CS 282A/MATH 209A: Foundations of Cryptography — Prof. Rafail Ostrovsky

Lecture 7

Lecture date: Monday, 28 February, 2005 — *Scribe: M.Chov, K.Leung, J.Salomone*

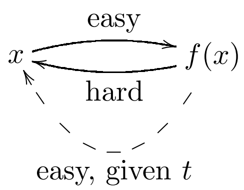
1 Oneway Trapdoor Permutations

Recall that a oneway function, f, is easy to compute, but hard to invert. Formally, for all PPT adversaries A, there is a c such that or eventually all n,

$$\Pr\left[A(f(x)) \in f^{-1}f(x)\right] < \frac{1}{n^c}$$

with the probability taken over $|x| = n$ and coin flips of A.

A oneway, trapdoor function is a oneway function f, which becomes easy to invert when given some extra information, t, called a trapdoor.



We formalize this as follows.

Definition 1 *A oneway trapdoor function is a parameterized family of functions* $\{f_k : D_k \to R_k\}_{k \in K}$*, with* K*,* D_k*, and* $R_k \subseteq \{0,1\}^*$.

1. *Key, trapdoor pairs are PPT sampleable: there is a polynomial* p *and PPT algorithm* **GEN** *such that* $\mathbf{GEN}(1^n) = (k, t_k)$*, with* $k \in K \cap \{0,1\}^n$*, and* $|t_k| \leq p(n)$*. Call* k *a key, and* t_k *the trapdoor for* f_k.
2. *Given* k*, the domain* D_k *is PPT sampleable.*
3. f_k^{-1} *is computable, given a trapdoor* t_k*: there is an algorithm* I*, such that* $I(k, t_k, f_k(x)) = x$*, for* $x \in D_k$.
4. *For all PPT* A*, the following is negligible:*

$$Pr\left[A(k, f_k(x)) \in f_k^{-1} f_k(x)\right]$$

where k *is sampled by* **GEN***, and the asymptotics are relative to the security parameter.*

In this definition, (1) is saying that we can randomly generate a function from the family, and its trapdoor. The the size of the trapdoor information must be polynomial in the size of the key. (3) says that an instance f_k is invertible, given its description k, and its trapdoor t_k. (4) says that $\{f_k\}$ is a oneway family. For clarity, we will often let the k be implied, and write (f, f^{-1}), instead of (k, t_k).

Note that it is important to use a family of functions. If we try to make the above definition for a single function, (4) will fail. There is always some adversary A, with a description of the trapdoor t, and the inverter I, "hard-wired" into its description. This adversary will always be able to invert.

2 Public Key Encryption

A public key encryption scheme (say, for entity A) consists of three algorithms, **KEYGEN** for key generation, and **ENC** and **DEC**, for encryption and decryption, respectively. Given a security parameter, 1^n, **KEYGEN** should return two keys, PK and SK. The idea is that PK is made public, and is used by any other entity B, as input to **ENC**, to encrypt a message for A. SK is kept secret by A, and is used in **DEC** to decrypt a ciphertext, and recover the original message. We will define the semantic security of this system so that no adversary E can recover the message, even with knowledge of the public key PK.

$$\begin{aligned} (PK, SK) &\leftarrow \mathbf{KEYGEN}(1^n, r) \\ c &\leftarrow \mathbf{ENC}(PK, m, r) \\ m' &\leftarrow \mathbf{DEC}(PK, SK, c) \end{aligned}$$

Of course, in the above procedure, we want $m' = m$, so that we recover the original message m. We demand that the scheme be *correct*: if (PK, SK) is generated by **KEYGEN**, then for all messages m,

$$\mathbf{DEC}\big(PK, SK, \mathbf{ENC}(PK, m, r)\big) = m. \tag{1}$$

To define semantic security, consider the following game. Challenger uses **KEYGEN** to generate a key pair (PK, SK) and publishes PK. Adversary, given PK, picks distinct messages m_0 and m_1, of equal length, and sends them to Challenger. Challenger picks a random bit b, and then sends to Adversary the ciphertext $c = \mathbf{ENC}(PK, m_b)$.

