University of Illinois, Urbana Champaign
CS 598DK Special Topics in Cryptography

Instructor: Dakshita Khurana
Scribe: Luke Myers, Ruta Jawale
Date: October 10, 2019

LECTURE
9

Review of ZK, and Learning with Errors

In the most recent lecture, we covered zero-knowledge proofs and non-interactive zero-knowledge proofs which we will review at the beginning of this lecture.The rest of the lecture will cover the learning with errors (LWE) problem including the search LWE problem and the decisional LWE problem. Finally, a construction for a single message private key encryption scheme using the decisional LWE problem is shown.

9.1 Review of Zero Knowledge

A zero-knowledge protocol must satisfy completeness, soundness and zero-knowledge.

9.1.1 Soundness

DEFINITION 9.1. Soundness: if $x \notin L$ then $\Pr[\mathsf{output}_{\mathsf{V}}(\langle \mathsf{P}(x, w), \mathsf{V}(x) \rangle)) = \mathsf{reject}] = 1 - negl(k)$.

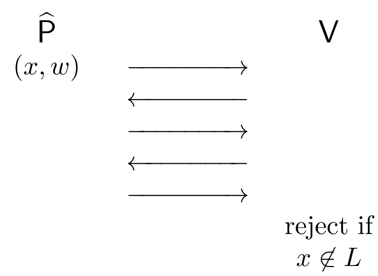

FIGURE 9.1:

REMARK 9.2. Proof of Knowledge: if V accepts then P "knows a witness for x". This is an informal definition for proof of knowledge where there exists a machine, that need not be the verifier, which can interact with a prover P that is able to convince the verifier and extract a witness w for x from the prover P. This informal definition captures the concept of proving the possession of a witness. Proof of knowledge implies soundness.

The graph 3-coloring problem which we covered in the last lecture was an example of a zero-knowledge proof of knowledge. Where the prover was proving knowledge of a valid 3-coloring without revealing the entire graph. We do not discuss details of how the proof of knowledge machine would extract such a coloring (witness) from the prover.

The soundness property is a property of security for a verifier that is interacting with a dishonest prover, as seen in figure. 9.1. This cheating prover should not be able to convince the verifier to accept if $x \notin L$.

9.1.2 Zero-Knowledge

Recall that in zero knowledge a malicious verifier $\widehat{\mathsf{V}}$ should not be able to gain any additional information, such as the witness w, from a prover P following the protocol. As seen in figure 9.2, a run of the simulator Sim which only has access to x must be computationally equivalent to a run of the protocol with a real prover that has as input both the value x and a witness w.

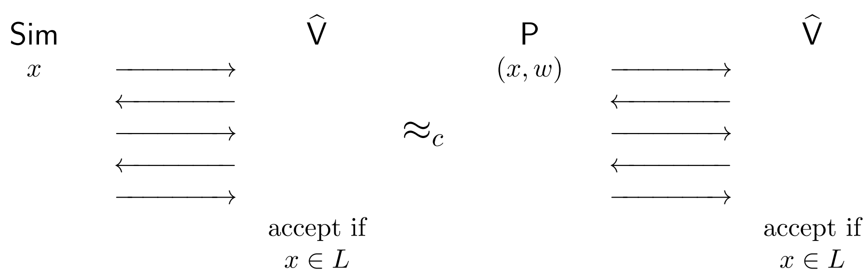

FIGURE 9.2: Zero-knowledge Property

The figure above implies that a simulator Sim given a value $x \in L$, without the witness w, must convince the verifier that $x \in L$, as seen in figure 9.3 (as otherwise the verifier can distinguish the two distributions in 9.2).

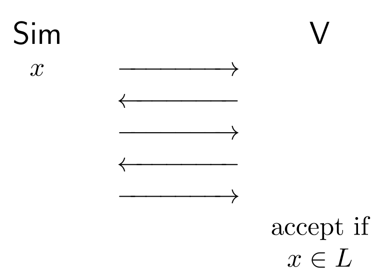

FIGURE 9.3:

EXAMPLE 9.3. Consider the scenario in figure 9.3. Clearly if we give the simulator Sim $x \in L$ then the verifier V should accept. Suppose we give the simulator $x \notin L$ what should be the output of the verifier? The verifier should accept. If the output of the verifier were to be reject, then since the simulator is also a probabilistic polynomial-time Turing machine, we would be able to build a polynomial time algorithm (consisting of the simulator and verifier together) that can determine membership in L which contradicts the NP-hardness of the problem.

