structure called meta-algorithm. However, most of the algorithms attacked in [14] did not pass the first round of the submission, except Frodo-640 and NewHope512. The security of NewHope1024 CPA algorithm in this misuse scenario is analyzed by Bauer *et al.* in [8] and an improvement is proposed in [9]. More recently,

in the same context, an attack on Kyber CPA KEM is proposed by Ding *et al.* [10].

Our contribution

In this article, we investigate the resilience of the LAC KEM under a misuse case: we assume that the same secret key is reused for multiple key establishment and we assume that an attacker can use a key mismatch oracle as introduced in [8].

Since LAC uses encoding and compression functions different from a classical RLWE scheme, Fluhrer's attack [12] cannot be applied directly. Furthermore, these functions are different from those used in NewHope or Kyber, so we cannot apply straightforwardly the attacks described in [8, 9, 10].

The main idea of these attacks is to send forged ciphertexts to a victim, ensuring that its decryption will leak partial information of her static secret key. LAC algorithms use two encoding functions including an error-correction code BCH that can correct a limited number of errors. If a message exceeds the number of errors that the error-correction code can correct, then a decryption failure occurs. Thus, we propose to use this failure as an oracle to provide us leak about the static secret key.

More precisely, we propose a deterministic key mismatch attack on LAC KE for the first two security levels: LAC-128 and LAC-192, which required at most 2 queries per coefficient of the secret key. Afterwards, we adapt our attack to the highest security level LAC-256 which is still deterministic but we need at most 8 queries per coefficient of the secret key.

We experimented our attack with the reference and optimized implementation in `C` provided by the LAC team [4]. The code of our attack is available in [15].

Organization

In Section 2, we introduce some notations and describe LAC.CPA and LAC Key Exchange Mechanism and present the different parameters used in LAC algorithms. In Section 3, we describe the notion of *key mismatch oracle* introduced in [8], the attack for the first two security levels and afterwards the attack adapted to the higher security level.

2 Preliminaries

2.1 Notation

For an integer $q \geq 1$, let $\mathbb{Z}_q$ be the residue class group modulo q such that $\mathbb{Z}_q$ can be represented as $\{0, \ldots, q-1\}$. We define R_q the polynomial ring

$R_q = \mathbb{Z}_q[x]/(x^n + 1)$. A polynomial in R_q is of degree at most $(n-1)$ with coefficients in $\mathbb{Z}_q$. Given $P \in R_q$, we denote by $P[i]$ or P_i the coefficient associated with the monomial x^i. P can also be represented as a vector with n coordinates. Let the message space $\mathcal{M}$ be $\{0,1\}^{l_m}$ and the space of random seeds $\mathcal{S}$ be $\{0,1\}^{l_s}$, where l_m and l_s are two integers values. In the following, the notation $(a)_{l_v}$ ($l_v \in \mathbb{N}$), where a is a polynomial (or a vector) of size $n > l_v$, means we keep the first l_v coordinates of a. Let ψ_σ be the centered binomial distribution on the set $\{-1, 0, 1\}$. We denote the centered binomial distribution for n independents coordinates by ψ_σ^n i.e. for a vector a of size n each coefficient is sampled with the centered binomial distribution. In LAC algorithms we use:

1. $\psi_1 : Pr(x = 0) = \frac{1}{2}$, $Pr(x = -1) = \frac{1}{4}$, $Pr(x = 1) = \frac{1}{4}$
2. $\psi_{\frac{1}{2}} : Pr(x = 0) = \frac{3}{4}$, $Pr(x = -1) = \frac{1}{8}$, $Pr(x = 1) = \frac{1}{8}$

Given a set A, $U(A)$ is the uniform distribution over A. We denote by H a hash function and $\boldsymbol{Samp}(D, seed)$ an algorithm which samples a random variable according to a distribution D with a given seed. We denote by $[n', k, d]$ a set of parameters of an error-correction code (in our case a binary BCH code). n' denotes the length of the codewords, k is the dimension and d is the minimal Hamming distance of the code.

2.2 LAC

LAC is a Ring-LWE based public key encryption scheme over R_q. In order to balance performance and size, LAC team chose $q = 251$, that fits on one byte. This choice of a small modulus implies a lower security or a higher decryption error rate. To overcome these issues, an error-correction code is used, allowing to keep a low decryption error rate and maintain the same security level than schemes using larger modulus. Three security levels are proposed for LAC: LAC-128, LAC-192 and LAC-256. In this section, we describe the four algorithms ***CPA.KeyGen***, ***CPA.Encrypt***, ***CPA.Decrypt***, ***CPA.Decrypt256*** of the CPA version of LAC, the four subroutines ***BCHEncode***, ***BCHDecode***, ***Compress*** and ***Decompress*** and the CPA-KEM scheme.

Note that ***KeyGen*** and ***Encrypt*** are common to the three security levels. However, the decryption depends on the security level: Algorithm 3 is the decryption process for LAC-128 and LAC-192. The decryption routine for LAC-256 is described in Algorithm 4.

Algorithm 1 CPA.KeyGen()

Ensure: Key pair (pk, sk)

1: $seed_a \leftarrow U(\mathcal{S})$
2: $a \leftarrow \boldsymbol{Samp}(U(R_q), seed_a)) \in R_q$
3: $s \leftarrow \psi_\sigma^n$
4: $e \leftarrow \psi_\sigma^n$
5: $b \leftarrow a \times s + e \in R_q$
6: **return** $(pk, sk) = ((seed_a, b), s)$

Algorithm 2 ***CPA.Encrypt***$(pk = (seed_a, b), m, seed)$

Ensure: Ciphertext $c = (c_1, c_2)$

1: $a \leftarrow \boldsymbol{Samp}(U(R_q), seed_a) \in R_q$
2: $\widehat{m} \leftarrow \boldsymbol{BCHEncode}(m) \in \{0,1\}^{l_v}$
3: $r \leftarrow \boldsymbol{Samp}(\psi_\sigma^n, seed)$
4: $e_1 \leftarrow \boldsymbol{Samp}(\psi_\sigma^n, seed)$
5: $e_2 \leftarrow \boldsymbol{Samp}(\psi_\sigma^{l_v}, seed)$
6: $c_1 \leftarrow ar + e_1 \in R_q$
7: $c_2 \leftarrow (br)_{l_v} + e_2 + \lfloor \frac{q}{2} \rceil \widehat{m} \in \mathbb{Z}_q^{l_v}$
8: **if** LAC-256
9: $c_2 \leftarrow c_2 || c_2$ //D2 encoding
10: **end if**
11: $c_2 \leftarrow \boldsymbol{Compress}(c_2)$
12: **return** $c = (c_1, c_2)$

Algorithm 3 ***CPA.Decrypt***$(s := sk, c = (c_1, c_2))$

Ensure: Plaintext m

1: $c_2 \leftarrow \boldsymbol{Decompress}(c_2)$
2: $\widehat{M} \leftarrow c_2 - (c_1 s)_{l_v} \in \mathbb{Z}_q^{l_v}$
3: **for** $i = 0$ to $l_v - 1$ **do**
4: **if** $\frac{q}{4} \leq \widehat{M}_i < \frac{3q}{4}$ **then**
5: $\widehat{m}_i \leftarrow 1$
6: **else**
7: $\widehat{m}_i \leftarrow 0$
8: **end if**
9: **end for**
10: $m \leftarrow \boldsymbol{BCHDecode}(\widehat{m})$
11: **return** m

