Using Homomorphic Encryption in Banking Transactions

Burcu Ağdar
Computer Engeenering
Dokuz Eylul Univeristy
Izmir, Turkey
burcu.agdar@ogr.deu.edu.tr

Emirhan Bilge Bulut
Computer Engeenering
Dokuz Eylul University
Izmir, Turkey
emirhanbilge.bulut @ogr.deu.edu.tr

***Abstract*— In most cryptography algorithms, the encrypted data is then decrypted, allowing changes to be made on it. Afterwards, the modified data is encrypted again and stored or sent to the user. Decrypting encrypted data exposes some security vulnerabilities, causing the data to become the most vulnerable. With homomorphic encryption, these security vulnerabilities are eliminated, allowing us to perform transactions on the data in encrypted form. We will try to explain this encryption type with examples.**

Keywords—cryptology, homomorphic encryption, cloud securty

I. INTRODUCTION

Homomophic encryption is a type of encryption that ensures encryption by performing certain linear algebra operations on the data and ensures that certain linear algebra operations can be performed on the encrypted data in the same way without decryption. Homomorphic encryption has its own types. These are of three types: Partially homomorphic, Somewhat homomorphic , is Fully homomorphic encryption. The most insecure state of the data is when it is unencrypted while processing the data. While storing the data , it is tried to be encrypted and kept in the most reliable way . However, all this security is compromised during operation. Homomorphic encryption is an encryption method used to prevent this. The widespread use of cloud systems today, the fact that users store and manage their data on cloud-based systems, and the transition to these systems in the future will increase the need for security in cloud systems. Homomorphic encryption, the user's data going to the cloud will be encrypted when going from the user to the cloud. When the user wants to act on these data, according to a certain formulation on the encrypted data, the desired operation can be detected and the data can be modified without decrypting it. Thus, since the data is never decrypted, the vulnerability will be minimized.

II. HOMOMORPHIC ENCRYPTION TYPES

The reason why homomorphic encryption is divided into types is the number and type of operations used in encryption. As we mentioned before, homomorphic encryption exists in 3 types:

- Partially Homomorphic Encryption
- Somewhat Homomorphic Encryption
- Fully Homomorphic Encryption

A. Partially Homomorphic Encryption

Partially Homomorphic Encryption is based on using only one of the mathematical operations while encrypting. When performing encryption, only multiplication or only addition is used. However, there are no restrictions on these transactions.

B. Somewhat Homomorphic Encryption

Somewhat Homomorphic Encryption allows using both processes. However, unlike Partially, there is a limitation on the number of transactions.

C. Fully Homomorphic Encryption

Fully Homomorphic Encryption is derived from Somewhat. It allows multiple operations. Also, there is no limit to these transactions. However, the reason why it is less preferred than others is that it is slower due to the multiplicity of transactions.

III. RELATED WORKS

A. Parallelizing Fully Homomorphic Encryption

In this article, it includes a test with parallelization method in cloud systems to optimize fully homomorphic encryption and gives information about its optimization. First of all, some basic algorithms and the reasons why they are not fully homomorphic are mentioned. These methods are . The RSA Cipher Scheme (Rivest, Shamir and Adelman, Diffie and Hellman)[1][2] indicates that the encryption method is not fully homomorphic due to its deficiency in the aggregation process, while the ElGamal encryption scheme, Diffie and Hellman key exchange is multiplicatively homomorphic but not additively homomorphic, the Paillier Cipher Scheme's explains with examples that it works on data that is encrypted in two homomorphic ways, but cannot be used because it is not homomorphic. Fully homomorphic Algorithms are Gentry's Encryption and Fujitsu's full homomorphic encryption in the article . There are some features that distinguish these algorithms, but the shortest and summary difference is that Fujitsu claims that encryption will accelerate on bulk data, while Gentry performs fully homomorphic encryption on 8-bit data with certain lattice methods and bootstrapping procedure.

In this study, methods such as reducing the public key size of Gentry and Halevi, or static bit encryption, which are

used to accelerate homomorphic encryption, were not used and a different solution was tried to be brought. At the same time, it is stated in the article that hardware-based acceleration methods such as FPGA are not yet available.

