CS 7880 Graduate Cryptography September 10, 2015

Lecture 15:

Lecturer: Daniel Wichs *Scribe: Mehraneh Liaee*

1 Topic Covered

- Threshold Cryptography
- Trapdoor Permutation
- RSA

2 Threshold Cryptography

Let's remind ElGamal public key encryption-decryption scheme from last session. In this scheme, we have a secret key, $sk = x \in \mathbb{Z}_q$, and a public key, $pk = f = g^x$, where g is the group generator. To encrypt a message m, we pick a random $y \in \mathbb{Z}_a$ and use the following encryption function:

$$c = Enc(pk, m) = (h = g^y, z = f^y m)$$

To decrypt a cipher text, we use the following decryption function:

$$Dec(sk, c = (h, z)) = \frac{z}{h^x} = \frac{f^y.m}{g^{xy}} = m$$

Since secret keys are very important and the whole secrecy of scheme is dependent on secrecy of the secret key, we need to protect it. One idea is not keeping the whole secret key in one machine, because once that machine is hacked, the secret key will be recovered by the adversary and everything is lost. The idea of threshold cryptography is to spread secret key over several machines in a way that if some subset of machines get broken, then the security of whole system is not broken. We saw the same idea in secret sharing. Here, we want to use the idea from ElGamal scheme to build such a scheme.

Assume we have n devices, and we want that even if the attacker breaks $n-1$ of devices, he does not learn anything.

- Each device i has x_i such that $x = \sum_{i=1}^{n} x_i \mod q$.
- To decrypt a cipher text $c = (h, z)$, we need to compute h^x without having the whole $sk = x$ in one device. Thus, each device sends $s_i = h^{x_i}$, and it yields that $h^x = \prod s_i$. Since we get h^x, we are able to decrypt c, and recover $m = z/h^x$.

Now, assume an attacker corrupts $n-1$ of devices and learns $x_i : i \neq j$ for all devices but one device j. Thus, the adversary can compute $h^{x_i} : i \neq j$. Assume that a message m and its cipher text $c = (h, z)$ is given to the adversary. Adversary is able to recover $h^x = \frac{z}{m}$. He is also able to recover $s_j = h^{x_j} = \frac{z}{m \prod_{i \neq j} h^{x_i}}$, but he does not learn anything about x_j. Thus the security of system is still not corrupted. This means even if adversary sees one message and its cipher text, and corrupts $n-1$ device, he won't be able to learn about the whole secret key.

3 Trapdoor Permutation or TDP

Trapdoor permutation f is a OWP with additional requirements, some additional secret key which associates with function f, if you have the secret key you can invert function f.

- $(pk, sk) \leftarrow Gen(1^n)$
- $f_{pk} : D_{pk} \to D_{pk}$ is a permutation:
  - $f_{pk}(x)$ is computable in polynomial time in terms of n (efficiently computable).
  - Sampling x from domain D_{pk}, $(x \leftarrow D_{pk})$, can be done in polynomial time in terms of n.
- **correctness:** $Inv_{sk}(y)$ is the inverse function of f_pk. If we have the secret key sk of function, then we can invert it efficiently.

$$Inv_{sk}(f_{pk}(x)) = x \text{ or } Inv_{sk} = f_{pk}^{-1}$$

- **security:** $\forall$ PPT attacker A:

$$\Pr[A(pk, y) = x : (pk, sk) \leftarrow Gen(1^n), x \leftarrow D_{pk}, y = f_{pk}(x)] = negl(n)$$

Remark 1 *The only difference of TDP from OWP is having secret key and inversion property. There is a question which asks if we have a TDP, is that a good candidate for public key encryption scheme? Then answer is that we can not use $f_{pk}(x)$ directly in a public key encryption scheme. The reason is because $f_{pk}(x)$ is deterministic and there is no randomnessss in the procedure. If we encrypt the same message twice, we will see the same ciphertext twice, which is not secure. The other problem is that even if we encrypt a random message, we won't be able to recover message but maybe we can learn some bits of it. So TDP does not say anything about hiding any information about the message. In the following, we try to fix these problems by using hard core bit of the function .*

3.1 Encryption from TDP

The idea for constructing encryption scheme from TDP is we don't use x as the message, we use it as the randomness. Consider the case that message m is only one bit. We have following encryption and decryption functions:

- $c = Enc(pk, m) : x \leftarrow D_{pk}, c = (f_{pk}(x), hc(x) \oplus m)$

- $m = Dec(sk, c = (y, b)) : hc(Inv_{sk}(y) \oplus b)$

Remark 2 *From the above construction, the correctness of the scheme can be easily verified, and the security follows from Goldreich-Levin theorem. To generalize this scheme to messages of longer bit, you can apply the same function for every bit of the message separately.*

Remark 3 *Note that instead of using hard core bit, one can use Random Oracle model by replacing $RO(x)$ with $hc(x)$.*

4 RSA

Instead of looking at sequence order groups, we look at composite order groups. We are going to look at group $\mathbb{Z}_N$, where N is the product of two prime numbers p, q. We have this assumption that factoring is hard, the definition comes in the following.

