# The problem of comparison

The problem of comparison over encrypted integers was first introduced by *Yao* in [3] and presented under the name *Yao's Millionaires' problem.* This particular problem discusses in short, about two millionaires *Alice* and *Bob* who are interested to know which one is richer, but without revealing their fortune. A variant of *Yao's Millionaires' problem* is the *Socialist Millionaires' problem* where the two millionaires wishes to compare their riches to determine if its equal or not. These two problems have become essential in cryptography's roadmap, having many applications in *data mining*, *cloud computing* and *e-commerce*, whilst many solutions have been presented, only a few do not require exponential time and space complexity.

One approach to this problem was made possible in *2009*, when a fully homomorphic encryption scheme was presented by *Gentry* [1]. As we've seen in the previous chapters, using properties of the fully homomorphic schemes, one is able to add and multiply homomorphically using encrypted data.

Using these schemes, and stepping on a procedure presented on [2] of comparing homomorphically over encrypted integers, we develop a procedure based on the *HElib* library and test it against its efficiency.

## 0.1 Bit comparison

In our attempt to compare two encrypted integers homorphically, we first need to distinguish how to compare two integers generally. In order to do this we need their binary representations. Let $\mathbf{x}, \mathbf{y}$ to integers and $\mathtt{x}_{n-1}\mathtt{x}_{n-2}...\mathtt{x}_0$, $\mathtt{y}_{n-1}\mathtt{y}_{n-2}...\mathtt{y}_0$ their binary representation respectively, from the most significant bit `(msb)` to less significant bit `(lsb)`. We first need to figure out how to compare two bits `x` and `y`.

The computations are done in $\mathbb{Z}_2$ so the addition and multiplication operations represent the bitwise `XOR` and `AND` gates respectively. The logical gates `XOR` and `AND` behave according to the above tables.

| x | y | x AND y |
|---|---|---|
| 1 | 0 | 0 |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 1 | 1 |

,

| x | y | x XOR y |
|---|---|---|
| 1 | 0 | 1 |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 1 | 0 |

The comparison operators can be expressed using the following relations in $\mathbb{Z}_2$:

1. $\mathtt{x} > \mathtt{y} \Leftrightarrow \mathtt{x} \cdot \mathtt{y} + \mathtt{x} = 1$.

2. $\mathtt{x} = \mathtt{y} \Leftrightarrow \mathtt{x} + \mathtt{y} + 1 = 1$.

Indeed, we can easy check by using the tables, that these two relations are correct.
Let us take the encryptions of $\mathtt{x}$ and $\mathtt{y}$, $\mathtt{Enc(x)}$ and $\mathtt{Enc(y)}$ respectively, we can compute

$$\mathtt{C} = \mathtt{Enc(x)} \cdot \mathtt{Enc(y)} + \mathtt{Enc(x)}$$

If we decrypt $\mathtt{C}$, we will have:

- $\mathtt{Dec(C)} = 1$ when $\mathtt{x} > \mathtt{y}$.
- $\mathtt{Dec(C)} = 0$ when $\mathtt{x} \leq \mathtt{y}$.

Now, if we consider two integers $\mathbf{x} = \mathtt{x}_1\mathtt{x}_0$ and $\mathbf{y} = \mathtt{y}_1\mathtt{y}_0$ with $2-$bits. In order to compare them, we will have:

$$\mathtt{x}_1\mathtt{x}_0 > \mathtt{y}_1\mathtt{y}_0 \Leftrightarrow (\mathtt{x}_1 > \mathtt{y}_1) \vee (\mathtt{x}_1 = \mathtt{y}_1) \wedge (\mathtt{x}_0 > \mathtt{y}_0) = True$$

$$\Leftrightarrow (\mathtt{x}_1\mathtt{y}_1 + \mathtt{x}_1) + (\mathtt{x}_1 + \mathtt{y}_1 + \mathtt{1})(\mathtt{x}_0\mathtt{y}_0 + \mathtt{x}_0) = 1$$

$$\Leftrightarrow \mathtt{x}_1\mathtt{y}_1 + \mathtt{x}_1 + \mathtt{x}_1\mathtt{x}_0\mathtt{y}_0 + \mathtt{x}_1\mathtt{x}_0 + \mathtt{y}_1\mathtt{x}_0\mathtt{y}_0 + \mathtt{y}_1\mathtt{x}_0 + \mathtt{x}_0\mathtt{y}_0 + \mathtt{x}_0 = 1.$$

