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

Instructor: Dakshita Khurana
Scribe: Eric Cao, Brikhoff Cheng
Date: September 22, 2020

LECTURE
9

Public Key Encryption 1

In previous lectures we discussed symmetric key encryption, which requires a secret key to be transferred through a covert channel. Since we cannot share the key covertly in symmetric key encryption, we must bootstrap the process. The method for setting up this shared secret is public key encryption. Unlike symmetric key encryption, each party in public key encryption has two keys (a public and private key). In this setup, Alice can encrypt a message with Bob's public key and Bob can decrypt the ciphertext with his secret key. From an adversarial perspective, all information here is either public or encrypted, which makes this a secure mechanism to distribute the shared secret key for symmetric key encryption.

In this lecture, we look at key exchange and the mathematical fundamentals that make public key encryption possible. Public key encryption is based on the assumption of fundamentally hard problems called **one-way trapdoor functions**. These are an extension on the one-way functions talked about previously when discussing hash functions. Just like one-way functions, these functions are easy to compute, but hard to invert. However, these functions have a **trapdoor**, which is a secret that allows us to efficiently invert the function. This trapdoor is what allows for decryption of the message to be efficient for Alice and Bob, while any adversaries without access to the secret key have a negligible probability of uncovering the plaintext. One such hard problem is the **discrete log problem**, which we explore in detail since the **Diffie-Hellman** problem is based on it.

9.1 Number theory fundamentals

As the discrete logarithm problem that underpins the Diffie-Hellman Key Exchange has its roots in number theory, we will begin by covering constructs in number theory and notation. We will be using $\mathbb{Z}_N$ to represent the set of equivalence classes modulo N and $(\mathbb{Z}_N)^*$ will be the set of invertible elements in $\mathbb{Z}_N$.

In group theory, there is a concept called a cyclic group. A cyclic group is a group that is generated by an element g such that every element in the cyclic group can be obtained by applying a single operation. We use multiplication modulo p for this.

Definition 9.1. $(\mathbb{Z}_p)^*$ is a cyclic group. This means there exists a generator g such that $\{1, g, g^2, ..., g^{p-2}\} = (\mathbb{Z}_p)^*$.

Example 9.2. For $p = 7$. 3 is a generator. $\{1, 3, 3^2, 3^3, 3^4, 3^5\} = \{1, 3, 2, 6, 4, 5\}$.

Despite 2 being in the generated set it is not a generator since $\{1, 2, 2^2, 2^3, 2^4, 2^5\} = \{1, 2, 4\}$

As shown in the example, though every element e appears in the cyclic group $(\mathbb{Z}_p)^*$, not every element in $(\mathbb{Z}_p)^*$ is a generator for $(\mathbb{Z}_p)^*$. So the natural question is how do we find generators for the group $(\mathbb{Z}_p)^*$? To do this, we must first define the order of a group. In $(\mathbb{Z}_p)^*$, the set $\{1, g, g^2, ...\} =< g >$ is the group generated by g. This means g is a generator if and only if $< g >= (\mathbb{Z}_p)^*$. We call the size of this group its order.

Definition 9.3. The order of $g \in (\mathbb{Z}_p)^*$ is the size of $< g >$

$$ord_p(g) =|< g >|= \text{smallest } a \text{ s.t. } g^a = 1$$

So, how does the order relate to the generator? Well Lagrange's theorem allows us to identify potential primes whose generators are easy to find.

Theorem 9.4. *Lagrange's theorem. For all $g \in (\mathbb{Z}_p)^*$ the $ord_p(g)$ divides $p - 1$*

So, we can see that as long as pick p such that $p - 1$ has very few factors. For example if $(p - 1) = a_1 a_2$ where a_1, a_2 are primes, then there are only 4 possibilities for the order $1, a_1, a_2, a_1 a_2$. So, we can pick a random element and compute the order. We know that one of $g^1 = 1, g^{a_1} = 1, g^{a_2} = 1, g^{a_1 a_2} = 1$, so we continue until we find a generator such that $g^{a_1 a_2} = 1$. By doing this, we can make guess and check an efficient algorithm to find generators.

A lot of public key cryptography is based on the notion of picking a group, finding its generators, and relying on hard problems for the one-way function. For example, RSA is based on Euler generalization of Fermat's little theorem. Recall that Fermat's little theorem states that for any number x and prime p that is relatively prime, then x^{p-1} is also relatively prime with p.

