Trends in Computational and Applied Mathematics, **25** (2024), e01772
Sociedade Brasileira de Matemática Aplicada e Computacional
Online version ISSN 2676-0029 www.scielo.br/tcam
ORIGINAL ARTICLE
doi: 10.5540/tcam.2024.025.e01772

The application of homomorphism in cryptography

A. L. Z. LUNKES[1] and F. BORGES[2]

Received on August 1, 2023 / Accepted on September 27, 2024

ABSTRACT. The increasing use of new technologies and the internet underscores the need to ensure users' privacy and security while browsing the web. To achieve this, encryption schemes rely on sharing keys among parties involved in message exchange. The concept of homomorphism, crucial in both mathematics and cryptography, provides a solution by enabling computations on encrypted data without the need for decryption. Furthermore, the RSA and ElGamal algorithms exemplify how homomorphism is applied in practice, each with its specific methods for maintaining data security and privacy during cryptographic processing.

Keywords: cryptography, homomorphism, computing, privacy, security.

1 INTRODUCTION

To ensure the security and privacy of information in the digital age, cryptography has become an essential tool. With the increasing use of technology in both industry and daily life, and the demand for applications that facilitate social network connectivity, banking transactions, and file storage on mobile devices and computers, there is a growing concern for data security and privacy. Thus, in this work, we introduce the concept of Homomorphic Encryption (HE).

Cryptography employs computational and mathematical techniques to transform information into codes, ensuring secure communication in the presence of third parties, commonly referred to as attackers or Eve, or in cloud computing scenarios where certain computable functions are performed on data while preserving the characteristics of the function used and the format of the encrypted data. Furthermore, there is a need to compute these data using operations such as addition and multiplication. In some schemes, only addition or multiplication operations are supported, as seen in algorithms like Paillier and RSA, respectively. Understanding the concept and applications of homomorphism is crucial for utilizing HE effectively.

[1]Laboratório Nacional de Computação Científica, SCP, Av. Getúlio Vargas, 333, 25651-075, Petrópolis, RJ Brasil – E-mail: alunkes@lncc.br https://orcid.org/0000-0002-1846-4853
[2]Laboratório Nacional de Computação Científica, SCP, Av. Getúlio Vargas, 333, 25651-075, Petrópolis, RJ, Brasil – E-mail: borges@lncc.br http://orcid.org/0000-0001-5159-9517

Furthermore, a HE scheme consists of four algorithms: 1) Key Generation for public and private keys; 2) Encryption $E(m)$; 3) Decryption $D(E(m))$; and 4) Homomorphism. The key generation process returns a pair of public and private keys for the asymmetric version (known as Public Key Cryptography) or a single key for the symmetric version.

The homomorphism algorithm is a specific operation in HE and was introduced by the authors [10]. Taking as input the ciphertexts of messages m_1 and m_2, $E(m_1)$ and $E(m_2)$, it outputs the ciphertext of a function f applied to m_1 and m_2, $E(f(m_1,m_2))$. Thus, the operation f is performed on the messages without the need for prior decryption.

One important point in HE is that the format of encrypted messages, after the homomorphic process, must be preserved to ensure correct decryption. In other words, the definition of homomorphism guarantees this preservation. Additionally, a logarithmic growth in ciphertext size is allowed with the number of homomorphic operations performed, as discussed in [5], often referred to as circuit depth.

Furthermore, the terms Client-Server are commonly used in computing. In this work, we denote them as Alice and Bob, respectively, as discussed in [11]. The Client is the entity that wishes to send messages or data to the Server but does not share any of its resources with the Server. The Server processes or stores the data and, when requested by the Client, performs certain functions and returns results to the Client. In HE, the Client sends encrypted data to the Server, and the Server does not decrypt the data; it only computes what the Client desires.

An example of Public Key Cryptography is digital signatures, where we can use the multiplicative homomorphic property of the RSA algorithm. Alice (Client) wishes to sign a document sent by Bob (Server) using Alice's public key, as discussed in [1]. Since Alice holds the private key for signing, her encrypted signature is decrypted using the public key, ensuring its authenticity in the signature.

There are HE algorithms that are additive homomorphic and are used in electronic voting, such as the Paillier algorithms discussed in [8], and Damgård-Jurik[1]. Additionally, there is the Fully Homomorphic Encryption (FHE) scheme, which allows performing both additive and multiplicative homomorphic operations in a single algorithm, such as the NTRU algorithm detailed in [7].

