Lecture 17 - Diffie-Hellman key exchange, pairing, Identity-Based Encryption and Forward Security

Boaz Barak

November 21, 2007

Cyclic groups and discrete log A group G is *cyclic* if there exists a generator g such that for every $a \in G$, $a = g^i$ for some i.

Theorem 1. *If $|G|$ is prime then G is cyclic.*

Proof. Let $g \neq 1$ be some element of G, and consider the set $A = \{g^i : i \in \mathbb{Z}\}$. Then A is a *subgroup* of G and hence $|A|$ divides $|G|$. But this means that either $|A| = |G|$ (and hence g is a generator) or $|A| = 1$ which is impossible since $1, g \in A$. □

Theorem 2. *If $G = \mathbb{Z}_p^*$ for a prime p then G is cyclic.*

Proof. We use the fact that the set of integers modulo p is a *field* and in a field every k-degree polynomial has at most k roots. The fact that the polynomial $x^k - 1$ has at most k roots implies that the group G has the property **(*)** that for every k the number of elements x satisfying $x^k = 1$ is always at most k. We will prove by induction that every group G satisfying **(*)** is cyclic.

Let $n = |G|$. We consider three cases:

- n is prime. Then G is cyclic by Theorem 1.
- $n = p^c$ for some prime p and $c > 1$. In this case if there is no element of order n, then all the orders must divide p^{c-1}. We get $n = p^c$ elements x such that $x^{p^{c-1}} = 1$, violating **(*)**.
- $n = pq$ for co-prime p and q. In this case let H and F be two subgroups of G defined as follows: $H = \{a : a^p = 1\}$ and $F = \{b : b^q = 1\}$. Then $|H| \leq p < n$ and $|F| \leq q < n$ and also as subgroups of G both H and F satisfy **(*)**. Thus by the induction hypothesis both H and F are cyclic and have generators a and b respectively. We claim that ab generates the entire group G. Indeed, let c be any element in G. Since p, q are coprime, there are x, y such that $xq + yp = 1$ and hence $c = c^{xq+yp}$. But $(c^{xq})^p = 1$ and $(c^{yp})^q = 1$ and hence c is a product of an element of H and an element of F, and hence $c = a^i b^j$ for some $i \in \{0, \ldots, p-1\}$ and $j \in \{0, \ldots, q-1\}$. Thus, to show that $c = (ab)^z$ for some z all we need to do is to find z such that $z = i \pmod{p}$ and $z = j \pmod{q}$, but this can be done using the Chinese Remainder Theorem.

□

Discrete logarithm If G is a cyclic group and g is a generator of G, then the *discrete logarithm* of $a \in G$ with basis g, denoted $\log_g a$, is the unique number $i \in \{0, \ldots, |G| - 1\}$ such that $a = g^i$.

Fixing G and g, the *discrete logarithm (DLOG)* problem is, given a random $a \in G$, find $\log_g h$. We say that the problem is *hard* if for every poly A, ϵ, $\Pr_{a \leftarrow_R G}[A(G, g, a) = \log_g a] < \epsilon$.

Theorem 3. *Let G be a group with known order and g, h generators for it. If the DLOG problem is hard w.r.t. g then it is also hard w.r.t. h.*

Proof. Suppose otherwise, that for a random a we could find with probability ϵ the logarithm $\log_h a$. Then, we'll run the algorithm on gh^i for a random i to obtain j such that $gh^i = h^j$ or in other words, $g = h^{j-i}$. Then, we use the equation $\log_g a = \log_h a / \log_h g$ to compute the discrete log w.r.t. g. □

Key exchange and the Diffie-Hellman protocol. Alice and Bob can communicate securely over a line eavesdropped by Eve by having Alice generate a keypair (e, d) for a public-key encryption scheme, send to Bob e, and then Bob can send messages to Alice by encrypting them with e.

However, this is not necessarily the only way to do so. A different approach is using a *key exchange protocol*. The first (and still most used) such protocol was given in the same paper by Diffie and Hellman where they first suggested the "crazy" notion of public key cryptography. We'll first present the protocol and then talk about its security goals.

They use the fact that the group $\mathbb{Z}_p^*$ for a prime p is *cyclic*. This means that there is some number $g \in \mathbb{Z}_p^*$ such that $\mathbb{Z}_p^* = \{1, g, g^2, g^3, \ldots, g^{p-2}\}$. g is called a *generator* for the group. In other words, for every element $x \in \mathbb{Z}_p^*$, there is an $i \in \{0, \ldots, p-2\}$ such that $x = g^i \pmod p$. This number i is called the *discrete log* of x with respect to g.