The team in the article parallelized Gently's algorithm by building a cloud system on Open Stack and tried to speed it up by dividing it into small processing particles. Here, the master-slave principle is adopted and the master divides operations on homomorphic encryption it into parts (if possible) and sends these small parts to the slaves and tries to make the operations parallel by waiting for the calculation results. The performance results of this are given in tabular form and some inferences are made. But here, there are no conditions such as sending the slave first, whichever process is longer. Parallelization is sent according to whether the incoming transaction can be split or not. (If the process is going to take a long time, it does not have a structure of send first these long operations.) In other words, it does not perform Data Dependency Analysis. In the article, they went into details and mentioned that there are 3 types of data dependencies. As a result, long-running processes can be optimized to a certain extent by parallelizing the gently algorithm, and he mentioned that some improvement can be made when paralleling for fully homomorphic encryption.[3]

B. Non-Interactive Exponential Homomorphic Encryption Algorithm

It is a study that can be secured with homomorphic encryption on the mobile code and that in homomorphic encryption operations used for polynomials, whether the coefficients of the polynomial are not encrypted , the article claims solution these situations. This article tried to present the concept of exponential homomorphic encryption and the exponential homomorphic encryption algorithm based on RSA, and it was mentioned that this algorithm has exponential, additive and multiplicative properties.

First of all, we need to start with the definition of mobile code: It is a system that allows us to run code or objects on it, which allows us to perform parallel operations in distributed systems. The difference between mobile code and direct parallelization is that it performs transactions between nodes. This is often viewed negatively as it will lead to some security vulnerabilities and easy propagation (computer viruses).It can be defined as It is mentioned here that since there may be security vulnerabilities and data leaks, this can be overcome with homomorphic encryption. For this, the sum and product features of homomorphic ciphers are exemplified, and mixed multiplicative or fully homomorphic encryption, in which 2 of them are used, is given as an example. Rivet etc.[5] He stated that he first introduced the concept of privacy homomorphism (PH) (HE has its foundations here).

When examined in detail in the article and the literature was defined at the time of this article, he claimed that there is no homomorphic encryption in which the exponents of any polynomial can be encrypted. Here, the source explains homomorphic encryption with examples from Bob and Alice. The non-interactive EEF protocol proposed by Sander and Tschudin[6][7] is the same as homomorphic encryption.

The concept of homomorphic encryption Sander etc. He gave the concepts of addition, multiplication, hash multiplication homomorphism and algebraic homomorphism and their corresponding algorithms. These included the methods to be applied to achieve this EEF [8] and were examined in detail in the article, but it was proved that it cannot obfuscate the polynomial, that is, it cannot provide complete secrecy. In this article, he claims to solve this problem with the concept of exponential homomorphism, which solves this problem, and explains its algorithm.

Basically, to do this, he first examines the RSA algorithm, on which his algorithms are based, and proves why it is safe or hard to break by formulating it according to Fermat's little theorem and euler extension. These algorithms can be briefly called EHA and the steps in the general working principle are as follows:

1) Exponential homomorphic algorithm (EHA):
   a) Generate the public and private key:
   b) Encryption algorithm E·
   c) Decryption algorithm D·

is in the form. Here, the author, after proving and exemplifying how he combines the RSA and Homomorphic algorithm in detail and how RSA provides the security of exponents according to the euler and fermat that RSA uses, suggests that it can be used as a solution, as well as a solution in mobile code operations.[8]

C. Homomorphic Encryption Applied to the Cloud Computing Security

In this paper, which focuses on providing security in cloud systems with homomorphic encryption, it is mentioned that companies want to switch to the cloud in order to take advantage of many advantages, but they are concerned about security. After the data is taken to the cloud, standard encryption is used, but the data must be decrypted when processing. In this paper, there is an introduction and suggestion of the method that gives the same result when the data is processed in unencrypted form, but enables the operation on the encrypted data.

