Applying Logarithm and Russian Multiplication Protocols to Improve Paillier's Cryptosystem

Hamid El Bouabidi[1], Mohamed EL Ghmary[2], Sara Maftah[3], Mohamed Amnai[4] and Ali Ouacha[5]
Department of Computer Science, Ibn Tofaïll University
Faculty of Science, Kenitra, Morocco[1,3,4]
Department of Computer Science, Faculty of Science, Dhar El Mahraz,
Sidi Mohamed Ben Abdellah University, Fez, Morocco[2]
Department of Computer Science
Mohammed V University in Rabat, Morocco[5]

***Abstract*—Cloud computing provides on-demand access to a diverse set of remote IT services. It offers a number of advantages over traditional computing methods. These advantages include pay-as-you-go pricing, increased agility and on-demand scalability. It also reduces costs due to increased efficiency and better business continuity. The most significant barrier preventing many businesses from moving to the cloud is the security of crucial data maintained by the cloud provider. The cloud server must have complete access to the data to respond to a client request. That implies the decryption key must be sent to the cloud by the client, which may compromise the confidentiality of data stored in the cloud. One way to allow the cloud to use encrypted data without knowing or decrypting it is homomorphic encryption. In this paper, we focus on improving the Paillier cryptosystem, first by using two protocols that allow the cloud to perform the multiplication of encrypted data and then comparing the two protocols in terms of key size and time.**

Keywords*—*Cloud computing; cloud security; homomorphic encryption; paillier cryptosystem; sockets

I. Introduction

Cloud computing opens up previously untapped possibilities for storage and computation outsourcing. Many people are interested in using this technology since it gives flexibility, accessibility, and cost savings [1], [2], [3]. Over the last two decades, a surge in data has been generated and stored due to the creation of the internet of things, artificial intelligence and cloud computing [4]. However many authors have proposed solutions to optimize the offloading decision and the computing resource allocation to minimize the overall tasks processing time and energy [5], [6], many users are hesitant to commit sensitive data to the cloud due to concerns about privacy and security, making cloud security a critical matter. Indeed, cloud security literature has proposed and evaluated different encryption schemes [7], [8]. Particularly intriguing is homomorphic encryption, which allows any data to remain encrypted while being processed and manipulated. The organization of this paper is described as follows: Section 2 will mention some related works to secure the cloud. Section 3 begins with a summary of Paillier's cryptosystem. Section 4 goes on to describe the two protocols that we use to perform multiplication on encrypted values, followed by our conclusion in Section 5.

II. Related Works

Research efforts are directed toward several types of homomorphic encryption to secure the cloud, including partially homomorphic encryption. These schemes allow for the execution of a single operation on encrypted data, mainly addition, as in Goldwasser-Micali [9] and DGK [10], or multiplication, as in El Gamal [11] and unpadded RSA [12]. This paper will focus on the well-known additively homomorphic Paillier scheme [13]. It enables the computation of sums on encrypted data, which is useful in a variety of applications, such as encrypted SQL databases [14], machine learning on encrypted data [15], and electronic voting [16]. The authors in [17] address issues, possibilities, and potential improvements related to homomorphic encryption. They describe how we can use homomorphic encryption to process computations in big data. The authors of [18] provide a comprehensive assessment of homomorphic encryption, highlighting current application needs and future potential in areas such as security and privacy. This paper will present two protocols to improve Paillier's encryption scheme and allow the cloud provider to perform multiplication on encrypted data.

III. Paillier's Cryptosystem

A. Background

Paillier [13] proposes a new probabilistic encryption method relying on group computations based on calculations over the group $\mathbf{Z_{n^2}}$, where n is an RSA modulus. This scheme is captivating because it is homomorphic, enables the encryption of many bits in a single operation with a constant expansion factor, and enables effective decryption. As a result, it has the potential to be suitable for a variety of cryptographic protocols, such as electronic voting and mix-nets. This approach is similar to Okamoto and Uchiyama's voting and mix-nets cryptosystem [19], in which the group $\mathbf{Z^*_{p^2 \cdot q}}$ is used, where p and q are large primes. The principal difference is that the homomorphic property of this scheme necessitates that the sum of the messages being added be less than p, which is unknown. Because the homomorphic computations in Paillier's method are simply modulo n, this problem is avoided.

