Programming Random Change of Variables for Homomorphic Encryption

Juan P. Ramirez
Operaciones Digitales y Procesamiento Integral de Datos Encriptados, SAS
Jalisco, México
jramirez@binaryprojx.com;
Personal Page: www.binaryprojx.com

***Abstract*—A Homomorphic Encryption scheme, based on a change of variables method, is outlined. The data is encrypted through a random change of variables, and then operations are performed on the ciphertext, that simultaneously process and decrypt the result. This improves traditional Homomorphic Encryption in some instances, by merging two steps into one, ensuring that the data can only be used for the intended purpose. Noise and precision are manageable, without huge efficiency trade-offs as in the case of bootstrapping techniques. A two-party scheme between a Client a Bank has been developed as a Proof-of-Concept. The Bank does not have access to inputs representing Client data, but is able to processes encrypted vectors. The method is flexible and can be adapted for a different number of parties, permission management, security levels, efficiency requirements, etc.**

***Index Terms*—Applied Mathematics, Homomorphic Encryption, Zero Knowledge Proof, Data Security, Machine Learning**

I. Introduction

Cryptography studies methods of encrypting messages that are to be shared with specific second-parties, and no one else. However, the evolving technological landscape calls for more complete encryption methods that are compatible with data processing. Traditionally, if one must process encrypted data, first the data is decrypted, and then the data is processed. This exposes the data to the second, and potentially third parties, in many instances. Homomorphic Encryption [1], [2] addresses this issue by changing the order of these two steps by first processing the data, while still encrypted, and then decrypting the result to plaintext. Machine Learning and AI applications that require sharing mass amounts of sensitive data, smart grids, large networks, traffic control, electronic voting, energy management, among many others, can be implemented if certain privacy issues are solved [3-7].

Let E a function that encrypts natural numbers. Suppose we encrypt two numbers x, y to obtain two new numbers Ex, Ey. We won't worry right now about which space the ciphertext is defined in, and suppose we have an operation $\oplus$ defined in that space. The operation of these gives $Ex \oplus Ey$. For most encryption functions, this is not equal to $E(x \oplus y)$. Therefore, decrypting $Ex \oplus Ey$ does not yield the expected result $x \oplus y$. If there exists a computable function D such that $x + y = D(Ex \oplus Ey)$, for any choice of $x, y \in \mathbb{N}$, we have a Partially Homomorphic Encryption Scheme for Addition (+). If the same property is also satisfied for multiplication $(\cdot)$, then it is a Fully Homomorphic Encryption Scheme. The first difficulty in HE is that an operation and an encryption function are almost never homomorphic, $E(x+y) \neq Ex \oplus Ey$. Mathematical homomorphisms that can be used for HE are not practical solutions because of the complicated structures involved. Consequently, precision and noise are not easily mitigated [8], [9] making the algorithms unpractical or energy inefficient [10], [11], [12] in many scenarios. Here we explore a Keyless Decryption method that merges processing and decryption into a single step and avoiding the necessity for Bootstrapping techniques. In this case, the processing function is also the decryption key. The first party encrypts the inputs by applying a random change of variable from a library of encryption functions. The second party possesses a corresponding library of process/decryption functions and determines which of these functions will process/decrypt the result. Applying any other function of the library yields a meaningless answer. A large class of encryption functions can be defined for a large class of processing functions, including addition and multiplication, and with some advantages over existing solutions. A specific mathematical model has been chosen to process a Credit Score for a personal one-year loan, as a Proof of Concept herein described.

If we wish to represent encryption of a message in a conceptual diagram, the mathematical expression for the commutative diagram is given by an encryption function $Em \in M'$ that encrypts the original message, and a decryption function $D(Em) \in M$ that decrypts the encrypted message.

$$\begin{tikzcd} M \arrow[d, shift right, "E"'] \\ M' \arrow[u, shift right, "D"'] \end{tikzcd}$$

Similarly, the commutative diagram for representing a HE Scheme, in its simplest conceptual form, is given by the equation $P = D \circ P^* \circ E$ shown below, where $P : A \to B$ is the process that we wish applied to the data plaintext and P^* is the modified process to be applied to the encrypted data.

$$\begin{tikzcd} A \arrow[r, "P"] \arrow[d, "E"'] & B \\ A \arrow[r, "P^*"] & B \arrow[u, "D"'] \end{tikzcd} \qquad (1)$$

This diagram illustrates the basic relation for Encrypting the domain of the function and decrypting the result. Applying the process P to the plaintext is equivalent to encrypting, applying P^* and then decrypting.

