Modified versions of ML-KEM based on Brazilian cryptographic resources

Vinícius Lagrota[1], **Beatriz L. Azevedo**[2], **Mateus de L. Filomeno**[2], **Moisés V. Ribeiro**[2]

[1] Research and Development Center for Communication Security (CEPESC)
Brasília, DF – Brazil.

[2]Electric Engineering Department
Federal University of Juiz de Fora (UFJF) – Juiz de Fora, MG – Brazil

`{vinicius.lagrota, mateus.lima, mribeiro}@engenharia.ufjf.br, beatriz.azevedo@estudante.ufjf.br`

Abstract. *This paper outlines the Module-Lattice-Based Key-Encapsulation Mechanism Standard (ML-KEM) based on Brazilian cryptography to safeguard sensitive information. In this sense, it details two Brazilian cryptographic algorithms, Forró and Xote, and discusses the modifications in the ML-KEM to enable their use as symmetric primitives. Relying on experimental results regarding execution time, we show that ML-KEM with Xote surpasses ML-KEM with SHAKE or Forró while maintaining an equivalent level of security in tasks such as key pair generation, encapsulation, and decapsulation.*

1. Introduction

Cryptography is a cornerstone in safeguarding national sovereignty, sensitive information, and critical infrastructure against cyber threats and espionage activities, which is crucial for maintaining trust and security in an interconnected world. By ensuring the confidentiality, integrity, and authenticity of government communications, cryptographic measures protect against foreign actions that may introduce vulnerabilities and aid compliance with regulatory standards and international agreements, bolstering trust among citizens and stakeholders in handling sensitive data and communications. Strong cryptographic schemes are vital for governments to maintain national security, protect digital sovereignty, and cultivate a secure atmosphere for citizens, businesses, and governmental activities in the face of evolving cyber threats.

Cryptography, on the other hand, faces significant challenges with the rapid advancement of quantum computing technology. Once a cryptographically relevant quantum computer (CRQC) emerges, Shor's algorithm [Shor 1994] will render public-key cryptography vulnerable due to its ability to efficiently solve problems like factoring large numbers and discrete logarithms. In anticipation of the quantum era, National Institute for Standards and Technology (NIST) started a competition in 2017 to standardize Public Key Encryption (PKE)/Key Encapsulation Mechanism (KEM) and digital signature algorithms resistant to quantum (and classical) computer attacks, i.e., the post-quantum cryptography (PQC). So far, four algorithms have been standardized or selected to be. Among them, CRYSTALS-Kyber [Avanzi et al. 2021], which uses Secure Hash Algorithm and Keccak (SHAKE) and Advanced Encryption Standard (AES) as symmetric primitives, is the only KEM. It has been standardized under the name FIPS-203 Module-Lattice-Based Key-Encapsulation Mechanism Standard (ML-KEM) [National Institute of Standards and Technology 2024].

As the ML-KEM underwent standardization by NIST, it garnered significant attention and research. As exemplified by AVX2 in its NIST submission, software optimizations have been explored, focusing on utilizing fast devices and intrinsic functions. For instance, [Wan et al. 2022] introduced a GPU-based implementation, whereas [Nguyen and Gaj 2021] proposed an ARMv8-targeted implementation leveraging NEON-based special instructions. Furthermore, hardware optimization strategies have also been investigated, including reconfigurable hardware implementations with side-channel protection [Jati et al. 2024], compact pure hardware implementations prioritizing performance and area efficiency [Xing and Li 2021], and the feasibility of ML-KEM on hardware-constrained devices [Costa et al. 2022], suggesting a system-on-a-chip (SoC) implementation for accelerating critical operations in hardware while managing the remainder operations in software.

A notable gap in the existing literature pertains to investigations into performance enhancements of ML-KEM using alternative symmetric primitives beyond SHAKE and AES. The use of other symmetric primitives is particularly relevant in cryptography; for instance, governments should use state-of-the-art encryption algorithms and protocols to safeguard sensitive information from unauthorized access and interception. In this regard, the use of PQC schemes such as the ML-KEM becomes imperative[1]. Nevertheless, each nation must develop its unique set of symmetric primitives to reduce dependence on third parties and mitigate external influence. In this regard, the Research and Development Center for Communication Security (CEPESC) has provided government cryptography for the Brazilian government since 1982. To enhance transparency, the CEPESC published a cryptographic library called *libharpia*, which was used in Brazilian elections [Pacheco et al. 2022]. Forró [Coutinho et al. 2022], a symmetric algorithm based on the add-rotate-XOR (ARX) architecture, was later proposed as a special public algorithm for Brazilian elections and, therefore, embedded in *libharpia*. However, it is important to highlight that it is not yet being used in the election process. Focusing on a slight structural modification of Forró, a fast implementation named Xote [Coutinho 2021] was introduced to reduce Forró's execution time. Integrating the Forró and Xote, cryptographic symmetric primitives, with ML-KEM is an interesting strategy to strengthen Brazilian national security and sovereignty.

To address this need, and based on an in-depth study of symmetric primitives in the ML-KEM, this paper introduces two modified versions of ML-KEM. The first modified version replaces SHAKE with Forró, while the second replaces SHAKE with Xote. Replacing SHAKE with Forró or Xote involves the use of modified core functions — i.e., pseudo-random function (PRF), extendable-output function (XOF), and key-derivation

[1]Regarding the use of PQC in government applications, National Cyber Security Centre (NCSC), a United Kingdom agency, and French Cybersecurity Agency (ANSSI) have reports in this matter, evaluating and discussing PQC [NCSC 2020, ANSSI 2022]. Nonetheless, as far as it is known, only the Federal Office for Information Security (BSI), a German federal agency, has officially recommended a PQC scheme [BSI 2020].

function (KDF) — including algorithm presentations, justifications, and integration in ML-KEM. Since the security level remains equivalent across ML-KEM, Forró-based ML-KEM, and Xote-based ML-KEM, we provide a comparison in terms of processing time. Experimental results indicate that the execution time for Forró-based ML-KEM is higher than that of ML-KEM (i.e., in between 3.63% and 6.42%), while Xote-based ML-KEM performs better than ML-KEM (i.e., in between 0.68% and 3.49%).