B. Paillier Original Algorithm

In [13], Paillier describes two partially homomorphic cryptosystems, schemes 1 and 2. Scheme 1 is the basic Paillier scheme, while scheme 2 is a quicker decryption variant. The Paillier scheme's security relies on the n-th residues in $\mathbf{Z_{n^2}}$ and the toughness of integer factorization. Therefore, we only concisely review the fundamentals and comment on key generation and parameter selection here. Finally, we refer to the original article [13] for more information on the scheme's security. The multiplicative group $\mathbf{Z_{n^2}}$, for $n = pq$ and two prime numbers p and q serve as the setting for the Paillier's scheme. Notice that $\mathbf{Z_{n^2}}$ has $|\mathbf{Z_{n^2}}| = \phi(n^2) = n \cdot \phi(n) = (p-1)(q-1)n$ elements. The Carmichael's function on n, $\lambda(n)$, is short-handed to λ .

1) Scheme 1: In Table. I, Paillier's method is provided in it's most basic form:

TABLE I. PAILLIER'S SCHEME 1

Parameters	prime numbers $n = p.q$ $\lambda = lcm(p-1, q-1)$ g, $g \in \mathbf{Z_{n^2}}$ the order of g is a multiple of n
Public key	n, g
Private key	p, q, λ
Encryption	plaintext $m < n$ select a random $r < n$ such that $r \in \mathbf{Z^*_{n^2}}$, ciphertext $c = g^m r^n mod\, n^2$
Decryption	ciphertext $c < n^2$ plaintext $m = \frac{L(c^\lambda mod\, n^2)}{L(g^\lambda mod\, n^2)} mod\, n$

Following the notation of [14], $L(u) = \frac{u-1}{n}$, for $u = 1\, mod\, n$. This function is only used on input values u that actually satisfy $u = 1\, mod\, n$.

2) Scheme 2: This is a faster version of the original Paillier algorithm. We work in the subgroup $< g >$ generated by an element g of order αn rather than the entire group $\mathbf{Z^*_{n^2}}$.This enables exponentiation decryption using the exponent $alpha$ instead of $lambda$, which speeds up decryption depending on the size of $alpha$. Scheme 2 is described in Table II:

TABLE II. PAILLIER'S SCHEME 2

Parameters	prime numbers $n = p.q$ α divisor of $\lambda = lcm(p-1, q-1)$ g, $g \in \mathbf{Z_{n^2}}$,the order of g is a multiple of αn
Public key	n, g
Private key	p, q, α
Encryption	plaintext $m < n$ select a random $r < \alpha$ ciphertext $c = g^m (g^n)^r mod\, n^2$
Decryption	ciphertext $c < n^2$ plaintext $m = \frac{L(c^\alpha mod\, n^2)}{L(g^\alpha mod\, n^2)} mod\, n$

C. Paillier's Cryptosystem Propreties

Paillier's homomorphic encryption has the following Propreties as it is shown in Fig. 1:

- As it's a public key system, anyone with the public key can encrypt, but decryption requires the private key, which is only known to a trustworthy individual.
- It is based on probabilities.It means, an attacker cannot tell whether two ciphertexts are encryptions of the same plaintext or not.
- For addition, it includes the homomorphic properties listed below:

$$E[(m1 + m2)]\, mod\, n = E[m_1] \cdot E[m_2]\, mod\, n^2 \quad (1)$$

$$E[(a \cdot m)]\, mod\, n = E[m]^a\, mod\, n^2 \quad (2)$$

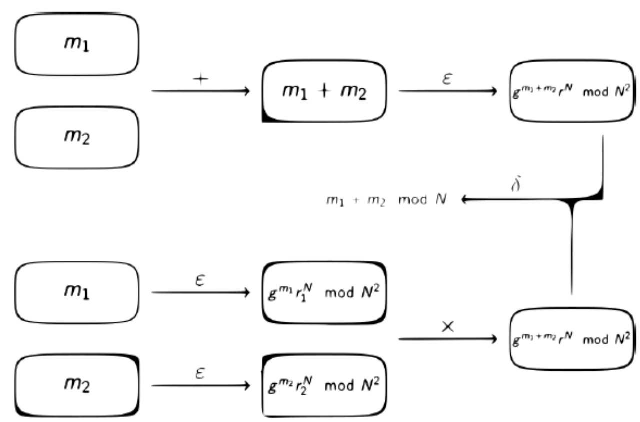

Fig. 1. Paillier Homomorphic Multiplicative Properties.

In which m is the encryption modulus and one of the public key elements. The key generation scheme is as follows:

- Choose p and q as two huge prime numbers such that:

$$gcd(p \cdot q, (p-1) \cdot (q-1)) = 1 \quad (3)$$

  This condition is guaranteed if p and q have the same bit lengths.
- Calculate $n = p \cdot q$ and $\lambda = lcm(p-1, q-1)$
- Choose a random integer g from $\mathbf{Z^*_{n^2}}$
- Ensure n divides the order of g by determining whether the following modular multiplicative inverse exists: $\mu = (L(g\, mod\, n^2))^{-1}\, mod\, n$ where $L(u)$ is the quotient of the Euclidean division of $\frac{u-1}{n}$
- The public encryption key is g and n
- The private encryption key is μ and λ

The following operations can then be used to encrypt the message: $m_1 + m_2\, mod\, n$

- Let m represent a message that has to be encrypted from $\mathbf{Z_n}$.
- Choose a random r from $\mathbf{Z^*_n}$
- Calculate ciphertext as:

$$E[m] = c = g^m r^n mod\, n^2 \quad (4)$$

The decryption is basically one exponentiation modulo n^2 :

$$m = L(c^{\lambda} \, mod \, n^2) \cdot \mu \, mod \, n \tag{5}$$

The decryption takes advantage of the fact that discrete logarithms are simple to compute,for example if g is chosen as $g = n + 1$ then $L(g^x) \, mod \, n^2 = x \, mod \, n$. Proof can be provided using the binomial. theorem.

IV. Applying Russian Multiplication and Logarithm Protocols

In this section, we will describe two protocols that make the Paillier encryption scheme act like fully homomorphic encryption by allowing multiplication of two encrypted values: the Russian multiplication and the continuous logarithm protocols.

A. Paillier Encryption and the Russian Multiplication Protocol

The Russian Peasant Multiplication Method is a common practice in Russian communities. This approach substitutes the frequently used multiplication procedure and only needs the usage of the table of twos. This theorem is currently included in many number theory textbooks [20]. To proceed, multiply the partial products on the left by two and divide the partial products on the right by two. It is similar to expressing the multiplier in base two and then doing multiplications and additions by two. It's a variation of the ancient Egyptian multiplication method. This method's algorithm is as follows:

Algorithm 1 Russian Multiplication

Input: m1,m2,table tab
Output: $m1 \times m2$
while $m1 > 0$ **do**
  if $(m1\%2 = 1\ )$ **then**
    $e2 = encrypt(m2, pubKey)$
    Add $e2$ to tab
  end if
  $m1 = m1//2$
  $m2 = m2 * 2$
end while
return tab

We used sockets that allow remote machines to communicate with each other using their IP addresses. When a client machine needs a service, it contacts a server machine. This is known as the client-server logic. One asks, the other answers, as illustrated in the sequence diagram Fig. 2:

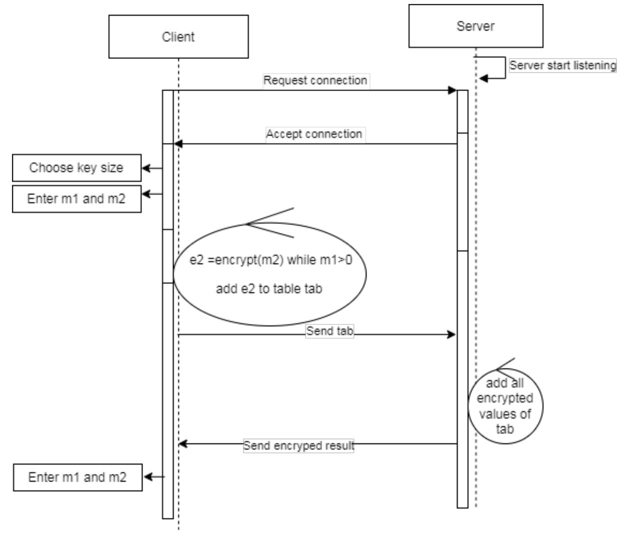

Fig. 2. Russian Multiplication Protocol Sequence Diagram.

Once the Server accepts the connexion, the client request multiplication of $m1$ times $m2$. Let's take the example where $m1 = 73$ and $m2 = 96$. The client interface will transform 73×91 by 91, 728 and 5824 using the Russian multiplication algorithm, encrypt those values and send them to the cloud provider. The cloud can easily add those encrypted values using the Paillier algorithm and then return the result to the client, who could use his private key to see the plaintext. As is shown in the following algorithm:

Algorithm 2 Russian Multiplication Protocol

Input: m1,m2,table tab
Output: $m1 \times m2$
{Client Side}
while $m1 > 0$ **do**
  if $(m1\%2 = 1\ )$ **then**
    $e2 = encrypt(m2, pubKey)$
    Add $e2$ to tab
  end if
  $m1 = m1//2$
  $m2 = m2 * 2$
end while
send tab
R = socket.recieve {result send by the cloud}
{Cloud Side}
R_tab = socket.recieve(tab)
sum = 0
for x in R_tab **do**
  sum = sum + x
end for
socket.send(sum)

B. Paillier Encryption and Continuous Logarithm

Without logarithms, many of our modern technological advances would be nearly impossible. We take advantage of the intriguing rule that transforms multiplication into an addition.

$$log(a \times b) = log(a) + log(b) \tag{6}$$

Algorithm 3 Continuous Logarithm Multiplication

Input: m1,m2
Output: $m1 \times m2$
$l1 = log(m1)$
$l2 = log(m2)$
$e1 = encrypt(m1)$
$e2 = encrypt(m2)$
$e = e1 + e2$
decrypt(e)
prod=exp(m)

Lets $e1$ and $e2$ be the respective encryption of $m1$ and $m2$ in Fig. 3, by applying the continuous logarithm protocol, the client will be able to compute $m1$ times $m2$ just by sending $e1$ and $e2$ to the cloud.

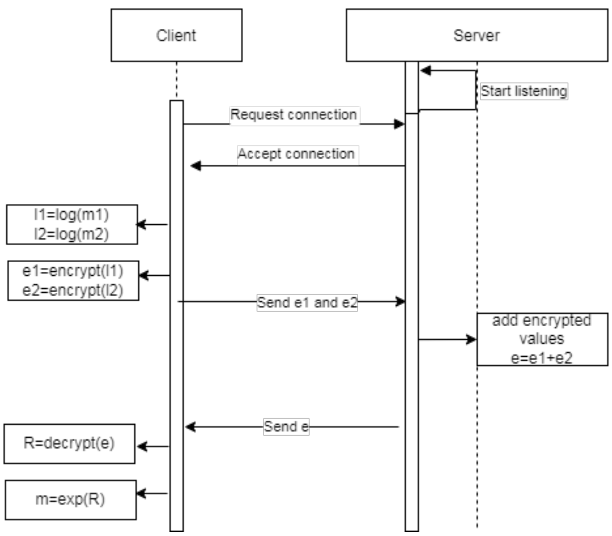

Fig. 3. Continuous Logarithm Protocol Sequence Diagram.