9.1.3 Review of Simulator and Prover Comparison

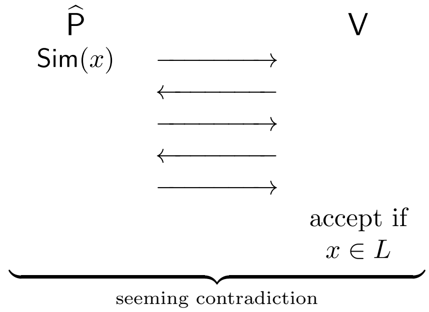

FIGURE 9.4:

There is a seeming contradiction, depicted in figure 9.4, where a dishonest prover P is able to convince the verifier V to accept x without knowing a witness w even if $x \notin L$ by simply passing x as input to the simulator code, as described in example 9.3. The resolution to the seeming contradiction is to realize that the simulator has additional capabilities that the prover does not. These capabilities are as follows:

- The ability to rewind the verifier (even if the verifier is malicious). After receiving any number of messages from the verifier, the simulator can transparently force the verifier to return back and resume the protocol at a previous step where the simulator will proceed with the knowledge of what the verifier will send next.
- A priori knowledge of all queries the verifier with make (in the honest verifier model). Since the verifier must follow the protocol and generate the queries according to the specified distribution the simulator is allowed to know ahead of time what the queries will be.

9.2 Non-interactive Zero Knowledge

In non-interactive zero knowledge, as seen in figure 9.5, the prover P sends a single message to the verifier V to convince the verfifier that $x \in L$ is true. The verifier does not query the prover. This is the most general form of non-interactive zero knowledge and it is not possible in the *plain model*. It is impossible to construct a simulator because the simulator's additional capabilities (rewinding the verifier, and a priori knowledge of queries) that allow it to convince the verifier are not applicable in the non-interactive settings, as there are no messages sent by the verifier to rewind and no queries to have advanced knowledge of. The simulator is just no more powerful than the prover and thus the prover could simply run the simulator to make the verifier accept even without w and when $x \notin L$, breaking soundness. Therefore, non-interactive zero-knowledge cannot exist unless we somehow find a way to give the simulator more power. In the following, we discuss two ways to give the simulator more power.

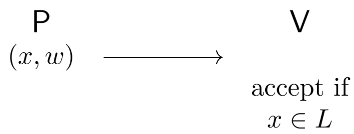

FIGURE 9.5:

In the common reference string model, as depicted in figure 9.6 we make an additional trust assumption that a string is generated by an oracle, trusted by both the prover and verifier, before the protocol begins. Both the prover and the verifier are able to arbitrarily query the common reference string and do local computation. The common reference string is assumed to be generated honestly according to some distribution specified in the protocol. The extra power that the simulator has in this model, is the ability to control what (secret) randomness is used to generate the common reference string.

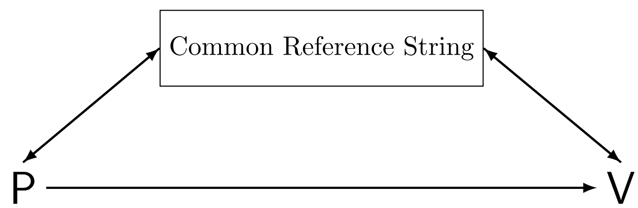

FIGURE 9.6:

In the random oracle model, as depicted in figure 9.7, we make an additional trust assumption that an oracle exists which gives only random-looking strings. The prover P and verifier V are able to communicate with the oracle to arbitrarily query it for random information which they can do local computation on. The extra power that the simulator has in this model, is the ability to see what queries the adversary makes to the random oracle, and to control the output that the random oracle sends to the adversary.

REMARK 9.4. The random oracle model is realized in practice by assuming the output of a secure hash function, e.g. SHA-256, behaves like the output of a random oracle.

