Cryptography Part 1 **Fall 2017**

Lecture 5: Basic Key Exchanges

Lecturer: Dan Boneh *Scribe: Lily Li*

5.1 Trusted Third Parties

Suppose there are n users in the universe. In order for each pair of users to share a secret key, we can make use of an Online Trusted 3^{rd} Party (TTP).

Each user u_i shares a secret key k_i with the TTP. If u_1 wishes to communicate with u_2, u_1 will contact the TTP. The TTP will send u_1 a pair of encrypted messages $E(k_1, \text{"}1, 2\text{"} \parallel k_{1,2})$ and $E(k_2, \text{"}1, 2\text{"} \parallel k_{1,2})$ where $k_{1,2}$ the secret key that u_1 and u_2 will share. u_1 can decrypt the first message and obtain the secret key $k_{1,2}$. Then u_1 sends u_2 the second ciphertext, known as the ticket which u_2 can decrypt and also obtain $k_{1,2}$.

Note: the encryption protocol (E, D) must be CPA-secure. This protects against eavesdropping attackers but does not ensure integrity.

The problem with this strategy is that the TTP is needed for every key exchange. If the TTP were compromised then all the sessions keys would be at risk. Further, this strategy is at risk of replay attacks.

Next we want to consider key exchange strategies which do not require a TTP.

5.2 Merkle Puzzles

Suppose that *Alice* and *Bob* wishes to establish a secret key k. Is the possible to do this using generic symmetric cryptography (hash functions, block ciphers, etc)? Yes! The Merkle construction only protects against an eavesdropper, but can do just that.

The central building block of this protocol is the concept of a puzzle.

Definition 5.1 *A **puzzle** is a hard problem which can be solved using a bit of work.*

E.g. It is nearly impossible to find a random 128-bit secret key k. However, if I told you that the k has the format: $00\cdots 0b_1b_2\cdots c_{32}$ then the problem becomes tractable. Simply iterate through all 2^{32} possibilities for $b_1b_2\cdots b_{32}$.

Suppose that *Alice* wishes to share a secret key with *Bob*. She will prepare 2^{32} different puzzles where for each puzzle she generates a random P_i, and two random strings $x_i, k_i \in \{0,1\}^{128}$. She then sends 2^{32} ciphertexts of the form $E(0^{96} \parallel P_i, \text{"}Puzzle\#x_i\text{"} \parallel k_i)$. *Bob* will choose one ciphertext from this set at random and spend $O(2^32)$ time solving the puzzle. A solution to a puzzle is any message which starts with *Puzzle*. Once *Bob* solves the puzzle, he sends back x_j. *Alice* looks up the key associated with x_j and the two can begin communicating with key k_j.

Generally, if n messages were send and it takes $O(n)$ to solve each puzzle, each of the communicating parties spent time $O(n)$ while an eavesdropper will need to spend time $O(n^2)$.

This gap between the amount of work required by the participants and the work required by the attacker is crucial. Unfortunately, it is believed, though not proven, that a quadratic gap is the best possible if use only use a general symmetric cipher (it has been proved that this gap is the best possible if the ciphers were treated as black box oracles).

5.3 Diffie-Hellman Protocol

This protocol uses some number theory to obtain an approximately exponential gap. Again, only protects against eavesdroppers.

The protocol between parties *Alice* and *Bob* are as follows:

1. Fix a large prime p, about 600 digits (PUBLIC).
2. Fix an integer $g \in \{1, ..., p\}$ (PUBLIC).
3. *Alice* chooses a random $a \in \{1, ..., p-1\}$. She sends $A \leftarrow g^a \mod p$ to *Bob*.
4. B also chooses a random $b \in \{1, ..., p-1\}$ and sends $B \leftarrow g^b \mod p$ to *Alice*.

The shared key $k_{AB} = g^{ab} \mod p$. Observe that both parties can calculate this value but the attacker who only observes the communications between the two cannot.

This protocol is completely insecure against an active **man-in-the-middle (MiTM)** attack. Can you see what the MiTM needs to do to listen in on the conversation between *Alice* and *Bob* without either party finding out?

Another nice property of the DH protocol is that it is non-interactive. Suppose we have users $u_1, u_2, u_3, ...$ Each user u_i choses a secret key i and puts their part of the DH protocol g^i in a public space. Notice that no communication is necessary between two parties to establish a secret key. Each person simply read the key from the public space and computer the shared key.