The aim of this work is to demonstrate that the concept of homomorphism operates similarly in both mathematics and cryptography. Additionally, we will explore two widely used homomorphic encryption algorithms: RSA and ElGamal. These algorithms are crucial for ensuring security and privacy in the exchange of sensitive information over insecure channels.

This work is structured as follows: In Section 2, we introduce the concept of homomorphism. In Section 3, we discuss homomorphic encryption algorithms, presenting their characteristics and operational principles. We then illustrate these concepts with practical examples of the RSA and ElGamal algorithms. Finally, in Section 4, we conclude the work, highlighting the importance and future challenges in the field of homomorphic cryptography.

[1] Available at: `http://security.hsr.ch/msevote/damgardjurik`

2 HOMOMORPHISM

In this section, we discuss the concept of Homomorphism, which will be used throughout this work. A homomorphism between two algebraic structures of the same type is a mapping that preserves these structures. For example, the structures can be groups, rings, or vector spaces, each with their respective operations. If G and H are non-empty sets with addition operation $+$ defined on both structures, then a homomorphism f from G H, for any $x, y \in G$, satisfies

$$f(x+y) = f(x) + f(y).$$

Now, if G and H are both rings with an additional multiplication operation $\cdot$, then, in addition to the previous condition, the homomorphism f also satisfies

$$f(x \cdot y) = f(x) \cdot f(y)$$

for any x, $y \in G$. Let us present two examples of homomorphisms, the first associated with rings, and the second with groups.

Example 2.1. *Let G, H be any two rings with operations $+, \cdot$. Consider the function $f : G \to H$ defined by $f(x) = 0_H$ for any $x \in G$, where 0_H is the additive identity element in H. Then, clearly, this function is a homomorphism.*

One application of homomorphism is its use to ensure that two rings are isomorphic to each other, meaning they have equivalent operations, and the homomorphism between them is also a bijection. Thus, if $f : A \to B$ is a ring homomorphism, the image of f as a subring of B is isomorphic to the quotient ring $A/\ker(f)$, where $\ker(f)$ is the kernel of f. More information about this result and its consequences can be found in [6] and [4].

Example 2.2. *Let G be a non-empty group with operation $\cdot$. Let $g \in G$ be a fixed element G, and define the function $\varphi : G \to G$ by $\varphi(x) = g \cdot x \cdot g^{-1}$, for any $x \in G$.*

Let $x, y \in G$ be any two elements. Then, by associativity and the identity element of the group, we have

$$\varphi(x) \cdot \varphi(y) = (g \cdot x \cdot g^{-1}) \cdot (g \cdot y \cdot g^{-1}) = g \cdot (x \cdot y) \cdot g^{-1} = \varphi(x \cdot y).$$

Therefore, φ is a Homomorphism.

3 HOMOMORPHIC ENCRYPTION

In this section, we will demonstrate the application of cryptography. Due to the high demand for network data storage, a secure system is necessary to protect this data, and cryptography plays a crucial role. Additionally, a homomorphic encryption scheme consists of four algorithms: 1) KeyGen (key generation); 2) Encrypt (encryption) denoted by $E(m)$; 3) Decrypt (decryption) denoted by $D(E(m))$; and 4) Homomorphism (homomorphism), specifically

1. **KeyGen** where we generate the public and private keys. These keys are used to encrypt and decrypt messages or data between the client and the server. The public key is known to everyone, while the private key is known only to the owner. In key generation for digital signatures, two functions are performed: authentication and encryption. In message authentication, a cryptographic hash function is used, which accepts inputs of arbitrary sizes, potentially up to gigabytes, and after encryption, produces standardized output values of fixed length.

   To access the message and produce a summary, finding a hash value can be challenging due to its one-way nature.

   For encryption, it utilizes the digest, which is the output of the hash function, and the private key to generate the digital signature. To verify the authenticity of the signature, software performs the following checks: 1) computes the hash function of the message; 2) decrypts the signature using the signer's public key; and 3) compares the computed result with the deciphered one.

2. **Encryption** denoted by $E(m)$, denoted by m is the message to be encrypted or the data to be concealed. In this algorithm, we use the public key.

