CSE 594 : Modern Cryptography 30 March 2017

Lecture 1: Public Key Encryption:II

Instructor: Omkant Pandey Scribe: Gangabarani Balakrishnan, Subathra Vijayakumar

1 (Weak) Indistinguishable security for PKE

Definition :

A Public Key Encryption (PKE) is said to be secure if for all non-uniform PPT D there exists a negligible function μ such that for all $n \in N$, for all pair of messages m_0 and m_1 in M such that $|m_0| = |m_1|$, D distinguishes the following distributions with at most $\nu(n)$ advantage.

$\{(p_k, s_k) \leftarrow Gen(1^n) : (p_k, Enc(p_k, m_0))\}$
$\{(p_k, s_k) \leftarrow Gen(1^n) : (p_k, Enc(p_k, m_1))\}$

The definition of Weak Indistingushable security for PKE states that the above two distributions should be computationally insdistinguishable for any pair of messages m_0 and m_1 and any non uniform PPT D, the only difference is the message m_0 and m_1.

2 DDH Problem

Definition : $\{x \leftarrow Z_p^*, y \leftarrow Z_p^* : (g^x, g^y, g^{xy})\} \approx_c \{x \leftarrow Z_p^*, y \leftarrow Z_p^*, z \leftarrow Z_p^* : (g^x, g^y, g^z)\}$

DDH problem states that the above two tuples are completely indistinguishable. $Z_p = \{0, 1, ..., p-1\}$ where p is a very large prime and Z_p^* contains all non-zero elements in above set such that $gcd(x, p) = 1$. g^{xy} which is dependant of x and y looks totally indistinguisable from a completely random number g^z.

Since $|Z_p^*|$ is $p-1$ which is not prime makes the problem easier in some cases and taking discrete logs also becomes easy. The way around was to work with safe primes. We work with prime order sub-group of Z_p^* by picking a safe prime $p = 2q + 1$ where q is also prime and $g = x^2$ for a random $x \in Z_p^*$. In general you can work with any prime order subgroup of Z_p^*.

Let G_q is the group generated by $g = \{g^0, g^1, ..., g^{q-1}\}$ and size of the group is q. So now instead of picking up elements directly from Z_p^*, if you pick up x and y from G_q and then compute the values for the tuple the problem becomes hard.

Definition :

Let G be a group of prime order q and $g \in G$ be a generator

$\{x \leftarrow Z_q, y \leftarrow Z_q : (g^x, g^y, g^{xy})\} \approx_c \{x \leftarrow Z_q, y \leftarrow Z_q, z \leftarrow Z_q : (g^x, g^y, g^z)\}$

g^x means applying the group operations x times.

There are many work around for this other than using prime order subgroups. For example, we can pick up group of prime number points in elliptic curve and then use those values to compute the tuple.

3 ElGamal Public-Key Encryption

$ElGamalScheme$: Pick G like before such that the DDH assumption holds. G is a prime order group because its order q is a prime number. The description of G and q are known to everyone.

M is the message space and we encrypt elements only in the group. So message space is $M = G$.

$Gen(1^n)$:
Pick up $g \leftarrow G$ and $x \leftarrow Z_q$ and compute $h = g^x \in G$. If you compute h and give h and g to someone they cannot compute x (by discrete log assumption). Since G is a prime order group x is unique. Outputs public key p_k and secret key s_k where
$p_k = (g, h)$ and $s_k = x$

$Enc(p_k, m)$:
Choose a random r from Z_q and output $(g^r, m.h^r)$
Here c_1 is g^r and c_2 is $m.h^r$ where $h = g^x$ so c_2 is $m.g^{xr}$

$Dec(s_k, c)$:
$c = (c_1, c_2)$ and $m = c_2/c_1^x$
$m = c_2 * Inv(c_1^x)$
Take the secret key x and apply group operations x times on c_1. Now apply group operation on c_2 and Inverse of c_1^x. $Inv(c_1^x)$ can be computed using Fermat's Little theorem or Extended Eucledian theorem.

