A BRIEF IDEA ON THE HOMOMORPHIC ENCRYPTION FOR DATA SECURITY

[1]Sabarna Ghosh Dastidar, [1] Sayantan Dasgupta, [1]Salini Bhattacharyya, [1]Somasree Majumder, [2]Pritha Ghosh, [2]Renaissance Adhikari, [1]Ahan Bhattacharjee, [1]Soumendu Bhattacharya, [3]Biswadip Basu Mallik

[1]Department of Electronics and Communication Engineering

Institute of Engineering & Management

D1 Management House, Salt Lake, Sector V, Kolkata-700091, West Bengal, India.

[2]Department of Electrical Engineering

Institute of Engineering & Management

D1 Management House, Salt Lake, Sector V, Kolkata-700091, West Bengal, India.

[3]Department of Basic Science and Humanities

Institute of Engineering & Management

D1 Management House, Salt Lake, Sector V, Kolkata-700091, West Bengal, India.

Corresponding author: gdsabarna353@gmail.com

Abstract

We have stepped into a world of undeniable data breach fatigues. Thus to ensure data security, encryption schemes come into action, one such highly talked about encryption nowadays is "Homomorphic Encryption" which is mainly used to compress data for their easy storage involving secure transmission and processing on cloud without compromising on privacy since special keys are needed for primary encryption and final decryption. Homomorphic encryption allows operation on two ciphertexts to give an encrypted (coded) result which when decrypted (decoded) maps to result of the operation, if it would have been on plaintext. It can be either multiplicative like the RSA or additive like the Pallier cryptosystem. Here we also focused on ideal lattice based public key encryption scheme which is almost bootstrappable. Multihop homomorphic encryption also has vital roles to play here. A strong homomorphic encryption is one which is resistant to all attacks using various algorithms. Thus our main objective is to allow encrypted computing on data, minimize memory usage and to save energy and time. Our presentation will try to focus on the researches done so far and also on the success rates of the scheme which is affecting the real world.

Keywords: *Homomorphic, Public Key Encryption, Decryption, RSA, Pallier Cryptosystem, Cloud, Ideal Lattice, Bootstrappable*

INTRODUCTION:

Fully homomorphic encryption (FHE) is a cryptographic primitive which facilitates arbitrary computations on encrypted data. It has been dubbed the holy grail of cryptography, an exclusive goal which could solve the IT world's problems of security and trust. The development of fully homomorphic encryption is a revolutionary advance, greatly extending the scope of the computations which can be applied to process encrypted data homomorphically. We know that the development of cloud storage and computing platform allows users to outsource storage and computations on their data, and allows business to offload the task of maintaining data-centers, yet, concerns over loss of privacy and business value of private data is an overwhelming barrier to the adoption of cloud services by consumers and business alike. But, after Craig Gentry [1] published his idea in 2009, the cloud can perform the computations on behalf of the user and return only the encrypted result with the help of fully homomorphic encryption and thus data can remain confidential while it is processed, enabling useful tasks to be accomplished with data residing in untrusted environments and research in this area exploded with regard to improving the schemes, implementing and applying them. The main purpose of this scheme is to allow one to evaluate arbitrary circuits over encrypted data without being able to decrypt. FHE has numerous applications like, it enables private queries to a search engine, i.e. the user submits an encrypted query and the search engine computes a succinct encrypted answer without ever looking at the query clearly. More broadly, FHE improves efficiency of secure multiparty computation. In highly regulated industries, such as health care, homomorphic encryption can be used to enable new services by removing privacy barriers inhibiting data security. Here, we focused on the technology constructed by the Craig Gentry, which begins, with a somewhat homomorphic "bootstrappable" encryption scheme working while the function is the scheme's own decryption function and then we will discuss the methodology of Craig Gentry, how bootstrappable encryption gives fully homomorphic encryption through recursive self-embedding without reducing the depth of the decryption circuits of the arbitrary circuits that the scheme can evaluate and this construction makes use of hard problems on ideal lattices. Our paper will provide a great help to the beginners for starting a massive study on homomorphic encryption.

