Introduction to Cryptography

Lecture 3

Benny Pinkas

Using a PRG for Encryption

- Key: a (short) random seed $s \in \{0,1\}^{|k|}$.
- Message $m = m_1,\ldots,m_{|m|}$.

- Encryption:
  - Use the output of the PRG as a one-time pad. Namely,
  - Generate $G(s) = g_1,\ldots,g_{|m|}$
  - Ciphertext $C = g_1 \oplus m_1,\ldots, g_{|m|} \oplus m_{|m|}$

Using a PRG for Encryption: Security

- One time pad:
  - $\forall\ m_1,m_2 \in M$, $\forall c$, the probability that c is an encryption of m_1 is equal to the probability that c is an encryption of m_2.
  - I.e., $\forall\ m_1,m_2 \in M\ \forall c$, it is impossible to tell whether c is an encryption of m_1 or of m_2.

- Security of pseudo-random encryption:
  - $\forall\ m_1,m_2 \in M$, no *polynomial time* adversary can distinguish between the encryptions of m_1 and of m_2.

- Proof by reduction: if one can break the security of the encryption (distinguish between encryptions of m_1 and of m_2), it can also break the security of the PRG (distinguish it from random).

Proof of Security

Polynomially indistinguishable? Same distribution

(1) Enc(m_1) with PRG

(2) Enc(m_2) with PRG

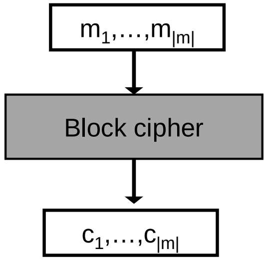
(3) Enc(m_1) with one-time pad

(4) Enc(m_2) with one-time pad

- Suppose that there is a D() which distinguishes between (1) and (2)
- No D() can distinguish between (3) and (4)
- We are given a string S and need to decide whether it is drawn from a pseudorandom distribution or from a uniformly random distribution
- Choose a random $b \in \{1,2\}$ and compute $m_b \oplus S$. Give the result to D().
- If D() outputs b then declare “pseudorandom”, otherwise declare “random”!

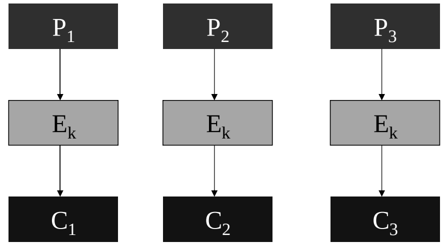
Block Ciphers

- 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.)



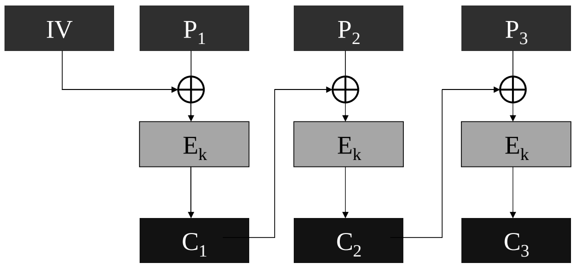
- Ideally, use a *random* permutation.
  - Can only be implemented using a table with $2^{|m|}$ entries ☹
- Instead, use a *pseudo-random* permutation, keyed by a key k.
  - Implemented by a computer program whose input is m,k.
- How can we encrypt longer inputs? different modes of operation were designed for this task.

ECB Encryption Mode (Electronic Code Book)



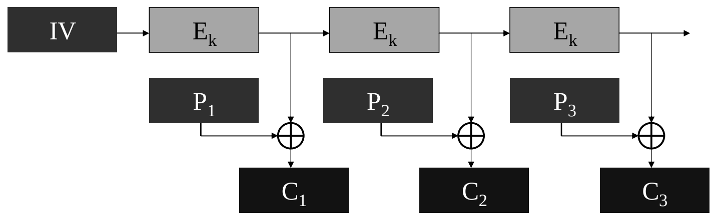
Namely, encrypt each plaintext block separately.

CBC Encryption Mode (Cipher Block Chaining)



Previous *ciphertext* is XORed with current *plaintext* before encrypting current block.
An initialization vector IV is used as a “seed” for the process.
IV can be transmitted in the clear (unencrypted).