Challenger		Adversary
$(PK, SK) =$	$\longrightarrow$	PK
$\mathbf{KEYGEN}(1^n, r)$		
m_0, m_1	$\longleftarrow$	Pick $m_0 \neq m_1$ of equal length
b random		
$\mathbf{ENC}(PK, m_b, r)$	$\longrightarrow$	c
		Guess b

We say that the cryptosystem is secure if Adversary can then guess b with probability which deviates only negligibly from $\frac{1}{2}$.

Remark For this definition to work, we needed **ENC** to be probabilistic. Otherwise, Adversary could simply compute $\mathbf{ENC}(m_0)$ and $\mathbf{ENC}(m_1)$, and compare them to c, thus determining b.

2.1 Example: PK Cryptosystem from Oneway Trapdoor Permutations

A semantically secure, public key cryptosystem can be constructed from a oneway, trapdoor permutation. The algorithms are as follows[1].

$\mathbf{KEYGEN}(1^n, r)$:

1. **compute** $(f, f^{-1}) := \mathbf{GEN}(1^n)$.
2. Pick a string p, uniformly at random, for computing hard-core bits.
3. **return** $PK = (f, p)$, $SK = f^{-1}$.

Encryption and decryption are done bit-wise on the plaintext and ciphertext.

$\mathbf{ENC}((f, p), m, r)$:

1. Pick x at random from the domain of f.
2. **compute** $c := (p \cdot x) \oplus m$.
3. **compute** $d := f(x)$.
4. **return** ciphertext (c, d).

[1] Recall that $p \cdot x = \bigoplus_{1 \leq i \leq n} p[i]x[i]$, where $|p| = |x| = n$.

DEC$((f,p), f^{-1}, (c,d))$:

1. **compute** $x := f^{-1}(d)$.
2. **compute** $m := (p \cdot x) \oplus c$.
3. **return** m.

Clearly this cryptosystem is correct; it is also semantically secure. If an adversary could distinguish two messages m_0 and m_1, then by a hybrid argument, it could distinguish two messages m'_0 and m'_1, which differ in only one bit. We could then use this adversary to compute hard-core bit, $p \cdot x$, knowing only $f_s(x)$.

3 Some Cryptographic Assumptions

3.1 Finite, Abelian Groups

Recall that an abelian group is a collection of elements G, with a binary operation $\star$ on G, satisfying:

$$\begin{array}{ll} (\forall a,b,c \in G)\ (a \star b) \star c = a \star (b \star c) & \text{(Associativity)} \\ (\forall a,b \in G)\ a \star b = b \star a & \text{(Commutativity)} \\ (\exists 1 \in G)(\forall a \in G)\ 1 \star a = a & \\ (\forall a \in G)(\exists a^{-1} \in G)\ a \star a^{-1} = e & \end{array}$$

Call 1 the identity element of G, and a^{-1} the inverse of a. The order of a finite group G is the number of elements in the group, denoted $|G|$. A useful fact is that if $|G| = n$ then for any element a, $a^n = 1$.

We will usually be concerned with a specific type of abelian group: Call $g \in G$ a *generator* iff $G = \{g^n | 0 \leq n < |G|\}$. In case G has a generator, say that G is *cyclic*, and write $G = \langle g \rangle$.

We wish to generate finite, cyclic groups randomly. Fix a PPT algorithm **GROUP**, which samples a finite, cyclic group, given a security parameter 1^n. In other words, if

$$(G, p, g) \leftarrow GROUP(1^n),$$

then G is a (binary description of a) finite group, $p = |G|$, and g is a generator.

3.2 Discrete Logarithm Problem

Suppose we are given a cyclic group G, of order p, with generator g, and a group element $a \in G$. The Discrete Logarithm Problem is to find an integer k, such that $g^k = a$. In other words, to compute $k = \log_g(a)$. The Discrete Logarithm Assumption say that this is computationally hard.

Assumption 2 (DLA) *For any PPT algorithm A*

$$Pr\left[g^k = a : (G, p, g) \leftarrow \mathbf{GROUP}(1^n); a \stackrel{R}{\leftarrow} G; k \leftarrow A(G, p, g, a)\right]$$

is negligible in n.

Many financial transactions are done using a **GROUP** which returns $G = \mathbb{Z}_p$ for p a prime.

