A Stream Symmetric Homomorphic Encryption Scheme with Finite Automata

Alejandro Llamas[1] **and Raúl González**[1]
[1]Computer Sciences, CINVESTAV, Guadalajara, Jalisco, México

Abstract—*This paper presents a new homomorphic encryption scheme which uses sequential machines as a basis to produce a stream-symmetric encryption, which can be expanded to a homomorphic encryption with the help of the operation over automata called serial composition, allowing the application of any operation that can be represented by means of a finite automaton, on the set which contains the cipher-text. Unlike conventional cryptographic schemes this is not aimed at sharing information securely, so key distribution is not necessary and a symmetrical or asymmetrical cryptosystem makes no difference. With regard to safety, the intruder can only design cryptanalytic algorithms having as input a set of encrypted text and a composite automaton, constructed with the keys and an operation automaton. To illustrate the correct operation of the cryptographic scheme, we include the implementation of the greatest common divisor of two numbers.*

Keywords: Homomorphic Encryption, Finite Automata, Cloud Computing.

1. Introduction

Stream-symmetric cryptography comes perhaps from the old VigenÃÍre code, in which one character is exchanged for another according to a key or pad, many studies and developments have been made in this field, perhaps one of the most important is shown by Shannon in his article [1], where under certain conditions proved that the one time pad can be perfectly safe, being the basis of stream-symmetric encryption. One of these conditions is to generate a random string, such that the cipher-text is calculated using as parameters the plain-text and the random string, for this reason the designers of stream-symmetric cryptosystems, use sequential machines to generate pseudo-random strings called key-streams, in order that these chains may be repeated if it is introduced to the sequential machine, the initialization vector (IV) and the key. However we use sequential machines, for they have the characteristic of being invertible and not due to the ability to generate pseudo random repeatable chains. One of the first to talk about invertible circuits and their inverses was James L. Massey in [2].

Returning to symmetric encryption, an important feature is that its keys must be kept secret. If we want to send data securely, keys have to be transmitted through a secure channel, this is the problem of key distribution, solved by Rivest, Shamir and Adleman with RSA in 1978 [3] (note that, in the scheme that we show here is not necessary key distribution). Also Rivest et al. were the first to propose, in [4], a homomorphic encryption, a problem that many have tried to solve (e. g. [5], [6]), but for some time could only design a partial homomorphic encryption, until 2009 when C. Gentry showed, using lattice-based cryptography, the first fully homomorphic encryption scheme. His scheme supports evaluations of arbitrary depth circuits by effectively refreshing the cipher-text, thereby reducing its associated noise and hence solving the problem of indecipherable cipher-text. However, this scheme is still impractical. For this reason, we attack the open problem of designing a cryptographic scheme which has an algebraic homomorphic ring, i.e. the two basic operations (addition and multiplication) can be performed on the cipher-text, this calculation must be done in a practical time for the implementation of more complex programs that use these basic operations in the cipher-text.

The rest of the paper is organized as follows. In section II we review elements of automata theory that requires the cryptographic scheme. Section III shows the design of a stream-symmetric encryption with the automata defined in section II. In section IV we extend the encryption designed in section III, to a full homomorphic encryption and we show the operations that can be performed on the cipher-text. Section V shows the implementation of the greatest common divisor of two integers, that can be executed by a third party without the data or the process is known by him. Section VI provides a new safety factor (covert operations) for homomorphic encryption schemes and the analysis of a possible attack to the encryption scheme shown here. Finally, in section VII we speak of work performed, current and future on this research.

1.1 Working in a parallel world

It all starts with "Alex", a researcher who discovers how to get access to parallel worlds, unfortunately send a person by the wormhole was impossible, he can only send data into a sequence of bits, Alex sends a string of bits and then receives a different string of the same length, seems that some kind of transformation is performed on the chain but is unknown. Days later he discovers that each different world responds differently to the same strings, definitely something amazing was happening. Eventually Alex discovers a tool that helps to communicate with these strange worlds, something called invertible finite automata and inverse finite automata, encoder and decoder respectively, one pair for each different world. He then meets “Ragde" an entity that works in something like a computer in that world. That gives Alex an idea: “What if I send some data to be processed on his computer". Soon he realized that the capacity of Ragde’s computer is somewhat limited. Ragde can only perform operations that can be represented by finite automata, so for really significant computations Alex would have to intervene in some way, but even this can take a long time running on Alex’s old computer, so he needed someone with a lot of computing power and that person was “Bugsy" a rich guy who was known to sniff the data of people and make them known. Due to this Alex needed to think of a way for Bugsy doing the remaining process that Ragde’s computer could not perform, keeping in secret most of the information. His solution was to give Bugsy access to the wormhole and instructions on how use it.

