Co-design architecture for accelerating cryptographic algorithms

1st Killian Marty
Institut National des Sciences Appliquées
Toulouse, France
killianmarty@outlook.com

2nd Thomas Chourret
Institut National des Sciences Appliquées
Toulouse, France
chourret@insa-toulouse.fr

3rd Baptiste Rébillard
Institut National des Sciences Appliquées
Toulouse, France
rebillar@insa-toulouse.fr

4th Leandro Rodriguez
Institut National des Sciences Appliquées
Toulouse, France
lerodrig@insa-toulouse.fr

5th Gabriel Lanoux
Institut National des Sciences Appliquées
Toulouse, France
lanoux@insa-toulouse.fr

6th Aurelien Pouilles
Institut National des Sciences Appliquées
Toulouse, France
pouilles@insa-toulouse.fr

7th Sofia Salinas
Institut National des Sciences Appliquées
Toulouse, France
ssalinas-ric@insa-toulouse.fr

***Abstract*—With the rise of quantum computing, traditional cryptographic methods are increasingly vulnerable. This necessitates the development of post-quantum cryptographic solutions. Attribute-Based Encryption (ABE) offers a promising approach for fine-grained access control, particularly in cloud computing. However, implementing post-quantum ABE efficiently remains a challenge due to the computational complexity of the underlying cryptographic operations. This paper focus on a Rank-Based Cryptography (RBC) approach, leveraging rank error-correcting codes to enhance security against quantum attacks. To optimize performance, we implemented the scheme in C using the RBC library, enabling efficient operations on resource-constrained environments. Our initial development includes the setup phase and a partial key generation module, providing a foundation for further optimizations. Future work will explore Field Programmable Gate Array (FPGA) acceleration to further enhance computational efficiency. This study contributes to the advancement of practical post-quantum ABE implementations, bridging the gap between security and performance. Our initial development includes the setup phase and a partial key generation module, providing a foundation for further optimizations. Preliminary benchmarks using the RBC library show that the setup phase executes in under 50 ms, and key generation requires around 150–200 ms depending on attribute count. Our results, obtained on a standard Central Processing Unit (CPU), validate the feasibility of rank-based cryptographic primitives in constrained environments. Future work will explore FPGA acceleration to further enhance computational efficiency. This study contributes to the advancement of practical post-quantum ABE implementations, bridging the gap between security and performance.**

***Index Terms*—Attribute Based Encryption, ABE, Cryptography, FPGA, Cloud Computing, Post-quantum, Post-quantum cryptography, Quantum attribute-based encryption**

I. Introduction

The rise of quantum computers threatens traditional cryptographic standards such as RSA and Diffie-Hellman, which rely on hard problems that quantum algorithms, such as Shor's, could efficiently solve. While these methods are secure against classical attacks, quantum computing could break them, prompting international efforts like the NIST competition to develop post-quantum cryptography (PQC) standards. [1] Among modern cryptographic approaches, ABE plays a key role, particularly in cloud computing, by enabling access control based on attributes. In ABE, data is encrypted under an access policy, and only users with matching attributes can decrypt it.

Though post-quantum ABE algorithms exist, none have been standardized yet. Their complexity, especially on hardware platforms, remains a challenge. Initially based on Elliptic Curve Cryptography (ECC), ABE has evolved toward lattice-based schemes to resist quantum threats. ECC is a widely adopted cryptographic technique known for its high security and efficiency, offering equivalent protection to traditional methods such as RSA with much smaller key sizes. [2] However, ECC is not post-quantum secure, as it relies on the elliptic curve discrete logarithm problem, which quantum computers could theoretically solve using Shor's algorithm. Elliptic Curve Attribute-Based Encryption (EC-ABE) enhances traditional ABE by integrating ECC, resulting in improved efficiency and reduced key sizes. This integration is particularly advantageous for resource-constrained environments such as Internet of Things (IoT) devices and mobile applications. [3]

A lattice is a mathematical structure that consists of a set of points in a multidimensional space, generated by all possible integer linear combinations of a given set of n linearly independent vectors. Lattice-based ABE schemes are considered quantum-safe because their security relies on problems that are difficult to solve even for quantum computers. In terms of attack resistance, lattice-based ABE schemes are also

resistant to collusion, meaning that a group of unauthorized users cannot combine their keys to decrypt data they are not individually authorized to access. Lattice-based ABE schemes has also a considerable computational cost. For example, a scheme based on Learning With Errors (LWE) may see the size of the keys increase proportionally with the number of attributes per user , leading to significant storage overheads in large-scale systems. All parts of the encrypt and decrypt processes are also computationally expensive. This can result in slower operations, posing a challenge for implementing these schemes at scale in cloud computing environments.

Error-correcting codes play a key role in systems designed to withstand the challenges of the post-quantum era. Codes such as Calderbank-Shor-Steane (CSS) are essential for ensuring data integrity and confidentiality against physical disruptions and quantum attacks. Unlike classical codes like Hamming, CSS codes leverage quantum mechanical properties such as entanglement and superposition to simultaneously correct bit and phase errors on qubits. CSS codes are widely used in protocols like Quantum Key Distribution (QKD) and cryptographic systems resistant to the computational power of quantum computers, providing a robust foundation for secure communication and computation.

In ABE schemes designed for the post-quantum era, CSS codes enhance the security and efficiency of secret sharing. CSS codes enable the distribution of secret keys in the form of qubits, ensuring that they can only be reconstructed by users whose attributes satisfy a predefined access policy. This approach guarantees that authorized users can reconstruct the necessary information, even in noisy or perturbed environments. By integrating these codes, ABE schemes become resistant to algorithmic attacks, such as those leveraging Shor's algorithm, while maintaining flexibility for dynamic and scalable access policies. CSS codes thus form a fundamental component of cryptographic systems tailored to the challenges of the post-quantum era. [4]

To overcome the computational challenges of Lattice-based ABE schemes, hardware solutions such as Field Programmable Gate Arrays could be employed. These dedicated hardware units can accelerate the complex computations involved in lattice operations, they are particularly efficients when computing parallel tasks like matrix multiplications. By using FPGA, it is possible to reduce the load on the CPU and significantly improve the efficiency of ABE systems while maintaining high security levels. Several post-quantum algorithms have already been implemented on FPGA boards, with some available as open-source projects [5]. Additionally, a CP-ABE implementation on FPGA has been previously developed [6].

