Fully Homomorphic Encryption and its Application to Privacy Preserving Machine Learning

Karthik Sridhar Pranit Sinha

1 Introduction

Advancements in Machine Learning have paved the way for the prevalence of Machine Learning as a Service (MLaaS) platforms hosted outside of a user's private computing environment, such as in the cloud. With the continuing success of ML methods in a diverse breadth of application areas, the frequency of individuals relying on such platforms to provide services that range from healthcare diagnosis to credit score approval is bound to increase. The sensitivity of the data involved in such applications should deter users from merely relying on the security of these platforms against breaches or malicious intent.

Fully Homomorphic Encryption (FHE), first described in [1], allows for the evaluation of functions over encrypted data without the evaluator needing to decrypt. Implementing a FHE scheme would guarantee the security of data from user relying on the functionality of a cloud-based server, and would also provide a stronger benefit than the naive solution of asking remote servers to send trained models to users for local inference. In particular, nothing about the server's function (here, a trained model) would have to be disclosed to the user beyond the output of the function the user's inputs. FHE thus provides two-way secure computation, which incentivizes adoption by platforms who seek to protect the weights and coefficients of their models.

Further, FHE is more advantageous than previous schemes for secure function evaluation as described in [2] and [3]. The latter schemes required $O(d)$ rounds of communication where d was the depth of the circuit representation of the function being evaluated. However, FHE achieves an efficiency similar to the insecure setting where the cost of communication is only a function of the length of the data and not the complexity of the function. In the forthcoming sections, we will construct a FHE system allowing homomorphic addition and multiplication, by which we mean that the following properties are preserved:

$$\forall\ (c_1, c_2) = (\mathsf{Enc}(x_1), \mathsf{Enc}(x_2)), \text{we have that } \mathsf{Dec}(f(c_1, c_2)) = f(x_1, x_2)$$

where f is an arithmetic circuit i.e, a computational representation of a polynomial expression. Our scheme is a variant of the GSW scheme [4], which we've worked to present in a manner requiring as few preliminaries not contained in CS-2362 as possible. The TFHE scheme, which is in turn a more complex variant of the GSW scheme, underlies the implementation used in our demonstration application.

2 Learning with Errors

In this section, we define a decisional variant of the Learning with Errors (LWE) problem already introduced in [5]. Informally, this problem presents a bunch of linear equations in $s_1, ..., s_n$ where each equation is approximately correct up to some small error, and a solution requires recovering the values of s_i.

Definition (Decisional LWE). For a security parameter λ, an integer dimension $n = n(\lambda)$, an integer $q = q(\lambda)$, and an error distribution $\chi - \chi(\lambda)$ which is bounded in that $e \leftarrow \chi \in [-b, b]$ for some b, a distinguisher must decide which of two distributions its input is sampled from. Sampling from the first distribution is done by uniformly sampling $(a_i = {}^{(a_{i_1} \ \cdots \ a_{i_n})}, b_i)$ from $\mathbb{Z}_q^{n+1}$. In the second distribution, a_i and s are uniformly sampled from $\mathbb{Z}_q^n$, e_i is drawn from χ and b_i is set to $\langle a_i, s \rangle + e_i$. The solution is 1 in the first case and 0 in the second case.

The assumption that LWE is hard holds that for any probabilistic polynomial time distinguisher $\mathcal{D}$,

$$\Pr\left[\mathcal{D}((a_i, b_i) \text{ drawn from distribution I}) \ = 1\right] \approx \Pr\left[\mathcal{D}((a_i, b_i) \text{ drawn from distribution II}) \ = 1\right]$$

This assumption holds when the value of n is large enough (implementations tend to use 1024 bits or more) and when $\frac{q}{b}$ is small enough (i.e, it is hard to distinguish the signal from the noise). However, the correctness of the scheme also requires $\frac{q}{b}$ to be large enough (see Section 3). The middle ground turns out to be that $\frac{q}{b} = O(2^{n^{1-\epsilon}})$ Finally, the number of vectors in the input to $\mathcal{D}$ is very large - some $m >> n$.

3 Constructing the Scheme

An FHE scheme is different from an ordinary symmetric/public-key encryption scheme in that it involves 4 algorithms - KeyGen, Enc, Dec, Eval. We will describe each of them sequentially and explain what Eval is when the time to define it arises.

