University of Illinois, Urbana Champaign
CS/ECE 498AC3/4 Applied Cryptography

Instructor: Dakshita Khurana
Scribe: Riccardo Paccagnella
Date: November 11, 2020

LECTURE
24

Learning with Errors 1

So far in this course we studied several cryptosystems whose security depends on the difficulty of certain number theoretic problems. All the assumptions we looked at (including, e.g., the RSA assumption, the DDH assumption, and the CDH assumption) are known to be broken by quantum computers. In this lecture we will cover a new cryptosystem based on a new assumption, called the learning with errors assumption, that is believed to hold even in the presence of quantum computers. We will first describe a problem that we believe to be quantum-hard, and then we will show how to build cryptosystems out of that.

24.1 Learning With Errors Problem

Consider a system of linear equations (all linearly independent):

$$\begin{aligned}
14s_1 + 5s_2 + 10s_3 + 2s_4 &= 8 \pmod{17} \\
13s_1 + 14s_2 + 14s_3 + 6s_4 &= 16 \pmod{17} \\
6s_1 + 10s_2 + 13s_3 + 15s_4 &= 3 \pmod{17} \\
6s_1 + 7s_2 + 16s_3 + 25s_4 &= 3 \pmod{17}
\end{aligned}$$

Normally, it is easy to solve these equations using Gaussian elimination to find s_1, s_2, s_3 and s_4. We will now try to make this system of equations harder to solve.

One way to make this system of equations harder to solve is to make it approximate. That is, instead of using exact equations we will use approximate equations. These approximate equations come with small error terms. For example, let $e_i \in \{-1, 0, 1\}$ for $i \in [1, 4]$.

$$\begin{aligned}
14s_1 + 5s_2 + 10s_3 + 2s_4 + e_1 &= 8 \pmod{17} \\
13s_1 + 14s_2 + 14s_3 + 6s_4 + e_2 &= 16 \pmod{17} \\
6s_1 + 10s_2 + 13s_3 + 15s_4 + e_3 &= 3 \pmod{17} \\
6s_1 + 7s_2 + 16s_3 + 25s_4 + e_4 &= 3 \pmod{17}
\end{aligned}$$

What can we do to solve a system of approximate equations? We can guess the errors e_1, e_2, e_3 and e_4. If our guess is right, this algorithm would work in polynomial time. However,

how many guesses do we need to try to find the solution? In this simple case, the number of possible assignments that we would have to iterate over would be 3^4.

However, consider now an example with $n = 256$ equations, 256 variables and 256 small error terms e_i defined as above. How many possible assignments to the set $\{e_1, e_2, ..., e_{256}\}$ would we need to try? The number of possible assignments that we would have to iterate over would be 3^{256}. Informally, we can already see that our strategy of solving a system of approximate linear equations by guessing errors and using Gaussian elimination will not work well if we have sufficiently many equations.

24.2 Learning With Errors Assumption

Consider again the system of approximate (also called "noisy") equations we described above. We discussed that Gaussian elimination does not work well in solving such system if we have sufficiently many equations (n is large). There has been a large body of research in mathematics and theoretical computer science that showed that this learning with errors problem is hard to solve for polynomial time adversaries in general (not just using Gaussian elimination). As a consequence, the learning with errors problems has been used to formulate an explicit assumption that a lot of recent cryptography has started relying on.

Let the coefficients of each equation be represented as an array $\vec{a_i}$ for $i \in [1, n]$ and the variables be represented as an array $\vec{s} = (s_1, ..., s_n)$. Informally, the "Search LWE Assumption" assumption says that for the "right parameters", given $\vec{a_1}, ..., \vec{a_n}$ and a set of equations of the form $\vec{a_i} \cdot \vec{s} + e_i$ it is hard to for PPT machines to find $\vec{s}$.

LWE Attack Game As usual, to define this assumption more formally, we construct an attack game based on the learning with errors problem. The attack game works as follows:

1. Sample a random prime number q (n bits long with large n, such as 256).
2. Sample array $\vec{s} = (s_1, ..., s_n)$ where each $s_i \leftarrow \mathbb{Z}_q$. Note that $(s_1, ..., s_n)$ are the system variables (the ones that the adversary will try to derive).
3. Set $m = n^2$, where m is the number of equations. Observe that we are constructing a system with more equations than variables.
4. For every $i \in [m]$, sample $\vec{a_i} = (a_{i1}, a_{i2}, ..., a_{in})$ where each $a_{ij} \leftarrow \mathbb{Z}_q$. Note that $(a_{i1}, a_{i2}, ..., a_{in})$ for each i are the coefficients of each equation, which will be public.
5. For every $i \in [m]$, sample $e_i \leftarrow \mathcal{X}$, where $\mathcal{X}$ is an error distribution that we will define below. Note that $(e_1, e_2, ..., e_m)$ for each i are the small error terms of each equation, which will be private.
6. Compute $b_i = \vec{a_i} \cdot \vec{s} + e_i \pmod{\mathbb{Z}_q}$. These are the right-hand side scalar values in the equations of the examples above.
7. Output vector $(\vec{a_1}, ..., \vec{a_m})$, $(b_1, ..., b_m)$.

