Stanford University
Topics in Cryptography (CS 355) **Lecture #7**
Instructor: Lior Rotem **April 25, 2023**

Disclaimer *These lecture notes are aimed to serve as a supplementary resource, and are not written as a replacement for attending the in-person lecture. Some material might appear here in a more sketched form than in the lecture, and vice versa. These notes probably contain typos and might even contain errors. If you find any, please let me know.*

Outline *In this lecture, we learn about groups equipped with a bilinear operation (or a pairing). We define such groups, dubbed bilinear groups, and present several constructions of cryptographic primitives, for some of which efficient constructions are not known from standard discrete-log hard groups without pairings. Concretely, we present Joux's non-interactive tri-partite key exchange protocol, Boneh-Franklin's identity-based encryption scheme, and BLS signatures.*

1 Bilinear Groups

In past lectures, we used discrete-log hard groups. Such a group $\mathbb{G}$ was of prime order p, generated by some generator g. We assume that discrete log is hard in $\mathbb{G}$, in the sense that it should be infeasible, given a uniformly-random $h \in \mathbb{G}$, to compute $x \in \mathbb{Z}_p$ such that $g^x = h$.

A *bilinear group* is a group as above, but it also equipped with a bilinear operation $e(\cdot, \cdot)$ that maps pairs of group elements to an element in some other group $\mathbb{G}_T$, often called the "target" group ($\mathbb{G}$ is often called the "source" group). This operation satisfies a number of conditions:

1. Bilinearity: For every $x, y \in \mathbb{Z}_p$ it holds that $e(g^x, g^y) = e(g, g)^{xy}$.
2. It is non-degenerate: $e(g, g)$ generates $\mathbb{G}_T$.
3. Computability: e is efficiently computable.

Asymmetric bilinear groups. One can consider generalizations of the above definition, in which there are two distinct source groups $\mathbb{G}_1$ and $\mathbb{G}_2$, and e maps pairs in $\mathbb{G}_1 \times \mathbb{G}_2$ to $\mathbb{G}_T$. For simplicity, in this lecture, we will focus on the symmetric case in which $\mathbb{G}_1 = \mathbb{G}_2$.

2 Tri-partite Key Exchnage

Reminder: Diffie-Hellman key exchange. Recall the Diffie-Hellman non-interactive key-exchange protocol between two parties, A and B:

1. Ahead of time, A chooses $x \xleftarrow{\$} \mathbb{Z}_p$ and publishes $X = g^x$. Similarly, B chooses $y \xleftarrow{\$} \mathbb{Z}_p$ and publishes $Y = g^y$.
2. When A and B want to agree on a shared key, A computes $k_A = Y^x$ and B computes $k_B = X^y$.

Note that the protocol is always correct since $k_A = Y^x = g^{xy} = X^y = k_B$. It is also non-interactive in the sense that the only two messages sent, X and Y, are independent and can be published ahead of time, before A and B even know they will need to share a key. Then, computing the shared key requires no further communication.

The common security requirement from a key-exchange protocol is that the key is pseudorandom in the view of an adversary that observes the communication. In our setting, this means that g^{xy} should be indistinguishable from a truly uniformly-random element in $\mathbb{G}$, even given $X = g^x$ and $Y = g^y$. The security of the Diffie-Hellman protocol then follows from the *Decisional Diffie-Hellman (DDH)* assumption, which assumes exactly that: $(g^x, g^y, g^{xy}) \approx^c (g^x, g^y, g^z)$, where $x, y, z \xleftarrow{\$} \mathbb{Z}_p$.

