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

Instructor: Dakshita Khurana
Scribe: Timur Javid
Date: October 20, 2020

LECTURE
17

Or Composition, Pairings

Last lecture, we looked at **range proofs**, in which the prover tries to show that a value is within a certain range without revealing the value itself. We constructed a simple example of a range proof where the prover wants to show that the message sent using the **ElGamal** scheme is an encryption of a single bit. In this example, we have a protocol to show that the prover sends a **Decisional Diffie-Hellman (DDH or DH) tuple** if the encrypted bit $b = 0$ and a protocol to show that the message is a DH tuple if $b = 1$. However, proving the DH tuple ultimately reveals the value of the b to the verifier. Thus, we have to OR compose these two protocols in order to prove the range of b without revealing the true value of b.

In this lecture, we construct a protocol for the OR composition that we started to look at in the last lecture. We also look at **non-interactive zero-knowledge proofs** in which we utilize a random oracle.

In the second half of the lecture, we start looking at **pairing based cryptography** in which we utilize groups in a **pairing** and their properties in order to obtain a different type of hardness assumption known as the **Co-CDH** assumption for our cryptographic operations. Lastly, we look at the **Boneh–Lynn–Shacham (BLS) signature** scheme which utilized **pairings** to sign and verify messages.

17.1 OR composition

In general, suppose that the prover wants to show that $\exists w$ such that $(x_0, w) \in R_0$ or $(x_1, w) \in R_1$ for witness w and statements x_0, x_1 without revealing which of the two statements x_0, x_1 is true. Here we show that if we have a protocol (P_0, V_0) for R_0 and protocol (P_1, V_1) for R_1, we can obtain a protocol for R_0 OR R_1 by running the protocols in parallel. We can assume, without loss of generality, that the prover knows that x_0 is true and x_1 is false. Because the prover knows a witness for x_0, it can answer all possible verifier challenges, so nothing special needs to be done for this protocol. However, for x_1, we allow the prover to "cheat" by letting the prover control the challenge for one of the two statements. The prover can only control one of the two challenges otherwise the prover could easily cheat the proof.

Essentially, the prover computes two zero-knowledge proofs in parallel and controls one challenge for a statement that is unknown to the verifier.

1. For statement x_0, the prover runs the proving algorithm as in the **Chaum-Pederson protocol** and sends the first message of the protocol to the verifier. In parallel for statement x_1, the prover runs a simulator that guesses the bit b_1 and sends the output of the simulator to the verifier.
2. The verifier sends the prover a random constraint c and b_0 as a challenge for x_0. The prover chooses a challenge b_1 for x_1 such that $b_0 \oplus b_1 = c$.
3. The prover responds to the challenges as normal. The verifier checks the proofs separately and also verifies that $c = b_0 \oplus b_1$.

The best a prover can do to cheat if they do not know how to prove either x_0 or x_1 is to guess both b_0 and b_1. If the prover guesses both b_0 and b_1, there is at least probability $\frac{1}{2}$ that $c \neq b_0 \oplus b_1$. Thus, this composition is sound.

Furthermore, the OR composition can be expanded to i statements. Without loss of generality, we can assume that the prover knows one statement x_0 is true and the rest of the statements $\{x_1, x_2, ..., x_i\}$ are false. The prover can run simulators for statements $\{x_1, x_2, ..., x_i\}$ similar to above, guessing the challenge from the verifier on each one. Again, we let the prover set only one challenge $b_j, j \in \{1, 2, ..., i\}$. Setting only one challenge b_j still allows the prover to satisfy the constraint that $c = b_0 \oplus b_1 \oplus ...b_i$ is true. However, this construction is not very efficient since it requires to simulate i statements. For a more efficient implementation, see Section 20.4.1 in the textbook.

17.2 A simulator for the OR composition protocol

A simulator can run the two challenges in parallel, similar to the prover in the scenario above. It can simply guess the challenges b_0 and b_1 at random and simulate the transcript for the first message. After receiving c from the verifier, the simulator can check if $c = b_0 \oplus b_1$. If it is, then the simulator can continue with the protocol as usual. Otherwise, the simulator can restart.

