0368.4162: Introduction to Cryptography	Ran Canetti
Lecture 5	
05 December 2008	Fall 2008

Scribes: S. Nemzer, A. Porat

Topics

1. Warm-up theorem: Indistinguishably under multiple samples.
2. Block Ciphers.
3. Pseudo-random permutations/functions.
4. Feistel transform (from PRF to PRP).

1 Warm-up theorem: Indistinguishably under multiple samples.

Definition 1:

Two ensembles $D_1 = \{D_n^1\}_{n\in N}$ and $D_2 = \{D_n^2\}_{n\in N}$, are indistinguishable in polynomial time if for every probabilistic polynomial-time algorithm A

$\left|Prob_{x\in D_n^1}[A(1^n, x) = 1] - Prob_{x\in D_n^2}[A(1^n, x) = 1]\right| < \nu(n)$

for some negligible function $\nu(n)$.

Let D_1, D_2 be two indistinguishable ensembles, and consider a distinguisher A that has access to a box $\boxed{\text{D}}$ that, on each press of a button, transmits a value that is taken from distribution D.

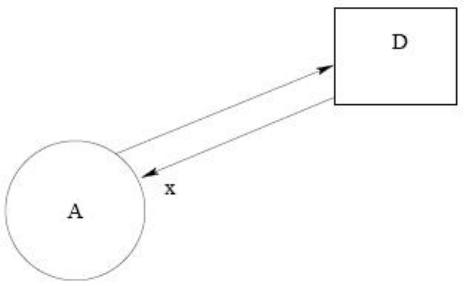

Figure 1: Oracle D

Definition 2:

For a distribution ensemble D=$\{D_n\}_{n\in N}$ let $A^{D_n}(1^n)$ denote the output of A, having input 1^n and access to the box $\boxed{D_n}$. Let A^D denote the ensemble $\{A^{D_n}(1^n)\}_{n\in N}$.

Theorem 1: If $D_1 \approx_c D_2$ then for any polynomial-time adversary A, $A^{D_1} \approx A^{D_2}$.

Remark 1: WLOG, we can restrict attention to A's that output one bit.

(Exercise: Prove that no generality is lost here.)

Remark 2: Let D_1, D_2 be distribution ensembles over {0,1}. Then $D_1 \approx_c D_2$ iff $D_1 \approx_s D_2$.

Proof of theorem 1: We prove theorem 1 using the hybrids technique. Let q(n) ($n \in N$) be the number of samples that A receives. Definitely, q(n) is bounded by the run-time of A and, therefore, q(n) is polynomial in n. Let $\mathcal{E} = \epsilon(t)$ be a distinguishing difference of A.

Define experiment E_i (the "i-th hybrid") : The first i queries by A are answered according to D_1, and the rest of the queries are answered according to D_2.

Let H_i be the probability that A outputs 1 in E_i.

By definition, E_0 describes the interaction of $A^{D_n^1}$ and E_q describes the interaction of $A^{D_n^2}$.

So, we have H_q - $H_0 > \mathcal{E}$. This means that there is an i, where $H_{i+1} - H_i \geq \frac{\mathcal{E}}{q}$.

We will use this observation as follows: Given n and A such that Prob[A^{D_1}] - Prob[A^{D_2}] > $\mathcal{E}$, we construct A' such that $Prob_{x \leftarrow D_1}[A'(x) = 1] - Prob_{x \leftarrow D_2}[A'(x) = 1] > \frac{\mathcal{E}}{q}$.

Construction of A': On input x,

- Choose $i \in \{1..q\}$

- run A as follows:

– for j<i, the j-th query of A is answered by $x_{j \leftarrow D_1}$

– for j=i, the j-th query of A is answered by x

– for j>i, the j-th query of A answered by $x_{j \leftarrow D_2}$.

The main observation is that:

– if x is drawn from D_1, then A sees experiment E_i.

– if x is drawn from D_2, then A sees experiment E_{i-1}.