The remainder of this paper is organized as follows: Section 2 presents fundamental concepts of cryptography and the algorithms employed in this work. Section 3 delves into the implementation details, with a primary focus on the roles of Forró and Xote as PRF, XOF, and KDF within ML-KEM. The experimental results are discussed in Section 4, showcasing the execution time of the proposed methods. Finally, Section 5 offers concluding remarks and potential directions for future research.

1.1. General notation

The notation employed in this paper aligns with that used in the supporting documentation of ML-KEM [National Institute of Standards and Technology 2024]. Functions within the schemes operate on byte arrays as both input and output, where $\mathcal{B} = \{0, ..., 255\}$ represents unsigned 8-bit integers or bytes. Additionally, $\mathcal{B}^K$ denotes the set of byte arrays of K-length, and $\mathcal{B}^*$ represents byte arrays of any length (i.e., a byte stream). $\mathcal{U} = \mathcal{B}^4$ represents unsigned 32-bit integers. The symbol $||$ denotes the concatenation of two-byte arrays. Given a byte array a and a non-negative integer k, $a[k]$ refers to the byte array starting at byte k of a (with indexing starting at zero). Matrices and vectors are denoted by uppercase and lowercase bold letters, respectively, and $\mathbf{A}^T$ refers to the transpose of matrix $\mathbf{A}$.

2. Fundamentals

This section presents brief descriptions of KEM, ML-KEM, and Forró and Xote algorithms that are necessary for understanding the modified version of the ML-KEM. Subsection 2.1 briefly describes the KEM, Subsection 2.2 overviews ML-KEM, and Subsection 2.3 details Forró and Xote algorithms.

2.1. Key Encapsulation Mechanism

Let us consider that Alice and Bob desire to communicate using a PQC scheme. To do so, public-key and symmetric cryptographic schemes should be properly used. A KEM is a public-key cryptography scheme used for exchanging a secret shared (i.e., a key for symmetric cryptography) between two parties. It is generally divided into three main functions: key pair generation, encapsulation, and decapsulation. Key pair generation, executed by Alice, is responsible for generating public and private keys. Alice securely stores the private key, whereas her public key is securely sent to Bob. In turn, Bob receives the public key generated by Alice. He obtains a shared secret, executes the encapsulation on it, and then returns a ciphertext to Alice. Finally, using the ciphertext and the private key, Alice executes the decapsulation, which returns the same shared secret achieved by Bob. After those operations, Alice and Bob hold the same shared secret, enabling them to communicate securely and efficiently with symmetric cryptography. Figure 1 shows a block diagram that illustrates how KEM (public-key cryptography) is employed in a cryptographic scheme.

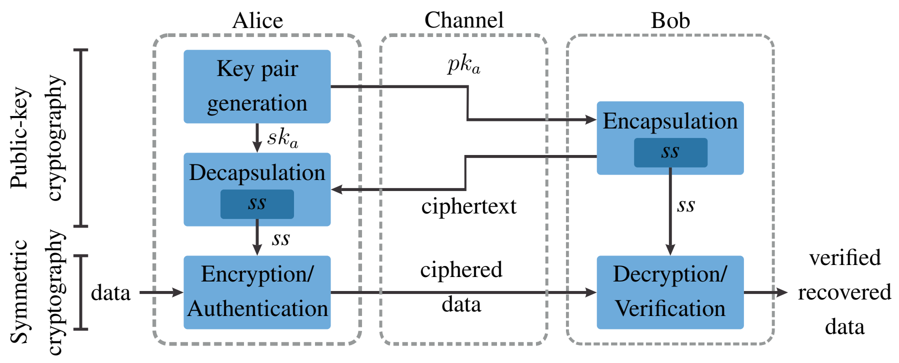

Figure 1. Block diagram illustrating the key agreement between parties and the encryption/decryption process. Note that pk_a, sk_a, and ss stand for public key, private key, and shared secret, respectively.

2.2. ML-KEM

The ML-KEM originated from the K-PKE via a slightly modified Fugisaki-Okamoto transform [Barbosa and Hülsing 2023]. It is the only PQC algorithm being standardized by NIST in the PKE/KEM category. Furthermore, the ML-KEM relies on the Module-Learning With Errors (M-LWE) problem [Langlois and Stehlé 2015, Albrecht and Deo 2017], which was designed to seek high performance without losing flexibility. The ML-KEM is available in three different versions, named ML-KEM-512, -768, and -1024, targeting distinct security levels and, consequently, ensuring a security level equivalent to AES-128, AES-192, and AES-256, respectively.

To run the ML-KEM, the following symmetric primitives are required: two hash functions, a XOF, a PRF, and a KDF. Moreover, to reduce code size and minimize vulnerability, ML-KEM opts for a single underlying symmetric primitive to fulfill all these functions. In this sense, ML-KEM defined SHAKE functions standardized in FIPS-202 [Dworkin 2015][2]. This standard also describes hash functions with the required output lengths and is designed to work as PRF and KDF. The symmetric primitives used by ML-KEM are detailed as follows:

- **Hash functions:** A hash function is an algorithm that takes input data and produces fixed-size output data. In the ML-KEM, the hash functions are instantiated as SHA3-256 and SHA3-512, as described in FIPS-202. The hash functions are required in key pair generation, encapsulation, and decapsulation processes.
- **Extended output function (XOF):** A XOF is a function that maps an arbitrary-length input bit string to an output bit string that can be extended to any desired length. The XOF instantiated by ML-KEM is the SHAKE-128, divided into two steps: absorbing and squeezing. The absorbing step has a seed and two distinct counters as input, which are absorbed by the sponge structure, modifying its internal state. On the other hand, the squeezing step outputs a bit string of the desired length based on the modified internal state provided by the absorbing step. The XOF is used as a step for generating two fundamental matrices for the ML-KEM,

[2]ML-KEM also offers AES-256 and SHA-2 as symmetric primitives, as those primitives can be hardware accelerated on various platforms. As this work does not approach hardware acceleration, only the SHAKE family is addressed.

namely $\hat{\mathbf{A}}$ and $\hat{\mathbf{A}}^T$, which are indirectly part of the public key. It significantly impacts the total execution time of the ML-KEM [Da Costa et al. 2022]. The XOF is required in key pair generation, encapsulation, and decapsulation processes.