We present a Homomorphic Encryption scheme based on the concept of change of variable. Suppose a calculation $P : \mathbb{R}^n \to \mathbb{R}$ has to be applied to an ordered n-tuple of variables $\mathbf{x} = (x_i)_{i=1}^n = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$. We encrypt these n variables by applying a computable function $E : \mathbb{R}^n \to \mathbb{R}^m$ to the ordered n-tuple. An *encryption function of P* is a computable function $E : \mathbb{R}^n \to \mathbb{R}^m$ if there exists a computable function $P^* : \mathbb{R}^m \to \mathbb{R}$ such that $P\mathbf{x} = (P^* \circ E)\mathbf{x}$. In this case we say P, P^* are an ordered pair of homomorphic functions [13] with respect to encryption function E. Let $k \in \mathbb{N}$ and let P be a fixed processing function and suppose we have an indexed set of encryption functions $E_j : \mathbb{R}^n \to \mathbb{R}^{m_j}$ and a corresponding set of functions $P_j^* : \mathbb{R}^{m_j} \to \mathbb{R}$, such that P, P_j^* are a homomorphic pair under its corresponding encryption function E_j, for every $j \leq k$. That is to say, $P = P_j^* \circ E_j$ for every $j \leq k$. We have a set of k-many encryption functions, $\{E_j\}_{j=1}^k$, for a single processing function P. Every time we run process P on a vector $\mathbf{x}$, we first choose an encryption function $E \in \{E_j\}_{j=1}^k$. The original vector $\mathbf{x}$ is transformed using E to obtain an encrypted vector $E\mathbf{x}$. If we select a different encryption function of $\{E_j\}_j$, we could get different data types and number of coordinates because every E_j has its own dimension of codomain. If $E : \mathbb{R}^n \to \mathbb{R}^{m_1}$ and $F : \mathbb{R}^n \to \mathbb{R}^{m_2}$ are elements of $\{E_j\}_j$, then m_1 and m_2 can be distinct natural numbers. There are k-many different methods for encrypting the variables of our processing function $P = P_j^* \circ E_j$. When an encryption function E_j is used, its corresponding processing function P_j^* is applied to the encrypted variables.

II. Encryption by Change of Variable

Suppose $P : \mathbb{R}^n \to \mathbb{R}$ is a function of n independent variables, and one party wishes for a second party to compute $P(x_1, x_2, \ldots, x_n)$ without having the capability of knowing the input vector $(x_1, x_2, \ldots, x_n)$. Let $E : \mathbb{R}^n \to \mathbb{R}^n$, defined by functions $e_1, e_2, \ldots, e_n : \mathbb{R}^n \to \mathbb{R}$ such that

$$P \circ E = D^{-1} \circ P, \qquad (2)$$

for some invertible function $D^{-1} : \mathbb{R} \to \mathbb{R}$.

$$\begin{tikzcd} \mathbb{R}^n \arrow[r, "P"] \arrow[d, "E"'] & \mathbb{R} \arrow[d, "D^{-1}"] \\ \mathbb{R}^n \arrow[r, "P"] & \mathbb{R} \end{tikzcd}$$

Suppose the change of variables E expresses $P \circ E$ in terms of an invertible function D^{-1}, of P. If the inverse function D is computable then we have encryption and decryption functions E, D, respectively.

For example, let $P : \mathbb{R}^2 \to \mathbb{R}$ be the function $P(x, y) = x + y^2$, and let $e_1(x) = a^2x^2 + 2a^2xy^2$ and $e_2(y) = ay^2$, for some parameter $a \in \mathbb{R}$. Then

$$(P \circ E)(x, y) = e_1(x) + (e_2(y))^2 = a^2(x + y^2)^2 = a^2(P(x, y))^2.$$

We have found $P \circ E$ to be expressed in terms of P, namely $(P \circ E)(x, y) = a^2(P(x, y))^2$. In this case, the function D^{-1} is given by $D^{-1}(x) = a^2x^2$. Therefore, the decryption function is given by $D(x) = \frac{1}{a}\sqrt{x}$. This implies $P = D \circ P \circ E = \frac{1}{a}(P \circ E)^{1/2}$. Let us understand what this relation means, and what it permits. Suppose we wish for a second party to compute $P(x_0, y_0) = x_0 + y_0^2$, for constants $x_0, y_0 \in \mathbb{R}$, but we do not wish to share the plaintext vector (x_0, y_0) with the second party. First we encrypt the inputs, using the function E described above, to obtain $E(x_0, y_0) = (e_1(x_0, y_0), e_2(x_0, y_0))$, and send this encrypted vector to the second party. The second party will then take this encrypted vector and apply function $P' = D \circ P$ to obtain the desired result. We collapse (1) into a three diagram. If we consider the composition $P' = D \circ P$ as a single function, the resulting relation is $P = P' \circ E$.