A common practice used in these implementations to enhance their performance is to design optimized dedicated modules for simple arithmetic operations. [6] It is also possible to parallelize costly and frequently used operations such as polynomial multiplication and modular reduction. For this purpose, solutions based on the Number-Theoretic Transform (NTT) and Barrett modular reduction algorithm are often employed in other algorithms, and can be transposed to an ABE implementation [7]

Lattice-based ABE schemes offer a robust and quantum-safe solution for secure data access management in cloud storage environments. While they provide significant advantages in terms of resistance to quantum attacks and flexible access policies, challenges remain in terms of performance, in particular the size of the parameters and computational costs.

Most cryptographic algorithms are implemented in C due to its efficiency, portability, and close interaction with hardware resources. These characteristics make C a preferred choice for developing performance-critical applications, such as cryptographic operations. In the context of FPGA-based implementations, C enables low-level hardware control and optimization, ensuring that computationally intensive tasks, such as polynomial multiplication and modular reduction, are executed efficiently. Additionally, C benefits from a well-established ecosystem of cryptographic libraries, facilitating secure and reliable implementations. Leveraging these advantages, we aim to explore how to implement post-quantum cryptographic algorithms on FPGA platforms to address existing performance challenges and enhance scalability for real-world applications.

II. Background and Required Knowledge

A. Symbols used in this paper

TABLE I
SYMBOLS TABLE

Symbol	Description
m	Degree of the field-extension $\mathbb{F}_{q^m}$
n	Length of codewords in $\mathbb{F}_{q^m}^n$
q	Size of the base field $\mathbb{F}_q$
r	Rank of the encryption support subspace $E \subset \mathbb{F}_{q^m}^n$
d	Rank of the secret-key support subspace $F \subset \mathbb{F}_{q^m}^n$
t	Output length (in bits) of Δ and h_i functions
$\mathbb{F}_{q^m}$	Extension field of degree m over $\mathbb{F}_q$
$\mathbb{F}_{q^m}^n$	n-dimensional vector space over $\mathbb{F}_{q^m}$
$\mathrm{wt}_R(x)$	Rank of x: $\dim_{\mathbb{F}_q}\langle x_1, x_2, \ldots, x_n\rangle$
$\langle \mathbf{x}\rangle_{\mathbb{F}_q}$	The support of the components of vector $\mathbf{x}$
b_s	Size of the Bloom filter bit-array
k	Number of independent hash functions $h_i, \ldots, h_k$
h_i	Independent hash functions $\{0,1\}^* \to \{0,1\}^t$
$BF(x)$	Bloom filter of the attribute set x
Att	Attributes set for the Bloom Filter $(Keys, Values)$
J	Hash function $\{0,1\}^{b_s} \to \mathbb{F}_{q^m}^n$
Δ	Hash function $\mathbb{F}_{q^m}^n \to \{0,1\}^t$
$p_k = h$	Public Key
$s_k = (F, x, y)$	Private Key

B. Operations over finite fields

Let $\mathbf{u} = (u_0, u_1, \ldots, u_{n-1}) \in \mathbb{F}_{q^m}^n$ and $\mathbf{v} = (v_0, v_1, \ldots, v_{n-1}) \in \mathbb{F}_{q^m}^n$ be two vectors over the finite field extension $\mathbb{F}_{q^m}$. Each vector can be naturally associated with a polynomial in the quotient ring $\mathbb{F}_{q^m}[X]/(X^n - 1)$.

The **addition** of two vectors $\mathbf{u}$ and $\mathbf{v}$ corresponds to the addition of their associated polynomials.

$$\mathbf{u}(X) + \mathbf{v}(X) = \sum_{k=0}^{n-1} (u_k + v_k) X^k \tag{1}$$

The **multiplication** $\mathbf{u} \cdot \mathbf{v} \in \mathbb{F}_{q^m}^n$ is defined as the vector corresponding to the product of the two polynomials modulo $X^n - 1$. [8]

$$\mathbf{u} \cdot \mathbf{v} = \left(\sum_{i=0}^{n-1} v_i X^i \right) \cdot \left(\sum_{j=0}^{n-1} u_j X^j \right) \mod (X^n - 1) \quad (2)$$

C. Rank Metric

The rank metric is a distance measure defined over vectors or matrices with entries in an extension field $\mathbb{F}_{q^m}$. For a vector $x \in \mathbb{F}_{q^m}^n$, its rank weight, denoted $\mathrm{wt}_R(x)$, is defined as the dimension of the $\mathbb{F}_q$-linear subspace generated by the components of x when viewed as elements of $\mathbb{F}_{q^m}$.

$$\mathrm{wt}_R(x) = \dim_{\mathbb{F}_q} \langle x_1, x_2, \ldots, x_n \rangle \quad (3)$$

Similarly, the rank distance between two vectors $x, y \in \mathbb{F}_{q^m}^n$ is defined as $\mathrm{wt}_R(x - y)$. [9]

This metric is particularly suited for cryptographic applications and network coding, where errors may affect multiple positions in a correlated manner. Unlike the Hamming metric, which counts the number of differing components, the rank metric captures the linear dependencies among entries over the base field.