- **Pseudo-random function (PRF):** a PRF is a structure used in cryptography that emulates a random oracle. It usually takes a key and a nonce as inputs and then outputs the requested number of bytes. In the ML-KEM, the PRF instantiated is the SHAKE-256. It uses random coins as a key and a counter as a nonce. In the ML-KEM, the PRF generates part of the private key and error vector, which is relevant for the M-LWE problem as it ensures computational hardness and security. The PRFs is required in key pair generation, encapsulation, and decapsulation processes.
- **Key-derivation function (KDF):** A KDF is a cryptographic algorithm that transforms a secret key, password, or key material into a derived key. The SHAKE-256 is also used as KDF in the ML-KEM. It receives a key as input and returns a derived key, i.e., the shared secret. The KDF is required by encapsulation and decapsulation processes.

2.3. Forró and Xote

Forró [Coutinho et al. 2022] and Xote [Coutinho 2021] are two symmetric algorithms. Aiming to enhance security performance, both algorithms evolved from Salsa20 [Bernstein 2008] and ChaCha20 [Bernstein et al. 2008], also known as Salsa20/R and ChaCha20/R where the variable R indicates the number of rounds. To understand Forró and Xote, the following paragraphs describe their evolution, tracing the progression from Salsa20 through ChaCha20 and Forró to Xote.

The stream cipher Salsa20 consists of addition, rotation, and XOR operations on 32-bits words, characterizing an ARX architecture. One of its main characteristics is its efficiency in hardware and software. Salsa20 operates on a state of 64-bytes (or 512-bits), organized as a 4×4 matrix with 32-bits integers. Its state is initialized with a 256-bits key, a 64-bits nonce, a 64-bits counter[3], and 4 constants of 32-bits each. The state matrix is altered in each round by a quarter round (QR) function, which receives 4 of the 16 integers of the state matrix and updates them. A round comprises 4 (four) applications of the QR function receiving different inputs at each call. The output of the Salsa20 is then defined as the sum of the initial state with the resulting state after R rounds[4]. For more details regarding the matrix construction, constants, and QR function of Salsa20, see [Bernstein 2008]. ChaCha20 was proposed as an improvement of Salsa20. Although they present the same structure, modifications in the QR function were made in ChaCha20, enhancing security. For more details regarding security, matrix construction, constants, and QR function of ChaCha20, see [Bernstein et al. 2008].

Forró was designed to evolve ChaCha20 using a novel concept called Pollination [Coutinho et al. 2022]. While ChaCha20 improves diffusion between rounds compared to Salsa20, Forró aims to enhance diffusion within rounds when compared to ChaCha20. This enhancement in Forró is achieved because the QR functions of Forró were modified to be applied dependently between columns and diagonals, which is not

[3]The concatenation of nonce and counter is the initialization vector (IV): $iv := nonce||counter$

[4]As default, R is defined as 20. However, reduced-round variants with R equal to 8 (eight) and 12 have also been introduced.

made in ChaCha20. Consequently, obtaining more diffusion with fewer operations (or rounds) is possible. As a result, the QR function of Forró receives 5 (five) integers instead of 4 (four) and needs to perform fewer rounds, 14 against 20 in ChaCha. For comparison, the best distinguishers against 5 rounds of Salsa, ChaCha, and Forró are, respectively, 2^8, 2^{16}, and 2^{130}. Therefore, Forró can deliver the same security as Salsa20 and ChaCha20 in fewer rounds.

Subsequently, Xote was developed as an evolution of Forró. It maintains identical parameters to Forró, including its QR function but employing two state matrices instead of one. Upon initializing the first matrix, it is duplicated into the second one with an incremented counter. While doubling the computational operations needed to process the state matrix, this design choice allows Xote to generate twice as much keystream as Forró without doubling the processing time. As a result, Xote exhibits improved execution time compared to Forró while delivering the same security level at the cost of using more memory. In summary, with fewer operations, Forró and Xote achieve the same security level as ChaCha20. Consequently, Forró and Xote can perform faster on certain platforms, especially constrained devices with low processing power.

3. ML-KEM instantiating Forró and Xote

To adapt ML-KEM to use Forró or Xote instead of SHAKE, we introduce modified versions of XOF, PRF, and KDF. The hash functions do not use Forró or Xote to maintain a domain separation, as recommended in [Avanzi et al. 2021]. To do so, the algorithms of modified versions of XOF, PRF, and KDF use the following functions[5]: $\{$Forro, Xote$\}$.Keysetup$(\cdot)$, $\{$Forro, Xote$\}$.IVsetup$(\cdot)$, $\{$Forro, Xote$\}$.QR$(\cdot)$, and $\{$Forro, Xote$\}$.Encrypt$(\cdot)$, found in [Coutinho et al. 2022]. In addition, $\{$Forro,Xote$\}$.GenerateBytes$(\cdot)$ is a slight modification of $\{$Forro,Xote$\}$.Encrypt$(\cdot)$. In $\{$Forro,Xote$\}$.GenerateBytes$(\cdot)$, the output is not the input *xored* with the keystream as in $\{$Forro,Xote$\}$.Encrypt$(\cdot)$; instead, the output of $\{$Forro,Xote$\}$.GenerateBytes$(\cdot)$ is directly the keystream. SHAKE functions are also described in detail by FIPS-202 [Dworkin 2015].

Note that the security of ML-KEM with Forró and Xote will be very similar to that of ML-KEM with SHAKE because Forró and Xote are also secure [Coutinho et al. 2022]. In particular, there is a small improvement in security since the XOF in ML-KEM using Forró and Xote provides 256-bits of security. In contrast, XOF in ML-KEM with SHAKE uses SHAKE-128, which provides 128-bits of security. It is important to note that the remaining introduced algorithms maintain the same security level as those they replace.

In the sequel, Subsections 3.1 to 3.3 detailed the modified versions of XOF, PRF, and KDF based on both Forró and Xote, which are required for instantiating symmetric primitives within the ML-KEM.

