Genetic Algorithm based key Generation for Fully Homomorphic Encryption

Majedah Alkharji[1], Mayyada Al Hammoshi[2], Chunqiang Hu[1], Hang Liu[1]
[1]Electrical Engineering and Computer Science
CUA, Washington, DC, USA,
{32alkharji, huc, liuh}@cua.edu
[2]School of Information Computer System
VIU, Fairfax, VA, USA,
mhammoshi@viu.edu

Abstract

Organizations and individuals have been moving to the cloud computing technology looking for effective and fast computing services. Confidential information is becoming more vulnerable to leak due to outsource computations to third-parties. The issue of data breaches could remove all the benefits organizations might get by moving to the cloud-based services. The main goal of securing information is to provide confidentiality, authenticity, integrity and data privacy. Data encryption is being widely employed to secure data. However, as users need to process data in the cloud, normal encryption schemes are practically inapplicable because they require the transmission of the secret keys to the server side to obtain the original data thus performing the required computation on the plaintext. Fully homomorphic encryption can be considered as an effective process that supports arbitrary computation on the ciphertext without requirement of decryption in the cloud. A genetic algorithm is a search operation based on natural genetic and natural selection. Applying the concept of Genetic Algorithms on cryptosystem provides strong randomness that hardens the attacking process for the ciphertext. In this paper, a method to use Genetic Algorithm to generate keys for the fully homomorphic encryption scheme is described and its effectiveness is examined. Moreover, some simple computations were performed on the encrypted data as well. Results showed that a GA generated key provides more randomness than other conventional methods used to generate public and private keys.

Keywords: Cryptography, Fully Homomorphic Encryption Schemes (FHE), Genetic Algorithm, Cloud Security, Confidentiality.

1 Introduction

Despite the efficient computing solution and economic advantages associated with cloud computing, users are very worried about security and confidentiality of data stored and processed in the cloud. One of the main solutions provided for safeguarding the data stored in the cloud is the encryption to harden the access by unauthorized personnel. Hence, in the era of "big data" and "cloud computing", encryption solutions must be applied to achieve the objective of data protection including confidentiality and integrity (Alkharji & Liu, 2016). One Time Pad (OTP) and Pseudo Random Number Generators (PRNG) are some of the traditional techniques utilized for generating unique keys. Many applications substitute these methods with more innovative one called Genetic Algorithm (GA). Selecting random key produced by GA gives the technique challenges that can hardly discover. Increasing the complexity included in the generation process can make the key more complex (Soni & Agrawal, 2013).

The usage of traditional symmetric or asymmetric (public key) encryption algorithms are not completely sufficient with cloud-based scenarios. When users need to process data in the cloud, normal encryption schemes will not work well because they require that the secret keys must be transmitted to the server side to decrypt and perform the calculations on the plaintext. Once encrypted data is opened for computations, it can't be processed safely within the cloud and this presents a major cloud computing constraint. This

drawback can be overcome by using Homomorphic Encryption (HE). Fully homomorphic encryption (FHE) supports an arbitrary number of addition and multiplication computations to be carried out on ciphertext without exposing the original data. The evaluation process encrypted outcomes matches the result of operations when performed on the plaintext. For secure data processing in cloud, FHE can be used to protect data privacy because it allows execution of operations on encrypted records without decryption. As such, the usage of FHE is a crucial step in enhancing cloud-computing security. More details about FHE found in (Alkharji & Liu, 2016) for the authors.

In this paper, we describe a method to use Genetic Algorithms to generate keys for the fully homomorphic encryption scheme and examine its effectiveness. Moreover, some simple computations were performed on the encrypted data as well. Results showed that a GA generated key provides more randomness than other conventional methods used to generate public and private keys. The remaining of the paper is organized as follows: section 2 gives more details about Genetic Algorithm (GA), while section 3 provides a review of the related works regarding FHE and GA algorithms. Section 4 discusses the proposed work, section 5 provides examinations to prove the security of the algorithm used, and finally, section 5 presents the conclusion and future work.

2 Genetic Algorithm

The paper tries to examine the key generating method for public key cryptography to be highly random and distinctive by utilizing genetic algorithm hence making PKC more secure. Genetic Algorithms (GA) is known as “randomized search and optimization algorithm” founded on the philosophy of “natural genetics and natural selection” (Soni & Agrawal, 2013; Mishra & Siddharth, 2013). In many applications, GA has proved to be a powerful and dependable technique (Sindhuja & Pramela, 2014). It could be applied in numerous ways in the PKC field either to generate keys, to enhance the standard encryption algorithm thus improving its degree of security, or to produce fresh symmetric/asymmetric algorithm. Fundamentally, genetic algorithm entails three operators utilized upon generation of the population including selection, crossover and mutation (Jawaid & Jamal, 2014). The application of GA is explored to establish the finest and more randomized key for the cryptographic algorithm. Superior complexity involved in the process of generating the key make the ciphertext tough for the cryptanalyst to decipher (Soni & Agrawal, 2013). Figure 1 (Jhingran et al., 2015) shows GA working mechanism.

