Scribe Notes for Lecture on 10/15/2020

By: Mihir Rajpal (From Lecture by Professor Dakshita Khurana)

(a) **Chaum-Pederson Protocol:**

This Zero-Knowledge Protocol is designed to allow one party to prove to another that a tuple of the form (g^a, g^b, g^f) where g is the generator of the group $\mathbb{Z}_p^*$ and $g^a, g^b, g^f \in \mathbb{Z}_p^*$ that the third element in the tuple, g^f, is equal to g^{ab} where $g^{ab} \in \mathbb{Z}_p^*$, making the tuple (g^a, g^b, g^f) a DH (Diffie-Hellman) tuple.

In this Zero-Knowledge Proof Protocol there are two parties, the Prover (P) who is trying to prove that the tuple is a DH tuple, and the Verifier (V) who is trying to ensure that the tuple is a DH tuple. The prover is also trying to ensure through this process that the Verifier cannot learn what a or b are without solving the discrete logarithm problem or the computational Diffie-Hellman (CDH) problem.

The procedure starts with a tuple of the form (g^a, g^b, g^f), denoted as (u, v, w) without loss of generality, and continues as follows:

1. P samples a random $d \in \mathbb{Z}_p$ and generates $v' = g^d$ and $w' = g^{ad} = u^d$.
2. P then sends the tuple (v', w') to V.
3. V samples a random $c \in \mathbb{Z}_p$.
4. V then sends c to P.
5. P computes e as $d + bc$.
6. P then sends e to V.
7. V first checks that e is of the correct form by ensuring that $g^e = v' \cdot (v)^c$, which is mathematically equivalent to $g^d \cdot (g^b)^c = g^d \cdot g^{bc} = g^{d+bc}$ if both sides are behaving honestly.
8. V then checks that $w = g^{ab}$ by ensuring that $u^e = w' \cdot (w)^c$, whose left hand side is mathematically equivalent to $(g^a)^e = g^{ae} = g^{a(d+bc)} = g^{ad+abc}$ and whose right hand side is mathematically equivalent to $u^d \cdot (u^b)^c = u^d \cdot u^{bc} = (g^a)^d \cdot (g^a)^{bc} = g^{ad} \cdot g^{abc} = g^{ad+abc}$ if both sides are behaving honestly.

Next, we prove the zero-knowledge property of this protocol.

We do this with a Simulator (S) that has no knowledge of whether or not the statement it is trying to prove is true and does not have the necessary data to prove the statement in question, but has the unique ability to rerun parts of the process (or the whole process) as many times as it likes until it is able to produce an output that the Verifier accepts. This is different from the Prover, who must know that the statement is true and must have the necessary data to prove the truth of the claim, and cannot rerun any part of the Verifier even once.

The procedure for this zero-knowledge proof starts with a tuple of the form (g^a, g^b, g^c), denoted as (u, v, w) without loss of generality, (which is the same as above) and continues as follows:

1. S guesses the value c (defined in same way as protocol above) that V is going to send and samples a random $e \in \mathbb{Z}_p$.
2. S then computes $v' = g^e \cdot v^{-c} = g^e \cdot (g^b)^{-c} = g^e \cdot g^{-bc} = g^{\frac{e}{bc}}$ and $w' = u^e \cdot w^{-c} = (g^a)^e \cdot (g^f)^{-c} = g^{ae} \cdot g^{-fc} = g^{\frac{ae}{fc}}$.
3. S then sends the tuple (v', w') to V.
4. V samples a random $c \in \mathbb{Z}_p$.
5. V then sends c to S.
6. If the value of c sent by V matches S's initial guess, continue as normal. If they don't match, S returns V to the first step in this process.
7. S then sends e to V.
8. V first checks that e is of the correct form by ensuring that $g^e = v' \cdot (v)^c$, which is mathematically equivalent to $g^{\frac{e}{bc}} \cdot (g^b)^c = g^{\frac{e}{bc}} \cdot g^{bc} = g^e$ in this case.
9. V then checks w by ensuring that $u^e = w' \cdot (w)^c$, whose left hand side is mathematically equivalent to $(g^a)^e = g^{ae}$ and whose right hand side is mathematically equivalent to $g^{\frac{ae}{fc}} \cdot (g^f)^c = g^{\frac{ae}{fc}} \cdot g^{fc} = g^{ae}$ in this case.