3.1. Extended-output function

Algorithms 1 and 2 respectively implement the $\{$Forro,Xote$\}$.XOF-absorb$(\cdot)$ and $\{$Forro,Xote$\}$.XOF-squeeze$(\cdot)$ functions proposed by the authors. As discussed in Sub-

[5]From now on, we use a notation where multiple functions are grouped by listing their names within curly braces before the function name. For instance, $\{$Forró, Xote$\}$.Function$(\cdot)$ refers to both Forro.Function$(\cdot)$ and Xote.Function$(\cdot)$. If any additional function like SHAKE has to be included, we add them to the list: $\{$SHAKE, Forró, Xote$\}$.Function$(\cdot)$.

section 2.2, the XOF is used for generating two matrices, $\hat{\mathbf{A}}$ and $\hat{\mathbf{A}}^T$, which are required in key pair generation, encapsulation, and decapsulation algorithms. The absorb step has a 3-tuple (ρ, j, i) and a state matrix st as input. In Forró and Xote, ρ plays as key, whereas i and j will be the first and second 32-bits words of nonce. As the output of the absorbing step, a modified state matrix st is returned. On the other hand, the squeeze step consists of receiving the processed state matrix st by {Forro,Xote}.XOF-absorb($\cdot$) and the length N of the output required. In this sense, {Forro,Xote}.XOF-squeeze($\cdot$) only calls the function {Forro, Xote}.GenerateBytes($\cdot$), generating N-bytes as output.

Algorithm 1 {Forro, Xote}.XOF-absorb(st, ρ, i, j)

```
Input:
State matrix: $st \in \mathcal{U}^{4\times4}$
Seed: $\rho \in \mathcal{B}^{32}$
Nonce: $i, j \in \mathcal{B}^4$
Output:
State matrix: $st \in \mathcal{U}^{4\times4}$
Procedure:
$iv = i||j$
{Forro, Xote}.Keysetup($st, \rho$)
{Forro, Xote}.IVsetup($st, iv$)
{Forro, Xote}.QR($st$)
Return:
State matrix: $st$
```

Algorithm 2 {Forro, Xote}.XOF-squeeze(st, N)

```
Input:
State matrix: $st \in \mathcal{U}^{4\times4}$
Output length: $N \in \mathcal{U}$
Output:
Byte string: $out \in \mathcal{B}^*$
Return:
Byte string: $out$ := {Forro, Xote}.GenerateBytes($st, N$)
```

3.2. Pseudo-random function

Algorithms 3 implements the {Forro, Xote}.PRF($\cdot$). This function receives a seed r, a nonce i, and the output length N as inputs. In this algorithm, {Forro, Xote}.Keysetup($\cdot$) sets r as a key while {Forro, Xote}.IVsetup($\cdot$) uses i as IV, initiating the state matrix st. Based on st and N, {Forro, Xote}.GenerateBytes($\cdot$) is called outputting a N-bytes string.

3.3. Key-derivation function

Algorithm 4 implements the {Forro, Xote}.KDF($\cdot$). This function receives 64-bytes of key material kr and operates on it to generate a shared secret of N bytes, which, in this

Algorithm 3 {Forro, Xote}.PRF(r, i, N)

```
**Input:**
Seed: $r \in \mathcal{B}^{32}$
Nonce: $i \in \mathcal{B}$
Output length: $N \in \mathcal{U}$
**Output:**
Byte string: $out \in \mathcal{B}^*$
**Procedure:**
State matrix: $st \in \mathcal{U}^{4\times4}$
IV: $iv \in \mathcal{B}^{32}$
$iv[0] = i$
{Forro, Xote}.Keysetup($st, r$)
{Forro, Xote}.IVsetup($st, iv$)
**Return:**
Byte string: $out$ := {Forro, Xote}.GenerateBytes($st, N$)
```

case, of the ML-KEM, is fixed to $N = 32$. Nonetheless, Forró and Xote can only use 32-bytes as a key and 16-bytes as a nonce. Consequently, two iterations are required to consume all bits of key material kr. Therefore, the first half of key material kr and iv are initialized in state matrix st_1, and the second half of kr and iv, incremented by one, are initialized in state matrix st_2. Both state matrices are *xored*, generating a state matrix st. Finally, a 32-bytes shared secret is achieved by performing {Forro, Xote}.GenerateBytes($\cdot$) with st as input.

4. Experimental results

This section analyzes the implementation of the modified versions of ML-KEM, based on Forró and Xote, and provides a comparison with the standard ML-KEM based on SHAKE. To provide comprehensive analyses, it details the execution time of the core functions—XOF, PRF, and KDF—using SHAKE, Forró, and Xote, separately from the ML-KEM. Moreover, it compares the execution time of composite functions—key pair generation, encapsulation, and decapsulation—using SHAKE, Forró, and Xote across all security levels. The performance results refer to several execution times conducted across different security levels. They are also presented as boxplots from a collection of 101 samples. Each sample represents the median of $10,001$ iterations of each function at each security level. The results were achieved in a Xeon E5-1650 v4 processor, using GCC compiler with $-O3$ optimization flag. It is important to mention that experiments ran in an Intel Core i5 10th Generation and Apple M2 Pro achieved similar results. The source code can be found in [Lagrota and Azevedo 2024]

This section is organized as follows: Subsection 4.1 pays attention to the results of the core functions; Subsection 4.2 details the results of the composite functions; and Subsection 4.3 discusses the execution time improvements of Forró and Xote compared to SHAKE.

Algorithm 4 $\{$Forro, Xote$\}$.KDF(kr, N)

```
Input:
Key material: $kr \in \mathcal{B}^{64}$
Output:
Shared secret: $ss \in \mathcal{B}^{32}$
Procedure:
State matrix: $st, st_1, st_2 \in \mathcal{U}^{4\times 4}$
IV: $iv \in \mathcal{B}^{32}$
$iv := 0$
▷ Consuming first half of key material $kr$
$\{$Forro, Xote$\}$.Keysetup($st_1, kr$)
$\{$Forro, Xote$\}$.IVsetup($st_1, iv$)
▷ Consuming second half of key material $kr$
$\{$Forro, Xote$\}$.Keysetup($st_2, kr[32]$)
$\{$Forro, Xote$\}$.IVsetup($st_2, iv + 1$)
$st := st_1 \oplus st_2$
Return:
Shared secret: $ss$ := $\{$Forro, Xote$\}$.GenerateBytes($st$, $N$)
```

4.1. Core functions

Core functions, composed of XOF, PRF, and KDF, are functions in the ML-KEM that use symmetric cryptography to accomplish their tasks. Their timing analyses are presented in Subsections 4.1.1, 4.1.2, and 4.1.3, respectively.