The security of our scheme is based on Decisional LWE. Thus, for suitable parameters q, n, χ our key generation algorithm generates a LWE instance over $\mathbb{Z}_q$.

- $\mathsf{KeyGen}(1^\lambda)$: Outputs $\mathsf{sk} = s \xleftarrow{\$} \mathbb{Z}_q^n$
- $\mathsf{Enc}_{\mathsf{sk}}(x)$: Sample $a \xleftarrow{\$} \mathbb{Z}_q^n$, $e \xleftarrow{\$} \chi$, and set $b = \langle a, s \rangle + e + x \left\lfloor \frac{q}{2} \right\rceil$. Outputs (a, b).

Here x is a single bit - either 1 or 0. Why we multiply the bit to encrypt by $\left\lfloor \frac{q}{2} \right\rceil$ will soon become clear. As for now, it should be apparent that the security of Enc relies on the assumption that LWE is hard for the given parameters.

- $\mathsf{Dec}_{\mathsf{sk}}(a, b)$: Compute $b - \langle a, s \rangle$. If $e + x \in [-b, b]$, output 0. Else output 1.

We utilise the fact that based on whether the encrypted bit was 0 or 1, $e + x \left\lfloor \frac{q}{2} \right\rceil$ should lie in $[-b, b]$ or $[-b + \frac{q}{2}, b + \frac{q}{2}]$ respectively. These intervals look different when b is sufficiently small ($b < \frac{q}{4}$).

We will define Eval not as a single algorithm but will demonstrate how it homomorphically carries out addition and multiplication.

3.1 Homomorphic Addition

- $\mathsf{Add}(c_1, c_2)$: Note that $c_i = (a_i, b_i)$. Outputs $c_3 = (a_3 = a_1 + a_2, b_3 = b_1 + b_2$.

We check the correctness of Add.

$$b_1 + b_2 = \langle a_1 + a_2, s \rangle + (e_1 + e_2) + (x_1 + x_2) \left\lfloor \frac{q}{2} \right\rceil$$

$$\mathsf{Dec}(c_3) = b_3 - \langle a_3, s \rangle$$

leaves us with

$$(e_1 + e_2 + e^*) + (x_1 \oplus x_2) \left\lfloor \frac{q}{2} \right\rceil$$

When $x_1 = x_2 = 1$, then $x_1 + x_2$ gives us $2\left\lceil \frac{q}{2} \right\rceil = q + 1$ so $e^* = 1$, otherwise it is 0. Clearly, this will be correct as long as $|e_1 + e_2 + e^*| < \frac{q}{4} \implies |e_i| < \frac{q}{9}$. This correctness will be maintained, as it turns out, for a very large number of additions - about $O(2^{n^{1-\epsilon}})$.

However, our scheme does not allow for homomorphic multiplication yet. In fact, allowing for multiplication turns out to be quite non-trivial. To attain it, we first make a modification to what our ciphertexts look like by redefining Enc and Dec.

3.2 Homomorphic Multiplication

- $\mathsf{Enc}_{\mathsf{sk}}(x)$: Outputs C, computed in the following manner:

$$C = \begin{bmatrix} | & | & & | \\ a_1 & a_2 & \dots & a_m \\ | & | & & | \\ b_1 & b_2 & \dots & b_m \end{bmatrix} + xG$$

  where $b_i = \langle a, s \rangle + e_i$ and G is called a "gadget matrix" and defined as follows:

$$G = \begin{bmatrix} 1 & 2 & 4 & \dots & Q & 0 & 0 & 0 & 0 & & & \dots & & & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 2 & 4 & \dots & Q & 0 & 0 & & \dots & & 0 \\ & & & & & & & \ddots & & & & & & & & \\ 0 & 0 & 0 & & & & \dots & & & & 0 & 1 & 2 & 4 & \dots & Q \end{bmatrix}$$

  Which is to say that $G = g \otimes I_{n+1}$ where g is a vector whose values are the powers of 2 up till Q, the largest power of 2 less than $\left\lceil \frac{q}{2} \right\rceil$. Note that the dimensions of G are $(n+1) \times (n+1)\log q$, and so m must be $(n+1)\log q$.

- $\mathsf{Dec}_{\mathsf{sk}}(a_m, C)$: Consider only the value in the last row and last column of C. This is $\langle a_m, s \rangle + e_m + x\left\lceil \frac{q}{2} \right\rceil$. Decryption then proceeds as defined earlier.

