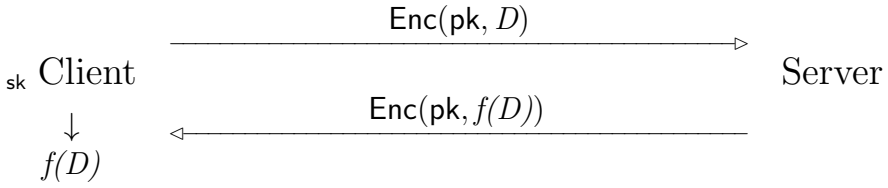
COS 533: Advanced Cryptography — Princeton University
Lecture 14 (Mar 25, 2021)
Lecturer: Mark Zhandry — Scribe: Ezra Edelman

Notes for Lecture 14

1 Introduction

In this lecture we will discuss fully homomorphic encryption. Consider the following motivating example where the client has some secret key sk, and the server and client share some public key pk,



A client has some large database D that they store on a server. Since the client doesn't want the server to know whats on the database, they encrypt it. Later, the client wishes to learn some function of the database (ex: some statistical analysis), so they want to find $f(D)$ for some function f.

One possible solution is for the server to send back the encrypted database, and then the client can decrypt it and evaluate $f(D)$. But, this is computationally intensive and the client would have to store the database (which they might not have the capacity to do). So, the client would like the server to do the calculation required to evaluate f, while keeping the database encrypted.

We want an encryption scheme that allows server to compute $f(D)$ without ever learning anything about it or D.

2 Fully Homomorphic Encryption

In a fully homomorphic encryption scheme there are some addition and multiplication operations on the cipher text that correspond to addition and multiplication of the plaintext.[1]

[1] Another way to think about this is that the decryption function is sort of a homomorphism between the plaintext ring and ciphertext ring. This is where the name comes from.

2.1 Key features

- CPA security[2]
- Additively homomorphic:

  $\mathsf{Enc}(pk, x_0) \oplus \mathsf{Enc}(pk, x_1)$[3] looks like $\mathsf{Enc}(pk, x_0 + x_1)$
- Multiplicativly homomorphic:

  $\mathsf{Enc}(pk, x_0) \otimes \mathsf{Enc}(pk, x_1)$ looks like $\mathsf{Enc}(pk, x_0 \times x_1)$

Questions

Q. Are we assuming that the x_is are elements of some field, or a ring (or some other structure)?

A. Usually we imagine some ring. In the scheme we'll see, the x_is will be binary (so operations will end being over the field of two elements).

Q. What do you mean "look like". Do you mean equal, or can be computed from?

A. For certain schemes, the additive and multiplicative operations don't result in the same distribution as the original encryption scheme. We're going to be using a scheme based off of learning with errors (LWE), and the noise term from LWE will be increased every time you do the operations. The point is that decryption still works, and you should be able to apply more and more operations.

Q. Do we expect encryption to be deterministic, with no random component.

A. No, this is one reason what we are doing is imprecise. If we have CPA security, we need randomness. The FHE can in principle be deterministic, but we still won't get equality[4] since the output of Enc is a distribution. The right way to define FHE is to define it in terms of the decryption function, since (we generally assume) that decryption is deterministic. Then, we can use an equality condition for the homomorphic properties.

[2] Can't get CCA security.

[3] $\oplus$ does not necessarily refer to exclusive or.

[4] Specifically, between $\mathsf{Enc}(pk, x_0) \oplus \mathsf{Enc}(pk, x_1)$ and $\mathsf{Enc}(pk, x_0 + x_1)$

2.2 Trivial "Solution"

$$c_0 \oplus c_1 = (\oplus, c_0, c_1)$$
$$c_0 \otimes c_1 = (\otimes, c_0, c_1)$$
$$Dec(\circ, c_0, c_1) = Dec(c_0) \circ Dec(c_1)$$

This is uninteresting since the client has to apply f themselves. So we have to put some restrictions on the format of the output, but don't need the output to look exactly like the encryption scheme. For the scheme we will construct, it will be obvious that we're doing something non trivial.