BACKGROUND OF CRAIG GENTRY'S FHE SCHEME:

In order to develop the application for this scheme, several software libraries implementing the latest schemes of Fully Homomorphic Encryption were found. HElib [3], the most complete, portable and well maintained, written in C and C++ implements a FHE scheme including both the BGV scheme with modulus switching and bootstrapping, SIMD operations and the permutation network optimisation. It also supports several other speed enhancements such as multi-threading and proposes an easy access to tweaking the many optimisations. But, for quite low level of HElib library, PhD student Grant Frame released open source Integrated Development Environment (IDE) for HElib, called HEIDE (2015) which uses python. Louis Aslett released "An R package for fully homomorphic encryption" to implement Fan and Vercauteren Scheme (2015). Then, in January 2016, WejDai from Vernam Group at Worcester Polytechnic Institute developed a source FHE library called CuHE to implement Doroz-Hu-Sunar (DHS) SwHE scheme based on the

Lopez-Tromer-Vaikuntanathan (LTV) scheme [3]. Krypto was another open source library, released in November 2015, by the company kryptonostic. Leo Ducas and Daniele Micciancio released the open source FHEW library in January 2015 which achieves a 1-bit NAND gate homomorphic encryption in less than asecond with bootstrapping. The last library found was released by Jean-Sebas tien Coron in 2012 and implements the DGHV FHE scheme in python with the SAGE mathematical library. So, these software libraries actually serve as a proof of practical use of their respective scheme. Indeed, some schemes such as Gentry's original FHE scheme could never be implemented due to its space complexity.

Craig Gentry's initial FHE scheme was abbreviated by SwHE, in which ciphertexts have some "noise", coming from randomness added for security purposes. The noise of a ciphertext grows with each additional operation performed on this one. The ciphertext can't be decrypted anymore once the noise grows above a certain threshold. To make of his SwHE scheme a FHE scheme, Gentry developed a technique called bootstrapping which decrypts and recrypts the ciphertext at each operation to reduce its associated noise. This allows an infinite amount of operations to be performed on ciphertexts. However, each bootstrapping is slow so performing an addition or multiplication will always be slow. Since then, many improvements have been made to make a more efficient, usable FHE scheme.

Apart from several small enhancements such as a reduction of the size of FHE keys, there are three main improvements since Gentry's first FHE scheme. The first one is the support of SIMD (Single Instruction Multiple Data) operations. Here, a plaintext is not a single value but a vector of plaintext elements, before being encrypted and this encrypted vector results in a "packed" ciphertext. Thus, using SIMD operations, this improvement allows to perform an operation on multiple entries at once and reduces either the time cost associated with bootstrapping or the number of levels needed in case of a leveled FHE. The second enhancement is that if one needs to perform an operation with the element of one plaintext vector at the position 2 and that of the other at the position 3, it spends significant amount of time to unpack and then repack the ciphertexts, but the solution came by bringing the concept of permutation network allowing to permute elements without needing to unpack and repack everything which provides a great flexibility and efficiency to the scheme. The third one is the modulus switching introduced with BGV scheme, aiming to provide an alternative to the slow bootstrapping. In this scheme, there is a predefined number of levels, proportional to the maximum number of operations to be performed on the ciphertext. This one is actually proportional to the accumulated noise added with each operation. The number of multiplication operations to be performed on a ciphertext should determine the number of levels. But for a high number of levels, bootstrapping becomes faster than the BGV scheme.

A BRIEF DISCUSSION ON RSA CRYPTOSYSTEM AND HOMOMORPHIC ENCRYPTION:

The first homomorphic encryption scheme was the basic RSA [5] invented by Rivest, Adleman and Shamir which is multiplicative, i.e. for two given ciphertexts $\psi_1 = \pi_1^e \bmod N$ and $\psi_2 = \pi_2^e \bmod N$, where, *e* is sometimes referred to as an *encryption exponent* or *public exponent*. Then, a ciphertext can be computed such that that $\psi \leftarrow \psi_1.\psi_2 = (\pi_1.\pi_2)^e \bmod N$ that encrypts the product of original plaintexts and here, these plaintexts are large primes, so, Euler's theorem, Euler's phi function and Chinese remainder theorem plays a vital role in this cryptosystem. It must be mentioned that RSA cryptosystem is not meant to replace symmetric ciphers because of being relatively slower than ciphers like AES (Advanced Encryption Standard, a subset of the Rijndael block cipher developed by two Belgian cryptographers, Vincent Rijmen and Joan Daemen) and so it is less commonly used to directly encrypt user data. More often, RSA passes encrypted shared keys for symmetric key Cryptography which in turn can perform bulk encryption-decryption operations at much high speed that's why it is widely used especially for key transport and digital signatures. But basic RSA is deterministic and so it is not semantically secure. Then the possibilities of fully homomorphic encryption scheme were introduced for the first time by Rivest, Adleman and Dertouzos which is called a privacy homomorphism and Gentry suggested a solution to it. But, before entering into this part of discussion, we should know that a homomorphic public key encryption scheme ξ has four algorithms [2].-