$$\begin{tikzcd} & \mathbb{R}^n \arrow[dr, "P'"] & \\ \mathbb{R}^n \arrow[ur, "E"] \arrow[rr, "P"] & & \mathbb{R} \end{tikzcd} \qquad (3)$$

Now we must ask: Can the second party decrypt the encrypted vector to find the original inputs x_0, y_0 with the information available? The information available is 1) Processing function P, 2) Encryption function E and Encrypted Variables e_1, e_2, and 3) Final result p_0. From 1) and 3) we have equation $x_0 + y_0^2 = p_0$, where $p_0 = (P' \circ E)(x_0, y_0)$. Two more equations are given by $a^2x_0^2 + 2a^2x_0y_0^2 = e_1$ and

$ay_0^2 = e_2$. A solution for a, x_0, y_0 must be found for the system of equations

$$\begin{aligned} x_0 + y_0^2 &= p_0 \\ a^2x_0^2 + 2a^2x_0y_0^2 &= e_1 \\ ay_0^2 &= e_2 \end{aligned}$$

Of course, if we find a solution for a, we can then calculate x, y. Let us also specify the bit size of the values in question. First, we have 64-bit inputs x_0, y_0. We also have a 128-bit encryption key, a. Of course, this key is unknown to the second party otherwise it could easily decrypt e_1 and e_2 to find the original inputs x_0, y_0. That is to say, for purposes of finding solutions to this system of equations the key a is an unknown integer of 128 bits. Now its clear that finding the original inputs is equivalent to finding the key a. If the second party knows the inner workings of the scheme, they would know to start by finding the factors of the 256-bit number e_2.

Now that we understand where vulnerabilities can arise, we can ask a few questions to maximize security. Let us start by asking if their may exist a type of integer that is optimal for the security parameter? For example, should we choose a to be a prime number every time, then finding a is trivial. To find a, you simply have to find the factors of $e_2 = ay^2$ that are smaller than 64-bits. Once we have the 128-bit key a, we can calculate y and then x. We conclude that the key should not be chosen to be a prime number. The more prime factors a has the better, because a can be any 128-bit number formed by multiplying a combination of the prime factors of e_2.

For every encryption function E, of some processing function P, we have different security parameters. In each case, the keys shall have different form and requirements to maximize security. Let us define an example with more general conditions. Let $P(x, y) = x + y$, and $E : \mathbb{R}^2 \to \mathbb{R}^2$ defined by coordinate functions $e_1(x) = e^{ax+b}$ and $e_2(y) = e^{ay+b}$. Multiplying gives $e_1(x)e_2(y) = e^{ax+b}e^{ay+b} = e^{a(x+y)+2b}$. This implies $P = P^* \circ E$, where $P^*(x, y) = \frac{1}{a}(\log x + \log y - 2b)$ and a, b are the security parameters. There is a slight difference from the last example, because $P^*(x, y) = \frac{1}{a}(\log x + \log y - \log 2b)$ is not of the form $D \circ P$. In the last example $P' = D \circ P$, but now the function $P^*(x, y)$ is not expressed in terms of $x + y$.

We give a definition of encryption and decryption functions that supersedes the previous ones. If, more generally, given an encryption function $E : \mathbb{R}^n \to \mathbb{R}^m$ we can find a computable function $P^* : \mathbb{R}^m \to \mathbb{R}$ such that $P = P^* \circ E$, then we have an encryption scheme for process P.

$$\begin{tikzcd} & \mathbb{R}^m \arrow[dr, "P^*"] & \\ \mathbb{R}^n \arrow[ur, "E"] \arrow[rr, "P"] & & \mathbb{R} \end{tikzcd}$$

Notice that there is no straightforward method for finding the most secure and efficient encryption and processing functions E, P^*, for a given process P. Instead of using one-way functions to encrypt the inputs, we use the fact that certain encryption functions are safe (the keys, and therefore the plaintext inputs, cannot easily be found). Furthermore, the process/decrypt function P^* remains unknown to unwanted parties. And, even if the encryption and processing functions E, P^* where to be known, finding the keys can be a very difficult problem. One way to ensure the encryption function remains unknown is to have a large library of encryption functions for each processing function, and then randomly select an encryption function each time. This solution is discussed in Sections 6 and 7. The general method presented here allows for a FHE scheme that can be adapted for different number of parties, permission configurations and security needs.