DEFINITION 1 **Factoring assumption.** For all PPT attacker A,

$$\Pr[A(N) = p, q : p, q \leftarrow \{\text{n-bit primes}\}, N = pq] = negl(n)$$

$\diamondsuit$

We first try to understand the structure of group $\mathbb{Z}_N^*$. We know the order of this group is $\varphi(N) = (p-1)(q-1)$, which is the number of integers less than N which are relatively prime according to N. Chinese remainder theorem tells that $\mathbb{Z}_N^*$ as a group is isomorphic to $\mathbb{Z}_p^* \times \mathbb{Z}_q^*$, and we denote it by $\mathbb{Z}_N^* \cong \mathbb{Z}_p^* \times \mathbb{Z}_q^*$. Also, from Chinees remainder theorem, we have $\mathbb{Z}_N$ as a ring is isomorphic to $\mathbb{Z}_p \times \mathbb{Z}_q$. Using Chinese remainder theorem, for any element a in $\mathbb{Z}_N$, we can think of it as a tuple (a_p, a_q), where $a_p = (a \mod p)$ and $a_q = (a \mod q)$. Then instead of doing operation in $\mathbb{Z}_N$, we separately do operation in $\mathbb{Z}_p$ and $\mathbb{Z}_q$, and using the isomorphism, recover the result in $\mathbb{Z}_N$.

Moreover, there are elements $e_p, e_q \in \mathbb{Z}_N$, such that their corresponding tuples are of the form $(1, 0)$ and $(0, 1)$ in $\mathbb{Z}_p * \mathbb{Z}_q$. Thus, whenever we have $(a_p, a_q) \in \mathbb{Z}_p * \mathbb{Z}_q$, we can recover a as follows:

$$a = a_p.e_p + a_q.e_q \in Z_n$$

We call these elements e_p, e_q basis elements. The same thing holds for $\mathbb{Z}_N^*$.

Now, we want to look at function $f_e(x) = (x^e \mod N)$, where $x \in \mathbb{Z}_N^*$. We want to show that this function is a permutation and one-way. It turns out that for some value of e this function is a permutation.

Observation 1 *If $gcd(\varphi(N), e) = 1$, then function $f_e(x) = (x^e \mod N)$ is a permutation.*

Observation 2 *If $gcd(\varphi(N), e) = 1$, then there exists an integer d such that $d.e = (1 \mod \varphi(N))$ and $f_d(f_e(x)) = x \mod N$.*

Remark 4 *Note that d is like a secret key, if we know $\varphi(N)$ or p and q then we can easily find d and ?nvert function $f_e(x)$, however relying on assumption of hardness of factoring problem, without having $\varphi(N)$ or p and q, we cannot obtain d efficiently, and we cannot invert $f_e(x)$ efficiently. It is also sufficient to pick $e = 3$.*

Now, we are ready to construct the RSA scheme.

4.1 RSA construction

Definition 2 RSA construction

- Key Generation: $(pk = (N, e), sk = d) \leftarrow RSAGen(1^n)$, where $gcd(e, \varphi(N)) = 1$.
- RSA Assumption: $\forall$ PPT attacker A

$$\Pr[A(N, e, y) = x : (N, e, d) \leftarrow RSAGen(1^n), x \leftarrow \mathbb{Z}_N^*, y = (x^e \mod N)] = negl(n)$$

- Encryption: $f_{pk}(x) = (x^e \mod N)$ and $f_{pk} : \mathbb{Z}_N^* \to \mathbb{Z}_N^*$
- Decryption: $f_d(y) = (y^d \mod N)$

$\diamondsuit$

Remark 5 *Sampling from $\mathbb{Z}_N$ is easy in polynomially, how about sampling from $\mathbb{Z}_N^*$? The probability of choosing a number which is not in $\mathbb{Z}_N^*$ but it is in $\mathbb{Z}_N$ is very tiny. So whenever we sample a number from $\mathbb{Z}_N^*$, we test it in polynomial time whether its relatively prime to N or not, if it was not in $\mathbb{Z}_N^*$, we ignore it and sample again.*

Remark 6 *How good and secure is RSA?*

- *RSA is insecure, if factoring is not hard, which means giving N, we can find p and q such that $N = pq$. Having p and q, we can find φN and d respectively.*
- *RSA is insecure, if we have $\varphi(N)$. In fact, we can show that finding $\varphi(N)$ is as hard as factoring. Having $\varphi(n)$, we can calculate $t = N - \varphi(N) - 1 = p + q$. Then, we have the following equality:*

  $$q^2 - tq - N = 0$$

  Thus, having N and $\varphi(N)$, we can obtain t and q respectively.
- *RSA is insecure, if we have d. In fact, having d, $\varphi(N)$ can be recovered doing more work.*