Without the encoder and decoder Bugsy could never communicate with Ragde to find out what calculation is done. The tasks of Bugsy are: store data and feed the wormhole, in this way the data and the calculation remain unknown. Once that Bugsy has completed the sequence of instructions with the “data" that was initially sent by Alex (data that were transformed with the encoder used to communicate with Ragde), he sends the results to Alex. Finally, Alex gets the results generated with the decoder. Results that he wouldn’t be able to obtain from his computer or that would simply take a long time to be useful. Summarizing, Alex managed to create a tool in order to use the services of supercomputers (such as Ragde’s computer), without the data used and the operations performed are known.

2. Finite Automata

Definition 1. A *finite automaton* is a quintuple $M = (\Sigma, A, Q, \delta, \lambda)$, where Σ, A and Q are non empty finite sets, δ is a function from $Q \times \Sigma$ to Q and λ is a function from $Q \times \Sigma$ to A. Σ, A and Q are called the input alphabet, the output alphabet and the state set of M, respectively. δ and λ are called the next state function and the output function of M, respectively. We can extend the domain of the function δ to $Q \times \Sigma^*$ as follows:

$$\boldsymbol{\delta} : Q \times \Sigma^* \to Q$$

Let $q \in Q$, $w \in \Sigma^n$, $\nu \in \Sigma^{n-1}$, $a \in \Sigma$ and $w = \nu a$

$$\boldsymbol{\delta}(q, \varepsilon) := q$$
$$\boldsymbol{\delta}(q, w) := \delta(\boldsymbol{\delta}(q, \nu), a)$$

Similarly, we can extend the domain of the function λ

$$\boldsymbol{\lambda} : Q \times \Sigma^* \to Q$$

Let $q \in Q$, $w \in \Sigma^n$, $\nu \in \Sigma^{n-1}$, $a \in \Sigma$ and $w = \nu a$

$$\boldsymbol{\lambda}(q, \varepsilon) := q$$
$$\boldsymbol{\lambda}(q, w) := \boldsymbol{\lambda}(q, \nu)\lambda(\boldsymbol{\delta}(q, \nu), a).$$

This kind of automata is also known as “Mealy machine" or “finite automata with outputs" for this paper is just “finite automata".

2.1 Finite automata with memory defined by a function

Definition 2. Let $f_M : A^t \times \Sigma^{r+1} \to A$, be a function and x_0, $x = x_{-1}, \ldots, x_{-r}$, $x' = x_{-1}, \ldots, x_{-r+1} \in \Sigma$, y_0, $y = y_{-1}, \ldots, y_{-t}$, $y' = y_{-1}, \ldots, y_{-t+1} \in A$. The (r, t)-*order memory finite automaton* defined by f_M is formed as follows:

$$M = (\Sigma, A, A^t \times \Sigma^r, \delta, \lambda)$$
$$\lambda(\langle y, x\rangle, x_0) = y_0 = f_M(y, x_0, x)$$
$$\delta(\langle y, x\rangle, x_0) = \langle y_0, y', x_0, x'\rangle.$$

In the case of $t = 0$ then we have a r-order input memory finite automaton, if $r = 0$ then we have a t-order output memory finite automaton.

2.2 Linear finite automata

The study of linear automata has been carried out for a long time and many advances have been made in their understanding, the references [7], [8], [9], [10] contain more information on this and other types of finite automata. Here we show only the necessary concepts.

Definition 3. Let M be a finite automaton defined by the function f_M. If f_M is a linear transformation then M is a *linear finite automaton*.

