Introduction to Cryptography

Lecture 4

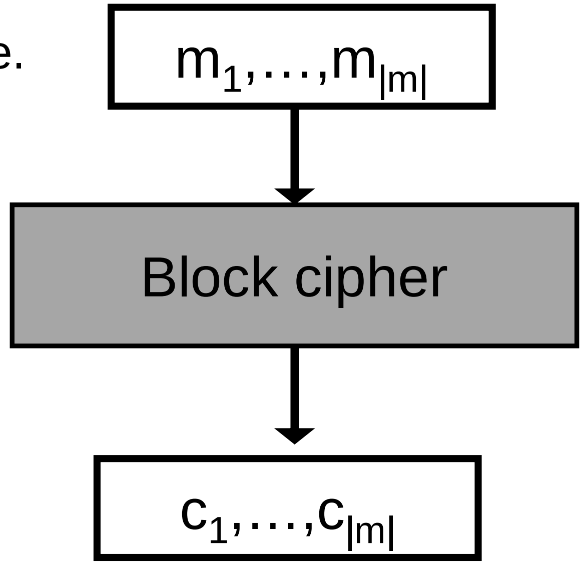
Benny Pinkas

Block Ciphers

- Plaintexts, ciphertexts of **fixed** length, |m|. Usually, |m|=64 or |m|=128 bits.
- The encryption algorithm E_k is a *permutation* over $\{0,1\}^{|m|}$, and the decryption D_k is its inverse. (They *are not* permutations of the bit order, but rather of the entire string.)

- Ideally, use a *random* permutation.
  - Can only be implemented using a table with $2^{|m|}$ entries ☹
- Instead, use a *pseudo-random* permutation, keyed by a key k.
  - Implemented by a computer program whose input is m,k.

- We learned last week how to use a block cipher for encrypting messages longer than the block size.



Pseudo-random functions (PRFs)

- $F : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^*$
  - The first input is the key, and once chosen it is kept fixed.
  - For simplicity, assume $F : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n$
  - $F(k,x)$ is written as $F_k(x)$

- F is pseudo-random if $F_k()$ (where k is chosen uniformly at random) is indistinguishable (to a polynomial distinguisher D) from a function *f* chosen at random from all functions mapping $\{0,1\}^n$ to $\{0,1\}^n$
  - There are 2^n choices of F_k, whereas there are $(2^n)^{2^n}$ choices for *f*.
  - The distinguisher D's task:
    - We choose a function G. With probability ½ G is F_k (where $k \in_R \{0,1\}^n$), and with probability ½ it is a random function *f*.
    - D can compute $G(x_1),G(x_2),\ldots$ for any $x_1,x_2,\ldots$ it chooses.
    - D must say if $G=F_k$ or $G=f$.
    - F_k is pseudo-random if D succeeds with prob ½+negligible..

Pseudo-random permutations (PRPs)

- $F_k(x)$ is a keyed permutation if for every choice of k, $F_k()$ is one-to-one.
  - Note that in this case $F_k(x)$ has an inverse, namely for every y there is exactly one x for which $F_k(x)=y$.

- $F_k(x)$ is a pseudo-random permutation if
  - It is a keyed permutation
  - It is indistinguishable (to a polynomial distinguisher D) from a permutation *f* chosen at random from all permutations mapping $\{0,1\}^n$ to $\{0,1\}^n$.
    - 2^n possible values for F_k
    - $(2^n)!$ possible values for a random permutation
  - It is known how to construct PRPs from PRFs

Block ciphers

- A block cipher is a function $F_k(x)$ with a key k and an |m| bit input x, which has an |m| bit output.
  - $F_k(x)$ is a keyed permutation
  - When analyzing security we assume it to be a PRP (Pseudo-Random Permutation)

- How can we encrypt plaintexts longer than |m|?

- Different modes of operation were designed for this task.
  - Discussed last week.

Design of Block Ciphers

- More an art/engineering challenge than science. Based on experience and public scrutiny.

  – Based on combining together simple building blocks, which support the following principles:
  – *"Diffusion" (bit shuffling):* each intermediate/output bit affected by many input bits
  – *"Confusion"*: avoid structural relationships (and in particular, linear relationships) between bits

- Cascaded (round) design: the encryption algorithm is composed of iterative applications of a simple round

Confusion-Diffusion and Substitution-Permutation Networks

- Construct a PRP for a large block using PRPs for small blocks
- Divide the input to small parts, and apply rounds:
  - Feed the parts through PRPs *("confusion")*
  - Mix the parts *("diffusion")*
  - *Repeat*
- Why both confusion and diffusion are necessary?
- Design musts: Avalanche effect. Using reversible s-boxes.