In Section 5, we outline a simple two-party scheme to simulate a personal loan Credit Score where the first party is a Client and the second party is a Bank. The Client will encrypt the inputs, and share them with the Bank. The Bank receives these encrypted values, processes the encrypted data and sends the result back to the Client.

In the general case defined above, we have provided an example where there is no key to decrypt the final result. The function P^* and its parameters are necessary and sufficient to process and decrypt the data. The inputs of function P^* are encrypted but the output is a plaintext value. After processing the data we do not have to decrypt the results because the data is already decrypted. The processing function P^* serves as decryption key also.

III. Keyless Decryption

We will illustrate a special case of relation (3). Let $P(x, y, z) = \frac{x^2+y^2}{z}$. If we make the change of variables $e_1(x) = ax$, $e_2(y) = ay$, $e_3(z) = a^2z$, then we have $P = P \circ E$. Again, $P \circ E$ is a function of P. In this case $D^{-1} = I$ is the identity function. This example of $P(x, y, z) = \frac{(ax)^2+(ay)^2}{a^2z}$ cancels nicely with the encryption function, so that $D = I$ is the identity function. When this happens we say this is a Keyless Decryption Scheme because $P' = I \circ P = P$. In this case, the function that has to be applied to the encrypted data is the same function that would be applied to the original plaintext. No new function has to be defined, $P' = P$.

$$\begin{tikzcd} & \mathbb{R}^n \arrow[dr, "P"] & \\ \mathbb{R}^n \arrow[ur, "E"] \arrow[rr, "P"] & & \mathbb{R} \end{tikzcd}$$

This example of Keyless Decryption is not semantically secure for several reasons, but non-trivial and semantically secure examples of Keyless Decryption can be found for certain functions. In this example, if any of x, y, z is zero, then its corresponding Encrypted Variable will be zero. Secondly, the bit-length of the Variables is easily deduced from the Encrypted Variables, which makes it useless for almost any application. Ultimately, the encryption function of this example does not work for the following reason. It is easy to find Key a, to calculate the Inputs. The Key is the number that divides e_1, e_2 once and divides e_3 two times.

There are different techniques for defining secure encryption functions and keys without recurring to excessive bit-lengths, because there are encryption functions whose equations do not determine the keys easily. The solution presented in Sections 6 and 7 uses strong and "semi-strong" encryption functions to create a large library of encryption functions, and randomly chooses a different one each time, adding an extra layer of security. Given a large library of encryption functions $\{E_j\}_j$, for a function P, the second party is provided with a library of processing functions $\{P_j^*\}_j$, such that each encryption function E_j is associated to a unique processing function P_j^*.

IV. Program Schematics

This section defines the schematics of the proposed two-party POC. Variants adapting to different needs can similarly be worked out. Privately communicating sensitive financial information between the Bank and the Client for numeric evaluation will require an infrastructure that will be briefly discussed without technical detail. Full-scale multi-purpose versions of this HE scheme are being developed for licensing.

-6 (Six) Data Elements. We begin by describing the data elements. First we have 6 (six) plaintext *i)* INPUTS representing the Client's data, in the form of 32-bit unsigned integers. The Inputs are Net Income, Expenses, Capital, Assets, Current Debt and Loan Amount. The Inputs will be processed, along with *ii)* PARAMETERS, at the Client's terminal to reduce the six Inputs to a total of 4 (four) numeric values we will call *iii)* VARIABLES. The four Variables are Net Income, Total Expenses, Worth and Total Debt. The Variables are then encrypted using the Parameters and *iv)* KEYS. The number of Keys will vary depending on the encryption function. Encrypting the Variables produces new data types, *v)* ENCRYPTED VARIABLES, of a chosen length. The encrypted values are sent to the Bank for processing. The Bank processes the Encrypted Variables together with another set of Parameters that fine tune and adjust the mathematical model for different cases. For example, Parameters can adjust interest rate, loan period, weights on debt or on different classes of assets, etc. The sixth data type is the *vi)* FINAL OUTPUT which is the final Credit Score resulting from the Bank processing the Encrypted Variables.

-5 (Five) Modules. The communications and internal operations for this scheme will be divided into (5) five MODULES, (3) three of which are for internal processes, and (2) two for communications between the Client and the Bank. Of the three processing modules, the first two are run by Client, while the third module is run by the Bank.

MODULE I: INTERFACE for INPUT/OUTPUT at CLIENT'S TERMINAL. Client Interface for Inputs. The Bank does not have access to Inputs nor keys. The Bank will only have access to Encrypted Variables.