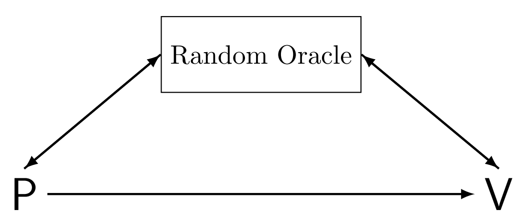

FIGURE 9.7:

REMARK 9.5. The different adversarial models of the verifier, honest, honest but curious, and malicious, no longer make sense as the verifier still sends no messages to the prover and can only get information according to the distribution specified in the protocol from the common reference string or random information from the oracle.

9.3 Learning With Errors Problem

Consider the following system of linear equations:

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

The solution set can be found using Gaussian elimination. However, if an independent error term is added to each equation and instead of saying the equation strictly equals a number we say it approximately equals a number the problem becomes much harder. Consider the following alternate system of equations:

$$\begin{aligned}
14s_1 + 15s_2 + 5s_3 + 2s_4 + e_1 &\approx 8 \bmod 17 \\
13s_1 + 14s_2 + 14s_3 + 6s_4 + e_2 &\approx 16 \bmod 17 \\
6s_1 + 10s_2 + 13s_3 + 15s_4 + e_3 &\approx 3 \bmod 17 \\
6s_1 + 7s_2 + 16s_3 + 25s_4 + e_4 &\approx 3 \bmod 17 \\
\text{where } e_i \in \{-1, 0, 1\}
\end{aligned}$$

To solve this new system of equations take the set of systems of equations that comes from creating a new system of equations from every possible permutation of $e_1, e_2, ...e_m$ and perform Gaussian elimination on each system of equations in the set until a solution is determined. The number of possible permutations grows exponentially in the size of the error and in the number of equations, m. When m is polynomial in the security parameter, there are exponentially many possibilities, and therefore there is no longer an efficient algorithm.

Remark 9.6. It is believed that the system of equations can be over-determined, i.e. have more equations than the number of s-variables, while remaining hard.

Remark 9.7. The domain of e is purposely kept small because cryptosystems built off of the learning with errors problem can only tolerate a certain amount of error before violating correctness.

9.4 Search LWE Assumption

It is assumed that solving the LWE set of equations is difficult for a non-uniform probabilistic polynomial-time machine. We can formalize this statement as a search assumption where we ask for an adversary to find any solution $\vec{s}^*$ to the set of equations of the form $\langle \vec{a}_i, \vec{s} \rangle + e_i$ given $\vec{a}_i$.

Importantly, we first sample a large prime q, sample an n element vector $\vec{s}$ for $n = O(\log q)$ where each element in $\vec{s}$ is an integer modulo q, set $m = O(n \log q)$, and for every

$i \in [m]$, sample $\vec{a}_i$ as an n-element array of integers modulo q. Finally, we define an error distribution $\mathcal{X}$, such that every element of the error vector $\vec{e}$ is sampled from $\mathcal{X}$.
We would then write the Search LWE assumption as follows:
$\mathsf{Search} - \mathsf{LWE}_{n,m,q,\mathcal{X}}$: $\forall$ non-uniform PPT adversaries $\mathcal{A}$

$$\Pr_{\vec{a}_i \leftarrow \mathbb{Z}_q^n, \vec{s} \leftarrow \mathbb{Z}_q^n, \vec{e} \leftarrow \mathcal{X}^m}[\mathcal{A}(\vec{a}_1, \langle \vec{a}_1, \vec{s} \rangle + e_1, \ldots, \vec{a}_m, \langle \vec{a}_m, \vec{s} \rangle + e_m) \to \vec{s}^*] = \text{negl}(n)$$

where we have that $\exists \vec{e}^*$ such that $(\vec{s}^*, \vec{e}^*)$ would satisfy the same set of equations.

9.5 Decisional LWE Assumption

It is also assumed to be difficult for any non-uniform probabilistic polynomial-time adversary to distinguish between the output of an equation $\langle \vec{a}_i, \vec{s} \rangle + e_i$ and and a randomly sampled element b_i. This can be formally stated as follows:
$\mathsf{Decision} - \mathsf{LWE}_{n,m,q,\mathcal{X}}$: $\forall$ non-uniform PPT adversaries $\mathcal{A}$

$$\left| \Pr_{\substack{\vec{a}_i \leftarrow \mathbb{Z}_q^n, \vec{s} \leftarrow \mathbb{Z}_q^n, \\ \vec{e} \leftarrow \mathcal{X}^m, b_i = \langle \vec{a}_i, \vec{s} \rangle + e_i}}[\mathcal{A}(\vec{a}_1, b_1, \ldots, \vec{a}_m, b_m) = 1] - \Pr_{\substack{\vec{a} \leftarrow \mathbb{Z}_q^n \\ b_i \leftarrow \mathbb{Z}_q}}[\mathcal{A}(\vec{a}_1, b_1, \ldots, \vec{a}_m, b_m) = 1] \right| = \text{negl}(n)$$

