International Journal of Engineering Research in Computer Science and Engineering (IJERCSE) Vol 3, Issue 2, February 2016

Fully Homomorphic HEROIC Framework

[1]Sreelekshmi S, [2] Devi Dath
[1] PG scholar [2] Assistant Professor
College Of Engineering Perumon, Kerala, India
[1]sreelekshmis2392@gail.com ,[2] devinirmal05@gmail.com

Abstract- **Outsourcing to cloud is now so common due to its enhanced features. Different services can be outsourced and one such useful service is computation. As with outsourcing, the data owners are always concerned about the data they share with a third party. Security and privacy of data are the most important concerns of the data owner. In order to protect the data inside the processor of a third party a framework called the HEROIC framework (Homomorphically EncRypted One Instruction Computer) was developed. Single instruction architecture and homomorphism made it stronger. In this framework partially homomorphic scheme was introduced which supported only additive homomorphism. This paper suggests a fully homomorphic scheme which makes the HEROIC framework more secure and time efficient. The scheme supports both additive and multiplicative homomorphism.**

Index Terms— **HEROIC framework, homomorphism, outsourcing, single instruction architecture**

I. INTRODUCTION

Cloud provides an environment that can be accessed by a person with an authorization. The different services offered by cloud include paas (platform as a service), iaas (infrastructure as a service) and saas (software as a service).the different applications of cloud are developing due to the low maintenance and up gradation cost for the cloud user. One such application is outsourcing computing [1] to cloud. Here the data owner is having advantage of low memory usage, time and cost efficiency. These features make the outsourcing spread from small tax computation to large firms outsourcing bigger coding to third party. Outsourcing computations to cloud force the data owner to share the data and program code with the third party as shown in figure 1. The sharing of data with a third party brings in the complex face of data security. The data owners are always aware of possible attacks that manipulate or hinder the safe sharing of data with the third party.

One could easily choose some strong cryptographic method to overcome such attacks. The data owner could possibly make his data securely reach the third party. In normal cryptographic methods the data before processing need to go through the decryption process. That is the data, which the data owner safely send to the third party is accessible inside the processor in its sensitive form. This data is vulnerable to large number of attacks. The data may be vulnerable to different eavesdropping. One such attack is the Hardware Trojan attack in which the Trojan circuits of the processor look physically same but the physical composition of material used is modified. In order to overcome such processor based attacks the data owner uses homomorphic encryption scheme [2]. These are public key cryptosystem schemes which allow manipulation of data in its encrypted form. The instructions to be executed are also in the encrypted form. The main drawback of this is that the current computer architectures (CISC and RISC) are not meant for data security or privacy rather for performance and efficiency.

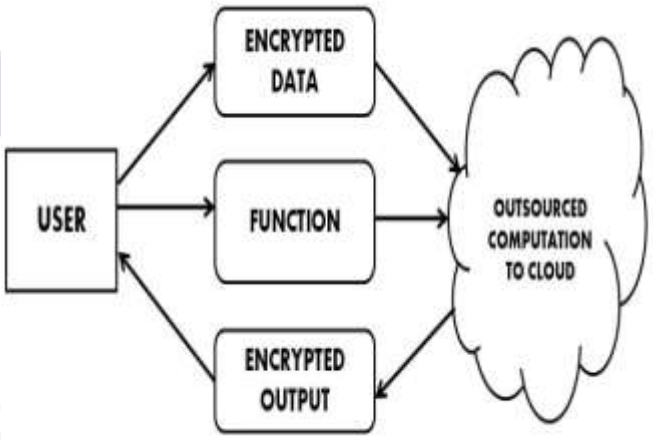

Fig 1: Outsourcing computation

The lack of architecture supporting encrypted instruction led to the design of HEROIC framework [3]. In this the features of homomorphism and single instruction are being used. The single instruction architecture [4] being Turing complete [5] gave the solution for understanding the instructions in its encrypted form. Here in this framework it used partially homomorphic scheme, Paillier scheme. In this framework only additive computations can be performed. In order to perform computations like multiplication n additions are to be performed. This main drawback is cleared in this paper by replacing the partially homomorphic scheme by a fully homomorphic scheme. In this scheme a somewhat homomorphic scheme is converted into fully homomorphic scheme [6] using Gentry's scheme [7]. The different features of the paper like single instruction and homomorphic scheme are included in the coming sections. Section III gives a detailed view of homomorphism and Section IV brings out the powerfulness and simplicity of