By the previous relation we have a polynomial

$$F(\mathbf{x}, \mathbf{y}) = F(\mathtt{x}_1, \mathtt{x}_0, \mathtt{y}_1, \mathtt{y}_0) = \mathtt{x}_1\mathtt{y}_1 + \mathtt{x}_1 + \mathtt{x}_1\mathtt{x}_0\mathtt{y}_0 + \mathtt{x}_1\mathtt{x}_0 + \mathtt{y}_1\mathtt{x}_0\mathtt{y}_0 + \mathtt{y}_1\mathtt{x}_0 + \mathtt{x}_0\mathtt{y}_0 + \mathtt{x}_0$$

and the degreee is 3.

Moreover, by a simple induction we can compare n-bit integers using the $n+1$-degree polynomial $F(\mathbf{x}, \mathbf{y})$ . But, by using this approach, the comparison of two $n-$bit integers using the function $F(\mathbf{x}, \mathbf{y})$ becomes more difficult as the $n$ increases, increases as well and the degree of the polynomial $F(\mathbf{x}, \mathbf{y})$.

A more elegant aproach to this problem is to decompose the $n-$bit integers by taking the $l = \lceil n/2 \rceil$ first bit as the most significant bits and the others as the less significant bits.
So if $\mathbf{x} = \mathtt{x}_{\mathtt{n}-1}...\mathtt{x}_l\mathtt{x}_{l-1}...\mathtt{x}_0$ then $\mathtt{msb}(\mathbf{x}) = \mathtt{x}_{\mathtt{n}-1}\mathtt{x}_{\mathtt{n}-2}...\mathtt{x}_l$ and $\mathtt{lsb}(\mathbf{y}) = \mathtt{x}_{l-1}\mathtt{x}_{l-2}...\mathtt{x}_0$.

So the polynomial relations which can evaluate the comparison $\mathbf{x} > \mathbf{y}$ can be done with the above decomposision:

$$\mathbf{x} > \mathbf{y} \Leftrightarrow \mathtt{x}_{\mathtt{n}-1}...\mathtt{x}_l\mathtt{x}_{l-1}...\mathtt{x}_0 > \mathtt{y}_{\mathtt{n}-1}...\mathtt{y}_l\mathtt{y}_{l-1}...\mathtt{y}_0$$

$$\Leftrightarrow (\mathtt{msb}(\mathbf{x}) > \mathtt{msb}(\mathbf{y})) \vee (\mathtt{msb}(\mathbf{x}) = \mathtt{msb}(\mathbf{y})) \wedge (\mathtt{lsb}(\mathbf{x}) > \mathtt{lsb}(\mathtt{y})).$$

Thus, from the above decomposition in order to make the comparison $\mathbf{x} > \mathbf{y}$, first split the bit strings $\mathbf{x}$ and $\mathbf{y}$ in equally parts, compare these parts and then output the final result. Furthermore, the previous decomposision consists boolean logic operations, then it is naturally applicable at the $\mathbb{Z}_2$ group. Therefore, to evaluate the $F(\mathbf{x}, \mathbf{y})$ for $\mathbf{x} > \mathbf{y}$ using $\mathbb{Z}_2$ arithmetic operations. We generalize the operations (1) and (2) to the above recursive functions.

**Theorem 0.1.1** *Let the two recursive functions* $\mathtt{t}$ *and* $\mathtt{z}$ *where:*

$\mathtt{t}_{\mathtt{i},\mathtt{j}}$ *represents the boolean value corresponding to the truth value of the expression*

$$\overline{\mathtt{x}_{\mathtt{i}+\mathtt{j}-1}...\mathtt{x}_{\mathtt{i}}} > \overline{\mathtt{y}_{\mathtt{i}+\mathtt{j}-1}...\mathtt{y}_{\mathtt{i}}}.$$

*$z_{i,j}$ represents the boolean value corresponding to the truth value of the expression*

$$\overline{\mathtt{x}_{\mathtt{i+j-1}}\ldots\mathtt{x}_{\mathtt{i}}} = \overline{\mathtt{y}_{\mathtt{i+j-1}}\ldots\mathtt{y}_{\mathtt{i}}}$$

