Attack on LAC Key Exchange in Misuse Situation

Aurélien Greuet, Simon Montoya, Guénaël Renault

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
firstname.lastname@idemia.com
[2] LIX, INRIA, CNRS, École Polytechnique, Institut Polytechnique de Paris, France
firstname.lastname@lix.polytechnique.fr

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 [18], led the National Institute of Standards and Technology (NIST), in 2016, to begin a call for post-quantum safe public-key cryptography [15]. 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) [12] 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 [19,1,9,2]. 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 [19], 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. In constrained environments it's interesting to determine the impact of key caching to evaluate the requirement of random generation. Furthermore, the specification of CCA version of LAC uses a static secret key due to security provided by the Fujisaki Okamoto (FO) transformation [7]. However, as shown in [16,4] without a secure implementation of FO transformation, a physical attack can bypass security provided by FO and modifies a CCA version to a CPA one with a static secret key.
Our study is inspired by previous works in [4,11,17], which evaluate the resilience in a misuse context offered by two other NIST KEM candidates. Here we propose 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 Menezes and Ustaoglu [13] 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 [6,5]. 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 [3], Baetu *et al.* proposed a generic attack for several algorithms using the same structure called meta-algorithm. However, most of the algorithms attacked in [3] did not pass the first round of the submission, except Frodo-640 and NewHope512. However in [10], Huguenin-Dumittan *et al.* pursue the work of generic attack for round 2 candidates. The security of NewHope1024 CPA algorithm in this misuse scenario is analyzed by Bauer *et al.* in [4] and an improvement is proposed in [11]. More recently, in the same context, an attack on Kyber CPA KEM is proposed by Ding *et al.* [17].

In [8], Guo *et al.* presented an attack against the CCA version of LAC. This attack is theoretically stronger than ours since it does not rely on a misuse hypothesis but it requires 2^{162} pre-computations that cannot be achieved in practice.

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 [4].

Since LAC uses encoding and compression functions different from a classical RLWE scheme, Fluhrer's attack [6] 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 [4,11,17]. A recent independent work in [10] attacks several round 2 candidates using the generic structure of these schemes. Their attack is applied to the first security level of LAC but is focused on the theoretical aspect. Our work complete this work by bringing a practical aspect and an extension to the others security levels.

The main idea of these attacks is to send forged ciphertexts to a victim, ensuring that its decryption will leak partial information of his 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 to provide leaks 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 [19] with parameters described in Section 2.2. The code of our attack is available in [14].

Organization

In Section 2, we introduce some notations, 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 [4] and the attack for the first two security levels. Finally, in Section 4 we adapt the attack to the higher security level.

2 Preliminaries

2.1 Notation

Ring definition. 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 being the polynomial ring $R_q = \mathbb{Z}_q[x]/(x^n + 1)$.

Polynomial. 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. In the following, the notation $(a)_{l_v}$ $(l_v \in \mathbb{N})$, where a is a vector (or a polynomial) of dimension $n > l_v$, means we keep the first l_v coordinates of a.

Message space. 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 integer values.

Random distribution. Let ψ_σ be the centred binomial distribution on the set $\{-1,0,1\}$. We denote the centred binomial distribution for n independent coordinates by ψ_σ^n i.e. for a vector a of dimension n each coefficient is sampled with the centred 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 $\textbf{\textit{Samp}}(D, seed)$ an algorithm which samples a random variable according to a distribution D with a given seed.

Error correction code. 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 (p_k, s_k)
$seed_a \leftarrow U(\mathcal{S})$
$a \leftarrow \boldsymbol{Samp}(U(R_q), seed_a)) \in R_q$
$s \leftarrow \psi_\sigma^n$
$e \leftarrow \psi_\sigma^n$
$b \leftarrow a \times s + e \in R_q$
return $(p_k, s_k) = ((seed_a, b), s)$

