Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 20 (2013) 502 – 509

Procedia Computer Science

Complex Adaptive Systems, Publication 3
Cihan H. Dagli, Editor in Chief
Conference Organized by Missouri University of Science and Technology
2013- Baltimore, MD

Homomorphic Encryption

Monique Ogburn[a*], Claude Turner[b], Pushkar Dahal[c]

[a,b,c] *Bowie State University, Department of Computer Science, 14000 Jericho Park Rd.,Bowie, MD 20715, United States*

Abstract

The study of homomorphic encryption techniques has led to significant advancements in the computing domain, particularly in the sphere of cloud computing. Homomorphic encryption provides a means for securely transmitting and storing confidential information across and in a computer system. The aim of this paper is to discuss the concepts and significance of homomorphic encryption along with the subdivisions and limitations associated with this type of encryption scheme. Recent studies conducted on the topic of homomorphic encryption are highlighted and some customary models of homomorphism are demonstrated. We also developed a proof of concept algorithm that demonstrates a practical use for a homomorphic encryption technique, the results of our algorithm are provided. The applications of homomorphic encryption methods are vast outside of the computational realm, and its purpose in other fields will be explored.

Keywords: ciphertext; fully homomorphic encryption; partially homomorphic encryption; encrypt; decrypt

1. Introduction

Webster's dictionary defines homomorphism as "a mapping of a mathematical set into or onto another set or itself in such a way that the result obtained by applying operations to elements of the first set is mapped onto the result obtained by applying those corresponding operations to their respective images in the second set" [1]. The word *homomorphism* is derived from the Greek word *homos* meaning "same" and *morphe* meaning "shape". In computer science homomorphic encryption is used in the conversion of plaintext to ciphertext.

Plaintext is any information that a senders desires to transfer to a receiver. It can be thought of as the input to any algorithm or as information being transmitted before an algorithm encrypts it. Some examples of plaintext include email messages, word processor files, images, or ATM and credit card transaction information. This plaintext is converted to ciphertext which is data that has been encrypted and is unreadable until it has been decrypted with a key.

Homomorphic encryption seeks to aid in this encryption process by allowing specific types of computations to be carried out on ciphertext which produces an encrypted result which is also in ciphertext. Its outcome is the result of operations performed on the plaintext. Case in point, one person could add two encrypted numbers and then another person could decrypt the result, without either of them being able to find the value of the individual numbers.

If a user wishes to process confidential data on another person's computer, which is a server "in the cloud" and would like to ensure no one else gains access to that data, including the owner of the computer, conventional methods of encryption would protect their data while it is in transit, but not while the computation is in progress. Homomorphic encryption provides security to the users' information from the moment the data stream leaves their computer until it returns. This method requires that all the arithmetical and logical operations needed in the computation, which may be represented by circuits or gates, be applied to the encrypted form of the data [2].

2. Significance of Homomorphic Encryption

In April 2011, Sony's PlayStation network was hacked into and millions of user's accounts were breached leaking credit card information, physical addresses, passwords, and other personal information [3]. Sony accepted responsibility for the incident admitting that they could have taken special precautions by encrypting the data on their network. Around the same time, researchers discovered that Dropbox was storing unencrypted user files. As a result users closed their accounts in protest angry at the company for not encrypting their confidential files [4].

The solution to this issue these two companies encountered was not as apparent as one might believe. Firstly, in order for data to have been used by their customers and clients, the data had to be decrypted. To do so, the decryption key had to be located somewhere between the data-store and the user. The ideal place for the decryption key was far from the data-store as possible and close to the user. However, this was extremely difficult to accomplish without jeopardizing the security of their client's data. For instance, Sony needed to be able to charge their customers credit card whether they were online or not and this required a billing address. Even if the credit card numbers and addresses were encrypted, they still needed to store the decryption key somewhere on their servers. If they offered an "update account" page with the address pre-filled, that decryption key had to be available to decrypt the data as soon as the customer clicked "update my account" [4]. Therefore, if Sony's web servers needed to be able to decrypt data, and hackers hacked into Sony's servers, there is only so much protection encryption would have been able to provide [4].