2.3 El Gamal: Multiplicative Homomorphism

Using the compent-wise $\times$ operation,

$$c_0 = g^{r_0}, h^{r_0} \times m_0$$
$$c_1 = g^{r_1}, h^{r_1} \times m_1$$
$$c = c_0 \times c_1 = g^{r_0+r_1}, h^{r_0+r_1} \times (m_0 \times m_1)$$

So c is $\mathsf{Enc}(pk, m_0 \times m_1)$ using the randomness $r_0 \times r_1$.

2.4 Lattice-based scheme: Additive Homomorphism

Adding the cipher text components together yields an encryption of the sum of the cipher texts, but the error rate is the sum of the error rates of the two cipher texts. The decryption will still work as long as error rate doesn't get too large.

Going back to the 70s, we had these additive or multiplicative homomorphic encryption schemes, but it wasn't until around a decade ago that it was discovered how to achieve additive and multiplicative homomorphism simultaneously.

3 Attempt 0

Won't be secure, but its a start.

$$\mathsf{sk} \in \mathbb{Z}_q^m \qquad \text{(prime } q\text{, integer } m\text{)}$$
$$\mathsf{Enc}(m) : \text{matrix } C \in \mathbb{Z}_q^{m \times m} \text{s.t. } C \cdot \mathsf{sk} = m \cdot \mathsf{sk}$$

In other words, sk is an eigenvector of C with eigenvalue m. Lets just check that this is homomorphic, and ignore security and decryption for now.

Suppose that C_0 encrypts m_0, and that C_1 encrypts m_1

$$\begin{aligned}(C_0 + C_1) \cdot \mathsf{sk} =& (m_0 + m_1) \cdot \mathsf{sk} \\ &\implies C_0 + C_1 \text{ encrypts } m_0 + m_1\end{aligned}$$

$$\begin{aligned}(C_0 \cdot C_1) \cdot \mathsf{sk} =& C_0 \cdot (m_1 \cdot \mathsf{sk}) = (m_0 \cdot m_1) \cdot \mathsf{sk} \\ &\implies C_0 \cdot C_1 \text{ encrypts } m_0 \cdot m_1\end{aligned}$$

This is insecure: use eigendecomposition to recover all eigenvalues and eigenvectors. One of the recovered eigenvectors will be sk, one of the eigenvalues will be the message. So, if you see enough ciphertexts, you can find the eigenvalue that appears for all of them, and that is sk.

4 Idea 1

We will try to add noise to prevent eigendecomposition.

We take motivation from LWE. LWE takes a task that is trivial with linear algebra (Gaussian elimination) then adds noise to make it secure while still preserving correctness. So we are going to try to "port" that idea to here, and add noise to Idea 0 while maintaining correctness but prevents linear algebra (eigendecomposition) from defeating the scheme.

$$\begin{aligned}&\mathsf{sk} \in \mathbb{Z}_q^m \qquad \text{(prime } q\text{, integer } m\text{)} \\ &\mathsf{Enc}(m) : C \in \mathbb{Z}_q^{m \times m} \text{ s.t. } C \cdot \mathsf{sk} = m \cdot \mathsf{sk} + e\end{aligned}$$

What we want is for e is a short error vector.

Now, eigendecomposition won't necessarily find sk or m.

Suppose that C_0 encrypts m_0, and that C_1 encrypts m_1

4.1 Addition

$$(C_0 + C_1) \cdot sk = (m_0 \cdot \mathsf{sk} + e_0) + (m_1 \cdot \mathsf{sk} + e_1) = (m_0 + m_1) \cdot \mathsf{sk} + (e_0 + e_1)$$

Notice that $e_0 + e_1$ is still a short error vector, but slightly larger than e_0 or e_1.

Questions

Q. Would we have a problem if we compose this many times resulting in the error growing exponential in terms number of operations?

A. For addition notice that the size of the error doubles each time we add. So the size of the error is exponential in operation depth. What we can do is set q to be exponentially large a priori, and then we can do computations up to a certain depth logarithmic in q. This limitation is going to be present in any scheme we can prove secure from LWE.