OFB Mode (Output FeedBack)



- An initialization vector IV is used as a “seed” for generating a sequence of “pad” blocks
  - $E_k(IV)$, $E_k(E_k(IV))$, $E_k(E_k(E_k(IV)))$,…
- Essentially a one time pad

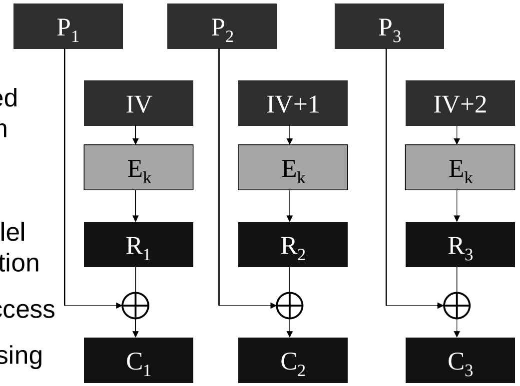
Properties of OFB

- Synchronous stream cipher. I.e., the two parties must know s_0 and the current bit position. ☹
- The parties must synchronize the location they are encrypting/decrypting. ☹
- Errors in ciphertext do not propagate ☺
- Implementation:
  - Pre-processing is possible ☺
  - No parallel implementation known ☹
  - No random access ☹
- Conceals plaintext patterns ☺
- Active attacks (by manipulating the plaintext) are possible ☹

CTR (counter) Encryption Mode

IV is selected as a random value

- easy parallel implementation
- random access
- preprocessing



Design of Block Ciphers

- More an art/engineering challenge than science. Based on experience and public scrutiny.
  - *"Diffusion":* each intermediate/output bit affected by many input bits
  - *"Confusion"*: avoid structural relationships between bits
- Cascaded (round) design: the encryption algorithm is composed of iterative applications of a simple round

Confusion-Diffusion and Substitution-Permutation Networks

- Divide the input to small parts, and apply rounds:
  - Feed the parts through random functions *("confusion")*
  - Mix the parts *("diffusion")*
  - Repeat
- Why both confusion and diffusion are necessary?
- Design choices: Avalanche effect. Using reversible s-boxes.

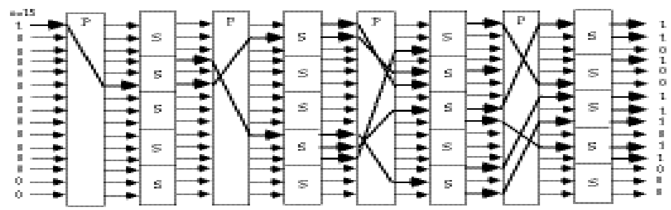


Fig 2.3 - Substitution-Permutation Network with the Avalanche Characteristic

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

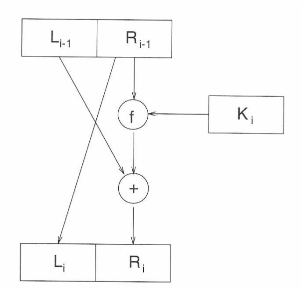
Rijndael animation

Reversible s-boxes

- Using reversible s-boxes
  - Allows for easy decryption
- However, we want the block cipher to be “as random as possible”
  - s-boxes need to have some structure to be invertible
- Enter Feistel networks
  - A round-based block-cipher which uses s-boxes which are not necessarily invertible

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 4 rounds give 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

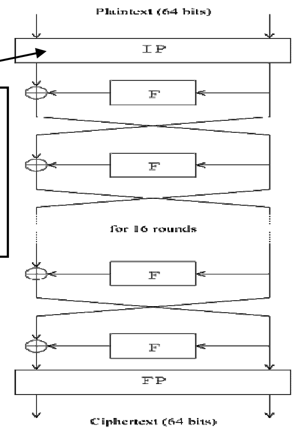
- Criticized for unpublished design *decisions* (designers did not want to disclose differential cryptanalysis).
- Very secure – the best attack in practice is brute force
  - 2006: $1 million search machine: 30 seconds
    - cost per key: less than $1
  - •2006: 1000 PCs at night: 1 month
    - Cost 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.