4.1.1. XOF

Figure 2 shows the boxplot of the XOF-absorb execution time. The data shows that XOF-absorb runs faster when SHAKE is used compared to Forró and Xote for all security levels. This increased speed is attributed to the use of SHAKE-128. SHAKE-128 outperforms SHAKE-256 due to its higher rate and lower capacity, although it offers less security [Dworkin 2015]. Additionally, Forró runs faster than Xote, which is explained by Xote's architecture, which uses two state matrices instead of one, as Forró does. Consequently, when Forro.QR($\cdot$) and Xote.QR($\cdot$) are called in Algorithm 1, Xote processes two state matrices, doubling the time required compared to Forró.

On the other hand, in the XOF-squeeze, Xote is faster than SHAKE and Forró regardless of the security level, see Figure 3. This advantage is also due to the state matrices, but in this case, it benefits Xote. As shown in Algorithm 2, the only required function is $\{$Forro, Xote$\}$.GenerateBytes($\cdot$), which uses $\{$Forro, Xote$\}$.QR($\cdot$) to generate bytes. Consequently, the double-state matrix construction of Xote allows it to outperform both SHAKE and Forró in this context.

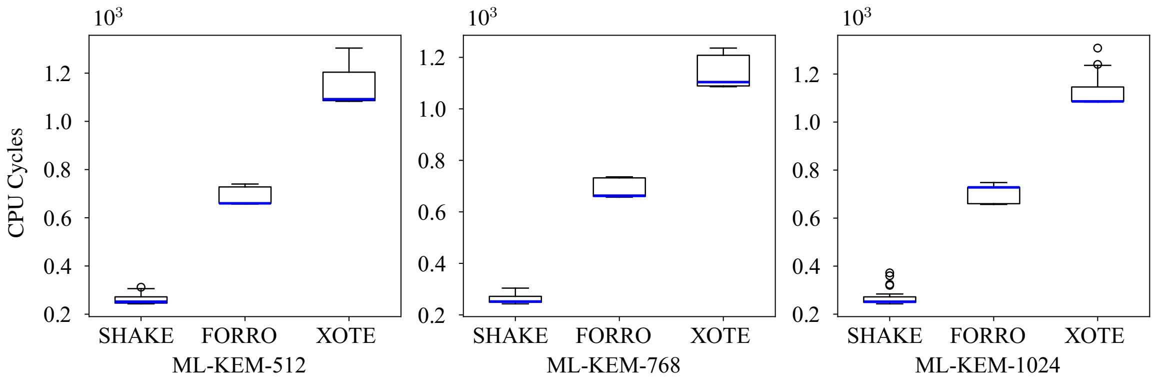

Figure 2. Boxplot of XOF-absorb execution time.

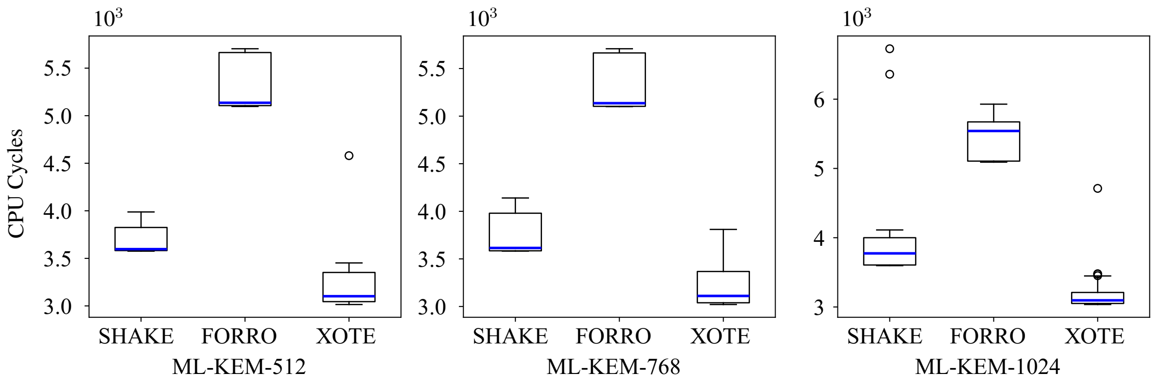

Figure 3. Boxplot of XOF-squeeze execution time.

4.1.2. PRF

The boxplot of the PRF execution time is shown in Figure 4 for all security levels. While SHAKE, Forró, and Xote exhibit similar performance, Xote slightly outperforms the other two, particularly at higher security levels. This is because $\{$Forro, Xote$\}$.PRF$(\cdot)$ relies on $\{$Forro, Xote$\}$.Encrypt$(\cdot)$, which in turn utilizes $\{$Forro, Xote$\}$.QR$(\cdot)$. The use of two-state matrices in Xote speeds up the process. As higher security levels require more pseudorandom data to be generated, the performance difference between SHAKE, Forró, and Xote becomes more pronounced with the increased data requirement.