$Correctness$:
$m = m.h^r/g^{rx}$ and we know that $h = g^x$. So substituting h we get
$m.g^{xr}/g^{rx}$ Here g^{xr} gets cancelled leaving us with m. Hence the correctness.

4 Security of ElGamal Scheme

We are now going to prove that ElGamal scheme is secure assuming that the DDH assumption holds.

$\{(p_k, s_k) \leftarrow Gen(1^n) : (p_k, Enc(p_k, m_0))\}$
$\{(p_k, s_k) \leftarrow Gen(1^n) : (p_k, Enc(p_k, m_1))\}$

Let D be the PPT algorithm which can tell apart encryption of m_0 from m_1. We have to show that for all m_0 and $m_1 \in G$ these two distributions are indistinguishable. This could be done using Hybrid approach. So we start from the first one and slowly go to the second one. Hybrid experiments in the middle kind of give you security. You will see that when you reach the second hybrid the prediction advantage is already satisfied. When we are doing hybrid approach at each step we are calculating what is the difference between the two distribution.

$Hybrid-0$: $\{(p_k, s_k) \leftarrow Gen(1^n) : (p_k, Enc(p_k, m_0))\}$
So we start with this distribution, g and h are the public key and c_1 is g^r and c_2 is $m_0.g^{xr}$. So the tuple is $\{g, h, g^r, m_0.h^r\}$ which is equivalent to $\{g, h, g^r, m_0.g^{xr}\}$ because $h = g^x$.

$Hybrid-1$: Replace g^{xr} with a random element z
$\{g, h, g^r, m_0.g^z\}$

How to prove that these hybrids are indistinguishable?
If we have an adversary D that can distinguish $Hybrid-0$ and $Hybrid-1$ then we can use it to break the DDH assumption.

Let D' be an adversary that can break the DDH assumption.D' gets the input (g, g^x, g^y, g^α) where α is either xy or z. So D' just multiplies the last element with the message m_0 and sends $(g, g^x, g^y, g^\alpha.m_0)$ to D. D' outputs whatever D outputs. So if α is xy D is in $Hybrid-0$ else in $Hybrid-1$. If D tells $Hybrid-0$ and $Hybrid-1$ apart then D' tells DDH apart. $(i.e)$ If D wins between $Hybrid-0$ and $Hybrid-1$ D' is winning DDH with same probability and same advantage.

Prediction advantage definition is already satisfied. This is because in the tuple $\{g, h, g^r, m_0.g^z\}$, z is not anywhere in the tuple except the last element and hence it is completely random which implies that the tuple is also completely random.So there is no information about m_0 and no one can decrypt it even in exponential time. So the probabilty of you guessing if its m_0 or m_1 is exactly half $1/2$. Distance between $Hybrid-0$ and $Hybrid-1$ is negligible.

Similarly we can show that $Hybrid-2$ and $Hybrid-3$ constructed using m_1 similar to m_0 are indistinguishable. So the total distance is two times negligible.

5 RSA Encryption

The correct way of encrypting in RSA function is to first encrypt a random element then apply hardcore predicate with the given message we are trying to encrypt.
For an arbitrary one way permutation or one way function, we can't predict which bit is hardcore. In RSA, every bit is a hardcore bit.

The public key $pk = (N, e)$
where e is between 1 and $\phi(n)$ prime number. The value ed is congruent to 1 mod $\phi(n)$.