Three is a crowd. Now say that we have a third party, C, and A, B and C all want to agree on a key to be shared by all of them. What can they do? One option is to add interaction. An interactive protocol could look like this (other variants also exist):

1. Ahead of time, A chooses $x \xleftarrow{\$} \mathbb{Z}_p$ and publishes $X = g^x$. Similarly, B and C choose $y \xleftarrow{\$} \mathbb{Z}_p$ and $z \xleftarrow{\$} \mathbb{Z}_p$, respectively, and publish $Y = g^y$ and $Z = g^z$.
2. When A, B and C want to agree on a shared key:
   (a) A computes $k_{AB} = Y^x$ and $k_{AC} = Z^x$ and sends k_{AB} to C and k_{AC} to B. Similarly, B computes $k_{BC} = Z^y$ and sends it to A.
   (b) A computes $k_A = k_{BC}^x$, B computes $k_B = k_{AC}^y$ and C computes k_{AB}^z.

This protocol is correct, since $k_A = k_B = k_C = g^{xyz}$. It is also secure based on the DDH assumption (try to prove it!). Alas, it is inherently interactive: k_{AB}, k_{AC} and k_{BC} all depend on previous messages sent by other parties, and cannot be computed ahead of time. In some applications this is fine, but for others, we still want a truly non-interactive protocol.

Pairings to the rescue. In 2000, Antoine Joux came up with a generalization of Diffie-Hellman key exchange to the three-party setting, by relying on bilinear groups. The protocol proceeds as follows:

1. Ahead of time, A chooses $x \xleftarrow{\$} \mathbb{Z}_p$ and publishes $X = g^x$. Similarly, B and C choose $y \xleftarrow{\$} \mathbb{Z}_p$ and $z \xleftarrow{\$} \mathbb{Z}_p$, respectively, and publish $Y = g^y$ and $Z = g^z$.
2. When A, B and C want to agree on a shared key, A computes $k_A = e(Y, Z)^x$ and B computes $k_B = e(X, Z)^y$ and C computes $k_C = e(A, B)^z$.

The protocol is non-interactive; no online communication is needed to compute to shared key. It is correct since

$$\begin{aligned} k_A &= e(Y, Z)^x = e(g^y, g^z)^x = (e(g, g)^{yz})^x = e(g, g)^{xyz} \\ k_B &= e(X, Z)^y = e(g^x, g^z)^y = (e(g, g)^{xz})^y = e(g, g)^{xyz} \\ k_C &= e(X, Y)^z = e(g^x, g^y)^z = (e(g, g)^{xy})^z = e(g, g)^{xyz} \end{aligned}$$

For security, we need that given $X = g^x, Y = g^y$ and $Z = g^z$, it should be hard to distinguish between $e(g, g)^{xyz}$ and a uniformly random element of $\mathbb{G}_T$. To the best of our knowledge, this does not follow from the standard DDH assumption in $\mathbb{G}_T$ (note that the DDH assumption in $\mathbb{G}$ is false!), so

a new assumption needs to be introduced. This assumption is called the *Decisional Bilinear Diffie-Hellman (DBDH)* assumption, and it states that: $(g^x, g^y, g^z, e(g,g)^{xyz}) \approx^c (g^x, g^y, g^z, e(g,g)^u)$ for uniformly-random $x, y, z, u \xleftarrow{\$} \mathbb{Z}_p$.

3 Identity-Based Encryption

Say that Alice and Bob work in the same organization and Alice wants to send an encrypted email to Bob's email address `bob@organization.com`. To do that, she needs to either share a key with Bob or know Bob's public key for some public-key encryption scheme. That is not too bad... just one more key to store, right? But what if this organization has thousands of employees? Alice needs to keep track of thousands of keys.

A more compact solution, first suggested by Shamir in 1984, is to rely on "identity-based encryption" (IBE for short). In such a system, each user is associated with a unique id, e.g., their email address. There is also central authority – e.g., the organization – that generates a global public key pk and a *master* secret key msk. When Alice wishes to encrypt a message to Bob, all she needs to know is the global public key pk and Bob's identity (in our example, his email address). No further information, like a Bob-specific public key, is required. To decrypt, Bob needs a special secret key sk_{bob} that is related it his identity. The organization can issue such a key using knowledge of msk. All in all, each user within the organization needs to record only the global public key pk and their own secret key.