1. KeyGen_ξ , which takes a security parameter λ and a positive integer d as input and gives a triple of secret key, public key and evaluation key(sk, pk, evk)
2. Encrypt_ξ , which takes $pk^{(d)}$ and a plaintext π from the set$(\pi_1, \pi_2, \ldots, \pi_t)$ as input and outputs a ciphertext, ψ
3. Decrypt_ξ , which takes $sk^{(d)}$ and a ciphertext, ψ as input and outputs $\text{Decrypt}_\xi(sk,\psi)$
4. Evaluate_ξ , which takes pk, a circuit C from a permitted set C_ξ of circuits and a tuple of ciphertexts $\Psi = (\psi_1, \psi_2, \ldots, \psi_t)$ and outputs a ciphertext ψ.

Now, the definition says that, if, $\psi \leftarrow \text{Evaluate}_\xi(pk, C, \Psi)$, then $\text{Decrypt}_\xi(sk, \psi) = C(\pi_1, \pi_2, \ldots \pi_t)$

Now, there are three definitions about Homomorphic Encryption Scheme.

1. ξ is homomorphic for circuits in C_ξ if ξ is correct for C_ξ and Decrypt_ξ can be expressed as a circuit D_ξ of size poly(λ).
2. ξ is fully homomorphic if it is homomorphic for all circuits.
3. A family of schemes $\{\xi^{(d)}: d \in Z^+\}$ is leveled fully homomorphic if they all use the same decryption circuit, $\xi^{(d)}$ is homomorphic for all circuits of depth at most d (that use some specified set of gates Γ), and the computational complexity of $\xi^{(d)}$'s algorithm is polynomial in λ, d, and (in the case of Evaluate) the size of C.

A BASIC INTRODUCTION TO BOOTSTRAPPABLE ENCRYPTION:

Following Gentry, we can construct homomorphic encryption for circuits of any depth from one that is capable of evaluating just a little more than its own decryption circuit. Here, the research started with a public key encryption scheme ξ_1 which uses ideal lattices and is homomorphic for shallow circuits. A ciphertext ψ has the form $\mathbf{v} + \mathbf{x}$ where $\mathbf{v}$ is in ideal lattice and $\mathbf{x}$ is an 'error' or 'offset' vector encoding the plaintext π. Now, ξ_1 is homomorphic for the shallow circuits only because it causes a decryption error where the error vector grows with the ring operations using addition and especially multiplication. To prevent this, we could refresh a ciphertext if we could completely decrypt it, so the new ciphertext will encrypt same plaintext but its noise will be small. Here, we must mention that decryption itself is a good reduction algorithm but we can't hand over the secret key. Now, this noise should remain below a certain threshold to perform the decryption correctly. So, this process is such a type of process where we publish the encryptions of the secret key bits and the process will evaluate the scheme's own decryption circuit homomorphically. Applying this setting and repeating this process we can evaluate the circuits of any arbitrary depth by keeping the error small with no requirements of secret key and this is the idea behind bootstrapping.

Now, for getting bootstrappable feature, an encryption is needed to be able to evaluate not only its decryption circuit but also slightly augmented versions of it, so that we can perform non-trivial operations on plaintexts and make progress through a circuit. But, here we have to be familiar with one more algorithm i.e. Augment$\xi^{(\delta)}$ [2], which takes $pk^{(\delta)}$, a circuit C_δ of depth at most δ with gates in Γ and a tuple of input ciphertexts Ψ_δ (each input ciphertexts should be under pk_δ). The algorithm augments C_δ with D_ξ ; which is called the resulting circuit $C^{\dagger}_{\delta\text{-}1}$.