Algorithm 2
$\boldsymbol{CPA.Encrypt}(p_k, m, seed)$

Ensure: Ciphertext $c = (c_1, c_2)$
$(seed_a, b) \leftarrow p_k$
$a \leftarrow \boldsymbol{Samp}(U(R_q), seed_a) \in R_q$
$\widehat{m} \leftarrow \boldsymbol{BCHEncode}(m) \in \{0,1\}^{l_v}$
$r \leftarrow \boldsymbol{Samp}(\psi_\sigma^n, seed)$
$e_1 \leftarrow \boldsymbol{Samp}(\psi_\sigma^n, seed)$
$e_2 \leftarrow \boldsymbol{Samp}(\psi_\sigma^{l_v}, seed)$
$c_1 \leftarrow ar + e_1 \in R_q$
$c_2 \leftarrow (br)_{l_v} + e_2 + \lfloor \frac{q}{2} \rceil \widehat{m} \in \mathbb{Z}_q^{l_v}$
if LAC-256
  $c_2 \leftarrow c_2 || c_2$ //D2 encoding
end if
$c_2 \leftarrow \boldsymbol{Compress}(c_2)$
return $c = (c_1, c_2)$

Algorithm 3
$\boldsymbol{CPA.Decrypt}(s_k, c = (c_1, c_2))$

Ensure: Plaintext m
$c_2 \leftarrow \boldsymbol{Decompress}(c_2)$
$\widehat{M} \leftarrow c_2 - (c_1 s_k)_{l_v} \in \mathbb{Z}_q^{l_v}$
for $i = 0$ to $l_v - 1$ **do**
  if $\frac{q}{4} \leq \widehat{M}_i < \frac{3q}{4}$ **then**
    $\widehat{m}_i \leftarrow 1$
  else
    $\widehat{m}_i \leftarrow 0$
  end if
end for
$m \leftarrow \boldsymbol{BCHDecode}(\widehat{m})$
return m

Algorithm 4
$\boldsymbol{CPA.Decrypt256}(s_k, c = (c_1, c_2))$

Ensure: Plaintext m
$c_2 \leftarrow \boldsymbol{Decompress}(c_2)$
$\widehat{M} \leftarrow c_2 - (c_1 s_k)_{2l_v} \in \mathbb{Z}_q^{2l_v}$
for $i = 0$ to $l_v - 1$ **do** //D2 Decoding
  $tmp_1, tmp_2 := \widehat{M}[i], \widehat{M}[i + l_v]$
  if $tmp_1 < \frac{q}{2}$
    $tmp_1 \leftarrow q - tmp_1$
  else if $tmp_2 < \frac{q}{2}$
    $tmp_2 \leftarrow q - tmp_2$
  end if
  if $tmp_1 + tmp_2 - q < \frac{q}{2}$
    $\widehat{m}_i \leftarrow 1$
  else
    $\widehat{m}_i \leftarrow 0$
  end if
end for
$m \leftarrow \boldsymbol{BCHDecode}(\widehat{m})$
return m

Subroutines

BCHEncode* and *BCHDecode*.** The function ***BCHEncode takes as input a message m of length l_m, pads 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.

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 pads 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 $l_v = l_m + (n' - k)$.

LAC Key Exchange

We describe the LAC Key Exchange introduced in [19], 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 by introducing the oracle defined in [4].

3.1 Attack Model

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 that Eve has access to a session 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 session key guess μ as input and outputs:*

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

This oracle can also be used by Bob during an honest key exchange with Alice, when he verifies the match between his session 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 four mains steps:

- Choose a session key μ.
- Construct c_1 such that some coefficients of the secret key are exposed.
- Construct c_2 depending of μ 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.

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. To do so, we need to forge c such that the impact of the secret key during the decryption is under our control.

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 will obtain 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

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

and the polynomial $c_1 s$ can be represented as the vector $(as_w, \ldots, -as_{w-1})$
During the computation of $\widehat{M}$, two cases are possible:
•$w < l_v$ then $\widehat{M} = c_2 - (c_1 s)_{l_v} = (\alpha_0 - as_w, \ldots, \alpha_w + as_0, \ldots, \alpha_{l_v-1} + as_{w+l_v-1})$
•$w \geq l_v$ then $\widehat{M} = c_2 - (c_1 s)_{l_v} = (\alpha_0 - as_w, \alpha_1 - as_{w+1}, \ldots, \alpha_{l_v-1} - as_{w+l_v-1})$
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:

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

If $\alpha_i = 0$ one gets:

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

In the first case, the three possible values for $\alpha_i \mp as_j$ lie in $\left[\frac{q}{4}, \frac{3q}{4}\right[$ if $0 \leq a \leq \frac{q}{4}$. In the case $\alpha_i = 0$, the three possible values do not lie in $\left[\frac{q}{4}, \frac{3q}{4}\right[$ when $0 \leq a < \frac{q}{4}$ 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 will obtain on the first l_v coordinates of $\widehat{m}$. Then, Eve can deduce, by applying BCH decoding, what Alice obtains at the end of the decryption procedure.