Syntax. Let us be a bit more formal about the syntax of an IBE scheme. Such a scheme is a 4-tuple of algorithms $\Pi = (\mathit{Setup}, \mathit{Extract}, \mathit{Enc}, \mathit{Dec})$:

- $\mathit{Setup}(1^\lambda) \to (msk, pk)$: takes in the security parameter and outputs a master secret key msk and a public key pk.
- $\mathit{Extract}(msk, id) \to sk_{id}$: takes in the master secret key msk and an identity id and outputs a secret key sk_{id} for id.
- $\mathit{Enc}(pk, id, m) \to c$: takes in the master secret key msk, an identity id, and a message m, and outputs a ciphertext c.
- $\mathit{Dec}(sk_{id}, c) \to m$: takes in an identity secret key sk_{id} and a ciphertext c, and outputs a message m.

An IBE scheme is *correct* if for every security parameter λ, identity id and message m, it holds that

$$\Pr\left[\mathit{Dec}(sk_{id}, c) = m \;:\; \begin{array}{l}(msk, pk) \xleftarrow{\$} \mathit{Setup}(1^\lambda) \\ sk_{id} \xleftarrow{\$} \mathit{Extract}(msk, id) \\ c \xleftarrow{\$} \mathit{Enc}(pk, id, m)\end{array}\right] = 1.$$

Security. How should the security of an IBE scheme be defined? Is standard public-key encryption (PKE) semantic security enough? Recall (informally) that a PKE scheme is semantically secure if an adversary, knowing the public key pk, cannot tell whether it received an encryption of m_0 or of m_1, for messages of its choice. This is insufficient. This notion of security is satisfied,

for example, by taking any public-key encrpytion scheme and just ignoring the identities. That is, $msk = sk_{id} = sk$ for every id, and the encryption algorithm ignores id. This is obviously not good, since everyone has the same secret key. So an email Alice encrypted to Bob can also be read by Charlie (and everyone else in the organization)!

So the security notion that we will consider will allow the adversary to obtain secret keys for identities $id_1, \ldots, id_q$ of its choice, but then it has to break semantic security with respect to a different identity id^*. In more detail, the security experiment with adversary A is defined as follows:

1. A key pair $(msk, pk) \xleftarrow{\$} \mathit{Setup}(1^\lambda)$ is sampled and pk is given to A.
2. A adaptively issues secret key queries: each query specifies an identity id, and in response A gets $sk_{id} sk_{id} \xleftarrow{\$} \mathit{Extract}(msk, id)$.
3. A outputs a challenge id id^* and two messages m_0 and m_1. In response, a bit $b \xleftarrow{\$} \{0,1\}$ is chosen and A gets $c \xleftarrow{\$} \mathit{Enc}(pk, id, m_b)$.
4. A can issue additional secret key queries.
5. Eventually, A outputs a bit b'.

We say that A wins in the experiment if $b' = b$ and A never queries for the secret key of id^*. An IBE scheme is secure if for every PPT adversary A, the probability that A wins in the above experiment is at most $1/2 + \mathsf{negl}(\lambda)$.

The Boneh-Franklin IBE scheme. We now introduce the Boneh-Franklin IBE scheme in bilinear groups. The scheme assumes the existence of a hash function H mapping identities to elements in $\mathbb{G}$. For simplicity of presentation, we show how to encrpyt messages in k

- $\mathit{Setup}(1^\lambda)$: Sample a bilinear group $(\mathbb{G}, p, g, e)$ and a element $s \xleftarrow{\$} \mathbb{Z}_p$. Output $msk \leftarrow x$ and $pk \leftarrow (\mathbb{G}, p, g, e, h = g^x)$.
- $\mathit{Extract}(msk, id)$: Output $sk_{id} \leftarrow H(id)^x$.
- $\mathit{Enc}(pk, id, m)$: Compute sample $r \xleftarrow{\$} \mathbb{Z}_p$ and compute $u \leftarrow g^r$ and $v \leftarrow e(H(id), h^r) \cdot m$. Output $c = (u, v)$.
- $\mathit{Dec}(sk_{id}, c = (u, v))$: Compute $z \leftarrow e(sk_{id}, u)$ and output $m = v/z$.