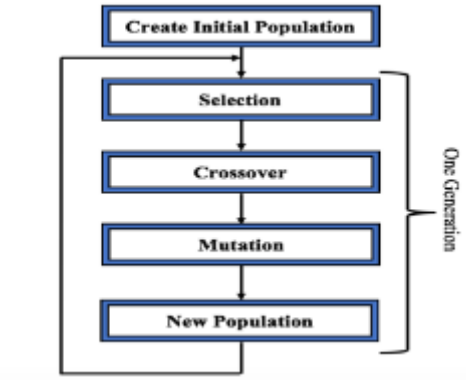

Figure 1: Genetic Algorithm Working Mechanism

3 Related Works

Researchers in the information security field are interested in exploring the contribution of genetic algorithm in the field of PKC. Numerous studies have conducted and analyzed the past efforts made in this field. Firstly, in 2013, Mishra & Bali presented an application that demonstrates the use of GA in the key generation using three different tests to check the replication of chromosome, thus providing effective and enhanced performance outcomes. Secondly, the approach suggested by (Naik P. & Naik G., 2013) illustrates an effort to use the randomness exhibited in crossover and mutation procedures for generating an asymmetric key pair for effective encryption and decryption of message. The number of crossover and mutation points coupled with permutation factor and arbitrary byte to be utilized in the private key

generation process dictates the span of the secret key, and thus the power of the algorithm. Thirdly, Soni & Agrawal (2013) introduced an approach founded on GA along with random number generator to add complexity. The key generation task experienced by several processes and the main one will be the population's fitness value. Moreover, in 2014, Jawaid & Jamal introduced another study in this field that demonstrates how to use this innovative approach of key generation in most effective manner along with its implementation by following the concept of natural key selection. Furthermore, Jawaid et al., (2015) also applied GA using autocorrelation test. The final key is chosen depending on the autocorrelation value and hence it is as unique and random as possible.

The first encryption scheme that is fully homomorphic based on ideal lattices was invented in 2009 by Craig Gentry. Gentry's innovation (2009a, 2009b) can be summarized into three stages including constructing a some-what homomorphic encryption (SWHE) scheme, "squashing" the decryption circuit until it is straightforward enough to be handled within the homomorphic capacity of the SWHE scheme, and finally, "bootstrapping" technique to get a FHE scheme. Since Gentry distributed the initial FHE system, this powerful discovery became a dynamic research subject and there has been huge enthusiasm for this scope. Researchers have been adopted other techniques e.g., integers instead of lattices, or learning with error. Consequently, the execution of the following schemes has been improved. But as a conclusion, FHE still needs an improvement regarding the limitation on efficiency, and operations overhead (Gentry, 2009a, 2009b; Brakerski & Vaikuntanathan, 2011b). More information about Gentry's work and the following FHE schemes along with their fundamental definitions, algorithms, semantic security, and possible applications are provided in (Alkharji & Liu, 2016).

The initial effort to improve Gentry's fully homomorphic public key encryption scheme was made by Smart and Vercauteren (2010). They executed a variation utilizing "principle ideal lattices" of prime determinant, thereby presenting a FHE scheme which has both relatively small key and ciphertext size. In order to obtain a faster FHE scheme than Gentry's invention, research by Stehle and Steinfeld (2010) depicted two main improvements considering ideal lattices and its examination. Gentry and Halevi (2010) proposed an optimized version of (Smart & Vercauteren, 2010), which permit to implement the squashing functionality, thus obtaining a bootstrappable scheme to convert to a FHE scheme. In their implementation, they proposed a number of major and minor optimizations along with facilitation that allow to execute all aspects of the scheme. In 2011, Smart & Vercauteren recalled their previous work (Smart & Vercauteren, 2010), and proved that it can support SIMD operations (Single Instruction, Multiple Data) in the finite field of characteristic two by modifying key generation. In 2011, Gentry and Halevi developed a new leveled FHE approach as the hybrid of a SWHE and a "Compatible Multiplicatively Homomorphic Encryption" (MHE) scheme. Basically, it demonstrated how to bootstrap excluding the method of "squashing" the decryption circuit. In 2012, Gentry et al. presented an approach that bypasses the reduction of one integer modulo another homomorphically to some degree, by using an arithmetic modulus near a power of two.
(Dijk et al., 2010) proposed a very simple SWHE framework (DGHV scheme) over the integers, in which all mathematical operations are done over the integers using only "elementary modular arithmetic computation" instead of ideal lattices over a "polynomial ring." Coron et al. (2011) proposed in their contribution a solution that minimize the public key size of the DGHV scheme (Dijk et al., 2010) from $O\,\square(\lambda^{10})$ to $O\,\square(\lambda^{7})$. Another research was proposed by Coron et al. (2012). The authors used a compression procedure and Modulus Switching to minimize the public key size of (Dijk et al., 2010) FHE cryptosystem over the integers (DGHV) from $O\,\square(\lambda^{7})$ to $O\,\square(\lambda^{5})$.