It is desirable to make sure that data is stored securely as long as it resides in the cloud and is even invisible to the cloud provider. However, the server must know the key in order to perform operations on the data in the cloud. Homomorphic encryption is the method to prevent this. In this paper, the authors focused on the use of homomorphic encryption for the security of cloud systems and examined this method. First of all, the introduction of the cloud concept and the necessity of homomorphic encryption to ensure the security of data in the cloud during calculation were mentioned. Although the innovations and conveniences brought by cloud computing are mentioned, it is mentioned that its security is not mentioned much. It has been argued that other services are not sufficient if there is no security. It was mentioned that in addition to storing and processing data securely, it is necessary to have an encryption method, which includes concepts such as response time to the customer. In the statement, it was argued that the way to ensure this security is to perform transactions without decrypting passwords, and this will be achieved with homomorphic encryption. In another part of

the paper, homomorphic encryption is introduced and exemplified. It is also mentioned in the historical development of homomorphic encryption. Later, Additive Homomorphic Encryption and Multiplicative Homomorphic Encryption were mentioned. An example of Additive Homomorphic Encryption is the collection of election results. The votes are sent in encrypted form, the transactions on the votes are encrypted and finally the calculated total value is decrypted. An example of Multiplicative Homomorphic Encryption is given by the operation performed on two messages.

In the applications part, which is a value part of the paper, it is stated that the data are added and multiplied without any limit, full homomorphic encryption, where the key can be stored for the password of the result, is the most ideal and IBM recommends it. The authors of the paper said that they perform performance analysis of homomorphic encryptions in their applications and that they work on a virtual platform as a server for this analysis. If they focus on

• The size and size of the public key influence the size of the encrypted message.
• Server latency of request handling based on the size of the encrypted message.
• The resulting decryption time of the request based on the size of the ciphertext the server sent.

In the conclusion part of the paper, it was said that their applications were made with fully homomorphic encryption. On the servers, the encryptions that allow the transactions requested by the clients were examined and the complex homomorphic encryptions were improved.[9]

D. Homomorphic Encryption for Cloud Computing Security

In this paper, a study was conducted on the security of data in cloud computing. In this study, the introduction of homomorphic encryption for security, types of homomorphic encryption etc. are explained. The study is exemplified by using the Rivest-Shamir-Adleman algorithm. In the first part of the paper, information was given on the concept of cloud. Cloud computing, arising from the need to store data, is the transfer of data to the cloud without the control of the user. In this case, the user leaves the security of their data to the cloud owner. However, decrypting the data during processing will make the data unprotected. In order to ensure security, the data must remain encrypted even during processing and only the user should have the key of the password. This is possible in homomorphic encryption. In the second part of the paper, an explanation is given about homomorphic encryption. In this paper, it is stated that the results of the data processed in the encrypted form in homomorphic encryption are the same as the ones made in the unencrypted form. The addition and multiplication feature of homomorphic encryption is exemplified as follows:

$x + y = Dec_K (Enc_K(x) + Enc_K(y))$
$x * y = Dec_K (Enc_K(x) * Enc_K(y))$

Homomorphic encryption is divided into 2 groups in this study. These are full and partially homomorphic encryptions. In partially homomorphic encryption, either multiplication or addition can be used, while in full homomorphic encryption both operations are allowed. RSA and El-Gamal algorithms are given as examples of partial encryption. These algorithms are partially homomorphic with respect to multiplication. The work of Gahi et al. is given as an example of full homomorphic encryption. An example of partial homomorphic encryption according to multiplication is given in the paper:

Encryption: $Enc(x) = x^z \pmod n = y$
Decryption: $Dec(y) = y^t \pmod n$

Here, x is the message, y is the encrypted version of the message, z is the public key, and t is the private key. It was then verified with numbers as follows.

$Enc(x_1) * Enc(x_2) = x_1^{\,z} * x_2^{\,z} \pmod n = (x_1 * x_2)^{\,z} \pmod n = Enc(x_1 * x_2)$

Example data are as follows $x_1 = 4$, $x_2 = 3$, $z = 7$, $t = 3$, $n = 33$ and the encrypted form of the data is expressed as E_x:

$E\_x_1 = Enc(x_1) = x_1^{\,z} \pmod n = 4^{\,7} \pmod{33} = 16$
$E\_x_2 = Enc(x_2) = x_1^{\,z} \pmod n = 3^7 \pmod{33} = 9$
$E\_x_1 * E\_x_2 = 16 * 9 = 144$
$Dec(E\_x_1 * E\_x_2) = (E\_x_1 * E\_x_2)^{\,t} \pmod n = (144)^{\,3} \pmod{33} = 12$

When the result of the multiplication operation performed after the messages were encrypted, the result was 12. If the messages were multiplied without encryption, the result would be:

$x_1 * x_2 = 4 * 3 = 12$

In the examples shown above, it is shown in the paper that RSA homomorphic encryption gives the correct result.