Note: it is an open problem to create a non-interactive protocol for a group four or more people communicating with each other.

5.4 Public Key Encryption

Definition 5.2 *A public-key encryption system is a triple of algorithms (G, E, D) where G is a randomized algorithm which outpust a key pair (pk, sk), $E(pk, m)$ is a randomized encryption algorithm, and $D(sk, c)$ is the deterministic decryption algorithm. As always, consistency is required:*

$$\forall (pk, sk) \text{ output by } G : \forall m \in M : D(sk, E(pk, m)) = m$$

Observe that the Encryption and decryption algorithms take different keys.

We define semantic security for a public key encryption system as follows:

Definition 5.3 *As always, define two experiments $EXP(b)$ for $b \in \{0, 1\}$. The challenger takes as input b and generates a pair of keys (pk, sk) using G. The challenger then gives the adversary the public key pk. The adversary sends back two messages $m_0, m_1 \in M : |m_0| = |m_1|$ and the challenger encrypts m_b, sending $c \leftarrow E(pk, m_b)$. The adversary outputs $b' \in \{0, 1\}$.*

The way we use public key cryptography to establish a shared key between two parties *Alice* and *Bob* is as follows:

1. *Alice* uses G to generate a pair of keys (pk, sk). She sends the message "Alice", pk to *Bob*.
2. *Bob* choses a random $x \in \{0,1\}^{128}$ and sends back: "Bob", $c \leftarrow E(pk, x)$.
3. Finally *Alice* gets back x from $D(sk, c)$ and x is now the shared key.

Note: this protocol is interactive, and the participants must communicate in turn. Further, this protocol is also insecure against a MiTM attack.

5.5 Number Theory Background

Just a reminder of Fermat's theorem. For a prime p, $\forall x \in (Z_p)^* : x^{p-1} = 1$ in Z_p.

We can use Fermat's theorem to generate random primes as follows: suppose we wanted a 1024-bit prime. We would choose a random integer $p \in [2^{1024}, 2^{1025-1}]$. Then text if $2^{p-1} = 1$ in Z_p. It turns out, with very high probability that if p passes the test then it is a prime i.e. $\mathsf{Pr}[p \text{ not prime }] < 2^{-60}$

Recall that Euler's ϕ function is defined as $\phi(N) = |(Z_N)^*|$. Further, if $N = pq$ where p, q are primes it is not difficult to see that $\phi(N) = (p-1) * (q-1)$.

Theorem 5.4 *(Euler):* $\forall x \in (Z_N^* : x^{\phi(N)=1}$ *in* Z_N.

5.5.1 Computing Modular eth Roots

Definition 5.5 *Let $x \in Z_p$ such that $x^e = c$ in Z_p. Then we call x an e^{th}* ***root*** *of c.*

The task of computing e^{th} roots of a number in Z_p is divided into different cases. First the case when $\gcd(e, p-1) = 1$. Then for all $c \in (Z_p)^*$.: $c^{1/e}$ exists and is easy to find since $\exists d \in (Z_p)^*$ such that $ed = 1$.

Next consider what happens when $\gcd(e, p-1) \neq 1$. In particular consider what happens when $e = 2$.

Definition 5.6 *$x \in \mathbb{Z}_p$ is a* ***quadratic residue (QR)*** *it it has a square root in $\mathbb{Z}_p$. If p is an odd prime then the number of QR in $\mathbb{Z}_p$ is $(p+1)/2$.*

Theorem 5.7 *(Also Euler) $x \in (Z_p)^*$ is QR if and only if $x^{(p-1)/2} = 1$ is in Z_p where p is an odd prime.*

Unfortunately Euler's theorem is not constructive, so we need another way to compute square roots modulo p. We do this by cases:

1. $p = 3 \mod 4$: if $c \in Z_p$ is QR then $c^{1/2} = C^{(p+1)/4}$ in Z_p.
2. $p = 1 \mod 4$: this can be in using a randomized algorithm with running time $O(\log^3 p)$.

Thus it is possible to solve a quadratic equation modulo Z.

Now we can consider computing the e^{th} roots modulo composite. It turns out that is might require difficult subroutines such as prime factorization.