Homomorphic Encryption Survey Paper

James Bird & **Leonidas Eleftheriou**
Professor Koc

June 13, 2018

Abstract

Homomorphic Encryption (HE) allows for computations on data in the encrypted domain. It provides the opportunity to upload encrypted data sets to a cloud service with privacy concerns and perform operations as advanced as machine-learning-related tasks. In this paper, we will give an overview of existing HE methods and then we will focus on recent attempts to do machine learning using HE.

Introduction

In the traditional sense, *homomorphism* is defined as a mapping that preserves structures. *Homomorphic Encryption*, or *HE*, alludes to the original sense of the word by letting a user perform operations on data while keeping the data hidden. This process preserves the structure of the data while keeping it anonymous to the user that is working with it. There are mainly three types of HE schemes, and they can be broken down into:

- *Somewhat Homomorphic Encryption (SHE)* is a scheme that allow some operations on the encrypted data but can only be done a certain amount of times.
- *Partial Homomorphic Encryption (PHE)* restricts the user to only one operation but that user may perform this operation as many times as they'd like.
- *Fully Homomorphic Encryption* combines the best of both worlds from SHE and PHE and lets the user do an unlimited amount of operations and they can do those operations an unlimited amount of times.

General Homomorphic Encryption

We will use the most common definition, which incorporates the set of all possible plaintexts, $\mathcal{M}$, and the set of all respective ciphertexts, $\mathcal{C}$. Given these two sets, an encryption scheme is homomorphic if, given a key $\mathcal{K}$, the encryption function E satisfies the following:

$$E(m_1) * E(m_2) = E(m_1 * m_2)$$

All HE schemes are primarily and generally denoted by four distinct operations that allow the process to stay secure. Just like in normal cryptographic applications, *KeyGen* generates a pair consisting of a secret and public key. *Enc* and *Dec* also function as they would in normal cryptography, as an encryption & decryption, respectively. The last function, *Ev*, is unique to HE and runs a function $f()$ over the set of ciphers (c_1,c_2) without the need for (m_1,m_2). The function $f()$ takes in a set of ciphertexts and outputs a different set of ciphertexts, importantly of the same size and with the format preserved for decryption.

Somewhat Homomorphic Encryption

SHE works with the assumption that homomorphism properties are restricted to only addition or multiplication. Until FHE was proposed, SHE was extremely useful. Now that FHE has seen significant exposure, some SHE versions of FHE schemes exist for performance reasons, since FHE is computationally expensive. SHE works following the outline for all HE schemes, namely:

- The *KeyGen*: A released public key is formed as ($n = pq$, G, G_1, e, g, h). e is a bilinear map, G and G_1 are groups of order n, and g and h are generators of G.
- The *Enc*: Using g and h, we can encrypt a message m using a random number r from the set $\{0,1,2, .. , n-1\}$ by the following method :

$$c = E(m) = g^m h^r mod\ \ n$$

- The *Dec*: In order to decrypt said ciphertext, first compute:

$$c* = c^{q_1} = (g^m h^r)^{q_1} = (g^{q_1})^m$$

$$m = D(c) = log_Z * c^{q_1}$$

where $Z = g^{q_1}$.

Using the SHE scheme, we can carry out the operations using these functions. For addition using m_1 and m_2:

$$c = c_1 c_2 h^r = (g^{m_1} h^{r_1})(g^{m_2} h^{r_2}) h^r = g^{m_1+m_2} h^{r'}$$

For multiplication, we must set a few requirements, but generally, the process looks like this:

$$c = e(c_1, c_2) h_1^r = e(g^{m_1} h^{r_1}, g^{m_2} h^{r_2}) h_1^r$$

Partial Homomorphic Encryption

PHE has many useful examples and different methods, but for this paper, we will not explore every variant. RSA is a very early example of PHE and uses the hardness of factoring. Once $n = pq$ is chosen, we calculate $\phi = (p-1)(q-1)$ and perform gcd calculations and multiplicative inverses in order to gain a key.

The *Goldwasser-Micali (GM)* scheme is like RSA except that is uses a value called a quadratic nonresidue with respect to $mod\ \ n$. There do exist differences, but we will choose to skip this along with RSA, as they are both very similar and were introduced a long time ago.