Algorithm 4 ***CPA.Decrypt256***$(s := sk, c = (c_1, c_2))$

Ensure: Plaintext m

1: $c_2 \leftarrow \boldsymbol{Decompress}(c_2)$
2: $\widehat{M} \leftarrow c_2 - (c_1 s)_{2l_v} \in \mathbb{Z}_q^{2l_v}$
3: **for** $i = 0$ to $l_v - 1$ **do** //D2 Decoding
4: $tmp_1, tmp_2 := \widehat{M}[i], \widehat{M}[i + \frac{l_v}{]}$
5: **if** $tmp_1 < \frac{q}{2}$
6: $tmp_1 \leftarrow q - tmp_1$
7: **else if** $tmp_2 < \frac{q}{2}$
8: $tmp_2 \leftarrow q - tmp_2$
9: **end if**
10: **if** $tmp_1 + tmp_2 - q < \frac{q}{2}$
11: $\widehat{m}_i \leftarrow 1$
12: **else**
13: $\widehat{m}_i \leftarrow 0$
14: **end if**
15: **end for**
16: $m \leftarrow \boldsymbol{BCHDecode}(\widehat{m})$
17: **return** m

2.2.1 Subroutines

BCHEncode* and *BCHDecode The function ***BCHEncode*** takes as input a message m of length l_m, pad it with $(k - l_m)$ zeros, where k is the dimension of the BCH code, and returns the corresponding value c on the

code. The function ***BCHDecode*** takes as input a message $\widehat{c}$ of length $n-1$, retrieves the codeword c closest to $\widehat{c}$ and returns m such that $c = mG$, where G is the generator matrix of the code.

Compress* and *Decompress*.** The function ***Compress takes as input a variable $c = (c_0, \ldots, c_{len_c})$ where each coefficient c_i is a 8-bits number and returns $c' = (c'_0, \ldots, c'_{len_c})$ where each c'_i is a 4 bits number obtained by keeping the highest 4 bits of c_i.

The function ***Decompress*** takes as input a variable $c' = (c'_0, \ldots, c'_{len_c})$ where each coefficient c'_i is a 4-bit number, and returns $\tilde{c} = (\tilde{c}_0, \ldots, \tilde{c}_{len_c})$ where each $\tilde{c}_i$ is a 8 bits number obtained by padding each coefficient c'_i with 4 zero bits.

2.2.2 Parameters

In the following we denote the secret key sk by s. Recall that LAC is a RLWE public-key encryption scheme on $R_q = \mathbb{Z}_q[x]/(x^n + 1)$, with input messages of length l_m.
LAC uses different parameters for its three algorithms:

Name	n	q	Distrib	l_m	l_v	Code(BCH) $[n', k, d]$	D2
LAC-128	512	251	ψ_1	256	$l_m + 144$	$[511, 367, 33]$	No
LAC-192	1024	251	$\psi_{\frac{1}{2}}$	256	$l_m + 72$	$[511, 439, 17]$	No
LAC-256	1024	251	ψ_1	256	$l_m + 144$	$[511, 367, 33]$	Yes

The value l_v depends on the BCH code. Let G be a generator matrix of the BCH code C. By the construction of LAC, G is on systematic form $G = (Id_k | A_{n'-k})$. In fact, we cannot keep only l_v bits of a codeword without this condition. The ***BCHEncode*** function takes as input a message m of length l_m and pad it with $(k - l_m)$ zeros. We obtain

$$(m_1, \ldots, m_{l_m}, 0_1, \ldots, 0_{k-l_m})G = (m_1, \ldots, m_{l_m}, 0_1, \ldots, 0_{k-l_m} | mA_{n'-k}) = c$$

We omit the $(k - l_m)$ zeros of c then $l_v = l_m + (n' - k)$.

2.2.3 LAC Key Exchange

We describe the LAC Key Exchange, based on the CPA version of the LAC public-key encryption scheme.

Alice		Bob
$(pk, s) \leftarrow$ ***CPA.KeyGen()***		
	$\xrightarrow{pk}$	
		$r \leftarrow U(\{0,1\}^{l_m})$
		$c \leftarrow$ ***CPA.Encrypt***(pk, r)
		$Key_B \leftarrow H(pk, r) \in \{0,1\}^{l_k}$
	$\xleftarrow{c}$	
$r' \leftarrow$ ***CPA.Decrypt***(s, c)		
$Key_A \leftarrow H(pk, r') \in \{0,1\}^{l_k}$		

If Key Exchange succeeds then $r' = r$ and $Key_B = Key_A$.

3 Attack on LAC Key Exchange

In this section, we present the main result of this paper. We start by defining the scenario of the attack.

3.1 Attack Model

We suppose that Alice does a misuse of the Key Exchange Mechanism by caching her secret s. More precisely:

Assumption 1. *Alice keeps her secret key constant for several CPA key establishments requests.*

Eve is a malicious active adversary who acts as Bob and can cheat and generate c that is not the encryption of a random r. To mount the active attack, we suppose Eve has access to a key mismatch oracle defined as follow.

Definition 1. *A* key mismatch oracle *outputs a bit of information on the possible mismatch at the end of the key encapsulation mechanism. In the LAC context, this oracle, denoted $\mathcal{O}$, takes any message c and any key guess μ as input and outputs:*

$$\mathcal{O}(c, \mu) = \begin{cases} 1 & \text{if } H(\textbf{\textit{CPA.Decrypt}}(s, c)) = \mu \\ -1 & \text{otherwise} \end{cases}$$

This oracle can also be used by Bob during an honest key exchange with Alice, when he verifies the match between his key and Alice's one.

The idea of the attack mounted by Eve is to send forged ciphertexts to Alice to ensure that she obtains information on some coefficients of Alice's secret key. As Eve knows that $c = (c_1, c_2)$ and s are used during the decryption algorithms (s is multiplied by c_1), she will mount an attack using this fact and following three mains steps:

- Construct c_1 such that some coefficients of the secret key are exposed

- Construct c_2 such that the result of Alice's decryption can be monitored as a function of the key guess
- Call to the oracle $\mathcal{O}$ to obtain information about our key guesses

The following section shows how to choose appropriate (c, μ) to retrieve information on s. We assume that Eve has access to the oracle $\mathcal{O}$.

3.2 Attack on LAC-128-KE and LAC-192-KE

First, we use a simplified version where we do not consider ***Compress*** and ***Decompress*** functions. We follow the different steps of the decryption algorithm 3.

3.2.1 Simplified version

In this first result, we show how one can forge a LAC ciphertext in order to impose which plaintext will be obtained after decryption.

Proposition 1. *Assume that Eve forges $c = (c_1, c_2)$ such that :*

- $c_1 = -ax^{n-w}$ *where* w *is an integer* $0 \leq w < n$ *and* $0 \leq a < \frac{q}{4}$
- $c_2 = (\alpha_0, \ldots, \alpha_{l_v-1})$ *where* $\alpha_i = \frac{q}{2}$ *or* 0 *for all* i *in* $[0, l_v - 1]$.

Then she can determine the plaintext m that Alice obtains after decryption.