The development of cloud storage systems such as Dropbox and computing platforms gives users the ability to outsource storage and computations on their data, and allows businesses to outsource an increasing amount of data storage and management to cloud services. Although they gain these advantages, the potential drawbacks of utilizing cloud services are losses of privacy and business value of confidential data. One effective method of dealing with these problems along with Sony's issues is to encrypt all the data stored in the cloud and perform operations on the encrypted data. If the encryption scheme is homomorphic, the cloud can still perform meaningful computations on the data, even though it is encrypted [5].

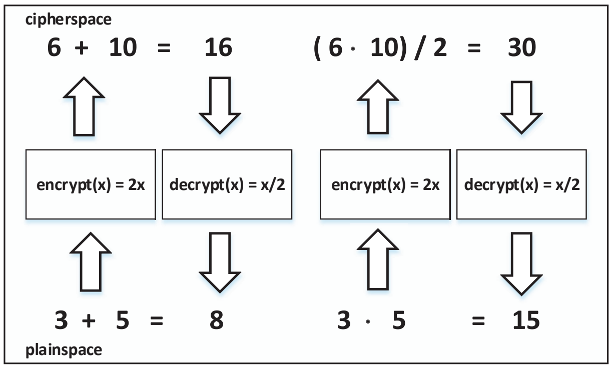
3. Examples

Earlier, homomorphic encryption was defined as a form of encryption where a specific algebraic operation performed on plaintext is equivalent to another algebraic operation performed on its ciphertext. In mathematics, these operations are called mappings or functions and are operation preserving (OP) mappings. For example, if (G,o) and $(H,*)$ represent groups, an OP mapping $h:(G,o)\to(H,*)$ is called a homomorphism from (G,o) *to* $(H,*)$. The groups (G,o) and $(H,*)$ represent sets of data and the mapping (function) that maps the set (G,o) onto or into the set $(H,*)$ is h [6]. This function is an operation that preserves the structure from one set of data to the other set of data.

Mathematical analogies can be utilized to explain the concepts behind homomorphic encryption. The succeeding explanation and illustration in this section has been adopted from Brian Hayes' *Alice and Bob in Cipherspace* article in the American Scientist Journal.

To further explain how homomorphism works, we consider two sets of data. One is the set of positive real numbers, R^+, and the other set is the logarithms of this set of real numbers. On these sets, the multiplication of real numbers and the addition of logarithms are homomorphic operations. If we consider any real positive numbers x, y and z, if $x \bullet y = z$, then $\log(x) + \log(y) = \log(z)$. This provides us with two alternate paths for obtaining our result, z. For the first method, if we are given x and y, we can multiply them together to obtain z. As the second option, we can add the logarithms of x and y and take the antilog of the sum to arrive at z. Both cases produce the same result [2]. With homomorphic encryption we can "perform arithmetic directly on the plaintext inputs x and y. Or we can encrypt x and y, apply a series of operations to the ciphertext values, then decrypt the result to arrive at the same final answer. The two routes pass through parallel universes: plainspace and cipherspace" [2]. Figs. 1a and 1b provide a visual representation of how homomorphic encryption is implemented.

In Fig. 1a, we see two sets of objects, the lower group contains the set of all integers, Z, and the higher one represents the set of even integers. The operations performed on these objects are addition and multiplication. Transitioning between the two sets requires doubling or halving a number. In the case of addition, the numbers 3 and 5 are the plaintext inputs and these inputs are encrypted by doubling their values producing 6 and 10. In cipherspace, addition is performed on the ciphertexts, 6 and 10, which yields 16. This encrypted result is decrypted by dividing by 2 to obtain its actual value. There is a slight variation in the example next to it which demonstrates the process for multiplication. Here again, the numbers 3 and 5 are used as the plaintext inputs. They are also converted to ciphertext by multiplying by 2 but in order to obtain the product of 3 and 5, we have to divide the product of the encrypted values in half.