The major PHE scheme, and a very popular one at that, is the *El-Gamal*. It is broken down according to our general HE definition in the following way:

- The *KeyGen*: A cyclic group G is used such that order(G) = n and its generator is denoted g. Let $h = g^y$ for a random y from the set $\mathbb{Z}_n^*$. The public key is then set as (G, n, g, h) and x is the secret key.
- The *Enc*: x is randomly chosen from the set 1,2,..,$n-1$ and we get:

$$c = E(m) = (g^x, mh^x) = (g^x, mg^{xy}) = (c_1, c_2)$$

- The *Dec*: The first step here is to compute $c_1^y = \kappa$. Then, we can get the original message via:

$$c_2 \cdot \kappa^{-1} = mg^{xy} \cdot g^{-xy} = m$$

It is also important to note that El Gamal does not work via addition over ciphertexts, only multiplication.

Fully Homomorphic Encryption

The existing and used FWE schemes are mostly based on a few sub-categories, mainly *Learning With Error (LWE)*, *Ring Learning With Error (R-LWE)*, *Lattice Based (LB)*, or the one we will cover here, which is the *Over Integers* FHE method. It is worth noting that [4] refers to it as *Analog In Integers*, whereas [2] does not introduce the *Analog* specification, it simply calls it as we did, *Over Intergers* FHE.

- The *KeyGen*: A random (odd) integer p is created. Security parameter λ and the length of p are also factors here but are not strictly used throughout, so will simply be mentioned.
- The *Enc*: We will encrypt a message, m, by choosing very large random primes p and q and a much smaller random number, r, such that $r << p$. Then we can get a ciphertext via

$$c = E(m) = m + 2r + pq$$

- The *Dec*: We must use the restriction that $m + 2r < p/2$. To get back to m:

$$m = D(c) = (c \ mod \ p) mod \ 2$$

Homomorphism is present over addition and multiplication and preserves the original format. Given that this scheme has many benefits, it must come at a cost somewhere, and that is in efficiency. Since the original method was introduced, many newer methods have come out that have improved some aspect of it. Some examples include: *Scale Invariant FHE Over Integers*, *Integer Plaintexts*, *SHE for large integers without bits*, and *Symmetric FHE without bootstrapping*.

It should be noted that another very interesting method that will not be covered in detail but should be mentioned is the *NTRU* FHE scheme. It uses some of the basics of *Over Integers*, such as $mod \ 2$ in the calculations, use of primes, and simple addition to create the ciphertext. The main unique difference is that the *KeyGen* uses sampled polynomials from a discrete analog to the Gaussian distribution ([2] calls it a 'discrete Gaussian', which should be avoided).

Advanced homomorphic encryption schemes

Garbled Circuits

Having discussed the basics on homomorphic encryption, we can now introduce the notion of *garbled circuits*. Garbled circuits are considered to be both a cryptographic technique and a cryptographic goal. Their history is quite complicated and although they first appeared as a technique independent of the homomorphic encryption, nowadays they are frequently utilized in the same context, interacting with each other, in order to provide secure multiparty secure computation without the presence of a third party.

The idea of a garbled scheme is similar to the idea of the homomorphic encryption. It was first introduced by [7], and the current scheme that is frequently used nowadays was fine-tuned by [6], based on [7]'s initial paper. A *garbling scheme* is a 5-tuple of algorithms $\mathcal{G} = (Gb, En, De, Ev, ev)$. Before we go to the analysis of what these algorithms do, it is important to undeline that, the input x is a sequence of booleans, i.e. $x \in \{0,1\}^n$, and the output y is a sequence of booleans too, $y \in \{0,1\}^m$.