Proof. When Alice deciphers Eve's ciphertext she:

1. Computes $\widehat{M} = c_2 - (c_1 s)_{l_v}$
2. Compares each coefficient of $\widehat{M}$ to $\frac{q}{4}$ and $\frac{3q}{4}$ to define $\widehat{m}$
3. Retrieves m using ***BCHDecode*** algorithm on $\widehat{m}$

Let $c_1 = -ax^{n-w}$ and $s = s_0 + s_1 x^1 + \ldots + s_{n-1}x^{n-1}$ then

$$c_1 s = as_{n-w} + as_{n-w+1}x + \ldots + as_{n-1}x^w - as_0 x^{w+1} - \ldots - as_{n-w-1}x^{n-1}$$

and the polynomial $c_1 s$ can be represented as a vector $c_1 s = (as_{n-w}, \ldots, -as_{n-w-1})$. During the computation of $\widehat{M}$, two cases are possible:

- $w < l_v$ then:

$$\begin{aligned}\widehat{M} &= c_2 - (c_1 s)_{l_v} \\ &= (\alpha_0 - as_{n-w}, \alpha_1 - as_{n-w+1}, \ldots, \alpha_w + as_0, \ldots, \alpha_{l_v-1} + as_{l_v-1})\end{aligned}$$

- $w \geq l_v$ then:

$$\begin{aligned}\widehat{M} &= c_2 - (c_1 s)_{l_v} \\ &= (\alpha_0 - as_{n-w}, \alpha_1 - as_{n-w+1}, \ldots, \alpha_{l_v-1} - as_{l_v-1})\end{aligned}$$

After this computation each coefficient of $\widehat{M}$ is compared to $\frac{q}{4} \leq \widehat{M}_i < \frac{3q}{4}$. Recall that since $s \leftarrow \psi_\sigma^n$, each of its coefficients belongs to $\{-1, 0, 1\}$. Let i be an integer such that $0 \leq i < n$ and $j \equiv n - w + i \mod n$. If $\alpha_i = \frac{q}{2}$ one gets:

$$\alpha_i \mp a s_j = \begin{cases} \frac{q \mp 2a}{2} & \text{if } s_j = \pm 1 \\ \frac{q}{2} & \text{if } s_j = 0 \\ \frac{q \pm 2a}{2} & \text{if } s_j = \mp 1 \end{cases}$$

These 3 values lie in $\left[\frac{q}{4}, \frac{3q}{4}\right[$ if $0 \leq a \leq \frac{q}{4}$. If $\alpha_i = 0$ one gets:

$$\alpha_i \mp a s_j = \begin{cases} \pm a & \text{if } s_j = \mp 1 \\ 0 & \text{if } s_j = 0 \\ \mp a & \text{if } s_j = \pm 1 \end{cases}$$

These 3 values do not lie in $\left[\frac{q}{4}, \frac{3q}{4}\right[$ if $0 \leq a < \frac{q}{4}$ or $\frac{3q}{4} \leq a \leq q$. Thus, Eve can choose $a < \frac{q}{4}$ and $\alpha_i = \frac{q}{2}$ or 0 to determine what Alice obtains on the first l_v coordinates of $\widehat{m}$ and then Eve can deduce, by applying BCH decoding, what Alice obtains at the end of the decryption procedure. □

Using Proposition 1 Eve can forge c such that she knows what Alice obtains after decryption. Let explain more precisely this result with a concrete example:

Example 1. *Suppose that Eve wants that Alice obtains, after decryption, the message* $m = \textbf{\textit{BCHDecode}}(1, 0, 1, 1, 0, \ldots, 0)$. *Then she forges* $c = (c_1, c_2)$ *such that:*

- $c_1 = -\frac{q}{5} x^{512}$. *In fact Eve can take any* c_1 *such that* $c_1 = -a x^{n-w}$ *with* $0 \leq a < \frac{q}{4}$
- $c_2 = (\frac{q}{2}, 0, \frac{q}{2}, \frac{q}{2}, 0, \ldots, 0)$

First Alice computes:

$$\begin{aligned} \widehat{M} &= c_2 - (c_1 s)_{l_v} \\ &= \left(\frac{q}{2}, 0, \frac{q}{2}, \frac{q}{2}, 0, \ldots, 0\right) - \frac{q}{5}\left(s_0, s_1, \ldots, s_{l_v}\right),\ s_i \text{ belongs to } \{-1, 0, 1\} \\ &= \left(\frac{q}{2} - \frac{q}{5}s_0, -\frac{q}{5}s_1, \frac{q}{2} - \frac{q}{5}s_2, \frac{q}{2} - \frac{q}{5}s_3, -\frac{q}{5}s_4, \ldots, -\frac{q}{5}s_{l_v}\right) \end{aligned}$$

Then, Alice compares each coefficients of $\widehat{M}$ *to* $\frac{q}{4}$ *and* $\frac{3q}{4}$. *She obtains (see proof of Proposition 1):*

$$\widehat{m} = (1, 0, 1, 1, 0, \ldots, 0)$$

At the end, Alice obtains m *by applying* ***BCHDecode*** *algorithm to* $\widehat{m}$. *Thus, Eve had forged* c *such that Alice has* $m = \textbf{\textit{BCHDecode}}(1, 0, 1, 1, 0, \ldots, 0)$.

Now Eve needs to construct forged ciphertexts that allow a key guessing strategy.

Proposition 2. *Let s'_w be a guess done by Eve on the w-th coefficient of the secret key s, where $0 \leq w < n$. Assume $s_w = 1$ or -1. If Eve forges $c = (c_1, c_2)$ such that:*

- $c_1 = -ax^{n-w}$ *with* $\frac{q}{8} < a < \frac{q}{4}$
- $c_2 = (as'_w, \alpha_1, \ldots, \alpha_{l_v-1})$ *where* $\alpha_i = \frac{q}{2}$ *or* 0 *for all* i *in* $[0, l_v - 1]$ *and* $\frac{q}{8} < a < \frac{q}{4}$.

Then she can verify her key guess.

Proof. Suppose that Eve wants to retrieve the w-th coefficient of s, when Alice decipher Eve ciphertext she computes:

$$\widehat{M} = c_2 - (c_1 s)_{l_v} = (as'_w - as_w, \alpha_1 - as_{w+1}, \ldots)$$

According to Proposition 1, Eve can determine what Alice obtains for every coefficient different from the key guess of Alice's side. Let see what happens with $as'_w - as_w$.

$$as'_w - as_w = \begin{cases} 0 & \text{if } s'_w = s_w \\ 2a & \text{if } s'_w = 1 \text{ and } s_w = -1 \\ -2a & \text{if } s'_w = -1 \text{ and } s_w = 1 \\ \mp a & \text{if } s'_w = 0 \text{ or } s_w = 0 \end{cases}$$

Let $\frac{q}{8} < a < \frac{q}{4}$ then

$$\frac{q}{4} < 2a < \frac{q}{2} \text{ and } -2a = q - 2a \text{ satisfies } \frac{q}{2} < q - 2a < \frac{3q}{4}$$

Then with $\frac{q}{8} < a < \frac{q}{4}$ and the good key guess a 1 is return at the first coordinate of $\widehat{m}$ and 0 if she did the wrong key guess. Eve can determine what Alice obtains by applying ***BCHDecode*** algorithm to $\widehat{m}$ and thus verifies her key guess. □

Proposition 2 ensures that if Eve does the good key guess, Alice obtains $m = \boldsymbol{BCHDecode}(1, \ldots)$. Otherwise, she obtains $m = \boldsymbol{BCHDecode}(0, \ldots)$. Computational details are give in the following example.