- Input: Six (6) 32-bit inputs as financial and loan data
- Output: Four (4) 32-bit plaintext Variables

MODULE II for ENCRYPTING VARIABLES at CLIENT'S TERMINAL. This module receives the Variables from Module I, and encrypts them. Encryption is divided into two functions:

1) Generate encryption Keys.
2) Encrypt Variables using Keys.

MODULE CI for COMMUNICATING ENCRYPTED VARIABLES. This module communicates the Encrypted Variables from the Client to the Bank. The Client can verify the Payload consists of the Encrypted Variables.
SEND: ENCRYPTED VARIABLES

from: Client Module II
to: Bank Module III

MODULE III for PROCESSING ENCRYPTED VARIABLES at Bank. This module will be run by the Bank, to calculate the final Credit Score using Encrypted Variables and Parameters. This module must support arbitrarily long arithmetic. Although it does not have an explicit decryption step, the Output of this module is the decrypted Credit Score.

- Input: Encrypted Variables (Output of Mod. II)
- Output: Final Credit Score in plaintext (decrypted)

MODULE CII for COMMUNICATING FINAL CREDIT SCORE. This module is used for communicating the final Credit Score to the Client. The Bank sends the Final Output from Module III, to Module I.
SEND: FINAL OUTPUT (Credit Score)

from: Bank Module III
to: Client Module I

-8 (Eight) Steps. Eight steps are sufficient to find the Final Output in the form of a plaintext floating point number between 0 and 10.

STEP 1: INPUT DATA at MODULE I. Inputs are 32-bit unsigned integers representing Net Income, Expenses, Capital, Assets, Current Debt and Loan Amount.

STEP 2: CALCULATE VARIABLES at MODULE I. Use Inputs to find Variables NI, TE, W and TD.

A. Net Income $\rightarrow$ Net Income (NI)
B. Loan Amount $\rightarrow$ Newly Incurred Debt (NID) & Service to Debt (SD)
C. Current Debt & Newly Incurred Debt $\rightarrow$ Total Debt (TD)
D. Expenses & Service to Debt $\rightarrow$ Total Expenses (TE)
E. Capital & Assets $\rightarrow$ Worth (W)

Variables are a simple transformation of Inputs. This step outputs four Variables, which are Net Income, Total Expenses, Worth and Total Debt.

STEP 3: SEND VARIABLES from MODULE I to MODULE II.

STEP 4: GENERATE KEYS at MODULE II.

STEP 5: ENCRYPT VARIABLES at MODULE II. Variables are encrypted by Client using Module II.

STEP 6: SEND ENCRYPTED VARIABLES from MODULE II to MODULE III using MODULE CI.

STEP 7: PROCESS ENCRYPTED VARIABLES at MODULE III. The Bank will receive Encrypted Variables, and operate on these.

STEP 8: SEND DECRYPTED OUTPUT from MODULE III to MODULE I using MODULE CII. The reader is invited to try a variety of combinations of inputs to verify validity of the mathematical model for predicting financial reliability. Finding a deterministic algorithm, that decrypts the Inputs, should prove impractical.

V. Mathematical Model

Encryption and Processing functions are defined that yield a decrypted plaintext Credit Score, to be shared with the Client. The function we wish to calculate is normalized. If the income to expense ratio and the worth to debt ratio go to infinity, the Credit Score is 10. If, on the other hand, the expenses and debt grow proportionally with respect to income and worth, the Credit Score is 0. The minimum approval score is suggested at 5 points, given the current settings and weights of the parameters, but these can be modified for different cases. A reasonably calibrated model for loan approvals based on a client's financial data (income, capital, debt, expenses, etc.) is proposed. Parameters can be modified to adapt the Credit Score model to different scenarios (different types of loans such as micro loan, business loan, loan period, interest, etc.). The proposed processing function P for this model is

$$P(NI,TE,W,TD) = \frac{A \cdot NI}{\sqrt{NI^2+\alpha TE^2}} + \frac{B \cdot W}{\sqrt{W^2+\beta TD^2}},$$

where A, α, B, β are parameters, which we will set to numeric values to calibrate our model. Current settings are fixed at $A = 8, B = 2$ and $\alpha = 6, \beta = 60$. Fixing parameter $A = 8$ means that the client can have a total of 8 points for having a good income to expense ratio. On the other hand, the client can accumulate a total of 2 points for having an excellent worth to debt ratio. The parameter α is a weight on the expenses, while β is a weight on debt. There are other parameters such as the interest rate and loan period which will be set to 15% interest rate for a one-year loan, and which are used to calculate the Newly Incurred Debt, Service to Debt, Total Debt and Total Expenses (including the new loan).