Let us show the following as the canonical form of the function of any linear finite automaton with memory:

$$f_M(x_i, \ldots, x_{i-r}, y_{i-1}, \ldots, y_{i-t}) = y_i = \sum_{j=0}^{j \le r} A_j x_{i-j} + \sum_{h=1}^{h \le t} B_h y_{i-h} \qquad (1)$$

where the A_j are matrices of $m \times l$, the B_j are matrices of $m \times m$, the x_j are vectors of dimension l and the y_j are vectors of dimension m, all over a Galois field of q elements denoted by $GF(q)$. We define the next as the transformation matrix of the finite automaton M:

$$G = [A_0 A_1 \ldots A_r B_0 B_1 \ldots B_t] \qquad (2)$$

where the matrices are accommodated horizontally to form a matrix of $m \times [(l * r + 1) + (m * t)]$, we also define

$$E(i) = [x_i \ldots x_{i-r} y_{i-1} \ldots y_{i-t}]^T \qquad (3)$$

where vectors are vertically arranged to form a new vector of dimension $[(l*r)+(m*t)]$, with this we can define the function f_M as follows:

$$f_M(x_i,\ldots,x_{i-r},y_{i-1},\ldots,y_{i-t}) = G*E(i) = y_i. \quad (4)$$

2.3 Inverse of a linear finite automaton

Consequent to the study of linear automata, it was found the property of this to be invertible, this implies to calculate the input from the output and an initial state, there are many ways to invert a finite automaton, either from the definition function or the finite automaton itself, each of these techniques requires the automaton or definition function meets certain conditions, references [8], [10] and [2] show some types. In this paper we will show how to invert the linear automata that we have defined. We will focus on finding the inverse from the values of the matrix G. In the article [11] Shimon showed how to get the inverse of a finite invertible automaton through the states of the machine. We begin the process of invert the given automata, verifying if it is invertible.

Proposition Let M be a finite automata and G its transformation matrix. If A_0 has left inverse then M is invertible.

Definition 4. Let M be an invertible finite automaton defined by f_M. *The inverse automaton* of M is M' and is defined by:

$$f_{M'}(y_i,\ldots,y_{i-t},x_{i-1},\ldots,x_{i-r}) =$$
$$x_i = A_0^{-1}y_i - \sum_{h=1}^{h\leq t} A_0^{-1}B_h y_{i-h} - \sum_{j=1}^{j\leq r} A_0^{-1}A_j x_{i-j} \quad (5)$$

2.4 Serial composition of finite automata

Definition 5. Let M=$(\Sigma, A, Q_1, \delta_1, \lambda_1)$ and M'=$(A, \Gamma, Q_2, \delta_2, \lambda_2)$ be two finite automata. We define the *composite finite automaton* as follows:

$$C(M_1,M_2) = (\Sigma,\Gamma,Q_1\times Q_2,\delta,\lambda)$$
$$\delta(\langle q_1,q_2\rangle,x) = \langle \delta_1(q_1,x),\delta_2(q_2,\lambda_1(q_1,x))\rangle$$
$$\lambda(\langle q_1,q_2\rangle,x) = \lambda_2(q_2,\lambda_1(q_1,x))$$
$$q_1\in Q_1, q_2\in Q_2, x\in\Sigma.$$

This kind of composition produces a finite automaton that output, the result of introducing the exit of the first automaton into the second, however is only half the time it would take execute separately, as shown in the figure 1 where $X = x_0,\ldots,x_t$, $Y = y_0,\ldots,y_t$ and $Z = z_0,\ldots,z_t$, $t\in\mathbb{N}$.

2.5 Operations with finite automata

Our scheme is limited to those operations that we can represent by a finite automaton but even if is not as powerful as a Turing machine it capacity is large, indeed any operation that we can design with a linear sequential circuit can be represent by a finite automaton, Kohavi [8] shows a method to transform any finite automaton in a linear sequential circuit and biseversa. Now even if we have an infinite number of operations that can be represented with finite automata, we are only interested in two in order to declare our scheme fully homomorphic encryption, these are the addition and multiplication in binary. The addition is simple, the problem is that a multiplication of two undefined numbers can not be represented by an automaton, however we can limit the size of the input, as it does with digital circuits and generate finite automata that multiply n-bit strings.

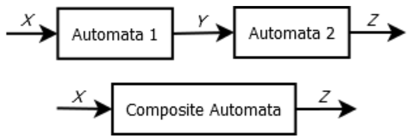


Fig. 1: Finite Automata Composition

Definition 6. We call an *operation automaton*, to any transformation that is represented by a finite automaton.