D. Difficult problems in rank metric

The fundamental challenge in rank-metric cryptography is the *Rank Syndrome Decoding (RSD)* problem: given a matrix $H \in \mathbb{F}_{q^m}^{(n-l)\times n}$, a syndrome s, and a target rank r, the goal is to find a vector x of rank r such that $Hx^T = s$. This problem has been shown to be NP-hard via a randomized reduction. Its inherent difficulty lies in the vast number of possible solutions, as the count of rank-r subspaces in $\mathbb{F}_{q^m}^n$ scales roughly like q^{rm}, making exhaustive search infeasible for large parameters. [10]

E. LRPC codes

Low Rank Parity Check (LRPC) codes are a class of error-correcting codes used in cryptographic schemes based on the rank metric. They rely on the construction of a parity-check matrix whose entries belong to a low-rank subspace, enabling efficient decoding algorithms. LRPC codes offer strong security guarantees while maintaining compact key sizes, making them suitable for post-quantum cryptography. Their structure enables the design of efficient trapdoor functions used in encryption and signature schemes. Rollo and other rank-metric-based systems often use LRPC codes for their performance and security. [11]

F. Rank Support Recovery (RSR)

The *Rank Support Recovery (RSR)* algorithm is a decoding procedure for rank metric codes that extracts the *error support*, a subspace E characterizing the structure of the error.

Given a syndrome vector s, RSR computes the intersection of transformed subspaces derived from to isolate E. Formally, for an input syndrome $s \in \mathbb{F}_{q^m}^n$ and a rank support space $F \subseteq \mathbb{F}_{q^m}$ with basis $\{F_1, \ldots, F_d\}$, RSR outputs the support E such that:

$$E = \bigcap_{i=1}^{d} F_i^{-1} \cdot \mathcal{S}, \quad (4)$$

where $\mathcal{S} = \langle s \rangle_{\mathbb{F}_q}$ is the $\mathbb{F}_q$-linear span of s. [10]

The algorithm is given by Algorithm 1.

Algorithm 1: Rank Support Recovery (RSR) [12]

Input: Syndrome vector $s \in \mathbb{F}_{q^m}^n$, support space $F = \{F_1, \ldots, F_d\} \subseteq \mathbb{F}_{q^m}$

Output: Error support subspace E

$\mathcal{S} \leftarrow \langle s \rangle_{\mathbb{F}_q}$
$\mathcal{S} \leftarrow \mathrm{GaussElim}(\mathcal{S})$
for $i \leftarrow 1$ ***to*** d **do**
  $F_i^{-1} \leftarrow \mathrm{Inverse}(F_i)$
  $S_i \leftarrow F_i^{-1} \cdot \mathcal{S}$
$E \leftarrow S_1$
for $i \leftarrow 2$ ***to*** d **do**
  $E \leftarrow E \cap S_i$
$E \leftarrow \mathrm{RREF}(E)$
return E

G. Probability of failure in LRPC

In a rank metric code of length n and dimension l, the LRPC decoding fails with probability $\leq q^{-(n-l+1-rd)}$ and has computational complexity $\mathcal{O}(4r^2d^2m + r^2n^2)$ when $rd \leq n-l$ where r denotes the rank of the *error support* and d the rank of the LRPC. [10]

H. Bloom filter

1) Principle: A *Bloom filter* is a probabilistic data structure designed to efficiently test whether an element belongs to a set, offering high space efficiency. It guarantees the absence of false negatives. If the filter indicates that an element is not in the set, this is always correct. However, it may produce false positives, it might indicate that an element is in the set when it is not. [13]

The Bloom filter consists of a bit array of size bs, initially filled with zeros, and k independent hash functions $h_1, \ldots, h_k$ that map elements to positions in the array. To insert an element a into the filter, the positions $h_1(a), \ldots, h_k(a)$ are computed and the corresponding bits in the array are set to 1. To test whether an element b is in the set, we need to check if all bits at positions $h_1(b), \ldots, h_k(b)$ are set to 1. If so, b is probably in the set; if any of them is 0, then b is definitely not in the set. [13]

Note : We applied the modulo operation with $b_s = bloom\_filter\_size$ to ensure that the hash output stays within the bounds of the Bloom filter array, thereby preventing buffer overflows in case a hash function returns an index that exceeds the array size.

Algorithm 2: BloomFilter [13]

Input: Set A, hash functions $\{h_1, \ldots, h_k\}$, integer b_s
Output: Bloom filter array

```
Initialize filter as an array of b_s bits set to 0;
foreach a_i ∈ A do
    foreach h_j ∈ {h_1, ..., h_k} do
        filter[h_j(a_i)%b_s] ← 1;
return filter
```

2) False positive rate: The probability of a false positive in a Bloom filter is given by the Equation 5.

$$p = \left(1 - e^{-\frac{kn}{b_s}}\right)^k \tag{5}$$

where:

- k is the number of independent hash functions used,
- n is the number of elements inserted into the filter,
- b_s is the size of the Bloom filter in bits.

This formula assumes uniform and independent hashing. The false positive rate increases with the number of elements n and decreases with the size of the filter b_s and a well-chosen number of hash functions k. In practice, there is a trade-off: increasing b_s and k can reduce the false positive probability, but it also increases the memory usage and the size of the ciphertext when the Bloom filter is included in cryptographic schemes. [13]

Therefore, parameter selection is crucial. For a given ratio m/n, the optimal number of hash functions that minimizes the false positive rate is given by the Equation 6.

$$k_{\text{opt}} = \frac{b_s}{n} \ln 2 \tag{6}$$

Choosing k close to this value ensures a good balance between accuracy and efficiency. [13]

III. The Scheme

A. Presentation

In this paper, we propose a new scheme built upon the **ROLLO-II Algorithm** [14], extended with an Attribute-Based Encryption layer. This scheme, whose system is illustrated in Figure 1, constitutes of a public-key encryption system based on low-rank parity-check (LRPC) codes, analogous to the McEliece cryptosystem in the rank metric setting.