Now that P is defined, with its variables and parameters, choose an encryption function $E : \mathbb{R}^4 \to \mathbb{R}^m$, where m is the number of encrypted variables. Let $E : \mathbb{R}_{2\times 2} \to \mathbb{R}_{4\times 2}$ the function that encrypts variables in the space of real matrices of size $\mathbb{R}_{2\times 2}$, in the space $\mathbb{R}_{4\times 2}$, and given by

$$E\begin{bmatrix} NI & W \\ TE & TD \end{bmatrix} = \begin{bmatrix} K_1 NI + K_2 & K_4 W + K_5 \\ K_3\sqrt{NI^2+\alpha TE^2} & K_6\sqrt{W^2+\beta TD^2} \\ \frac{K_2}{K_3\sqrt{NI^2+\alpha TE^2}} & \frac{K_5}{K_6\sqrt{W^2+\beta TD^2}} \\ \frac{K_3}{K_1} & \frac{K_6}{K_4} \end{bmatrix}$$

where the K_i are keys of the desired bit length. The resulting object represents the encrypted variables and we will call it matrix EV, where EV_j^i is the element in the i-th column and j-th row. These eight encrypted variables are going to be processed by the bank. We can verify

$$\begin{aligned} P(NI,TE,W,TD) &= 8EV_4^1\left(\frac{EV_1^1}{EV_2^1} - EV_3^1\right) \\ &\quad +2EV_4^2\left(\frac{EV_1^2}{EV_2^2} - EV_3^2\right) \end{aligned}$$

which implies

$$P^*(x_1,x_2,\ldots,x_8) = 8x_4\left(\frac{x_1}{x_2} - x_3\right) + 2x_8\left(\frac{x_5}{x_6} - x_7\right)$$

In this case it is easy to see that keys K_2 and K_5 can be easily found if one has knowledge of the encryption function and the encrypted variables, because $K_2 = EV_2^1 EV_3^1$ and $K_5 = EV_2^2 EV_3^2$. However, to find the original inputs NI, TE, W, TD, one would also have to know K_1, K_3, K_4, K_6, which are not readily found.

Let us illustrate another trivial example of an encryption function. Given the form of the processing function, we can express it in terms of the quotients $Q_1 = \frac{TE}{NI}$ and $Q_2 = \frac{TD}{W}$.

$$P(NI,TE,W,TD) = P(Q_1,Q_2) = \frac{A}{\sqrt{1+\alpha Q_1}} + \frac{B}{\sqrt{1+\beta Q_2}}.$$

This simply means we will encrypt Q_1 and Q_2, instead of encrypting NI, TE, W, TD. We will use the encryption function $E : \mathbb{R}^2 \to \mathbb{R}_{3\times 2}$ defined by

$$E\begin{bmatrix} Q_1 & Q_2 \end{bmatrix} = \begin{bmatrix} \frac{K_1}{Q_1+K_2} & \frac{K_3}{Q_2+K_4} \\ \frac{K_1(\alpha K_2^2+1)}{Q_1+K_2} - 2\alpha K_1^2 K_2^2 & \frac{K_3(\beta K_4^2+1)}{Q_2+K_4} - 2\beta K_3^2 K_4^2 \\ \alpha K_1^2 & \beta K_3^2 \end{bmatrix}.$$

It is easy to verify

$$P(Q_1,Q_2) = \frac{8EV_1^1}{\sqrt{EV_1^1 EV_2^1 + EV_3^1}} + \frac{2EV_1^2}{\sqrt{EV_1^2 EV_2^2 + EV_3^2}}.$$

The corresponding processing function for this encryption function is defined by

$$P^*(x_1,x_2,\ldots,x_6) = \frac{8x_1}{\sqrt{x_1x_2+x_3}} + \frac{2x_4}{\sqrt{x_4x_5+x_6}}.$$

VI. Randomizing Methods

The two encryption functions used in the last section constitute trivial examples of encryption functions, yet it is still difficult to find the original inputs even if one knows the encrypted variables and the encryption function used. We can still add another layer of security. Suppose we have a library of k different encryption functions $\{E_j\}_{j=1}^{k}$, of a given processing function P. If the client encrypts the variables using E_i, for some $E_i \in \{E_j\}_j$, then the encrypted variables will have to be processed with the corresponding processing function P_j^*. Furthermore, the codimension of E_j can be different from another encryption functions codimension.