Q. Does this mean that we are only able to do computations in the class AC^0?

A. No. Recall that with an exponential modulus, the bit length is still polynomial. So, for any computation of depth t, we can choose q such that it is larger than 2^t. But, this does mean that the client will have to do more work, so the only advantage is for shallow computation since the client pays for the depth. Next time we will talk about a "bootstrapping" that can be done to further increase the depth of the computation done by the server, but it comes at the cost of a heuristic component that we don't know how to prove.

4.2 Multiplication

$$\begin{aligned}(C_0 \cdot C_1) \cdot \mathsf{sk} =& C_0(m_1 \cdot \mathsf{sk} + e_1) \\ =& C_0(m_1 \cdot \mathsf{sk}) + C_0(e_1) \\ =& m_0 \cdot m_1 + \mathsf{sk} + (m_1 \cdot e_0 + C_0 \cdot e_1)\end{aligned}$$

If $m_1 \cdot e_0 + C_0 \cdot e_1$ remains small, then $C_0 \cdot C_1$ encrypts $m_0 \cdot m_1$. So, we need

1. m_1 to be small

   This is the easier case. We will always assume messages are binary. Notice that multiplication of binary messages will result in binary. Addition (which is mod q) does not preserve binary, because $1 + 1 = 2$.[*citation needed*] We need to make sure that addition also keeps it in binary, to do this we will use a multiplication to turn addition into an xor.

   $$m_0 \oplus m_1 = m_0 + m_1 - 2(m_0 \cdot m_1)$$

   So the size of m_1 isn't much of an issue. So the bigger issue is the size of C_0.

2. C_0 to be small

   Our solution to this will use a gadget matrix.

5 Idea 2

Gadget Matrix

$$G = \begin{pmatrix} 1 & 2 & 4 & 8 & \dots & \lceil 2^{\log_2 q} \rceil & & & & & & & & & \\ & & & & & & 1 & 2 & 4 & 8 & \dots & \lceil 2^{\log_2 q} \rceil & & & \\ & & & & & & & & & & & & 1 & 2 & \dots \\ & & & & & & & & & & & & & & \ddots \end{pmatrix}$$

$$G \in \mathbb{Z}_q^{m \times (m \log_2 q)}$$

$$v \in \mathbb{Z}_q^m \xrightarrow{\text{bit decomp}[6]} v' \in \mathbb{Z}_q^{z \log q}$$

$$v' := G^{-1}(v)$$

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

This can also be extended to matrices

$$M \in \mathbb{Z}_q^{m \times n} \xrightarrow{G^{-1}(M)} M' \in \mathbb{Z}_q^{m \log q \times n}$$

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

G takes in a matrix, does bit decomposition component wise, and assembles each component into a vector, then these vectors in a block matrix. Let $m = O(n \log q)$, then

- $\mathsf{Gen}() \to \mathsf{sk}, \mathsf{pk}$

$$\mathsf{sk} = \begin{pmatrix} s \\ -1 \end{pmatrix} \qquad s \leftarrow \mathbb{Z}_q^{n-1} \text{ is uniform}$$

$$\mathsf{pk} = \begin{pmatrix} P \\ s^T p + e^T \end{pmatrix} \qquad P \leftarrow \mathbb{Z}_q^{(n-1) \times m} \text{ is uniform}, e \text{ is short noise}$$

Questions

- $\mathsf{Enc}(\mathsf{pk}, m)$ (recall) m is binary but can be interpreted as base q.

$$C = pk \cdot R + mG$$
$$R \leftarrow \{0,1\}^{m \times m}$$

Take pk, multiply it (on the right) by a random $(0,1)-$matrix, then add the message m times the gadget matrix.

[6] For each entry of v, interpret that entry as an integer between 0 and $q-1$, and then each bit will be an entry in v'.

- $\mathsf{Dec}(\mathsf{sk}, C) \to \mathsf{sk}^t \cdot C$

$$\begin{aligned}\mathsf{sk}^T \cdot C &= \mathsf{sk}^T \cdot \mathsf{pk}R + m \cdot \mathsf{sk}^T \cdot G \\ \mathsf{sk}^T \cdot \mathsf{pk} &= s^T \cdot P - (s^T \cdot P + e^T) = -e^T \\ \mathsf{sk}^T \cdot C &= m \cdot \mathsf{sk}^T \cdot G - e^T \cdot R\end{aligned}$$

  Since $\mathsf{sk}^T \cdot G$ looks like $\begin{pmatrix}\mathsf{sk}^T & 2\mathsf{sk}^T & 4\mathsf{sk}^T & \dots\end{pmatrix}$. There will be some component where $\mathsf{sk}^T G$ is big but $e^T \cdot R$ is small, so by checking if this component is close to zero we find if m is 0.