Now the mathematical working of the bootstrappablity can be understood by considering the following algorithm where the plaintext space is P and D_ξ is a Boolean circuit in C_ξ . Let, (sk_1, pk_1) and (sk_2, pk_2) be two ξ key-pairs, ψ_1 be an encryption of $\pi \in P$ under pk_1 and $\overline{sk_{1j}}$ be an encryption of j-th bit of the first secret key sk_1 under the second public key pk_2 . Now, the concept of bootstrapping describes a Recrypt [1] step in the algorithm that allows for the more elegant decryption inside encryption. So, for the algorithm we considered here for example, Recrypt takes those things as input and outputs an encryption of π under pk_2 .

Recrypt $(pk_2 , D_\xi , (\overline{sk_{1j}}), \psi_1)$.

Set $\overline{\psi_{1j}} \leftarrow \text{Encrypt}_\xi (pk_2 , \psi_{1j})$

Set $\psi_2 \leftarrow \text{Evaluate}_\xi (pk_2 , D_\xi ,((\overline{sk_{1j}}), (\overline{\psi_{1j}})))$

Output ψ_2

Here, the remark is that the Recrypt algorithm allows the owner of sk_1 to generate a tag that enables an untrusted proxy to convert an encryption of π under pk_1 to an encryption of π under pk_2 , but not the reverse.

Now, there are some theorems related to bootstrappablity [1], which are as following.-

1. One can construct a (semantically secure) family $\{\xi^{(d)}\}$ of leveled fully homomorphic encryption schemes from any (semantically secure) bootstrappable encryption scheme

ξ . But, one drawback lies in this theorem i.e. $\xi^{(d)}$ is merely leveled fully homomorphic because of its dependence on d, and when it is independent of d, the scheme will be fully homomorphic.

2. Let, C_ξ be a set of circuits with respect to which ξ is homomorphic. We say that ξ is bootstrappable with respect to Γ if $D_\xi(\Gamma) \subseteq C_\xi$. It gives the definition of Bootstrappable Encryption.
3. Let ξ be bootstrappable with respect to a set of gates Γ . Then the family $\{\xi^{(d)}\}$ is leveled fully homomorphic (for circuits with gates in Γ).
4. Let A be an algorithm that breaks the semantic security of with advantage. Then, there is an algorithm B that breaks the semantic security of ξ with probability at least $\in/l(d+1)$, and time poly (l, d) times that of A, where D_ξ takes a secret key and ciphertext as input formatted as elements of $P^{l(\lambda)}$. Here we denote the set of g-augmented decryption circuits, gate $g\epsilon\ \Gamma$, by $D_\xi(\Gamma)$.

If ξ is KDM-secure [1], the public key can be shorten to include pk and an encryption of sk under pk, a "self-loop" rather than an acyclic chain of encrypted secret keys.

A LITTLE INTRODUCTION TO IDEAL-LATTICE CONCEPT:

This concept comes from Lattice cryptography which is one of the latest developments in theoretical cryptography. Now, the most interesting fact is that a lattice can be created with interesting algebraic structure which enables cryptographers to create cryptosystems that can do things which were previously impossible. A fully specification of an n-dimensional lattice contain n^2 entries, which is huge. So, a very popular approach came i.e. the application of "ideal lattices", which are additionally symmetric and here we can specify a public key using only n entries. The notion of bootstrappablity gives us a new angle on constructing FHE. It suggests looking at the encryption schemes whose decryption algorithms have low circuit complexity. But, on the other side, encryption schemes using lattices or linear codes have very simple decryption algorithms typically dominated by a matrix-vector multiplication, an operation in NC1. It is not enough only to minimize the circuit complexity of decryption but we also should maximize the evaluative capacity of the scheme to evaluate its own (augmented) decryption circuit. While an additively homomorphic encryption scheme can easily be constructed from ordinary lattices, a scheme is needed having both additive and multiplicative homomorphism to evaluate arbitrary circuits and this thinking leads us to focus on ideal lattices. The basic introduction will be like that, an abstract construction is implemented using a polynomial ring and ideal lattices [1] i.e. let R=Z[x]/f(x), where $f(x) \in Z[x]$ is monic and of degree n. Now, we view an element $\mathbf{v} \in R$ both as a ring element and as a vector $\mathbf{v} \in Z^n$. The ideal (**v**) generated by v directly corresponds to the lattice generated by the column vectors $\{\mathbf{v} \times x^i \bmod f(x): i \in [0, n-1]\}$, this is called rotation basis of ideal lattice (**v**). An ideal $\mathbf{I} \subset R$ need not be principal- i.e. have a single generator- and a basis B_I of I need not be a rotation basis. The Hermite normal form of a lattice I is an upper triangular basis that can be efficiently computable from any other basis of **I** which makes it well-suited to be a public key. The main application of this concept is that they are used to design a wide range of cryptographic