Therefore, as far as the Verifier is concerned, both checks have passed even though neither the Simulator nor the Verifier knew anything about whether or not $f = ab$ or the values of a or b, proving that this proof is zero-knowledge.

Finally, we prove the soundness of this protocol by running the original protocol (with the Verifier) twice except for the fact that d is the same for both executions of the protocol and the two c values, denoted c_1 and c_2 for our purposes, are different. In this case the Verifier gets two different e values, similarly denoted e_1 and e_2 for our purposes. Since $e_1 = d + bc_1$ and $e_2 = d + bc_2$, the Verifier has two linear equations and two unknowns (d and b), so the Verifier can solve for both unknowns, including b. The fact that the Verifier can solve for b soley from the output of the Prover when the Verifier couldn't efficiently compute the value of b beforehand (without the Verifier's help) means that the Verifier must know b and therefore must know whether $f = ab$, which the Verifier could check by raising u to the power b and comparing the result of u^b with w.

This leads to a useful Zero-Knowledge Proof Protocol for proving that a tuple is, in fact, a DH tuple.

(b) **General Linear Relations on Exponents:**
We now extend the protocol described above such that it can be used for proving linear relations among exponents to generators without revealing the exponents themselves. This protocol is sufficiently similar to the Chaum-Pederson protocol that generalizations of the proofs described above are enough to build an intuitive sense that the protocol is correct, with a more concrete formal proof possible when the generality surrounding the predicate is removed and therefore the security of this construction does not hold for all predicates and the security of the construction relies on the security of the predicate.

The statement $\mathcal{R}$ we are trying to prove must be in the complexity class NP and is defined:

$$\mathcal{R} = \{\mathsf{Pred}, (u_1, u_2, \ldots, u_n) : \exists\ (a_1, a_2, \ldots, a_n) \mid u_i = \prod_{j=1}^{n} g_{i_j}^{a_j} \wedge \mathsf{Pred}(a_1, a_2, \ldots, a_n) = \mathsf{True}\}$$

The procedure starts with a tuple of the form $(u_1, u_2, \ldots, u_n)$ as well as a Predicate Pred and continues as follows:

1. P samples a random $(d_1, d_2, \ldots, d_n) \in \mathbb{Z}_p \mid \mathsf{Pred}(d_1, d_2, \ldots, d_n) = \mathsf{True}$.
2. P then computes $(u'_1, u'_2, \ldots, u'_n)$ using the formula $u'_i = \prod_{j=1}^{n} g_{i_j}^{d_j}$.
3. P then sends the tuple $(u'_1, u'_2, \ldots, u'_n)$ to V.
4. V samples a random $c \in \mathbb{Z}_p$.
5. V then sends c to P.
6. P computes $e_j\ \forall\ j \in [1, n]$ as $d_j + a_j c$.
7. P then sends $(e_1, e_2, \ldots, e_n)$ to V.
8. V first checks that $(e_1, e_2, \ldots, e_n)$ is of the correct form by ensuring that $\forall\ i \in [1, n]$, $\prod_{j=1}^{n} g_{i_j}^{e_j} = u'_i \cdot (u_i)^c$, which is mathematically equivalent to $\prod_{j=1}^{n} g_{i_j}^{d_j} \cdot \left(\prod_{j=1}^{n} g_{i_j}^{a_j}\right)^c = \prod_{j=1}^{n} g_{i_j}^{d_j} \cdot \prod_{j=1}^{n} g_{i_j}^{a_j c} = \prod_{j=1}^{n} g_{i_j}^{d_j + a_j c}$ if both sides are behaving honestly.
9. V then performs a second check that utilizes a modified form of the predicate where the original a values would satisfy the predicate if and only if the modified predicate would be satisfied with $(g^{a_1}, g^{a_2}, \ldots, g^{a_n})$ using a check that is implementation specific.

The general nature of the definition of the second check is what makes the proof and associated security for this construction entirely dependent on the nature of the predicate.