The next example explains how one can use Proposition 1.

Example 1. Suppose that Eve wants that Alice will obtain, 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:

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

From c Alice first 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 $m =$ ***BCHDecode***$(1, 0, 1, 1, 0, \ldots, 0)$.

With Proposition 1 we construct a ciphertext such that the secret key has no impact during decryption. Now Eve needs to construct forged ciphertexts that allow a key guessing strategy in order to retrieve the secret key. Thus, we need that the secret key has an impact during decryption if and only if we did a good key guess.

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 $c = (c_1, c_2)$ as given in Proposition 1 and modify the first coordinate of c_2 such that:*

- $c_2 = (as'_w, \alpha_1, \ldots, \alpha_{l_v - 1})$ *with* $\frac{q}{8} < a < \frac{q}{4}$.

Then she can verify her key guess from the plaintext computed by Alice from c.

Proof. Suppose that Eve wants to retrieve the w-th coefficient of s. When Alice will decipher Eve ciphertext she first 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 will obtain for every coefficient different from her guess s'_w. Let see what happens with this coefficient by analysing $as'_w - as_w$.

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

Let $\frac{q}{8} < a < \frac{q}{4}$ then $\frac{q}{4} < 2a < \frac{q}{2}$ and $-2a = q - 2a$ satisfies $\frac{q}{2} < q - 2a < \frac{3q}{4}$ Then with $a \in]\frac{q}{8}, \frac{q}{4}[$. The key guess is good (resp. wrong) when a 1 (resp. 0) is returned at the first coordinate of $\widehat{m}$. Hence Eve can effectively determines what Alice obtained by applying ***BCHDecode*** algorithm to $\widehat{m}$ and thus deterministically verifies her key guess from $\widehat{m}$.

Proposition 2 ensures that if Eve guessed the good key then Alice will obtains $m = \boldsymbol{BCHDecode}(1, \ldots)$. Otherwise, she will obtain $m = \boldsymbol{BCHDecode}(0, \ldots)$. Computational details are given 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:

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

As in Example 1, Alice first computes $\widehat{M} = c_2 - (c_1 s)_{l_v} = (\frac{q}{5}s'_0 - \frac{q}{5}s_0, -\frac{q}{5}s_1, \frac{q}{2} - \frac{q}{5}s_2, \frac{q}{2} - \frac{q}{5}s_3, -\frac{q}{5}s_4, \ldots, -\frac{q}{5}s_{l_v})$ where s_i belongs to $\{-1, 0, 1\}$. Then, Alice compares each coefficients of $\widehat{M}$ to $\frac{q}{4}$ and $\frac{3q}{4}$. She obtains (see proof of Proposition 2):

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

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

Proposition 2 already gives interesting information to Eve but it is not enough to mount an attack since Eve needs a way to verify if Alice obtains:

- either $m = \boldsymbol{BCHDecode}(1, 0, 1, 1, 0, \ldots, 0)$
- or $m = \boldsymbol{BCHDecode}(0, 0, 1, 1, 0, \ldots, 0)$

without knowing m. Moreover, most of the time $\boldsymbol{BCHDecode}(1, 0, 1, 1, 0, \ldots, 0)$ will not differ from $\boldsymbol{BCHDecode}(0, 0, 1, 1, 0, \ldots, 0)$.

To overcome these issues we need to instantiate precisely the oracle given in Definition 1 using Proposition 2.

Instantiation of the Oracle. The oracle defined in Definition 1 gives information about the success of a key session establishment between Alice and Bob. Eve can use such an oracle with the help of Proposition 2 and the BCH code decryption failure to overcome issues mentioned above.

In the sequel, we show how Eve can practically mount an attack by forging specific inputs to this oracle and deduce information on Alice's secret key. The following theorem and its proof detail this construction then Algorithm 5 and Algorithm 6 formally describe the attack.