3. Stream-symmetric Encryption

The traditional stream-symmetric encryption is based on the idea of the one time pads, an encryption scheme that Shannon proved to be perfectly safe in [1], for this stream encryption designers, focus on algorithms that generate pseudo random strings called key-streams, these generators have the property that the same seed (the key and initialization vector IV) can repeat the key-stream, as with the one time pads, perform a simple XOR between the plain-text and the key-stream generates the cipher-text, for decoding is necessary to repeat the key-stream with the generator and operate along with the cipher-text to obtain the plain-text. The security of these systems is mainly based on the following:

- The difficulty of estimating the initial state of the sequential machine that generates the key-stream.
- The chain length of the repeating pattern e.g. the chain 132132 has a pattern of length 3, ideally it is a very large number.
- The distribution of generated key-stream, ideally the distribution is uniform.

For more information about the types of stream cipher see [12], also are many stream-symmetric ciphers available, in [13] shown the algorithms selected as the best.

3.1 Stream-symmetric encryption with finite automata

The Stream-symmetric encryption with finite automata requires, a key generator algorithm $KeyGen$, which in our case is a generator of invertible finite automata and their inverses, an encryption algorithm $Encrypt$ and one for decryption $Decrypt$.

$KeyGen$. To generate the keys we have to define the following: the value of q, i.e. the number of elements in the field, normally we handle bit sequences to represent the data so q is usually 2, the space vector of input elements l, these depend on the size of the character being used as input, normally 8 bit, the space vector of output elements m which must be equal or greater that l because if is less, there is a loss of information and retrieve the plain-text is impossible, the larger the value m, more secure is the encryption but beware because this value increases exponentially the number of states in the finite automaton. Finally select the values for the input memory r and output memory t, as before higher safer but more states in the automaton. Just to get an idea of how involved the selected memory are, the number of states that have the finite automata is defined by the following rule:

$$q^{(r*l)+(t+m)} \tag{6}$$

while the key space of our encryption scheme is greater than:

$$\prod_{j=1}^{j\leq r} q^{m*l} * \prod_{h=1}^{h\leq t} q^{m*m}. \tag{7}$$

Once you choose those parameters, select random numbers from the field $GF(q)$ to form the transformation matrix G, before continuing, we verified that the matrix A_0 from G is invertible, if it is then G is the transformation matrix for our invertible finite automaton, otherwise select new values for A_0 until find a matrix that has left inverse. Once we have selected the matrix G, form f_M as shown in equation (1) and using the technique of the section II, generate its corresponding finite automaton, then generate its inverse. All that remains is to select the corresponding initial states, to do this arbitrarily select a state $s_0 = \langle x_{i-1}, \ldots, x_{i-r}, y_{i-1}, \ldots, y_{i-t}\rangle$ from M as initial state, then the initial state of M' is $s_0^* = \langle y_{i-1}, \ldots, y_{i-t}, x_{i-1}, \ldots, x_{i-r}\rangle$. Summarizing M is our finite automaton for encryption with initial state s_0 and M' is our finite automaton for decryption with initial state s_0^* i.e. the private keys for the stream-symmetric scheme. Must take into account that the encryption key can be any invertible finite automaton, need not be linear, the linear automata is used to show a basic technique of inversion, we can use nonlinear finite automata or invertible finite automata with delay as shown Tao in [7].

$Encrypt$. Let $p_1, \ldots, p_n \in \Sigma^n$ be the plain-text and $M = (\Sigma, A, Q, \delta, \lambda)$ our invertible automaton used to encrypt, we calculate the cipher-text $c_1, \ldots, c_{2*n} \in A^{2*n}$ as follows:

$$\lambda(s_0, d_1p_1d_2p_2, \ldots, d_np_n) = c_1, \ldots, c_{2*n} \tag{8}$$

where $d_1, \ldots, d_n \in \Sigma^n$ is a string selected at random with a uniform distribution.

$Decrypt$. Let $c_1, \ldots, c_{2*n} \in A^{2*n}$ be the cipher-text and $M' = (A, \Sigma, Q', \delta', \lambda')$ our inverse automaton used to decrypt, we recover the plain-text by computing:

$$\lambda'(s_0^*, c_1, \ldots, c_{2*n}) = d_1p_1d_2p_2, \ldots, d_np_n \tag{9}$$

finally remove all $d_j, 1 \leq j \leq n$ and we get $p_1, \ldots, p_n$ i.e. the plain-text.

