0368.4162: Introduction to Cryptography	Ran Canetti
Lecture 6	
03 November 2008	Fall 2008

Scribes: M. Hakimi, M. Radashkovich

Today's lecture topics:

1. Show PRFs → PRPs.
2. Direct constructions of block ciphers, design principles.
3. Key exchange.
4. Trapdoor permutations.

1 From PRFs to PRPs

Definition 1. *Let R_n be the family of all functions from $0,1^n$ to $0,1^n$.*

Definition 2. *Let P_{2n} be the family of permutations on $\{0,1\}^{2n}$.*

Definition 3. *Ensemble $\mathcal{F} = \{F^{a(n),n}\}_{n\in\mathcal{N}}$ such that*

$$F^{a(n),n} = \{F_k = \{0,1\}^n \to \{0,1\}^n\}_{k\in\{0,1\}^{a(n)}}$$

is a PRF if for every polytime opponent A

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

Where $v(n)$ is a negligible function and F is randomly drawn from R_n.

We will now recall the Feistel ladder:

Let:

$$Fe_{\mathcal{F}} = \{Fe_{F^n}\}_{n\in N}, Fe_{F^n} = \{Fe_f\}_{f\in F^n}$$
$$Fe_f(l,r) = (r, l \oplus f(r))$$

Now for the composition, Let:

$$Fe^k_{\mathcal{F}} = \{Fe^k_{F^n}\}_{n\in N}$$
$$Fe^k_{F^n} = \{g(r, l \oplus f(r)\}_{f\in F^n, g\in Fe^{k-1}_{F^n}}$$

Theorem [Luby and Rackoff 88]: 1. *If $\mathcal{F}$ is a PRF then $Fe^3_{\mathcal{F}}$ is a PRP.*

Our modus operandi for proving: We'll first prove a few claims which we will later use the prove the theorem, then we'll reduce the theorem statement to the case where the F_i's are truly random and prove it.

Lemma 1.1. • *Let A be an oracle machine that makes at most q queries.*

Then:

$$|Prob_{F\leftarrow Fe^3_{R_n}}[A^F(1^n) = 1] - [Prob_{P\leftarrow P_{2n}}[A^P(1^n) = 1]| < \frac{4q^2}{2^n} \quad (*)$$

Proof. We use the following notation to describe the internal values in the evaluation of Fe_F^3 on input (l_0, r_0):

$$\begin{array}{ll} l_0 & r_0 \\ l_1 = r_0 & r_1 = l_0 + F_1(r_0) \\ l_2 = r_1 & r_2 = l_1 + F_2(r_1) \\ l_3 = r_2 & r_3 = l_2 + F_3(r_2) \end{array}$$

- Let l_j^i the value of l_j in the i'th question of A.
- Let r_j^i the value of r_j in the i'th question of A.

(Assuming A never asks the same question twice wlog)

Claim 1.1.1. *$\forall A$ and $\forall i \neq i'$*

$$Prob[r_1^i = r_1^{i'}] \leq \frac{1}{2^n}$$

Proof.

- If $r_0^i = r_0^{i'}$ then $r_1^i \neq r_1^{i'}$ is always true.(Since $l_0^i \neq l_0^{i'}$(or the queries are identical) $\rightarrow F_1(r_0^i) + l_0^i \neq F_1(r_0^{i'=i}) + l^{i'}$)
- if $r_0^i \neq r_0^{i'}$ then $r_1^i \neq r_1^{i'}$ is true with probability $\frac{1}{2^n}$ (Since F_1 is a random function with no dependencies to $l_0^i, l_0^{i'}$ values).

□

Claim 1.1.2. *$\forall A$ and $\forall i \neq i'$*

$$Prob[r_2^i = r_2^{i'}] < \frac{2}{2^n}$$

Proof.

$$Prob[r_2^i = r_2^{i'}] = Prob[r_2^i = r_2^{i'} | r_1^i = r_1^{i'}] Prob[r_1^i = r_1^{i'}] + Prob[r_2^i = r_2^{i'} | r_1^i \neq r_1^{i'}] Prob[r_1^i \neq r_1^{i'}]$$