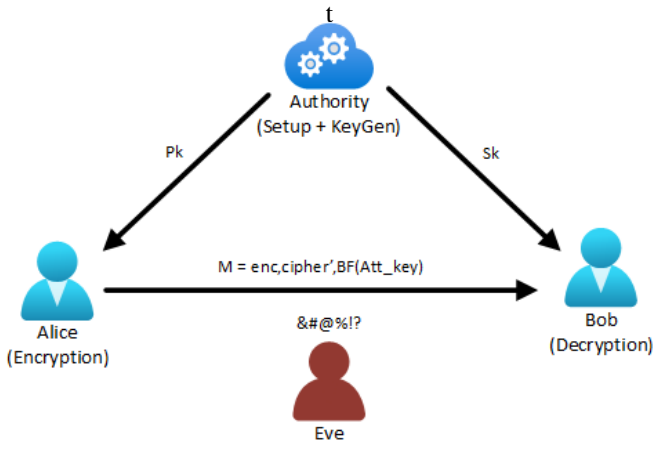

Fig. 1. System model architecture

B. Scheme specification

a) Setup: We choose the parameters d, r, m, and n, as well as two hash functions $J : \{0,1\}^m \to \mathbb{F}_{q^m}^n$ and $\Delta : \mathbb{F}_{q^m}^n \to \{0,1\}^t$, with t the length of the bit array output. We specifically choose $q = 2$, thereby operating over the binary field $\mathbb{F}_2$. This choice is motivated by the fact that addition in $\mathbb{F}_2$ corresponds to bitwise XOR operations, which are extremely efficient. Consequently, computations over $\mathbb{F}_2$ are well-suited for cryptographic applications where performance is critical. For the Bloom filter, we needed to choose the size of the filter b_s and k hash functions.

b) Keygen: The Keygen phase is the same as in Rollo-I and Rollo-II. A random support F is generated, and two elements $x, y \in F$ are selected. The public key is then computed as $p_k = h = x^{-1}y$, while the secret key is defined as $s_k = (F, x, y)$. The pseudocode of this algorithm is presented in Algorithm 3. [14]

Algorithm 3: Rollo-II Keygen [14]

Input: A rank d
Output: Public key p_k and secret key s_k

```
Generate a random support F of rank d;
Choose x, y ∈ F ;
Compute the public key: p_k ← x^{-1} y ;
Compute the secret key: s_k ← (F, x, y) ;
return p_k, s_k
```

c) Encryption: Encryption requires the public key p_k, a list of attributes(s) of the recipient(s) Att (pairs of $(key, value)$), and a message $plaintext$ as input.

First, a random subspace $E \subset \mathbb{F}_{q^m}^n$ of dimension r is generated, and two vectors $(e_1, e_2) \in E^2$ are selected.

Then, the following steps are performed:

1) compute $cipher = e_1 + e_2 \times p_k$

2) derive $s = J(BF(Att))$
3) compute $cipher' = cipher \cdot s$
4) finally $enc = plaintext \oplus \Delta(cipher')$

The final message to transmit is

$$M = (enc, cipher', BF(Att_{keys})) \tag{7}$$

The pseudocode of the algorithm is defined in Algorithm 4..

Algorithm 4: Encryption

Input: d, p_k, recipient(s) attribute(s) Att, $plaintext$
Output: Encrypted message M

Generate a random subspace $E \subset \mathbb{F}_{q^m}^n$ of dim r ;
Select two vectors $e_1, e_2 \in E^2$;
$cipher \leftarrow e_1 + e_2 \times p_k$;
$s \leftarrow J(BF(Att))$;
$cipher' \leftarrow cipher \cdot s$;
$enc \leftarrow M \oplus \Delta(cipher')$;
return *M = (enc, cipher', $BF(Att_{keys})$)*

Note: Only the message M and the Bloom filter of the attribute keys, $BF(Att_{keys})$ are transmitted; the Bloom filter of the attributes themselves, $BF(Att)$, is not sent.

d) Decryption: For decryption, the Bloom filter $BF(Att)$ is reconstructed using $BF(Att_{keys})$ and the actual attribute values Att (the receiver's attributes), as detailed in Algorithm 5.

Algorithm 5: BF_Recover Algorithm

Input: $BF(Keys)$, recipient(s) attribute(s) Att
Output: Recovered Bloom Filter M

$BF_{\text{out}} \leftarrow \{0\}^{b_s}$;
foreach *(key, value) in Att* **do**
    if *key* $\in BF(Keys)$ **then**
        Add (key, value) to BF_{out};
return BF_{out}

Then compute $s = J(BF(Att))$, recover $cipher = cipher' \cdot s^{-1}$, and compute $E = \text{RSR}(x \cdot cipher, F)$.
Finally, the original message is recovered with

$$decode = enc \oplus \Delta(E) \tag{8}$$

The pseudocode of the algorithm is defined in Algorithm 6.

C. Implementation

a) Language: The implementation was developed in C, a low-level language well-suited for cryptographic applications due to its performance and fine-grained control over memory. This choice also aligns with the need for compatibility with embedded systems or FPGA environments.

Algorithm 6: Decryption

Input: $M = (enc, cipher', BF(Att_{keys}))$,
$s_k = (F, x, y)$, $MyAtt$
Output: The $plaintext$

$BF(Att) \leftarrow BF\_Recover(MyAtt, BF(Att_{keys}))$;
$s \leftarrow J(BF(Att))$;
$cipher \leftarrow cipher' \cdot s^{-1}$;
$E \leftarrow \text{RSR}(x \cdot cipher, F)$;
$plaintext \leftarrow enc \oplus \Delta(E)$;
return $plaintext$

b) RBC Library: We used the RBC Lib – Rank-Based Cryptography library [12], an open-source C library implementing several post-quantum encryption schemes based on rank metric codes. This library provides efficient primitives for encoding, decoding, and key generation, which were instrumental in constructing the presented scheme.

In addition, RBC Lib offers a comprehensive set of low-level algebraic tools, such as linear system solvers and operations over finite fields, which greatly facilitated the potential implementation of alternative cryptographic schemes within the same framework.

c) Repository: The full implementation, including source code and example usage, is available on our public repository: https://github.com/killianmarty/Rank-Based-ABE-PKE.

D. Decrypting failure rate

a) RSR: Since the Rank Support Recovery (RSR) algorithm can fail, and is essential to decrypt data, a cipher can be not decryptable with the same probability as the RSR fail which is in our case, as expressed in Equation 9. [15].

$$DFR \leq q^{-(d-1)(m-rd-r)} + q^{-(n-rd+1)} \tag{9}$$

E. Parameters

For this scheme, parameters are the following:

- m : Size of $x \in \mathbb{F}_{q^m}$ extension field elements.
- n : Size of $x \in \mathbb{F}_{q^m}^n$ vectors elements.
- r : Rank of the support E (used for encryption).
- d : Rank of the private key support F (used for decryption).