single instruction architecture. Section V gives the architecture of the fully homomorphic framework and its design considerations. The rest of the paper is organized as Section VI describing the experimental results and finally to the conclusion and directions to future work.

II. RELATED WORKS

Different schemes where proposed to protect data inside the processor like Fletcher [8] proposed a block cipher for encryption outputs of CPU. It slows down the system by 12 to 13.5 times. Another such scheme was suggested by Suh [9] which proposes a tamper resistant processor. This scheme had a drawback of high overhead. These schemes are vulnerable to side channel attacks. The most recent development in this field is the HEROIC (Homomorphically Encrypted One Instruction Computer) framework. This scheme uses single instruction architecture and homomorphic features to secure outsourced data. The main drawback of this system is the use for partially homomorphic scheme which reduces the efficiency of the system.

III. HOMOMORPHISM

In order to protect data, the different schemes can be used to encrypt. This conversion of data into a form unknown is called cipher text. The encryption can be either symmetric or asymmetric. In symmetric encryption same key same key is shared by both sender and receiver. It is a faster system and it requires sharing a single key in a secure way. In case of asymmetric encryption it uses two different keys instead of one key. It is slower compared to the symmetric scheme. Homomorphism is a type of asymmetric encryption.

The homomorphic encryption [10] allows meaningful manipulation of data in encrypted data without revealing the actual plain data. The homomorphism encryption scheme consists of four algorithms.

Key Gen (λ)

- The security parameter λ as the input and two keys (*sk, pk)* are obtained as output where sk is the secret key and pk is the public key

Encrypt (pk, π)

- The public key *(pk)* and plaintext (*π*) are the inputs to the algorithm and output is the cipher text (ψ)

Decrypt (sk, ψ)

- The private key *(sk)* and cipher text (ψ) are the inputs to the algorithm and output is the plaintext (*π*)

Evaluate (pk, C, ψ)

- The public key *(pk)*, a circuit C with t inputs and a set of cipher texts and plaintext (ψ1, ψ2…. ψt) are the inputs to the algorithm and output is the cipher text (ψ)

The different types of homomorphic algorithm include partial homomorphism, somewhat homomorphism and fully homomorphism as show in figure 2. Partially homomorphic scheme support either additive or multiplicative homomorphism. In case of somewhat homomorphism limited number of additions and multiplication before decryption fails. Depth of decryption is limited in case of somewhat homomorphism. A scheme is fully homomorphic if it correctly evaluates all circuits and the decryption circuit is bounded to some fixed polynomial.

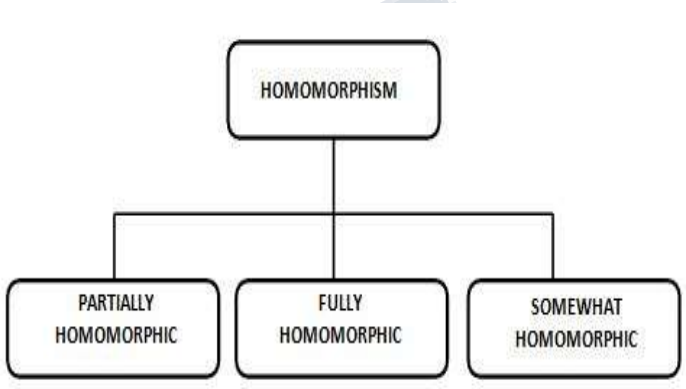

Fig. 2 Types of homomorphism

The fully homomorphic scheme is generated from a somewhat homomorphic scheme by using Gentry's transformation. It is a simple mechanism which merely supports only addition and multiplication over integers. The somewhat homomorphic algorithm has four algorithms for key generation, encryption, evaluation and decryption. The algorithm can be defined as