Theorem 1. *Let $s'_w \in \{-1, 1\}$ be the guessed value of s_w $(0 \leq w < n)$ done by Eve. If Eve takes a session key $\mu_{s'_w}$ then she can forge $c_{s'_w} = (c_1, c_2)$ depending of $\mu_{s'_w}$ by using properties given in Proposition 2 such that by calling $\mathcal{O}(c_{s'_w}, \mu_{s'_w})$ with $s'_w \in \{-1, 1\}$, she retrieves the w-th coefficient of s. In consequence, Eve needs at most 2 calls to the oracle in order to retrieve a coefficient of Alice's secret key.*

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 $\frac{d-1}{2}$ errors or $cdword$ with $\frac{d}{2}$ 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 provides a wrong key guess. At this point, Eve's session key is $sess_E = H(pk, m)$ and Alice's session key $sess_A$ depends on Eve's key guess. Eve can verify whether she did the correct key guess with the oracle as follow:

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

Else If $s'_w = -1$ and $\mathcal{O}(c, sess_E) = 1$ then $s_w = 1$ and $sess_A = sess_E$

Otherwise $s_w = 0$

Algorithm 5 and Algorithm 6 are based on the construction described in the proof of Theorem 1. Here, we fix the constant a to $\frac{q}{7}$ in the construction of c_1.

Algorithm 5 forge(hyp,bit)

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

$c_1 := -\frac{q}{7}x^{n-bit}$
$m :=$ [0 : **for** $i := 0$ 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 recoverOneBit(bit)

Ensure: A bit of s

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

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

Full version

The subroutine ***Compress*** removes the 4 lowers bits of each coeff of c_2. 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 [14] 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.

In comparison of the recent work of Huguenin-Dumittan *et al.* in [10], our upper-bound for LAC128 is 2 times less than theirs. Indeed, they need at most 2^{11} queries to retrieve the entire secret key, while we need at most 2^{10} queries.

Implementation results

We have developed a `C` implementation of the attack (see [14]). To assess its efficiency we use the reference code of LAC [19] as a target. In the following, we present practical results on the average of 1000 attacks launched on 1000 random secret keys for LAC-128 and LAC-192. Timing results have been evaluated on core i5-8350U at 1.90GHz.

	Nb of coeff of sk	Average oracle requests	Average time
LAC-128	512	896	$2,94$ ms
LAC-192	1024	1920	$15,53$ ms

The size of LAC-192 secret key is 2 times larger than LAC-128 one, but the number of required request to retrieve sk is more than 2 times larger. This is due to a different probability distribution between these two levels of security.
In average we need $1,75 \times 512$ oracle requests for LAC-128 and $1,875 \times 1024$ requests for LAC-192. For both cases, the practical result is less than the upper bound of $2 \times n$ where $n = 512$ or 1024.

4 Attack on LAC-256-KE

4.1 Attack on LAC-256-KE

Since LAC-256 encryption uses $D2$ encoding , the decryption procedure is slightly different. Let $c = (c_1, c_2)$, $D2$ encoding duplicates the coordinate of c_2: $c_2 = (c_2||c_2)$. The use of this encoding allows to decrease decoding errors.

In Attack on LAC-128/192 we forged c_1 as a monomial to avoid linear combination between coefficients of s during computation of $c_1 s$. This allows to do key guess on only one coefficient of s. But, despite the use of a monomial for c_1, $D2$ encoding ensures that each coefficient of $c_1 s$ is a linear combination of at least 2 coefficients of s. It implies that we need to do key guesses on two coefficients of s. In this section, we adapt our previous attack to allow to do two key guesses rather than one. The attack procedure is the same as previously:

- Choose a session key μ.
- Construct c_1 such that some coefficients of the secret key are exposed.
- Construct c_2 depending of μ 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.

For the sake of clarity all proposition proofs are in Appendix.

CPA.Decrypt256 description

The first step of the decryption it's to compute $\widehat{M} = c_2 - (c_1 s)_{2l_v}$ as previously. However the comparison is different for LAC-256. 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*** checks whether: $\frac{\widehat{M}[i]+\widehat{M}[i+l_v]}{2} \in]\frac{q}{4}, \frac{3q}{4}[$

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*** checks whether $\frac{|\widehat{M}[i]-\widehat{M}[i+l_v]|}{2} \in]0, \frac{q}{4}[$

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

4.2 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 $0 \leq w < (n - l_v)$ 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.