Brakerski and Vaikuntanathan (2011b) proposed a radical change to develop FHE schemes, known as (BV) scheme, whose security linked with the hardness of the decisional (standard) LWE assumption proposed by Regev (2010). This scheme is unique as it does not totally follow the Gentry blueprint (2009a, 2009b), and DGHV scheme (Dijk et al., 2010) over the integers. This resulting FHE scheme has very short ciphertexts, making it more effective than prior ones. Lauter et al. (2011) proposed in their work an implementation of the "Somewhat" public key encryption scheme from BV scheme (Brakerski & Vaikuntanathan, 2011b), while employing the computer algebra system Magma. Brakerski et al., (2012) constructed a leveled BGV cryptosystem on techniques of the (BV) scheme (Brakerski & Vaikuntanathan, 2011b) while using R-LWE problem from (Lyubashevsky et al., 2010). The main contribution in their work was a new strategy of constructing a leveled FHE schemes that is able to evaluate "arbitrary polynomial-size circuits", while eliminating the bootstrapping procedure proposed by Gentry. Another technique by Brakerski and Vaikuntanathan (2011a) is applied by getting rid of the additional presumption "circular

security." Their public key encryption scheme is relied on the "Polynomial Learning with Errors" (PLWE) assumption, which is a simplified form of R-LWE problem proposed by Lyubashevsky et al. (2010).

4. FHE Model Using GA Key Generation

This section elaborates all the practical methods including generating the keys for the FHE scheme using a GA algorithm, RSA modulo, FHE procedures, and examine its efficiency.

4.1 Key Generation Using Genetic Algorithm

Figure 2 demonstrates the key generation process using GA mechanism to be used in the FHE. Before starting GA process, key length should be chosen in order to decide the GA working parameters. To generate RSA with 4096-bit key, we would like to have a lots of random key stream bits (more than 4000), since we need to throw away and retest parts in case the primality test fails. So, in our case, the proposed GA key length that can provide maximum randomness of numbers is $4k$. When using 160-bits initial seed of PRNG or less, it is barely enough to generate RSA with 1024-bits key length. However, internal state of at least 4096-bits is required to securely generate RSA with 4096-bits key. The proposed parameters for the GA process as follows:

- Maximum number of generations = 10000.
- Initial population size = 64.
- Selection function = roulette-wheel.
- Crossover type = uniform crossover.
- Mutation rate = 0.003.