3. **Decryption** denoted by $D(E(m))$, denoted by $E(m)$ is a message to be decrypted using the private key.

4. **Homomorphism** is a specific application of HE, which converts encrypted messages into encrypted messages, preserving the operations performed.

Given the messages m_1 and m_2, $\square$ denotes an operation to be performed, which can be additive or multiplicative. For the encrypted messages $E(m_1)\square E(m_2)$, then the homomorphic operations are

$$E(m_1)+E(m_2)=E(m_1+m_2),$$
$$E(m_1)\times E(m_2)=E(m_1\times m_2).$$

In Figure 1, we have a physical problem that needs to be solved. The client has some data about the problem and uses their public key to encrypt this data, sending it to the server. The server receives and stores the encrypted data, performs homomorphic processing on the encrypted data, and returns the solution to the client. The client then uses their private key to correctly decrypt the data.n Figure 1, we have a physical problem that needs to be solved. The client has some data about the problem and uses their public key to encrypt this data, sending it to the server. The server receives and stores the encrypted data, performs homomorphic processing on the encrypted data, and returns the solution to the client. The client then uses their private key to correctly decrypt the data.

In an example of cryptography, we have Alice as the Client who wants to send a message to Bob, the Server. To facilitate this exchange securely, public and private keys are utilized. A pair of

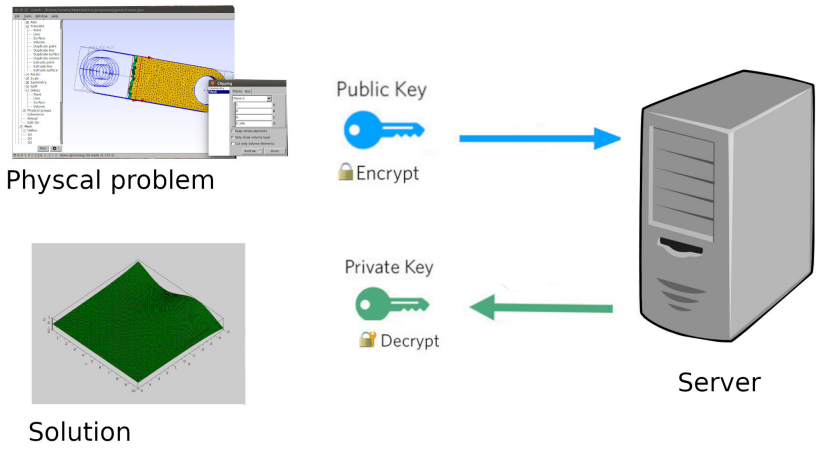

Figure 1: HE-client-server scenario.

public keys (accessible to any user) and private keys (accessible only to their respective owner) are generated for both parties.

Alice uses HE to encrypt the message with Bob's public key and sends it to him. Bob, upon receiving the encrypted message, decrypts it using his private key. Importantly, if there is an eavesdropper (Eve) intercepting communication between Alice and Bob over an insecure channel, Eve cannot decipher the message because she lacks access to Bob's private key. This ensures the confidentiality of the communication between Alice and Bob. In the example described, homomorphism plays a crucial role in ensuring secure message exchange using HE.

After Alice encrypts the message m with Bob's public key, denoted as $E(m)$, she sends this encrypted message to Bob. Bob then applies a homomorphism f on the encrypted message $E(m)$ without needing to know the original message m, resulting in $f(E(m))$. This operation preserves the structure and format required for the message to be correctly decrypted by Bob using his private key. In Figure 2, illustrates how this process unfolds, demonstrating the application of homomorphism in maintaining the integrity and security of the encrypted communication between Alice and Bob.

In Figure 3, Alice and Dave aim to exchange a message through an insecure channel. The attacker (Eve) intercepts their communication but can only access encrypted messages, rendering her attempts unsuccessful in decrypting the content.