*are defined as:*

$$\mathtt{t}_{\mathtt{i,j}} = \begin{cases} \mathtt{x_i y_i + x_i}, & \mathtt{j = 1} \\ \mathtt{t}_{\mathtt{i}+l,\mathtt{j}-l} + \mathtt{z}_{\mathtt{i}+l,\mathtt{j}-l}\mathtt{t}_{\mathtt{i},l}, & \mathtt{j > 1} \end{cases}$$

$$\mathtt{z}_{\mathtt{i,j}} = \begin{cases} \mathtt{x_i + y_i + 1}, & \mathtt{j = 1} \\ \mathtt{z}_{\mathtt{i}+l,\mathtt{j}-l}\mathtt{z}_{\mathtt{i},l}, & \mathtt{j > 1} \end{cases}$$

*Where on each iteration we take $l = \lceil \mathtt{j}/2 \rceil$. The solution for the $\mathbf{x} > \mathbf{y}$ and $\mathbf{x} = \mathbf{y}$ can be obtained by computing $t_{0,n}$ and $z_{0,n}$.*

So we can apply the above procedure over encrypted integers, taking as result the encrypted 1 if $\mathbf{a} > \mathbf{b}$ or the encrypted 0 if $\mathbf{a} \leq \mathbf{b}$.

## 0.2 Maximum-Minimum

Knowing how to compare two encrypted integers, on the present section we will describe a method to obtain the maximum-minimum value from two encrypted integers. First, we present an additional function that outputs the maximum of two values. The result of this function depends on the result of the comparison of two input integers.

**Theorem 0.2.1 (Maximum selection operator)** *Let $\mathbf{a}, \mathbf{b}$ two integers and the value $\mathtt{t}$ (where $\mathtt{t} = 1$ if $\mathbf{a} > \mathbf{b}$ or $\mathtt{t} = 0$ if $\mathbf{a} \geq \mathbf{b}$) the comparison operator. We define the selection function depending on value of $\mathtt{t}$ as:*

$$\max : \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z}_2 \to \mathbb{Z}$$
$$\max(\mathbf{a}, \mathbf{b}, \mathtt{t}) = \begin{cases} \mathbf{a}, & \mathtt{t} = 1 \\ \mathbf{b}, & \mathtt{t} = 0 \end{cases} = \mathbf{a}\mathtt{t} + (1 - \mathtt{t})\mathbf{b}.$$

The above function returns the maximum value of the comparison $\mathbf{a} > \mathbf{b}$.
We can consider the above function into a bit-leveled function having all operations in $\mathbb{Z}_2$ which means that the multiplication and the addition can be expressed as `AND` an `XOR` bit-operators.

Using the binary representation for $\mathbf{a} = \mathtt{a}_{\mathtt{n-1}}\mathtt{a}_{\mathtt{n-2}}\ldots\mathtt{a}_0$ and $\mathbf{b} = \mathtt{b}_{\mathtt{n-1}}\mathtt{b}_{\mathtt{n-2}}\ldots\mathtt{b}_0$ and defining as $T = \mathtt{t}_{\mathtt{n-1}}\mathtt{t}_{\mathtt{n-2}}\ldots\mathtt{t}_0$ the integer value having all bits 1 or 0 which represents the output of the recursive function $t_{0,n}$, with input the binary form of $\mathbf{a}, \mathbf{b}$ . We now define the polynomial form of the function max:

$$\max : \mathbb{Z}_2^n \times \mathbb{Z}_2^n \times \mathbb{Z}_2^n \to \mathbb{Z}_2^n$$
$$\max(\mathbf{a}, \mathbf{b}, T) = \sum_{i=0}^{n-1} \mathtt{a}_i\mathtt{t}_i + (1 + \mathtt{t}_i)\mathtt{b}_i.$$

The output of this function will be the binary representation of the maximum number between $\mathbf{a}, \mathbf{b}$. Respectively, we can define the min function as:

$$\begin{aligned} \min &: \mathbb{Z}_2^n \times \mathbb{Z}_2^n \times \mathbb{Z}_2^n \to \mathbb{Z}_2^n \\ \min(\mathbf{a}, \mathbf{b}, T) &= \sum_{i=0}^{n-1} \mathtt{a}_i(1 + \mathtt{t}_i) + \mathtt{t}_i \mathtt{b}_i. \end{aligned}$$

We can easily generalise the comparison between two encrypted integers into a comparison of $N$ encrypted integers.