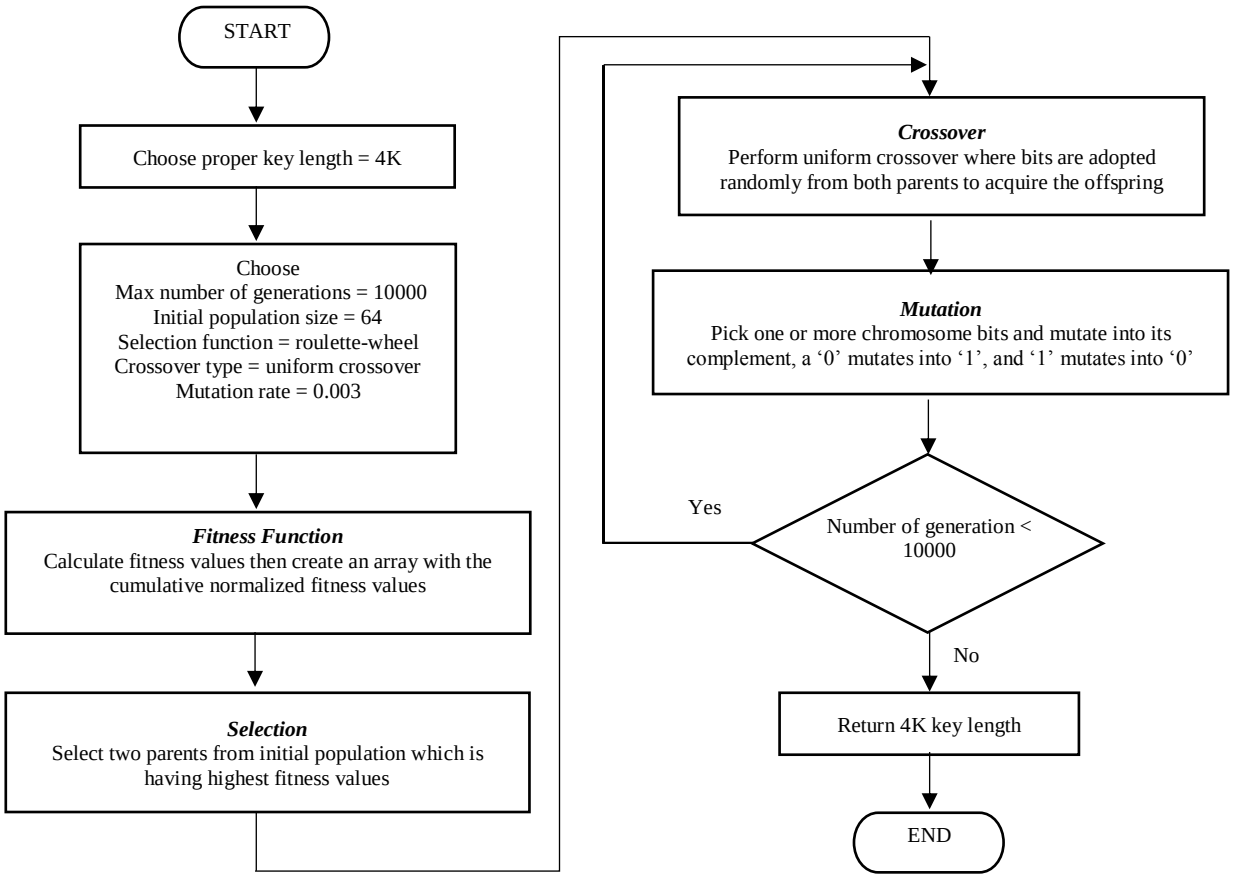

Figure 2: Key Generation Using GA

The following steps show the GA process starting from initial population all the way up to extracting the final population:

Step 1. Initial Population Generation. The GA's first step is to find the initial population, which is the process of randomly creating the set of individuals. The result of the initial population step is the first generation. A set of operators is applied to the preceding population to create the next generation (Jawaid & Jamal, 2014; Jawaid et al., 2015).

Step 2. Fitness Function. The second step is to calculate the fitness function which is served an extremely critical role in guiding genetic algorithm. It is an objective function that tests for the chromosome's suitability and defines how close the output is to the anticipated goal value. The individual chromosomes with higher fitness value is chosen from the initial population for the advanced process (Jawaid et al., 2015; Mishra & Siddharth, 2013). In this work, we normalized the fitness values obtained and further created an array with the cumulative normalized fitness outcomes.

Step 3. Selection. Selection level starts once the population has been generated based on the fitness values result. In this work, using roulette-wheel selection, two parents are picked randomly from the initial population in the selection process. The procedure is going to be repeated until there are enough selected individuals.

Step 4. Crossover. After creating a fresh population, parents are paired together by applying crossover operator. It refers to a genetic operator which assists in linking two parent's chromosomes from the selected parents to generate a fresh chromosome. This newly produced chromosome known as a child that picks a single trait of chromosome from each parent for the subsequent generation (Dutta et al., 2014; Jawaid et al., 2015; Mishra & Siddharth, 2013). Numerous forms of crossover techniques exist including single point crossover, two-point crossover, as well as uniform crossover (Sindhuja & Pramela, 2014). In this paper, uniform crossover is being used where bits are adopted randomly from both parents to acquire the offspring. Figure 3 illustrates the strategy of uniform crossover.

Parent1	0	1	0	0	1	1	**0**	0	1	1	1	**0**
Parent2	**1**	1	1	0	0	1	0	1	1	0	0	1

Offspring After Uniform Crossover	**0**	1	0	0	0	**1**	0	0	1	0	0	**1**
	1	1	1	0	1	1	0	1	**1**	1	1	0

Figure 3: Uniform Crossover

Step 5. Mutation. The last step of the GA is Mutation which summarize in altering one or more gene values within a chromosome from its original state. The bit inversion technique is being applied 30 times in this work which entails flipping one or more chromosome bits into its complement, for instance, a '0' mutates into '1' and the vice versa (Jhingran et al., 2015; Sindhuja & Pramela, 2014). After mutation, the solution implies that the much more fit chromosomes replace the less fit ones (Jawaid et al., 2015).

4.2 Modulo Keys Creation from GA Inputs

Figure 4 shows the detailed process of finding the public and secret vectors from the GA key outcome. The following steps elaborate the RSA modulo process:

- Pick two prime numbers from the key vector that matches the RSA conditions in order to provide maximum randomness and to prevent any manual interception in picking the key.
- Calculate $n = p. q$
- Calculate $\varphi(n) = (p - 1) . (q - 1)$
- Calculate e, such that $1 < e < \varphi(n)$ by finding the $gcd\ (e, \varphi(n))$, where $(e, \varphi(n))$ are co-primes.
- Calculate d, such that $(d * e)\ mod\ \varphi(n) = 1$