The garbling scheme consists of the following components:

- y and x are associated by the function f which is the function whose functionality the garbling scheme is attempting to preserve. Ideally, the output of the garbling scheme will be identical to the $ev(f, x)$.
- The function $f = (n, m, q, A, B, G)$ resembles a boolean circuit, with n and m being the amount of inputs and outputs, respectively, q being the number of gates, A and B determining the first and second inputs to the gates, respectively, and G determining the functionality of the gates.
- Gb is a randomized algorithm that takes as input the function f and outputs a triplet $(F, e, d) \leftarrow Gb(1^k, f)$. It is required that f be the composition of the three components of the output of the Gb algorithm.
- F is the so called *garbled function* and, in plain english, it is the function that, when evaluated on the encrypted input, and when its output is decoded, should give the same result as evaluating the output of the original function f on the unencrypted input.
- e and d is the pair of the encoding and decoding function, respectively. e operates on the unencrypted input (see next bullet point) and d operates on the encrypted output.
- En and De are the set of encoders and decoders illustrated as boxes. En takes the encoding function e and the initial input x and outputs the garbled input $X = En(e, x)$. De has also 2 input arguments, the garbled output Y and the decoding function d and outputs the final output $y = De(d, Y)$
- Ev is the evaluator box that has inputs X and F and output the garbled output $Y = Ev(F, X)$.
- The *correctness condition* for the garbled scheme strictly mathematically is

$$De(d, Ev(F, En(e, x))) = ev(f, x) \tag{1}$$

  The intuition behind the correctness condition is simply that, the result of calculating the initial function f on the initial input x should be identical to the result that we would obtain by decoding the garbled output that results from applying the garbled function on the garbled input. This condition is the most important part of the garbling scheme since it guarantees the proper functionality of the scheme. Furthermore, this condition is the connection of the garbling scheme to homomorphic encryption schemes that were discussed earlier in this paper.

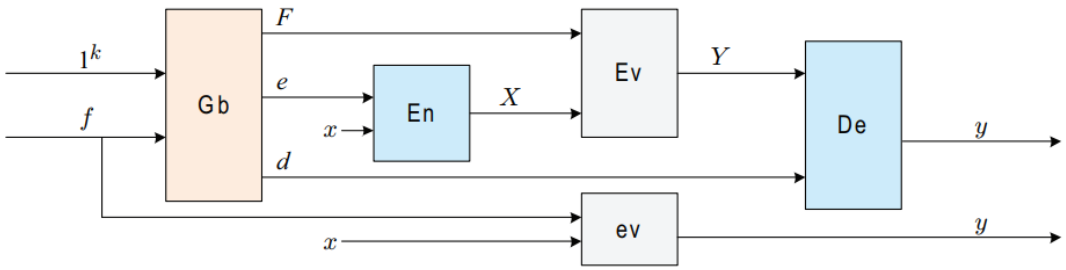

Figure 1: Overview of the garbling scheme

An overview of the scheme can be seen in fig. 1

Another important issue that arises with the appearance of the garbling schemes is the communication protocol of the garbling scheme. For the overview of the garbling scheme protocol we will use the popular analogy of Alice and Bob.

- Both Alice and Bob know the circuit.
- Alice is the garbler and holds the initial inputs. Alice garbles the circuit and the inputs too.
- Alice sends the garbled circuit to Bob, as well as the garbled inputs to Bob using oblivious transfer.
- Bob is the evaluator and, now that he has received the garbled circuit and the garbled inputs, he evaluates the garbled outputs.
- Communication between Alice and Bob is established so they can compare the outputs.

Important part of the aforementioned process is the *oblivious transfer*. In the aforementioned scheme, Alice sends one garbled input at a time in such a way that,

- The other garbled input is not revealed to Bob, and
- Bob cannot know which input specifically was sent to him.

Usually, RSA encryption is used for the implementation of the oblivious transfer protocol.