The choice of these parameters affects directly the scheme security but also the decryption failure rate, so they must be chosen carefully.

The rank r and d must be low as defined by LRPC codes. If they are too high, the RSR algorithm will probably fail or take too much time to compute.

m and n increases the security but also the computational complexity.

In our implementation, the choice of parameters m and n is limited as following:

- m : 67, 83, 97, 127, 151 and 181.
- n : 83, 113, 149, 179, 189, 193 and 211.

F. Security

1) Security of ROLLO-II layer: The ROLLO-II layer ensures post-quantum security based on the rank metric. For each security level, the chosen parameters achieve a negligible decoding failure rate (DFR), ensuring that legitimate ciphertexts can be decrypted reliably: [12]

- **128-bit security** : DFR $\approx 2^{-134}$
- **192-bit security** : DFR $\approx 2^{-130}$
- **256-bit security** : DFR $\approx 2^{-136}$

The security is independent of the ABE layer and relies solely on the hardness of the rank syndrome decoding problem, which is believed to be resistant even to quantum attacks.

2) Security of ABE layer:

a) Bloom Filter false positives consequences: The ABE layer uses Bloom Filters to efficiently check attribute sets. Since Bloom Filters are probabilistic, they may return false positives. This means an unauthorized user might satisfy an access policy by mistake. However, a false positive at this stage does **not** imply access to the plaintext, as the message remains protected by the ROLLO-II encryption. Although a false positive, given by the probability (Equation 5) may allow an access policy check to pass incorrectly, the attacker still cannot decrypt the ciphertext without the correct ROLLO-II private key. By choosing appropriate parameters to keep p low (e.g., $< 2^{-20}$), the probability of unauthorized decryption remains negligible.

3) Parameters for each level of security: The parameters required to assure security levels are listed below.

TABLE II
PARAMETERS FOR EACH SECURITY LEVEL

Security Level	N	M	R	D	DFR
128	189	83	7	8	2^{-134}
192	193	97	8	8	2^{-130}
256	211	97	8	9	2^{-136}

G. Key sizes

a) Private Key Size.: The private key size $|\mathsf{sk}|$ is given by the sum of size of the tuple (F, x, y), that is :

$$|\mathsf{sk}| = (d + 2n) \cdot m \log_2(q) \text{ (basis of } F) \tag{10}$$

Which is the size of F plus the size of x and y. $\log_2 q$ is the size of one element, but we are working with binary elements (so $q = 2$).

b) Public Key Size.: The public key p_k is a single element $h = x^{-1}y$, where $x, y \in \mathbb{F}_{q^m}^n$. Since h is a vector of length n over $\mathbb{F}_{q^m}$, the size of the public key $|\mathsf{pk}|$ is:

$$|\mathsf{pk}| = n \cdot m \cdot \log_2(q) \text{ bits} \tag{11}$$

For typical parameter choices in ROLLO-II (e.g., $q = 2$, $m = 83$, $n = 189$, $d = 8$), we obtain the following sizes.

$$|\mathsf{pk}| = 83{\cdot}189 = 15687 \text{ bits}, \quad |\mathsf{sk}| = (8{+}2{\cdot}189){\cdot}83 = 32038 \text{ bits}$$

These compact key sizes contribute to the overall efficiency of the scheme, particularly in resource-constrained environments, which is a key motivation for code-based post-quantum cryptography. However, storing the random seed would be more memory efficient, as it only requires 40 bytes whether it is 128-bit or 192-bit security levels, or 48 bytes for 256-bit security level [16]. But it is not the main propose of this paper.

c) Cipher Size: The ciphertext is given by:

$$M = (\text{enc},\ \text{cipher}',\ BF(Att_{keys})))$$

Then the total size of M in bits is:

$$|M| = \underbrace{|\text{enc}|}_{=\ell} + \underbrace{|\text{cipher}'|}_{=n\cdot m} + \underbrace{|BF(Att_{keys})|}_{=b_s}$$

Hence, the ciphertext size equation [17] is:

$$\boxed{|M| = \ell + n \cdot m + b_s \quad \text{bits}} \tag{12}$$

- $|\text{enc}| = \ell$ since it is the XOR of the plaintext with a hash of fixed size.
- $|\text{cipher}'| = n \cdot m$ bits, as $\text{cipher}' \in \mathbb{F}_{q^m}^n$.
- The Bloom filter size b_s depends on the number of attribute keys and the desired false-positive rate p. [17] It can be approximated as:

$$b_s \approx -\frac{|\text{Att keys}| \cdot \ln(p)}{(\ln 2)^2} \tag{13}$$

H. Limitations

a) Restricted Policy Expressiveness: The current scheme **only supports conjunctive ("AND") attribute policies**, where decryption requires *all* specified attributes to be satisfied. This excludes more flexible policies such as:

- **Disjunctive ("OR")** access (e.g., "Admin **OR** Auditor")
- **Threshold-based policies:** Access is granted if at least k out of n attributes are satisfied. For example, a policy like $(2 \text{ of } \{A, B, C\})$ means any two attributes among A, B, and C are sufficient.
- **Non-monotonic policies:** These allow negation of attributes (e.g., A $\wedge$ $\neg$B). They support rules that exclude certain attributes, enabling more expressive but more complex policies.