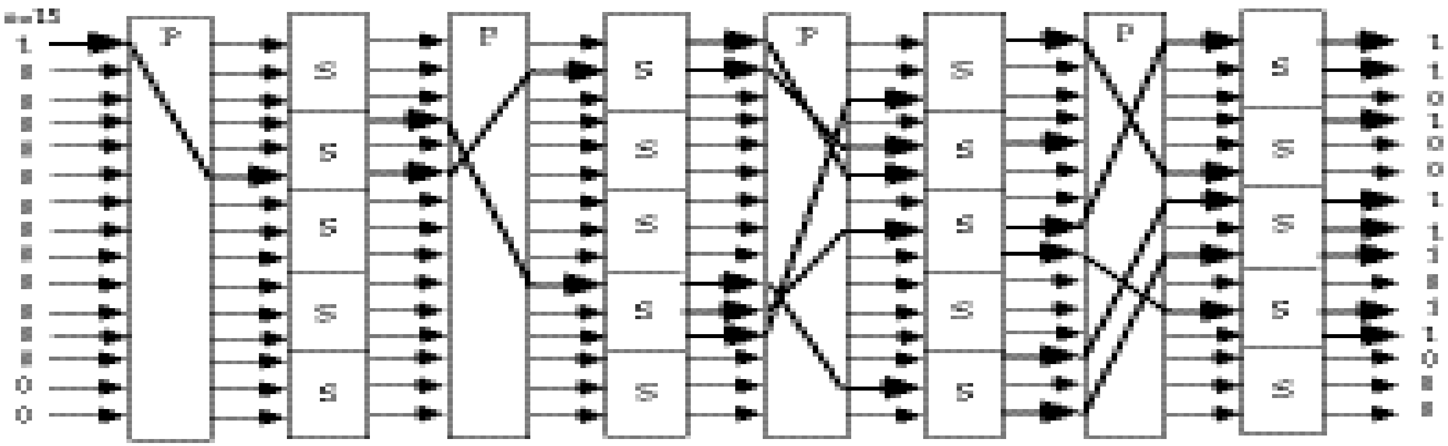


Fig 2.3 - Substitution-Permutation Network, with the Avalanche Characteristic

AES (Advanced Encryption Standard)

- Design initiated in 1997 by NIST
  - Goals: improve security and software efficiency of DES
  - 15 submissions, several rounds of public analysis
  - The winning algorithm: Rijndael

- Input block length: 128 bits
- Key length: 128, 192 or 256 bits
- Multiple rounds (10, 12 or 14), but does not use a Feistel network

Rijndael animation

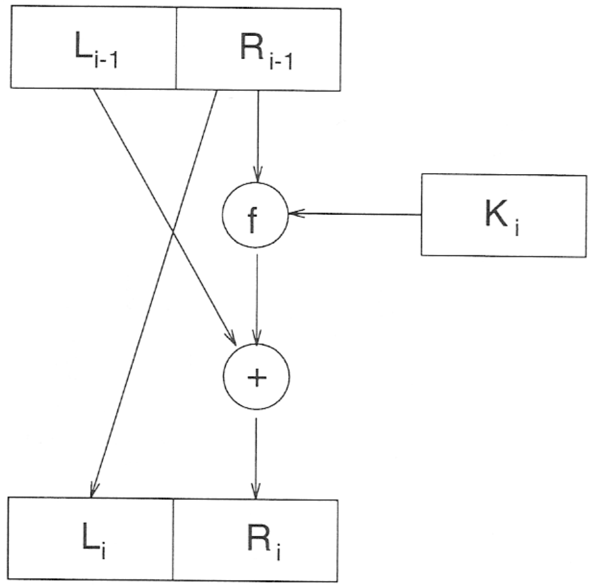
Reversible s-boxes

- Substitution-Permutation networks must use reversible s-boxes
  - Allow for easy decryption
- However, we want the block cipher to be “as random as possible”
  - s-boxes need to have some structure to be reversible
  - Better use non-invertible s-boxes

- Enter Feistel networks
  - A round-based block-cipher which uses s-boxes which are not necessarily reversible
  - Namely, building an invertible function (permutation) from a non-invertible function.

Feistel Networks

- Encryption:
- *Input:* $P = L_{i-1} | R_{i-1}$. $|L_{i-1}|=|R_{i-1}|$
  - $L_i = R_{i-1}$
  - $R_i = L_{i-1} \oplus F(K_i, R_{i-1})$
- Decryption?



- No matter which function is used as F, we obtain a permutation (i.e., F is reversible even if *f* is not).
- The same code/circuit, with keys in reverse order, can be used for decryption.
- Theoretical result [LubRac]: If *f* is a pseudo-random *function* then a 4 rounds Feistel network gives a pseudo-random *permutation*

DES (Data Encryption Standard)

- A Feistel network encryption algorithm:
  - How many rounds?
  - How are the round keys generated?
  - What is F?

- DES (Data Encryption Standard)
  - Designed by IBM and the NSA, 1977.
  - 64 bit input and output
  - 56 bit key
  - 16 round Feistel network
  - Each round key is a 48 bit subset of the key
- Throughput ≈ software: 10Mb/sec, hardware: 1Gb/sec (in 1991!).