Example 3. Suppose that Eve wants that Alice obtains, after decryption, the plaintext $m = \textbf{\textit{BCHDecode}}(1, 1, 0, 1, 0, \ldots, 0)$. Eve forges $c = (c_1, c_2)$ such that:

- $c_1 = -\frac{q}{5}x^n = \frac{q}{5}$ on R_q.
- $c_2 = (\frac{q}{2}, \frac{q}{2}, 0, \frac{q}{2}, 0 \ldots, 0 || \frac{q}{2}, \frac{q}{2}, 0, \frac{q}{2}, 0 \ldots, 0)$. The symbol $||$ delimit the l_v first part to the l_v second part of c_2 (we duplicate c_2 due to $D2$ encoding in Algorithm 2). The two parts are symmetric .

When Alice deciphers c, she computes $\widehat{M} = c_2 - (c_1 s)_{2l_v}$ and uses the comparison procedure describes in Algorithm 4 to obtain $\widehat{m}$ of length l_v. If c_1 and c_2 are constructed according to Proposition 3, then (cf Proof 5):

- If $c_2[i] = c_2[i + l_v] = \frac{q}{2}$ then $\widehat{m}[i] = 1$
- If $c_2[i] = c_2[i + l_v] = 0$ then $\widehat{m}[i] = 0$

Then, with our $c_2 = (\frac{q}{2}, \frac{q}{2}, 0, \frac{q}{2}, 0 \ldots, 0 || \frac{q}{2}, \frac{q}{2}, 0, \frac{q}{2}, 0 \ldots, 0)$ Alice obtains $\widehat{m} = (1, 1, 0, 1, 0, \ldots, 0)$. Thus, Alice retrieves $m = \boldsymbol{BCHDecode}(1, 1, 0, 1, 0, \ldots, 0)$.

So Eve can choose $a < \frac{q}{4}$ and $\alpha_i = \frac{q}{2}$ or 0 to know what Alice obtains on the l_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 allow to verify her key guesses.

Proposition 4. *Let s'_w and s'_{w+l_v} be guesses done by Eve on the w-th and $w+l_v$ coefficients of the secret key s. Assume $s_w, s_{w+l_v} = 1$ or -1. If Eve forges $c = (c_1, c_2)$ as given in Proposition 3 and modify the first and l_v-th coordinates of c_2 such that:*

- $c_2 = (as'_w, \alpha_1, \ldots, \alpha_{l_v-1}, as'_{w+l_v}, \ldots, \alpha_{l_{v-1}})$ *with* $\frac{q}{8} < a < \frac{q}{4}$.

Then she can verify her key guesses from the plaintext computed by Alice from c.

Example 4. Suppose that Eve wants to learn information about the first and the l_v-th bit of Alice's secret key. Eve forges $c = (c_1, c_2)$ such that:

- $c_1 = -\frac{q}{5} x^n = \frac{q}{5}$ on R_q.
- $c_2 = (\frac{q}{5} s'_0, \frac{q}{2}, 0, \frac{q}{2}, 0 \ldots, 0 || \frac{q}{5} s'_{l_v}, \frac{q}{2}, 0, \frac{q}{2}, 0 \ldots, 0)$ where s'_0 and s'_{l_v} are key guesses

When Alice deciphers c she computes $\widehat{M} = c_2 - (c_1 s)_{2l_v}$ and uses the comparison procedure describes in Algorithm 4 to obtain $\widehat{m}$ of length l_v. If c_1, c_2, s'_0 and s'_{l_v} are constructed according to Proposition 4, then (cf Proof 5):

- If $s'_0 = -s_0$ and $s'_{l_v} = -s_{l_v}$ then $\widehat{m}[0] = 1$
- Else $\widehat{m}[0] = 0$
- The value of the others coefficients of $\widehat{m}$ are determined as in the previous example

If Eve does correct key guesses then Alice obtains $\widehat{m} = (1, 1, 0, 1, 0, \ldots, 0)$. Otherwise, Alice obtains $\widehat{m} = (0, 1, 0, 1, 0, \ldots, 0)$

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. *Let s'_w and s'_{w+l_v} be guesses done by Eve on the w-th and $w+l_v$ coefficients of the secret key s. Assume $s_w = 0$ or $s_{w+l_v} = 0$. If Eve forges $c = (c_1, c_2)$ as given in Proposition 3 and modify the first and l_v-th coordinates of c_2 such that:*

- $c_2 = (as'_w, \alpha_1, \ldots, \alpha_{l_v-1}, as'_{w+l_v}, \ldots, \alpha_{l_{v-1}})$ *with* $\frac{q}{6} < a < \frac{q}{4}$.