On the l.h.s of the addition we use claim 1.1.1 and bound $Prob[r_2^i = r_2^{i'} | r_1^i = r_1^{i'}]$ by 1 And on the r.h.s of the addition we observe that $Prob[r_2^i = r_2^{i'} | r_1^i \neq r_1^{i'}] < \frac{1}{2^n}$ (See proof for claim 1.1.1), and bound $Prob[r_1^i \neq r_1^{i'}]$ by 1. □

Claim 1.1.3. *Let C_k be the event where $\exists i, i' \;\; i \neq i' \wedge i, i' \leq k$ s.t.*

$$r_2^i = r_2^{i'} \vee r_1^i = r_1^{i'}$$

Then

$$Prob[C_k] < \frac{3k^2}{2^n}$$

Proof.
From claim 1.1.1 and claim 1.1.2 (we check if the k'th query satisfies the conditions with any of the previous k-1 queries) we get:

$$Prob[C_k | \overline{C_{k-1}}] < \frac{3(k-1)}{2^n} \tag{1}$$

By partitioning to conditional probabilities we have:

$$Prob[C_k] \leq Prob[C_k | C_{k-1}] Prob[C_{k-1}] + Prob[C_k | \overline{C_{k-1}}] Prob[\overline{C_{k-1}}] \tag{2}$$

By bounding $Prob[C_k|C_{k-1}]$ and $Prob[\overline{C_{k-1}}]$ by one we get

$$Prob[C_{k-1}] + Prob[C_k|\overline{C_{k-1}}]$$

Performing induction on k we get $\sum_{i=1}^{k} Prob[C_k|\overline{C_{k-1}}]$ and now we apply (1) to get (3).

$$Prob[C_k] \leq \sum_{i=1}^{k-1} \frac{3i}{2^n} \leq \frac{3k^2}{2^n} \tag{3}$$

□

Let us remember the lemma we wished to prove

$$|Prob_{F \leftarrow Fe^3_{R_n}}[A^F(1^n) = 1] - [Prob_{P \leftarrow P_{2n}}[A^P(1^n) = 1]| < \frac{4q^2}{2^n} \quad (*)$$

We will look at the case where the event $\overline{C_q}$ happened i.e. (C_q didn't happen)
Note that $\{r_3^1, ..., r_3^q\}$ are now all jointly uniform, i.e. they are uniform in $\{\{0,1\}^n\}^q$. $\{l_3^1, ..., l_3^q\}$ On the other hand are also jointly uniform, subject to the condition that they are all different. Learning this, we conclude that the only way an adversary could distinguish between the two distributions(one created from a truly random permutation and the other from the Feistel construction) is to find $j \neq i$ s.t. $l_3^i = l_3^j$ and $r_3^i \neq r_3^j$ (such pairs could appear if the distribution is taken from P_{2n} but never if taken from Fe_k^3). There are at most $\binom{q}{2}$ such pairs in P_{2n}.

Therefor the probability of an adversary distinguishing the result of Fe_R^3 from a P$\leftarrow$ P_{2n} is (using claim 1.1.3)

$$\frac{3q^2}{2^n} + \frac{\binom{q}{2}}{2^n} \leq \frac{4q^2}{2^n} \tag{4}$$

□

Notice that in numerical terms the bound here is not very good, the probability an adversary of distinguishing grows quadratically in the number of queries. We, of course, would like it to grow slower. Such constructions exists which result in better bounds. Also, the main lemma comes in handy in other areas in cryptography.

Proof of the theorem from the lemma:
By reduction to the PRFness of F.
We'll need an intermediary lemma: If F is a PRF then no adversary can distinguish three independent functions drawn from F from three independent random functions. This is shown via a straightforward hybrids argument.

Now, assume that there is a polytime adversary A that distinguishes Fe_F^3 from P_{2n} w.p. e. We construct a polytime adversary A' that distinguishes three functions drawn from F from three random functions $R1, R2, R3$ w.p. $e - \frac{4q^2}{2^n}$. Given oracles $O1, O2, O3$, A' runs A, where each query of A is answered via $Fe^3_{O1,O2,O3}$. A' then outputs whatever A outputs. Now, if the oracles of A' are taken from F, then A sees Fe_F^3. If the oracles of A' are taken from R_n, then A sees Fe_R^3 rather than P_{2n}. (If A saw P_{2n} then A' would succeed w.p. e.) However, by (4) the difference between the output of A given P_{2n} and given Fe_R^3 is at most $\frac{4q^2}{2^n}$. Thus, A' distinguishes with probability $e - \frac{4q^2}{2^n}$. □