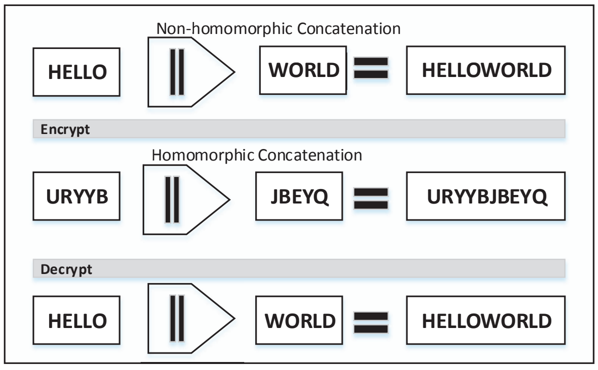


Fig. 1. (a) Example of an encryption of numerical values as plaintext using homomorphic encryption; (b) Example of the concatenation of two words using homomorphic encryption

Fig. 1b illustrates an example of using homomorphic encryption to concatenate two words. In this example, we see that the operation used to concatenate the plaintexts HELLO and WORLD is the same operation used for concatenating their respective ciphertexts. This is not always the circumstance. The point to be made is that we can still perform some operation on the input ciphertext which will produce new ciphertext that when decrypted, will produce plaintext corresponding to a desired operation on the input plaintext [7].

Three topics that must be highlighted are the concepts of partially homomorphic (PHE), somewhat homomorphic (SWHE), and fully homomorphic encryption (FHE) schemes. In partially homomorphic encryption, it is possible to perform one operation on encrypted data, such as multiplication or addition but not both. Somewhat homomorphic encryption techniques can perform more than one operation but can only support a limited number of addition and multiplication operations. A cryptosystem which sustains both addition and multiplication, and can compute any function is known as a fully homomorphic encryption system. The value of using FHE over PHE or SWHE is that with this model, circuits can be homomorphically evaluated. This in turn, successfully permits the construction of programs which may be run on encryptions of their inputs to produce an encryption of their output. Because these programs never decrypt their inputs, they can be run by untrusted individuals without risking and revealing its inputs and internal state. As of today, there is one advantage that PHE and SWHE methods have over FHE techniques; they have been found to be more efficient in their processes.

Most of the current encryption schemes that can be feasibly used on cloud services are somewhat or partially homomorphic. Microsoft researchers, Kristin Lautner, Vinod Vaikuntanathan, and Michael Naehrig developed a prototype that demonstrates a somewhat homomorphic encryption technique. Their technique allows addition and a few multiplications to be performed on encrypted data, which in turn allows them to perform simple statistical functions on this ciphertext. In the article *Can Homomorphic Encryption be Practical?*, Lautner discusses the operations in particular that can be computed on encrypted data. The computations used may calculate averages, compute standard deviations, or perform other operations such as logistical regressions which can be used to predict the likelihood of specific health issues [5].

Microsoft researchers are optimistic about using this SWHE scheme as a foundation for developing an applicable FHE technique in the future. The implementation of this type of encryption on large scale data sets has also been explored. David Wu and Jacob Haven proposed a method for "computing the mean and variance of univariate and multivariate data as well as performing linear regression on multidimensional encrypted data" [8].

These types of encryption techniques are continuing to progress and evolve. Carson Sweet, founder of Cloudpassage, a technology company that provides cloud security services, stated that homomorphic encryption technology will need further developments before companies such as his will become interested its usage. He states that, "You can push encrypted data into a cloud service today, but it can't be indexed, searched, or operated on" [9]. These are some of the most desired or applied operations onto data sets today, and at the same time they are the current limitations of homomorphic encryption techniques.

4. Current Studies

4.1 Fully Homomorphic Encryption without Squashing Using Arithmetic Circuits

In 2009, Craig Gentry presented the first fully homomorphic encryption scheme in his PhD dissertation. This technique allowed one to compute arbitrary functions over encrypted data without the use of a decryption key [10]. In his method, Gentry first constructs a somewhat homomorphic encryption, then compresses the decryption circuit to a more uncomplicated form. It is then bootstrapped to obtain a fully homomorphic encryption procedure. In 2011 Craig Gentry and Shai Halevi devised an advanced approach that consisted of a fusion of SWHE and another type of encryption called multiplicatively homomorphic encryption (MHE). This novel process eliminated the need for the compression step Gentry originally proposed in his dissertation. In this method, Gentry and Halevi devised a system to condense the FHE ciphertext into a single ciphertext whose security was superior. This latter approach still required bootstrapping but "shows how to bootstrap without having to 'squash' the decryption circuit" [11].