Then she can verify her key guesses from the plaintext computed by Alice from c.

Proposition 5 works like Proposition 4 but for the case where one of the two targeted coefficient is equal to 0. However, as previously, Proposition 4 and Proposition 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 Proposition 4 and Proposition 5, the BCH code decryption failure and the oracle to overcome these issues.

Theorem 2. *Let $s'_w, s'_{w+l_v} \in \{-1, 1\}$ be the guessed values of s_w and s_{w+l_v} done by Eve. If Eve takes a session key $\mu_{s'_w, s'_{w+l_v}}$ then she can forge $c_{s'_w, s'_{w+l_v}} = (c_1, c_2)$ depending of $\mu_{s'_w, s'_{w+l_v}}$ by using properties given in Proposition 2 such that by calling $\mathcal{O}(c_{s'_w, s'_{w+l_v}}, \mu_{s'_w, s'_{w+l_v}})$ with $s'_w, s'_{w+l_v} \in \{-1, 0, 1\}$, she retrieves the w-th and $w + l_v$-th coefficients of s. In consequence, Eve needs at most $8 \times (n - l_v)$ calls to the oracle in order to retrieve two coefficients of Alice's secret key.*

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 a wrong key guess. Since at most $\frac{d-1}{2}$ errors can be corrected, a decoding errors occurs.
According to Proposition 4 and Proposition 5, Eve can monitor Alice's decryption procedure if she does the good key guess.
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 at the first and l_v-th coefficient of c_2: $c_2[0] = as'_w$ and $c_2[l_v] = as'_{w+l_v}$ 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 a wrong key guess. At this point, Eve's session key is $sess_E = H(pk, m)$ and Alice's session key $sess_A$ depends on Eve's key guesses. Eve can verify if she did a good key guess with the oracle.
First Eve determines if s_w and s_{w+l_v} are different from 0 (see Proposition 4):

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

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

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

Eve can applies this procedure for $0 \leq w < (n - l_v)$ to retrieve the entire secret key.

To recover the entire key we need at most $8 \times (n - l_v)$ requests to the oracle due to Theorem 2, where $l_v = 400$ and $n = 1024$.

Full version

The subroutine ***Compress*** removes the 4 lowers bits of each coeff of c_2. 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 choose $a \approx \frac{q}{7}$ for Proposition 4 and $a \approx \frac{q}{5}$ for Proposition 5. Then, we only consider the values $\frac{q}{7}$, $-\frac{q}{7}$, $\frac{q}{5}$, $-\frac{q}{5}$ and $\frac{q}{2}$. In our implementation [14] 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 $\frac{q}{2} \approx 128$. Proposition 4 and Proposition 5 are still verified and we still retrieve s with at most $8 \times (n - l_v)$ requests to the oracle by the Theorem 2.

Implementation results

We assess our attack implementation [14] plug in the reference code of LAC. Following results are the average of 1000 attacks launched on 1000 random secret keys for LAC-256. Timing results have been evaluated on core i5-8350U at 1.90GHz.

	Nb of coeff of sk	Average oracle requests	Average time
LAC-256	1024	3355	30, 31 ms

In average we need $5,4 \times (1024 - 400)$ oracle requests that is much less than the upper bound of $8 \times (n - l_v)$ requests