(c) **Equality of Ciphertexts:** This problem relied on a modified version of the El-Gamal Encryption system discussed in previous lectures where the Public Key (PK), the Secret Key (SK), and the Encryption Function ($\mathsf{Enc}_{\mathsf{PK}}$) are defined as follows:
$\mathsf{PK} = (g, h) = (g, g^a)$, $\mathsf{SK} = a$, and $\mathsf{Enc}_{\mathsf{PK}}(m; r) = (g^r, h^r \cdot m)$ for $a \in \mathbb{Z}_p$, r being randomly sampled from $\mathbb{Z}_p$, an arbitrary message m, and g is the generator for $\mathbb{Z}_p^*$.
The problem is how to construct a zero-knowledge proof that two ciphertexts $\mathsf{ct}_1 = (u_1, v_1)$ and $\mathsf{ct}_2 = (u_2, v_2)$, encrypted with two seperate public keys $\mathsf{PK}_1 = (g, h_1)$ and $\mathsf{PK}_2 = (g, h_2)$ respectively and two seperate random values r_1 and r_2 respectively, decrypt to the same value. This proplem was left as a homework problem, so providing a solution would be ethically wrong, but the hint was given that this problem is equivalent to proving that:
$\exists\ r_1, r_2 \mid u_1 = g^{r_1} \wedge u_2 = g^{r_2} \wedge v_1 \cdot (v_2)^{-1} = h_2^{r_2} \cdot (h_1^{r_1})^{-1}$.

(d) **AND Composition of Zero-Knowledge Protocols:** The AND compositions of two Zero-Knowledge Protocols (P_0, V_0) for R_0 and (P_1, V_1) for R_1 is as simple as running the two either in parallel or sequentially since both relations are satisfied in a Zero-Knowledge context when both

protocols run independently are satisfied.

(e) **OR Composition of Zero-Knowledge Protocols:** The OR composition of two Zero-Knowledge Protocols (P_0, V_0) for R_0 and (P_1, V_1) for R_1 where S_0 is the Simulator for R_0 and S_1 is the Simulator for R_1 is slightly more involved than the AND composition of Zero-Knowledge Protocols.
For this process, we assume the output of each simulator is of the form (a_i, c_i, z_i) where a_i is the first message from S for S_i, c_i is the first message from V for S_i, and z_i is the final message from S for S_i. We also asssume that the output of each protocol-specific prover with no arguments $(P_i())$ is of the form a_i and the output of each protocol-specific prover with one argument, c_i, $(P_i(c_i))$ is of the form z_i.
Then the Proof proceeds as follows in the case where R_0 is true and R_1 is false:

1. P gets the output (a_1, c_1, z_1) from S_1.
2. P then gets the output a_0 from $P_0()$.
3. P then sends a_0 and a_1 to V.
4. V samples a random bitstring c.
5. V then sends c to P.
6. P computes $c_0 = c \oplus c_1$.
7. P then gets the output z_0 from $P_0(c_0)$.
8. P then sends z_0, z_1, c_0, and c_1 to V.
9. V first checks that $c = c_0 \oplus c_1$ to ensure that at least one of the cases that V was trying to prove could only be simulated with a negligible probability due to the fact that a random bitstring XOR'd with a fixed value is still indistiguishable from random (one-time-pad property).
10. V then performs the checks that V_0 would have performed with a_0, c_0, and z_0.
11. V finally performs the checks that V_1 would have performed with a_1, c_1, and z_1.

In the case where R_0 is false and R_1 is true, the proof instead proceeds as follows:

1. P gets the output (a_0, c_0, z_0) from S_0.
2. P then gets the output a_1 from $P_1()$.
3. P then sends a_0 and a_1 to V.
4. V samples a random bitstring c.
5. V then sends c to P.
6. P computes $c_1 = c \oplus c_0$.
7. P then gets the output z_1 from $P_1(c_1)$.
8. P then sends z_0, z_1, c_0, and c_1 to V.
9. V first checks that $c = c_0 \oplus c_1$ to ensure that at least one of the cases that V was trying to prove could only be simulated with a negligible probability due to the fact that a random bitstring XOR'd with a fixed value is still indistiguishable from random (one-time-pad property).
10. V then performs the checks that V_0 would have performed with a_0, c_0, and z_0.