Theorem 9.5. *Fermat's little theorem. Formally, let p be an arbitrary prime number. Then, $\forall x \in (\mathbb{Z}_p)^*\ x^{p-1} \equiv_p 1$.*

Fermat's theorem has a few applications, which include a way to compute inverses as $x * x^{p-2} \equiv_p 1 \implies x^{-1} \equiv_p x^{p-2}$ and as a primality test. You can draw a random number p and test if $2^{p-1} \equiv_p 1$ efficiently. Note that this test is not perfect as $2^{341-1} \equiv_{341} 1$. Since $341 = 11 \times 13$, the test incorrectly identified 341 as a prime.

Euler later generalized Fermat's little theorem to non-prime values. To do this, he defined a totient function φ to be the number of invertible elements in $\mathbb{Z}_N$ for any integer N. Notably, for $n = pq$, $\varphi(N) = (p - 1)(q - 1)$.

Definition 9.6. Euler's totient function. For integer N, $\varphi(N) = |(\mathbb{Z}_N)^*|$.

Theorem 9.7. *$\forall x \in (\mathbb{Z}_N)^*$, $x^{\varphi(N)} = 1$ in $\mathbb{Z}_N$.*

9.2 Hard problems

Within a group, there are both easy and hard problems. We rely on the hard problems to serve as our one-way trapdoor functions in order to make public key decryption hard for adversaries and possible for Alice and Bob given the secret key. One particular hard problem is the discrete log problem.

PROBLEM 9.8. Discrete Log. Fix a prime $p > 2$ and g in $(\mathbb{Z}_p)^*$ of order q.

Computing $f(x) = g^x$ can be efficiently computed through repeated squaring, which takes roughly $log(N)$ iterations to compute.

However, the inverse $Dlog_g(g^x) = x$ is hard to compute. The naive way to compute this is by cycling through p^x for x in $\{0, \ldots, q-2\}$. This can be improved to only cycle through $2^{\sqrt{q}}$ possibilities, but for $q = 1024$, this is clearly infeasible.

We say that discrete log is hard in a cyclic group G with generator g if all adversaries have a negligible probability of finding x given the G, q, the g, and g^x. This makes it a candidate for a one-way function since it is easy to compute g^x, but hard to find x given g^x.

One application of this hard problem is a collision-resistant hash function. Suppose we have a group G where discrete logarithm is hard. Then we choose a generator g of G and a secret s and set $h = g^s \bmod G$. Then for any message $(x, y) \in |G|^2$ the hash of (x, y), $H(x, y) = g^x \cdot h^y \bmod G$

LEMMA 9.9. *Finding a collision* $(x_0, y_0) \neq (x_1, y_1)$ *such that* $H(x_0, y_0) = H(x_1, y_1)$ *is as hard as finding* $s = discrete \log_g(h)$.

Proof. For the sake of contradiction, suppose discrete logarithm in G is hard and we have a collision $(x_0, y_0) \neq (x_1, y_1)$ such that $H(x_0, y_0) = H(x_1, y_1)$.

Since $H(x_0, y_0) = H(x_1, y_1)$, by the definition of H,

$$g^{x_0} \cdot h^{y_0} = g^{x_1} \cdot h^{y_1}$$

$$g^{x_0 - x_1} = h^{y_1 - y_0}$$

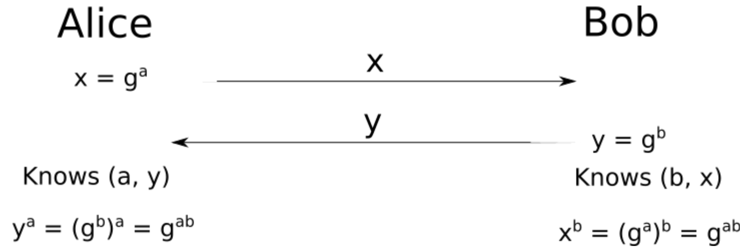
$$h = g^{\frac{x_0 - x_1}{y_1 - y_0}}$$

$$s = \frac{x_0 - x_1}{y_1 - y_0}$$

We have found the discrete log of h, there is a contradiction. So H is collision resistant. ∎

9.3 Key Exchange

Now that we have a one-way trapdoor function, let use it to see if we can use it to exchange keys.



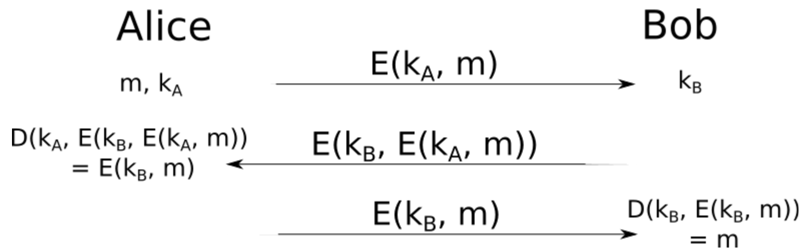
So, know both Alice and Bob know the value g^{ab}. As long as we assume that given g^a, g^b, and g it should be hard to find g^{ab}, they have successfully shared a secret key that the adversary cannot figure out. This problem is called the **Computational Diffie-Hellman Problem**. This problem is hard only if discrete log problem is hard. This is fairly trivial since if discrete log is easy, we can find a and b by computing the discrete log and then compute g^{ab}. However, this is a stronger assumption than what discrete log uses so it is possible for the Computational Diffie-Hellman problem to be easy while the discrete log problem is hard.