17.3 OR composition application: Encrypting bits

Suppose that a prover wants to show a verifier that they have encrypted a single bit. The encryption scheme has public parameters (g, g^α) for group g and secret α. The prover encrypts the bit b with randomness c as $m = \text{Enc}(b, (g, g^\alpha), c) = (g^{\alpha c} g^b, g^c)$. We want to show that m is either $m = (g^{\alpha c} g^0, g^c)$ or $m = (g^{\alpha c} g^1, g^c)$. If $b = 0$, then $m = (g^{\alpha c} g^0, g^c) = (g^{\alpha c}, g^c)$ which when combined with the public parameters (g, g^α) creates a DH tuple $(g, g^\alpha, g^c, g^{\alpha c})$. If $b = 1$, then $m = (g^{\alpha c} g^1, g^c) = (g^{\alpha c+1}, g^c)$. In this case, we can create a DH tuple if we divide the last term of the tuple $(g, g^\alpha, g^c, g^{\alpha c+1})$ by g. However, revealing if either of the tuples are DH tuples reveals the bit b to the verifier.
To put it more clearly, the statements are:

1. $(g, g^\alpha, (g^c, w))$ is a DH tuple where $w = g^{\alpha c} g^b$ for $b \in \{0, 1\}$
2. $(g, g^\alpha, (g^c, \frac{w}{g}))$ is a DH tuple where $w = g^{\alpha c} g^b$ for $b \in \{0, 1\}$

Applying the OR composition from above, we can simply take the statement 1 as x_0 and statement 2 as x_1 to prove that $b \in \{0, 1\}$ without revealing the true value of b. Futhermore, we can expand this to multiple bits as we did above with the OR composition.

17.4 Non-Interactive Zero-Knowledge: Fiat-Shamir

For a prover to show a certain property, protocols need to run a sufficient amount of times to reduce the probability that the prover is cheating. In a real world system, we do not want this level of interaction between a prover and verifier. We can utilize a public random oracle to select challenges randomly for the prover so that we reduce the interaction with the verifier.

1. A prover appends all of their commitments into one message $\{com_1, com_2, ...com_n\}$ and sends them to a public random oracle.
2. The oracle outputs the challenges $\{ch_1, ch_2, ...ch_n\}$ for these commitments to the prover.
3. The prover sends all of their commitments $\{com_1, com_2, ...com_n\}$ to the verifier and the openings to the challenges $\{open_1, open_2, ...open_n\}$. Having the prover send the challenges $\{ch_1, ch_2, ...ch_n\}$ is optional since the random oracle is public and the verifier can compute the challenges on their own.
4. The verifier can verify the proof with $\{com_1, com_2, ...com_n\}$, $\{ch_1, ch_2, ...ch_n\}$, and $\{open_1, open_2, ...open_n\}$.

This non-interactive method of zero-knowledge proofs only works if all of the commitments and challenges are sent together. If not, the prover can easily cheat by continuously creating new commitments and asking for challenges until the prover can cheat on the challenge it receives. Thus, this works for n bit challenges and does not work for single bit challenges.

17.5 Pairing-based Cryptography

Pairing based groups have extra operations that we have not yet seen in other hard problems. We treat these groups as a black box and do not cover how to implement these groups or how to obtain their operations. Let G_0, G_1, G_T be 3 cyclic groups where $g_0 \in G_0$ and $g_1 \in G_1$ are generators.

Definition 17.1. A pairing is an efficiently computable function e: $G_0 \times G_1 \to G_T$ such that:

1. $g_T = e(g_0, g_1)$ is a generator of G_T
2. For all $(u, u') \in G_0$ and $(v, v') \in G_1$,

   e(u.u',v) =e(u,v).e(u',v) and e(u, v.v') = e(u,v).e(u,v')

We call G_0 and G_1 base groups and G_T as the target of the pair mapping $G_0 \times G_1$. The function e outputs a generator in the target group given the two base groups. Furthermore, the function e can distribute to the products of groups, as we can see from the second point of Definition 17.1. We provide examples of how this property is used below. We can take exponents out of the inner groups and apply it to the output of the function e.