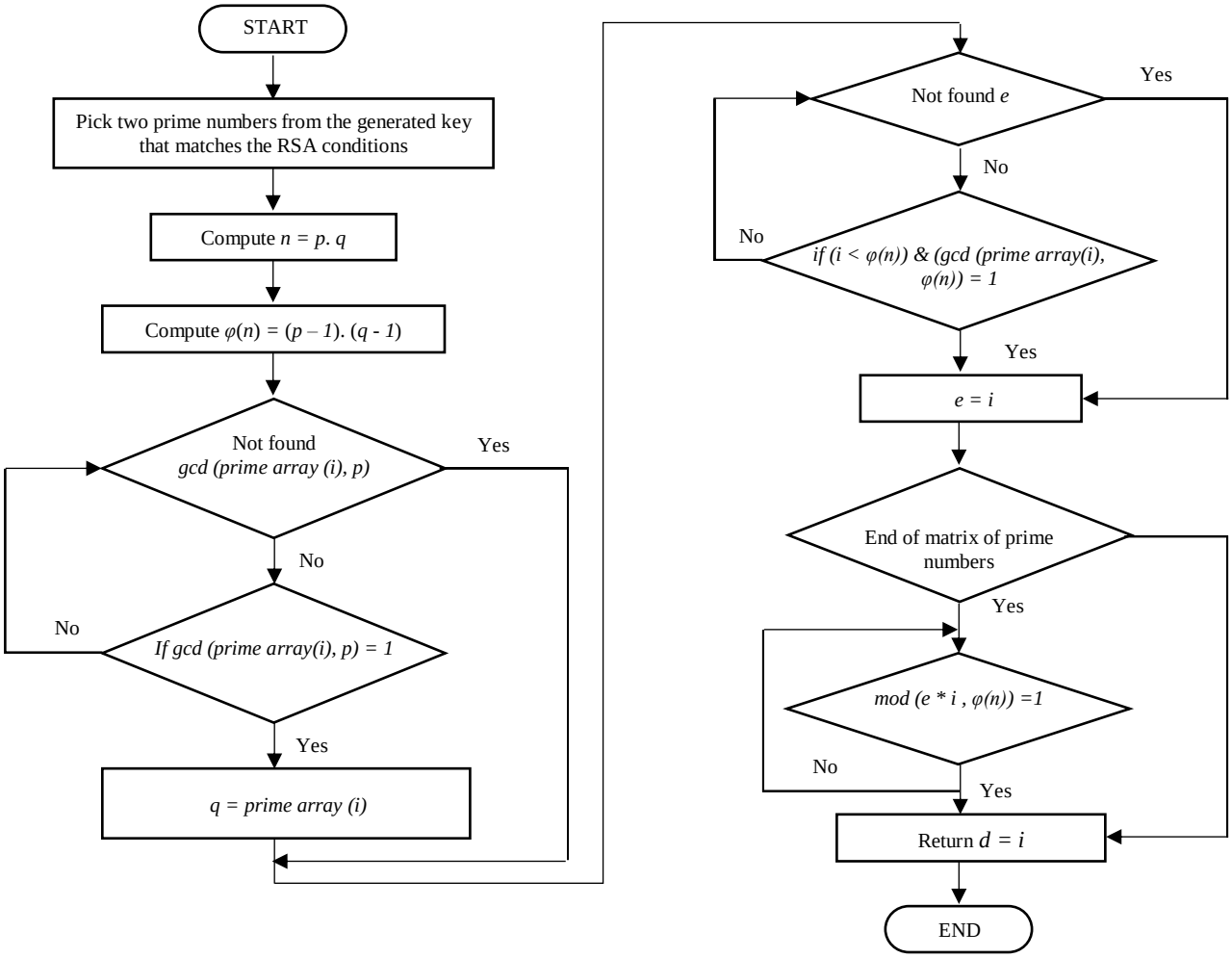

Figure 4: Create Public and Private Key

4.3 Encryption

The following algorithm (Figure 5) explains the encryption process. User will be requested to enter the message. A conversion procedure to ASCII code is performed on the entered text to start encryption method using modular exponentiation.

Encryption: Encrypt (plaintext, pk): ciphertext

Input:

$Plaintext \in Z_n$, where $Z_n = \{0, 1, \ldots, n\text{-}1\}$,

$pk = (n, e)$

Computation:

Compute $ciphertext = plaintext^{e} \bmod n$

Output:

$Ciphertext \in Z_n$

Figure 5: Encryption Algorithm

4.4 Fully Homomorphism

Fully homomorphism is the process of applying number of different operators on the ciphertext without decryption. These operators vary from mathematical to logical operators. Examples of operators are

addition, multiplication, oring, anding, …etc. In this paper, the evaluation process on the ciphertext are addition and multiplication. Figure 6 illustrates FHE mechanism on the encrypted text. Assuming that we may have two ciphertext (*cipher1, cipher2*), the length of two ciphertexts could be matched so the evaluation process will be run smoothly. However, if they are not matched, one of the following procedures will be performed according to the type of operation on the encrypted data:

- If the operation is addition, the additive neutral element would be added to the last element of the matrix with higher length.
- If it is multiplication, the multiplicative neutral element would be multiplied with the last element of the matrix with higher length.

