1 Lecture 11: Public-Key Encryption

1.1 Key Agreement

We begin with a question regarding how our encryption scheme will actually take place. Throughout what we have done so far, we have made the assumption that malicious parties would not have the key we were using to communicate secretly. Even with PRG's and OTP, we still need to communicate the original seed securely. So how do we do this?

1.1.1 Diffie-Hellman Key Exchange

The following is an idea for how to create a shared OTP key between two parties that no third party can figure out, even over insecure communication channels.

0. Both parties, A and B know some (large) prime p, a generator g mod p, and a PRG.
1. A generates a random number x, and sends $a = g^x$ to B.
2. B generates a random number y, and sends $b = g^y$ to A.
3. A and B compute $seed = b^x = a^y = g^{xy}$.
4. A and B compute $key = PRG(seed)$.

The reason no eavesdropper can learn the key is called the **Decisional Diffie-Hellman (DDH) Assumption**, which states that the following distributions are computationally indistinguishable:

$$(g, g^a, g^b, g^{ab}) \sim (g, g^a, g^b, g^r),$$

where all of a, b, r are completely random.

Basically, what this says is that given a generator g, and two values g^x and g^y, it is as hard for a **PPT** adversary to figure out $g^x y$ as it is to figure out g^r for some random, undisclosed r.

2 Public Key Encryption

We now define public-key encryption, which is a slightly more generalized notion over key exchanges. We do something similar to what we did in DHKE, but we do it for every message we send.

The idea here is that one party, A, will have a private (decryption) key that will be shared with no one, and there will be a public (encryption) key that is available to everyone. So, anyone will be able to encrypt and send messages, but only A will be able to decrypt and read them.

There are 3 stages:

1. Keygen: given a security parameter k and some randomness, we generate a public key and secret key.
2. Encryption: given a message M, a public key, and some randomness, we generate some ciphertext CT.
3. Decrption: given CT and the secret key, we can reproduce the message M.

It is extremely important to note that the encryption step takes in randomness as input. This encryption is not one-to-one, it is one-to many. Each message can have multiple possible ciphertexts, and that is precisely what makes it secure.

2.0.1 Security Notions

There are two important security notions for public-key encryption:

1. **CORRECTNESS**: if the public and secret keys are generated honestly, then the decryption of an encryption of a message M, using the proper public and secret keys, should reproduce M.
2. **INDISTINGUISHABILITY**: We want to get as close to OTP's as possible, so we want the encryption distributions of any two messages to be computationally indistinguishable.

This concept of indistinguishability is also called "**semantic security**". We define a game between a challenger C and an adversary A, to establish what we mean by semantic security:

0. We set a security parameter k.
1. C generates pk and sk, and sends pk to A.
2. A chooses $M_0 \neq M_1$ with $|M_0| = |M_1|$, and sends them to C.
3. C chooses a random bit b and encrypts M_b, and sends the corresponding ciphertext.
4. A sends back a bit b' and wins if $b = b'$

We say that a scheme is semantically secure if no **PPT** adversary can win with probability $1/2 + \varepsilon$, for any non-negligible ε.

In the style of computational indistinguishability, we also define semantic security in the following way:

Let $C_k(m)$ be the distribution of encryptions of a message m, when keygen uses the security parameter k. A protocol is semantically secure if, $\forall\, c \in \mathbb{N}, A \in \mathbf{PPT}$ and for all pairs of messages m_0, m_1, $\exists N_c \in \mathbb{N}$ such that if $n > N_c$, then

$$|Pr[A(C_n(m_0)) = 1] - Pr[A(C_n(m_1)) = 1]| < \frac{1}{n^c}.$$

2.1 Building PKE: El Gamal

Now, we build a public-key encryption scheme.

Secret Key: a randomly chosen $x \in \mathbb{Z}/p\mathbb{Z}$.
We pick a generator g and define $h = g^x$.

Public Key: the tuple (p, g, h)

Encryption: for a message M, we send the tuple $(g^r, h^r \cdot M)$,
for some randomly generated $r \in \mathbb{Z}/p\mathbb{Z}$.

Decryption: given an encrypted tuple (u, v),
we extract the message by computing $M = v/u^x$.

Correctness: We can see that the decryption produces

$$M \cdot h^r/(g^r)^x = M \cdot h^r/(g^x)^r = M \cdot h^r/h^r = M.$$

2.1.1 Semantic Security of El Gamal

We use a proof by contrapositive. Suppose that some adversary $A \in \mathbf{PPT}$ is able to distinguish between two El Gamal encryptions with probability $1/2 + \varepsilon$.

By our assumption, given two messages M_0 and M_1, A can determine b from the El Gamal encryption $E(M_b)$ with probability $1/2 + \varepsilon$.

Then, consider the following distinguisher D which attempts to distinguish between the distributions of (g, g^a, g^b, g^{ab}) and (g, g^a, g^b, g^r), modulo $\mathbb{Z}/p\mathbb{Z}$. Given a tuple (g_1, g_2, g_3, g_4) and a prime p, we do the following:

1. Provide A with a public key (p, g_1, g_2)
2. Receive two messages M_0, M_1 from A
3. Choose a random bit b and send A an encryption of M_b
4. receive a bit b' back from A:

(a) If $b' = b$, return 1

(b) If $b' \neq b$, return 0

Now we examine the two possible cases :

Case 1: $(\mathbf{g}, \mathbf{g^a}, \mathbf{g^b}, \mathbf{g^{ab}})$: In this case, the encryption we send is exactly the El Gamal encryption, so the adversary will be able to guess it with probability $1/2 + \varepsilon$. In particular, the probability that we return 1 in this case is $1/2 + \varepsilon$.