It is known how to efficiently find a generator g for $\mathbb{Z}_p^*$ given a prime p. It is not known how to compute the discrete logarithm and this problem is believed to be hard.

The Diffie-Hellman protocol:

- Alice chooses prime p at random and finds a generator g.
- Alice chooses $x \leftarrow_R \{0, 1, \ldots, p-2\}$ and sends p, g and $\hat{x} = g^x \pmod p$ to Bob.
- Bob chooses $y \leftarrow_R \{0, 1, \ldots, p-2\}$ and sends $\hat{y} = g^y \pmod p$ to Alice.
- Alice and Bob both compute $k = g^{xy} \pmod p$. Alice does that by computing $\hat{y}^x$ and Bob does this by computing $\hat{x}^y$.
- They then use k as a key to exchange messages using a private key encryption scheme.

Clearly, if Eve can compute the discrete log and obtain x from $\hat{x}$ or y from $\hat{y}$ then this protocol is insecure. Thus the assumption that DH key exchange is secure is stronger than the assumption that the discrete log function is hard to compute (or in other words, that the exponentiation function is a one-way permutation). However, as far as we know, this assumption is *not* sufficient for the security of Diffie-Hellman protocol. We can also make the following stronger assumption:

Computational Diffie Hellman (CDH) assumption: For the group $G = Z_p^*$, with a generator g, the problem of computing g^{xy} from g^x, g^y is hard on the adverage.

However, even this is not known to suffice for security. What we need is a stronger assumption which is the following:

Decisional Diffie Hellman (DDH) assumption — Take 1. For every prime p and generator g of $\mathbb{Z}_p^*$, the following two distributions A and B over triplets are computationally indistinguishable: $A = \langle g^x, g^y, g^{xy} \rangle$ for random x and y in $\{1, \ldots, p-2\}$ and $B = \langle g^x, g^y, z \rangle$ for random x and y in $\{1, \ldots, p-2\}$ and $z in \mathbb{Z}_p^*$.

This assumption implies that as far as Eve is considered, the key k is a random element in Z_p^* (i.e., a random number between 1 and $p-1$) and hence can be safely used as a key for any private key encryption scheme. For example, to send a message m of length ℓ, Bob can send Alice $k \oplus m$.

Unfortunately, this assumption is not true (although as far as we know it is "morally true") for a very simple reason: given a number $\hat{y} \in \mathbb{Z}_p^*$, we can check if it has a square root modolu p (i.e., whether it is a quadratic residue). It is known that g^x is a quadratic residue if and only if x is even. Thus, given g^x and g^y we can test whether x and y are even (which happens with probability 1/4) and in this case g^{xy} will be also a quadratic residue, while a random element in $\mathbb{Z}_p^*$ will only be in QR_p with probability 1/2.

Fortunately, the assumption can be made for other groups in which it is believed to be true. One such group is the subgroup of quadratic residues mod p, for p of the form $p = 2q+1$. See `http://crypto.stanford.edu/~dabo/abstracts/DDH.html` for more about this assumption.

El-Gamal Encryption

Forward security. Protecting secret keys is crucial for cryptography. But what happens if the adversary does learn the key? Indeed, suppose I have a secret decryption key, corresponding to some known public key which I use for a long time, and at some point an attacker breaks into my computer and learns the secret key without my learning all about it.

It's clear that from now on, the attacker will be able to read all the encrypted messages that are sent to me. It's also seems intuitively clear that if the attacker recorded previously the ciphertexts of the encrypted messages that were sent to me before he gained access to my computer, then now he will be able to use my secret key to decrypt these messages. Surprisingly (or perhaps not, since this is crypto and strange things always happen) this intuition is false, and it is possible to ensure that the attacker will only be able to decrypt message sent after he broke into my computer, even if I don't know when or whether or not he broke into it.

Forward-secure encryption schemes Encryption schemes that maintain this property are called *forward secure*. A forward secure public key encryption scheme has the following components:

Key generation As usual, G outputs a public key e and a secret key, which we denote by d_0.

Encryption algorithm The encryption algorithm E takes as usual as inputs the encryption key e and the message to be encrypted m. However, it takes also an additional input t, which is the current time (or time period).