A notationally simpler rewriting of the same statement would be to construct a matrix A of our vectors $\vec{a}_i$ and vectors for the error e and the secret $\vec{s}$:

$$A = \begin{bmatrix} \vec{a}_1 & \vec{a}_2 & \ldots & \vec{a}_m \end{bmatrix}_{n \times m}$$
$$\vec{e} = \begin{bmatrix} e_1 & \ldots & e_m \end{bmatrix}_{1 \times m}$$
$$\vec{s} = \begin{bmatrix} s_1 & \ldots & s_n \end{bmatrix}_{1 \times n}$$

$$(A, \vec{s}A + \vec{e}) \approx_c (A, \mathsf{uniform}_{1 \times m})$$

where $\approx_c$ indicates that the two distributions are computationally indistinguishable.

Remark 9.8. As it turns out, Decisional LWE and Search LWE reduce to each other in the parameter ranges that are of interest to us, which we write as Decisional LWE $\iff$ Search LWE. Looking ahead, we will almost exclusively rely on the Decisional LWE assumption, which we will simply refer to as the LWE assumption.

Remark 9.9. Note that the Decisional and Search LWE assumptions are not true for all $q, m, n, \mathcal{X}$. In fact, when $\mathcal{X}$ is the distribution that always outputs 0, then these assumptions are clearly false (since the equations can be solved via Gaussian elimination). This indicates that the error distribution must be "large enough" to introduce sufficient uncertainty into the system and make the equations indistinguishable. Also, looking ahead, large errors will make it difficult to guarantee correctness of systems built out of LWE, and therefore errors must be "small enough" to not affect correctness. We will discuss the distribution $\mathcal{X}$ in more detail in the next lecture.

9.6 LWE Based Symmetric Key Encryption

In a previous lecture when we introduced the Decisional Diffie-Hellman assumption, we talked about how to construct a secure symmetric key encryption scheme directly from this hardness assumption, namely El-Gamal Encryption. In a similar manner, we'll now construct a symmetric key encryption based on the LWE assumption.

Let's start brainstorming one-bit symmetric key encryption schemes based off of the One-Time Pad (OTP) idea. We split up the OTP's idea into two parts. There is a key which needs to be kept secret and there is a source of randomness used to hide the message. In OTP, they are the same, but here we have two parts to the LWE assumption.

By the Search LWE assumption, we know that it is difficult for any non-uniform PPT adversary to find $\vec{s}$ given $\vec{a}$ and $\langle \vec{a}, \vec{s} \rangle + e$. This hiding property is important for a encryption key. As such, we reason that we can use $\vec{s}$ as a secret key.

By the Decisional LWE assumption, we have that $\langle \vec{a}, \vec{s} \rangle + e$ is indistinguishable from a random. Previously in OTP, we xor-ed randomness with our message. In this case, since we are dealing with modular integers with respect to a prime q, $\mathbb{Z}_q$, we will use addition (modulo q) in place of XOR. Our first idea for a scheme will be the following:

KeyGen$(1^n) \to \vec{s}$:

Sample secret key $\vec{s} \leftarrow \mathbb{Z}_q^n$

Encrypt$(1^n, m; r) \to (c, \vec{a})$:

Sample randomness $\vec{a} \leftarrow \mathbb{Z}_q^n$ and error $e \leftarrow \mathcal{X}$

Set $c = \langle \vec{a}, \vec{s} \rangle + e + m \pmod{q}$

Let m be a one-bit message that we wish to encrypt using this scheme. We sample our secret key $\vec{s}$ from our modular space $\mathbb{Z}_q^n$ during the **KeyGen** function and output it. Later in the **Encrypt** function, we will sample the adversarially known randomness $\vec{a}$ known from $\mathbb{Z}_q^n$ and the adversarially unknown randomness or error e from some fixed distribution $\mathcal{X}$. We then construct our randomness meant to hide the message $\langle \vec{a}, \vec{s} \rangle + e$ and add it to our message m and output $\vec{a}$ and our ciphertext c.