This limitation reduces practicality in scenarios requiring granular access control (e.g., healthcare systems where multiple roles might need access). Future work could integrate **Linear Secret Sharing Schemes (LSSS)** or **monotone span programs** to address this [18].

b) Non-Zero Decryption Failure Rate: The **Rank Support Recovery (RSR)** algorithm may fail with a little probability (Equation 9). While negligible for conservative parameters, this introduces a **probabilistic correctness** guarantee. Applications demanding **deterministic decryption** (e.g., real-time systems) would require post-processing or redundancy mechanisms.

c) Bloom Filter False Positives: The ABE layer's reliance on Bloom filters introduces a false positive rate (Equation 5). While false positives **do not break confidentiality** (decryption still requires the private key), they may:

- **Increase unnecessary computation** for unauthorized users
- **Require careful parameter tuning** (e.g., $m \geq 8n$, $k \approx (m/n) \ln 2$) to maintain $p < 2^{-20}$

d) Parameter Constraints: The RBC library restricts m and n to fixed values (e.g., $m \in \{67, 83, \dots\}$), limiting adaptability. Larger m, n improve security but **increase ciphertext size and computation time**, while smaller values risk **reduced failure tolerance**.

e) Lack of Hardware Acceleration: The current implementation relies on **Advanced Vector Extensions (AVX) optimized CPU code**, leaving **FPGA/ASIC acceleration** unexplored. Critical operations (e.g., finite field inversion in RSR) could benefit from hardware offloading but require RTL co-design.

IV. Acceleration

A. Required accelerations

a) RSR: The RSR algorithm is well-suited for FPGA acceleration due to its reliance on finite field arithmetic and iterative subspace operations. FPGAs exploit fine-grained parallelism to optimize critical bottlenecks:

1) **Finite field inversion** (Step 4) can be implemented via pipelined extended Euclidean algorithms or precomputed lookup tables [19].
2) **Subspace intersections** (Steps 7–8) benefit from custom systolic arrays for low-latency nullspace computation.
3) **Gaussian elimination** (Steps 2,9) can be accelerated through dedicated arithmetic units. [20]

FPGAs prevent memory bottlenecks for small-field operations and enable deterministic latency.

b) Multiplications: Finite field multiplications, particularly in large extension fields like $\mathbb{F}_{q^m}$, are fundamental to most cryptographic primitives. Optimizing these operations, either through vectorization (e.g., AVX instructions) or hardware implementations (e.g., DSP blocks in FPGAs), can yield significant speedups in both encryption and decryption. [21]

c) Inversion: Inversion of field elements or matrices is one of the most computationally expensive operations, especially when performed over extension fields. Accelerating inversion, for instance using hardware units or by reducing the number of inversions required through algorithmic improvements, can contribute substantially to overall performance enhancements. [19]

B. Acceleration methods

a) Advanced Vector Extensions: AVX is an instruction set developed by Intel to enhance the performance of vector computations on CPUs. Introduced with the Sandy Bridge microarchitecture, AVX enables Single Instruction, Multiple Data (SIMD) operations on wide registers (256-bit for AVX, up to 512-bit with AVX-512), making it particularly suitable for parallel data processing such as finite field multiplications used in cryptographic algorithms. In our project, AVX was employed to significantly speed up critical operations such as key generation, encryption, and decryption. [22]

b) Field-Programmable Gate Arrays: FPGAs are configurable hardware components that enable the implementation of custom digital architectures after manufacturing.

In the context of cryptographic acceleration, FPGAs offer a significant advantage due to their capacity for massive parallelization of operations. They are particularly well-suited for tasks involving intensive arithmetic computations such as multiplications, additions, and other operations that can be structured as pipelines or parallel executions. [23] The critical operations identified are the computation of Rank Support Recovery (RSR) and the Bloom filter, the latter relying primarily on hash functions and modulo operations. These types of computations are highly amenable to FPGA implementation, as they can be parallelized and optimized in hardware to reduce latency and increase throughput.

Moreover, the use of a hardware accelerator, whether FPGA- or ASIC-based, also enables a reduction in energy consumption per operation, which is a key concern in embedded systems and large-scale infrastructures. [6]

C. Performance

a) Benchmark Configuration: All benchmarks were performed over `1000` iterations on the following system:

- **CPU:** AMD Ryzen 7 8845HS
- **RAM:** 32 GB LPDDR5X-6400MT
- **Architecture:** 64-bit
- **Operating System:** Debian 12

b) Benchmark Results: We evaluated the performance of our implementation in terms of CPU cycles (in millions) for key generation, encryption, and decryption. Results are given for both the standard (as shown in Table III) and the AVX-optimized versions (as shown in Table IV).

TABLE III
STANDARD IMPLEMENTATION PERFORMANCE (IN MILLIONS OF CPU CYCLES)

Security Level	Keygen	Encrypt	Decrypt
128	182.2	16.2	194.9
192	212.7	18.1	227.7
256	254.4	19.5	269.0

TABLE IV
AVX-OPTIMIZED IMPLEMENTATION PERFORMANCE (IN MILLIONS OF CPU CYCLES)

Security Level	Keygen	Encrypt	Decrypt
128	8.1	7.8	12.8
192	7.3	8.1	12.3
256	7.9	7.7	13.8

c) Comparison: The AVX-optimized version shows a drastic improvement in performance, especially during key generation, where the number of CPU cycles was reduced by more than 20 times for the 128-bit security level. Similar performance gains are observed across encryption and decryption steps, making the AVX-enhanced implementation highly better than the original implementation.

D. Our achievements

a) : During our project, we focused on optimizing the software implementation using AVX (Advanced Vector Extensions) instructions to significantly accelerate cryptographic operations on standard CPUs.

b) : Due to time constraints and the complexity of hardware integration, we did not implement an FPGA-based acceleration. However, we studied the feasibility of such an approach and proposed several strategies for future hardware deployment, as detailed in the next section.