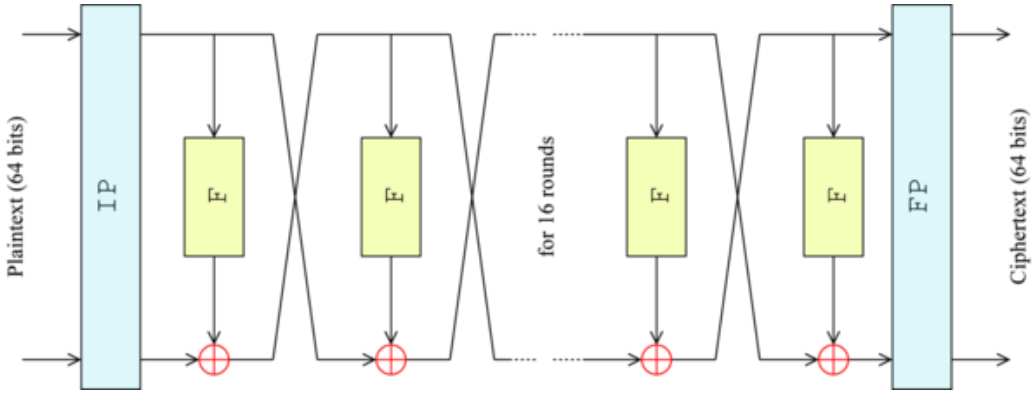
This proof technique highlights the power of the concept of PRFs: Allows us to think and reason only about the case where F is a random function. But note that this works only as long the secret keys of F remain unknown to the attacker! Turns out that Fe^3 is not enough(There is an attack). However Fe^4 is in fact enough.

[Luby and Rackoff 88] 2. *If $\mathcal{F}$ is a PRF then $Fe^4_{\mathcal{F}}$ is a strong PRP We will not include a proof here however one can be found at-*
http://www.wisdom.weizmann.ac.il/ reingold/publications/LR.PS

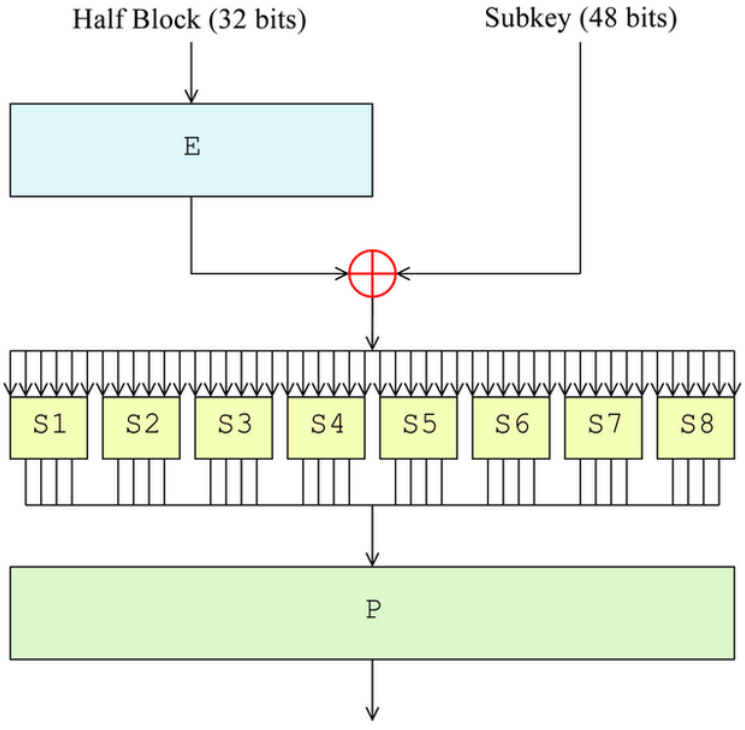
2 Direct constructions of block ciphers (SPRPs):

In practice, people do not use the constructions based on OWFs that we saw in class. Instead, they use direct("dirty") constructions. We'll give a brief overview of two main constructions: DES and AES.