11. V finally performs the checks that V_1 would have performed with a_1, c_1, and z_1.

In the case where both statements are true (included purely for the sake of generality), P can either pretend one of the statements is false and proceed accorrding to the corresponding steps or follow one of sets of steps below.
Option A:

1. P gets the output a_0 from $P_0()$.
2. P next chooses a random c_0.
3. P then gets the output a_1 from $P_1()$.
4. P then sends a_0 and a_1 to V.
5. V samples a random bitstring c.
6. V then sends c to P.
7. P computes $c_1 = c \oplus c_0$.
8. P then gets the output z_0 from $P_0(c_0)$.
9. P next gets the output z_1 from $P_1(c_1)$.
10. P then sends z_0, z_1, c_0, and c_1 to V.
11. V first checks that $c = c_0 \oplus c_1$ to ensure that at least one of the cases that V was trying to prove could only be simulated with a negligible probability due to the fact that a random bitstring XOR'd with a fixed value is still indistiguishable from random (one-time-pad property).
12. V then performs the checks that V_0 would have performed with a_0, c_0, and z_0.
13. V finally performs the checks that V_1 would have performed with a_1, c_1, and z_1.

Option B:

1. P gets the output a_0 from $P_0()$.
2. P next chooses a random c_1.
3. P then gets the output a_1 from $P_1()$.
4. P then sends a_0 and a_1 to V.
5. V samples a random bitstring c.
6. V then sends c to P.
7. P computes $c_0 = c \oplus c_1$.
8. P then gets the output z_0 from $P_0(c_0)$.
9. P next gets the output z_1 from $P_1(c_1)$.
10. P then sends z_0, z_1, c_0, and c_1 to V.
11. V first checks that $c = c_0 \oplus c_1$ to ensure that at least one of the cases that V was trying to prove could only be simulated with a negligible probability due to the fact that a random bitstring XOR'd with a fixed value is still indistiguishable from random (one-time-pad property).
12. V then performs the checks that V_0 would have performed with a_0, c_0, and z_0.

13. V finally performs the checks that V_1 would have performed with a_1, c_1, and z_1.

These constructions are all valid because they ensure that the Verifier must genuinely prove one of the two statements or will have a high probability of being caught cheating.

(f) **Encrypting Bits:**

This problem is an example of a problem where the Verifier tries to prove that a value is within a certain range without revealing the value itself and is therefore a type of Range Proof. In this problem, we use the modified El-Gamal Encryption system where the Public Key (PK), the Secret Key (SK), and the Encryption Function ($\mathsf{Enc}_{\mathsf{PK}}$) are defined as follows:
$\mathsf{PK} = (g, h) = (g, g^a)$, $\mathsf{SK} = a$, and $\mathsf{Enc}_{\mathsf{PK}}(m; r) = (g^r, h^r \cdot m)$ for $a \in \mathbb{Z}_p$, r being randomly sampled from $\mathbb{Z}_p$, an arbitrary message m, and g is the generator for $\mathbb{Z}_p^*$.
The statement to prove is defined as:

$$\{(g, h, v, x) : \exists\ (b, r) \mid b \in [0, 1] \land (v, x) = \mathsf{Enc}_{\mathsf{PK}}(m; r) = (g^r, h^r \cdot m \land (g, h) = \mathsf{PK}\}$$

We also have that the ciphertext and public key (aka (g, h, v, x)) are publically available to both P and V. We start by realizing that that in the case where $b = 0$, we can use the Chaum-Pederson protocol as it is defined if we set $w = x$ and $u = h$. In the case where $b = 1$, however, we can use the Chaum-Pederson protocol as it is defined if we set $w = x \cdot g^{-1}$ and $u = h$. Since x is publically known, we need a way to prove that one of the two statements is true without revealing the exact value of b. Therefore, an OR-composition-based Zero-Knowledge Proof Structure is used to combine the two cases and complete the contruction of the Zero-Knowledge Proof for this problem.