Update mechanism We'll use a different secret key to decrypt at each time period t. We'll denote this secret key by d_t. Of course, we must be able to efficiently compute d_t for

every t given the original key d_0 (since that is all the information the receiver has). However, we'll actually do it in the following way: there is an algorithm $UPDATE$ that on input t and d_{t-1} outputs d_t. At the beginning of each time period t, the receiver will compute $d_t = UPDATE(t, d_{t-1})$ and *erase* d_{t-1} from its memory.[1] (It will be the case that this algorithm $UPDATE$ is hard to invert, that is, from d_t it's hard to come up with d_{t-1}.)

Decryption To decrypt a message sent at time t, we'll use d_t. Thus the validity condition is that for every m, $\mathsf{D}_{d_t}(\mathsf{E}_e(m,t)) = m$.

We can define forward-secure variants of both CPA security and CCA security. The idea is that we run the usual attack game (either CPA or CCA), except that there is a global time counter t that the adversary can ask to increase by one from time to time. The adversary then chooses two messages m_1 and m_2 and gets the challenge — an encryption of m_b for $b \leftarrow_{\mathrm{R}} \{1,2\}$. The game again continues as in the usual CPA/CCA case. we continue this game as usual. However, before the adversary needs to guess b, the time counter t is increased by one, and the adversary is given the secret key d_t. The adversary can then use that information in order to try to guess b with probability greater than $1/2$.

Other forward secure primitives The notion of forward security is pretty general, and there are definitions and constructions for forward secure signature schemes, pseudorandom generators, private key encryption, and others.

Constructing forward secure encryption schemes We are going to construct forward secure encryption schemes using another object that is called *identity-based encryption schemes*. Identity-based encryption schemes are themselves just as fascinating (and perhaps even more) as forward-secure encryption schemes. The idea was first suggested by Shamir in the 80's but a construction was only given in 2001 by Boneh and Franklin. Even that construction was only proven secure in the random oracle model and getting a random-oracle free construction seemed to be a very hard problem to many researchers (including myself). However in 2004, Boneh and Boyen (improving on Canetti, Halevi and Katz) managed to get a construction proven secure under reasonable computational assumptions, without any random oracles.

Identity based encryption The idea of identity based encryption is very simple - what if your name could be your public key? That is, where in standard public-key crypto, if I want to send Dave a secure email he has to send me his public key (or perhaps publish it in a public key directory) in IBE my encryption algorithm simply takes the string "Dave Xiao" as an input.

More accurately, an identity-based encryption (IBE) is comprised of the following parts:

Master key generation There is an algorithm G_{master} that generates the master public and private keys pub_{master} and $priv_{master}$.

Key derivation There is an algorithm $Derive$ that gets as input the private key $priv_{master}$ and an arbitrary string $id \in \{0,1\}^*$, and outputs a decryption key d_{id}.

Encryption To encrypt a message m to ID id, run $\mathsf{E}_{pub_{master},id}(m)$.

[1] Note that there are many technical difficulties involved in securely erasing memory from modern computers, that use hard-drives, virtual memory and paging. We ignore these issues here.

Decryption There is a decryption algorithm that takes as input the decryption key k_{id} and a ciphertext y, where the validity condition is that for every m and id,

$$\mathsf{D}_{d_{id}}(\mathsf{E}_{pub_{master},id}(m)) = m$$

Again IBE can be defined with either a CPA or CCA variant. In both cases the adversary gets the public master keys pub_{master} and runs the usual CCA/CPA attack. However, it now gets an additional oracle access to the key derivation algorithm, to which it can query a string id and get back d_{id}. When making the challenge the adversary not only specifies two messages m_1 and m_2 but also an ID id^*, and gets the challenge ciphertext $y^* = \mathsf{E}_{pub_{master},id^*}(m_b)$ for $b \leftarrow_{\mathrm{R}} \{1,2\}$. The adversary has now additional access to key derivation algorithm, but conditioned on not asking the query id^*, and if it's a CCA attack also access to the decryption oracle, where it can make any query of the form $\langle id, y\rangle$ as long as either $id \neq id^*$ or $y \neq y^*$. Again, the adversary is successful if it guesses b with probability noticably higher than $1/2$.

Forward-secure encryption from IBE Given an IBE scheme, one can construct a forward-secure encryption in the following way:

- The public and private keys are generated as follows: generate pub_{master} and $priv_{master}$ using the generator for the IBE scheme. Assuming we're going to use this scheme for T time period, for every $1 \leq t \leq T$, let id_t denote the string "time slot t" and let k_t denote $DERIVE(priv_{master}, id_t)$. The private key d_0 will be the concatenation of k_1 until k_T.
- To encrypt at time t simply run $\mathsf{E}_{pub_{master},id_t}(m)$.
- The key d_t will be the concatenation of k_t until k_T. That is, the update mechanism at time t involves erasing the key k_t from the list.