Key Gen (λ)

- The security parameter λ as the input and two keys (*sk, pk)* are obtained as output where sk is the secret key and pk is the public key
- *sk* a random odd n bit number
- $p= (2Z+1) \cap [2^{n-1},2^n)$
- $q_0,\ldots,q_t = Z \cap [0,2^{\zeta}/sk]$
- choose $r_0, \ldots r_t = Z \cap [-2^{\gamma},2^{\gamma}]$
- $X_0=q_0+p+2r_0$
- $X_i=[q_ip+2r_i]$
- $pk =\{x_0,\ldots..x_t\}$

Encrypt (pk, m∈{0,1})

- r random set chosen from S subset of {1,2…, t} and get cipher text as output

Decrypt

- $m'=[[c]_p]_2$
- $m'= (c \bmod 2)\ [c/p] \bmod 2$

Evaluate (pk, C, $c_1 \ldots c_t$)

- The public key *(pk)*, a circuit C with t inputs and a set of cipher texts and plaintext ($\psi_1, \psi_2 \ldots \psi_t$) are the inputs to the algorithm and output is the cipher text (ψ)

Squashing the decryption circuit can be done by adding extra information to public key. The information can be used to post process cipher text. For a somewhat homomorphic scheme to be converted to fully homomorphic it should be bootstrappable. Let ε be a homomorphic scheme with security parameter λ and $C_\varepsilon(\lambda)$ is a set of circuits with respect to which it is correct. ε is bootstrappable if $D_\varepsilon(\lambda)$ is subset of $C_\varepsilon(\lambda)$for every λ.

In somewhat homomorphic scheme the number of computations is limited due to the increased noise component in the cipher text. Increased noise term makes it intolerable as it makes it difficult to get the correct decrypted output. The depth of circuit or degree of polynomial defines the number of operations that can be performed correctly. To reduce the noise, instead of evaluating cipher text and secret key as in normal computation Gentry did the same homomorphically on the encryption of those bits. Thus the obtained result is not plaintext but a cipher text for the required plaintext as shown in figure 3. This process is called cipher text refresh procedure as it lowers the noise component of the actual cipher text.

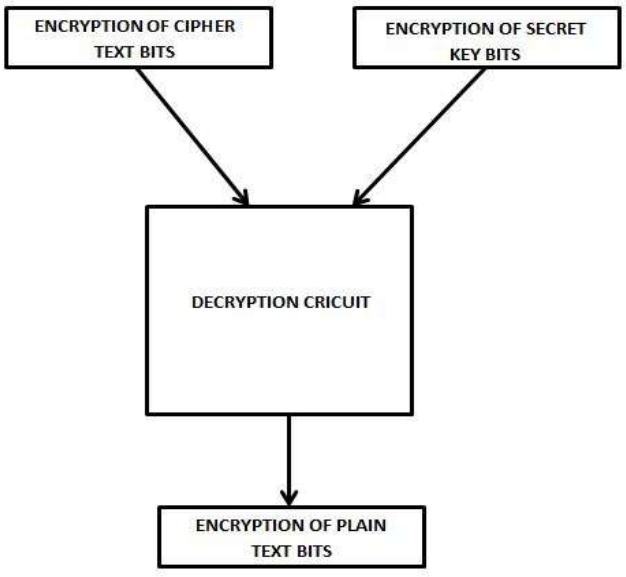

Fig. 3 Cipher texts refresh procedure

In order to make the scheme fully homomorphic it should be bootstrappable for which the depth of circuits that can be evaluated in cipher text should exceed that of decryption polynomial. For attaining such a situation Gentry used a sparse subset of values on addition gives the secret keys rather than the original secret key. This procedure is known as squashing decryption.