The first block cipher constructed was the DES, developed at IBM with the NSA around 1976, long before a lot of the theory we've learned was developed. The block size was n=64 bits, with key size a=56 bits.
The main structure is that of a Feistel ladder with 16 rounds, where each round function is a univertible function on 32 bits. This is a sketch of a 16-round Feistel ladder



We will now take a closer look at these round functions. A 56-bit key is expanded (via a "key schedule") to 16 ("pseudo random") strings of 48-bit each. Next each round function starts by expanding the 32 bit input to 48 bits, xoring these 48 bits with the corresponding 48-bit string derived from the key. Then, the 48 bits are partitioned into 8 groups of 6 bits each, and each group goes via some fixed "random" function, called an s-box that goes from 6 to 4 bits. The output is the resulting 32 bits, permuted by some fixed permutation P. Below, a sketch of the s-boxes MO.



We notice that:

- The only non-linear (or, "hard to invert") operations in the cipher are the s-boxes
- The s-boxes are performed only on small chunks of inputs.
- The key is incorporated only in a "mild" way (via the initial xor).

Thus one might expect that the cipher is weak. However, it turns out that no significant "practical" weaknesses have been found in DES - the reason it was phased out is that the block size is small (allowing for semi-feasible exhaustive-search attacks), and its speed isn't too great either.

DES might be regarded as an instantiation of the "confusion and diffusion" principle put forth by Shannon. The principle calls for working as follows.

1. Partition the input bits to chunks.
2. Apply a "random function" locally on each chunk
3. Perform some simple "diffusion" operation that makes the each chunk of bits depend on each other chunk.
4. Repeat the process

The only non-trivial attacks against DES are of a type called "differential cryptanalysis". We have seen an example for such an attack, namely the attack against a 2-round Feistel. Essentially, one feeds the cipher with random inputs with known (and carefully chosen) xor, and hopes to see similar correlations in the outputs. Somewhat more sophisticated attacks, called "linear cryptanalysis", replace the xor with other linear relations. However, these attacks never got close to being practically viable(Though they were successful against other ciphers).

In 1997 NIST(National Institute of Standards and Technology) has announced a competition for a block cipher that would replace DES. The security requirements were essentially that the new cipher be a good SPRP for a block size of 128 bits and variable key size 128-160 bits. There were lots of efficiency requirements (including efficiency in software, hardware, on low end and high end machines, small footprint and simplicity) There were 5 finalists which were largely accepted to be all very good. The winning algorithm, Rijndael (now called simply AES) is a very elegant construction:

Similarly to DES, AES works in rounds. (The number of rounds, r, depends on the key length, which is variable.) Initially the key is expanded to r chunks (of 128 bits each) where each chunk is xored with the current state at the onset of round r.

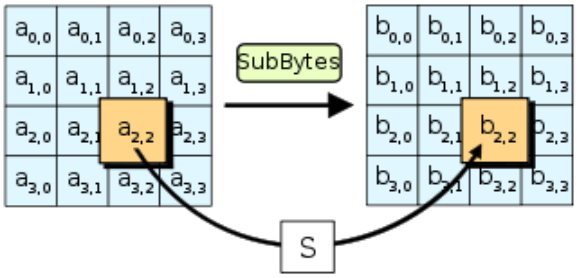
Similarly to DES, the "heart" of each round is a bunch of s-boxes, or non-linear operation that are done one small chunks of bits, plus some simple operations that "diffuse" the information among the individual chunks.

However, there are a number of significant differences regarding the s-boxes:

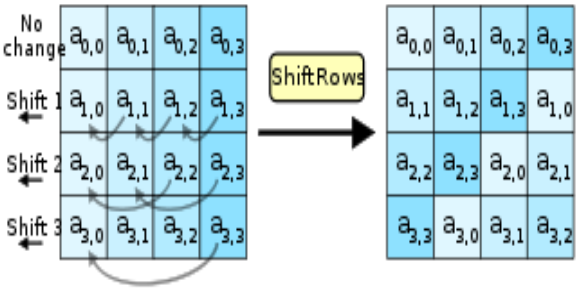
1. In AES the s-boxes are invertible permutations (specifically, they are permutations on bytes (8 bits)).
2. The s-boxes are not "hand crafted" as the s-boxes in DES appear to be. Rather, they are constructed via some non-linear algebraic mechanism.

The fact that the s-boxes (and all the other operations in the cipher) are invertible, means that there is no need in the Feistel structure. Instead, inversion of AES is performed by inverting each internal operation in sequence.