Example 2. *Suppose that Eve wants to learn information about the first bit of Alice's secret key. Eve forges $c = (c_1, c_2)$ such that:*

- $c_1 = -\frac{q}{5}x^{512-0}$.
- $c_2 = \left(\frac{q}{5}s'_0, 0, \frac{q}{2}, \frac{q}{2}, 0, \ldots, 0\right)$ *where s'_w is Eve's key guess.*

First Alice computes:

$$\begin{aligned}\widehat{M} &= c_2 - (c_1 s)_{l_v} \\ &= \left(\frac{q}{5}s'_0, 0, \frac{q}{2}, \frac{q}{2}, 0, \ldots, 0\right) - \frac{q}{5}\left(s_0, s_1, \ldots, s_{l_v}\right),\ s_i \text{ belongs to } \{-1, 0, 1\} \\ &= \left(\frac{q}{5}s'_0 - \frac{q}{5}s_0, -\frac{q}{5}s_1, \frac{q}{2} - \frac{q}{5}s_2, \frac{q}{2} - \frac{q}{5}s_3, -\frac{q}{5}s_4, \ldots, -\frac{q}{5}s_{l_v}\right)\end{aligned}$$

Then, Alice compares each coefficients of $\widehat{M}$ *to* $\frac{q}{4}$ *and* $\frac{3q}{4}$*. She obtains (see proof of Proposition 2):*

$$\begin{aligned}\widehat{m} &= (1, 0, 1, 1, 0, \ldots, 0) \text{ if } s'_0 = -s_0 \text{ and } s_0 \neq 0 \\ \widehat{m} &= (0, 0, 1, 1, 0, \ldots, 0) \text{ otherwise}\end{aligned}$$

At the end, Alice obtains m *by applying* ***BCHDecode*** *algorithm to* $\widehat{m}$*. Thus, Eve did the good key guess if Alice gets* $m =$ ***BCHDecode***$(1, 0, 1, 1, 0, \ldots, 0)$.

Proposition 2 already gives interesting information to Eve but it is not enough to mount and attack since:

- Eve needs a way to verify if Alice obtains
  - either $m =$ ***BCHDecode***$(1, 0, 1, 1, 0, \ldots, 0)$
  - or $m =$ ***BCHDecode***$(0, 0, 1, 1, 0, \ldots, 0)$.
- Moreover, most of the time

$$\textbf{\textit{BCHDecode}}(1, 0, 1, 1, 0, \ldots, 0) = \textbf{\textit{BCHDecode}}(0, 0, 1, 1, 0, \ldots, 0).$$

Nonetheless, Eve can use Proposition 2, the BCH code decryption failure and the oracle to overcome these issues as it is stated in the following theorem.

Theorem 1. *If Eve forges* $c = (c_1, c_2)$ *such that:*

- $c_1 = -ax^{n-w}$ *where* w *is an integer,* $0 \leq w < n$*, and* $\frac{q}{8} < a < \frac{q}{4}$
- $c_2 = (as'_w, \alpha_1, \ldots, \alpha_{l_v - 1})$ *where* $\alpha_i = \frac{q}{2}$ *or* 0 *for all* $i \in [0, l_v - 1]$*,* s'_w *is Eve's key guess and* $\frac{q}{8} < a < \frac{q}{4}$.

Then with at most 2 *calls to the oracle* $\mathcal{O}$*, Eve retrieves the* w*-th coefficient of* s*.*

Proof. According to Proposition 2, Eve can monitor Alice's decryption procedure if she does the good key guess.

An error-correction code can correct at most $\frac{d-1}{2}$ errors (where d is the minimal Hamming distance of the BCH code). The idea is that after comparison with $\frac{q}{4}$ and $\frac{3q}{4}$, $\widehat{m}$ is a codeword with $\frac{d}{2}$ errors if Eve did the wrong key guess, causing a decoding error. Suppose Eve wants to retrieve the w-th coefficient of s:

1. Eve chooses a codeword called *cdword* with a 1 at the first coordinate such that $cdword = mG$ where G is the generator matrix of the BCH code
2. Eve injects $\frac{d-1}{2}$ errors to *cdword* at any coordinate except the first one
3. Eve chooses a verifying $\frac{q}{8} < a < \frac{q}{4}$ according to Proposition 2
4. Eve constructs c_1, c_2 with her key guess at the first bit of c_2: $c_2[0] = as'_w$ and such that after comparison with $\frac{q}{4}$ and $\frac{3q}{4}$, Alice retrieves *cdword* with $\frac{d-1}{2}$ or *cdword* with $\frac{d}{2}$
5. Eve sends $c = (c_1, c_2)$ to Alice

With this construction, Alice obtains a codeword with $\frac{d}{2}$ errors if Eve provides a wrong key guess. Then Eve can verify whether she did the correct key guess with the oracle as follow:

If $s'_w = 1$ and $\mathcal{O}(c, m) = 1$ then $s_w = -1$
If $s'_w = -1$ and $\mathcal{O}(c, m) = 1$ then $s_w = 1$
Otherwise $s_w = 0$

□

Algorithms 5 and 6 describe the attack. In our implementation [15], we choose $a = \frac{q}{7}$.

Algorithm 5 forge(hyp,bit)

Ensure: Forge ciphertext $c = (c_1, c_2)$

$c_1 := -\frac{q}{7}x^{n-bit}$
$m := [0 : \textbf{for } i := 0 \text{ to } 255]$
$m[0] := 1$
$codeword := (m||0..0)G$
Add $\frac{d-1}{2}$ errors to codeword (but not on codeword[0])
For $i = 0$ **to** Len($codeword$) :
  if $i == 0$:
    $c_2[0] \leftarrow \text{hyp} \times \frac{q}{7}$
  else if $codeword[i] == 1$:
    $c_2[i] \leftarrow \frac{q}{2}$
  else
    $c_2[i] \leftarrow 0$
  end if
end for
Return$(m, c = (c_1, c_2))$

Algorithm 6 recover_one_bit(bit)

Ensure: A bit of s

$m, c := \textbf{forge}(-1, bit)$
If $\mathcal{O}(c, m) == 1$:
  Return 1
end if
$m, c := \textbf{forge}(1, bit)$
If $\mathcal{O}(c, m) == 1$:
  Return -1
end if
Return 0

A key of length n can be fully recovered using Theorem 1 with at most $2 \times n$ requests to the oracle. LAC-128 works with keys of length $n = 512$ and LAC-192 with length $n = 1024$.

3.2.2 Full version

The subroutine ***Compress*** removes the 4 lowers bits of c_2 and they are replaced by 4 zero-bit when the subroutine ***Decompress*** is applied at the beginning of the decryption process. Thus, each coefficient of c_2 can be only equal to 16, 32, 64, 128 and any sum of theses values.
For c_2 in our attack, we only consider the values $\frac{q}{7}$, $-\frac{q}{7}$ and $\frac{q}{2}$. In our implementation [15] we approximate $\frac{q}{7} \approx 32$, $-\frac{q}{7} \approx 128+64+16 = 210$ and $\frac{q}{2} \approx 128$. Proposition 2 is still verified and we still retrieve s with at most $2 \times n$ requests to the oracle by the Theorem 1.

3.3 Attack on LAC-256-KE

The key exchange procedure is mainly the same as previously but for decryption, Alice uses the ***CPA.Decrypt256*** routine. Since the $D2$ encoding is used in this routine, the coordinates of c_2 are duplicated: for all $0 \leq i \leq l_v - 1$, $c_2[i + \frac{length(c_2)}{2}] = c_2[i]$. This redundancy in c_2 allows to decrease the decoding error. Thus, the decryption procedure is different from LAC-128 and LAC-192.

