Implementing and Benchmarking a LWE-based Fully Homomorphic Encryption Scheme

Meghan L. Clark
University of Michigan

Alex L. James
University of Michigan

Travis B. Martin
University of Michigan

Abstract

Fully homomorphic encryption (FHE) provides a way for third parties to compute arbitrary functions on encrypted data. This has the potential to revolutionize cloud computing services. Unfortunately, since their emergence in 2009, FHE schemes have become notorious for incurring enormous costs in time and space. Over the last four years optimizations have been proposed with impressive rapidity. However, these improvements are usually only asymptotically beneficial. The newness of the field and relative opacity of the literature has resulted in few implementations and evaluations of actual performance. To fill this gap, we implement a recent FHE scheme based on the Learning with Errors (LWE) hardness problem. We compare the performance of our system with an implementation of an earlier FHE scheme based on the Approximate GCD (AGC) hardness problem. We find that the LWE scheme performs [BETTER/WORSE] than the AGCD scheme. We release our system to the public to promote additional experimentation and to increase the accessibility of this new cryptographic construct.

1 Introduction

Fully homomorphic encryption (FHE) is a cryptographic scheme where any function can be computed on encrypted data, and the decrypted result will be the correct answer.

FHE has profound implications for cloud computing services. The security and privacy concerns associated with providing your data to a third party are eliminated, as they will never see your decrypted data, and neither will anyone who compromises their system. Some applications of this are with medical records - hard to get a hold of, run statistics on, but no longer. Satellites - no longer need to trust a third party to keep secrets. Private search queries. Private location-based-services. Secure voting.

Unfortunately, in their current state, FHE schemes incur too much overhead to be used in actual applications. [EXAMPLES OF COSTS].

Since the first FHE scheme was proposed in 2009, combatting the performance costs has been the primary focus of the FHE research community. They propose many optimizations, but mostly asymptotic results. Since they are theory folks and optimizations are coming out so rapidly, there have been few implementations. However, this means that it is hard to gauge whether or not optimizations that are asymptotically faster represent actual performance improvements. Without this feedback, it's hard to tell which branch of FHE to pursue.

Recently a scheme was proposed by [AUTHORS] that involves a new branch of FHE based on LWE. The authors state explicity that while their solutions are asymptotically faster, they don't know whether or not they're actually faster, than AGCD schemes(?).

We answer this call by implementing it and comparing it to an optimized AGCD scheme by Coron, Naccache, and Tibouchi. This comparison adds to the sparse set of real-world FHE performance data points. We are also the first to release LWE FHE code to the public. This can be used for additional paramter exploration, cracking attempts, or a hands-on introduction to an FHE system.

2 Background

In this section we provide a brief history and an explanation of the mechanics of the two main FHE schemes. This will provide the background required for understanding the FHE implementations that we evaluate.

2.1 Circuits

The goal of FHE is to be able to perform arbitrary computations on ciphertext, such that the decrypted result is the same as the result of performing the same computations on the plaintext.

To be capable of arbitrary computation, the cryptosystem need only support two operations, which we will call addition and multiplication, such that

$$D(E(m_1)+E(m_2)) = m_1 + m_2$$

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

where E and D are the encryption and decryption functions.

It has been shown elsewhere that supporting these two operations provides Turing completeness. However, for an intuitive explanation, consider that your computer can run arbitrary programs, yet all of the logical circuits in the hardware can be theoretically implementing using just AND and XOR gates.

For this reason, in the FHE literature all functions to be computed on ciphertext are called "circuits" and are constructed as nested binary operations, such as $(E(1)+((E(1)*E(0)+E(1))$.

Following suit, throughout the rest of this paper we assume that the operands m_1 and m_2 are single bits (either 0 or 1), and we assume that the addition and multiplication operations are equivalent to binary addition (XOR) and binary multiplication (AND).

2.2 Partial Homomorphism

Cryptographic schemes that support just one of the two necessary operations are called *partially homomorphic*. The possiblity of FHE was first suggested after examining the partially homomorphic properties of RSA in 1978 [3], although an actual FHE scheme was not successfullly devised until over thirty years later.

A cryptographic scheme that supports two operations but only for a limited number of successive operations is called *somewhat homomorphic*. A somewhat homomorphic encryption (SWHE) scheme was crucial to the development of the first FHE scheme.

2.3 Lattices

The first working FHE scheme was proposed in 2009 by Craig Gentry [1, 2]. The system was based on a mathematical construct called lattices. In particular, the security of the scheme rested on the hardness of solving certain problems using lattices. However, this system was so complicated and difficult to understand that it immediately gave way to equivalent yet more intuitive integer-based schemes [4, 5] which we will describe in detail in the next section.