IV. SINGLE INSTRUCTION MECHANISM

In HEROIC Framework the instructions are in the encrypted form. The existing architectures like RISC and CISC does not support the processing of encrypted instructions. These architectures were designed for performance and efficiency, security was never a design option. These architectures bring down the security of sensitive data as they may lead to eavesdrop. One of such Trojan attack is the use of hardware Trojans [11] inside the processor. This type of Trojan was developed by modification of the material's physical composition. Like these virtually undetectable Trojans large number of attacks are possible, which are of great threat to the sensitive data of the data owner.

The need for different op codes is ruled out by the use of single instruction architecture. The OISC (One Instruction Set Computer) supports computations in the encrypted domain. Single instruction architectures should be Turing Complete so as to use it for computational application. The idea of Turing Complete is that it should be capable of recognizing any algorithm. The rules [12] that define Turing Completeness are sequence rule, selection rule and rule. The sequence rule defines the flow of the control through instructions. The selection rule brings in the idea of decision making, whether control should move to next instruction or jump to another set of instructions. Repetition rule helps in flow of control without a decision making stage.

The four basic computer instructions used to follow the above mentioned rules are LOAD, STORE, INCrement and GOTO. One instruction set computer supports only one instruction. Micro operations of a single instruction can be performed using the four instructions. These properties make Single Instruction Architecture and alternative of RISC. A wide variety of OISC variants are available. Addleq, subleq, pleg are some commonly used variants.

The Subleq micro operation is a set of instruction given as

Mem[B]=Mem[B]−Mem[A]
if Mem[B]<= 0then goto C
else goto next instruction

The memory value of B is subtracted from A and based on the output the next instruction to be executed is decided. If value is less or equal to zero the next instruction stored in C is to be executed otherwise the succeeding instruction is to be executed. The flow of control in subleq instruction set is shown in Fig 4

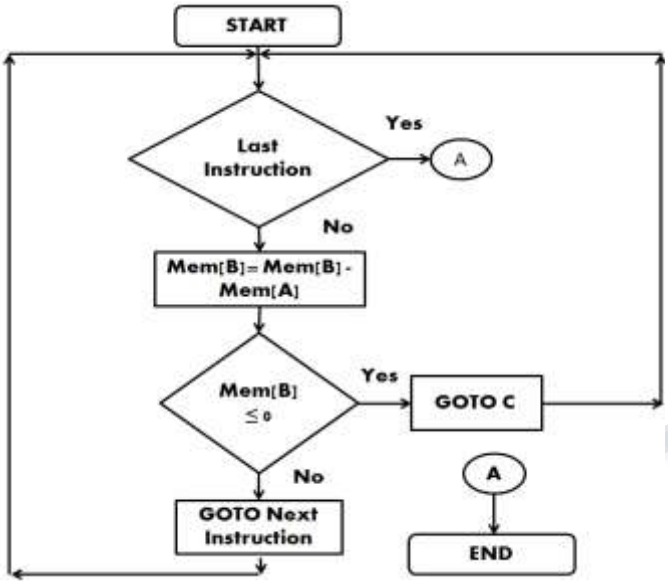

Fig.4 Single instruction architecture flow chart

V. FRAMEWORK

The instruction and data that are to be used or send to the third party for outsourcing are to be encrypted. For securing the data and program even inside the processor it is homomorphically encrypted. First of all the instruction of the program code are converted to single instruction format. The subleq complier is used for this purpose. It helps to generate a set of three address values. The same program is then encrypted with fully homomorphic scheme and these two codes that single instruction are send to the third party. The required data with same encryption is send to the third party for outsourced computations. On receiving both instruction and data the third party starts computation. Since the computer architectures like CISC and RISC does not support encrypted instruction format, single instruction code generated. From the beginning of single instruction code each line of three values are taken and performs Mem[B] = Mem[B] - Mem[A] and if the value is less than or equal to zero then the next instruction in memory location C is to be executed. Otherwise the next instruction is to be executed next. This helps in finding the correct flow of instructions even if it has loops or goto statements.