## 0.3 Implementation

First of all we implemented the function `z` from theorem 0.1.1, over encrypted integers. The function `compute_z`, takes as input the encrypted bits of $\mathbf{a}$ and $\mathbf{b}$ and returns the encrypted boolean value of `Enc`(1) if $\mathbf{a} = \mathbf{b}$ or `Enc`(0) otherwise.

**Algorithm 1** `compute_z`

```
1: Input  : (bits of $\mathtt{Enc}(\mathbf{a})$, bits of $\mathtt{Enc}(\mathbf{b})$)
2: Output :  Enc(1) if $\mathbf{a} = \mathbf{b}$ or Enc(0) if not
3: if j==1 then
4:      $\mathtt{compute\_z}_{\mathtt{i,j}} = \mathtt{Enc}(\mathtt{a_i}) + \mathtt{Enc}(\mathtt{b_i}) + \mathtt{Enc}(1)$
5:    Return $\mathtt{compute\_z}_{\mathtt{i,j}}$
6: else if j>1 then
7:    $l = \lceil \mathtt{j}/2 \rceil$
8:    $\mathtt{compute\_z}_{\mathtt{i,j}} = \mathtt{compute\_z}_{\mathtt{i}+l,\mathtt{j}-l} \cdot \mathtt{compute\_z}_{\mathtt{i},l}$
9:    Return $\mathtt{compute\_z}_{\mathtt{i,j}}$
```

Also, we implemented the function `t` from the theorem 0.1.1 as `compute_t`, with input the encrypted bits of $\mathbf{a}$ and $\mathbf{b}$ and output the encrypted boolean value of the comparison $\mathbf{a} > \mathbf{b}$.

**Algorithm 2** `compute_t`

```
1: Input  : bits of $\mathtt{Enc}(\mathbf{a})$, bits of $\mathtt{Enc}(\mathbf{b})$
2: Output :  Enc(1) if $\mathbf{a} > \mathbf{b}$ or Enc(0) if not
3: if j==1 then
4:      $\mathtt{compute\_t}_{\mathtt{i,j}} = \mathtt{Enc}(\mathtt{a_i}) \cdot \mathtt{Enc}(\mathtt{b_i}) + \mathtt{Enc}(\mathtt{a_i})$
5:    Return $\mathtt{compute\_t}_{\mathtt{i,j}}$
6: else if j >1 then
7:    $l = \lceil \mathtt{j}/2 \rceil$
8:    $\mathtt{compute\_t}_{\mathtt{i,j}} = \mathtt{compute\_t}_{\mathtt{i}+l,\mathtt{j}-l} + \mathtt{compute\_z}_{\mathtt{i}+l,\mathtt{j}-l} \cdot \mathtt{compute\_t}_{\mathtt{i},l}$
9:    Return $compute\_t_{i,j}$
```

Also, the minimum function will be:

$$\min(\texttt{Enc}(\mathbf{a}), \texttt{Enc}(\mathbf{b}), \texttt{compute\_t}(\texttt{Enc}(\mathbf{a}), \texttt{Enc}(\mathbf{b}))) =$$

$$= \sum_{i=0}^{n-1} \texttt{Enc}(\texttt{a}_\texttt{i}) \cdot (\texttt{Enc}(1) + \texttt{compute\_t}_\texttt{i}) + \texttt{compute\_t}_\texttt{i} \cdot \texttt{Enc}(\texttt{b}_\texttt{i})$$

over encrypted values, where $\texttt{Enc}(\texttt{a}_\texttt{i}), \texttt{Enc}(\texttt{b}_\texttt{i})$ is the i-th encrypted bit of **a** and **b** respectivelly. Above we describe the algorithm of the minimum function.