4.2 Fully Homomorphic Encryption without Squashing Using Arithmetic Circuits

In the last year there has been a significant amount of advances made in developing fully homomorphic encryption techniques, but efficiency still remains an issue. "The existing schemes are still too inefficient for most practical purposes" [12]. In an effort to make fully homomorphic techniques more proficient, Gentry, along with fellow researchers Halevi, Peikert, and Smart studied a system to reduce the polynomial ring needed for homomorphic computation of the lower levels of a circuit. In ring theory or abstract algebra, a ring homomorphism is a function between two rings which preserves the operations of addition and multiplication. In their studies, they analyzed how to transform cipher texts over a big ring into small-ring ciphertexts that encrypt the same data. This procedure is known as ring switching. In their proposed method, they used a polynomial composition technique that splits a high degree polynomial into several lower degree polynomials. The idea behind this procedure was that the plaintext encrypted in the original large-ring packed ciphertext would be recovered as a simple linear function of the plaintexts encrypted in the smaller-ring ciphertexts [12]. By now transferring smaller sizes of ciphertext instead of larger ones, the efficiency of the process would improve.

5. Noise Reduction

There are some inevitable issues that arise when encrypting and decrypting data. In particular, the issue with noise reduction continues to be problematic during this process. Each conversion from plaintext to ciphertext has some noise associated with it. This noise continues to enlarge as one adds and multiplies ciphertexts, and as a result, the final ciphertext becomes indecipherable. A remedy to this concern was proposed by Gentry in his 2009 doctoral dissertation. He observed that if a noisy ciphertext could be decrypted and then re-encrypted, it would be restored with reduced noise [10]. The problem was that this decryption would require a secret key, which was not available. His solution was to run the ciphertext through the decryption algorithm, but with an encrypted version of the decryption key. This resulted in a new ciphertext that contained lower noise and was as secure as the original ciphertext.

Hayes provides a detailed explanation of Gentry's solution and the principles behind his work in his *Alice and Bob in Cipherspace* article. He explains Gentry's approach to designing a cryptosystem to model the process of a fully homomorphic encryption scheme. Gentry's system consisted of an encryption and decryption function which converted bits from plaintext to ciphertext and vice versa. Also included in his system was an evaluate function. This evaluate function can be thought of as a complete computer embedded into a cryptosystem. Its main purpose was to allow any computation to be performed on the ciphertext. This computation can be represented as a circuit or network where input signals traverse a series of boolean or logic gates. Because this evaluate function must be able to calculate any function, the circuit representing this function should be allowed to expand to any depth. The issue Gentry encountered, hence the motivation behind his work, was that after data has been encrypted, it contained a significant amount of noise. Gentry observed that if the noise continued to magnify, it would eventually overwhelm the signal. Because of this, the number of operations performed on the data would have to be restricted or inaccuracies would accrue. If the number of operations is limited then consequently, the circuit depth must also be limited. As a result, his system would not have been a fully homomorphic one but instead a somewhat homomorphic system because the number of operations performed on it is confined [2].

Hayes goes on further to expound on Gentry's solution for reducing the amount of noise with each operation. Gentry recommended setting a critical threshold and decrypting the data each time the noise level approached this point. The purpose of this process was to aid in filtering out the noise. Once this occurred, the data is then encrypted resetting the noise to its initial low level. As mentioned earlier, the issue with decrypting the data is that this requires the use of the secret key and the main advantage of using FHE is to allow computation on the ciphertext without this key. Gentry suggested running the evaluate function with the decrypt function. The decrypt function contains the secret key so the key supplied to the evaluate function is an encrypted version of the normal key. In other words the secret key supplied to the decrypt function in the evaluate function is the ciphertext produced when the encrypt function is applied to the plaintext (normal key) of the secret key. Hence, when the decrypt function is run with the encrypted key (secret key), the result is not plaintext; the decrypt function does not convert it back into plaintext. This produces a new encryption of the ciphertext with a decreased noise level. By re-encrypting and restoring the ciphertext, Gentry was able to create a fully homomorphic encryption system; the computer is now capable of handling a circuit of any depth and can implement any complex computation on the encrypted data [2].