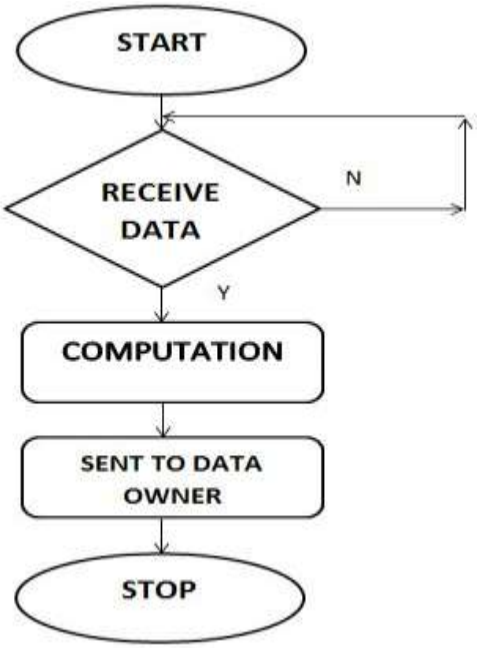

Fig. 5 Third Party

Figure 5 and 6 shows the working of third party and data owner respectively. One special case of computation is the addition of two negative numbers. Consider an example of two negative numbers -20 and -1. In order to perform addition we take two's complement of each number that is, $(2^{16}-20)$ and $(2^{16}-1)$. The expected result is $(2^{16}-21)$ and the result obtained will be $(2^{17}-21)$. To get correct result an inverse modular multiplicative module is implemented. The different design considerations that are satisfied by the normal HEROIC framework are also satisfied by this modified HEROIC framework. It provides a more efficient computation than the existing one.

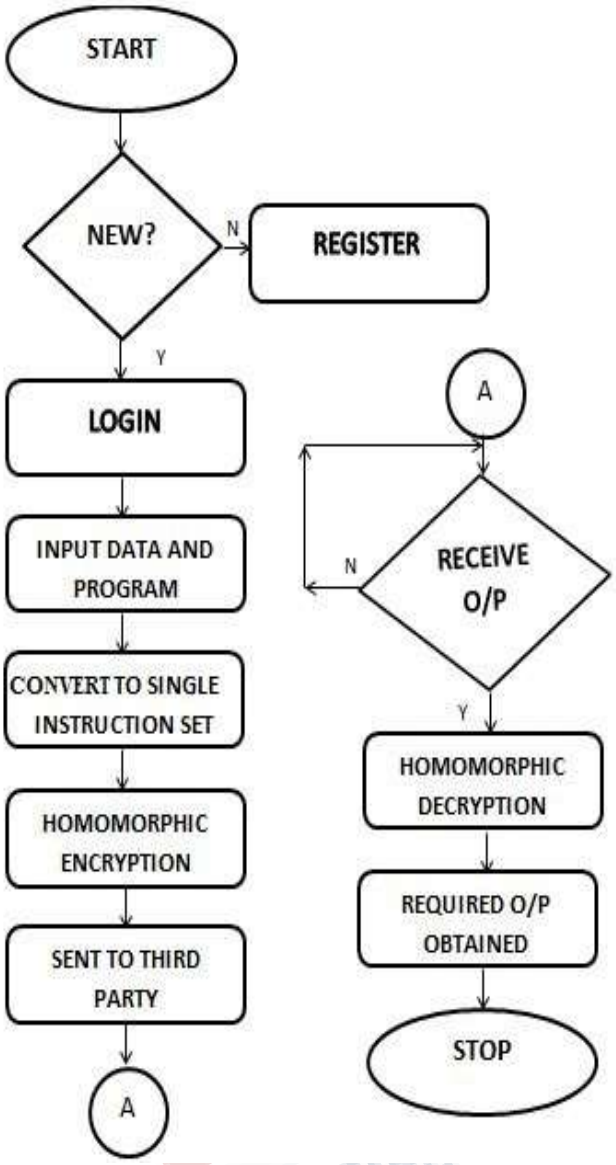

Fig. 6 Data owner

VI. EXPERIMENTAL RESULT

The comparison between existing HEROIC framework and the new fully homomorphic framework is done in this section.