More precisely, we have:

$Prob_{x \leftarrow D_1}[A'(x) = 1] - Prob_{x \leftarrow D_2}[A'(x) = 1] =$

$= \sum_{i=1}^{q} (Prob_{x \leftarrow D_1}[A'(x) = 1 \mid i = i_0] \cdot Prob[i = i_0] -$

$Prob_{x \leftarrow D_2}[A'(x) = 1 \mid i = i_0] \cdot Prob[i = i_0]) =$

$\frac{1}{q} \cdot \sum_{i=1}^{q} (H_i - H_{i-1}) = \frac{1}{q} \cdot (H_q - H_0) \geq \frac{\mathcal{E}}{q}$

We got an algorithm A' that can distinguish between D_1 and D_2 at probability greater then $\mathcal{E}$, in contradiction to the assumption.■

2 Block Ciphers

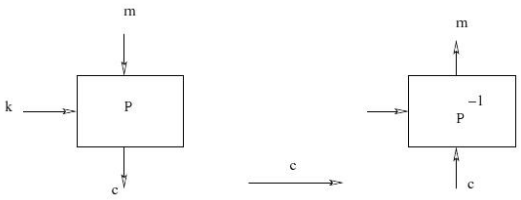

Figure 2: Block Cipher

In lecture 3 we have seen how stream ciphers are used to encrypt messages between parties that have a shared key. However, stream ciphers have a significant usability drawback: The parties need to keep local state for encryption and decryption. Moreover, the states of the sender and receiver need to be perfectly synchronized. This is problematic in many cases, e.g when the receiver is not always on line, or, when information packets get out of order (or might get lost). In such cases (and others) it is convenient to allow for "stateless" encryption and decryption, where the encryption process depends only on the ciphertext and the key. A block cipher provides this functionality.

Definition 3:

A block cipher is a keyed invertible permutation, namely a family of functions

$P_k : \{0,1\}^a \times \{0,1\}^n \to \{0,1\}^n$,

where for any k, $k \in \{0,1\}^a$, $P(k, \bullet)$ is a permutation on $\{0,1\}^n$.

An alternative notation is: $P^{a,n} = \{P_k\}_{k \in \{0,1\}^n}$.

The usability requirements are:

1. P_k is a permutation on $\{0,1\}^n$.
2. There exists a polynomial-time algorithms, that efficiently computes P_k and P_k^{-1}.

In order to "encrypt" message m with key k we compute "cipher" $c = P_k(m)$.

To "decrypt", we compute $m = P_k^{-1}(c)$.

What are the security specifications of P?

Unlike the case of stream ciphers, where the security specification were that the output is indistinguishable from a random string, here we require that P "looks like" a random function (in fact, a random permutation on $\{0,1\}^n$). (See figure 2)

3 Pseudo-random permutations/functions

Definition 4:

A family of functions $P^{a,n}$ is $\epsilon(t)$- pseudo random permutation (PRP) if for any adversary A that runs at time t:

$$|Prob_{k \in \{0,1\}^a}[A^{P_k} = 1] - Prob_{P \in \mathcal{P}_n}[A^P = 1]| < \mathcal{E}(t)$$

Where $\mathcal{P}_n$ is the family of all the permutations, $P : \{0,1\}^n \to \{0,1\}^n$

Definition 5:

A family of functions $P^{a,n}$ is $\epsilon(t)$-strong pseudo random permutation (SPRP) if for any adversary A that runs at time t:

$$|Prob_{k \in \{0,1\}^k}[A^{P_k, P_k^{-1}} = 1] - Prob_{P \in \mathcal{P}_n}[A^{P, P^{-1}} = 1]| < \epsilon E(t)$$

This definition is important in case the adversary has an access to oracle which get a cipher c and returns $m = P^{-1}(c)$. This might be crucial if adversary A can generate smart ciphers and send them to the oracle, in order to learn whether a real cipher was generated from a random permutation or from pseudo random permutation.

Until now we discussed security in concrete numbers, for simplification we will move to asymptotic model.

Definition 6:

Ensemble $P = \{P_n\}_{n \in N}$ is an asymptotic SPRP if for any polynomial adversary A and large enough n:

$$|Prob_{k \in \{0,1\}^{a(n)}}[A^{P_k, P_k^{-1}}(1^n) = 1] - Prob_{P \in \mathcal{P}_n}[A^{P, P^{-1}}(1^n) = 1]| < \nu(n),$$

for some negligible function $\nu(n)$.