Case 2: $(\mathbf{g}, \mathbf{g^a}, \mathbf{g^b}, \mathbf{g^r})$: In this case, the encryption we send is completely random, so no adversary, no matter how powerful, will be able to guess it with probability greater than $1/2$. So, in this case, the probability we return 1 is at most $1/2$.

However, this means that our distinguisher can distinguish between the two cases with non-negligible probability, which breaks the DDH assumption. So, El Gamal encryption has semantic security as strong as the DDH assumption.

2.1.2 Homomorphic Encryption

Another important property of the El Gamal encryption scheme is that it is **homomorphic**. In general, this means that $E(a) \otimes E(b) = E(a \otimes b)$ or $E(a) \otimes E(b) = E(a \oplus b)$, where $\oplus$ and $\otimes$ denote modular addition and multiplication (mod p), respectively.

We call an encryption scheme **Fully Homomorphic** if it satisfies:

$$E(a) \otimes E(b) = E(a \otimes b), \quad E(a) \oplus E(b) = E(a \oplus b)$$

Tangentially related, a **re-randomizable encryption** is one where, given only an encryption $E(x)$ and the public key, we can convert it to a new encryption $E(x)'$, which is still an encryption of x, but is not the same– as if x had been decrypted and then re-encrypted with new randomness.

El Gamal turns out to be multiplicatively homomorphic, with

$$E(M_1) \otimes E(M_2) = (g^a, h^a \cdot M_1) \otimes (g^b, h^b \cdot M_2) = (g^{a+b}, h^{a+b} \cdot M_1 \otimes M_2) = E(M_1 \otimes M_2),$$

and it is re-randomizable, as we may use the public key to produce g^r, h^r and convert

$$(g^a, h^a \cdot M_1) \mapsto (g^{a+r}, h^{a+r} \cdot M_1).$$

2.2 Other important security notions

There are a couple other important security notions for encryptions:

2.2.1 Chosen Ciphertext Attack (CCA1)

This attack is also known as the "Lunchtime Attack" and is defined by the following game between a challenger C and an adversary A:

1. C sends a public key
2. repeat polynomially many times:
   (a) A sends a ciphertext
   (b) C decrypts and sends back the result
3. A chooses two messages (M_0, M_1)
4. C chooses a random bit b and sends back an encryption of M_b
5. A sends back a bit b', and wins if $b = b'$.

The significance of this game is that seeing previous message-ciphertext pairs should not improve an adversary's ability to distinguish later ciphertexts.

2.2.2 CCA-2

We play the following game between a challenger C and adversary A:

1. C sends a public key
2. repeat polynomially many times:
   (a) A sends a ciphertext
   (b) C decrypts and sends back result
3. A sends two messages M_0, M_1
4. C chooses a random bit b and sends back $C^* = E(M_b)$
5. polynomially many times:
   (a) A sends a ciphertext $CT' \neq C^*$
   (b) C decrypts CT' and sends back the result
6. A sends back a bit b', and wins if $b = b'$.

Note that homomorphic encryption schemes cannot pass this game, since A would be able to choose $CT' = E(M_2) \otimes C^*$, from which it would get back $M_2 \otimes M_b$. Then, it could simply figure out b by computing $M_2 \otimes M_0$ and $M_2 \otimes M_1$.

3 Trapdoor Permutation Encryption

One last form of public key encryption that we will discuss is trapdoor encryption.

Recall that a Trapdoor function is a function that is easy to compute and hard to invert, in the same sense as a 1WF. However, we have an addition that there is some "trapdoor", some piece of information (essentially the inverse of the function) which makes the function easy to invert. Additionally, recall that to make this feasible, we have to have an (exponential) *family* of permutations and trapdoors (f, f^{-1}), so that any **PPT** adversary only knows the trapdoor of any randomly selected permutation with negligible probability.

The encryption scheme works as follows:

Keygen: Choose a random pair (f, f^{-1}) from the family of trapdoor permutations. The secret key will be inverse f^{-1}, and the public key will be the permutation f, which operates on strings of length n, along with some random string p of length n, for some security parameter n.

Encryption: We encrypt a bit b using some randomness x as:

$$E(b, x) = (f(x), \langle p, x\rangle \oplus b).$$

Decryption: We decrypt a pair (u, v) as:

$$D(u, v) = \left(\langle p, f^{-1}(u)\rangle \oplus v, f^{-1}(u)\right).$$

Correctness can be checked easily, as:

$$D(E(b, x)) = D(f(x), \langle p, x\rangle \oplus b) = \left(\langle p, f^{-1}f(x)\rangle \oplus \langle p, x\rangle \oplus b, f^{-1}f(x)\right)$$

$$= (\langle p, x\rangle \oplus \langle p, x\rangle \oplus b, x) = (b, x).$$

We can also see that this is semantically secure, as any adversary who is able to distinguish an encryption of 0 from 1 with non-negligible advantage would have to predict the hardcore bit $\langle p, x\rangle$ of f with non-negligible advantage, which is a contradiction to hardcore bits.

But there is a problem with this scheme, which is efficiency. Recall that in general, security parameters have to be very large, so n would be quite large. And, since f is a permutation, all the ciphertexts would have length n, but we encrypt each bit individually. So, the final ciphertext would be n times as large as the original message. Furthermore, we would have to generate an n-bit random string for every single bit we want to encrypt! The problems are:

1. We have to generate too much randomness to get this to work
2. The ciphertexts are too long for this to be feasible over multiple uses

So, our solution is:

0. Share a PRG between the two communicating parties
1. Generate a random seed
2. Use this trapdoor encryption to share the seed
3. Use the PRG to generate randomness
4. Communicate using OTP's, with the PRG outputs.