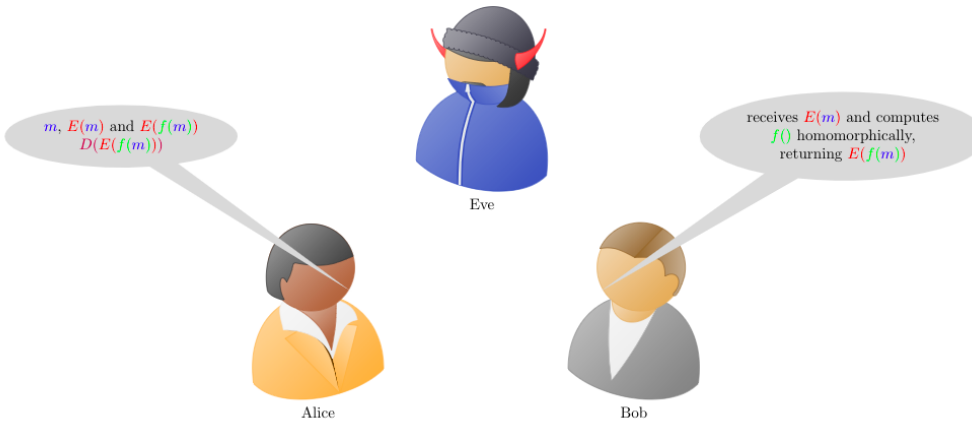

Figure 2: Homomorphism in the HE scheme.

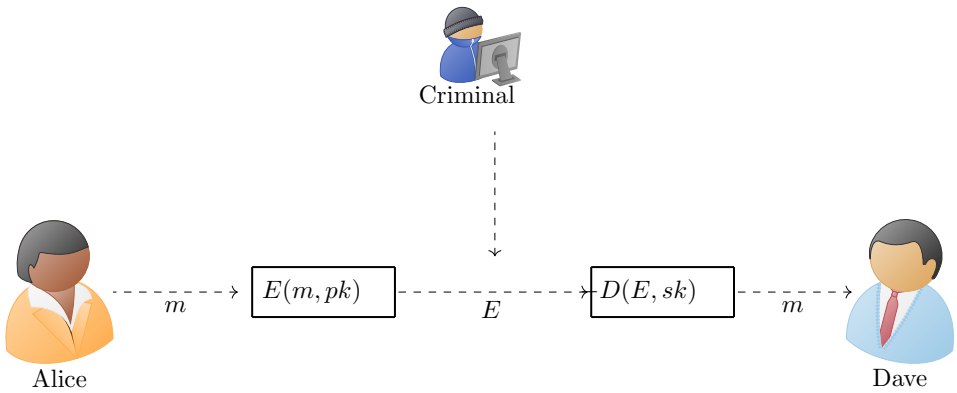

Figure 3: Secure communication scenario between Alice, Bob, and Dave.

In Figure 4, Alice and Bob are attempting to securely send a message to Dave over an insecure channel. They achieve this by transmitting encrypted messages, which Dave receives and decrypts using his secret key. Additionally, Dave has access to the homomorphic sum of the messages m_1 and m_2. This allows Dave to perform computations on the encrypted data without decrypting it first, maintaining the security of the communication while enabling meaningful operations on the encrypted content.

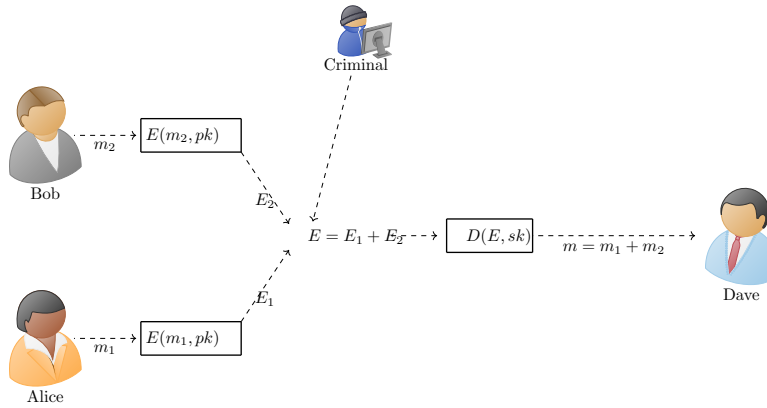

Figure 4: Secure communication scenario between Alice, Bob, and Dave.