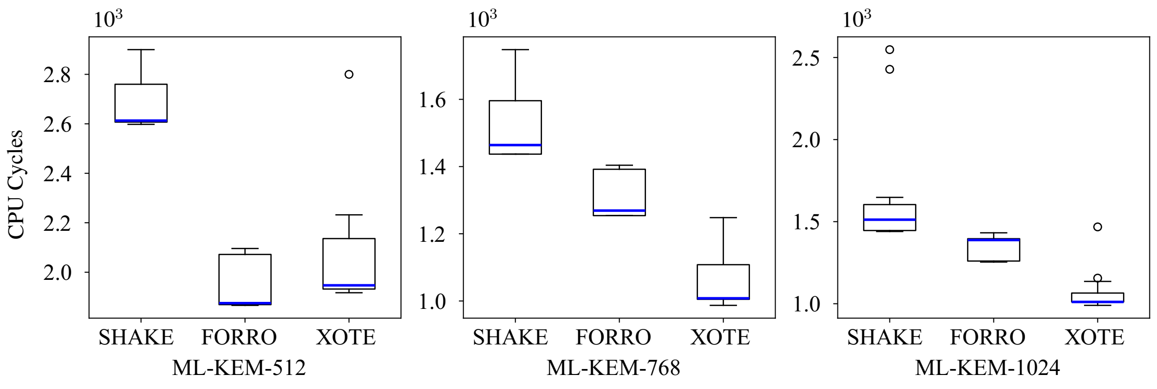

Figure 4. Boxplot of PRF execution time.

4.1.3. KDF

Figure 5 presents the boxplot of KDF execution time for all security levels. In this case, Forró outperforms Xote and SHAKE. The doubled state matrices work against Xote in this scenario. Although {Forro, Xote}.GenerateBytes(·) is used, the required data length is only 32-bytes. This amount can be generated in a single operation by one state matrix. Consequently, Xote processes more data than necessary, reducing its efficiency regardless of the security level.

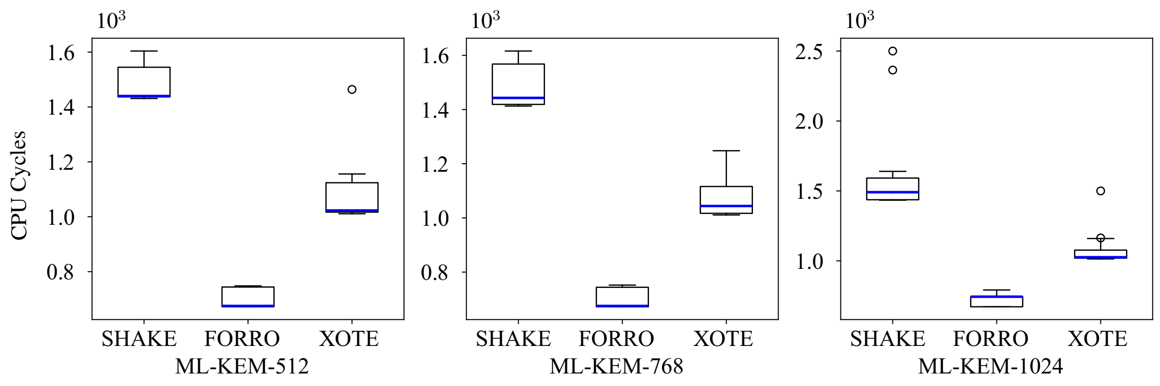

Figure 5. Boxplot of KDF execution time.

4.2. Composite functions

Composite functions are the primary interfaces of ML-KEM used by applications. These functions invoke various operations, including core functions. Table 1 details the frequency with which each core function is called in ML-KEM. It provides essential information for understanding the execution time of composite functions, as core functions directly influence their execution time. Next, Subsections 4.2.1, 4.2.2, and 4.2.3 present the execution time for key pair generation, encapsulation, and decapsulation, respectively.

Table 1. Number of times that XOF-absorb, XOF-squeeze, PRF, KDF are called in key pair generation, encapsulation, and decapsulation. $K \in \{2,3,4\}$ **refers to the security level of ML-KEM-**512**, -**768**, -**1024**, respectively.**

	XOF-absorb	XOF-squeeze	PRF	KDF
Key pair generation	K^2	K^2	$2K$	0
Encapsulation	K^2	K^2	$2K+1$	1
Decapsulation	K^2	K^2	$2K+1$	1

4.2.1. Key pair generation

Figure 6 presents the boxplots of the total execution time for key pair generation. It is important to note that key pair generation involves extensive use of XOF-absorb and XOF-squeeze functions (see Table 1) and is therefore significantly impacted by their execution times. XOF-squeeze has a greater influence among these functions due to its longer total execution time. Consequently, the boxplot of key pair generation execution time closely resembles that of the XOF-squeeze function. Specifically, for all security levels, the ML-KEM instantiated with Xote demonstrates the best performance, followed by its versions based on SHAKE and Forró, in that order.

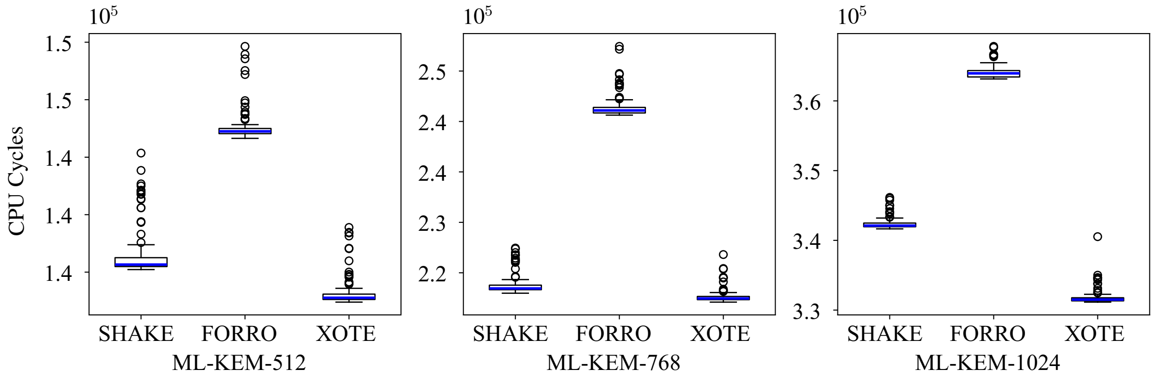

Figure 6. Boxplot of key pair generation execution time.

4.2.2. Encapsulation

The boxplot of encapsulation execution time is presented in Figure 7. The overall behavior of the results is similar to that of key pair generation for all security levels. Thus, the ML-KEM instantiated with Xote shows the best performance, followed by the versions based on SHAKE and Forró, respectively. These results are primarily due to the impact of XOF-squeeze, but also reflect the influence of an additional PRF. Encapsulation involves one more PRF and KDF compared to key pair generation. Consequently, the total execution time of encapsulation is greater, although the performance of the ML-KEM with different symmetric primitives follows the same order.