In the continuation of the paper, it was mentioned that the need for security increased as a result of the widespread use of cloud computing, and it was argued that homomorphic encryption would provide the desired security in cloud computing. In the application part of the paper, an application has been developed with the java programming language. This application works using the RSA algorithm. At the beginning of the application, two numerical data received from the user are encrypted and their encrypted versions are multiplied. Then this multiplication is decrypted and the result is shown on the screen. In addition, in order to check the accuracy of the result, the results of the operations performed on the numbers without encryption are shown.[10]

E. Use of Homomorphic Encryption with GPS in Location Privacy

In the content of the article, it is mentioned that after ensuring the security of GPS data with homomorphic encryption, some data will be obtained by performing operations on it without decrypting it, and that this data will be used to ensure the privacy and security of VIPs. First of all, the article argues that homomorphic encryption is slow and cannot be applied in many areas for this. Afterwards, he explains the application areas of HE (Homomorphic Encryption) and describes the literature reviews. They are grouped under 4 headings and are as follows:

1) Protection of mobile agents
2) Lottery protocols

3) Mix-nets- Mix-nets are protocols
4) Data aggregation in wireless sensor network

Let's go through the example suggested by the article. For example, a singer will go to the concert venue. For this, the organizers of the organization may want to follow the location live and want to know how soon it will arrive. If live location sharing is made directly here, the security and privacy of the celebrity or the VIP will not be ensured, since the place where the celebrity or the VIP stays or the way out will be determined. If a homomorphic encryption is applied to the GPS, this data will be sent to the owner of the organization in encrypted form and information such as how many minutes it will arrive or whether it is approaching will be obtained, while the location information will be shown as another region providing the same distance. The time here and the remaining path will be obtained over the homomorphic encryption applied data, but its original location will never be known. In the declaration, it is stated that there will be a certain loss of time depending on the complexity of the operations on the homomorphic encryption, but this will increase the security. It is an article about a project idea where the GPS information of VIPs will be encrypted with homomorphic encryption and privacy will be provided over this.[11]

F. Analysis of Partially and Fully Homomorphic Encryption

In this article, in which patially and fully homomorphic encryption is analyzed, first of all, it is mentioned that homomorphic encryption is a type of encryption that allows operations to be performed on data without decrypting it. It has been mentioned that partially homomorphic encryption is implemented using addition or multiplication. Examples of partially homomorphic encryption are RSA, ElGamal and Pailler. It is written that the fully homomorphic encryption is implemented using both addition and multiplication. It has been stated that the first example of this is a Lattice-based system (Craig Gentry). Homomorphic encryption has many benefits. In the article, it is stated that the biggest advantage is to carry out transactions without the need to decrypt the cipher, and that this system is used in many places such as banking, elections and search engines. Disadvantages have also been noted. Most importantly, it is complex. In particular, fully homomorphic encryption is quite complex. Also, this encryption may not be able to deal with malware. For example, withdrawing from the bank or changing the amount of money to be deposited.[12]

G. Homomorphic Encryption in the Cloud

Before moving on to the importance of homomorphic encryption in the cloud, it was mentioned in the article that there are two types of homomorphic encryption. For a better understanding of fully homomorphic encryption and partially homomorphic encryption, additive homomorphic encryption, multiplicative homomorphic encryption and homomorphic encryption where every operation is present are mentioned. Then, the cloud, which is the main purpose of the article, was mentioned. The importance of cloud computing in the developing internet world has been mentioned. Lack of security is one of the major drawbacks for the cloud. It has been argued that solving this problem will be with homomorphic encryption. Then the HElib library was mentioned. It is a young library developed in C++. As a result, it has been said that homomorphic encryption is making progress but still not quite enough. However, it has been said that in the following years, designs will be made that will increase the efficiency of homomorphic encryption.[13]

H. Secure Information Aggregation for Smart Grids Using Homomorphic Encryption

It is a study that includes the application of homomorphic encryption encryption method and performance measurements in order to ensure the security of data in Smart Electric Networks during transmission. Today, data such as instantaneous power consumption tracking and pricing can be followed in smart grids (electricity distribution network in this study). It has been suggested to apply homomorphic encryption in order to prevent Man in the middle attack attacks in the follow-up of these data.[14]