It is possible to implement a randomized selection of the encryption functions that adds an extra layer of security. After transforming the inputs into the variables, in Module I, the variables are sent to Module II. Now, when Module II receives the variables, it chooses a method from the library of encryption functions $\{E_j\}_j$, to encrypt the variables. Once j is fixed, the vector of variables in $\mathbb{R}^n$ will be transformed into a new vector of encrypted variables. The bank has a corresponding library of processing functions $\{P_j^*\}_j$. Upon receiving the vector of encrypted variables, the bank will be able to identify and run the correct processing function. Having a large library of encryption methods is beneficial for the security of the overall scheme because the data type and the number of encrypted variables can be different each time process P has to be applied to a vector $\mathbf{x}$, making it harder to implement pattern recognition. A basic version of the personal loan Credit Score, detailed above, has been implemented and will be made available for auditing purposes, as well as for establishing collaboration opportunities.

VII. Generalization

In many instances it is desirable to have data encrypted in such a way that it can then be reencrypted for use in one of many different processing functions. Consider a one-party scheme where the User has a numerical database $\mathcal{E}(DB)$ encrypted at rest and wishes to perform operations on elements of the data base, but wants to maintain the data confidential. The finite library of functions for processing the data is $\{P\}_P = \{R, S, \ldots, T\}$. The User requests for any of the functions to be applied to a vector of the database. Suppose we have a library of at least two processing functions, where $R(x, y) = x + y$ and $S(x, y) = xy$. Every processing function $P \in \{P\}_P$ has its own library of encryption functions. The encryption libraries are $\{E_{R,i}\}_i = \{E_{R,1}, E_{R,2}, \ldots\}$, $\{E_{S,j}\}_j = \{E_{S,1}, E_{S,2}, \ldots\}$, etc. Each encryption library has its own cardinality. The User sends a request to the database which includes an encrypted index for a processing function $P : \mathbb{R}^n \to \mathbb{R}$, and the encrypted addresses of the elements of the database. The request is a vector of $n + 1$ coordinates $(\#P, x_1, x_2, \ldots, x_n)$. The first coordinate of the request is an index, $\#P$, indicating to the database the processing function. The other n coordinates of the request are the addresses of the elements of the database. Then, the database randomly selects an encryption function $E : \mathbb{R}^n \to \mathbb{R}^m$ from the set $\{E_{P,j}\}_j$.

The database has a library of processing functions $\{P_E^*\}_{P,E} = \{R_{E_{R,1}}^*, R_{E_{R,2}}^*, \ldots, S_{E_{S,1}}^*, S_{E_{S,2}}^*, \ldots\}$, for all $P \in \{P_j\}_j$ and all $E \in \{E_{P,j}\}_j$. The processing function P_E^* depends on the processing function P and the encryption function E, so that if we change the process P then the encrypted processing function P_E^* is also different. Similarly, if we change the encryption function E then the function P_E^* changes again. The database identifies and applies the correct function P_E^* to the other n encrypted values.

A robust, general purpose encryption scheme for a large library of processing functions, with large libraries of encryption functions is in development. Such a system requires minimizing the number of processing functions and maximizing the number of encryption functions per processing function.

VIII. Conclusions

Although HE is a promising concept in the science of cryptography, the same reasons that make it safe also make it non applicable in many situations. Current energy and time efficiency standards of HE are not met in a wide range of crucial applications. This change of variable method offers many advantages including the fact that the encrypted data can only be used for the intended purposes, and noise and precision are managed without significant efficiency trade-offs.

This scheme is applicable to a wide range of processing functions, including addition and multiplication from which a FHE scheme can be constructed. Trigonometric functions and the numeric derivative can also be homomorphically encrypted. It is a flexible scheme that can be adapted to different number of participants, permission configurations, security demands, efficiency requirements, etc. These characteristics make it a far and wide reaching solution with applications in online security, private Digital Signal Processing, smart cities and traffic problems, Machine Learning such as in specific cases of AI and Neural Network training utilizing sensitive data, operating vectors of encrypted databases, smart grids and energy management, among other activities directly dependent on the difficult marriage of data security and efficient computing. The proposed method can be integrated to SoC by implementing this encryption scheme in a processor architecture with a Simple and Linear Fast Adder having a linear and scaleable topology [14], [15], [16]. Encrypted Processing Units can help achieve secure cloud computing without sacrificing efficiency at hardware and software level.

Acknowledgements

The forementioned software schematics, architecture, algorithms, protocols, diagrams, pseudocode, steps, modules, formulas, strategies, and mathematical models are shared for educational purposes and public exposition, and hold no commercial value. Other specific and general versions, variants derived thereof, trivial modifications from such manipulations of homomorphic encrytpion by change of variable are intellectual property and trade secrets developed by the author for OPERACIONES DIGITALES Y PROCESAMIENTO INTEGRAL DE DATOS ENCRIPTADOS, SAS (Mexico).