In other words, any adversary A that runs at polynomial time and has access to both P and P^{-1} can't distinguish between permutations from $P = \{P_n\}_{n\in N}$ and $P^a = \{P^{a(n),n}\}_{n\in N}$.

Theorem 2:

If there exist OWF then there exist Strong Pseudo Random permutations (SPRP).

Before proceeding to pseudo random permutations we will define an intermediate term pseudo random function (PRF). This term will help us to move from PRGs to PRPs. This term is also important by itself as we will see later at the course.

Definition 7 - Pseudo Random Functions (PRF):

Ensemble F = $\left\{F^{(a(n),n)}\right\}_{n\in N}$ is pseudo random if for any polynomial adversary A and for large enough n:

$$|Prob_{k\in\{0,1\}^n}[A^{F_k}(1^n) = 1] - Prob_{F\in\mathcal{F}^{n,n}}[A^F(1^n) = 1]| < \nu(n),$$

where $\mathcal{F}^{n,n}$ is the family of all functions $\mathcal{F}^{n,n} = \{F : \{0,1\}^n \to \{0,1\}^n\}$.

We will Prove Theorem 1 (OWF -> SPRP) in stages:

1. OWF->PRG (Already proven at the last lecture)
2. PRG->PRF (We will prove today)
3. PRF->PRP (We will prove at the next lecture)
4. PRP->SPRP (will not see in class)

Theorem 3 ([Goldreich, Goldwasser, Micali 1985] - GGM:)

If there exists pseudo random generators then there exists pseudo random functions (PRG->PRF).

Proof for theorem 3:

We will show now how to construct a pseudo random function from pseudo random generator.

Let G be a pseudo random generator that expands input of length n into input of length 2n.

$$G : \{0,1\}^n \to \{0,1\}^{2n}$$

We Denote by $G_0(x)$ the first n bits of G(x) and by $G_1(x)$ the last n bits of G(x).

$$G(x) = G_0(x), G_1(x)$$

First attempt:

Given a generator G:$\{0,1\}^n \to \{0,1\}^{2n}$ and an input x=$x_1, ..., x_n$. A first attempt to build a pseudo random function might go as follows:

$$F(x) = \overbrace{G_1(...(G_1(G_1}^{x\ times}(k)))...),$$

In words, run G(k) x times each time on the last n bits of the result of G, (figuratively, we will get an output of size $2^n * n$) and return the n bits at index x of the output.

There are two main flaws at this implementation.

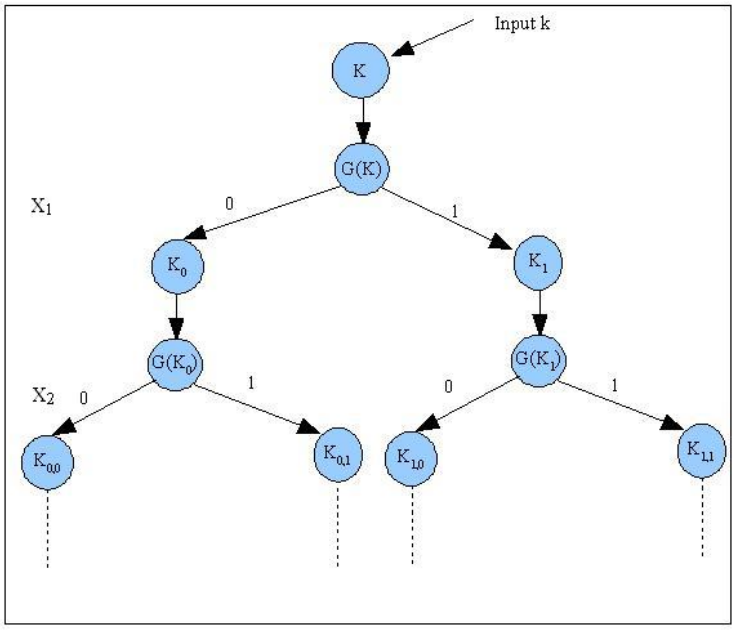

Figure 3: F^{GGM} Tree

The first one is that computation of the PRF is not efficient, the complexity time will take 2^n for input of size n. The second flaw is that it is not clear whether exponential number of activations of the generator keeps the output pseudo-random.