6. Homomorphic Encryption Example

To demonstrate a practical use for a homomorphic encryption technique, we developed an algorithm that models this process. This is a proof of concept example that illustrates the feasibility of the approach. We considered a scenario where a hospital outsources its patient information to a contracting company. The hospital encrypts this information and sends this data to the contractor whose purpose is to determine the patient(s) with high blood pressure. Once the agency determines these persons they return this information, in encrypted form, back to the hospital. The hospital then decrypts the person's name; there is no need to provide the contractor with the key. This process is illustrated in Fig. 2.

In our algorithm, a list of patients and their blood pressures are read in plaintext form through an input file. This information contains personal data such as the patient's first and last name as well as their diastolic and systolic blood pressure numbers. The input text file used contained a small sample data set of 10 patients and was only 1 kB in size.

The algorithm encodes the data by converting the characters in each line of the file to its ASCII value and this number is multiplied by 100. The algorithm then searches for diastolic pressure numbers greater than 80 and systolic blood pressure numbers than are higher than 120. The values of 80 and 120 are also encrypted using the same key; they are also converted to their corresponding ASCII value and multiplied by 100. The algorithm compares the encoded value for the patients' blood pressures to the encoded values for 80 and 120 to determine which patients have high blood pressure. This is the operation the contractor will perform on the encrypted data. The encrypted names that correspond with these values are sent to an output file. The algorithm decrypts these names by converting them from integers back to characters and these names are sent to a separate file. The flowchart that describes the process of the algorithm is shown in Fig. 3.

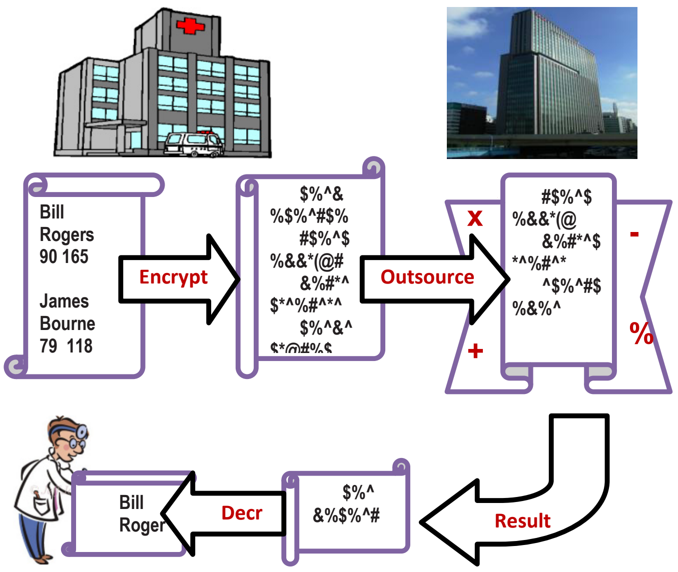


Fig. 2 Outsourcing encrypted data to contractor.

