# Analysis of Security-Cost Trade-off of Fully Homomorphic Encryption Schemes

Kais Chaabouni
ENSIMAG
Grenoble INP
Grenoble, France
kais.chaabouni@ensimag.imag.fr

Amrit Kumar
ENSIMAG-Ecole Polytechnique
Grenoble INP
Grenoble, France
amrit.kumar@ensimag.imag.fr

## ABSTRACT

We present a study on variants of fully homomorphic schemes and the trade-off cost/security of basic operations on bits, integers, strings...

## Keywords

Fully Homomorphic Encryption, Security, Cost

## 1. INTRODUCTION

(Motivation and significance of fully homomorphic encryption scheme such as Gentry's lattice-based scheme. This work evaluates the efficiency of FHE implementations for operations varying from XOR to sort on remote machines. Trade-off between cost and security is analyzed.)

A partially homomorphic encryption allows certain operations on ciphertexts such as addition or multiplication which implies a known operation on plaintext like RSA Encryption

$$\mathtt{E}(\prod_{i=1}^{n} a_i) = \prod_{i=1}^{n} \mathtt{E}(a_i)$$

then Gentry proved that there is a fully homomorphic encryption which allows both addition and multiplication on encrypted bits:

$$\mathtt{FHE}(m1) + \mathtt{FHE}(m2) = \mathtt{FHE}(m1 + m2)$$

$$\mathtt{FHE}(m1) \times \mathtt{FHE}(m2) = \mathtt{FHE}(m1 \times m2); \quad m1, m2 \in \{0, 1\}$$

## 2. GENTRY'S FULLY HOMOMORPHIC SCHEME

FHE

### 2.1 The Framework

(Overview of the algorithm and its complexity. ) In this study we use two variants of fully homomorphic encryption:

- Smart-Vercauteren implementation
- Gentry-Halevi-Smart implementation

### 2.2 Example

(Description using a typical operation like multiplication of two $n$-bit integers. Comparison with partial homomorphic encryption scheme such as RSA.)
The multipliction of two FHE-encrypted integers of n bits consists on multiplications and additions on their FHE-encrypted bits:

$$(\mathtt{a}_n, \mathtt{a}_{n-1}, ..., \mathtt{a}_1).(\mathtt{b}_n, \mathtt{b}_{n-1}, ..., \mathtt{b}_1) = (\mathtt{r}_{2n}, \mathtt{r}_{2n-1}, ..., \mathtt{r}_1)$$

We do $n^2$ multiplications $a_i.a_j$ and $2.(n + (n-1)(n-2)$ additions

$$r_1 = a_1.b_1; c_{1,1} = 0$$

$$(r_2, c_{1,2}) = \mathtt{add}(a_2.b_1, a_1.b_2, c_{1,1})$$

$$(r_3, c_{2,3}) = \mathtt{add}(a_1.b_3, \mathtt{add}(a_3.b_1, a_2.b_2, c_{1,2}))$$

$$\vdots$$

$$(r_{n-1}, c_{n-1,n-1}) = ...$$

$$(r_n, c_{n,n}) = \mathtt{add}(a_1.b_n, ...)$$

$$(r_{n+1}, c_{n+1,n+1}) = ..$$

$$\vdots$$

$$(r_{2n-1}, c_{2n-1,2n-1}) =$$

$$(r_{2n}, c_{2n,2n}) = \mathtt{add}(a_n.b_n, c_{2n,2n-1})$$

So it has $O(n^2)$ complexity. In addition to that there is the cost of key generation, encryption and decryption, and there is also the recrypt algorithm called after each addition or multiplication. On the other hand the RSA-Encryption is much more easier and has a O(1) complexity for the multiplication of n-bits integer.

## 3. EXPERIMENTAL EVALUATION OF COST

(Describe the FHE algorithm and its corresponding implementation choice. Installation, set-up procedure and usage of the implementation. We detail the measurements to be taken, like Wall time, CPU time and memory usage based on certain security or algorithmic parameters.

Statistical analysis of the measurements using minimum, max, mean and standard deviation.)

In this section we will experimentally measure time cost of Xor-bits operations using two versions of fully homomorphic encryption programs: libScarab-1.0.0 implementing Smart-Vercauteren algorithm and HElib implementing Gentry-Halevi-Smart algorithm.

## 4. EVALUATION ON BENCH MARKS

We describe the system parameters and test vectors. This section also includes description of the process of evaluation for the following operations :

As the hCrypt implementation provides us with the two gates : `XOR` and `AND` which form a complete set of functional gates. The $\neg$`a` is `XOR(a,1)` while `OR(a,b)` is `XOR(XOR(a,b), AND(a,b))`.

*XOR of $n$-bits.* We start with the simplest operations on bits. `XOR` of $n$-bits requires $n-1$ `XOR` gates.

*Majority of $n$-bits.* Majority bit will be evaluated with the following circuit :

$$\prod_{i=1}^{n} a_i + \sum_{i=1}^{n} (\prod_{j!=i} a_j) \neg a_i$$

Hence it uses $(n^2-1)$ `AND` gates and $n$ `OR` gates.

*Sum of integers of $n$-bits.* Sum of two integers can be performed using an $n$-bit adder based on `full-adder` circuits. The cost of a full adder circuit is 2 `XOR` to calculate the sum and the cost of calculating $3-bit$ majority function for the carry.

*Sum of arbtrary integers*

- Sum of :
  - bounded integers
  - unbounded integers
- String sorting with the following methods :
  - Insertion sort
  - Merge sort
  - Quick Sort
- Matrix product

Sorting is tested on a sorting network where comparator gates are used. For insertion sort we would need $n(n-1)/2$ such gates. Each comparator is designed using the following algorithm.

algoruled [H] $(a_0, a_1, \ldots, a_{n-1})$, $(b_0, b_1, \ldots, b_{n-1})$ Return Max(a,b) and Min(a,b) aIsGreater=0

$i \leftarrow n-1$ 0 aIsGreater=aIsGreater $+\neg(b_i)a_i$

$i \leftarrow 0$ $n-1$ $Max_i$ = aIsGreater*$a_i$ + $\neg(aIsGreater) * b_i$ $Min_i = \neg(aIsGreater) * a_i$ + aIsGreater*$b_i$

return$((Max_0, Max_1, \ldots, Max_{n-1}), (Min_0, Min_1, \ldots, Min_{n-1}))$ Comparator

Each comparator gate uses $n$ `AND` gate and $n$ `OR` gates and $n$ `NOT` gates to find which one of the bit sequence is bigger and then to regenerate the maximum and the minuimum it would require $4n$ `AND` gates $2n$ `NOT` and `OR` gates.

For each operation, we provide the initial algorithm and the results obtained. Observations based on the measurements are described.

## 5. ANALYSIS OF THE RESULTS

Trade-off analysis on the results obtained. Comparison with other protocols in terms of gain with security-cost trade-off. Observations and recommendations on the cost and security trade-off.

## 6. CONCLUSION

Conclusion

## 7. REFERENCES