3.3 Decisional Diffie-Hellman Problem

The Decisional Diffie-Hellman Problem is similar to the Discrete Log Problem, except that one tries to distinguish to powers of a generator, rather than trying to compute a log. Suppose we are given a group G, of order p, with generator g. Then integers $x, y, z \in \mathbb{Z}_p^*$ are selected randomly. From this, two sequences are computed:

$$\langle G, p, g, g^x, g^y, g^z \rangle \qquad \text{(Random sequence)}$$
$$\langle G, p, g, g^x, g^y, g^{xy} \rangle \qquad \text{(DDH sequence)}$$

The DDH problem is to determine which sequence, Random or DDH, we have been given. The DDH Assumption is that the DDH Problem is hard.

Assumption 3 (Decisional Diffie-Hellman) *Let G be a sampled group of order p, with generator g. Pick $x, y, z \in \mathbb{Z}_p^*$ uniformly at random. Then it is asymptotically difficult (with respect to the security parameter), for a PPT adversary A to distinguish $(G, p, g, g^x, g^y, g^{xy})$ from (G, p, g, g^x, g^y, g^z).*

Remark The DDH assumption is stronger than the DLP assumption. Computing discrete logarithms would allow one to trivially distinguish g^{xy} from g^z, for a random z.

4 The ElGamal Public Key Cryptosystem

The security of the ElGamal cryptosystem is based on the difficulty of DDH and DLP. The algorithms are:

KEY(1^n):

1. **compute** $(G, p, g) :=$ **GROUP**(1^n).
2. Sample $x \in \mathbb{Z}_p^*$, uniformly at random.
3. **compute** $w := g^x$.
4. **return** $PK = (G, p, g, w)$, $SK = x$.

ENC$((G, p, g, y), m)$ (for $m \in G$):

1. Sample $r \stackrel{R}{\leftarrow} \mathbb{Z}_p^*$.
2. **compute** $c := w^r m, d := g^r$.
3. **return** ciphertext (c, d).

DEC$((G, p, g, y), x, (c, d))$:

1. **compute** $m := \frac{c}{d^{-x}}$.
2. **return** m.

To see that the cryptosystem is correct, compute $cd^{-x} = w^r mg^{-rx} = g^{xr} mg^{-rx} = m$. It is also secure, assuming the DDH assumption holds.

Theorem 4 *ElGamal is semantically secure, if the DDH assumption holds.*

Proof Suppose we have a PPT adversary A, which breaks ElGamal's semantic security. We can use it to construct an algorithm A', which solves the DDH problem. A' is given a sequence $\langle G, p, g, g_1, g_2, g_3 \rangle$ and must decide whether this is a Random Sequence or a DDH Sequence. A' will play the semantic security "game", using A's responses to identify the sequence, thus solving the DDH problem.

$A'(G, p, g, g_1, g_2, g_3)$:

1. **compute** messages $(m_0, m_1) := A(G, p, g, g_1)$.
2. Pick $b \in \{0, 1\}$ uniformly at random.
3. **compute** A's guess $b' := A(g_2, g_3 m_b)$.
4. **if** $b' = b$ **then return** 1 **else return** 0.

A' takes an input (G, p, g, g_1, g_2, g_3) (with G, p, g sampled). (G, p, g, g_1) is used as an ElGamal public-key, which is given to A. The adversary returns a pair of messages m_0, m_1, which it can distinguish. After selecting a random bit b, $(g_2, g_3 m_b)$ is returned to A, as a potential cipher-text. Then A returns b', its guess for b. If $b' = b$ we return 1, which we interpret as identifying the DDH sequence. Otherwise, we return 0, identifying the Random sequence.

Note that if we give A' the input $(G, p, g, g^x, g^y, g^{xy})$, then $(g_2, g_3 m_b) = (g^y, (g^x)^y m_b)$. This is a valid ciphertext encryption of m_b, with public key (G, p, g, g^x), and secret key x. Since A can distinguish m_0 from m_1, it will guess $b' = b$ correctly. In this case A' will output 1 with as high a probability as A can distinguish the messages.

On the other hand, if we give input (G, p, g, g^x, g^y, g^z) for independently chosen z, $g_3 m_b = g^z m_b$ will just be a random element of G. Thus $g^z m_0$ and $g^z m_1$ will have equal probability of appearing in the ciphertext. So A will not be able to guess b, and A' will output 0 with high probability (and output 1 with low probability).

Thus A' can solve the DDH problem with non-negligible probability, assuming that A can break the semantic security of ElGamal. ■