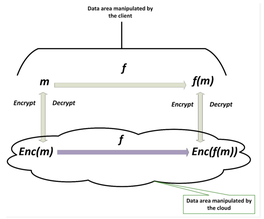
Computing on Encrypted Data with Homomorphic Encryption

Vir Pathak
University of California, Santa Barbara

Introduction

Fully Homomorphic encryption has long been considered the "holy grail" of cryptography because of its far reaching applications. In a nutshell, homomorphic encryption allows us to add and multiply values while these values are encrypted. The encryption function almost serves as a "homomorphism" between the plaintext and ciphertext spaces, if one is willing to think algebraically. An efficient homomorphic encryption scheme would have important applications in privacy-preserving machine learning, since such an algorithm would allow us to run machine learning functions on encrypted data.



Using ideal lattices from algebraic number theory, Craig Gentry was the first to create an encryption scheme which was fully homomorphic. However, his scheme was computationally impractical. In our research, we continued the work on making this technology more applicable to real world scenarios.

Fully Homomorphic Encryption with CKKS

One of the most promising homomorphic encryption algorithms is the CKKS scheme. This algorithm is especially useful because it has a special rescaling procedure which allows us homomorphically perform operations on real(or complex) valued data. Most other schemes are only integer-compatible. We give a rough procedure describing how to encrypt data and with CKKS.

Encoding: Say we have $n/2$ data vectors whose values are entries in $\mathbb{C}$. We encode this data into a plaintext polynomial by exploiting a ring isomorphism $\sigma : \mathbb{C}^{\frac{n}{2}} \to R$ where R is the ring of plaintext polynomials. To be precise, σ is the canonical embedding between $\mathbb{C}^{\frac{n}{2}}$ and $R = \mathbb{Z}[x]/(\phi_M(x))$ where $M = n/2$ and ϕ_M is the Mth cyclotomic polynomial. (Note that $C^{\frac{n}{2}}$ is isomorphic to its image in R under σ, not R itself.)

Encryption: Encryption takes a plaintext polynomial in R and computes a ciphertext polynomial in R_q^2 for some parameter q. Encryption is done with a public key parameter pk, which is essentially a sample from the Ring Learning with Errors (RLWE) Distribution. The security of CKKS relies on the (Decision) Ring Learning With Errors Problem.

Decryption: To decrypt a ciphertext $c = (c_0, c_1) \in R_q^2$, compute $\langle sk, c \rangle \pmod q$.

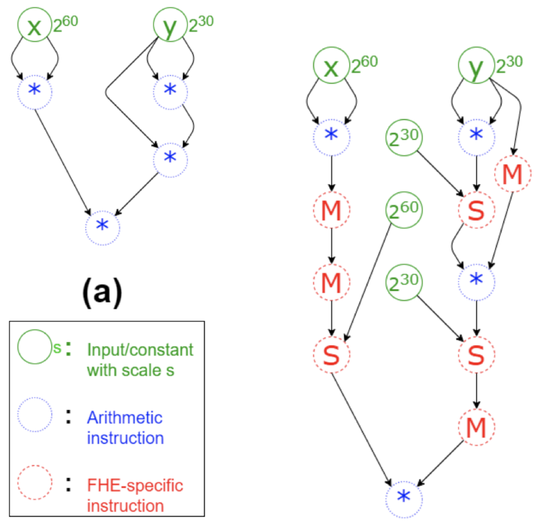
Noise Reduction and Rescaling

CKKS has Add and $Mult$ functions which let us homomorphically add and multiply data vectors using an evaluation key evk. Repeated homomorphic multiplication results in significant noise growth. If the noise exceeds a certain threshold, we will be unable to decrypt properly. For this reason, we create a ladder of gradually decreasing moduli and "switch moduli" in order to compute our ciphertexts modulo a smaller integer compared to before. Roughly, this procedure reduces our noise.

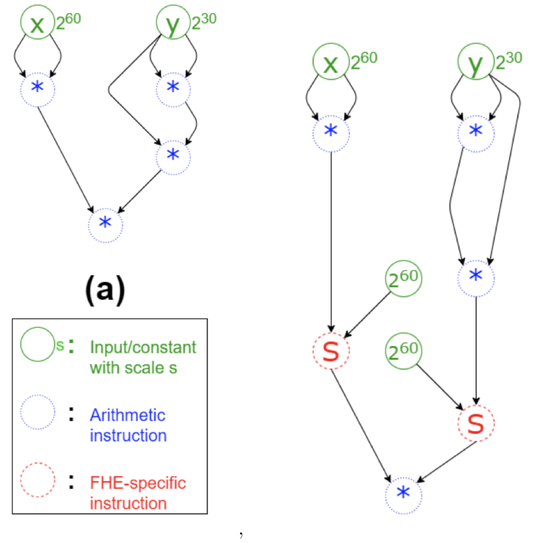
Another problem which arises is that the size of our messages embedded in our ciphertexts grows exponentially with repeated multiplication. For this reason, we need to introduce a rescaling procedure which will let us reset the size of the messages embedded within ciphertexts. Rescaling is essentially

Our Work

To implement Rescale and Modswitching in the most efficient way, we viewed our homomorphic operations as a graph, as in Roshan Dathathri's EVA paper. Each homomorphic operation (add, $mult$, $rescale$, $modswitch$) now becomes a node in our graph. The placement of the add and $mult$ nodes are already determined by the function f which we want to homomorphically compute (in practice think of f as some polynomial). The challenge then reduces to inserting the least amount of $Rescale$ and $Modswitch$ nodes. The solution to this optimization problem is the *Waterline Rescale* method described by Dathathri. To make this more concrete, consider the example given in the paper where they compute the function x^2y^3 homomorphically. The first set of graphs on the top is evaluating f with no rescaling or modswitching and naively evaluating f by rescaling after every single multiplication. We have:



Now look at the graph when we choose to insert our nodes according to Waterline Rescaling.



Clearly, Waterline Rescaling greatly reduces the amount of times we need to $modswitch$ or $Rescale$.

Further Work

To do machine learning on encrypted data, we must understand some other functionalities in CKKS. One such functionality is creating plaintext slots, so one ciphertext encrypts multiple plaintext polynomials. This is called batching, which relies on the concept of *reciprocity* in the polynomial ring R. Once we can batch, we might want permute the batched plaintexts which are encrypted within the corresponding ciphertext. There are techniques available to do this, which rely on some more algebraic number theory along with some Galois Theory. I want to keep reading the papers describing these techniques, because I think the mathematics behind them is very interesting.

Conclusion

In our work, we investigated some ways in making fully homomorphic encryption increasingly practical, and especially focusing on its applications in privacy-preserving machine learning. In particular, we used the techniques in Roshan Dathathri's EVA paper to optimize homomorphic computations in CKKS.

Acknowledgements

I would like to thank the Kelly family for their generosity. I would also like to thank my advisor Cetin Kaya Koc, along with Sam Green. Thank you so much for your help and guidance this summer. I would also like to thank Daniel Guo.

Finally I would like to thank Dylan Adams for being Dylan Adams.

References

Roshan Dathathri - *EVA: An Encrypted Vector Arithmetic Language and Compiler for Efficient Homomorphic Computation*

Zvika Braskerski, Craig Gentry, Vinod Vaikuntanathan - *Fully Homomorphic Encryption Without Bootstrapping*

Jung Hee Cheon, Andrey Kim, Miran Kim, Yongsoo Song - *Homomorphic Encryption for Arithmetic of Approximate Numbers*