COS 533: Advanced Cryptography — Princeton University
Lecture 18 (November 20, 2017)
Lecturer: Mark Zhandry — Scribe: Fermi Ma

Notes for Lecture 18

1 Gentry's Fully Homomorphic Encryption Scheme

A fully homomorphic encryption scheme is a scheme that allows you to perform computations on ciphertexts without having a secret key. Last time we sketched an approach with approximate eigenvectors and eigenvalues.

1.1 Construction

Last time, we covered something called a gadget matrix. This is a fancy word for bit decomposition. The matrix looked like

$$\begin{pmatrix} 1 & 2 & 4 & 8 & \cdots & & & & & & \\ & & & & 1 & 2 & 4 & 8 & \cdots & & \\ & & & & & & & & & 1 & 2 \end{pmatrix}$$

We called this matrix G. This has m rows and $n \log q$ columns. We also have a matrix G^{-1} that takes vectors $V \in \mathbb{Z}_q^n$, and the product is $u \in \{0,1\}^{n \log q}$. It has the property that $G \cdot G^{-1}(V) = V$. G^{-1} takes things and writes them down as bits. Basically we think of G as reassembling the vector using the powers of 2.

We can also think of this as applying to matrices. We have $G \cdot G^{-1}(A \in \mathbb{Z}_q^{m \times m}) = A$.

The scheme works as follows.

- $\mathsf{Gen}()$. The secret key is $sk = s = \begin{pmatrix} s' \\ -1 \end{pmatrix}$ where $s' \leftarrow \mathbb{Z}_q^{n-1}$ is randomly sampled. We have a public key $pk = P = \begin{pmatrix} P' \\ s'^T P' + e^T \end{pmatrix}$ where $P' \leftarrow \mathbb{Z}^{(n-1) \times m}$ is randomly sampled. So first I generate a random matrix P', which gives me the first $n-1$ rows of my public key. The final row is given by multiplying $s'^T P'$ and then adding some noise e^T. Basically we are constructing an LWE sample in the bottom row.
- $\mathsf{Enc}(pk, x)$. For now we think of $x \in \mathbb{Z}_q$. Later we'll change it to be in binary. The ciphertext is of the form $PR + xG$ where G is the gadget matrix and $R \leftarrow \{0,1\}^{m \times m}$.

- $\mathsf{Dec}(sk, C)$. I take my secret key s and left multiply by C. Note that the dimensions of $s^T C$ match up. We get $s^T C = s^T PR + x s^T G$ (note that x is a scalar so it commutes with everything). Notice that $s^T P$ is small, since $s^T P = s'^T P'^T - (s'^T P' + e^T) = e^T$, and e^T was chosen to be a small error term. Note that $s^T G$ looks like $\begin{pmatrix} s^T & 2s^T & 4s^T & \cdots \end{pmatrix}$. Since this is a big vector (components are large relative to q), I can look at one coordinate of the vector and recover the scalar x.

Why do we need these powers of 2? They'll be necessary for the homomorphic operations.

1.2 Homomorphic Operations

- Addition. Given two ciphertexts C_0 encrypting x_0 and C_1 encrypting x_1. Adding these ciphertexts and decrypting gives $s^T(C_0 + C_1) = s^T C_0 + s^T C_1 = s^T P(R_0 + R_1) + (x_0 + x_1) s^T G$. We need to be careful here, since we said we were restricting x to be binary, and now it may be up to 2 (for example). The small terms are still small, but less small than before.

- Multiplication. Remember last time when we talked about approximate eigenvectors, we said that when you multiply two ciphertexts together as matrices, you get the a matrix whose eigenvalues are the product of the eigenvalues of the original matrices. But here we can't do that, as the dimensions don't even match up. But we can compute $C_0 \cdot G^{-1}(C_1)$. Remember that the bit decomposition works column by column, so it transforms C_1 into a binary matrix of the correct dimensions. Now I try to decrypt as

$$\begin{aligned} s^T C_0 \cdot G^{-1}(C_1) &= (s^T P R_0 + x_0 s^T G) \cdot G^{-1}(C_1) \\ &= s^T P R_0 G^{-1}(C_1) + x_0 s^T C_1 \\ &= s^T P R_0 G^{-1}(C_1) + x_0 s^T P R_1 + x_0 x_1 s^T G \end{aligned}$$

  The observation here is that if we group the first two terms together, it looks like an error term, and the last term is the part of the ciphertext encrypting $x_0 x_1$. All we have to verify is that $s^T P(R_0 G^{-1}(C_1) + x_0 R_1)$. We know that $s^T P$ is small. We know that R_0 is small since it was picked that way. $G^{-1}(C_1)$ is a binary matrix, so it's small (this is where the binary decomposition helps us). And $x_0 R_1$ is small as long as the plaintext x_0 is small. So if we keep the plaintexts binary, this will be small and we will decrypt correctly.