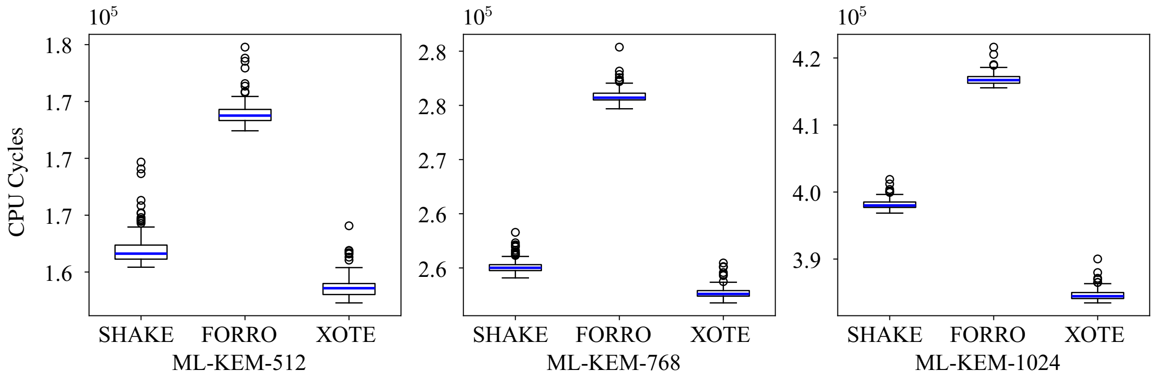

Figure 7. Boxplot of encapsulation execution time.

4.2.3. Decapsulation

Finally, the boxplot of decapsulation execution time is presented in Figure 8. Again, the ML-KEM instantiated with Xote is the fastest, with the SHAKE and Forró versions following in that order. These results follow the same pattern observed for key pair generation and encapsulation. Since encapsulation and decapsulation execute the same number of symmetric primitive functions, similar performances are equally attributed to the impact of XOF-squeeze and PRF. However, note that the overall execution time for decapsulation is greater than for encapsulation. This difference is due to additional functions beyond symmetric primitives required only by decapsulation.

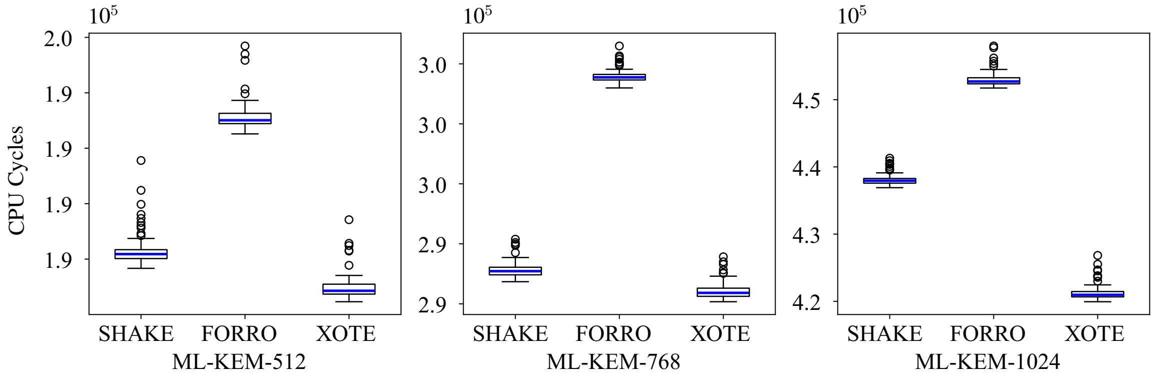

Figure 8. Boxplot of decapsulation execution time.

4.3. Execution time improvement

Table 2 presents the execution time improvement of key pair generation, encapsulation, and decapsulation when ML-KEM, Forró-based ML-KEM, and Xote-based ML-KEM are considered. The execution time improvement is given by

$$\gamma_{\alpha,\beta,K} = 1 - \frac{T_{\alpha,\beta,K}}{T_{\alpha,\text{SHAKE},K}}, \tag{1}$$

where $\alpha \in \{\text{key pair generation, encapsulation, decapsulation}\}$ and $\beta \in \{\text{Forró, Xote}\}$. $T_{\alpha,\text{SHAKE},K}$ and $T_{\alpha,\beta,K}$ refer to the execution times demanded by the ML-KEM and its modified versions, respectively.

Note that ML-KEM with Xote shows a small performance improvement over ML-KEM with SHAKE. This improvement is attributed to the efficient performance of Xote.XOF-squeeze($\cdot$) and Xote.PRF($\cdot$) (see Figures 3 and 4), which enable ML-KEM to be executed slightly faster with Xote than with SHAKE. Conversely, ML-KEM with Forró experiences a small performance decrease compared to ML-KEM with SHAKE. This is due to Forro.XOF-squeeze($\cdot$) significantly impacts the performance, making it slower than SHAKE in total execution time. The performance comparison remains consistent across all security levels (K).

5. Conclusion

This paper has explored replacing SHAKE with Forró and Xote as cryptographic primitives in ML-KEM. To achieve this, detailed modifications to the core functions of ML-KEM were made to fulfill Forró and Xote, and their integration with ML-KEM was discussed. Numerical results from experiments show that ML-KEM, Forró-based ML-KEM, and Xote-based ML-KEM exhibit distinct execution times for their core functions. Interestingly, the optimal performance for each core function is achieved by a different symmetric primitive. However, when evaluating the composite functions—key pair generation, encapsulation, and decapsulation—integral to a KEM, Xote-based ML-KEM

Table 2. Execution time improvement ($\gamma_{\alpha,\beta,K}$) of Forró- and Xote-based ML-KEM in percentage ($\%$).