Example 17.2. $e(g_0^2, g_1^b) = e(g_0, g_1^b).e(g_0, g_1^b)$

More generally:

EXAMPLE 17.3. $e(g_0^a, g_1^b) = (e(g_0, g_1^b))^a = (e(g_0, g_1))^{ab} = e(g_0^b, g_1^a)$

Let's consider when $G_0 = G_1$. Then, $g_0 = g_1$. Going back to the DDH assumption, let's suppose we have a triple $(u, v, w) = (g_0^a, g_0^b, g_0^{ab})$. The DDH assumption states that (u, v, w) is indistinguishable from (g_0^a, g_0^b, g_0^c) for a random c. We show that the DDH assumption does not hold for pairing based groups when $G_0 = G_1$.

Proof. We are given (g_0^a, g_0^b, g_0^c). We can compute $e(g_0^a, g_0^b) = (e(g_0, g_0))^{ab}$. Furthermore, we can compute $e(g_0^c, g_0) = (e(g_0, g_0))^c$. Let $e(g_0, g_0) = g_T$. Since we can easily compute g_T^{ab} and g_T^c, we can distinguish between the two DDH tuples $(u, v, w) = (g_0^a, g_0^b, g_0^{ab})$ and $(u, v, w) = (g_0^a, g_0^b, g_0^c)$ for a random c. Thus, the DDH assumption does not hold for pairing based groups. $\square$

A simple distinction between pairing based groups and normal groups is that in normal groups, given g^a, g^b, it is hard to compute g^{ab}. In pairing based groups however, since we have the pairing function e, given g_0^a, g_1^b, it is easy to compute g_T^{ab}.

17.6 Co-CDH assumption with pairing based groups

Recall: the old Computational Diffie-Hellman (CDH) assumption stated that given g_0^a, g_0^b, calculating g_0^{ab} is hard. We play off this same assumption here. Note that computing g_T^{ab} is easy, but we can assume that calculating g_0^{ab} is hard since the pairing function e maps $G_0 \times G_1$ to a different target group G_T.

ASSUMPTION 17.4. We sample a random $(a, b) \in Z_q$. We construct a triple (u_0, u_1, v_0) where $u_0 = g_0^a$, $u_1 = g_1^a$, $v_0 = g_0^b$ and send the triple to the attacker. The attacker outputs some $z' \in G_0$. If $z' = g_0^{ab}$, the attacker wins.

This is a stronger version of the old CDH assumption, since we also provide the attacker with g_1^a and maintain the hardness of computing g_0^{ab}. The attacker should not be able to learn either a or b through this assumption.

17.7 Pairing application: BLS signatures

We construct the BLS signature scheme that is based off of the Co-CDH assumption we discussed in Section 17.6. We construct this signature scheme as follows, assuming we have pairing-based groups G_0, G_1 and a function e that pairs these to a target group G_T.

1. Sample a random k-bit length key a. We use this as the signing key $sk = a$ and use public key $pk = g_1^a$.

2. To generate a signature for a message m, we hash it and raise to the secret key a. More formally, the signing operation is defined as $\text{Sign}(sk, m) : \sigma = (H(m))^a \in G_0$, where $H(m)$ is a hashing operation on the message m and outputs $H(m) \in G_0$.

3. To verify a message m with a signature σ, we use the properties of the function e, the public key g_1^a and the hash of the message $H(m)$. More formally, the verification operation as $\text{Verify}(pk, m, \sigma) : 1$ iff $e(H(m), pk) = e(\sigma, g_1)$.

In more detail, given a message m, a public key g_1^a, and a signature σ, we verify it as follows:

1. We take the hash of the message and compute the function e on the hash of the message and the public key g_1^a.
2. Since $H(m) \in G_0$, $e(H(m), g_1^a) = (e(H(m), g_1))^a$.
3. Taking the exponent a back in, $(e(H(m), g_1))^a = e(H(m)^a, g_1)$
4. We then compare $e(H(m)^a, g_1) \in G_T$ to $e(\sigma, g_1) \in G_T$. If $H(m)^a = \sigma$, then the signature verifies since $e(H(m)^a, g_1) = e(\sigma, g_1)$. Otherwise, the message is forged.

An important note is that, according to the Co-CDH assumption, it is hard to forge a signature of a message m because of the one-way nature of the pairing operation e.

Acknowledgement

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