E. FPGA acceleration suggestions

a) Time Cost of Offloading Computations to FPGA: Offloading cryptographic computations to FPGAs can lead to significant performance gains and cost savings. For instance, a study comparing FPGA and GPU accelerations on AWS found that FPGA implementations performed nearly 14 times the work of GPU per dollar cost. Additionally, FPGAs consumed less power, with measurements indicating an average power consumption of 10W when idle and up to 19W under load, compared to 250W for GPUs. [24]

b) Designing Specific Units: Designing dedicated hardware units tailored to specific cryptographic operations can enhance performance. For example, implementing a point multiplier based on binary fields with reconfigurable key lengths allowed for effective resource utilization and improved efficiency in elliptic curve cryptography operations. [25]

c) Parallelization: FPGAs inherently support parallel processing, making them well-suited for accelerating cryptographic algorithms. By leveraging their parallel structures, FPGAs can provide substantial acceleration for compute-intensive tasks, encryption/decryption processes.

d) Other Methods: Additional methods to optimize FPGA-based cryptographic acceleration include:

- **Algorithm Agility**: FPGA's reprogrammable nature enables cryptographic algorithms updates post-deployment, enhancing flexibility and future-proofing systems.
- **Partial Reconfiguration**: This technique enables dynamic updates to portions of the FPGA without halting the entire system, insuring efficient resource utilization and adaptability.
- **Hardware-Software Co-Design**: Integrating FPGA acceleration with software components can optimize performance and resource allocation, as demonstrated in homomorphic encryption co-processors. [26]

V. Security considerations of the presented scheme

A. Limitations of Current Analysis

No formal security proof: Unlike standardized ABE schemes, our construction lacks a reductionist security proof (e.g., IND-CPA/CCA under RSD hardness). This means:

- Security relies on heuristic arguments about the composition of ROLLO-II and Bloom filters
- No theoretical guarantees against adaptive attacks
- Security level claims are provisional pending formal analysis

B. Composition Risks

The hybrid design combining ROLLO-II with attribute-based Bloom filters introduces new attack surfaces:

- **Attribute-hiding weakness:** While ROLLO-II provides message confidentiality, the Bloom filter leaks policy information through hash collisions
- **Key reuse vulnerability:** The same ROLLO-II key pair is used across all ciphertexts - compromising one attribute set could weaken others
- **Interaction attacks:** No formal study of how RSR failures (Equation 9) interact with Bloom filter false positives

C. Practical Attack Vectors

Even without quantum computers, several threats emerge:

1) **Support recovery attacks:** An adversary with many ciphertexts could statistically reconstruct the private support F.
2) **Syndrome manipulation:** Carefully crafted ciphertexts might trigger RSR failures to leak key material.
3) **Attribute probing:** Observing decryption attempts reveals which Bloom filter bits correspond to specific attributes.

D. Recommendations for Future Security Analysis

To properly evaluate the scheme's security, we recommend:

1) **Formal reduction proofs:** Demonstrate equivalence to RSD or other hard problems.
2) **Side-channel analysis:** Especially for the FPGA implementation.
3) **Composition theorems:** Adapt frameworks like Universal Composability to analyze the ABE layer.

E. Security Status Summary

TABLE V
Security Properties Overview

Component	Security Guarantee	Major Threats
ROLLO-II core	Heuristic (RSD) [17]	Support recovery
ABE layer	None (novel)	Attribute probing
Composition	Untested	Interaction attacks

- This honest assessment highlights that while the scheme shows promise, its security depends fundamentally on future theoretical work

- The experimental results should be interpreted as performance benchmarks rather than security validation

VI. Conclusion

This work presents a novel attribute-based encryption scheme based on ROLLO and Low-Rank Parity Check (LRPC) codes, resulting in a post-quantum construction that leverages rank metric cryptography for strong security guarantees. Our implementation uses the RBC library and employs AVX vectorization to optimize key generation, encryption, and decryption operations, achieving significant performance gains on standard CPUs. While these software-level optimizations yield promising results, the scheme remains computationally intensive and would benefit from hardware acceleration—particularly for rank support recovery and finite field arithmetic—to enable practical deployment in performance-sensitive environments.

Although our current implementation focuses on software optimization, we identified critical computational bottlenecks—such as Rank Support Recovery (RSR) and finite field arithmetic—that could benefit from FPGA acceleration in future work. The integration of hardware acceleration could further enhance performance, making the scheme more viable for real-world applications in cloud computing and IoT environments where resource constraints are a concern.

While, several challenges remain. The scheme currently supports only conjunctive ("AND") policies, limiting its expressiveness compared to more advanced ABE constructions. Additionally, the probabilistic nature of Bloom filters introduces a small but non-negligible false positive rate, which must be carefully managed in security-sensitive deployments. Finally, while rank-based cryptography offers strong post-quantum security guarantees, a formal security analysis of our composed scheme (combining ROLLO-II with ABE) is still needed to validate its resilience against adaptive attacks.

Future research directions include:

- **Extending Policy Support:** Implementing threshold-based or disjunctive policies using Linear Secret Sharing Schemes (LSSS).
- **Hardware Acceleration:** Developing FPGA-optimized modules for RSR and finite field operations to further reduce latency.
- **Security Proofs:** Conducting a rigorous security analysis to establish formal guarantees under chosen-ciphertext attacks.

Our work contributes to the growing body of research on practical post-quantum ABE schemes, offering a balance between security, efficiency, and fine-grained access control. As quantum computing advances, such constructions will play a crucial role in securing next-generation distributed systems.

Acknowledgments

The authors would like to express their sincere gratitude to Dr. Vincent Migliore (INSA Toulouse / LAAS-CNRS) for his invaluable guidance, insightful feedback, and continuous support throughout this research.

The authors also wish to thank Mary-Ann Flannery-Meyer for her helpful feedback on the English language, as well as her invaluable advice and suggestions that contributed to improving the overall quality of this paper.

The authors also thank the developers of the RBC (Rank-Based Cryptography) library for providing a valuable and well-documented implementation framework that supported our experimental work.

The authors also acknowledge the use of OpenAI's ChatGPT and DeepSeek during the preparation of this paper. The tool was employed to assist with initial proofreading, reformulating certain technical explanations for clarity, and enhancing language consistency. All scientific content, results, and conclusions remain the sole responsibility of the authors.

References

[1] L. Chen, S. Jordan, Y.-K. Liu, D. Moody, R. Peralta, R. Perlner, and D. Smith-Tone, "Report on Post-Quantum Cryptography," Tech. Rep. NIST IR 8105, National Institute of Standards and Technology, Apr. 2016.

[2] T. D. P. Bai, S. A. Rabara, and A. V. Jerald, "Elliptic Curve Cryptography based Security Framework for Internet of Things and Cloud Computing," *InternatIonal Journal of Computer Science and technology*, vol. 6, no. 3, 2015.