4. Homomorphic Encryption

This form of encryption allows specific types of calculations are conducted in the cipher-text and obtain an encrypted result, which corresponds to the encryption result of the operations performed on the plain-text. When we speak of homomorphic encryption, we can refer to partial homomorphic encryption or fully homomorphic encryption, the first has the characteristic that we can find over elements encrypted a binary operation resulting in an algebraic group, homomorphic to the group we have in the plain-text, these operations are called homomorphic operations e.g. RSA is multiplicative homomorphic, we can multiply two encrypted numbers and the result after decoding, is the multiplication of the two corresponding plain-texts.

If we have a fully homomorphic encryption, then we can find over elements encrypted, two binary operation leading to an algebraic ring, homomorphic to the ring we have in the plain-text, such operations as in the previous case are called homomorphic operations, in other words, let P be the set of plain-texts, $+$ and $\times$ binary operations over P and $+_c$ and $\times_c$ binary operation over the set of cipher-texts then for all $p_1, p_2 \in P$:

$$Encrypt(p_1 + p_2) = Encrypt(p_1) +_c Encrypt(p_2), \tag{10}$$
$$Encrypt(p_1 \times p_2) = Encrypt(p_1) \times_c Encrypt(p_2), \tag{11}$$

normally $+$ and $\times$ are modular addition and multiplication. In recent years this type of scheme has been largely worked, however most of the schemes are based on the same blueprint, using as basis an asymmetric cipher, then find an adaptation that allows get operations over the cipher-text that lead to a fully homomorphic scheme, some are RSA and “El Gamal", this and other examples are described by Fontaine in [14]. Gentry shows in his PhD thesis [15] the generation of a homomorphic scheme with latices, however its outline as many more, presents the noise problem i.e. after a finite number of operations, plain-text can not be obtained from the cipher-text, which Gentry resolves by encrypting the cipher-text again, however this makes the scheme impractical. Unlike regular schemes, ours is based on stream-symmetric encryption with sequential machines,

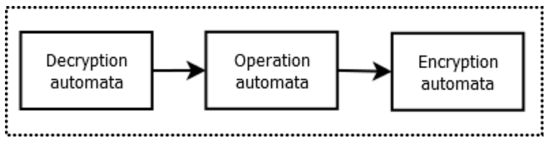


Fig. 2: $Transform$

which does not have the noise problem, as consequence operations are unlimited.

4.1 Stream-Symmetric-Homomorphic Encryption (SSHE)

A symmetric-homomorphic encryption scheme has four algorithms, $KeyGen$, $Encrypt$, $Decrypt$ and an additional algorithm that differs in the definition given by Gentry in [16], for our case the algorithm $Transform$, that takes as input the encryption key (the invertible finite automaton), the decryption key (the inverse finite automaton) and an operation $C \in \Omega$ = the set of finite automata, (i.e. the operation can be represented by a finite automaton), it outputs an automaton C^c that can process encrypted data. C^c take as input a set of cipher-texts $\Psi = \langle \psi_1, \dots, \psi_t \rangle$, it outputs a cipher-text ψ. in other words, Let $\psi_j = Encrypt(\pi_j), 1 \leq j \leq t$.

If $\psi = C^c(\Psi)$ Then $C(\pi_1, \dots, \pi_t) = Decrypt(\psi)$.

$Transform$. The other 3 algorithms needed for the homomorphic encryption are the same that we show in section III, we now describe the last. The algorithm $Transform$ has the function of composing three finite automata that receives as input, these are the decryption, the operation and the encryption automata, as shown in the figure 2. The result is an automaton that can perform the operation realized by the operation automaton, but receives as input a cipher-text and outputs an encryption of the result given by the operation automata.

4.2 SSHE operations

What operations can we apply on the cipher-text? As discussed in section II, any operation performed by a sequential linear circuit can be represented by a finite automaton and with the algorithm $Transform$ we can convert any operation automaton, to a SSHE operation, so unlike other schemes here we can design an infinite number of SSHE operations, even if we only need addition and multiplication to declare our scheme fully homomorphic, the point of this difference is that with an infinite number of possible homomorphic operations (SSHE operations), we introduce the new security parameter *covert operations*, if it is required not only the host unknown the data but also the operations performed in the cipher-text, more on this in the security section.

5. A worked example

The following example shows an implementation that calculates, the greatest common divisor of two encrypted numbers.

Definition 7. Let x, y, c and c' be positive integers, then the *greatest common divisor* is c, if $c|x$ and $c|y$ also their not exist $c' > c$ such that $c'|x$ and $c'|y$. Where $c|x$ means that c exactly divides x.