3.3.1 *CPA.Decrypt256* description

The first step of the decryption it's to compute $\widehat{M} = c_2 - (c_1 s)_{2l_v}$. The decryption algorithm considers two cases:

Case 1. If $\widehat{M}[i]$ and $\widehat{M}[i+l_v] < \frac{q}{2}$ or $\widehat{M}[i]$ and $\widehat{M}[i+l_v] \geq \frac{q}{2}$ then algorithm ***CPA.Decrypt256*** compares:

$$\frac{q}{4} < \frac{\widehat{M}[i] + \widehat{M}[i+l_v]}{2} < \frac{3q}{4}$$

Case 2. If $\widehat{M}[i] < \frac{q}{2}$ and $\widehat{M}[i+l_v] \geq \frac{q}{2}$ or $\widehat{M}[i] \geq \frac{q}{2}$ and $\widehat{M}[i+l_v] < \frac{q}{2}$ then ***CPA.Decrypt256*** compares:

$$0 < \frac{|\widehat{M}[i] - \widehat{M}[i+l_v]|}{2} < \frac{q}{4}$$

In the following we notice when we are in the case 1 or 2.

3.3.2 Attack on LAC-256-KE simplified

As previously we first use a simplified version where we do not consider ***Compress*** and ***Decompress*** subroutines.

Proposition 3. *Assume that Eve forges $c = (c_1, c_2)$ such that:*

- $c_1 = -ax^{n-w}$ *where* w *is an integer* $0 \leq w < n$ *and* $0 \leq a < \frac{q}{4}$
- $c_2 = (\alpha_0, \ldots, \alpha_{l_v-1}, \alpha_{l_v}, \ldots, \alpha_{2l_v-1})$ *where* $\alpha_i = \frac{q}{2}$ *or* 0 *for all* i *in* $[0, 2l_v - 1]$

Then she can determine the plaintext m that Alice obtains after decryption.

Proof. Assuming Alice receives $c = (c_1, c_2)$ then she:

1. Computes $\widehat{M} = c_2 - (c_1 s)_{2l_v}$
2. Compares $\frac{q}{4} < \frac{\widehat{M}[i]+\widehat{M}[i+l_v]}{2} < \frac{3q}{4}$ or $0 < \frac{|\widehat{M}[i]-\widehat{M}[i+l_v]|}{2} < \frac{q}{4}$ for $i = 0$ to $l_v - 1$ to define each coefficient of $\widehat{m}$
3. Retrieves m using ***BCHDecode*** algorithm on $\widehat{m}$

Let $c_1 = -ax^{n-w}$ and $s = s_0 + s_1x^1 + \ldots + s_{n-1}x^{n-1}$ then

$$c_1 s = as_{n-w} + as_{n-w+1}x + \ldots + as_{n-1}x^w - as_0x^{w+1} - \ldots - as_{n-w-1}x^{n-1}$$

We can represent $c_1 s$ as a vector $c_1 s = (as_{n-w}, \ldots, -as_{n-w-1})$. During the computation of $\widehat{M}$ we can have two cases:

- $w < 2l_v$ then:

$$\begin{aligned}\widehat{M} &= c_2 - (c_1 s)_{2l_v} \\ &= (\alpha_0 - as_{n-w}, \alpha_1 - as_{n-w+1}, \ldots, \alpha_w + as_0, \ldots, \alpha_{2l_v-1} + as_{2l_v-1})\end{aligned}$$

- $w \geq 2l_v$ then:

$$\begin{aligned}\widehat{M} &= c_2 - (c_1 s)_{2l_v} \\ &= (\alpha_0 - as_{n-w}, \alpha_1 - as_{n-w+1}, \ldots, \alpha_{2l_v-1} - as_{2l_v-1})\end{aligned}$$

Recall that since $s \leftarrow \psi_\sigma^n$, each of its coefficients belongs to $\{-1, 0, 1\}$. Let i be an integer such that $0 \leq i < l_v$ and $j \equiv n - w + i \mod n$.
For the sake of clarity, we consider the case where $\widehat{M}[i] = \alpha_i - as_j$ and $\widehat{M}[i+l_v] = \alpha_{i+l_v} - as_{j+l_v}$. The other case is $\widehat{M}[i] = \alpha_i + as_j$ and $\widehat{M}[i+l_v] = \alpha_{i+l_v} + as_{j+l_v}$ the proof is in the appendix.
If $\alpha_i = \frac{q}{2}$ one gets:

- If $s_j = s_{j+l_v}$ or $s_j + s_{j+l_v} = -1$ we are in the Case 1 described in Paragraph 3.3.1, where $\alpha_i - as_j = \widehat{M}[i]$:

$$\alpha_i = \alpha_{i+l_v} = \frac{q}{2},$$

$$\frac{(\alpha_i - as_j) + (\alpha_{i+l_v} - as_{j+l_v})}{2} = \begin{cases} \frac{q-2a}{2} & \text{if } s_j = s_{j+l_v} = 1 \\ \frac{q+2a}{2} & \text{if } s_j = s_{j+l_v} = -1 \\ \frac{q}{2} & \text{if } s_j = s_{j+l_v} = 0 \\ \frac{q+a}{2} & \text{if } s_j + s_{j+l_v} = -1 \end{cases}$$

  These 3 values lie in $\left]\frac{q}{4}, \frac{3q}{4}\right[$ if $0 \leq a < \frac{q}{4}$.

- Otherwise we are in the Case 2 described in Paragraph 3.3.1, where $\alpha_i - as_j = \widehat{M}[i]$:

$$\alpha_i = \alpha_{i+l_v} = \frac{q}{2},$$

$$\frac{|(\alpha_i - as_j) - (\alpha_{i+l_v} - as_{j+l_v})|}{2} = \begin{cases} \frac{a}{2} & \text{if } s_j + s_{j+l_v} = 1 \\ a & \text{if } s_j = -1, s_{j+l_v} = 1 \\ & \text{or } s_j = 1, s_{j+l_v} = -1 \end{cases}$$

  These values lie in $\left[0, \frac{q}{4}\right[$ if $0 \leq a < \frac{q}{4}$.

Then for both cases, if $c_1 = -ax^{n-w}$ with $\alpha_i, \alpha_{i+l_v} = \frac{q}{2}$, we can ensure that we have a 1 after comparison.
If $\alpha_i = 0$ then we are in the Case 1 described in Paragraph 3.3.1, where $\alpha_i - as_j = \widehat{M}[i]$:

$$\frac{(\alpha_i - as_j) + (\alpha_{i+l_v} - as_{j+l_v})}{2} = \begin{cases} a & \text{if } s_j = s_{j+l_v} = -1 \\ 0 & \text{if } s_j = -s_{j+l_v} \text{ or } s_j = s_{j+l_v} = 0 \\ -a & \text{if } s_j = s_{j+l_v} = 1 \\ \pm\frac{a}{2} & \text{otherwise} \end{cases}$$

Then these 3 values do not lie in $\left]\frac{q}{4}, \frac{3q}{4}\right[$ for $0 \leq a < \frac{q}{4}$.

□

So Eve can choose $a < \frac{q}{4}$ and $\alpha_i = \frac{q}{2}$ or 0 to know what Alice obtains on the $2l_v$ coordinates of $\widehat{m}$ and then Eve can deduce what Alice obtains at the end of decryption for m. Eve needs to construct forged ciphertexts which allows to verify her key guesses.