However, we soon run into a problem. Our **Decrypt** function cannot decrypt the ciphertext c given $\vec{s}$ and $\vec{a}$ alone. There is the error term e to contend with. In order to fix our idea, we can attempt to pass the error e to the **Decrypt** function in two different ways: include e as part of the secret key outputted from **KeyGen** or include e as part of the ciphertext outputted from **Encrypt**. We consider the first strategy:

KeyGen$(1^n) \to \vec{s}$:

Sample secret key $\vec{s} \leftarrow \mathbb{Z}_q^n$

Encrypt$(\vec{s}, m; r) \to (c, \vec{a}, e)$:

Sample randomness $\vec{a} \leftarrow \mathbb{Z}_q^n$ and error $e \leftarrow \mathcal{X}$

Set $c = \langle \vec{a}, \vec{s} \rangle + e + m \pmod{q}$

Decrypt$(\vec{s}, (c, \vec{a}, e)) \to m$:

Set $m = c - \langle \vec{a}, \vec{s} \rangle - e \pmod q$

We sample the terms $\vec{s}$, $\vec{a}$, and e in the same fashion as before. This time we provide the error e as output of our **Encrypt** function. However, this scheme is certainly not secure. The power of the LWE assumption relied heavily on the property that e was kept hidden from the adversary. We can then try the second technique to send the error e to the **Decrypt** function:

KeyGen$(1^n) \to (\vec{s}, e)$:

Sample secret key $\vec{s} \leftarrow \mathbb{Z}_q^n$ and error $e \leftarrow \mathcal{X}$

Encrypt$(\vec{s}, m; r) \to (c, \vec{a})$:

Sample randomness $\vec{a} \leftarrow \mathbb{Z}_q^n$

Set $c = \langle \vec{a}, \vec{s} \rangle + e + m \pmod q$

Decrypt$((\vec{s}, e), (c, \vec{a})) \to m$:

Set $m = c - \langle \vec{a}, \vec{s} \rangle - e \pmod q$

We sample the terms $\vec{s}$, $\vec{a}$, and e and provide the error e as output of our **KeyGen** algorithm. However, we still have a problem, since now the error is fixed for multiple messages. The adversary can just add this as an additional unknown. The reparameterization would be an extension of $\vec{s}$ and $\vec{a}$ as follows:

$$\vec{s}^* = [\vec{s}, e]$$

$$\vec{a}^* = [\vec{a}, 1]$$

where we can then represent the problem as solving for the fixed value $\vec{s}^*$ in linear equations of the form $\langle \vec{s}^*, \vec{a}^* \rangle$. Then the adversary can solve these equations using Gaussian elimination.

As these are the only two ways to pass the error e to the **Decrypt** algorithm and neither technique is secure, we conclude that we cannot simply pass the error to the **Decrypt** function. In order to deal with this, we decide to make use of the fact that the LWE assumption is believed to be true even when the error e is quite small. If we can assume with high probability that $\|e\| \leq \frac{q}{4}$, then we can amplify the relative amplitude of our message with respect to the error in our output ciphertext. We do the following:

KeyGen$(1^n) \to \vec{s}$:

Sample secret key $\vec{s} \leftarrow \mathbb{Z}_q^n$ and error $e \leftarrow \mathcal{X}$

Encrypt$(\vec{s}, m; r) \to (c, \vec{a})$:

Sample randomness $\vec{a} \leftarrow \mathbb{Z}_q^n$

Set $c = \langle \vec{a}, \vec{s} \rangle + e + m \cdot \left\lfloor \frac{q}{2} \right\rfloor \pmod q$

Decrypt$(\vec{s}, (c, \vec{a})) \to m$:

Set $\alpha = c - \langle \vec{a}, \vec{s} \rangle \pmod q$. Set $m = \alpha / (\lfloor \frac{q}{2} \rfloor)$.

Our method of amplifying is to multiply our message m by a term that is larger than the magnitude of our error e, which we set as $\lfloor \frac{q}{2} \rfloor$. In terms of correctness, since the error is so small, the decrypt function will succeed with high probability. We will see the proof of security next lecture.

Remark 9.10. This encryption technique extends quite naturally to multi-bit message.

Acknowledgement

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