[3] A. Simon Francia, J. Solis-Lastra, and E. A. Papa Quiroz, "Elliptic Curves Cryptography for Lightweight Devices in IoT Systems," in *Emerging Research in Intelligent Systems* (M. Botto-Tobar, H. Cruz, A. Díaz Cadena, and B. Durakovic, eds.), vol. 405, pp. 71–82, Cham: Springer International Publishing, 2022. Series Title: Lecture Notes in Networks and Systems.

[4] A. Samanta, A. Maitra, and S. S. Chaudhury, "Proposal for Quantum Ciphertext-Policy Attribute-Based Encryption," Mar. 2022. arXiv:2203.10888.

[5] H. Li, Y. Tang, Z. Que, and J. Zhang, "FPGA Accelerated Post-Quantum Cryptography," *IEEE Transactions on Nanotechnology*, vol. PP, pp. 1–7, Jan. 2022.

[6] M. M. Abdel-Aziz and A. T. Abdel-Hamid, "Hardware low power implementation of attribute-based encryption," in *2016 28th International Conference on Microelectronics (ICM)*, (Giza, Egypt), pp. 273–276, IEEE, Dec. 2016.

[7] Y. Su, B. Yang, C. Yang, and L. Tian, "FPGA-Based Hardware Accelerator for Leveled Ring-LWE Fully Homomorphic Encryption," *IEEE Access*, vol. 8, pp. 168008–168025, 2020. Conference Name: IEEE Access.

[8] V. Yousefipoor and T. Eghlidos, "An Efficient Post-Quantum Attribute-Based Encryption Scheme Based on Rank Metric Codes for Cloud Computing," *IEEE Access*, vol. 11, pp. 99990–100000, 2023. Conference Name: IEEE Access.

[9] H. Bartz, L. Holzbaur, H. Liu, S. Puchinger, J. Renner, and A. Wachter-Zeh, "Rank-Metric Codes and Their Applications," Mar. 2022. arXiv:2203.12384 [cs].

[10] P. Gaborit, O. Ruatta, J. Schreck, J.-P. Tillich, and G. Zémor, "Rank based Cryptography: a credible post-quantum alternative to classical cryptography," in *NIST 2015: Workshop on Cybersecurity in a Post-Quantum World 2015*, (Gaithersburg, United States), pp. 1–13, Apr. 2015.

[11] A. K. Yazbek, J.-P. Cances, and V. Meghdadi, "Les codes Low Rank Parity Check (LRPC)." working paper or preprint, Aug. 2017.

[12] "RBC Lib." https://rbc-lib.org/.

[13] M. Ripeanu and A. Lamnitchi, "(PDF) Bloom Filters – Short Tutorial," 2001.

[14] A. Cheriere, L. Mortajine, T. Richmond, and N. E. Mrabet, "Exploiting ROLLO's constant-time implementations with a single-trace analysis," *Designs, Codes and Cryptography*, vol. Coding and Cryptography 2022, p. 1, 2023.

[15] C. Aguilar Melchor, N. Aragon, S. Bettaieb, O. Bidoux, O. Blazy, J.-C. Deneuville, P. Gaborit, and G. Zémor, "Rank-ouroboros, lake & locker - rollo," tech. rep., PQCRYPTO, 2020. Specification document detailing ROLLO-I/II parameters and key size equations.

[16] C. e. a. Aguilar Melchor, "Optimized implementation of the nist pqc submission rollo," in *Post-Quantum Cryptography*, pp. 1–25, 2019. Discusses seed-based private key storage (40/48 bytes) vs full tuple storage.

[17] M. Lequesne and V. Mateu, "Exploiting rollo's constant-time implementations with a single-trace analysis," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2021, no. 2, pp. 1–25, 2021. Analyzes ROLLO-II parameters and ciphertext structure.

[18] A. Lewko, T. Okamoto, A. Sahai, K. Takashima, and B. Waters, "Fully Secure Functional Encryption: Attribute-Based Encryption and (Hierarchical) Inner Product Encryption," in *Advances in Cryptology – EUROCRYPT 2010* (H. Gilbert, ed.), (Berlin, Heidelberg), pp. 62–91, Springer, 2010.

[19] P. K. G. Nadikuda and L. Boppana, "An area-efficient architecture for finite field inversion over GF(2m) using polynomial basis," *Microprocessors and Microsystems*, vol. 89, p. 104439, Mar. 2022.

[20] D. Rui, N. Horacio, and V. Mário P., "(PDF) Double-precision Gauss-Jordan Algorithm with Partial Pivoting on FPGAs," in *ResearchGate*, 2009.

[21] S. C. Oliva Madrigal, G. Saldamlı, C. Li, Y. Geng, J. Tian, Z. Wang, and C. K. Koç, "Reduction-Free Multiplication for Finite Fields and Polynomial Rings," in *Arithmetic of Finite Fields* (S. Mesnager and Z. Zhou, eds.), (Cham), pp. 53–78, Springer International Publishing, 2023.

[22] Intel Corporation, "Manuals for intel® 64 and IA-32 architectures." https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html.

[23] I. Skliarova and V. Sklyarov, *FPGA-based hardware accelerators*. Lecture Notes in Electrical Engineering 566, Cham: Springer, 2019.

[24] T. Furkan, *Research Portal - FPGA Accelerators for Cryptography and Their Protection in the Cloud*. PhD thesis, Faculty of Engineering Science - KU Leuven, Oct. 2022.

[25] M. R. Hossain, M. S. Rahman, K. S. Zaman, W. E. Fezzani, M. A. S. Bhuiyan, C. C. Kang, T. J. Yew, and M. H. Miraz, "Low latency FPGA implementation of twisted Edward curve cryptography hardware accelerator over prime field," *Scientific Reports*, vol. 15, p. 15097, Apr. 2025. Publisher: Nature Publishing Group.

[26] S. Baskaran and P. Rajalakshmi, "Hardware-software co-design of AES on FPGA | Proceedings of the International Conference on Advances in Computing, Communications and Informatics," Aug. 2012.