Proposition 4. *Assume Eve puts key guesses at the w-th and the $(w+l_v)$-th coefficients of s.*
Assume that s_w and s_{w+l_v} are different from 0 and Eve forges $c = (c_1, c_2)$ such that:

- $c_1 = -ax^{n-w}$ *where w is an integer $0 \leq w < n$ and $\frac{q}{8} < a < \frac{q}{4}$*

- $c_2 = (\alpha_0, \ldots, as'_w, \ldots, \alpha_{l_v-1}, \alpha_{l_v}, \ldots, as'_{w+l_v}, \ldots, \alpha_{l_{v-1}})$ *where* $\alpha_i = \frac{q}{2}$ *or* 0 *for all* i *in* $[0, 2l_v - 1]$, s'_w *and* s'_{w+l_v} *are Eve's key guesses and* $\frac{q}{8} < a < \frac{q}{4}$

Then she can verify her key guesses.

Proof. According to Proposition 3 Eve can determine what Alice obtains at the end of the decryption procedure for every coefficient different from the key guesses. Assume that Eve wants to retrieve the w-th and $(w + l_v)$-th coefficients of s, $s'_w = s'_w = 1$ and $\frac{q}{8} < a < \frac{q}{4}$. So we are in the Case 1 described in Paragraph 3.3.1, let see what happens with $\frac{\widehat{M}_w + \widehat{M}_{w+l_v}}{2} = \frac{as'_w - as_w + as'_{w+l_v} - as_{w+l_v}}{2}$:

$$\frac{a - as_w + a - as_{w+l_v}}{2} = \begin{cases} 2a & \text{if } s_w = s_{w+l_v} = -1 \\ 0 & \text{if } s_w = s_{w+l_v} = 1 \\ \frac{3a}{2} & \text{if } s_w = 0,\ s_{w+l_v} = -1 \\ & \text{or } s_w = -1,\ s_{w+l_v} = 0 \\ \frac{a}{2} & \text{otherwise} \end{cases}$$

Then only the case $\frac{a - as_0 + a - as_{l_v}}{2} = \frac{3a}{2}$ can put a 1 to $\widehat{m}_w$ if $\frac{q}{8} < a < \frac{q}{4}$.
With the same condition on a and with the same method Eve can have :

- If $s'_w = s'_{w+l_v} = 1$ and $\widehat{m_w} = 1$ then $s_w = s_{w+l_v} = -1$
- If $s'_w = s'_{w+l_v} = -1$ and $\widehat{m_w} = 1$ then $s_w = s_{w+l_v} = 1$
- If $s'_w = 1$, $s'_{w+l_v} = -1$ and $\widehat{m_w} = 1$ then $s_w = -1$ and $s_{w+l_v} = 1$
- If $s'_w = -1$, $s'_{w+l_v} = 1$ and $\widehat{m_w} = 1$ then $s_w = 1$ and $s_{w+l_v} = -1$

□

Proposition 4 ensures that Eve can know what Alice obtains if Alice's secrets coefficients are different from 0. Let see what happens when one of the two coefficient is equal to 0.

Proposition 5. *Suppose Eve puts key guesses at the* w*-th and the* $(w+l_v)$*-th coefficients of* s*.*
Assume that $s_w = 0$ *or* $s_{w+l_v} = 0$ *and Eve forges* $c = (c_1, c_2)$ *such that:*

- $c_1 = -ax^{n-w}$ *where* w *is an integer* $0 \leq w < n$ *and* $\frac{q}{6} < a < \frac{q}{4}$
- $c_2 = (\alpha_0, \ldots, as'_w, \ldots, \alpha_{l_v-1}, \alpha_{l_v}, \ldots, as'_{w+l_v}, \ldots, \alpha_{l_{v-1}})$ *where* $\alpha_i = \frac{q}{2}$ *or* 0 *for all* i *in* $[0, 2l_v - 1]$, s'_w *and* s'_{w+l_v} *are Eve's key guesses and* $\frac{q}{6} < a < \frac{q}{4}$

Then she can verify her key guesses.

Proof. Assume that Eve wants to retrieve the w-th and $(w + l_v)$-th coefficients of s.
Suppose $a < \frac{q}{4}$, $s'_w = 1$ and $s'_{w+l_v} = 1$. Let see what happens with $\frac{\widehat{M}_w + \widehat{M}_{w+l_v}}{2} = \frac{as'_w - as_w + as'_{w+l_v} - as_{w+l_v}}{2}$ (Case 1 described in Paragraph 3.3.1):

$$\frac{a - as_w + a - as_{w+l_v}}{2} = \begin{cases} \frac{3a}{2} & \text{if } s_w = -1 \text{ and } s_{w+l_v} = 0 \\ & \text{or } s_w = 0 \text{ and } s_{w+l_v} = -1 \\ \frac{a}{2} & \text{if } s_w = 0 \text{ and } s_{w+l_v} = 1 \\ & \text{or } s_w = 1 \text{ and } s_{w+l_v} = 0 \\ a & \text{if } s_w = s_{w+l_v} = 0 \end{cases}$$

With $\frac{q}{6} < a < \frac{q}{4}$ then only the case where the result is $\frac{3a}{2}$ can put a 1 to $\widehat{m}_w$. However Eve needs to determine if $s_w = -1$ or $s_{w+l_v} = -1$.

Suppose $a < \frac{q}{4}$, $s'_w = -1$ and $s'_{w+l_v} = 1$, $s_w = -1$ and $s_{w+l_v} = 0$ or $s_w = 0$ and $s_{w+l_v} = -1$. Here, we need to consider the both decryption cases described in Paragraph 3.3.1. Let see what happens:

- If $s_w = -1$ and $s_{w+l_v} = 0$ we are in Case 1 3.3.1:

$$\frac{q}{4} < a < \frac{3q}{4}$$

- If $s_w = 0$ and $s_{w+l_v} = -1$ we are in Case 2 3.3.1:

$$0 < \frac{|-a - 2a|}{2} < \frac{q}{4} \text{ implies } 0 < a < \frac{3q}{8}$$

However $a < \frac{q}{4}$, then only one case can put a 1 to $\widehat{m}_0$.

With the same condition on a and with the same method, Eve can retrieve the others values:

- If $s'_w = 1$, $s'_{w+l_v} = 1$ and $\widehat{m_w} = 1$ then $s_w = -1$, $s_{w+l_v} = 0$ or $s_w = 0$, $s_{w+l_v} = -1$
  - If $s'_w = -1$, $s'_{w+l_v} = 1$ and $\widehat{m_w} = 1$ then $s_w = 0$, $s_{w+l_v} = -1$ else $s_w = -1$, $s_{w+l_v} = 0$
- If $s'_w = -1$, $s'_{w+l_v} = -1$ and $\widehat{m_w} = 1$ then $s_w = 1$, $s_{w+l_v} = 0$ or $s_w = 0$, $s_{w+l_v} = 1$
  - If $s'_w = 1$, $s'_{w+l_v} = -1$ and $\widehat{m_w} = 1$ then $s_w = 0$, $s_{w+l_v} = 1$ else $s_w = 1$, $s_{w+l_v} = 0$

□

As previously, Propositions 4 and 5 are not enough to mount an attack for the same reasons:

- Eve needs a way to verify what Alice obtains.
- A bit of difference on $\widehat{m}$ is corrected by the BCH code. Thus, at the end of the decryption procedure Alice and Eve have the same plaintext.

Nonetheless, Eve can use Propositions 4 and 5, the BCH code decryption failure and the oracle to overcome these issues.