Machine learning classification using garbled circuits and homomorphic encryption

The operability of garbling schemes on Boolean functions enables us to utilize them in order to perform secure comparisons. Comparisons are the basic component some decision trees that resemble branching programs. Recent attempts have been able to successfully incorporate garbled circuits in order to perform secure comparisons within the context of a decision tree that can be seen in fig. 2 are the following:

- [3] used a simple version of the garbled circuits in order to perform comparisons in a decision tree. Comparison over numeric inputs can be easily translated to a boolean circuit. This boolean circuit can

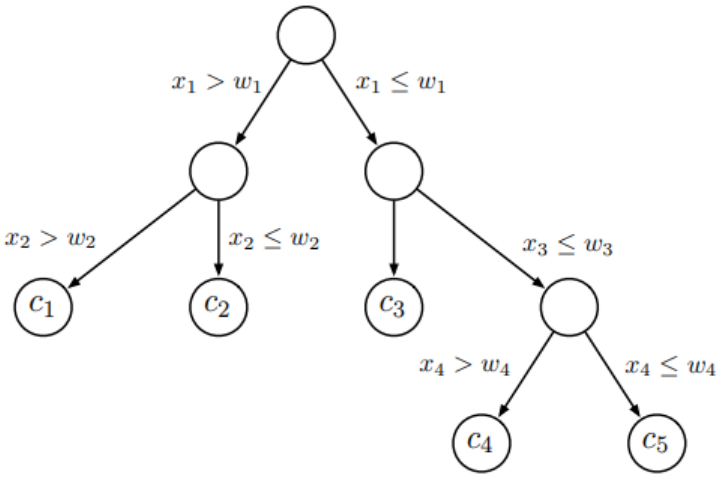

Figure 2: Decision tree that allows for garbled computations

then be garbled along with the inputs and allow for secure branching along its height. Additionally, [3] used the classic homomorphic encryption schemes to perform the argmax operation and the dot product operation, that are commonly used in Naive Bayes classifiers.

- [5] did something similar to [3], but in a more complicated and efficient way. Furthermore, they invented a technique that combines the classic homomorphic encryption and garbled circuits. They applied these techniques in order to classify signals from electrocardiography data. In particular
  - Input to the decision tree is a quantized (namely digitalized, so it fits the boolean circuit context) vector $f^l = (f_1^l, ..., f_n^l)$ whose components will be compared with thresholds of the decision tree in the later steps of the procedure. l is a superscript, not a power, and stands for the size of the digital numbers.
  - Each component f_i^l is encrypted to $[f_i^l]$ using the homomorphic cryptosystem.
  - The n encrypted values are projected onto the homomorphic vector $\mathcal{S}$, a procedure that results in d homomorphic scalar values $[y_i]$.
  - We are almost done, the only step that is left is to give the homomorphic values $[y_i]$ the form that is necessary so they can fit in the garbling scheme. More precisely, we convert them to their garbled equivalents $\tilde{y}_i^{l'}$, a collection of d garbled binary values of size l'.
  - The aforementioned garbled values $\tilde{y}_i^{l'}$ are compared to the thresholds values $\tilde{t}_i^{l'}$ using d garbled circuits.

The goal of the above process is that the overall outcome of the comparisons is the same as if we had not encrypted the data. [5] also use homomorphic encryption schemes in the context of neural network classifiers, a procedure that also breaks down to an argmax calculation in the last step, like the cases that [3] considered.

References

[1] C. Fontaine, F. Galand. "A survey on Homomorphic Encryption for Nonspecialists" *EURASIP Journal on Information Security, Article ID 13801* (2007).

[2] A. Acar, H. Aksu, A. Selcuk Uluagac, M. Conti. "A Survey on Homomorphic Encryption Schemes: Theory and Implementation" *eprint arXiv:1704.03578v2* (2017).

[3] R. Bost, R. Popa, S. Tu, S. Goldwasser. "Machine Learning Classification over Encrypted Data.".

[4] J. Huang, M. Zhang, W. Chen, Y. Tung. https://courses.csail.mit.edu/6.857/2014/files/20-huang-zhang-chen-tung-computing-on-encrypted-data.pdf

[5] M. Barni et al. "Privacy-preserving ECG classification with branching programs and neural networks." IEEE Transactions on Information Forensics and Security 6.2 (2011): 452-468.

[6] M. Bellare, V.T. Hoang, P. Rogaway (2012, October). Foundations of garbled circuits. In Proceedings of the 2012 ACM conference on Computer and communications security (pp. 784-796). ACM.

[7] A.C. Yao (1982, November). Protocols for secure computations. In Foundations of Computer Science, 1982. SFCS'08. 23rd Annual Symposium on (pp. 160-164). IEEE.