The scheme is correct: fix any $\lambda \in \mathbb{N}$, identity id, and message $m \in \mathbb{G}_T$. Let $msk = x \in \mathbb{Z}_p$ be the master secret key and $pk = h = g^x$ be the corresponding public key. Then, an encryption of m is of the form $c = (u, v)$ where $u = g^r$ and $v = e(H(id), h^r) \cdot m$. From bilinearity, $v = e(H(id), h^r) \cdot m = e(H(id), g)^{xr} \cdot m$. Hence, $\mathit{Dec}(sk_{id}, c) = v/e(sk_{id}, u) = e(H(id), g)^{xr} \cdot m / e(H(id)^x, g^r) = m$.

Security of Boneh-Franklin. The security of the Boneh-Franklin IBE scheme follows from the DBDH assumption in the random oracle model. Suppose there is a PPT algorithm A that breaks the security of Boneh-Franklin; it wins in the security game with probability $1/2 + \epsilon$ for some non-negligible ϵ. Let $q = q(\lambda)$ be the number of random oracle queries issued by A. Assume without loss of generality that A only queries the random oracle on each id once. Further assume that

before asking for the secret key of id or before requesting a challenge ciphertext on (id, m_0, m_1), A always queries the random oracle on id. We construct a PPT algorithm B that breaks the DBDH assumption. On input $(a = g^x, h_2 = g^y, h_3 = g^z, h_T)$ where h_T is either $e(g,g)^{xyz}$ or a uniformly random $\mathbb{G}_T$ element. B does the following:

1. Guess $j \xleftarrow{\$} \{1, \ldots, q\}$ and invoke A on $pk = h_1$.
2. B has to simulate the random oracle and the answers to the secret key queries of A. To simulate the random oracle:
   - For query $i \neq j$, say that the query was id. Sample $x_{id} \xleftarrow{\$} \mathbb{Z}_p$ and return $g^{x_{id}}$.
   - For query j, reply with h_2.
3. Whenever A issues requests for the secret key of id:
   - If A already asked j random oracle queries and id was the jth query, output a random bit and abort.
   - Otherwise, let x_{id} be the exponent used to reply to the random oracle query on id (recall that we assumed A always queries the random oracle before asking for secret keys). B replies with $h_1^{x_{id}}$.
4. When A outputs a target identity id^* and two messages m_0, m_1: If id^* was not the jth query to the random oracle, output a random bit and abort. Otherwise, sample a bit $b \xleftarrow{\$} \{0,1\}$, compute $h_3, h_T \cdot m_b$.
5. When A outputs b', if $b' = b$, output 1 and otherwise output 0.

Conditioned on B guessing the index j incorrectly, it outputs 1 with probability $1/2$ regardless of its input. So we focus on the case in which it guesses j correctly. In this case, if $h_T = g^{xyz}$, then B perfectly simulates the security game to A. Hence, B outputs 1 with probability $1/2 + \epsilon$. If h_T is a uniformly-random group element, then the view of A is independent of the bit b and the probability that $b' = b$ is $1/2$. Hence, conditioned on guessing the index j correctly, B distinguishes between $h_T = g^{xyz}$ and h_T being a random element with advantage ϵ. Since it guesses j correctly with probability $1/q$, the overall advantage of B is ϵ/q which is non-negligible.

Programming the random oracle. Our security proof uses a very strong property of the random oracle model: the reduction can "program" the random oracle. That is, B could choose to answer a query with a specific reply, in our case h_2. This is okay according to the definition of the random oracle model since h_2 indeed comes from the correct distribution, so A is oblivious to this programming going on. Do note, however, that this is an unrealistic feature of the model; concrete hash functions (such as SHA-3) cannot be programmed by the reduction, and their outputs are fixed. Still, programable random oracles are widely assumed for security proofs, so far without dire consequences.

CCA security. The scheme can actually be shown to be secure against chosen ciphertext attacks (CCA), but we will not see the proof. Students are encouraged to try to extend to above security definition and proof to account for such attacks.