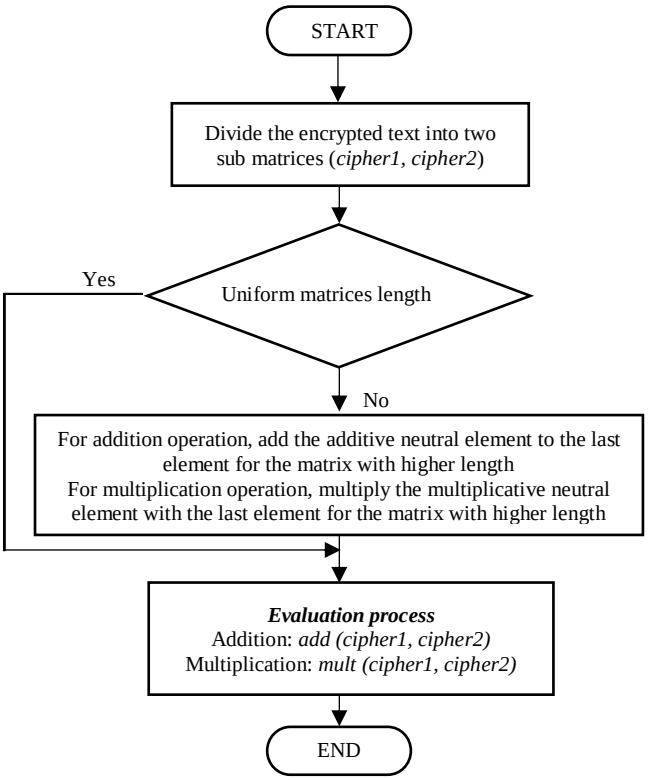

Figure 6: Fully Homomorphism

4.5 Decryption

The final stage at the receiver side is to recover the plaintext from the encrypted text. Secret vector exponent *d* will be used to get the original message. Figure 7 shows the decryption algorithm.

Decryption: Decrypt (ciphertext, sk): plaintext

Input:

Ciphertext) $\in Z_n$

$sk = (d)$

Computation:

Compute $plaintext = ciphertext^{\,d} \bmod n$

Output: (*plaintext*)

$plaintext \in Z_n$

Figure 7: Decryption Algorithm

5. Analysis & Result

5.1 Time Complexity

The main goal of applying genetic algorithm is to get the maximum randomness for the 4096-bits RSA key thus guaranteeing more security. The common way to measure the efficiency of an algorithm is to observe the execution time for a given size of inputs according to processor speed. In this paper, 10000 iteration including 30 mutation operations are performed on 64 chromosomes, each has length of 64. The execution is done on i3 Dual-Core Processor, with 4GB RAM. The performance tests ran for generation sizes k= 32, 48, 64, and number of generations g = 500, 1000, 2000, 5000, and 10000. For each pair of these values, the average execution time is measured and displayed in Table 1. Increasing the size of GA key length and the number of generations will cause slightly increase in running time (see Figure 8), but it's worth it because the possibility of getting prime numbers to compute modulo operation for RSA key generation will increase significantly.

In this work, the time taken to generate the final randomized key is 3.5760 milliseconds. But Soni & Agrawal (2013) stated that in their experiment "the time taken to generate key for 300 iteration with 10 new population each time and 10 crossover and mutation operations each iteration is 75.382 seconds." The huge difference in the population size and maximum number of generations in their case is expected to give less complexity but in contrast the upper bound on the execution time of our experiment is much more less with 64 initial population and 10000 iteration.

		Number of Generations				
		***g*=500**	***g*=1000**	***g*=2000**	***g*=5000**	***g*=10000**
Population Size	***K*=32**	0.1560 *ms*	0.2030 *ms*	0.3440 *ms*	0.7500 *ms*	1.4380 *ms*
	***K*=48**	0.187 *ms*	0.2960 *ms*	0.5160 *ms*	1.1720 *ms*	2.3130 *ms*
	***K*=64**	0.2500 *ms*	0.4370 *ms*	0.7650 *ms*	1.8280 *ms*	3.5760 *ms*