Let n be an integer with $n > 1$. So the modular ring $\mathbb{Z}_n$ inherits some properties of $\mathbb{Z}$, as it is a quotient ring from $\mathbb{Z}$. Consider the set U_n formed by the units of $\mathbb{Z}_n$, defined as $U_n = \{x \in \mathbb{Z}_n | x \text{ has a multiplicative inverse in } \mathbb{Z}_n\}$, where (x,n) denotes the greatest common divisor of x and n. The set U_n can also be expressed as $U_n = \{x \in \mathbb{Z}_n | (x,n) = 1\}$, and that U_n forms a multiplicative group under modulo n multiplication. The order of U_n is denoted in general by $\varphi(n)$, where φ function is called Euler's function.

The following results present properties of Euler's function, and their proofs can be found in [6]. One of the fundamental results is known as Fermat's Little Theorem.

Theorem 1 (Fermat's Little Theorem). *Let p be a prime number and suppose $x \in \mathbb{Z}$ satisfies $(x,p) = 1$. Then, $x^{p-1} \equiv 1 \pmod{p}$.*

Theorem 2. *Suppose p and q are distinct primes, $n = pq$, and $m = \varphi(n) = (p-1)(q-1)$ is Euler's totient function. If a and b are integers such that $ab \equiv 1 \pmod{m}$, then $x^{ab} \equiv x \pmod{n}$ for all $x \in \mathbb{Z}$.*

3.1 RSA

The RSA algorithm, based on the integer factorization problem of products of two large prime numbers [10], is the most renowned asymmetric public-key cryptographic system. Due to its relative slowness, it is not recommended for users to use it directly to encrypt data, but rather to transmit encrypted cryptographic keys that will be used in symmetric algorithms. These symmetric algorithms allow for faster encryption and decryption processes. Below is a description of the RSA algorithm:

> **KeyGen:** let p and q be randomly generated prime numbers. Define $n = pq$ and $\varphi(n) = (p-1)(q-1)$. Choose e such that $\text{mdc}(e, \varphi(n)) = 1$. By Euclid's algorithm, compute d

as the modular inverse of e modulo $\varphi(n)$, denoted as $d = e^{-1} \mod \varphi(n)$. Thus, we derive the corresponding keys for this RSA scheme:

The public key: (e,n).

The secret key: (d,n).

Encryption: consider m the message to be encrypted, and M the set of all messages to be encrypted. To encrypt m, we use the public key pair (e,n):

$$E(m) = m^e \mod n, \ \forall\, m \in M.$$

where:

$E(m)$ is the encrypted ciphertext,

e is the public exponent,

n is the modulus.

This modular exponentiation operation $m^e \mod n$ ensures that m is encrypted into c using the public key pair (e,n).

Decryption: To retrieve the original message m from the ciphertext $E(m)$, use the secret key pair (d,n) as follows:

$$D(E(m)) = E(m)^d \mod n = m.$$

Homomorphism: For RSA encryption, if c_1 and c_2 are the encrypted ciphertexts of messages m_1 and m_2, respectively, using the public key pair (e,n), then the homomorphic property holds:

$$\begin{aligned} E(m_1) \cdot E(m_2) &= (m_1^e \mod n) \cdot (m_2^e \mod n) \\ &= (m_1 \cdot m_2)^e \mod n \\ &= E(m_1 \cdot m_2). \end{aligned}$$

where $E_{m_1 \cdot m_2}$ is the encrypted ciphertext of $m_1 \cdot m_2$, and $\cdot$ denotes multiplication in the plaintext domain. However, RSA does not support homomorphic addition of ciphertexts. In other words, for encrypted messages E_{m_1} and E_{m_2}:

$$E_{m_1+m_2} \neq E_{m_1} + E_{m_2} \mod n.$$

Example 3.1. *Let us consider that Alice wants to send a message "CNMAC" to Bob. To ensure the message's security, we need to generate keys for encrypting and decrypting the message correctly.*

*For **KeyGen:***

1. *Randomly choose two large prime numbers, $p = 11$ and $q = 23$;*
2. *we calculate $n = p \cdot q = 11 \cdot 23 = 253$, and the Euler function $\varphi(n) = (p-1)(q-1) = (11-1)(23-1) = 220$;*
3. *Choose an integer e such that, $1 < e < \varphi(n)$, so that e and e is coprime with $\varphi(n)$. For example, select $e = 7$, because $mdc(e, \varphi(n)) = (7, 220) = 1$;*
4. *Use Euclid's algorithm to compute $d = e^{-1} \mod \varphi(n)$, where d is the multiplicative inverse of e $\mod \varphi(n)$. For instance, calculate $d = 7 \mod 220$, see Table 1.*

Table 1: Table for calculating the Euclid's Algorithm.