Supplementary Material

References

[1] Ronald L. Rivest, Len Adleman, and Michael L. Detouzos. On data banks and privacy homomorphisms. In Foundations of Secure Computation, pages 165–179. Academic Press, 1978. Available at https://people.csail.mit.edu/rivest/pubs.html#RAD78.

[2] C. Gentry, (2009). "A Fully Homomorphic Encryption Scheme," Doctoral Dissertation, Symposium on the Theory of Computing, NY, New York, USA, 2009.

[3] Nathan Dowlin, Ran Gilad-Bachrach, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. CryptoNets: "Applying Neural Networks to Encrypted Data with High Throughput and Accuracy." In 33rd International Conference on Machine Learning (ICML 2016), volume 48 of Proceedings of Machine Learning Research, pages 201–210. PMLR, 2016. URL: http://proceedings.mlr.press/v48/ gilad-bachrach16.html.

[4] Florian Bourse, Michele Minelli, Matthias Minihold, and Pascal Paillier. "Fast Homomorphic Evaluation of Deep Discretized Neural Networks." In Advances in Cryptology – CRYPTO 2018, Part III, volume 10993 of Lecture Notes in Computer Science, pages 483–512. Springer, 2018. doi:10.1007/978-3-319-96878-0_ 17.

[5] Christina Boura, Nicolas Gama, Mariya Georgieva, and Dimitar Jetchev. "Simulating Homomorphic Evaluation of Deep Learning Predictions." In Cyber Security Cryptography and Machine Learning (CSCML 2019), volume 11527 of Lecture Notes in Computer Science, pages 212–230. Springer, 2019. doi:10.1007/ 978-3-030-20951-3_20

[6] Marcelo Blatt, Alexander Gusev, Yuriy Polyakov, and Shafi Goldwasser. "Secure Large-Scale Genome-Wide Association Studies Using Homomorphic Encryption." Cryptology ePrint Archive, Report 2020/563, 2020. https://ia.cr/2020/563.

[7] iDASH secure genome analysis competition. http://www.humangenomeprivacy.org.

[8] Martin R. Albrecht, Rachel Player, and Sam Scott. "On the concrete hardness of learning with errors." Journal of Mathematical Cryptology, 9(3):169–203, 2015. doi:10.1515/jmc-2015-0016.

[9] Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. "Classical hardness of learning with errors." In 45th Annual ACM Symposium on Theory of Computing, pages 575–584. ACM Press, 2013. doi:10.1145/2488608.2488680.

[10] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast fully homomorphic encryption over the torus. Journal of Cryptology, 33(1):34–91, 2020. Earlier versions in ASIACRYPT 2016 and 2017. doi:10.1007/s00145-019-09319-x.

[11] Léo Ducas and Daniele Micciancio. "FHEW: Bootstrapping Homomorphic Encryption in Less than a Second." In Advances in Cryptology – EUROCRYPT 2015, Part I, volume 9056 of Lecture Notes in Computer Science, pages 617–640. Springer, 2015. doi:10.1007/978-3-662-46800-5_24.

[12] Joppe W. Bos, Kristin E. Lauter, Jake Loftus, and Michael Naehrig. Improved security for a ring-based fully homomorphic encryption scheme. In Cryptography and Coding (IMACC 2013), volume 8308 of Lecture Notes in Computer Science, pages 45–64. Springer, 2013. doi:10.1007/978-3-642-45239-0_4.

[13] Ramirez, J. 2015. Systems and Categories. arXiv:1509.03649v5 [math.CT]

[14] J. P. Ramírez, "Simple and Linear Fast Adder of Multiple Inputs and Its Implementation in a Compute-In-Memory Architecture," 2024 International Conference on Artificial Intelligence, Computer, Data Sciences and Applications (ACDSA), Victoria, Seychelles, 2024, pp. 1-11, doi:10.1109/ACDSA59508.2024.10467957.

[15] Ramirez, J. "SIMPLE AND LINEAR FAST ADDER," WIPO, Patentscope. Publication Number: WO/2023/220537. Publication Date: 16/11/2023. Applicant's and Inventor's name: Juan Pablo Ramirez

[16] Ramirez, J. 2023. "Canonical Set Theory with Applications from Matrix Operations and Data Structures to Homomorphic Encryption" Monograph Exclusively on Author's personal home page: www.binaryprojx.com