algorithms by using computationally hard lattice problems to design robust cryptographic functions and in cryptanalysis also and its security solely relies on the hardness of finding approximate solutions to lattice problems.

ANALYSIS OF HOMOMORPHIC ENCRYPTION SUPPORTED BY HElib SOFTWARE LIBRARY:

The methods for analysis of homomorphic encryption scheme can be categorized as the limits of HElib and its potential, settings of level parameter and logic gates and their complexity, designing combinational and sequential circuits, binary multiplication operation, average operation, Euclidean division etc. But, here we discuss some parts of this concept very briefly as follows.

HElib supports homomorphic subtraction and negation operation besides addition and multiplication operations on ciphertexts and these operations are done in the field Fp^d [3] where p is prime and results are modulo p and it will be meaningful if $r < p$. The solution to unlock the potential from these operations is to implement binary logic compatible with FHE from the ground up. Addition and multiplication operations actually implement XOR and an AND logic gates and there is one another logic gate i.e. NOT, and we are able to derive all other logic gates using these three gates. Now, multiplication operation adds a significant amount of noise to the ciphertext which increases time complexity of a homomorphic function. So, to prevent this, HElib supports a leveled homomorphic encryption, consisting in setting a level parameter L at the key generation stage, which is proportional to the maximum number of homomorphic encryptions acted on a ciphertext, so that, if L is lower, operations will be faster. So, we need to avoid those logic gates also, which use multiplication operations (i.e. AND, NAND, OR, NOR gates) as per possibility. HElib supports one more operation, i.e. SIMD mode which requires N integer values to be encrypted and packed into a ciphertext and the number of plaintext slots N depends on level parameter L and security parameter k. The packed ciphertexts are used such that each slot is independent from the other ones and these grow larger with the increase of L and K.

IMPLEMENTATION OF HOMOMORPHIC ENCRYPTION:

Homomorphic encryption has a huge implementation.

1. Homomorphic encryption mainly consists of 4 steps Key Generation Encrypt Decrypt and Evaluate.
2. In the core Application Program Interface or API development where the source file provides complex features and several features , the homomorphic encryption is incorporated .Here ciphertext are stored in the unordered or in the dictionary format where firstly it simplifies the code and secondly it requires less storage.
3. It is even implemented in the logic gates {and, or, nor) using the additive and the null operators. Here it creates n copies of the bits and left shifts the copied rows with the binary numbers and overwrites on the respective result.

4. In Fully Homomorphic Encryption or FHE we apply functions on the encrypted data where it allows a secret key to be issued using a master key dependent on function f(x).Obscuration is the scheme which raises the security of the encrypted data. This also finds an application in the consumer piracy allowing the function to be computed and choose the advertisement for each user while the advertising remains encrypted.
5. It is also implemented in the BGV scheme along with optimum utilization of ciphertext packing.
6. A homomorphic encryption is bootstrappable if its homomorphic entity includes the all the augmented and decryptive circuits. A homomorphic encryption scheme can be transformed into a compact leveled homomorphic scheme.
7. It even has an implementation in efficient encoding of integers for arithmetic operations, choice of parameters, mean of variance computation, potential improvements and optimization of the communication with cloud. We can even explain the choice of parameters in the scheme.
8. Somewhat homomorphic encryption [8] is done using the computer algebra. It has a wide application in the development of cloud storage.
9. Cryptography is based on the ideal lattice concept where interrelated problems are on the lattices and we can learn with and from the errors.
10. Homomorphic encryption is a bitwise encryption in which bits are transformed into partial n bit string pack. Cloud security provide valuable security to users and the system provide the data to the cloud to do any other computation .It is also implemented in the machine learning applications like logistic regression where data are encrypted bitwise rather than its elements.