If only one of the values in this matrix are needed to store the ciphertext, what is the need for the entire matrix? Indeed, because it allows us to multiply.

To motivate how multiplication works, observe that if we multiplied C by $[S \mid -1]$ where S was a block matrix consisting only of sk, we'd get $[S \mid -1]C = E + x[S \mid -1]G$, where E is the vector of errors. If we consider E to be negligible for a moment, we may appreciate the x is almost like an eigenvalue and $[S \mid -1]$ is like an eigenvector for G. The values on the right hand side are close to zero. Now, we want something like

$$[S \mid -1]C_1 \cdot C_2 = x_1[S \mid -1]C_2 = x_1 x_2[S \mid -1]$$

but the fact is that direct multiplication does G is not a square matrix, and so C_1 and C_2 are not commensurable.

To resolve this, we seek a new operator

$$G^{-1} : C_{(n+1)\times(n+1)\log q} \mapsto G^{-1}(C)_{(n+1)\log q \times (n+1)\log q}$$

with the property that

$$G \cdot G^{-1}(C) = C$$

If G^{-1} transforms its input C by taking each entry and converting it into a vector which is the binary representation of itself, then we can proceed with multiplication as follows.

$$\begin{aligned} [S \mid -1]C_1 \cdot G^{-1}(C_2) &= (e_1 + x_1[S \mid -1]G)G^{-1}(C_2) \\ &= e_1 G^{-1}(C_2) + x_1(e_2 + x_2[S \mid -1]G) \\ &= w + x_1 x_2[S \mid -1]G \end{aligned}$$

where w is some small error since $G^{-1}(C)$ only has zeroes or ones for entries, and x_1 is a bit so its product with e_2 remains small.

3.3 NAND Circuits

We are now a small step away from achieving fully homomorphic encryption, which is the ability to evaluate arbitrary circuits on encrypted data. Recall that the NAND gate is functionally complete, meaning that any Boolean expression can be re-expressed by an equivalent expression using only NAND operations. Thus, to achieve completeness, we must show that the NAND of two ciphertexts is computable by our system. This is not hard to see, as $\text{NAND}(x_1, x_2) = 1 - x_1 x_2$.

$$\begin{aligned}[S \mid -1](G - C_1 G^{-1}(C_2)) &= -e^* - x_1 x_2 [S \mid -1]G + [S \mid -1]G \\ &= -e^* + (1 - x_1 x_2)[S \mid -1]G \\ &= \text{NAND}(x_1, x_2)\end{aligned}$$

4 Noise analysis and Levelled FHE

Dec fails once e^* exceeds $\frac{q}{4}$. We analyse the boundaries within which the noise components of our ciphertexts are acceptable.

Suppose $\|e_1\|$,$\|e_2\|$ are both at most β. Then since the NAND operation multiplies e_1 with $G^{-1}(C_2)$, which has 0 or 1 entries and m rows, and multiplies e_2 with x_1 which is just a bit, we see that

$$\|e^*\| \leq \beta \cdot m + \beta \leq 2m\beta$$

. This means that with an initial error of β_0, after evaluation of a circuit of depth d (where depth is more or less the number of gates any input must pass through before the circuit outputs),

$$\|e_d\| \leq (2m)^d \cdot \beta_0 \leq \frac{q}{4}$$

where the latter inequality is the condition for correctness. In other words, $\frac{q}{\beta_0}$ must be at least $4(2m)^d$. However, recall that $2^{n^{1-\epsilon}} \geq \frac{q}{\beta_0}$ was the condition for security. Now, since m, n, and ϵ are fixed parameters, we can determine the depth of circuits we can evaluate using our scheme before the error grows too large. We ignore constants for the time being.

$$\begin{aligned}&2^{n^{1-\epsilon}} \geq ((n+1)\log q)^d && \text{(since } m = (n+1)\log q\text{)} \\ &\implies n^{1-\epsilon} \geq d \log((n+1)\log q) && \text{(taking logarithms)} \\ &\implies d \leq O(\frac{n^{1-\epsilon}}{\log n})\end{aligned}$$

What we really have, then, is a *Leveled* Fully Homomorphic Encryption scheme since we can only homomorphically evaluate circuits up to a certain depth.

Now, operations like running an inference on encryption data tend to only involve low depth computations, and so this scheme suffices for our application. For completeness, however, we modify our scheme to allow it to evaluate circuits of arbitrary depth.