Features	HEROIC framework	Fully homomorphic framework
Scheme	Paillier Scheme	Somewhat to fully homomorphism
Speed	Slow	Faster
Supporting computation	Only addition	Both addition and multiplication

Table 1: Showing comparison with existing and suggested systems.

A test for comparing the different execution time between existing and suggested systems can be found as follows. The fully homomorphic scheme is more efficient than the partially homomorphic scheme. The multiplication of two numbers based on the number of digits is taken as test program and the result is obtained as shown in graph below. The figure 7 shows the time efficiency of ne algorithm compared to the existing one.

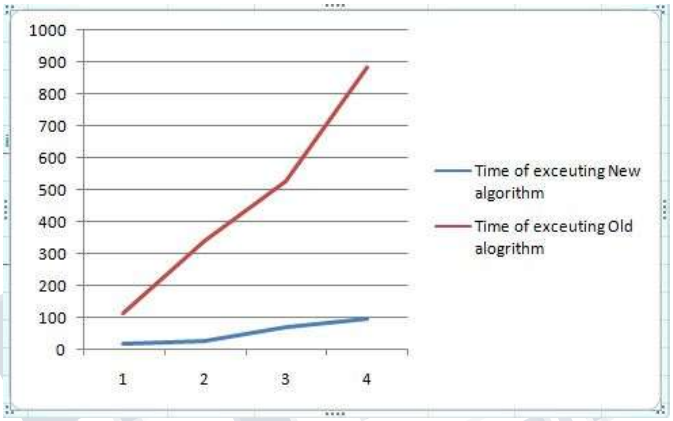

Fig.7 Graph showing execution time of different algorithms in y axis and data size in terms of number of digits in x axis.

VII. CONCLUSION

The data security and privacy concerns of data owner in outsourcing are reduced by this HEROIC framework with fully homomorphism. A somewhat homomorphic scheme is converted to the fully homomorphic scheme using Gentry's scheme. Fully homomorphism supporting both additive and multiplicative homomorphism improves the time efficiency of the existing system. The single instruction architecture used helps in understanding the existing encrypted instruction without decrypting them.

The future work can be concentrated on further improving the efficiency and security of the system. Modifications in architecture and exploiting parallelism can be other areas of attractions which could reduce overhead.

REFERENCES

[1] Wikipedia,https://en.wikipedia.org/wiki/Outsourcing computing

[2] C. Fontaine and F. Galand, "A survey of homomorphic encryption for nonspecialists," EURASIP Journal on Information Security, vol. 2007, no. 1, pp. 26-35, 2007.

[3] Nektarios Georgios Tsoutsos, Michail Maniatakos "HEROIC: Homomorphically EncRypted One

Instruction Computer" 2014

[4] O. Mazonka and A. Kolodin,"A simple multi-processor computer based on subleq," arXiv preprint arXiv:1106.2593, 2011

[5] A. Teller,"Turing completeness in the language of genetic programming with indexed memory," in Proc. 1st IEEE Conf. Evol. Comput., Orlando, FL, USA, 1994, pp. 136-141.

[6] Marten van Dijk, Craig Gentry, Shai Halevi, Vinod Vaikuntanathan, (2009), "Fully Homomorphic Encryption over the Integers", *IACR Cryptology ePrint Archive*

[7] Craig Gentry, (2009). *A fully homomorphic encryption scheme*. PhD thesis, Stanford University.

[8] C. W. Fletcher, M. van Dijk, and S. Devadas, "A secure processor architecture for encrypted computation on untrusted programs," in *Proc. ACM Workshop Scal. Trust. Comput.*, Raleigh, NC, USA, 2012, pp. 3–8.

[9] [Online]. Available: https://github.com/mikeivanov/paillier [48] G. E. Suh, D. Clarke, B. Gassend, M. van Dijk, and S.