Line	Q	R	U	V
-1	-	220	1	0
0	-	7	0	1
1	31	3	1	-31
2	2	1	-2	63

Therefore, $220(r) + 7(d) = 1 \longrightarrow 220(-2) + 7(63) = 1 \longrightarrow d = 63$. Thus, the public key is $(n, e) \longrightarrow (253, 7)$, and the private key is the pair $(63, 253)$.

It seems you want to describe a bijection $\sigma : L \longrightarrow \mathbb{Z}_{26}$, where $L = A, B, C, \cdot, X, Y, Z$ represents the letters of the alphabet, and represents integers modulo 26. The function σ maps each letter to a corresponding integer as follows: $\sigma(A) = 0, \sigma(B) = 1, \ldots, \sigma(Z) = 25$, to transform letters into whole numbers. The correspondences for each σ value are listed in the Figure 5.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

Figure 5: Message matching Table.

***Encryption:** Based on the Figure 5, we will encode our message as follows, using the provided mappings of letters to numbers.*

Table 2: Message exchange Table.

C	N	M	A	C
2	13	12	0	2

To encrypt the message, where $n-1$ is given, we use Alice's public key and perform a modular potentiation:

$$E(m) \equiv (CNMAC)^7 \mod 253.$$

The message is split into blocks to be transmitted over an insecure channel to Bob.

$$\begin{cases} 2^7 & = 128 \mod 253, \\ 13^7 & = 216 \mod 253, \\ 12^7 & = 177 \mod 253, \\ 0^7 & = 0 \mod 253, \\ 2^7 & = 128 \mod 253. \end{cases} \tag{3.1}$$

The encrypted message is $\{128,216,177,0,128\}$. *Using Figure 5, we can decipher this message as* $E(m) = YIVAY$. *To decrypt the message using the private key, we perform another modular potentiation:*

$$E(m)^{63} \equiv (CNMAC) \mod 253.$$

Hence,

$$\begin{cases} 128^{63} & = 2 \mod 253, \\ 216^{63} & = 13 \mod 253, \\ 177^{63} & = 12 \mod 253, \\ 0^{63} & = 0 \mod 253, \\ 128^{63} & = 2 \mod 253. \end{cases} \tag{3.2}$$

Decryption: *message is* $\{2,13,12,0,2\}$, *using Figure 5, we find that m =CNMAC. To apply homomorphism, let us consider the messages* $m_1 = CNMAC$ *and* $m_2 = 2022 = CACC$ *(where we replace numbers by corresponding alphabet letters as per Figure 5). Using the RSA algorithm,*

$$\begin{aligned} E(m_1) \cdot E(m_2) &= ((CNMAC)^7 \mod 253)) \cdot ((CACC)^7 \mod 253) \\ &= ((CNMAC) \cdot (CACC))^7 \mod 253 \\ &= E((CNMAC) \cdot (CACC)). \end{aligned}$$

If the RSA algorithm allows additive homomorphism, then we can compute the encrypted sum $E(m_1) + E(m_2)$ *as follows:*

$$\begin{aligned} E(m_1) \cdot E(m_2) &= ((CNMAC)^7 \mod 253)) + ((CACC)^7 \mod 253) \\ &= ((CNMAC) + (CACC))^7 \mod 253 \\ &= E((CNMAC) + (CACC)). \end{aligned}$$

3.2 ElGamal

ElGamal encryption is an asymmetric public key encryption algorithm that ensures message confidentiality and supports digital signatures, providing authentication and integrity verification. Its security stems from the complexity of solving the Discrete Logarithm Problem in finite fields

or groups. This feature allows secure communication over insecure channels, ensuring messages can be authenticated without compromising their confidentiality.

Moreover, ElGamal encryption supports multiplicative homomorphism, akin to RSA. In the ElGamal scheme ([3]), which relies on the difficulty of computing discrete logarithms, this property enables operations on plaintexts before encryption to correspond with operations on their resulting ciphertexts. This capability is valuable in scenarios where computations must be conducted on encrypted data without the need for prior decryption, preserving data privacy throughout processing.