For this we use the technique of successive subtraction so we only need three basic operation automata, the “subtraction automaton", the “comparison automaton" and the “equal automaton". Once these have been designed, we use the algorithm $Transform$ to operate on the cipher-text and we get to host.

5.1 Basic operations

The following subsection shows the designed operation automata, noting that all consecutive automata have been composed, whereby the keys (here encryption and decryption automata) remain safe, even we can make that the host does not know which operation has being performed. Furthermore the operation automata are designed to ignore the values corresponding to the random input embedded in the plain-text. The following three automata form the basis of the implementation:

- SubC. This automaton subtracts two bit strings that have been encrypted.

- CMC. This lets us know, given two encrypted strings two things, if the output is 0, the first component is greater than or equal to the second, if the output is 1, the first component is less than the second. Basically do a subtraction and observe the value of the most significant bit, attempting to destroy the value of the decoded data.

- EqualC. Here we have a finite automaton that checks bitwise if the input data are equal, leaving the most significant bit in 1 if they are, zero otherwise, again all possible traces of data that could be used in cryptanalysis is destroyed.

5.2 How the GCD works

Figure 3 shows how the host will use the basic operation automata to calculate the greatest common divisor, as mentioned at the beginning of the section, SSHE operations hide the operation actually applied on the plain-text, however not all the process is performed with a finite automata, storage and decisions on program flow are still taken by the host freely, however the final goal remains unknown. The host has a general structure of the software as shown

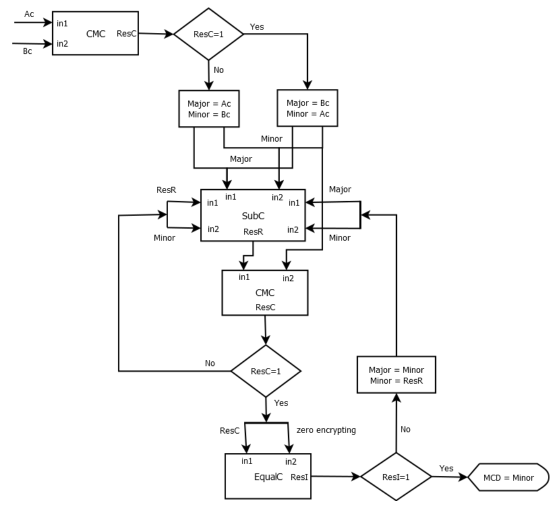


Fig. 3: Flow diagram of the GCD

in figure 3, for the response of a user "A" the host use the SSHE operations that corresponds to the user without modify the software structure. Each one of the SSHE operations, functions like a black box, responding after receiving inputs. Once built the software on the host's machine steps are:

1) The user A, which generated the SSHE operations with their encryption and decryption keys, select any two positive integers x, y, computes its binary representation, separately encrypts and sends to the host.

2) The host introduces the two encrypted data into the software and using the SSHE operations of the user A, calculates the answer and sends it to the user.

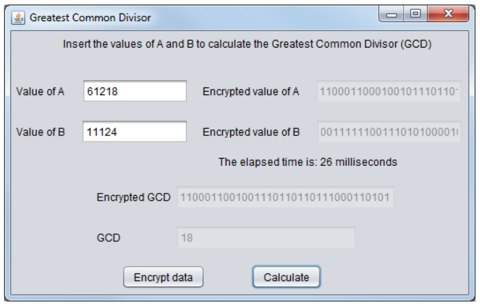
3) Finally the user A, using its decryption key, calculates the plain-text corresponding to the greatest common divisor of x and y, not forgetting eliminate random elements embedded in the string at the beginning.

5.3 How the software works

When we implement the software, the host was ignore, however the calculation is performed as described in the previous section, the software takes as input two integers greater than 0 and when we press "Encrypt data", using an invertible finite automaton, encrypts the data and once we press "calculate" takes the encrypted data and calculates the cipher-text that correspond to the greatest common divisor, finally decrypts the value to verify that the algorithm works. One program screen shown in figure 4.

6. Security

The safety analysis of an encryption scheme is very complicated, because the actions that the intruder can perform are vast, which is why to know how safe it is an encryption


Greatest Common Divisor

Insert the values of A and B to calculate the Greatest Common Divisor (GCD)

Value of A 61218 Encrypted value of A 1100011000100101110110

Value of B 11124 Encrypted value of B 0011111100111010100001

The elapsed time is: 26 milliseconds

Encrypted GCD 1100011001001110110110111000110101

GCD 18

Fig. 4: GCD software view

scheme, are designed some possible attacks and is observed the complexity of implement them.