Second attempt:

We define a function F_k^{GGM}:$\{0,1\}^n \rightarrow \{0,1\}^n$

$$F_k^{GGM}(x_1, ..., x_n) = G_{x_n}(...(G_{x_2}(G_{x_1}(k)))...)$$

Basically, we can look at the function F_k^{GGM} as n-steps walk down on a full binary tree of depth n. Each choice for left or right is decided by the value of the bit x_i (see figure 3).

For example, at the root of the tree if x_1 = 0 then f will choose left, otherwise right.

To show that F^{GGM} is a PRP, we will use a reduction to the warm up theorem using the hybrid technique. Specifically, we will use a hybrid tree where the first i levels (from the root) will be chosen uniformly and the i+1...n last level will be computed using the function F^{GGM}. More precisely, for every $k_1, ..., k_{2^i} \in \{0,1\}^n$ we define the function $F^{GGM}_{k_1,...,k_{2^i}}$: $\{0,1\}^n \rightarrow \{0,1\}^n$ such that

$$F^{GGM}_{k_1,...,k_{2^i}}(x_{i+1}, ..., x_n) = G_{x_n}(...(G_{x_i}(k_{x_1,...,x_i}))...)$$

Now, given an distinguisher A for F^{GGM}, we construct a distinguisher A' for the underlying PRG G. (In fact, using the warm-up theorem, A' will obtain multiple samples of either $G(U_n)$ or U_{2n} and will use A to tell which is the case.) A' will run A on an interaction that will look to A like an interaction with a hybrid tree. The level i of the switch in the tree will depend on whether A gets samples from $G(U_n)$ or from U_{2n}.

We are ready to move to the actual proof of security.

Lets assume by contradiction that there exist an adversary A that can distinguish between the ensemble F^{GGM} and the ensemble $\mathcal{F}^{n,n}$.

$$|Prob_{k\in\{0,1\}^n}[A^{F_k^{GGM}}(1^n) = 1] - Prob_{F\in\mathcal{F}^{n,n}}[A^F(1^n) = 1]| > \epsilon$$

A gets input from oracle O.

We define experiment E_i:

On the first i levels A walks down on the tree randomly. Actually, since the values of the first i levels are randomly selected it is not important what is the exact path on these levels. At each node of these first i levels A sees random values and ignores them.

From levels i to n A generates G(x) on each level, and walks down the tree as described at F^{GGM}.

Let E_i be the following experiment:

1. Randomly choose $k_0, ..., k_{2^i-1}$
2. On input $x_1, ..., x_n$ return $F^{GGM}_{k_1,...,k_{2^i}}(x_{i+1}, ..., x_n)$

E_0 is exactly the experiment$A^{F_k^{GGM}}(1^n)$ where F_k^{GGM} is taken from F^{GGM}.

E_n is exactly the experiment $A^{F(1^n)}$ where F is taken from $\mathcal{F}^{n,n}$

Let H_i be the probability that A outputs 1 in E_i.

H_0 is the probability that $A^{F_k^{GGM}(1^n)}$ outputs 1 where F_k^{GGM} is taken from F^{GGM}.

H_n is the probability that $A^F(1^n)$ outputs 1, where F is taken from $\mathcal{F}^{n,n}$.

Given adversary A, we will build A' that speaks with an oracle O which gives samples from distributions $G(U_n)$ or U_{2n}.

In order to keep on consistency of two different queries of prefixes at size i, we will use a memory M which holds the intermediate values for the length-i prefixes of all the queries so far.