There is more extensive "diffusion" of information among the blocks than in DES. Specifically, the diffusion step proceeds as follows: The 16 blocks are thought of as arranged in a 4x4 matrix. Then, after the s-box functions are applied to each block separately,



the i-th row of blocks is rotated by i (to the left).



Then, the i-th column is treated as a length-4 vector with entries in F_2^8, and is multiplied (in F_2^8) by some fixed 4x4 matrix.

We not that this description is not complete, nor is it 100% precise. However it gives the main idea. For more information please refer to
http://en.wikipedia.org/wiki/Data_Encryption_Standard
http://en.wikipedia.org/wiki/Advanced_Encryption_Standard

As of today, there are no known "weaknesses" in AES.

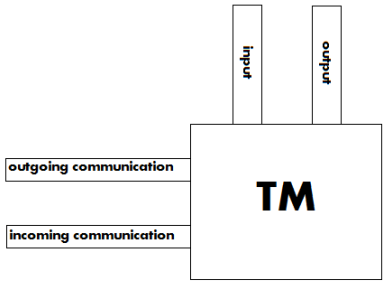
3 Key Exchange

So far we have seen how to construct stream ciphers (PRGs) and block ciphers (PRPs) from OWFs. Key exchange is an example for something that we don't know how to construct from OWFs.
The key exchange problem: Consider two parties that wish to agree on a secret key, but they can only communicate over an open (tappable) communication medium. How can they do it?

3.1 Formulation of the problem:

Definition 1. ***An interactive algorithm*** *is an algorithm with two types of input, and two types of output:*
- Local input
- Local output

- Incoming communication
- Outgoing communication



We can think of an interactive algorithm as a Turing Machine with two extra tapes (These are TM's are called ITM - Interactive Turing Machine).
A two party protocol $P = (A, B)$ is a pair of interactive algorithms (A, B).

Definition 2. *We say that an ITM M is polytime if at any point the overall runtime of M so far is bounded by n^c for some $c > 0$, where n is the overall accumulating length of inputs to M in its execution. (We stress that here we measure only the length on the information written on the* **input tape***. The information written on the incoming communication tape is not counted towards providing runtime.)*

Definition 3. *For an **Execution of a Protocol** P, we look at the system where the outgoing communication tape of one is unified with the incoming communication tape of the other. More formally, an execution of a two party protocol is a sequence of activations. In each activation one out of (A, B) is activated in his turn. When activated, the algorithm runs until it reaches a waiting state or until it stops.*

Definition 4. *For $P = (A, B)$, let $P(x_A, r_A; x_B, r_B) = (y_A, y_B)$ denote the **output of the protocol** on input x_A and random input r_A for A, and input x_B, random input r_B for B. Here y_A is A's output and y_B is B's output. $P(x_A; x_B)$ is the corresponding random variable over r_A, r_B.*

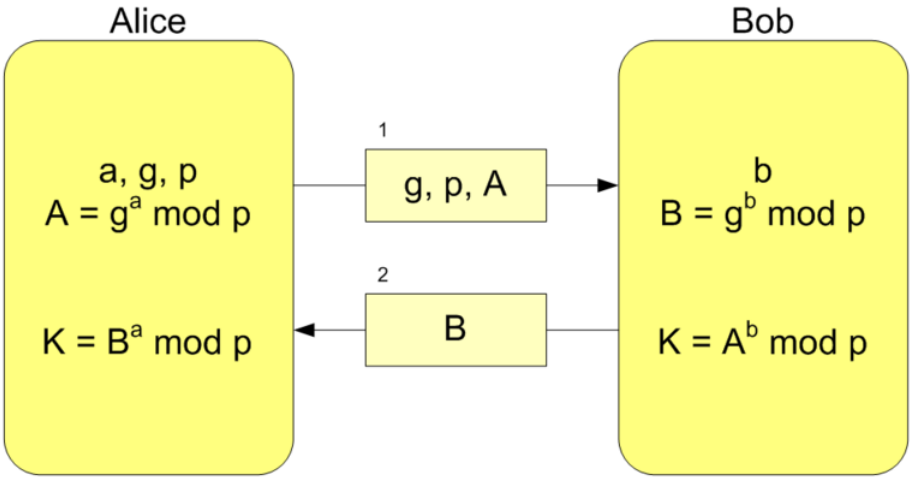
Definition 5. *Let $COM_P(x_A, r_A; x_B, r_B)$ denote the **protocol's communication** in the run of P on inputs and random inputs (x_A, r_A, x_B, r_B), namely the sequence of values written over the communication tapes between the parties.*
$COM_P(x_A, x_B)$ is the corresponding random variable over r_A, r_B.
$\{OUT + COM_P\}(x_A, r_A; x_B, r_B) = P(x_A, r_A; x_B, r_B)|COM_P(x_A, r_A; x_B, r_B)$
$\{OUT+COM_P\}(x_A, x_B)$ is the random variable describing the joint r.v. $P(x_A, x_B), COM_P(x_A, x_B)$, where both the output and the communication are computed based on the same values of r_A and r_B.