Security is actually pretty straightforward from the Learning With Errors (LWE) assumption. Note that the LWE step appears when we look at the bottom row of P, which is $s'^T P' + e^T$. LWE tells us that I can change P to be a random matrix,

since from the adversary's perspective an honest public key generated in this fashion is indistinguishable from a random matrix. But when P is totally random, the plaintext is completely hidden. If we have truly random P, then the ciphertext has an additive PR for a random $R \in \{0,1\}^{m \times m}$, which is indistinguishable from a random matrix T. R basically takes a random subset sum of columns of P to get each column of T. For a random matrix P, assuming its wide enough, taking a random subset sum will give a uniformly random column (and thus we get a matrix very close to a uniformly random matrix). So the ciphertexts look like $T + xG$, and the random T makes this ciphertext look like a random matrix.

1.3 Number of Operations

How many operations can this scheme handle? We need to keep plaintexts in $\{0,1\}$, so we'll have to modify the addition operation (multiplication is still okay). Note that for $x_0, x_1 \in \{0,1\}$, we can write $x_0 + x_1 \bmod 2 = x_0 + x_1 - 2x_0x_1$. So addition becomes $C_0 + C_1 - 2C_0G^{-1}(C_1)$.

Let's see how the noise accumulates. When we multiply, the R vector goes from being binary to being order m. After another multiplication we get to order m^2. So when we try to evaluate a depth d circuit homomorphically, the R matrix is on approximately on the order of m^d. The addition of x_0R_1 contributes some noise, but the dominating factor is the $R_0G^{-1}(C_1)$ term. We need $m^d \leq q$ for depth d computations.

Recall the motivating factor of fully homomorphic encryption, where I want to outsource computations to the cloud. Suppose I know the depth of my computation. I just set up the scheme with q sufficiently large. Note that exponential q is large, since the operations are logarithmic in q (we just don't want to have q be doubly exponential). So I need to have an a priori polynomial depth bound, and if the computations end up going beyond that, the noise will be too high.

1.4 Bootstrapping

This is a neat idea that allows us to take an FHE scheme that can handle some number of operations and turn it into one that can handle an arbitrary number of operations.

We are given a ciphertext C encrypting some message x. Let's suppose the noise is at the point where we can't perform more operations. We want some way to "refresh" C to get low noise.

Let's suppose we're in the setting the client is outsourcing their computation to the cloud. An easy interactive solution would be to send C back to the client and ask the client to decrypt and re-encrypt. The better idea would be to have the cloud

perform this procedure via the homomorphic operations (and avoid interacting with the client).

We include in the public key an encryption of the secret key. So we give $C_{sk} = \mathsf{Enc}(pk, sk)$.

Given C that I'm trying to refresh, I construct a circuit D_C with C hard-coded where $D_C(sk) = \mathsf{Dec}(sk, C)$. Now we homomorphically evaluate D_C on C_{sk}. Under the hood, I'm evaluating D_C on the plaintext hidden in C_{sk}, which is sk. So the result is $\mathsf{Enc}(pk, D_C(sk)) = \mathsf{Enc}(pk, x)$ where x is the plaintext encoded by C.

As long as the scheme can handle operations as complicated as the decryption procedure, this idea will work. To verify this, observe that decryption is just computing an inner product between a vector and a matrix. But we only cared about one component of the resulting vector, so really it's an inner product between two vectors. Then there's a rounding step.

For an inner product, the depth of the computation is small. An inner product is a pointwise multiplication of two vectors. The pointwise multiplication takes one layer, and addition can be done in $\log m$ layers by setting up a binary tree structure for the additions. If I want to handle vectors in $\mathbb{Z}_q$, I have to replace the binary additions and multiplications with $polyloglog(q)$ depth circuits. So the depth will be $(\log m)(polyloglogq)$. Rounding takes $polylogq$ depth. So overall the binary depth is $(\log m)(polylogq)$. So we need $m^d \leq q$. We compute

$$\begin{aligned} m^d &= m^{(\log m)(polyloglogq)} \\ &= 2^{\log^2 mpolyloglogq} \\ &= (\log q)^{log^2 mpolylogq} \leq q \end{aligned}$$

1.5 Security

Remember that in regular CPA security experiment, there's no way to obtain an encryption of the secret key, since that would rely on the adversary actually querying on the secret key.

What we need is circular security, which guarantees security even if you're given an encryption of sk. We know that there exist schemes with circular security based on LWE. We can also construct schemes without circular security under LWE. So we can't make schemes like "all schemes based on LWE are circularly secure".

What we want is an FHE scheme with circular security based on LWE, but this is still open. So if you want FHE that can handle arbitrarily many computations, we have to make an assumption of circular security.