5 Conclusion

In this paper, we show how to mount an attack on CPA version of LAC-KE when the same secret key is reused. Moreover, on constrained environment this attack can be applied on the CCA version by applying physical attack on the Fujisaki-Okamoto transformation as shown in [7,16]. 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 [11] that requires $882,794$ queries and the one on Kyber in [17] that requires $2,4 \times 1024$ queries. Hence, in the context of key reuse, LAC-256 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. Alkim, E., Avanzi, R., Bos, J., Ducas, L., de la Piedra, A., Pöppelmann, T., Schwabe, P., Stebila, D.: NewHope (2019), available at https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Round-2-Submissions
2. Avanzi, R., Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., M. Schanck, J., Schwabe, P., Seiler, G., Stehlé, D.: CRYSTALS-Kyber (2019), available at https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Round-2-Submissions
3. Băetu, C., Durak, F.B., Huguenin-Dumittan, L., Talayhan, A., Vaudenay, S.: Misuse Attacks on Post-Quantum Cryptosystems. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 747–776. Springer (2019)
4. Bauer, A., Gilbert, H., Renault, G., Rossi, M.: Assessment of the Key-Reuse Resilience of NewHope. In: Cryptographers' Track at the RSA Conference. pp. 272–292. Springer (2019)
5. Ding, J., Fluhrer, S., Rv, S.: Complete Attack on RLWE Key Exchange with Reused Keys, Without Signal Leakage. In: Australasian Conference on Information Security and Privacy. pp. 467–486. Springer (2018)
6. Fluhrer, S.: Cryptanalysis of ring-LWE based key exchange with key share reuse. Cryptology ePrint Archive, Report 2016/085 (2016), available at https://eprint.iacr.org/2016/085
7. Fujisaki, E., Okamoto, T.: Secure Integration of Asymmetric and Symmetric Encryption Schemes. In: Annual International Cryptology Conference. pp. 537–554. Springer (1999)
8. Guo, Q., Johansson, T., Yang, J.: A Novel CCA Attack using Decryption Errors against LAC. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 82–111. Springer (2019)
9. Hamburg, M.: Post-Quantum Cryptography Proposal: THREEBEARS (2019), available at https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Round-2-Submissions
10. Huguenin-Dumittan, L., Vaudenay, S.: Classical Misuse Attacks on NIST Round 2 PQC. In: International Conference on Applied Cryptography and Network Security. pp. 208–227. Springer (2020)

11. Liu, C., Zheng, Z., Zou, G.: Key Reuse Attack on NewHope Key Exchange Protocol. In: International Conference on Information Security and Cryptology. pp. 163–176. Springer (2018)
12. Lyubashevsky, V., Peikert, C., Regev, O.: On Ideal Lattices and Learning with Errors over Rings. Journal of the ACM (JACM) **60**(6), 43 (2013)
13. Menezes, A., Ustaoglu, B.: On Reusing Ephemeral Keys in Diffie-Hellman Key Agreement Protocols. IJACT **2**(2), 154–158 (2010)
14. Montoya, S.: LAC attack, https://github.com/ayotnomis/LACAttack
15. Moody, D.: Post-Quantum Cryptography NIST's plan for the future (2016), available at https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/pqcrypto-2016-presentation.pdf
16. Oder, T., Schneider, T., Pöppelmann, T., Güneysu, T.: Practical CCA2-Secure and Masked Ring-LWE Implementation. Cryptology ePrint Archive, Report 2016/1109 (2016), https://eprint.iacr.org/2016/1109
17. Qin, Y., Cheng, C., Ding, J.: An Efficient Key Mismatch Attack on the NIST Second Round Candidate Kyber. IEEE (2019)
18. Shor, P.W.: Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. SIAM review **41**(2), 303–332 (1999)
19. Xianhui, L., Yamin, L., Dingding, J., Haiyang, X., Jingnan, H., Zhenfei, Z.: LAC: Lattice-based Cryptosystems (2019), available at https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Round-2-Submissions

Appendix

Proof of Proposition 3

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 $c_1 s = as_w + as_{w+1}x + cldots + as_{n-1}x^w - as_0x^{w-1} - \cdot - as_{w-1}x^{n-1}$.

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

- $w < 2l_v$ then $\widehat{M} = c_2 - (c_1 s)_{2l_v} = (\alpha_0 - as_w, \alpha_1 - as_{w+1}, \ldots, \alpha_w + as_0, \ldots, \alpha_{2l_v-1} + as_{(2l_v-1+w \mod n)})$
- $w \geq 2l_v$ then $\widehat{M} = c_2 - (c_1 s)_{2l_v} = (\alpha_0 - as_w, \alpha_1 - as_{w+1}, \ldots, \alpha_{2l_v-1} - as_{(2l_v-1+w \mod n)})$

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 $j \equiv i + w \mod n$. For decryption there are the three following cases. (We cannot have the case where $\widehat{M}[i] = \alpha_i + as_j$ and $\widehat{M}[i + l_v] = \alpha_{i+l_v} - as_{j+l_v}$ because that implies $j + l_v \leq w + l_v$ and $w < j$ with $l_v > 0$ and $j \geq 0$.)