APPLICATIONS OF FHE:

This part of the paper throws light upon the innumerable applications of the different tastes of Homomorphic Encryption. Some need fully homomorphic encryption which can compute anything on encrypted data while, on the other hand, some just need somewhat homomorphic encryption which is more restricted.

It is basically divided into two parts among which the first one signifies the current scenario and the applications that are presently feasible and the second one emphasizes the constructions where homomorphic encryption is the building block.

 # Practical Applications of FHE: Although still slow, homomorphic encryption has been proposed for many different practical uses. This part lists those applications that are conceivable with the present technology.

1. Consumer Privacy in Advertising: These days many users are concerned about the privacy and security of their data, in this case their preferences and location. This problem has been faced in many different methods.

In one approach, Jeckmans et al. [6] proposed that a user wants recommendations for a product based on the tastes of the user's friends with the condition of confidentiality. The proposed system applies homomorphic encryption to allow a user to obtain recommendations from friends without the identity of the recommender being revealed.

In some other approach Armknecht and Strufe [7] wrote that a recommender system builds upon a very simple but highly efficient homomorphic encryption scheme where a user gets encrypted recommendations without the system being aware of the content.

2. Medical Applications: In this field Naehrig et al. [4] brought in front that a patient's medical information is continuously uploaded in encrypted form to a service provider where the user is the data owner, so the data is encrypted under the user's public key and only the user can decrypt. Then the service provider computes on the encrypted data.

3. Data Mining: Data mining from large data sets, on one hand, serves the advantage of great value but in return affects the user's privacy. The scheme actually uses functional encryption which is often considered as a common confusion.

4. Financial Privacy: In the case of financial privacy homomorphic encryption is used to upload both the data and the algorithm in encrypted form in order to outsource the computations to a cloud service. Although, homomorphic encryption is not known for keeping the algorithm secret, but is rather part of obfuscation research.

5. Forensic Image Recognition: The police and the other law enforcement agencies use weapons similar to this to detect illegal images in a hard drive, network data streams and other data sets.

Here a somewhat homomorphic encryption scheme can be used where the company's legitimate network traffic stays private while at the same time the police database is encrypted where the later in turn is compared by the company with the hashed and encrypted picture data stream.

 # Homomorphic Encryption Schemes as Building Blocks: Homomorphic Encryption Schemes can be used to construct cryptographic tools such as zero knowledge proofs, signatures, MACs

1. Zero Knowledge Proofs: Gentry [1] told that homomorphic encryption can also be used in the construction of Non-Interactive Zero Knowledge (NIZK) proofs of small size. A standard NIZK proof is attached to prove that each ciphertext encrypts either 0 or 1 and that the output of the evaluation encrypts 1.

2. Outsourcing Storage and Computations: Perhaps the most direct application of FHE is for outsourcing storage and computation without revealing sensitive information. Considering a small company trying to move its computing facilities to the cloud, but that is worry of the cloud provider having access to the company's confidential information. FHE provides an elegant solution to this conundrum. The company can keep the information in the cloud in encrypted form, and the cloud

provider can process information in this form and send only the processed result back to the company to be decrypted.

3. PIR and other private queries: Another direct application of FHE is to enable private queries to a database or a search engine. The simplest such example is private information retrieval, where a server is holding a large database and a client wants to retrieve one record of this database without the server learning which record was retrieved. FHE lets the user encrypt the index of the record that it wants to retrieve. The server can evaluate the function on the encrypted index, returning the result to the client, who can decrypt it and obtain the plaintext record.

4. Signatures: The homomorphic signature scheme can evaluate arbitrary circuits with maximal depth d over signed data and homomorphically produce a short signature which can be verified by anybody using the public verification key. This work also introduces the notion of Homomorphic Trapdoor Functions (HTDF), one of the building blocks for the signature construction. HTDF themselves are based on the Small Integer Solution (SIS) problem.

5. Multiparty Computation (MPC): MPC is a technology that allows us to compute on encrypted values. Using MPC a number of servers can jointly compute any function without learning the inputs to the function.