Once the Server accepts the connexion, the client request multiplication of $m1$ times $m2$, and the client interface will apply logarithms on both values. Then, encrypt them and send them to the cloud provider. The cloud will add those encrypted values using the Paillier algorithm and then return the result to the client interface, which could decrypt and apply exponential to display the answer for the client. For a better understanding, we provide the algorithm of the logarithm protocol that allows the cloud to compute production on encrypted values:

Algorithm 4 Continuous Logarithm Multiplication

Input: m1,m2
Output: $m1 \times m2$
{Client Side}
$l1 = log(m1)$
$l2 = log(m2)$
$le1 = encrypt(l1)$
$le2 = encrypt(l2)$
send(le1,le2)
sum=socket.receive
m=decrypt(sum)
message=exp(m)
{Cloud Side}
sum=0
c1=socket.receive(le1)
c2=socket.receive(le2)
$sum = c1 + c2$
socket.send(sum)

C. Implementation and Results

In this section, we propose a description of an implementation of a desktop interface that will allow clients to encrypt a database and request the cloud provider to make calculations on the encrypted data as it is shown in Fig. 4:

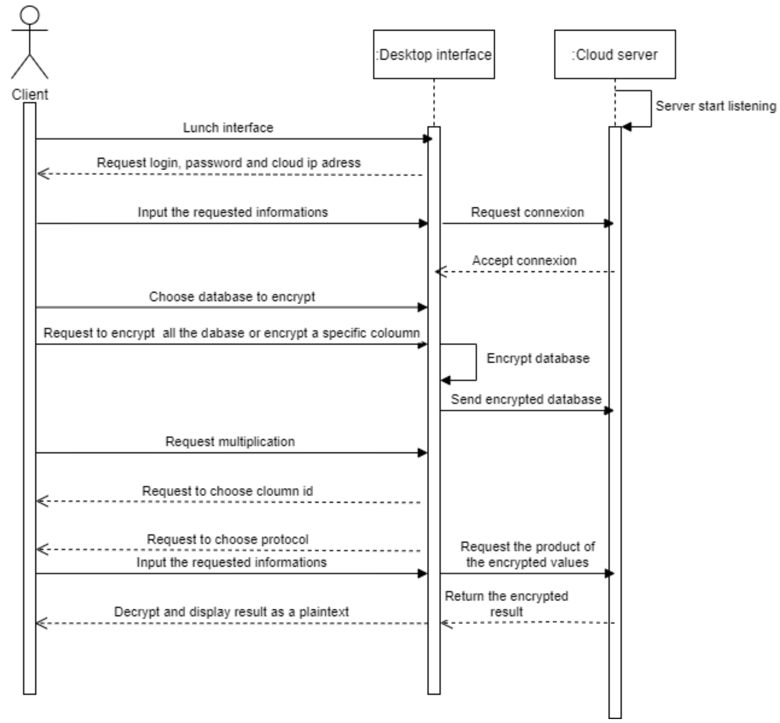

Fig. 4. The Client Desktop Interface Sequence Diagram.

The client desktop interface encrypts the database and sends it to the cloud server. Since the database is stored all encrypted in the cloud, the client sends a request to perform an operation or processing to benefit from the storage and calculation capacity of the cloud servers. Then, the client application decrypts it and returns the same result as if the operation is performed on the data in clear. We validate the applicability of our approach in different cloud solutions by implementing and managing encrypted database operations on a real cloud Hetzner. The current version of our prototype supports TinyDB databases. We chose TinyDB [21] because of the following advantages:

- Written in pure Python
- No dependencies
- Python2 and Python3 compatible
- Easy to use, very clean API Lightweight (2000 lines of code)

To improve security, we encrypted the database name, table names, and field names. This technique will allow us to do a variety of tasks without revealing any information about what we want to accomplish or the contents of our database. Which can be beneficial in many fields, such as medicine to protect the privacy of patients' information [22] or finance and Banking

[23]. To provide additional flexibility to the client so that he does not need to encrypt the whole database, especially if the vital data is on a single column. The client might use the following algorithm to encrypt just that single column:

Algorithm 5 Encrypt Column

```
Input: column_names table
Output: encrypted column
Function encryptcolumn(id)
if column type is String then
    encrypt column using RSA
else
    encrypt column using Paillier
end if
```

Fig. 5 illustrate the result that the cloud gets after a client decided to encrypt all the database using the Algorithm 6.