1. $\widehat{M}[i] = \alpha_i - as_j$ *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 4.1, where $\alpha_i - as_j = \widehat{M}[i]$. Then

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

  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 4.1, 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 $[0, \frac{q}{4}[$ 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 4.1, 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 $]\frac{q}{4}, \frac{3q}{4}[$ for $0 \leq a < \frac{q}{4}$.

2. $\widehat{M}[i] = \alpha_i + as_j$ *and* $\widehat{M}[i+l_v] = \alpha_{i+l_v} + as_{j+l_v}$. The proof is the same as above. We give here the different decryption cases:

- If $\alpha_i = \frac{q}{2}$ then two cases are possible: if $s_j = s_{j+l_v}$ or $s_j + s_{j+l_v} = 1$ then we are in the decryption Case 1 otherwise in the Case 2.
- If $\alpha_i = 0$ then we are in the decryption Case 1.

3. $\widehat{M}[i] = \alpha_i - as_j$ *and* $\widehat{M}[i+l_v] = \alpha_{i+l_v} + as_{j+l_v}$. The proof is the same as above. We give here the different decryption cases:

- If $\alpha_i = \frac{q}{2}$ then two cases are possible: if $s_j = -s_{j+l_v}$ or $s_j = 0, s_{j+l_v} = 1$ or $s_j = -1, s_{j+l_v} = 0$ then we are in the decryption Case 1, otherwise in the Case 2.
- If $\alpha_i = 0$ then we are in the decryption Case 1.

Proof of Proposition 4

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. Let $\widehat{M} = c_2 - (c_1 s)_{2l_v}$, due to $0 \leq w < (n - l_v)$ the only case to consider is $\widehat{M}[0] = as'_w - as_w$ and $\widehat{M}[l_v] = as'_{w+l_v} - as_{w+l_v}$.

Let $s'_w = s'_w = 1$ and $\frac{q}{8} < a < \frac{q}{4}$, so we are in the Case 1 described in Paragraph 4.1. Let see what happens with $\frac{\widehat{M}_0+\widehat{M}_{l_v}}{2} = \frac{as'_w - as_w + as'_{w+l_v} - as_{w+l_v}}{2}$:

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

Then only the case $\frac{a-as_w+a-as_{w+l_v}}{2} = \frac{3a}{2}$ can put a 1 to $\widehat{m}_0$ 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 $\widehat{m_0} = 1$ then $s_w = s_{w+l_v} = -1$
- If $s'_w = s'_{w+l_v} = -1$ and $\widehat{m_0} = 1$ then $s_w = s_{w+l_v} = 1$
- If $s'_w = 1$, $s'_{w+l_v} = -1$ and $\widehat{m_0} = 1$ then $s_w = -1$ and $s_{w+l_v} = 1$
- If $s'_w = -1$, $s'_{w+l_v} = 1$ and $\widehat{m_0} = 1$ then $s_w = 1$ and $s_{w+l_v} = -1$

Proof of Proposition 5

Proof. Assume that Eve wants to retrieve the w-th and $(w + l_v)$-th coefficients of s. As Proof 5 the only case to consider is $\widehat{M}[0] = as'_w - as_w$ and $\widehat{M}[l_v] = as'_{w+l_v} - as_{w+l_v}$. Suppose $\frac{q}{6} < a < \frac{q}{4}$, $s'_w = 1$ and $s'_{w+l_v} = 1$. Let see what happens with $\frac{\widehat{M}_0+\widehat{M}_{l_v}}{2} = \frac{as'_w - as_w + as'_{w+l_v} - as_{w+l_v}}{2}$ (Case 1 described in Paragraph 4.1):

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

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 4.1. Let see what happens:

- If $s_w = -1$ and $s_{w+l_v} = 0$ we are in Case 1 4.1 thus $\frac{q}{4} < a < \frac{3q}{4}$.
- If $s_w = 0$ and $s_{w+l_v} = -1$ we are in Case 2 4.1 thus $0 < \frac{|-a-2a|}{2} < \frac{q}{4}$ which 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 $\widehat{m_0} = 1$ 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 $\widehat{m_0} = 1$ 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 $\widehat{m_0} = 1$ 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 $\widehat{m_0} = 1$ then $s_w = 0$, $s_{w+l_v} = 1$ else $s_w = 1$, $s_{w+l_v} = 0$