Theorem 2. *Assume Eve forges $c = (c_1, c_2)$ such that:*

- $c_1 = -ax^{n-w}$ *where w is an integer $0 \leq w < n$ and $\frac{q}{8} < a < \frac{q}{4}$*
- $c_2 = (\alpha_0, \ldots, as'_w, \ldots, \alpha_{l_v-1}, \alpha_{l_v}, \ldots, as'_{w+l_v}, \ldots, \alpha_{l_{v-1}})$ *where $\alpha_i = \frac{q}{2}$ or 0 for all i in $[0, 2l_v - 1]$, s'_w and s'_{w+l_v} are Eve's key guesses and $\frac{q}{8} < a < \frac{q}{4}$ or $\frac{q}{6} < a < \frac{q}{4}$*

then with 8 calls to the oracle $\mathcal{O}$, Eve retrieves the w-th and $(w + l_v)$-th coefficients of s.

Proof. The idea is the same as LAC-128 and 192, Eve takes c_2 to ensure, after comparison in ***CPA.Decrypt256***, that $\widehat{m}$ is a codeword with $\frac{d}{2}$ errors if she did the wrongs key guesses, causing a decoding error (where d is the minimal Hamming distance of the BCH code).
According to Propositions 4 and 5, Eve can monitor Alice's decryption procedure if she does the goods key guesses.
An error-correction code can correct at most $\frac{d-1}{2}$ errors. The idea is that after comparison, $\widehat{m}$ is a codeword with $\frac{d}{2}$ errors if Eve did the wrongs key guesses, causing a decoding error. Suppose Eve wants to retrieve the w-th and the $(w + l_v)$-th coefficients of s:

1. Eve chooses a codeword called *cdword* with a 1 at the first coordinate such that $cdword = mG$ where G is the generator matrix of the BCH code
2. Eve injects $\frac{d-1}{2}$ errors to *cdword* at any coordinate except the first one
3. Eve chooses a verifying $\frac{q}{8} < a < \frac{q}{4}$ if she is on the case of Proposition 4 or $\frac{q}{6} < a < \frac{q}{4}$ if she is on the case of Proposition 5
4. Eve constructs c_1 and c_2 with her key guesses are at the first and l_v-th coefficient of c_2: $c_2[0] = as'_w$ and $c_2[l_v] = as'_{l_v+w}$ and such that after comparison, Alice retrieves *cdword* with $\frac{d-1}{2}$ errors or *cdword* with d errors
5. Eve sends $c = (c_1, c_2)$ to Alice

With this construction Alice obtains a codeword with $\frac{d}{2}$ errors if Eve does wrongs key guesses. Then Eve can verify if she did the goods key guesses with the oracle.
First Eve determines if s_w and s_{w+l_v} are different from 0 (Due by Proposition 4):

If $s'_w = 1, s'_{w+l_v} = 1$ and $\mathcal{O}(c, m) = 1$ then $s_w = -1$ and $s_{w+l_v} = -1$
If $s'_w = -1, s'_{w+l_v} = -1$ and $\mathcal{O}(c, m) = 1$ then $s_w = 1$ and $s_{w+l_v} = 1$
If $s'_w = 1, s'_{w+l_v} = -1$ and $\mathcal{O}(c, m) = 1$ then $s_w = -1$ and $s_{w+l_v} = 1$
If $s'_w = -1, s'_{w+l_v} = 1$ and $\mathcal{O}(c, m) = 1$ then $s_w = 1$ and $s_{w+l_v} = -1$

If the oracle do not return 1, then Eve determines which coefficient is equal to 0 (Due by Proposition 5):

If $s'_w = 1, s'_{w+l_v} = 1$ and $\mathcal{O}(c, m) = 1$ then $s_w = -1$ and $s_{w+l_v} = 0$
or $s_w = 0$ and $s_{w+l_v} = -1$
  If $s'_w = -1, s'_{w+l_v} = 1$ and $\mathcal{O}(c, m) = 1$ then $s_w = 0$ and $s_{w+l_v} = -1$
  Else If $\mathcal{O}(c, m) = -1$ then $s_w = -1$ and $s_{w+l_v} = 0$
If $s'_w = -1, s'_{w+l_v} = -1$ and $\mathcal{O}(c, m) = 1$ then $s_w = 1$ and $s_{w+l_v} = 0$
or $s_w = 0$ and $s_{w+l_v} = 1$
  If $s'_w = 1, s'_{w+l_v} = 1$ and $\mathcal{O}(c, m) = 1$ then $s_w = 0$ and $s_{w+l_v} = 1$
  Else if $\mathcal{O}(c, m) = -1$ then $s_w = 1$ and $s_{w+l_v} = 0$
Otherwise $s_w = 0$ and $s_{w+l_v} = 0$

□

In our implementation [15] we choose $a = \frac{q}{7}$ for Proposition 4 and $a = \frac{q}{5}$ for Proposition 5.
To recover the entire key we need at most $8 \times n$ requests to the oracle due to the Theorem 2. But we can optimize this result and find the key with $8 \times l_v + 2 \times (n - l_v)$ requests to the oracle where $l_v = 400$ and $n = 1024$. In fact, when Eve does $8 \times l_v$ requests she had found 800 coefficients of the secret key. If she wants to retrieve the following bits she can perform the same attack as LAC-128 because $s_{i+800+l_v} = s_{i+800+l_v-n} = s_{i+176}$ and Eve had already found this coefficient.

3.3.3 Full version

The subroutine ***Compress*** removes the 4 lowers bits of c_2 and they are replaced by 4 zero-bit when the subroutine ***Decompress*** is applied at the beginning of the decryption process. So each coefficient of c_2 can be only equal to 16, 32, 64, 128 and any sum of these values.

For c_2 in our attack, we only consider the values $\frac{q}{7}$, $-\frac{q}{7}$, $\frac{q}{5}$, $-\frac{q}{5}$ and $\frac{q}{2}$. In our implementation [15] we approximate $\frac{q}{7} \approx 32$, $-\frac{q}{7} \approx 128+64+16 = 210$ or $-\frac{q}{7} \approx 128+64+32 = 224$ (we use two different values to compensate the approximation), $\frac{q}{5} \approx 16+32 = 48$, $-\frac{q}{5} \approx 128+64 = 192$ and $\frac{q}{2} \approx 128$. Propositions 4 and 5 are still verified and we still retrieve s with at most $8 \times n$ requests to the oracle by the Theorem 2.

4 Conclusion

In this paper, we show how to mount an attack on CPA version of LAC-KE when the same secret key is reused. We prove that this attack needs at most 8×1024 queries of key exchanges. This low number of queries to recover the secret confirmed the necessity to not reuse the same private key even for a very small number of key exchanges. One can compare this number with the key mismatch attack on NewHope in [9] that requires $882,794$ queries and the one on Kyber in [10] that requires $2,4 \times 1024$ queries. Hence, in the context of key reuse, LAC is much less resilient than NewHope but a little more resilient than Kyber. It is important to note that this situation is a misuse and thus, LAC is still believed to be safe when a fresh secret key is used for each exchange. (The same remark applies to NewHope and Kyber.)

References

[1] Peter W Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. volume 41, pages 303–332. SIAM, 1999.

[2] Dustin Moody. Post-quantum cryptography NIST's plan for the future. *URL: https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/pqcrypto-2016-presentation.pdf*, 2016.

[3] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. volume 60, page 43. ACM, 2013.