6.1 A new security approach (covert operations)

The idea can be implemented in two ways, one where software is designed by the host and another where the user designs, being designed by the host, security level covert operations, can not be reached. In case that the software is designed for the user, given that the SSHE operations are the operation automata compounds with encryption and decryption keys, it can achieve new security approach where not only the host unknown the data but also the calculation being made by the SSHE operations, to know the operations performed on the plain-text, the intruder must decompose the basic operation in its components automata and do this is considered a complicated task, as shown in [17] and [8], besides that the resulting automata from the decomposition are not unique, there is a very large number of automatons which can be the answer of the decomposition.

6.2 Algebraic attack

In most of the attacks, we has to suppose that parts of the data that should be kept in secret have been discovered, in the following we will analyze a case where the functions defining the encryption and decryption automata have been discovered by the intruder and also is able to obtain the corresponding cipher-text from a selected plain-text, the attack is carried out as follows: let Σ be a l-dimensional vector space over $GF(q)$, A be a m-dimensional vector space over $GF(q)$, $x_j \in \Sigma, y_j \in A$ where $j \in \mathbb{Z}$ and the equation (1) is the definition function of the encryption automata.

Suppose we know A_j and B_h, $0 \leq j \leq r, 1 \leq h \leq t$. The goal now is to get the initial state formed by $x_-1, \ldots, x_{-r}, y_{-1}, \ldots, y_{-t}$. To achieve this we need to create a system of $(r+t)$ equations with $r+t$ unknowns, to generate them do the following, assume that P_i^x corresponds to a known value of x_i and C_i^y corresponds to a known value of y_i. Introduced a sequence of length r to observe the following behavior of the function:

$$A_0 P_0^x + A_1 x_{-1} + \ldots + A_r x_{-r} +$$
$$B_1 y_{-1} + B_2 y_{-2} + \ldots + B_t y_{-t} = C_0^y$$

$$A_0 P_1^x + A_1 P_0^x + A_2 x_{-1} + \ldots + A_r x_{1-r} +$$
$$B_1 C_0^y + B_2 y_{-1} + \ldots + B_t y_{1-t} = C_1^y \tag{12}$$
$$\vdots$$
$$A_0 P_{r-1}^x + A_1 P_{r-2}^x + \ldots + A_{r-1} P_0^x + A_r x_{-1} + B_1 C_{r-2}^y$$
$$+ \ldots + B_{r-1} C_0^y + \ldots + B_t (y_{(r-1)-t} || C_{(r-1)-t}^y) = C_{r-1}^y$$

where $(y_{(r-1)-t} || C_{(r-1)-t}^y)$ indicates that there may be a known variable (C_i^y) or unknown (y_i), to find out we have to verify, if $(r-1)-t$ is negative then is $y_{(r-1)-t}$, if it is zero or positive then is $C_{(r-1)-t}^y$. After we introduce a sequence of length t to observe the following behavior of the function:

$$A_0 P_0^x + A_1 x_{-1} + \ldots + A_r x_{-r} +$$
$$B_1 y_{-1} + B_2 y_{-2} + \ldots + B_t y_{-t} = C_0^y$$

$$A_0 P_1^x + A_1 P_0^x + A_2 x_{-1} + \ldots + A_r x_{1-r} +$$
$$B_1 C_0^y + B_2 y_{-1} + \ldots + B_t y_{1-t} = C_1^y \tag{13}$$
$$\vdots$$
$$A_0 P_{t-1}^x + \ldots + A_{t-1} P_0^x + \ldots + A_r (x_{(t-1)-r} || P_{(t-1)-r})$$
$$+ B_1 C_{t-2}^y + \ldots + B_{t-1} C_0^y + B_t y_{-1} = C_{t-1}^y$$

where $(x_{(t-1)-r} || P_{(t-1)-r})$ indicates that there may be a known variable (P_i^x) or unknown (x_i), to find out we have to verify, if $(t-1)-r$ is negative then is $x_{(t-1)-r}$, if it is zero or positive then is $P_{(t-1)-r}^x$. Finally we solve the system of equations for $x_-1, \ldots, x_{-r}, y_{-1}, \ldots, y_{-t}$. Note that Solving systems of multivariate equations is known as a complex task. The complexity of this attack is defined by the memory of the automata.