Though the FHE literature quickly shifted away from representing ciphertext and keys using lattices, it is worth noting that all current FHE schemes can still ultimately trace their security back to lattice-based hardness problems.

2.4 Approximate GCD Schemes

A year after Gentry described the first FHE scheme, he released a second scheme that translated his lattice-based system into an integer-based system [5]. We will explain FHE using the integer-based system.

Gentry took three stages to create a fully homomorphic scheme.

2.4.1 Somewhat Homomorphic Encryption Scheme

Start with SWHE scheme (describe),

2.4.2 Bootstrapping

(describe).

2.4.3 Squashing the decryption circuit

2.4.4 Problems

The bootstrapping theorem was key to creating FHE. Unfortunately, gotta bootstrap all the time, major time consumer. Plus the size of ciphertexts and keys (though some papers optimized), and also this extra reliance on SSSP.

2.5 Learning With Errors Schemes

Another scheme arose which made intuitive sense. Conveniently did not require SSSP, also eventually produced optimizations that allowed for elimination of bootstrapping, though it could still be implemented as an additional optimization.

3 Related Work

Specifically, the implementation papers - the AES paper, Implementing Gentry (maybe? They actually wrote code, right?), the "Practical" paper, the CNT/Scarab projects, whichever LWE was implemented but not released.

This section is for highlighting what makes our work different from previous work.

4 Methodology

This is about the experimental design. Start off with a super high-level description of our approach. A.k.a, given an existing Python SAGE implementation for AGCD, write our own LWE implementation also in Python SAGE, and then benchmark each on a variety of parameters described here.

Figure 1: Description of what information is being displayed. Results/trends of note. What those results/trends mean (impact).

IMPORTANT: This is where we make the case that our benchmarking comparisons are fair. This whole section is not just describing but also *justifying* our experimental design choices. Namely, the optimizations we chose, the parameters we chose, etc., and why those choices make it fair.

This is the most important section of the paper. This is where the reader can tell whether we did good science or bad science.

5 Implementation

The nitty-gritty about the implementation. This can just be a straight-up description about how our code works.

If you want to include code snippets, here's some example LaTeXfor that:

```
int main() {
    int result = 1
    if (result != 2) {
        printf("You get here every time.\n");
    }
}
```

6 Results

Here's a typical figure reference. The figure is centered at the top of the column. It's scaled. It's explicitly placed. You'll have to tweak the numbers to get what you want.

This text came after the figure, so we'll casually refer to Figure 1 as we go on our merry way.

7 Discussion

Can be short. Talk about the main results, particularly the impact. Were our results reasonable or surprising? Do our results mean that we should advocate for LWE or AGCD? Are they too close to call? Or are our results too specific to the implementations and optimizations used? When might we get different results?

8 Future Work

Can be short. Stuff not only that you'd want to do on this project in the future, but what any FHE researcher might try next after having read this.

9 Conclusion

Despite the alacritous pace at which theoretical work is being done to improve the performance of fully homomorphic encryption, no one is doing sanity checks by implementing these schemes. This makes it difficult to know what actual progress is being made and which directions are most promising. To address this deficit, we implemented a LWE-based FHE scheme in Python and Sage and compared it to an existing implementation of an AGCD-based FHE scheme, also written in Python and Sage. We found that the LWE implementation performed [BETTER? To WHAT DEGREE?] than the AGCD implementation. In addition to expanding the corpus of FHE performance benchmarks, we also publicly release our code. We hope that this work helps provide direction to future FHE research and increases the accessibility of this new and exciting area of cryptography.

10 Availability

We feel very strongly that releasing our implementation to the public for examination and experimentation is one of the major contributions of this work. Our code and directions for running it can be found at:

`https://github.com/tbmbob/bootstraplessfhe`

If you have questions, we will do our best to answer them.

References

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

[2] Gentry, C. Fully homomorphic encryption using ideal lattices. In *STOC '09: Proceedings of the 41st Annual ACM Symposium on Theory of Computing* (2009), pp. 169–178.

[3] Rivest, R. L., Adleman, L., and Dertouzos, M. L. On data banks and privacy homomorphisms. *Foundations of secure computation 32*, 4 (1978), 169–178.

[4] Smart, N. P., and Vercauteren, F. Fully homomorphic encryption with relatively small key and ciphertext sizes. In *Public Key Cryptography–PKC 2010*. Springer, 2010, pp. 420–443.

[5] Van Dijk, M., Gentry, C., Halevi, S., and Vaikuntanathan, V. Fully homomorphic encryption over the integers. In *Advances in Cryptology–EUROCRYPT 2010*. Springer, 2010, pp. 24–43.