Definition 6. *A two party protocol is a **key exchange protocol** with respect to domain $D = \{D_n\}_{n\in N}$ if:*
-Agreement: For every r_A, r_B, $P(1^n, r_A; 1^n, r_B) = (y_A, y_B)$ where $y_A = y_B$.
-Secrecy: $\{OUT + COM_P(1^n, 1^n)\}_{n\in N} \approx \{COM_P(1^n, 1^n), y, y\}_{n\in N}$, where y is chosen uniformly from D_n.
The secrecy criterion formalizes the intuition that the key should "look random" to a polynomial time outside observer.
We remark that there are other, more sophisticated definitions of security for key exchange. Still, this

one will do for our purposes.
As said, we don't know how to construct key exchange protocols assuming only existence of OWFs. In fact, there is some evidence that this might be hard to prove (essentially, proofs that simply use the underlying OWF "in a black box way" probably cannot work).

Definition 7. *Assume A, B agree on a prime order group G and a generator g of G (In fact, we consider an ensemble $G = \{G_n\}_{n\in N}$, where G_n is of order $p, p \sim 2^n$ is a prime).*
***The Diffie-Hellman protocol**:*
Alice: On input 1^n, choose random generator $g \in G_n$ and an element $a \in 1..|p|$, send g, g^a to Bob.
Bob: Choose random $b \in G$, send g^b to A, and output $(g^a)^b$.
Alice: output $(g^b)^a$.

The above key exchange protocol can be applied on every prime order group G. The following is an example for a more specific case where G is the group Z_p^*.



$K = A^b \bmod p = (g^a \bmod p)^b \bmod p = g^{ab} \bmod p = (g^b \bmod p)^a \bmod p = B^a \bmod p$

It is obvious that both parties agree on the same key, but what about secrecy?
Theorem: Assume that the DDH assumption holds with respect to the ensemble G. Then protocol DH with ensemble G is a key exchange protocol for G.
Proof: Agreement is immediate. Secrecy follows immediately from the DDH assumption. (Recall: the DDH assumption over G states that $g, g^a, h, h^a \approx g, g^a, h, h^r$ where g, h are random in G and a, r are random in [1..|p|]).
In our case $\{OUT + COM_P\} = \{g, g^a, g^b, g^{ab}\}$ and $\{COM_P, y, y\} = \{g, g^a, g^b, g^{br}\}$ (In our case, the adversary sees $\{g, g^a, g^b, g^{br}\} \approx \{g, g^a, g^b, g^{ab}\}$)

3.2 Trapdoor Permutation:

How to generalize from the specific properties of DL_P to get a structure (akin to OWFs) that would give the desired properties of key exchange?
We'll see a primitive that allows having key exchange (but is somewhat stronger than simply key exchange). The idea is to have one way functions with "trapdoor" - namely one way functions that can actually be inverted with additional "trapdoor" information.

How to formalize it?

3.2.1 Formulation of the problem:

Attempt 1: $f : \{0,1\}^* \to \{0,1\}^*$ is a TDP if
- for every n, f is a permutation on $\{0,1\}^n$.
- It is a OWF.
- There exist an algorithm f^{-1} that inverts f (on input y outputs x s.t. $f(x) = y$).