Table 1: Running time of the GA for *g* generations of population size *k*

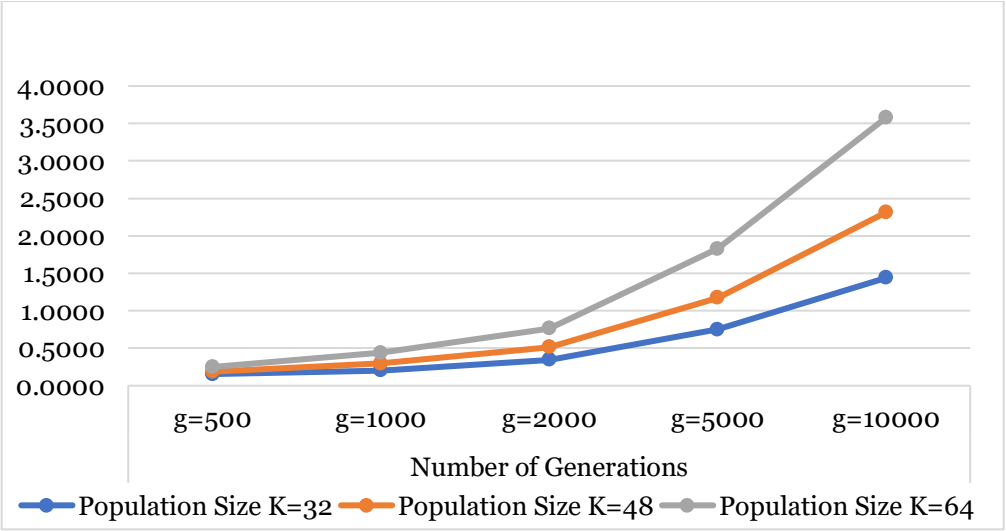

Figure 8: Time Complexity for GA Key Size

5.2 The Nature of Randomness

Degree of movement test is calculated to observe the level of randomness in the GA final key. Table 2 shows the sample of 10×10 key length represented in double symbols, while in practice 64×64 key length is analyzed. The test shows that the final population has '0' correlation with the initial one. The final key generated is purely random, unique, and unpredictable. Comparing with Mishra & Bali's result (2013), they observed the change for 8-bit key length sample in binary representation which is barely noticed. Their result shows some difference in bits between the two populations for a small sample.

	Length									
Population Size	341	479	388	442	502	424	295	261	261	454
	357	358	363	275	441	288	377	443	382	293
	301	458	478	435	345	425	309	394	297	407
	342	487	506	270	325	422	310	375	397	294
	454	275	372	350	464	462	509	270	484	498
	280	341	336	344	363	301	330	312	391	385
	412	358	326	294	314	349	435	297	419	268
	380	395	286	406	270	488	351	318	381	407
	260	328	347	445	404	350	399	345	494	489
	490	257	433	334	299	454	293	282	318	306

Table 2: Sample of Degree of Movement

Results depict the effectiveness of the proposed scheme under two different scenarios which summarized as follows. First, the time taken to generate key from 10000 iteration with 64 populations and 30 mutation operations is 3.5760 milliseconds. Second, the test performed to check the nature of randomness shows that the final population has '0' correlation with the initial one. Both results prove that the key generated for FHE is non-repeating and unique.

6. Conclusion & Future Work

Fully homomorphic encryption addresses the cryptographic needs for secure cloud computing applications. Therefore, FHE has attracted a lot of interests in recent years. This paper argues that strong, non-repeating, high-quality random keys generated by GA will enhance the security of FEH schemes, making it more difficult for the cryptanalysts to break the data. The two directions in our future work will be to use machine Learning with Error (LWE) to harden the process for the attacker to find the secret key, and to give formal security proof on our scheme.

References

Alkharji, M., & Liu, H. (2016). Homomorphic Encryption Algorithms and Schemes for Secure Computations in the Cloud. *Proceedings of 2016 International Conference on Secure Computing and Technology*, Virginia International University, Fairfax, VA.

Brakerski, Z., Gentry, C., & Vaikuntanathan, V. (2012). Fully Homomorphic Encryption without Bootstrapping. *ITCS '12 Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, 309-325. doi:10.1145/2090236.2090262

Brakerski, Z., & Vaikuntanathan, V. (2011a). Fully Homomorphic Encryption for Ring-LWE and Security for Key Dependent Messages. In P. Rogaway (Eds.), *LNCS: vol. 6841*. Advances in Cryptology – CRYPTO 2011 (pp. 505–524). Berlin, Heidelberg: Springer. doi:10.1007/978-3-642-22792-9_29

Brakerski, Z., & Vaikuntanathan, V. (2011b). Efficient Fully Homomorphic Encryption from (Standard) LWE. *FOCS'11 Proceedings of the 2011 IEEE 52nd Annual Symposium on Foundations of Computer Science,* 97-106. doi:10.1109/FOCS.2011.12

Coron, J. S., Mandal, A., Naccache, D., & Tibouchi, M. (2011). Fully Homomorphic Encryption over the Integers with Shorter Public Keys. In P. Rogaway (Eds.), *LNCS: vol. 6841*. Advances in Cryptology – CRYPTO 2011 (pp. 487–504). Berlin, Heidelberg: Springer. doi:10.1007/978-3-642-22792-9_28

Coron, J. S., Naccache, D., & Tibouchi, M. (2012). Public Key Compression and Modulus Switching for Fully Homomorphic Encryption over the Integers. In D. Pointcheval, & T. Johansson (Eds.), *LNCS: vol.* 7237. Advances in Cryptology – EUROCRYPT 2012 (pp. 446–464). Berlin, Heidelberg: Springer. doi: 10.1007/978-3-642-29011-4_27

Dutta, S., Das, T., Jash, S., Patra, D., & Paul, P. (2014). A Cryptography Algorithm Using the Operations of Genetic Algorithm & Pseudo Random Sequence Generating Functions. *International Journal of Advances in Computer Science and Technology (IJACST)*, 3.

Gentry, C. (2009a). A fully homomorphic encryption scheme. Ph.D. dissertation, Stanford University. Retrieved from https://crypto.stanford.edu/craig/craig-thesis.pdf.