A small example is that of a group of people desiring to compute their average salary, without any individual group member revealing their personal salary to others. Using MPC it is possible for them to jointly compute a function which takes as input the secret salary of each group member and reveals only one piece of information i.e. the average of all these secret numbers.

ACKNOWLEDGEMENT:

This paper would have been impossible without the support and mentoring of our advisor, Prof. Biswadip Basu Mallik. We owe special thanks to him for the idea of this paper and the initiative to write it. We can't thank him enough for introducing us to Homomorphic Encryption, which allowed us to develop this paper. We would also like to thank him for endorsing our paper and teaching us most of what we know about Fully Homomorphic Encryption (FHE).

We also drew on many sources for this tutorial, most extensively on Craig Gentry's PhD thesis.

We, all the members of this group, are also thankful to each other because we have received lots of inputs and support through the discussions on this topic. We all have collaborated with each other to work in this area and became able to overcome several design restrictions for our paper through many discussions. These discussions have led to significant performance optimisations.

Finally, we would like to thank the professors of math and computer science departments of our college for their encouragements and suggestions. Without them we could not have finished this paper.

CONCLUSION

The project has shown several facts. First, it has demonstrated how real cloud computing calculations could be implemented with Homomorphic Encryption (HE). The limits of current HE schemes were explained and analysed. Several tricks were found to overcome those and to design efficient homomorphic binary circuits. Some complex circuits such as the pure average circuit clearly showed the bounds of HE. But with enough design, cleverness and compromises, it was also demonstrated that HE can be used in a few cases.

Many of the optimisations presented in our paper are general purpose and can be applied to solving challenging problems dealing with large datasets in other application domains.

In this paper we have simplified and structured the jungle of definitions in the field of HE. We investigated whether existing applications need HE as a solution to their problems.

There is still much work to be done. Current schemes have some way to go to be practical in daily applications. Thus we can expect continuing focus on making existing schemes more efficient and on constructing new efficient schemes.

A framework for group HE schemes has been presented in our paper. All secure schemes which go beyond simple group homomorphic operations are noise-based and one of the main challenges is to control the noise. In fact this is often the reason why fully homomorphic encryption schemes are considerably less efficient. A unified view on somewhat/fully Homomorphic encryption schemes may be very useful in gaining a better understanding of the expected security and on the possible design space.

All in all the topic of FHE is an interesting and challenging research area with great potential, and there is much to be done. However, if research (specifically the advancement of efficiency) continues at its current pace, we are confident that real-world applications may be right around the corner.

REFERENCES

1. Craig Gentry. A Fully Homomorphic Encryption Scheme. PhD thesis, Standford University, pages: 9-15, 43-50, 2009.
2. Craig Gentry. Fully Homomorphic Encryption Using Ideal Lattices. In The 41st ACM Symposium on Theory of Computing (STOC), pages: 169-171, 2009.
3. Q.D.MCGAW. Homomorphic encryption: Cryptography for Cloud computing. Final Year Project Report, Imperial College London, pages: 11-24, 2016.
4. Kristin E. Lauter, Michael Naehrig, Vinod Vaikuntanathan. Can homomorphic encryption be practical? In Christian Cachin and Thomas Ristenpart, editors, Proceedings of the 3rd ACM Cloud Computing Security Workshop, CCSW, pages: 113-124. ACM, 2011.
5. C. Paar and J. Pelzl. Understanding Cryptography. Copyright Springer-Verlag, pages: 173-175. From: wiki.crypto.rub.de.

6. Arjan Jeckmans, Andreas Peter, and Pieter H. Hartel. Efficient privacy-enhanced familiarity-based recommender system. In Jason Crampton et al., editors. Computer Security – ESORICS 2013, volume 8134 of Lecture Notes in Computer Science, pages: 400-417. Springer, 2012.
7. Frederik Armknecht and Thorsten Strufe. An efficient distributed privacy-preserving recommendation system. In The 10th IFIP Annual Mediteranean Ad Hoc Networking Workshop, Med-Hoc-Net 2011, pages: 65-70. IEEE, 2011.
8. Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. 17th March 2012. From: https://eprint.iacr.org/2012/144.pdf.