I. Generating Private Recommendations Efficiently Using Homomorphic Encryption and Data Packing

It is a study that asserts that the user does not have any privacy against the service providers in smart recommendation systems and brings a solution to this. The alleged service providers only have privacy protection for 3rd parties or individuals, but there is no privacy or security provider between the service provider and the user. They have developed a protocol to ensure that service providers do not see the user's data, and its performance measures are designed in a networking protocol. They conducted the study tests on a data set of 10 thousand people and aimed that the security of the user's data in the recommendation system should not be seen by the service providers.[15]

IV. PROPOSED WORK

A. *Flowchart for Application*

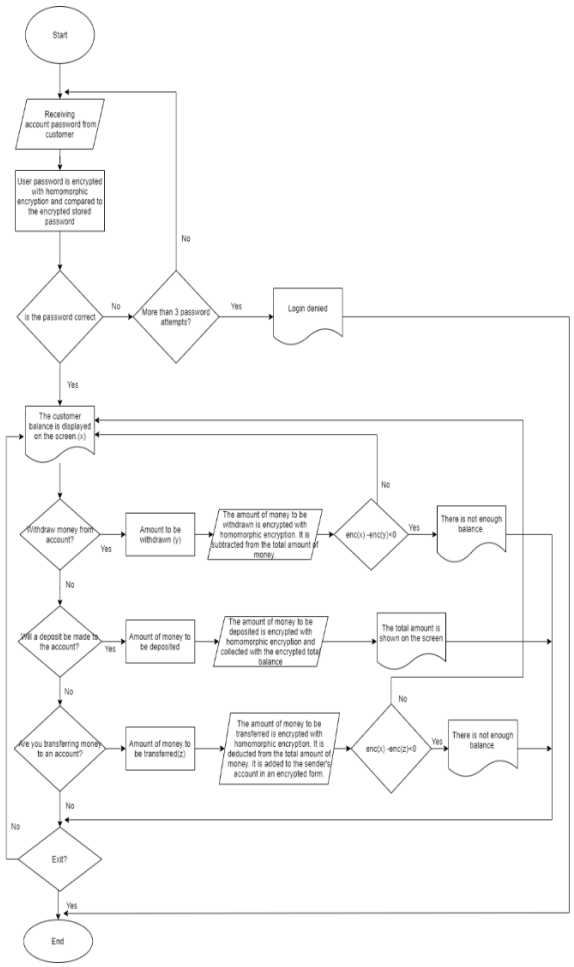

Figure 1 Flowchart

B. *Description of the Work*

Today, the usage rates in electronic banking systems have increased rapidly due to the increase in e-commerce and the closures caused by covid-19. Ensuring data security has gained great importance in the banking sector as users gradually turn to mobile banking. We recommend homomorphic encryption as one of the ways to ensure security during the hacking or leaking of millions of money transfer transactions made every day. Exposure of user or company based transactions such as money withdrawal, sending, depositing money to the account will create a major security and privacy weakness . If homomorphic encryption is used, the security of users or companies will be ensured in cases such as data leakage, since we can perform these operations on encrypted data. If it will be explained through Figure 1, the user will first enter a password to login to the account . After the password verification is done, the balance will be displayed on the screen. The balance will be encrypted with homomorphic encryption. Since the balance is updated with homomorphic encryption when the user wants to deposit money into his account, even if this transaction is exposed, the amount deposited by the user will never be known. At the same time, the transaction will be successful as the balance is updated over encrypted data. The same scenario also applies to the withdrawal case. A similar situation will also apply during online transactions .

C. *Test Environment and Test Scenarios.*

Have Alice want to transfer money to Bob online. As soon as Alice clicks "Make the transaction", first of all, the specified amount of money must be deducted from her balance. This transaction is carried out over the data encrypted with homomorphic encryption, and then this transfer transaction and the amount of money are carried out by guaranteeing the security with homomorphic encryption. Since the increase of Bob's balance will also be done using HE, any security vulnerability is not allowed. Alice or Bob can see the balance in their account only by decrypting it on their own screen. Since the balances and money flows will be kept encrypted in the banks, the confidentiality of the user will be ensured.