The Message Space $\mathcal{M} = \mathbb{Z}_n^*$

Enc(pk, m) for $pk = (N, e)$ outputs $f_{N,e}(m) = m^e mod N$

Dec(sk, c) for $sk = (N, d)$ outputs $c^d mod N$

The problem with the above scheme is that there is no randomness involved. In SSL version 3 which is only IND-CPA secure, we did not consider any scenario where the adversary asks chal-

lenger specially constructed cipher texts to decrypt. The PKCS - I standard in SSL V3 uses RSA. They came up with their own method where they took the message and padded it with some random text. Then they encrypted it using the system.
Though it was random it had some provision for error messages, it will take a message, pad it and convert it into a proper length message and convert it into integer.
While decrypting it will first convert the integer into proper PKCS format check if it is correct format and extract the message from blocks. In case there is format error, The server will tell client there is some error/ packet corruption occurred.
Though this seems an innocuous attack, The scheme does not consider these types of attacks, to protect against such an attack you should have CCA secure encryption. This information enables you to decrypt the entire message. This attack is called bleichenbacher attack. It only requires about 8000 cipher text queries to decrypt the entire message. An ability to learn even a single bit information about encryption is dangerous.
The approach to encryption considered in the previous class was inefficient in that, It allowed only one bit to be encrypted. The more efficient method is $RSA - OAEP^{+}$.

6 RSA Signature

The RSA can also be seen as a digital signature which inverts the role of the encryption key and decryption key. The encryption key is used as the verification key and decryption key is the secret signing key. The Key generation remains the same, message space remains the same. The Verification key $vk = (N, e)$
where e is between 1 and $\phi(n)$ prime number. The value ed is congruent to 1 mod $\phi(n)$.

The Message Space $\mathcal{M} = \mathbb{Z}_n^*$

Sign(sk, m) for $sk = (N, d)$ outputs $\sigma = m^d \bmod N$

Verify(vk, m, σ) for $vk = (N, e)$ outputs 1 iff $\sigma^e = m \bmod N$

The verification just checks whether m = $\sigma^e mod N$. The Signature is deterministic but that is not a problem in the signatures. The question here is whether the signatures can be forged?
The answer is it can be, If we use this naive mechanism. We just need to choose some random $\alpha = \mathbb{Z}_n^*$. We already know the verification key is $\beta = \alpha^e$. The signature becomes α and the message becomes β.

$$\beta = \alpha^e$$
$$\beta^d = \alpha^{ed}\ mod N$$
$$\beta^d = \alpha\ mod N$$

Hence if we use this scheme we face the issue of existential forgeries, which are forgeries that simply exist and are easy to pick. But we may not be able to pick the message. This scheme provides trapdoor permutation functionality but it doesn't provide encryption scheme or signature scheme.

There are other ways to use RSA signature scheme, the RSA signature scheme is not the most efficient because of the long key sizes.

7 Attacks on the RSA Function

Here are some of the attacks on the RSA Function. Sometime ago Before We fully understood the RSA, People thought that using a smaller exponent such as $e = 3$ will make the verification step faster, in case of encryption the encryption function faster. This is often a big problem.
Example: Lets say you have a message m, which is to be broadcast to 3 people, all of them have different RSA public keys, they choose their own factors and their secret keys are completely different. But they choose a common exponent $e = 3$ to make verification and encryption faster.
In this scenario, If we are going to use naive RSA scheme, We are broadcasting the following cipher texts.

$$C_1 = m^3 \ mod N_1$$
$$C_2 = m^3 \ mod N_2$$
$$C_3 = m^3 \ mod N_3$$

All the adversary needs to do is multiply the values together C_1, C_2, C_3. Suppose that the N_1, N_2, N_3 are co-primes they don't have any common factors, which is highly likely otherwise m will be easy to calculate. Then by Chinese remainder theorem the product will be of the form.

$$C' = m^3 \ mod N_1 N_2 N_3$$

The value of $N_1N_2N_3$ becomes much larger than m^3. Hence the m^3 becomes $\sqrt[3]{C'}$. Modulus has no role to play. This attack can be applied in scenarios where we have as many messages as value of e. This attack cannot be used where the value of e is very high. However low exponents carefully used along with padding is still secure. There are also other ways of improving the speed of verification/encryption by using special forms of exponent $e = 2^16+1$. Instead of multiplication, we are just shifting bits.
Another way to improve the efficiency is to keep the public exponent large but by keeping the secret key small. The key will be small enough that process is more efficient but big enough that it can't be guessed. The value of $d > N^{0.292}$ otherwise it can be easily guessed.