5.1 Addition

$$C_0 \oplus C_1 = \mathsf{sk}^T(C_0 + C_1) = \mathsf{sk}^T C_0 + \mathsf{sk}^T C_1 = \mathsf{sk}^T \cdot \mathsf{pk}(R_0 + R_1) + (m_0 + m_1)\mathsf{sk}^T \cdot G$$

Notice that $R_0 + R_1$, while not $(0, 1)$, is still small, so everything is fine.

5.2 Multiplication

We can't just do $C_0 \cdot C_1$ since the dimensions don't match, and if C_1 is large, the error will be large. Instead,

$$C_0 \otimes C_1 = C_0 \cdot G^{-1}(C_1)$$

First, this fixes the dimension mismatch, and also fixes problem with large entries. Notice that the error will just depend on $G^{-1}(C_1)$, which has $(0, 1)$ entries which are small.

$$\begin{aligned}\mathsf{sk}^T \cdot C_0 \cdot G^{-1}(C_1) =& (\mathsf{sk}^T \cdot \mathsf{pk} \cdot R_0 + m_0 \cdot \mathsf{sk}^T \cdot G) \cdot G^{-1}(C_1) \\ =& \mathsf{sk}^T \cdot \mathsf{pk} \cdot R_0 \cdot G^{-1}(C_1) + m_0 \cdot \mathsf{sk}^T \cdot C_1 \\ =& \underbrace{\mathsf{sk}^T \cdot \mathsf{pk} \cdot R_0 \cdot G^{-1}(c_1) + m_0 \cdot \mathsf{sk}^T \cdot \mathsf{pk} \cdot R_1}_{\text{error}} + m_0 \cdot m_1 \cdot \mathsf{sk}^T G\end{aligned}$$

$$\begin{aligned}\text{error} =& \mathsf{sk}^T \cdot \mathsf{pk} \cdot R_0 \cdot G^{-1}(c_1) + m_0 \mathsf{sk}^t \cdot \mathsf{pk} \cdot R_1 \\ =& \underbrace{\mathsf{sk}^T \mathsf{pk}}_{e^T}(R_0 \cdot G^{-1}(C_1) + m_0 \cdot R_1)\end{aligned}$$

Notice that each term is small, so error is small. So, we got what we want,

$$C_0 \oplus C_1 = C_0 \cdot G^{-1}(C_1) = m_0 \cdot m_1 sk^T G + e'$$

where e' is small.

Questions

Q. Just because G^{-1} and R_1 are zero/one, they are still large matrices so multiplying them might result in large error.

A. Error is going to be $\mathsf{poly}(m)$. After computation of depth t, error is $\mathsf{poly}(n)^t$. One solution is to ensure $q >> poly(n)^t$. Another is bootstrapping.

6 Next Time: Bootstrapping Theorem

The bootstrapping theorem will show how to take FHE for shallow computations (ones with a priori bounded depth) and build FHE for arbitrary computation. There will be a heuristic component (we can't necessarily prove this in general) but it seems to work for the encryption schemes we have.

Also, we will show security for the FHE scheme.

Questions

Q. Are there any other ways to do this, this seems rather inefficient.

A. Short answer is no. If you just want only additive or only multiplicative homomorphic encryption, there are much faster ways. But for fully homomorphic encryption, there are only two known routes, and the one we didn't show, obfuscation, is even worse in terms of efficiency. Overhead is around 10^6 to 10^9 operations per homomorphic operation, and bootstrapping is even worse, one step of the bootstrap scheme corresponds to many of the underlying scheme.

Using clever optimizations, the state-of-the-art is to have the bootstrapping in something on the order of one second. This is certainly not ready for commercial applications, that said in the past ten years since its been invented, the overhead has come down substantially, and we can now do non-trivial operations with FHE.