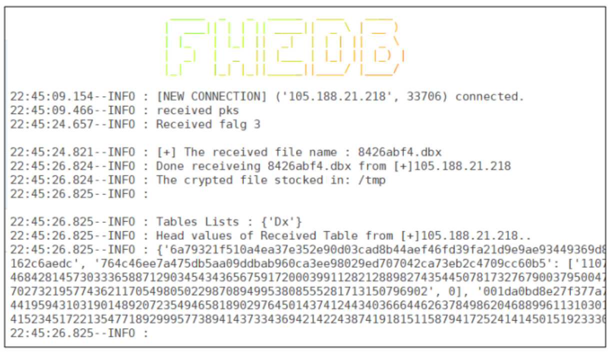

Fig. 5. Encrypted Database Received by the Cloud.

To encrypt the entire database we used the following algorithm:

Algorithm 6 Encrypt All Database

```
Input: Xtable table ,checked_ele String
Output: encrypted database
if Xtable exists then
    for non encrypted column do
        checked_ele = id_of_nonencryptedcolumn
        encryptcolumn(checked_ele)
    end for
else
    create table Xtable
    for each column in Xtable do
        if column type is String then
            encrypt column using RSA
        else
            encrypt column using Paillier
        end if
    end for
end if
```

Table III describes the comparison of the two protocols, using sockets, in terms of time and key size:

TABLE III. Result and Comparison of the Two Protocols using Sockets

Key Length	Russian Multiplication Protocol(ms)	Logarithm Protocol(ms)
N=64	3.3	2.3
N=128	3.6	2.5
N=256	5.3	5.0
N=512	19.7	23.6
N=1024	66.4	60.4
N=2048	425.6.6	409.0
N=4096	2936.0	2830.0

With a tiny key size, we notice that both algorithms provide the same result in about the same amount of time. However, when the key size is increased, the Product encryption protocol employing Russian multiplication takes longer but performs better than the continuous logarithm method. As a result, if the operations performed on the cloud require precision, we should use the Russian multiplication protocol. Still, we can use the continuous logarithm protocol if we want speed with approximate values.

V. Conclusion

In this paper, we focused on improving Paillier's method by implementing two protocols that allow the cloud to conduct multiplication on encrypted data by including two protocols that transform multiplication into addition. To show the effectiveness of our approach, we created a desktop interface that enables users to benefit from the cloud while protecting the security of sensitive data held on remote servers and controlled by cloud providers. The client interface adds an extra layer of security to a database by encrypting the names of columns and tables in the database. The proposed solution would have a significant economic effect due to the assurance of data security, confidentiality, and data protection through its use. Additionally, this would encourage more businesses and financial institutions to keep their data in the cloud.

References

[1] M. Ahmadi and N. Aslani, "Capabilities and advantages of cloud computing in the implementation of electronic health record," *Acta Informatica Medica*, vol. 26, no. 1, p. 24, 2018.

[2] A. ALI, "An overview of cloud computing for the advancement of the e-learning process," *Journal of Theoretical and Applied Information Technology*, vol. 100, no. 3, 2022.

[3] P. Modisane and O. Jokonya, "Evaluating the benefits of cloud computing in small, medium and micro-sized enterprises (smmes)," *Procedia Computer Science*, vol. 181, pp. 784–792, 2021.

[4] A. Ouacha and M. El Ghmary, "Virtual machine migration in mec based artificial intelligence technique," *IAES International Journal of Artificial Intelligence*, vol. 10, no. 1, p. 244, 2021.