4 BLS signatures

Another important application of pairings is short signatures. Also in 2001, Boneh, Lin and Shacham introduced BLS signatures, who are simple and shorter and other digital signatures out there.

Recall that a digital signature scheme is a 3-tuple $\Pi = (KGen, Sign, Verify)$ of algorithms:

- $KGen(1^\lambda) \to (sk, vk)$ where sk is the secret signing key and vk is the public verification key.
- $Sign(sk, m) \to \sigma$. σ is a *signature* on message m.
- $Verify(vk, m, \sigma) \to 0/1$, with 0 implying rejection and 1 acceptance.

For security, we require existential unforgeability against chosen message attacks (EU-CMA). This means that an efficient adversary A that gets the verification key vk and signatures on messages of its choice, should not be able to produce a signature on a *new* message m^* (by "new", we mean that A has not requested a signature on m^*.

The BLS scheme . The BLS scheme assumes a hash function H mapping messages to the source group $\mathbb{G}$. We will model this function as a random oracle in the security proof. The scheme is defined as follows:

- $KGen(1^\lambda)$: Samples a random $x \xleftarrow{\$} \mathbb{Z}_p$ and sets $sk \leftarrow x$ and $vk \leftarrow g^x$.
- $Sign(sk = x, m)$: Outputs $\sigma \leftarrow H(m)^x$.
- $Verify(vk, m, \sigma)$: Outputs 1 iff $e(H(m), vk) = e(\sigma, g)$.

Proving security. The scheme is proven secure under the Computational Diffie-Hellman (CDH) assumption in the source group $\mathbb{G}$: Given $g^x, g^y \in \mathbb{G}$ (for uniformly-random x, y) it should be hard to compute g^{xy}.

Suppose that there is an efficient adversary A breaking the security of BLS. That is, it outputs a signature on a new message with non-negligible probability ϵ. Let $q = q(\lambda)$ be the number of oracle queries issued by A and assume wlog that before requesting a signature on a message m, A always queries the random oracle on m. Also, assume that before outputting a forgery on a message m^*, A queries the random oracle on m^*, and that A never asks for a signature on m^*. We construct an adversary B breaking the CDH assumption. On input $h_1 = g^x$ and $h_2 = g^y$, B does the following:

1. Invoke A on verification key $vk = h_1$ and guess an index $j \xleftarrow{\$} \{1, \ldots, q\}$.
2. Reply to random oracle queries as follows:
   - For query $i \neq j$, let m be the query. B samples $x_m \xleftarrow{\$} \mathbb{Z}_p$ and replies with g^{x_m}.
   - For query j, B replies with h_2.
3. When A requests a signature on a message m: if m was the jth query to the random oracle, abort. Otherwise, B replies with $h_1^{x_m}$.

4. When A outputs a forgery (m^*, σ), check that m^* was the jth random oracle query and that σ is a valid signature on m^*. If not, abort. If so, output σ.

Note that whenever B guesses j correctly, it simulates the security game to A perfectly. If, in addition, A outputs a valid forgery, B breaks CDH. That is because a valid forgery must satisfy $e(H(m^*), vk) = e(\sigma, g)$. But $H(m^*) = h_2 = g^x$ and $vk = h_1 = g^y$. So $e(H(m^*), vk) = e(g, g)^{xy}$. This implies that $\sigma = g^{xy}$.

Overall, the probability that B breaks CDH is ϵ/q.

Signatures from any IBE scheme. Any secure IBE scheme gives a signature scheme in a natural manner. The signing key is the master secret key of the IBE scheme. To sign a message m, one hashes it to the identity space $id \leftarrow H(m)$ and extracts a secret key sk_{id} for this identity. This secret key serves as a signature on the message m. To verify the signature, one can just check that sk_{id} indeed correctly decrypts ciphertexts encrypted to id. BLS signatures can be seen as an instantiation of this paradigm, transforming Boneh-Franklin IBE to a signature scheme.