8 LWE-based Public Key Encryption

$$S = (S_1, S_2, S_3, ..., S_n) \epsilon \mathbb{Z}_q^n$$

where q ≥ 2 and S_x be solutions to system of linear equations like below

$$a_{11}S_1 + a_{12}S_2 + a_{13}S_3 + ... + a_{1n}S_n = b_1 + e_1$$
$$.....$$
$$a_{m1}S_1 + a_{12}S_2 + a_{13}S_3 + ... + a_{mn}S_n = b_m + e_m$$

Where m is a polynomial. These equations are easy to solve using Gaussian elimination in case there are no errors. In case there are errors that are small enough that there is a unique solution to the set of system equation, but large enough that if we do Gaussian elimination these errors are going to add up and create too much noise. The errors are picked over Gaussian distribution with a standard deviation of αq. where $\alpha << 1$.
When we start with this problem we pick a modulus q, we pick a security parameter n, we pick an $/alpha << 1$ such that $\alpha q > \sqrt{n}$.

8.1 LWE Instance

choose a random (column) vector $s \xleftarrow{\$} \mathbb{Z}_q^n$

choose a random matrix of coefficients $A \xleftarrow{\$} \mathbb{Z}_q^{MXN}$

choose a Gaussian error vector $e \xleftarrow{\mathcal{X}} \mathbb{Z}_q^n$
where $\mathcal{X}$ is a Gaussian distribution over $\mathbb{Z}$ with parameter αq.

Let b = A.s + e

s is the system of equations. The LWE instance is: (A, b)

8.2 Decisional LWE Assumption

It is hard to distinguish an LWE pair from a random instance.

$$(A, b) \approx_c (A, u)$$

where $u \epsilon \mathbb{Z}_q^m$ is a random column vector.

Key Generation:
choose the same parameters as before,

$$s \xleftarrow{\$} \mathbb{Z}_q^n, A \xleftarrow{\$} \mathbb{Z}_q^{MXN}, e \xleftarrow{\mathcal{X}} \mathbb{Z}_q^n, b = A.s + e$$

$$pk = (A, b),\ sk = s$$

Encryption:
pick a row-vector of bits $X \xleftarrow{\$} 0, 1^m$ and output is,

$$c = xA, c' = xb + bit.\ \frac{q}{2}$$

Hence if the bit is 0, the value of $c' = xb$ else it will be $c' = xb + \frac{q}{2}$.

Decryption:

$$\begin{aligned} c' - c.s &= \left(xb + bit.\ \frac{q}{2} \right) - xAs \\ &= \left(xb + bit.\ \frac{q}{2} \right) - xb + xe \approx bit.\ \frac{q}{2} \end{aligned}$$

Parameters:

$$n^2 \leq q \leq 2n^2, m = 1.1n \log q, \alpha = 1/(\sqrt{n} \log^2 n)$$

Correctness: if not for the error term, the value would be either 0 or $\frac{q}{2}$.
The error is adding at most m independent normally distributed variables whose standard deviation is $\sqrt{m\alpha q} < q/\log n$.
The probability that it goes over q/4 is negligible.

Security: (LWE + LHL)
Game 0: Real pk = LWE instance = (A, b)
Game 1: change pk to a random instance = (A, u)
Game 2: change bit from 0 to 1 (one-time pad, due to LHL)
Game 3: change pk back to LWE instance.
where LHL is Left over hash lemma, and we use hybrid arguments to prove security. We need the LHL in the second step. We finish the argument using prediction advantage or use game theory where we use LWE instance.