	Algorithm	Forró	Xote
ML-KEM-512	Key pair generation	-4.13	1.03
	Encapsulation	-3.66	0.92
	Decapsulation	-3.22	0.88
ML-KEM-768	Key pair generation	-7.90	0.44
	Encapsulation	-6.04	0.93
	Decapsulation	-5.61	0.63
ML-KEM-1024	Key pair generation	-6.38	3.10
	Encapsulation	-4.70	3.40
	Decapsulation	-3.37	3.88

demonstrates slightly better performance than ML-KEM while maintaining equal or superior security across various security levels. Furthermore, the numerical results reveal that Forró-based ML-KEM is outperformed by both Xote-based ML-KEM and ML-KEM in terms of execution time, despite offering similar security. Overall, this work presented ML-KEM, a PQC algorithm, instantiated with Forró and Xote, two Brazilian cryptographic primitives, presenting better or similar performance. Future work should include a similar analysis incorporating AES with hardware acceleration and optimizing SHAKE, Forró, and Xote using Advanced Vector Extensions 2 (AVX2).

Acknowledgments

This research was supported by Centro de Pesquisa e Desenvolvimetno para a Segurança das Comunicações (CEPESC), Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) under Grant 001, Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) under grants $404068/2020-0$ and $314741/2020-8$, Fundação de Amparo à Pesquisa do Estado de Minas Gerais (FAPEMIG) under grants APQ-$03609-17$, TEC-PPM $00787-18$, and APQ-04623-22, Instituto Nacional de Energia Elétrica (INERGE).

References

[Albrecht and Deo 2017] Albrecht, M. R. and Deo, A. (2017). Large modulus ring-LWE $\geq$ module-LWE. In *Proc. Int. Conf. on the Theory and Application of Cryptology and Information Security*, pages 267–296. Springer.

[ANSSI 2022] ANSSI (2022). Anssi views on the post-quantum cryptography transition. Technical report, ANSSI.

[Avanzi et al. 2021] Avanzi, R., Bos, J. W., Ducas, L., Eike Kiltz, T. L., Lyubashevsky, V., Schanck, J. M., Schwabe, P., Seiler, G., and Stehlé, D. (2021). CRYSTALS-Kyber algorithm specifications and supporting documentation (version 3.0). Technical report, NIST, Gaithersburg, MD.

[Barbosa and Hülsing 2023] Barbosa, M. and Hülsing, A. (2023). The security of kyber's fo-transform. *IACR Cryptology ePrint Archive*, 2023(755).

[Bernstein 2008] Bernstein, D. J. (2008). The salsa20 family of stream ciphers. In *New stream cipher designs: the eSTREAM finalists*, pages 84–97. Springer.

[Bernstein et al. 2008] Bernstein, D. J. et al. (2008). Chacha, a variant of salsa20. In *Workshop record of SASC*, volume 8, pages 3–5.

[BSI 2020] BSI, C. M. (2020). Recommendations and key lengths. Technical report, BSI.

[Costa et al. 2022] Costa, V. L. R. D., Camponogara, Â., López, J., and Ribeiro, M. V. (2022). The feasibility of the crystals-kyber scheme for smart metering systems. *IEEE Access*, 10:131303–131317.

[Coutinho 2021] Coutinho, M. (2021). Forró and xote cipher. `https://github.com/murcoutinho/forro_cipher`.

[Coutinho et al. 2022] Coutinho, M., Passos, I., Grados Vásquez, J. C., de Mendonça, F. L. L., de Sousa, R. T., and Borges, F. (2022). Latin dances reloaded: Improved cryptanalysis against salsa and chacha, and the proposal of forró. In Agrawal, S. and Lin, D., editors, *Advances in Cryptology – ASIACRYPT 2022*, pages 256–286, Cham. Springer Nature Switzerland.

[Da Costa et al. 2022] Da Costa, V. L., Camponogara, Â., López, J., and Ribeiro, M. V. (2022). The feasibility of the crystals-kyber scheme for smart metering systems. *IEEE Access*, 10:131303–131317.

[Dworkin 2015] Dworkin, M. (2015). Sha-3 standard: Permutation-based hash and extendable-output functions.

[Jati et al. 2024] Jati, A., Gupta, N., Chattopadhyay, A., and Sanadhya, S. K. (2024). A configurable crystals-kyber hardware implementation with side-channel protection. *ACM Trans. Embed. Comput. Syst.*, 23(2).

[Lagrota and Azevedo 2024] Lagrota, V. and Azevedo, B. (2024). Ml-kem instantiated with forró and xote cipher. `https://github.com/beatrizufjf/kyber.git`.

[Langlois and Stehlé 2015] Langlois, A. and Stehlé, D. (2015). Worst-case to average-case reductions for module lattices. *Designs, Codes and Cryptography*, 75(3):565–599.

[National Institute of Standards and Technology 2024] National Institute of Standards and Technology (2024). Module-lattice-based key encapsulation mechanism standard. Federal Information Processing Standards Publication (FIPS) NIST FIPS 203, Department of Commerce, Washington, D.C.

[NCSC 2020] NCSC (2020). Preparing for quantum-safe cryptography. Technical report, NCSC.

[Nguyen and Gaj 2021] Nguyen, D. T. and Gaj, K. (2021). Optimized software implementations of crystals-kyber, ntru, and saber using neon-based special instructions of armv8. In *Proceedings of the NIST 3rd PQC Standardization Conference (NIST PQC 2021)*.

[Pacheco et al. 2022] Pacheco, R., Braga, D., Passos, I., Araújo, T., Lagrota, V., and Coutinho, M. (2022). libharpia: a new cryptographic library for brazilian elections. In *Anais do XXII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*, pages 250–263. SBC.

[Shor 1994] Shor, P. W. (1994). Algorithms for quantum computation: Discrete logarithms and factoring. In *Proc. 35th Annual Symposium on Foundations of Computer Science*, pages 124–134.

[Wan et al. 2022] Wan, L., Zheng, F., Fan, G., Wei, R., Gao, L., Wang, Y., Lin, J., and Dong, J. (2022). A novel high-performance implementation of crystals-kyber with ai accelerator. In Atluri, V., Di Pietro, R., Jensen, C. D., and Meng, W., editors, *Computer Security – ESORICS 2022*, pages 514–534, Cham. Springer Nature Switzerland.

[Xing and Li 2021] Xing, Y. and Li, S. (2021). A compact hardware implementation of cca-secure key exchange mechanism crystals-kyber on fpga. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 328–356.