7. Conclusion

In this paper, we create a new fully homomorphic encryption scheme, using as a base stream-symmetric encryption and finite automata theory. We show that our cryptographic scheme has the following characteristics:

- Is defined by the following algorithms: $KeyGen$, $Encrypt$, $Decrypt$ and $Transform$.
- By using finite automata to encrypt and decrypt, bitwise or character by character, we have a stream cipher, so is faster that any other know homomorphic encryption.
- We can define on the cipher-text, any operation that can be represented with a finite automaton, not only addition and multiplication.
- The ability to perform an infinite number of consecutive operations on the cipher-text, without losing the connection with the plain-text that corresponds.
- Keys are private, so we do not have the problem of key distribution.
- The security of the cryptosystem is based on the difficulty of decomposing a finite automaton.

Finally, we show the performance of the Stream-symmetric-homomorphic Encryption scheme, generating a software that calculates the greatest common divisor of two encrypted numbers. Currently we are working on techniques to reduce the size of the keys and consequently of SSHE operations, If this line of research is successful, it could reduce the size of SSHE operations, to a few kilobytes, remember, that by the time these operations are automata and for define an automaton $M = (\Sigma, A, Q, \delta, \lambda)$, we can use a matrix of size $Q \times \Sigma$, having as elements, pairs of the form: (Q, A).

References

[1] C. Shannon, "Communication theory of secrecy systems," *Bell System Technical Journal, Vol 28, pp. 656âĂŞ715*, Oktober 1949.

[2] J. L. Massey and M. K. Sain, "Inverses of linear sequential circuits," *IEEE Trans. Comput.*, vol. 17, no. 4, pp. 330–337, Apr. 1968.

[3] R. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, pp. 120–126, 1978.

[4] R. L. Rivest, L. Adleman, and M. L. Dertouzos, "On data banks and privacy homomorphisms," *Foundations of Secure Computation, Academia Press*, pp. 169–179, 1978.

[5] R. Cramer, R. Gennaro, and B. Schoenmakers, "A secure and optimally efficient multi-authority election scheme." Springer-Verlag, 1997, pp. 103–118.

[6] M. Kuribayashi and H. Tanaka, "Fingerprinting protocol for images based on additive homomorphic property," *IEEE Transactions on Image Processing*, vol. 14, p. 2005.

[7] R. Tao, *Finite Automata and Application to Cryptography*. Springer Publishing Company, Incorporated, 2009.

[8] Z. Kohavi, *Switching and Finite Automata Theory: Computer Science Series*, 2nd ed., R. W. Hamming and E. A. Feigenbaum, Eds. McGraw-Hill Higher Education, 1990.

[9] A. Gill and C. U. B. E. R. LAB., *Analysis of Linear Sequential Circuits by Confluence Sets*. Defense Technical Information Center, 1964.

[10] I. Amorim, A. Machiavelo, and R. Reis, *On Linear Finite Automata and Cryptography*, 2011.

[11] S. Even, "On information lossless automata of finite order," *Electronic Computers, IEEE Transactions on*, vol. EC-14, no. 4, pp. 561–569, Aug.

[12] A. Maximov, "Some words on cryptanalysis of stream ciphers," Ph.D. dissertation, Lund University, 2006.

[13] S. Babbage, C. De Canniere, A. Canteaut, C. Cid, H. Gilbert, T. Johansson, M. Parker, B. Preneel, V. Rijmen, and M. Robshaw, *The eSTREAM Portfolio*, 2008.

[14] C. Fontaine and F. Galand, "A survey of homomorphic encryption for nonspecialists," *EURASIP J. Inf. Secur.*, vol. 2007, pp. 15:1–15:15, Jan. 2007.

[15] C. Gentry, "A fully homomorphic encryption scheme," Ph.D. dissertation, Stanford University, 2009.

[16] Gentry, "Fully homomorphic encryption using ideal lattices," in *Proceedings of the 41st annual ACM symposium on Theory of computing*, ser. STOC '09. New York, NY, USA: ACM, 2009, pp. 169–178.

[17] C. Halatsis, M. Sigala, and G. Philokyprou, "Polylinear decomposition of synchronous sequential machines," *IEEE Transactions on Computers*, vol. 27, no. 12, pp. 1144–1152, 1978.