When the user wants to deposit money into his account, he first logs into his account with the password. Then he sees the balance of the screen. Enter the amount of money you want to deposit. This amount is encrypted and added to the total balance in encrypted form. Then the user sees the total balance on the screen without a password.

Spyder in Anaconda and the Python programming language will be used as the development environment. Python version must be at least greater than 3.5. An application will be developed on the library named Pyfhel (PYthon For Homomorphic Encryption Libraries) in Python. Pyfel is actually an optimized API, not a direct HE encryption library. It is a compilation with Microsoft Seal and PALISADE in the background. Pyfhel is built on top of Afhel, an Abstraction Hmomorphic Encryption Libraries in C++.

V. IMPLEMENTATION & TESTING

Separate applications were written for the client and server sides. When the user registers on the client side, his private key and other encrypted data are kept in his own area. The server is set up as an ftp server. The user sends encrypted data and files that enable the server to process by logging into the server. Since the server does not have the user's private key, the server cannot see the original version of the data, but it can operate on the file from the user.

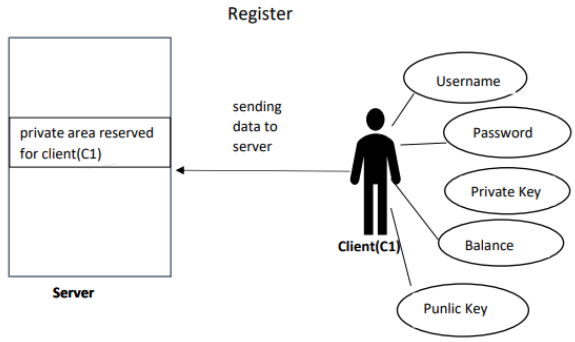

Figure 2 Register

The user has a username, password, balance, private and public keys. The user sends their data, except for the private key, to the server in an encrypted form. A special area is created on the server for the user and the data is saved as encrypted.

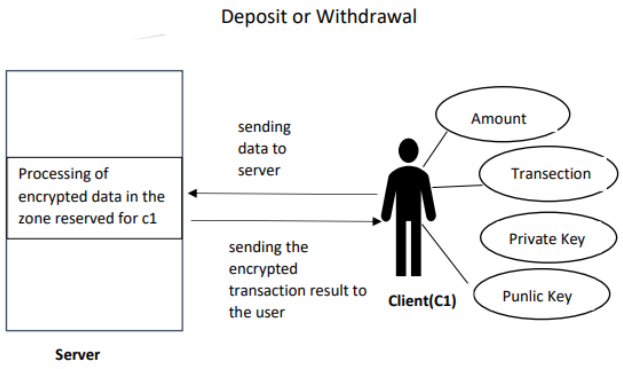

Figure 3 Deposit or Withdrawal

When the user wants to make a deposit or withdrawal. It sends the amount, transaction type and publickey to the server. The server is adding or subtracting from the balance in encrypted form. The result is saved to the server again.

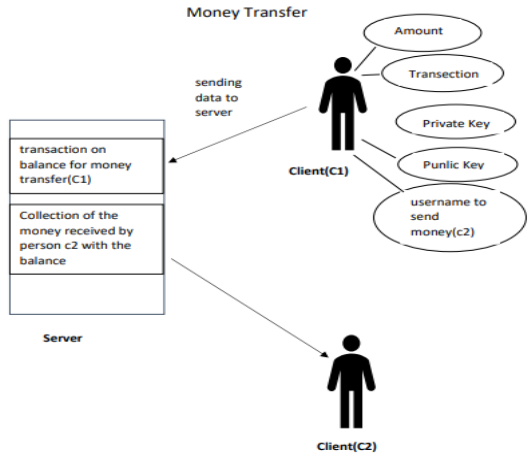

Figure 4 Money Transfer

In the money transfer process, the user sends the amount of money he wants to send, to whom he will send it, the type of transaction and the public key to the server. The server performs an encrypted reduction in the balance of the user who wants to transfer. Then, the transaction is added to the balance of the person to whom the money will be transferred, in encrypted form. The result can appear as decrypted on the screen of the user who receives the money.