[5] M. El Ghmary, T. Chanyour, Y. Hmimz, and M. O. Cherkaoui Malki, "Processing time and computing resources optimization in a mobile edge computing node," in *Embedded Systems and Artificial Intelligence*. Springer, 2020, pp. 99–108.

[6] M. El Ghmary, Y. Hmimz, T. Chanyour, and M. O. C. Malki, "Energy and processing time efficiency for an optimal offloading in a mobile edge computing node," *International Journal of Communication Networks and Information Security*, vol. 12, no. 3, pp. 389–393, 2020.

[7] H. Pham, J. Woodworth, and M. Amini Salehi, "Survey on secure search over encrypted data on the cloud," *Concurrency and Computation: Practice and Experience*, vol. 31, no. 17, p. e5284, 2019.

[8] G. Sen Poh, J.-J. Chin, W.-C. Yau, K.-K. R. Choo, and M. S. Mohamad, "Searchable symmetric encryption: Designs and challe," *ACM Computing Surveys*, vol. 50, no. 3, 2017.

[9] F. Ízdemir, Z. Ídemiş Ízger *et al.*, "Goldwasser-micali algorithm," in *Partially Homomorphic Encryption*. Springer, 2021, pp. 43–49.

[10] I. Damgård, M. Geisler, and M. Krøigaard, "Efficient and secure comparison for on-line auctions," in *Australasian conference on information security and privacy*. Springer, 2007, pp. 416–430.

[11] E. R. Arboleda, "Secure and fast chaotic el gamal cryptosystem," *Int. J. Eng. Adv. Technol*, vol. 8, no. 5, pp. 1693–1699, 2019.

[12] S. Rubinstein-Salzedo, "The rsa cryptosystem," in *Cryptography*. Springer, 2018, pp. 113–126.

[13] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *International conference on the theory and applications of cryptographic techniques*. Springer, 1999, pp. 223–238.

[14] K. Foltz and W. R. Simpson, "Extending cryptdb to operate an erp system on encrypted data." in *ICEIS (1)*, 2018, pp. 103–110.

[15] K. Muhammad, K. A. Sugeng, and H. Murfi, "Machine learning with partially homomorphic encrypted data," in *Journal of Physics: Conference Series*, vol. 1108. IOP Publishing, 2018, p. 012112.

[16] H. Kim, K. E. Kim, S. Park, and J. Sohn, "E-voting system using homomorphic encryption and blockchain technology to encrypt voter data," *arXiv preprint arXiv:2111.05096*, 2021.

[17] B. Alaya, L. Laouamer, and N. Msilini, "Homomorphic encryption systems statement: Trends and challenges," *Computer Science Review*, vol. 36, p. 100235, 2020.

[18] M. Alloghani, M. M. Alani, D. Al-Jumeily, T. Baker, J. Mustafina, A. Hussain, and A. J. Aljaaf, "A systematic review on the status and progress of homomorphic encryption technologies," *Journal of Information Security and Applications*, vol. 48, p. 102362, 2019.

[19] R. Suwandi, S. M. Nasution, and F. Azmi, "Secure e-voting system by utilizing homomorphic properties of the encryption algorithm," *Telkomnika*, vol. 16, no. 2, pp. 862–867, 2018.

[20] B. Cevizci, "How and why does the multiplication method developed by the russian peasants work?" *Journal of Inquiry Based Activities*, vol. 8, no. 1, pp. 24–36, 2018.

[21] M. Siemens, "Tinydb," 2016.

[22] T. Oladunni and S. Sharma, "Homomorphic encryption and data security in the cloud," in *Proceedings of 28th International Conference*, vol. 64, 2019, pp. 129–138.

[23] S. Mittal, P. Jindal, and K. Ramkumar, "Data privacy and system security for banking on clouds using homomorphic encryption," in *2021 2nd International Conference for Emerging Technology (INCET)*. IEEE, 2021, pp. 1–6.