Security of DES

- Criticized for unpublished design *decisions* (designers did not want to disclose differential cryptanalysis).
- Very secure – the best attack in practice is brute force
  - 2006: $1 million search machine: 30 seconds
    - cost per key: less than $1
  - •2006: 1000 PCs at night: 1 month
    - Cost per key: essentially 0 (+ some patience)
- Some theoretical attacks were discovered in the 90s:
  - Differential cryptanalysis
  - Linear cryptanalysis: requires about 2^{40} known plaintexts
- The use of DES is not recommend since 2004 , but 3-DES is still recommended for use.

Iterated ciphers

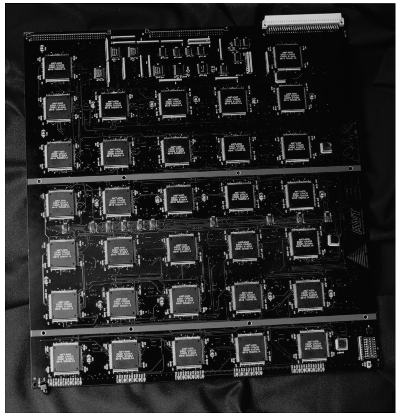
- Suppose that E_k is a good cipher, with a key of length *k* bits and plaintext/ciphertext of length *n*.
  - The best attack on E_k is a brute force attack with has O(1) plaintext/ciphertext pairs, and goes over all 2^k possible keys searching for the one which results in these pairs.
- New technological advances make it possible to run this brute force exhaustive search attack. (Or, I'm willing to invest more in order to get more security.) What shall we do?
  - Design a new cipher with a longer key.
  - Encrypt messages using *two* keys k_1, k_2, and the encryption function $E_{k2}(E_{k1}())$. Hoping that the best brute force attack would take $(2^k)^2 = 2^{2k}$ time.

Iterated ciphers – what can go wrong?

- If encryption is closed under composition, namely for all k_1,k_2 there is a k_3 such that $E_{k2}(E_{k1}())=E_{k3}()$, then we gain nothing.
  - Could just exhaustively search for k_3, instead of separately searching for k_1 and k_2.
  - Substitution ciphers definitely have this property (in fact, they are a permutation group and therefore closed under composition).
  - It was suspected that DES is a group under composition. This assumption was refuted only in 1992.

Iterated Ciphers - Double DES

- DES is out of date due to brute force attacks on its short key (56 bits)



- Why not apply DES twice with two keys?
  - Double DES: DES $_{k1,k2} = E_{k2}(E_{k1}(m))$
  - Key length: 112 bits
- But, double DES is susceptible to a meet-in-the-middle attack, requiring ≈ 2^{56} operations and storage.
  - Compared to brute a force attack, requiring 2^{112} operations and O(1) storage.

Meet-in-the-middle attack

- Meet-in-the-middle attack
  - $c = E_{k2}(E_{k1}(m))$
  - $D_{k2}(c) = E_{k1}(m)$

- The attack:
  - Input: *(m,c)* for which $c = E_{k2}(E_{k1}(m))$
  - For every possible value of k_1, generate and store $E_{k1}(m)$.
  - For every possible value of k_2, generate and store $D_{k2}(c)$.
  - Match k_1 and k_2 for which $E_{k1}(m) = D_{k2}(c)$.
  - Might obtain several options for (k_1,k_2). Check them or repeat the process again with a new *(m,c)* pair (see next slide)

- The attack is applicable to any iterated cipher. Running time and memory are $O(2^{|k|})$, where $|k|$ is the key size.

Meet-in-the-middle attack: how many pairs to check?

- The plaintext and the ciphertext are 64 bits long
- The key is 56 bits long

- Suppose that we are given one plaintext-ciphertext pair (m,c)
  - The attack looks for k1,k2, such that $D_{k2}(c) = E_{k1}(m)$
  - The correct values of k1,k2 satisfy this equality
  - There are 2^{112} (actually 2^{112}-1) other values for k_1,k_2.
  - Each one of these satisfies the equalities with probability 2^{-64}
  - We therefore expect to have $2^{112-64}=2^{48}$ candidates for k_1,k_2.

- Suppose that we are given two pairs (m,c), (m',c')
  - The correct values of k1,k2 satisfy both equalities
  - There are 2^{112} (actually 2^{112}-1) other values for k_1,k_2.
  - Each one of these satisfies the equalities with probability 2^{-128}
  - We therefore expect to have $2^{112-128}<1$ false candidates for k_1,k_2.

Triple DES

- 3DES $_{k1,k2,k3} = E_{k3}(D_{k2}(E_{k1}(m))$
- Two-key-3DES $_{k1,k2} = E_{k1}(D_{k2}(E_{k1}(m))$

- Why use Enc(Dec(Enc( ))) ?
  - Backward compatibility: setting $k_1=k_2$ is compatible with single key DES

- Two-key-3DES (key length is only 112 bits)
  - There is an attack which requires 2^{56} work and memory, but needs also 2^{56} encryptions of *chosen* plaintexts. Therefore not practical.
  - Without chosen plaintext, best attack needs 2^{112} work and memory.
  - Why not use 3DES ? There is a meet-in-the-middle attack against three keys with 2^{112} operations

- 3DES is widely used. Less efficient than DES.