**Algorithm 3** min

```
1: Input  : bits of $\texttt{Enc}(\mathbf{a})$, bits of $\texttt{Enc}(\mathbf{b})$, $\texttt{compute\_t}(\texttt{Enc}(\mathbf{a}), \texttt{Enc}(\mathbf{b}))$
2: Output :  The minimum of $\mathbf{a}, \mathbf{b}$
3: for ( $i = 0$ ; $i < n - 1$ ; $i++$) do
4:     $\min = \texttt{Enc}(\texttt{a}_\texttt{i}) \cdot (\texttt{Enc}(1) + \texttt{compute\_t}_\texttt{i})) + \texttt{compute\_t}_\texttt{i} \cdot \texttt{Enc}(\texttt{b}_\texttt{i})$
5: Return min
```

The result of the function **Algorithm 5** is the encrypted minimum number between **a** and **b**.

Moreover, by generalizing the **Algorithm 5** we are able to find the minimum value on an array of encrypted integers.

**Algorithm 4** minimum of an array

```
1: Input  : An array $V$ of encrypted integers
2: Output : The encrypted minimum integer of the array
3: $\min = \texttt{V}[0]$
4: for ( $i = 1; i < N; i++$) do
5:     t=compute_t(V[i], min)
6:     $\min = \min(\texttt{V}[\texttt{i}], \min, \texttt{t})$
7: Return min
```

By using this approach it is obvious that all the computations are done on the server over encrypted integers. The server does not know the neither the value of the integers nor the value of the minimum or it's position on the input list. The only disadvantage is that the server knows the bit length of the encrypted integers.

To sum up, with the above approach if the client send a list of bit-wise encrypted integers, the server is able to find the encrypted minimum of this list, send it back and if the client decrypt the result, it will be the minimum of this list.

## 0.4 Homomorphic comparison using HElib

The library is an open-source software library developed by *Shai Halevi* and *Victor Shoup* in *2013* and was written in **C++** language alongside with *NTL* mathematical library. Specifically *HElib* is an implementation of the *(BGV)* homomorphic encryption scheme. Up until this time, *HElib* considered as the state of the art library on homomorphic encryption and provides many low-level routines over encrypted ciphertexts such as add, multiply, shift.

A usefull site for understanding *HElib* is `https://mshcruz.wordpress.com/blog/` on which we based our initialization.

In our experiments we implemented the algorithms of the previous section on *HElib*. Next, we provide on tables the homomorphic comparison of 2 and 3 encrypted numbers. We can compare homomorphically with this proccess more numbers than 3. In order to do this, we must increase the level $L$ which will increase also the the degree $\phi(m)$ of the cyclotomic polynomial $\Phi_m(x)$.

Next, we provide the tables in respect to $\mod 2^r$ we use to compare the ecrypted numbers, the minimum level $L_{\min}$ of homomorphic operations and the time $Time$ that the server has done to compare homomorphically.

| mod $2^r$ | $2^4$ | $2^5$ | $2^6$ | $2^7$ | $2^8$ | $2^9$ | $2^{10}$ | $2^{11}$ | $2^{12}$ | $2^{13}$ | $2^{14}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $L_{\min}$ | 9 | 11 | 11 | 13 | 16 | 23 | 25 | 28 | 31 | 33 | 35 |
| $Time(sec)$ | 4.06 | 7.6 | 10.5 | 17.02 | 27.6 | 107.3 | 140.5 | 167.2 | 206.3 | 246.1 | 492.03 |

Table 1: Homomorphic comparison of 2 encrypted numbers. On each instance we ran 4 examples of comparison.

| mod $2^r$ | $2^2$ | $2^3$ | $2^4$ | $2^5$ |
|---|---|---|---|---|
| $L_{\min}$ | 9 | 11 | 15 | 38 |
| $Time(sec)$ | 2.7 | 7.5 | 20.1 | 223.1 |

Table 2: Homomorphic comparison of 3 encrypted numbers.

# Bibliography

[1] C. Gentry. A fully homomorphic encryption scheme. Ph.D. thesis, Stanford University, 2009.

[2] M. Togan, L. Morogan, and C. Plesca. Comparison-Based Applications for Fully Homomorphic Encrypted Data. In: Proceedings of the Romanian Academy Series A16 (2015), pp. 329-338.

[3] A. Yao, Protocols for Secure Computation, Proc. of the 23 FOCS (1982) , 160-164.1