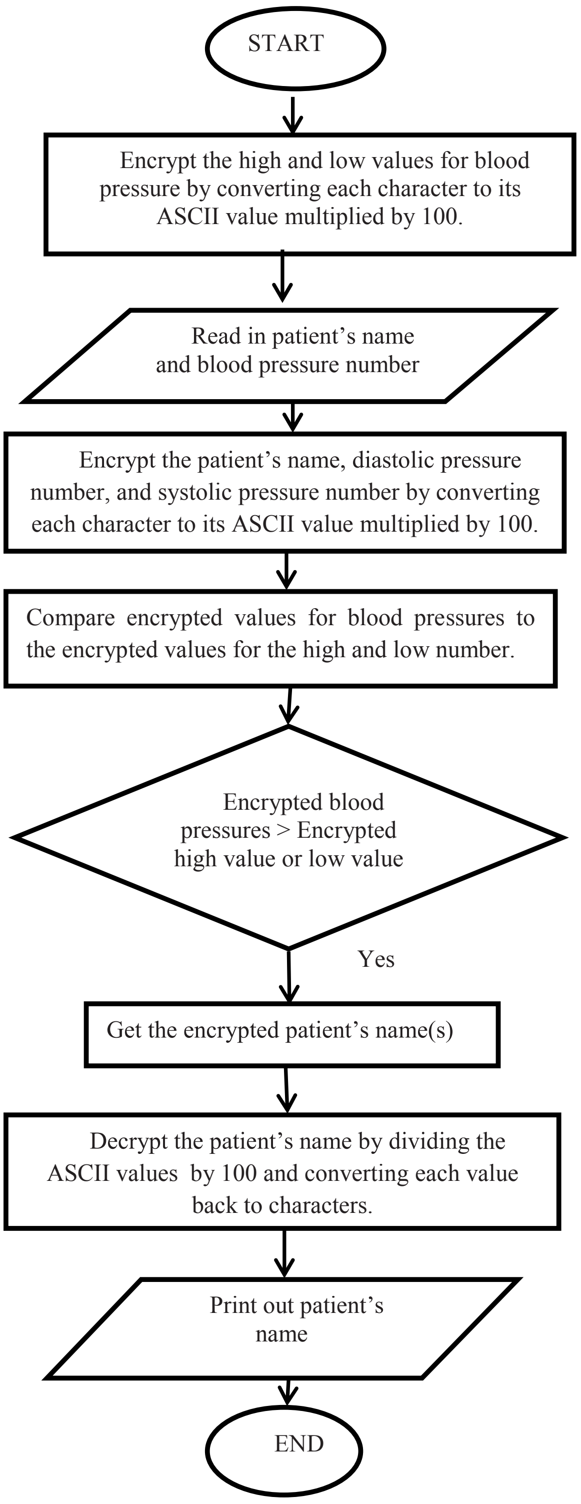

START
Encrypt the high and low values for blood pressure by converting each character to its ASCII value multiplied by 100.
Read in patient's name and blood pressure number
Encrypt the patient's name, diastolic pressure number, and systolic pressure number by converting each character to its ASCII value multiplied by 100.
Compare encrypted values for blood pressures to the encrypted values for the high and low number.
Encrypted blood pressures > Encrypted high value or low value
Yes
Get the encrypted patient's name(s)
Decrypt the patient's name by dividing the ASCII values by 100 and converting each value back to characters.
Print out patient's name
END

Fig. 3 Homomorphic algorithm flowchart

Because the blood pressures are declared as integers, their ASCII values are also integers; this allows us to perform comparison operations such as greater than and less than on the encrypted data. This may not always be the case depending on the parameters of the encryption scheme and the encryption key chosen. Our program outputs the names associated with those values. No comparison operations are performed on the patient's names, only on the numerical blood pressures. Although the encryption function is not the most robust function, this example illustrates the potential for implementing a homomorphic encryption method. While the data set used in our case is a small set, the use of homomorphic encryption schemes can be extended to sets containing a sizeable amount of information. In *Using Homomorphic Encryption for Large Scale Statistical Analysis*, Wu and Haven conducted two experiments where they performed linear regressions on large sets of encrypted data and calculated the variance and mean of this data. In these experiments the number of data points increased up to four million elements and one million elements respectively [8].

7. Other Applications

Lautner, Naehrig, and Vaikuntanathan also outline various potential real-world applications of homomorphic encryption. Its relevance can be seen in the realms of the financial and medical industries.

7.1 Medical Industry

In the past, private cloud medical records storage systems (Patient Controlled Encryption) have been proposed. In this system, all data for patient's medical records is encrypted by the healthcare providers before being uploaded to the patient's record in the cloud storage system. The patient has control over sharing and access to their records by sharing secret keys with specific providers. The patients and those they grant permission, have the ability to search the encrypted data, but the problem is that this system does allow the cloud to do any computations on the data other than searching for keywords [5]. With the use of FHE, the cloud will be able to perform operations on the encrypted data in support of the patient. For instance, monitors or other devices may be constantly streaming patient-related data to the cloud. With this type of implementation, the cloud will be able to perform operations on the encrypted data and send patients updates, alerts, or other pertinent information. Some sample encrypted plaintext input may include blood pressure numbers, blood sugar readings, ages of patients, and other information that patients and healthcare administrators may deem private.