But this is self contradictory...
So we need something else. We'll require to have a family of functions (or, permutations), where it is possible to generate "forward computing" and "inversion" keys, and where the inversion key for one function in the family does not give the ability to invert other functions in the family. More specifically:

Definition 8. ***One way permutation families with trapdoor (TDPs)****:*
Ensemble $F = \{F_n\}_{n\in N}$ is ensemble of a TDP if for every n, $F_n = \{f_i^n : D_i^n \to D_i^n\}_{i\in I_n}$ (I_n is the key index) and there exist the following algorithms:
G - a randomized Index generation algorithm: $G(1^n) = (i,t)$.
S - a randomized domain Sampling algorithm: $S(1^n, i) \in D_i^n$.
F - an evaluation algorithm: for all i and $x \in D_i^n$, $F(1^n, i, x) = f_i^n(x)$.
F^{-1} - an inversion algorithm: for every i and $x \in D_i$, $F^{-1}(1^n, t, f_i^n(x)) = x$.

One wayness: For any polynomial time non uniform algorithm A, $Pr_{(i,t)\in G(1^n), x\in S(1^n,i)}[A((f_i^n(x)) = x] < \nu(n)$ for a negligible function $\nu(n)$.

Notes:
• Notice that here the probability of choosing x is determined by S, and is not necessarily uniform.
• The definition makes sense as a definition of one way functions even without trapdoor. This formulation is useful for "more structured families" which are not defined overall of $\{0,1\}^*$, or where distributions other than the uniform one make more sense.
• There are a number of additional properties that are often useful for TPDs. for instance:
– It's often useful to be able to recognize whether i is a valid index in the range of g.
– It's often useful to be able to recognize whether a given x is in D_i for a given i.
• Why insist on permutations? Because this will be convenient. The definition makes sense (and is still useful) when F is a family of functions that are not too-many-to-one. When the functions are too-many-to-one the ability to invert seems much less useful.

Theorem: If there exist TDPs then there exist key exchange protocols.
Proof: Homework.

3.2.2 Candidates for TDPs

We will now see two candidates for TDPs. These candidates are based on the hardness of factoring, specifically factoring of $n = p \cdot q$ where p, q are prime numbers.
Both candidates are based of the following facts:
• In Z_p^* where p is a prime, it is easy to compute e-th roots for any e: Given y,e, it is easy to find x s.t. $x^e = y(mod\, p)$.

• In Z_N^*, where $N = p \cdot q$, p and q are primes, if p,q are given, then computing roots can be reduced to computing e-th roots modulo p,q (and is thus easy).
• If p,q are not given, it is not known how to compute e-th roots mod N.

Candidate 1: RSA

$RSA_{N,e}(X) = X^e (mod\ N)$, $N = p \cdot q$, p,q are prime numbers and $gcd(e, (p-1)(q-1)) = 1$.
In this case:
• The index i is (N, e), $G(1^n) = (i = (N, e), t = (N, d))$ where $gcd(e, (p-1)(q-1)) = 1$ and d is such that $x^{e \cdot d} = x (mod\ N)$ for all x.
• The sampling algorithm will choose an element uniformly from Z_N^*.
• The evaluation and inversion algorithms are straightforward.

Note:
• It turns out that RSA is a permutation whenever $gcd(e, (p-1)(q-1)) = 1$.
• If factoring is easy then RSA is not a OW. However, we don't know whether hardness of factoring implies OWness of RSA.

Candidate 2:Rabin

$Rabin(X) = X^2 (mod\ N)$
That is, the function is the same as RSA, except that e is fixed to 2, and the index generator is $G(1^n) = (i = N, t = p, q)$. The good news is that there is a reduction from inverting Rabin to factoring, so in this case we know why it is hard. It at least hard as factoring, and we know that factoring is hard.

The bad news is that the Rabin function is not a permutation... each square in Z_N^* has exactly 4 square roots. But this is fixable. It turns out that when both p and q are 3 mod 4 (such numbers are called Blum integers), $N = p \cdot q$ has a specific structure:
Let QR_N be the set of quadratic residues mod N (That is, $QR_N = \{x |\ there\ is\ y \in Z_N^*\ s.t.\ y^2 = x (mod\ N)\}$). Then, when N is a Blum integer, each $x \in QR_N$ has exactly one square root in QR_N. Consequently, when N is a Blum integer, the Rabin function is a permutation over QR_N.
More precisely:
• The index generator is $G(1^n) = (i = N, t = p, q)$. Here N is a Blum integer.
• The sampling algorithm S samples uniformly from QR_N. (This is done by choosing a value in Z_N^* and squaring it.)
• The evaluation and inversion algorithms are straightforward.

Note that here for OWness one has to assume that factoring is hard even when restricted to Blum integers.