A':

- $M \leftarrow \phi$
- Run A.
- Answer each oracle query of A as follows:
- Choose randomly i from 1 to n.
- If the entry (x_1,...,x_i,k) exists in M return $F_k^{GGM}(x_{i+1},...,x_n)$
- Otherwise, take sample $s_0, s_1, M \leftarrow M \bigcup (x_1, ..., x_i, s_{x_i})$ and return $F^{GGM}_{s_{x_i}}(x_{i+1},...,x_n)$
- Whenever A outputs a value, output the same value.

Note:

If A' receives sample from U_{2n} then A sees exactly the experiment E_i

If A' receives sample from G(U_n) then A sees exactly the experiment E_{i-1}.

$|Prob_{x\in G(U_n)}[A'(x) = 1] - Prob_{x\in U_{2n}}[A'(x) = 1]| =$
$= \sum_{i=1}^{n}[Prob_{k\in\{0,1\}^n}[A^{F_k}(1^n) = 1|j = i]] \cdot Prob[j = i]-$
$\sum_{i=1}^{n}[Prob_{F\in\mathcal{F}^{n,n}}[A^F(1^n) = 1|j = i]] \cdot Prob[j = i] =$
$= 1/n * \sum_{i=1}^{n}(|H_{i-1} - H_i|) = 1/n * (H_n - H_0) > \epsilon/n$

Using the warm up theorem, we get a contradiction with the assumption that G is a PRG.

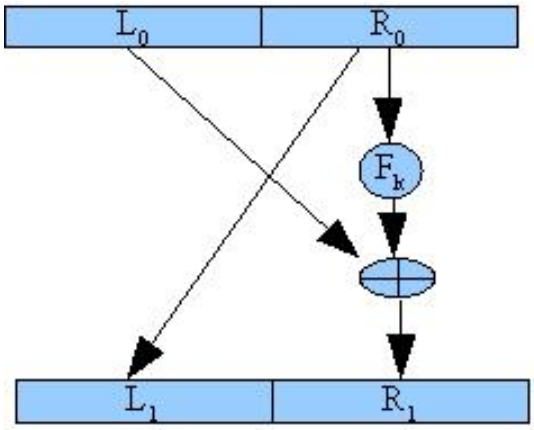

Figure 4: Feistel

4 Feistel transform (from PRF to PRP):

How can we obtain a PRP from a PRF? In fact, how to obtain a permutation from a PRF, regardless of security? By definition, a PRF look completely unstructured so how to regain the permutation structure? since almost impossible by definition...

Still, there is a simple method to do so, and the price is doubling the input length. This is called the Feistel ladder:

$FE_F(l,r) = (r, l \oplus F(r))$ or $FE_F(l,r) = (r, l \oplus F(r))$. (see figure 4)

Generally we can define $FE_F^k(l_{k-1}, r_{k-1}) = (r_{k-1}, l_{k-1} \oplus F_{k-1}(r_{k-1}))$.

Where F_k is taken from a PRF family F. Each iteration of the Feistel transform uses a fresh function F chosen from the family.

Main observation, $FE_F^{-1}(l,r) = (r \oplus F(l), l)$.

This means that $FE_F(l,r)$ is an efficiently reversible permutation, regardless of F.

A Feistel transform is used in the construction of block ciphers, it is named after the German IBM cryptographer Horst Feistel, who was one of the inventors of DES. Indeed, this structure is used there, as we will see later.

Is the Feistel permutation is a PRP (assuming the F is PRF)?

Certainly not... given (l', r') such that $FE_{F_1}(l,r) = (l', r')$ we can easily find (l,r). This definitely won't happen with random permutation RP(l,r).

How about a two-step Feistel?

$F_e^2(l,r)$

$l_1 = r_0$.

$r_1 = l_0 \oplus F_0(r_0)$.

$l_2 = r_1$.

$r_2 = l_1 \oplus F_1(r_1)$.

Note that the two applications of F have independent keys.

Attack: Pick l, l', r at random, set x=(l,r) and x'=(l', r), ask for O(x) and O(x'), and obtain answers (l_2, r_2) and (l_2', r_2'). Note that if O is a Feistel, then $l_2 \oplus l_2' = l \oplus'$. If O is a random permutation then this happens only with negligible probability.

Next week we will see what happens with FE_F^3.