[4] Lu Xianhui, Liu Yamin, Jia Dingding, Xue Haiyang, He Jingnan, and Zhang Zhenfei. LAC: Lattice-based cryptosystems. *URL: https://csrc. nist. gov/Projects/Post-Quantum-Cryptography/Round-2-Submissions.*

[5] Erdem Alkim, Roberto Avanzi, Joppe Bos, Léo Ducas, Antonio de la Piedra, Thomas Pöppelmann, Peter Schwabe, and Douglas Stebila. NewHope. *URL: https://csrc. nist. gov/Projects/Post-Quantum-Cryptography/Round-2-Submissions.*

[6] Mike Hamburg. Post-quantum cryptography proposal: THREE-BEARS. *URL: https://csrc. nist. gov/Projects/Post-Quantum-Cryptography/Round-2-Submissions.*

[7] Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Kyber. *URL: https://csrc. nist. gov/Projects/Post-Quantum-Cryptography/Round-2-Submissions.*

[8] Aurélie Bauer, Henri Gilbert, Guénaël Renault, and Mélissa Rossi. Assessment of the key-reuse resilience of newhope. In *Cryptographers' Track at the RSA Conference*, pages 272–292. Springer, 2019.

[9] Chao Liu, Zhongxiang Zheng, and Guangnan Zou. Key reuse attack on newhope key exchange protocol. In *International Conference on Information Security and Cryptology*, pages 163–176. Springer, 2018.

[10] Yue Qin, Chi Cheng, and Jintai Ding. An efficient key mismatch attack on the NIST second round candidate kyber. *IEEE*, 2019.

[11] Alfred Menezes and Berkant Ustaoglu. On reusing ephemeral keys in Diffie-Hellman key agreement protocols. *IJACT*, 2(2):154–158, 2010.

[12] Scott R Fluhrer. Cryptanalysis of ring-LWE based key exchange with key share reuse. *IACR Cryptology ePrint Archive*, 2016:85, 2016.

[13] Jintai Ding, Scott Fluhrer, and Saraswathy Rv. Complete attack on RLWE key exchange with reused keys, without signal leakage. In *Australasian Conference on Information Security and Privacy*, pages 467–486. Springer, 2018.

[14] Ciprian Băetu, F Betül Durak, Loïs Huguenin-Dumittan, Abdullah Talayhan, and Serge Vaudenay. Misuse attacks on post-quantum cryptosystems. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 747–776. Springer, 2019.

[15] Simon Montoya. LAC attack. https://github.com/ayotnomis/LACAttack.

[16] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In *Annual International Cryptology Conference*, pages 537–554. Springer, 1999.

Appendix

Proof of the other case of Proposition 3

Proof. Suppose Alice receives $c = (c_1, c_2)$ then she:

1. Computes $\widehat{M} = c_2 - (c_1 s)_{2l_v}$
2. Compares $\frac{q}{4} < \frac{\widehat{M}[i]+\widehat{M}[i+l_v]}{2} < \frac{3q}{4}$ or $0 < \frac{|\widehat{M}[i]-\widehat{M}[i+l_v]|}{2} < \frac{q}{4}$ for $i = 0$ to $l_v - 1$ to define each coefficient of $\widehat{m}$
3. Retrieves m using ***BCHDecode*** algorithm on $\widehat{m}$

Let $c_1 = -ax^{n-w}$ and $s = s_0 + s_1 x^1 + \ldots + s_{n-1}x^{n-1}$ then

$$c_1 s = as_{n-w} + as_{n-w+1}x + \ldots + as_{n-1}x^w - as_0 x^{w+1} - \ldots - as_{n-w-1}x^{n-1}$$

We can represent $c_1 s$ as a vector $c_1 s = (as_{n-w}, \ldots, -as_{n-w-1})$. During the computation of $\widehat{M}$ we can have two cases:

- $w < 2l_v$ then:

$$\begin{aligned}\widehat{M} &= c_2 - (c_1 s)_{2l_v} \\ &= (\alpha_0 - as_{n-w}, \alpha_1 - as_{n-w+1}, \ldots, \alpha_w + as_0, \ldots, \alpha_{2l_v-1} + as_{2l_v-1})\end{aligned}$$

- $w \geq 2l_v$ then:

$$\begin{aligned}\widehat{M} &= c_2 - (c_1 s)_{2l_v} \\ &= (\alpha_0 - as_{n-w}, \alpha_1 - as_{n-w+1}, \ldots, \alpha_{2l_v-1} - as_{2l_v-1})\end{aligned}$$

Recall that since $s \leftarrow \psi_\sigma^n$, each of its coefficients belongs to $\{-1, 0, 1\}$. Let i be an integer such that $0 \leq i < l_v$ and $j \equiv n - w + i \mod n$.
Here we consider the case where $\widehat{M}[i] = \alpha_i + as_j$ and $\widehat{M}[i+l_v] = \alpha_{i+l_v} + as_{j+l_v}$ (the other case is $\widehat{M}[i] = \alpha_i - as_j$ and $\widehat{M}[i+l_v] = \alpha_{i+l_v} - as_{j+l_v}$).

If $\alpha_i = \frac{q}{2}$ one gets:

- If $s_j = s_{j+l_v}$ or $s_j + s_{j+l_v} = 1$ we are in the Case 1 described in Paragraph 3.3.1:

$$\alpha_i = \alpha_{i+l_v} = \frac{q}{2},$$

$$\frac{\alpha_i + as_j + \alpha_{i+l_v} + as_{j+l_v}}{2} = \begin{cases} \frac{q-2a}{2} & \text{if } s_j = s_{j+l_v} = 1 \\ \frac{q+2a}{2} & \text{if } s_j = s_{j+l_v} = -1 \\ \frac{q}{2} & \text{if } s_j = s_{j+l_v} = 0 \\ \frac{q+a}{2} & \text{if } s_j + s_{j+l_v} = 1 \end{cases}$$

These 3 values lie in $]\frac{q}{4}, \frac{3q}{4}[$ if $0 \leq a < \frac{q}{4}$.

- Otherwise we are in the Case 2 described in Paragraph 3.3.1

$$\alpha_i = \alpha_{i+l_v} = \frac{q}{2},$$
$$\frac{|(\alpha_i + as_j) - (\alpha_{i+l_v} + as_{j+l_v})|}{2} = \begin{cases} \frac{a}{2} & \text{if } s_j + s_{j+l_v} = -1 \\ a & \text{if } s_j = -1, s_{j+l_v} = 1 \\ & \text{or } s_j = 1, s_{j+l_v} = -1 \end{cases}$$

  These values lie in $[0, \frac{q}{4}[$ if $0 \leq a < \frac{q}{4}$.

Then for both cases, if $c_1 = -ax^{n-w}$ with $\alpha_i, \alpha_{i+l_v} = \frac{q}{2}$, we can ensure that we have a 1 after comparison.
If $\alpha_i = 0$ then we are in the Case 1 described in Paragraph 3.3.1:

$$\frac{\alpha_i + as_j + \alpha_{i+l_v} + as_{j+l_v}}{2} = \begin{cases} -a & \text{if } s_j = s_{j+l_v} = -1 \\ 0 & \text{if } s_j = -s_{j+l_v} \text{ or } s_j = s_{j+l_v} = 0 \\ a & \text{if } s_j = s_{j+l_v} = 1 \\ \pm\frac{a}{2} & \text{otherwise} \end{cases}$$

Then these 3 values do not lie in $]\frac{q}{4}, \frac{3q}{4}[$ for $0 \leq a < \frac{q}{4}$.

□