It's not hard to prove this scheme is secure. However, its drawback is that it requires the private key to be of size nT and maintaining such a large secret storage may be infeasible. It can be easily improved however, to require the receiver to only public storage of this length: instead of storing $k_1, \ldots, k_T$ store k_1 in private and keep a non secret file $y_2, \ldots, y_T$ where $y_t = \mathsf{E}_{pub_{master},k_{t-1}}(y_{t-1})$. There are also constructions without need for any storage that depends on T worse than logarithmically.

Other applications IBE has several other potential applications. For example, suppose that when sending email to me, people use the ID "Boaz Barak $\circ$ current date". Then, when I go to a conference with my laptop, I can keep in the laptop only the private keys corresponding to these dates. It has also been suggested that a manager can use IBE to provide assistants with "restricted private keys" that can only decrypt messages with particular subjects. In any case IBE is quite cool. In fact, Boneh and others formed a company (Voltage) based on the IBE technology.

Construction of IBE We present the random oracle based construction of Boneh and Franklin.

Pairing diffie hellman assumption The DDH assumption says that in an appropriate cyclic group G with a generator g, it is impossible to distinguish between the triple (g^a, g^b, g^{ab}) and the triple (g^a, g^b, g^c) for x, y, z chosen independently at random from $\{0, \ldots, |G| - 1\}$.

Consider the following question: can there be a group where it's actually *easy* to compute g^{ab} from g^a, g^b but given g^a, g^b, g^c it's *hard* to compute g^{abc}. It's not hard to see that this is

impossible - if you have an algorithm to compute the first problem, you can apply it twice to obtain first g^{ab} and then g^{abc} to solve the second problem. However, we will somehow manufacture a situation where this is "morally true". We are going to consider two cyclic groups G and H with $|G| = |H|$ and generators g and h respectively and a function $f : G \times G \to H$ satisfying the following: $f(g^a, g^b) = h^{ab}$. It turns out it is possible to come up with such groups and a function. In some sense we manage to solve the first problem, but only when moving to a different group.

We are going to make the following assumption (called **pairing DDH**): for random a, b, c, d it is impossible to distinguish between $\langle g^a, g^b, g^c, f(g^a, g^b)^c = h^{abc} \rangle$ and $\langle g^a, g^b, g^c, h^d \rangle$.

Assuming this, we will build an identity-based cryptosystem as follows:

Public and private master keys Generate groups G, H and generators g, h and function $f : G \times G \to H$ such that $f(g^a, g^b) = h^{ab}$. Let $R : \{0,1\}^* \to G$ denote a random oracle. Choose a at random from $\{0, \ldots, |G| - 1\}$ and publish g^a. a is the secret key.

Identity keys For an identity id, let $e_{id} = R(id)$ (the random oracle applied to the string id. We let $b \in \{0, \ldots, |G| - 1\}$ be a number such that $e_{id} = g^b$. Note that no one (including even the holder of the master private key) knows b. The secret key for id, $d_{id} = e_{id}^a = g^{ab}$. Note that it can be derived using the private key.

Encryption To encrypt a message m for ID id, choose $c \leftarrow_{\mathrm{R}} \{0, \ldots, |G| - 1\}$, compute $\pi = f(g^a, e_{id})^c = h^{abc}$ and send $g^c, \pi \oplus m$.

Decrypt Given the secret key g^{ab} and the message $g^c, h^{abc} \oplus m$, the receiver computes $\pi = f(g^{ab}, g^c) = h^{abc}$ and uses that to retrieve the message.

Assuming that g^d for a random d is represented as a random string, this scheme can be shown to be CPA secure under the pairing DDA assumption. By further using (or abusing?) the random-oracle it can be shown secure under a weaker assumption (namely pairing computational Diffie-Hellman (CDH) assumption) and can also be made CCA secure. For more details of the proof, see the paper by Boneh and Franklin. **Note:** In that paper, as in most other papers in this subject, *additive* notation is used for the group G (but not for H). Thus, instead of g^a you will see there $a \cdot g$, and the pairing DDH assumption will be that for random a, b, c, d the tuple $\langle a \cdot g, b \cdot g, c \cdot g, f(a \cdot g, b \cdot g)^c \rangle$ is indistinguishable from $\langle a \cdot g, b \cdot g, c \cdot g, f(a \cdot g, b \cdot g)^d \rangle$