KeyGen: let g be a generator of the cyclic group $\mathbb{G}$ module p, where p and q are prime numbers such that $n = pq$. In a cyclic group, the generator g allows us to generate all elements of the group. Consider $h = g^x \mod p$, where x is a randomly chosen from the set $\{1,2,\ldots,p-1\}$. Thus, we obtain the corresponding keys for this scheme:

The public key: (p,g,h).

The secret key: x.

This scheme utilizes the properties of cyclic groups and the computational complexity of the Discrete Logarithm Problem for secure key generation and exchange.

Encrypt: a message m is encrypted using g and $y \in \mathbb{Z}_p$, resulting in a pair of encrypted messages, so

$$E(m) = (g^y, mh^y) \mod p = (g^y, mg^{xy}) = (E_1(m), E_2(m)) \mod p.$$

Decrypt: let $s = h^{-y}$, then

$$D(E(m)) = E_2(m) \cdot s \mod p = mg^{xy} \times g^{-xy} \mod p = m.$$

Homomorphism:

$$\begin{aligned} E(m_1) \times E(m_2) &= (g^{x_1}, m_1 h^{x_1}) \mod n \times (g^{x_2}, m_2 h^{x_2}) \mod n \\ &= (g^{x_1+x_2}, (m_1 \times m_2) h^{x_1+x_2}) \mod n. \end{aligned}$$

Furthermore, this method supports only multiplicative operations.

Example 3.2. *Let us consider that Alice wants to send a message to Bob, where the message m is "CNMAC". Therefore, we need to generate the keys to encrypt and decrypt the message correctly.*

*For **KeyGen:***

1. *Randomly choose a large prime number,* $p = 13$*;*
2. *Randomly choose an integer x, such that* $1 \leq x \leq p-2,\ x = 7$*;*
3. *Calculates the public key:* $h = g^x \mod p = 2^7 \mod 13 = 11$.

The public key is given by $(p,g,h) \longrightarrow (13,2,11)$*, and the private key* (x) *is* (7).

Then, since we have converted the letters into integers for the RSA algorithm, we will follow the same process. The mappings for each σ *value are listed in Figure 5 and Figure 2.*

To encrypt the message, where $n-1$*, we use Alice's public key. We choose another random integer* $y = 5$ *such that* $1 \leq y \leq p-2$*, and perform a modular potentiation,*

$$E(m) \equiv (2^5, (0213120002) \cdot 11^5) \mod 13 \equiv (E_1(m), E_2(m)).$$

The encrypted message that can then be transmitted over an insecure channel to Bob is $\{(1,7)(1,8)\ (1,2)(1,10)(1,7)\}$*. Using Figure 5, this corresponds to the following message:*

$$E(m) = \{(B,H)(B,I)(B,C)(B,L)(B,H)\}.$$

The decrypted message is $\{(1,7)(1,8)(1,2)(1,10)(1,7)\}$*, for each pair* $(E_1(m), E_2(m))$.

For each pair given by $(E_1(m), E_2(m))$ *in the ciphertext, calculate*

$$D(E(m)) = E_1(m)^x \mod p = 1^7 \mod 13 = 1,$$

and $M = (E_2(m) \cdot E_1(m)^{-1}) \mod p$.

Remembering that $E_1(m)^{-1}$ *is the modular multiplicative inverse of* $E(m) \mod p$*. In the case of* $p = 13, E(m)^{-1} = 1$*, because 1 is the inverse of itself* $(1 \cdot 1 \mod 13 = 1)$*. Implying that* $M = (E_2(m) \cdot 1) \mod 13 = E_2(m)$*. Using Figure 5, we have m =CNMAC.*

To apply the homomorphism, we consider the messages $m_1 = CNMAC$ *and* $m_2 = 2022 = CACC$ *(where numbers are replaced by their corresponding letters from the alphabet, as shown in Figure 5). Using the ElGamal algorithm,*

$$\begin{aligned} E(m_1) \cdot E(m_2) &= ((2^5, (CNMAC) \cdot 11^5), (2^5, (CACC) \cdot 11^5) \mod 13) \\ &= E((CNMAC) \cdot (CACC)). \end{aligned}$$

3.3 Complexities the Algorithms

In this subsection, we analyze the computational complexities inherent to the RSA and ElGamal encryption algorithms. RSA encryption exhibits a time complexity of $O(log(e))$, where e denotes the encryption exponent, typically small, and involves primarily modular exponentiation operations. In contrast, ElGamal encryption operates with a complexity of $O(log(n))$, necessitating two modular exponentiations modulo n and a scalar multiplication operation $(m \times h)$, as detailed in [9]. Efficiency enhancements are achievable through strategies that optimize the computation of modular exponentiation products, as explored in [2].