5 Bootstrapping

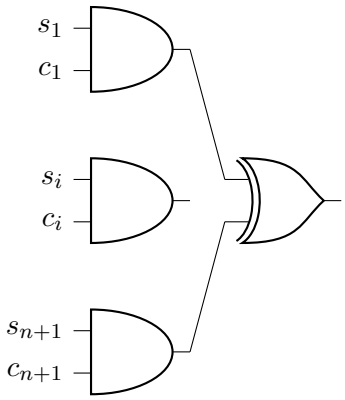
It would be easy to "refresh" the noise to an acceptable level if the untrusted evaluator could decrypt the intermediate value of the computation and generate a new ciphertext homomorphically. Is there some way to simulate this functionality? It turns out that there is - if we encrypt sk and send the evaluator $\mathsf{Enc}_{\mathsf{sk}}(\mathsf{sk})$, they can perform precisely what is described above. Some bad ideas are bad; some bad ideas are good.

Thus, the evaluator only needs the Dec circuit and the encrypted value of sk. Then, they can compute $\mathsf{Dec}_{\mathsf{sk}}(\mathsf{Enc}_{\mathsf{sk}}(\mathsf{sk}), c')$ where c' is a ciphertext holding the value from some intermediate computation of the original circuit being evaluated.

$$\mathsf{Dec}_{\mathsf{sk}}(\mathsf{Enc}_{\mathsf{sk}}(\mathsf{sk}), c') = \mathsf{Enc}_{\mathsf{sk}}(x')$$

Further, the noise in $\mathsf{Enc}_{\mathsf{sk}}(x')$ is only a function of the depth d of the Dec circuit as well as the initial noise $\approx (2m)^d$. It is independent of the noise in c'. The evaluator may apply this procedure whenever the noise grows too large. We may also be sure, due to the indistinguishability of ciphertexts because of the LWE assumption underlying Enc, that sending the evaluator $\mathsf{Enc}_{\mathsf{sk}}(\mathsf{sk})$ does not permit the possibility of sk being recovered.

Finally, we must check that the depth of the Dec circuit is small enough. But we know that $\mathsf{Dec}_{\mathsf{sk}}(C)$ only computes the inner product of $[S \mid -1]$ and the last column of C and checks which interval this value lies in. So the depth of computation depends only on the cost of this inner product computation, which is extremely parallelizable.



So the total number of gates is polynomial in $\log \log q$.

6 ML Inference on a Server

ML models and data-frames can be easily deployed in a client/server setting, enabling the creation of privacy-preserving services in the cloud. As seen in the concepts section, once compiled to FHE, a ML model or data-frame generates machine code that execute prediction, training or pre-processing on encrypted data. Secret encryption keys are needed so that the user can securely encrypt their data and decrypt the execution result. An evaluation key is also needed for the server to securely process the user's encrypted data. Keys are generated by the user once for each service they use, based on the model the service provides and its cryptographic parameters.

The steps detailed above are:

- The model developer deploys the compiled machine learning model to the server. This model includes the cryptographic parameters. The server is now ready to provide private inference. Crypto-graphic parameters and compiled programs for data-frames are included directly in Concrete ML.
- The client requests the cryptographic parameters (also called "client specs"). Once it receives them from the server, the secret and evaluation keys are generated.
- The client sends the evaluation key to the server. The server is now ready to accept requests from this client. The client sends their encrypted data. Serialized data-frames include client evaluation keys.

- The server uses the evaluation key to securely run prediction, training and pre-processing on the user's data and sends back the encrypted result.
- The client now decrypts the result and can send back new requests.

Link to the Git repository: `https://github.com/karthiksridhar22/Karthik_Pranit_CS2362_Project`

References

[1] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, ser. STOC '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 169–178. [Online]. Available: https://doi.org/10.1145/1536414.1536440

[2] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game," in *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, ser. STOC '87. New York, NY, USA: Association for Computing Machinery, 1987, p. 218–229. [Online]. Available: https://doi.org/10.1145/28395.28420

[3] D. Boneh, C. Gentry, and B. Waters, "Collusion resistant broadcast encryption with short ciphertexts and private keys," in *Advances in Cryptology – CRYPTO 2005*, V. Shoup, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 258–275.

[4] C. Gentry, A. Sahai, and B. Waters, "Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based," in *Advances in Cryptology – CRYPTO 2013*, R. Canetti and J. A. Garay, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 75–92.

[5] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," 2024.