7.2 Finance Sector

In the financial industry clients and businesses alike work with confidential information. Because of this the functions computed on the data as well as the data itself should remain private. "As an example, data about corporations, stock prices or their performance or inventory is often relevant to making investment decisions. Data may even be streamed on a continuous basis reflecting the most up-to-date information necessary for making decisions for trading purposes" [5]. Because of these factors, functions needed to perform these computations on this data must be exclusive. This information may contain new predictive models for stock price performance that may be the result of expensive research conducted by financial analysts. As to be expected, most companies would like to keep these models private to remain competitive in their respective fields and to protect their investments. If a FHE encryption technique is incorporated, some of these functions can be evaluated privately. For instance, a client can upload an encrypted version of the function to the cloud, such as a program where some of the evaluations are plaintext encrypted inputs. The streaming data could be encrypted to the customer's public key and uploaded to the cloud. The cloud service would then evaluate the private function by applying the encrypted description of the program to the encrypted inputs it receives. After processing, the cloud then returns the encrypted output to the customer [5].

8. Conclusion

Homomorphic Encryption is one of the most relevant types of encryption methods studied in the computational sciences today. Why is it important? All the techniques, including fully, somewhat, and partially homomorphic encryption allows one to securely transmit, store, and process encrypted data without jeopardizing the confidentiality of the information. The success of its use can be seen in many areas of industry with the finance and medical industries being some of the numerous fields. Researchers have benefited from incorporating the study of homomorphic encryption in their own work and will continue to excel in its use in the future.

Acknowledgements

We would like to acknowledge and thank Dr. Claude Tuner from Bowie State University, for his support and encouragement to study and research this pioneering and ground-breaking field of encryption.

References

1. (2012) The Miriam-Webster's Dictionary website. [Online]. Available: http://www.merriam-webster.com.
2. B. Hayes, "Alice and Bob in Cipherspace" in American Scientist vol. 100, 2012, paper 5 p. 362.
3. K. Thomas. "Sony Makes it Official: PlayStation Network Hacked," PC Computing, pp. 12, April 2011.
4. B. Adida. "Encryption is (mostly) not magic," Benlog, December 21, 2011,http://benlog.com /articles/2011/ 12/21/encryption-is-mostly-not-magic.
5. K. Lauter, M. Naehrig, V. Vaikuntanathan, "Can Homomorphic Encryption be Practical?," in CCS'11, 2011, paper 15, pp. 113 – 124.
6. D. Smith, M. Eggen, R. St.Andre, Transition to Advanced Mathematics, 7th ed., Boston, MA: Brooks/ Cole, 2011.
7. C. Stuntz. "What is Homomorphic Encryption, and Why Should I Care," Craig Stuntz Weblog, March 18, 2010, http://blogs.teamb.com /craigstuntz/2010/03/18/38566/.
8. D. Wu and J. Haven, "Using Homomorphic Encryption for Large Scale Statistical Analysis," 2012.
9. T. Simonite, "A Cloud that Can't Leak,". Computing News. August 8, 2011.
10. C. Gentry, (2009). "A Fully Homomorphic Encryption Scheme," Doctoral Dissertation, Symposium on the Theory of Computing, NY, New York, USA, 2009.
11. C. Gentry and S. Halevi, "Fully Homomorphic Encryption without Squashing Using Depth-3 Arithmetic Circuits," in IEEE' 11, 2011 pp. 107-116.
12. C. Gentry, S. Halevi, C. Peikert, N. Smart, "Ring Switching in BGV-Style Homomorphic Encryption" in SCN' 12 vol. 7485, 2012, pp. 19-37.
13. Encrypting Numerical Values [Picture]. Retrieved November 4. 2012, from: http://www.american scientist.org/issues/pub/alice-and-bob-in-cipherspace.
14. Homomorphic Concatenation [Picture]. Retrieved November 4, 2012, from: http://blogs.teamb.com /craigstuntz/2010/03/18/38566/.