Regarding decryption processes, both RSA and ElGamal algorithms share a comparable complexity framework. RSA decryption involves modular exponentiation under the condition $e < n$, achieving a time complexity of $O(log(n))$. In contrast, ElGamal decryption relies on modular

exponentiation operations with n as the group order, also operating at $O(log(n))$ complexity. The homomorphic properties inherent to these algorithms constrain operations to either addition or multiplication. Consequently, RSA decryption typically employs modular multiplication, while ElGamal decryption necessitates two modular multiplications—one for each component of the encrypted message. These complexities underscore the nuanced computational demands and efficiency considerations in cryptographic applications utilizing RSA and ElGamal algorithms.

4 CONCLUSION

In Abstract Algebra, the concept of homomorphism plays a fundamental role in simplifying complex algebraic structures into more manageable forms. This concept finds practical application in cryptographic techniques, especially in HE. HE is important because it preserves both additive and multiplicative operations, allowing computations to be performed on encrypted data without the need for prior decryption, thereby maintaining data privacy and security. This capability facilitates secure communication among parties involved in sensitive transactions or data exchanges. The computational complexities of homomorphic cryptographic algorithms illustrate the efficiency and feasibility of RSA and ElGamal in practical cryptography scenarios. These complexities highlight the computational demands and optimizations required in implementing secure and efficient cryptographic algorithms, essential for protecting sensitive information in modern digital communications.

REFERENCES

[1] S.S. Al-Riyami & K.G. Paterson. Certificateless Public Key Cryptography. In C.S. Laih (editor), "Advances in Cryptology - ASIACRYPT 2003". Springer Berlin Heidelberg, Berlin, Heidelberg (2003), p. 452–473.

[2] F. Borges, P. Lara & R. Portugal. Parallel algorithms for modular multi-exponentiation. *Applied Mathematics and Computation*, **292** (2017), 406–416.

[3] T. El Gamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In "Proceedings of CRYPTO 84 on Advances in Cryptology". Springer-Verlag, Berlin, Heidelberg (1985), p. 10–18.

[4] A. Gonçalves. "Introdução à Álgebra". Projeto Euclides - IMPA, 3nd ed. (1995).

[5] S. Halevi. "Homomorphic Encryption". Springer International Publishing, Cham (2017), p. 219–276. doi:10.1007/978-3-319-57048-8_5.

[6] R. Klima, N. Sigmon & E. Stitzinger. "Applications of Abstract Algebra with MAPLE". CRC Press, 1nd ed. (1999).

[7] A. López-Alt, E. Tromer & V. Vaikuntanathan. On-the-fly Multiparty Computation on the Cloud via Multikey Fully Homomorphic Encryption. In "Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing", STOC '12. ACM, New York, NY, USA (2012), p. 1219–1234.

[8] P. Paillier. Trapdooring Discrete Logarithms on Elliptic Curves over Rings. In T. Okamoto (editor), "Advances in Cryptology – ASIACRYPT 2000". Springer Berlin Heidelberg, Berlin, Heidelberg (2000), p. 573–584.

[9] D. Pereira, M. Aranha & F. Borges. HTTPS Keys in the Mediterranean. In "2019 II Workshop on Metrology for Industry 4.0 and IoT (MetroInd4.0 & IoT)" (2019), p. 449–454. doi:10.1109/METROI4.2019.8792830.

[10] R.L. Rivest, L. Adleman & M.L. Dertouzos. "On Data Banks and Privacy Homomorphisms", 4. Academia Press, Massachusetts (1978), p. 169–180.

[11] A. Sinha. Client-Server Computing. *Commun. ACM*, **35**(7) (1992), 77–98. doi:10.1145/129902.129908. URL `https://doi.org/10.1145/129902.129908`.

How to cite

A. L. Z. Lunkes & F. Borges. The application of homomorphism in cryptography. *Trends in Computational and Applied Mathematics*, **25**(2024), e01772. doi: 10.5540/tcam.2024.025.e01772.