Search LWE Assumption (SLWE) Formally, the "Search LWE Assumption" says that for every PPT adversary A, the probability of finding $\vec{s} = (s_1, ..., s_n)$ given $(\vec{a_1}, ..., \vec{a_m})$, $(b_1, ..., b_n)$ (computed as in the game defined above) is negligible.

Decision LWE Assumption (DLWE) We can also define a "Decision LWE Assumption" stating that the probability that a PPT adversary can distinguish between an instance where both $(\vec{a_1}, ..., \vec{a_m})$, $(b_1, ..., b_m)$ are sampled as in the game above and an instance where $(\vec{a_1}, ..., \vec{a_m})$ is sampled as above but $(b_1, ..., b_m)$ is sampled at random is negligible.

Additional Facts There are two important facts on the two definitions above. The first fact is that for the parameters that we care about (up to a certain renaming of the parameters), the two assumptions are equivalent (they imply each other). This is a well known fact, easy to prove one way but hard to prove the other way. From this point on we will only care about the DLWE assumption because it is easier to build cryptography using the DLWE. The second fact is that these two assumptions are not hard for all choices of q, n, m and $\mathcal{X}$. For example if $\mathcal{X}$ always outputs zero errors, then the LWE problem becomes easy. Likewise, if q is too small, the LWE problem becomes easy as well. For our attack game, we usually assume the error distribution $\mathcal{X}$ to be a discrete Gaussian over $\mathbb{Z}_q$. Although we will not dive into it, note that this is a well studied topic in the literature.

24.3 Building Crypto From DLWE

We saw above that finding the array $\vec{s}$ is hard given $(\vec{a_i}, \vec{a_i} \cdot \vec{s} + e_i)$. We also saw that $b_i = \vec{a_i} \cdot \vec{s} + e_i$ is indistinguishable from a random $b_i \leftarrow \mathbb{Z}_q$ for a PPT adversary. We can use the latter to construct a (randomized) symmetric encryption scheme as follows:

1. $KeyGen$: sample symmetric key $\vec{s} = (s_1, ..., s_n)$ where each $s_i \leftarrow \mathbb{Z}_q$. Note that we want to use this symmetric key to encrypt an unbounded number of messages.

2. $Enc(m, \vec{s}, r)$, where m is a single-bit message (either 0 or 1) that we want to encrypt, $\vec{s}$ is the symmetric key and r is some fresh randomness: using r, sample the coefficients $\vec{a} = (a_1, ..., a_n)$ and a small error term e, where we know that with high probability (since e is usually sampled from a discrete Gaussian distribution) $e \in [\lfloor -q/4 \rfloor, \lfloor q/4 \rfloor]$; output ciphertext $ct = (\vec{a},\ m \cdot \lfloor q/2 \rfloor + b \pmod q)$ where $b = \vec{a} \cdot \vec{s} + e \pmod q$. Informally, this gives us a way to encrypt a message by adding to it a mask that is indistinguishable from random to an adversary (due to the DLWE assumption).

3. $Dec(ct, \vec{s})$, where ct is the ciphertext and $\vec{s}$ is the symmetric key: parse $ct = (\vec{a}, y)$; compute $\vec{a} \cdot \vec{s}$; compute $y - \vec{a} \cdot \vec{s}$, which is a "noisy" decryption of m (within a small error e of $y - b$); notice that:

   - if $m = 0$, then we have that $y = b = \vec{a} \cdot \vec{s} + e \pmod q$, which implies that $y - \vec{a} \cdot \vec{s} = e$, meaning that y will lie between $\lfloor -q/4 \rfloor$ and $\lfloor q/4 \rfloor$.
   - if $m = 1$, then we have that $y = \lfloor q/2 \rfloor + b = \lfloor q/2 \rfloor + \vec{a} \cdot \vec{s} + e \pmod q$, which implies that $y - \vec{a} \cdot \vec{s} = \lfloor q/2 \rfloor + e$, meaning that y will lie between $\lfloor q/4 \rfloor$ and $\lfloor 3q/4 \rfloor$.

   That means that to decrypt ct we can compute $y - \vec{a} \cdot \vec{s}$. If $y - \vec{a} \cdot \vec{s}$ lies between $\lfloor -q/4 \rfloor$ and $\lfloor q/4 \rfloor$, then we know that with high probability $m = 0$. Otherwise, if $y - \vec{a} \cdot \vec{s}$ lies between $\lfloor q/4 \rfloor$ and $\lfloor 3q/4 \rfloor$ then we know that with high probability $m = 1$.

It is important to note that this algorithm works because we know that the small error term e is with high probability between $\lfloor -q/4 \rfloor$ and $\lfloor q/4 \rfloor$.

Acknowledgement

These scribe notes were prepared by editing a light modification of the template designed by Alexander Sherstov.