Open Problem 9.10. Can we use the discrete log problem or some one-way function by itself to establish shared secrets? Do we need to rely on another assumption like the computational discrete log problem?

There are a lot of impossibilities that make key exchange hard only using discrete log. In practice, we know that one-way functions are easier to implement than mechanisms to share secret keys.

Now let's see that if we have a pre-shared key, can we use a private encryption scheme with rerandomization to do public key encryption. This is called re-randomizable encryption. We propose two methods for implementing re-randomizable encryption.

We can use the three-pass protocol in order to do this. The intuition for this is pretty simple. Imagine the key is a treasure chest, Alice locks the chest with her key before passing the chest to Bob to prevent anyone from opening the chest. Bob will then add his key before passing it back to Alice. Alice removes her key and passes it back to Bob. Bob removes his key and has the unlocked chest. Since any adversary monitoring the communication never sees an unlocked chest, they can never open it. To make this more formal, we rely on the assumption that for our encryption algorithm $D(d, E(k, E(e, m))) = E(k, m)$ for any arbitrary key k. This is possible with a commutative encryption scheme. We declare keys e, d to be Alice's encryption and decryption keys and k, l to be Bob's decryption keys.



Alternatively, you can use re-randomization. Alice publishes $c_0 = E(k_A, 0), c_1 = E(k_A, 1)$. If Bob can inject randomness by picking c_b and **rerandomizes** to get $\widetilde{c_b}$. Send $\widetilde{c_b}$ back to Alice. We want $\widetilde{c_b}$ to map to the same bit that was originally chosen by Bob and for it to

be computationally hard for an attacker to tell from $\widetilde{c_b}$ if 0 was rerandomized or if 1 was rerandomized. This property is satisfied by homomorphic encryption.

Note that the three pass protocol from before is not secure with xor encryption since we reuse Bob's key twice as an encryption for the plaintext and as a reencrpytion of Alice's ciphertext

$$m_1 = k_A \oplus m.$$

$$m_2 = k_A \oplus m \oplus k_B.$$

$$m_3 = k_B \oplus m.$$

So, the attacker can compute

$$m1 \oplus m2 \oplus m3 = m.$$

So, we have create a way to distribute the keys using public key cryptography. Is this mechanism secure? It is not. If we have a man in the middle, the man in the middle can pretend to be Bob to Alice and pretend to be Alice to Bob.

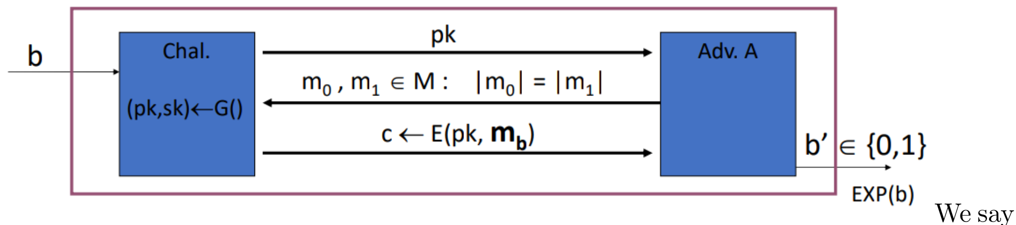
9.4 Public Key Encryption

Definition 9.11. Public key encryption system $\mathbb{E} = (G, E, D)$

G(): randomly generates a key pair (sk, pk)
E(pk, m): randomized algorithm that encrypts message m and outputs ciphertext c.
D(sk, c): deterministic algorithm that decrypts ciphertext c and outputs message m.
such that

$$\forall (pk, sk), \forall m \in M, D(sk, E(pk, m)) = m$$

Definition 9.12. Semantic Security of Public Key Encryption System



We say a cipher $\mathbb{E}$ is sematically secure if $\forall A$,

$$Adv_{SS}[A, \mathbb{E}] = |\Pr[EXP(0) = 1] - \Pr[EXP(1) = 1]| < \delta$$

where δ is negligible.

With a semantically secure public key encryption system, we can easily construct a key exchange protocol, where Alice publishes her public key pk, Bob sends a secret key x encrypted with that public key, and Alice decrypts that message to get the secret key x.

Acknowledgement

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

References