Gentry, C. (2009b). Fully homomorphic encryption using ideal lattices. *STOC'09 Proceedings of the forty-first annual ACM symposium on Theory of computing*, 169–178. doi:10.1145/1536414.1536440

Gentry, C., & Halevi, S. (2010). Implementing Gentry's Fully-Homomorphic Encryption Scheme. In K. G. Paterson (Eds.), *LNCS: vol. 6632*. Advances in Cryptology – EUROCRYPT 2011 (pp. 129–148). Berlin, Heidelberg: Springer. doi:10.1007/978-3-642-20465-4_9

Gentry, C., & Halevi, S. (2011). Fully Homomorphic Encryption without Squashing Using Depth-3 Arithmetic Circuits. *FOCS'11 Proceedings of the 2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, 107-109. doi:10.1109/FOCS.2011.94

Gentry, C., Halevi, S., & Smart, N. P. (2012). Better Bootstrapping in Fully Homomorphic Encryption. In M. Fischlin, J. Buchmann, & M. Manulis (Eds.), *LNCS: vol. 7293*. Public Key Cryptography – PKC 2012 (pp. 1–16). Berlin, Heidelberg: Springer. doi:10.1007/978-3-642-30057-8_1

Jawaid, S., & Jamal, A. (2014). Generating the Best Fit Key in Cryptography using Genetic Algorithm. *International Journal of Computer Applications (IJCA*), 98, 0975 – 8887. doi:10.5120/17301-7767

Jawaid, S., Saiyeda, A., & Suroor, N. (2015). Selection of Fittest Key Using Genetic Algorithm and Autocorrelation in Cryptography. *Journal of Computer Sciences and Applications (JCSA)*, 3, 46-51. doi:10.12691/jcsa-3-2-5

Jhingran, R., Thada, V., & Dhaka, S., (2015). A Study on Cryptography using Genetic Algorithm. *International Journal of Computer Applications (IJCA)*, 118, 10 – 14. doi:10.5120/20860-3559

Lauter, K., Naehrig, M., & Vaikuntanathan V. (2011). Can Homomorphic Encryption Be Practical.? *CCSW'11 Proceedings of the 3rd ACM workshop on Cloud computing security workshop,* 113-124. doi:10.1145/2046660.2046682

Lyubashevsky, V., Peikert, C., & Regev, O. (2010). On Ideal Lattices and Learning with Errors over Rings. In H. Gilbert (Eds.), *LNCS: vol. 6110*. Advances in Cryptology – EUROCRYPT 2010 (pp. 1-23). Berlin, Heidelberg: Springer. doi: 10.1007/978-3-642-13190-5_1

Mishra, S., & Bali, S. (2013). Public Key Cryptography Using Genetic Algorithm. *International Journal of Recent Technology and Engineering (IJRTE)*, 2, 150-54.

Naik, P. G., & Naik G. R. (2013). Asymmetric Key Encryption using Genetic Algorithm. *International Journal of Latest Trends in Engineering and Technology (IJLTET)*, 3. doi: 10.13140/2.1.3621.0889

Regev, O. (2010). The Learning with Errors Problem. *CCC '10 Proceedings of the 25th Annual IEEE Conference on Computational Complexity,* 191-204. doi:10.1109/CCC.2010.26

Sharma, I. (2013). Fully Homomorphic Encryption Scheme with Symmetric Keys. Master Thesis, Rajasthan Technical University. Retrieved from https://cryptome.org/2013/10/homo-crypto-sym.pdf.

Sindhuja, K., & Pramela, D. S. (2014). A Symmetric Key Encryption Technique Using Genetic Algorithm. *International Journal of Computer Science and Information Technologies (IJCSIT)*, 5, 414-416.

Smart, N. P., & Vercauteren, F. (2010). Fully Homomorphic Encryption with Relatively Small Key and Ciphertext Sizes. *PKC'10 Proceedings of the 13th international conference on Practice and Theory in Public Key Cryptography*, 420-443. doi:10.1007/978-3-642-13013-7_25

Smart, N. P., & Vercauteren, F. (2011). Fully Homomorphic SIMD Operations. *Designs, Codes and Cryptography,* 57–81. doi:10.1007/s10623-012-9720-4

Soni, A., & Agrawal, S. (2013). Key Generation Using Genetic Algorithm for Image Encryption. *International Journal of Computer Science and Mobile Computing (IJCSMC)*, 2, 376 – 383.

Stehlé, D., & Steinfeld, R. (2010). Faster Fully Homomorphic Encryption. In M. Abe (Eds.), *Lecture Notes in Computer Science: vol. 6477.* Advances in Cryptology – ASIACRYPT 2010 (pp. 377–394). Berlin, Heidelberg: Springer. doi:10.1007/978-3-642-17373-8_22

van Dijk, M., Gentry, C., Halevi, S., & Vaikuntanathan, V. (2010). Fully Homomorphic Encryption over the Integers. In H. Gilbert (Eds.), *LNCS: vol.* 6110. Advances in Cryptology – EUROCRYPT 2010 (pp. 24-43). Berlin, Heidelberg: Springer. doi:10.1007/978-3-642-13190-5_2