Table 1 Withdrawal Performance

Withdrawal Operation	
Number of Users	How many seconds does it take
1	0.2632 s
10	0.8942 s
100	1.9673 s
1000	13.346 s

As the number of users increases, it is seen that the performance starts to decrease. There are two main features that degrade performance here. The first is that the encrypted data takes up too much space, and the second is that the cost of processing the encrypted data is high.

Table 2 Money Transfer Performance

Money Transfer	
Number of Users	How many seconds does it take
1	1.3564s
10	3,9865s
100	17.6546s
1000	38.1141 s

The money transfer speed is the slowest transaction. The main reason for this is that homomorphic encryption is applied on both the receiver and the sender side. In our application, after the sending user's transaction is done, the receiver's account is found and then the homomorphic encryption process is used for the second time to update the balance.

V. CONCLUSION

Monitoring transactions such as money flow in banks, withdrawals and deposits, or viewing the amount of payments made may pose a privacy concern for users. A firm's payments or the amount of money they invest, the emergence of investments made can cause problems. If banks perform these transactions with the homomorphic encryption method, the balance of the user or money transfers will be secured. This will also prevent Man in the middle attacks. We suggest that by using homomorphic encryption in banking transactions, the transactions and confidentiality of users or companies can be ensured and implemented.

REFERENCES

[1] W. Diffie and M. Hellman, "New directions in cryptography," IEEE Transactions on Information Theory, vol. 22, no. 6, pp. 644-654, 1976.

[2] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," Communications of the ACM, vol.21, no. 2, pp. 120-126, 1978.

[3] Hayward, R., & Chiang, C. C. (2014, June). Parallelizing fully homomorphic encryption. In 2014 International Symposium on Computer, Consumer and Control (pp. 721-724). IEEE.

[4] Fuggetta, Alfonso; Gian Pietro Picco; Giovanni Vigna (1998). "Understanding Code Mobility". IEEE Transactions on Software Engineering. 24 (5): 342–361. CiteSeerX 10.1.1.20.3442. doi:10.1109/32.685258. ISSN 0098-5589. Retrieved 29 July 2009.

[5] R.Rivest, L.Adleman, M. Dertouzos. On data banks and privacy homomorphisms[J], in: Foundations of Secure Computation, Academic Press, New York, 1978, pp.169–179

[6] T.Sander, C Tschudin. Towards mobile cryptography[C], In:Proc of the 1998 IEEE Symposium on Security and Privacy, IEEE Computer Society Press, Oakland California, 1998

[7] T.Sander, C.Tschudin. Protecting Mobile Agents Against Malicious Hosts[J], In G.Vigna, editor, Mobile Agent Security,Springer-Verlag, Heidelberg, Germany, 1998, pp.44-60

[8] Chen, L., Tong, Z., Liu, W., & Gao, C. (2012, October). Non-interactive exponential homomorphic encryption algorithm. In 2012 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (pp. 224-227). IEEE.

[9] Tebaa, M., El Hajji, S., & El Ghazi, A. (2012, July). Homomorphic encryption applied to the cloud computing security. In *Proceedings of the World Congress on Engineering* (Vol. 1, No. 2012, pp. 4-6).

[10] Çalık, E., Erdem, H. A., & Aydın, M. A. Bulut Bilişim Güvenliği için Homomorfik Şifreleme.

[11] Gupta, S., & Arora, G. (2019, November). Use of homomorphic encryption with gps in location privacy. In 2019 4th International Conference on Information Systems and Computer Networks (ISCON) (pp. 42-45). IEEE.

[12] Morris, L. (2013). Analysis of partially and fully homomorphic encryption. *Rochester Institute of Technology*, 1-5.

[13] Hrestak, D., & Picek, S. (2014, May). Homomorphic encryption in the cloud. In *2014 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)* (pp. 1400-1404). IEEE.

[14] Li, F., Luo, B., & Liu, P. (2010, October). Secure information aggregation for smart grids using homomorphic encryption. In 2010 first IEEE international conference on smart grid communications (pp. 327-332). IEEE.

[15] Erkin, Z., Veugen, T., Toft, T., & Lagendijk, R. L. (2012). Generating private recommendations efficiently using homomorphic encryption and data packing. IEEE transactions on information forensics and security, 7(3), 1053-1066.