DES diagram (Data Encryption Standard)

Initial permutation of bit locations:

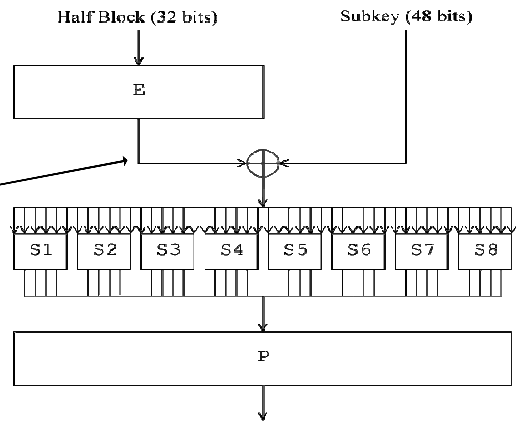
- not secret

- makes implementations in software less efficient



DES F functions

Expansion to 48 bits



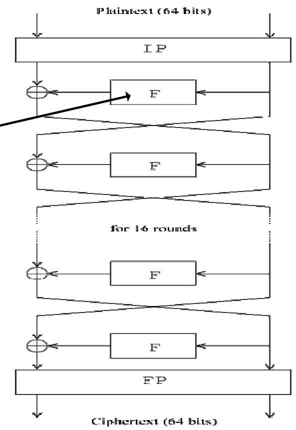
The S-boxes

- Very careful design (it is now clear that random choices for the S-boxes result in weak encryption).
- Each s-box maps 6 bits to 4 bits:
  - A 4×16 table of 4-bit entries.
  - Bits 1 and 6 choose the row, and bits 2-5 choose column.
  - Each row is a *permutation* of the values 0,1,…,15.
    - Therefore, given an output there are exactly 4 options for the input
  - Changing one input bit changes at least two output bits ⇒ avalanche effect.

Differential Cryptanalysis of DES

DES diagram:

S-boxes



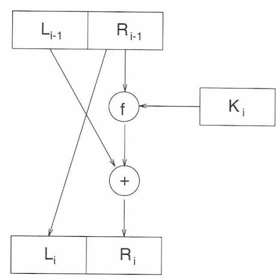
Differential Cryptanalysis [Biham-Shamir 1990]

- The first attack to reduce the overhead of breaking DES to below exhaustive search
- Very powerful when applied to other encryption algorithms
- Depends on the structure of the encryption algorithm
- Observation: all operations except for the s-boxes are linear
- Linear operations:
  - $a = b \oplus c$
  - *a* = the bits of *b* in (known) permuted order
- Linear relations can be exposed by solving a system of linear equations

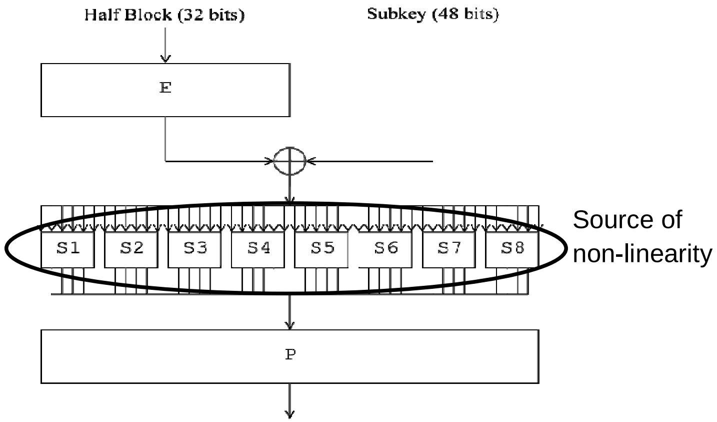
A Linear F in a Feistel Network?

- Suppose $F(R_{i-1},K_i) = R_{i-1} \oplus K_i$
  - Namely, that F is linear
- Then $R_i = L_{i-1} \oplus R_{i-1} \oplus K_i$

  $L_i = R_{i-1}$
- Write L_{16}, R_{16} as linear functions of L_0, R_0 and K.
  - Given L_0R_0 and $L_{16}R_{16}$ Solve and find K.
- F must therefore be non-linear.
- F is the only source of non-linearity in DES.



DES F functions



Differential Cryptanalysis

- The S-boxes are non-linear
- We study the differences between two encryptions of two different plaintexts

- Notation:
  - The plaintexts are P and P*
  - Their difference is $dP = P \oplus P^*$
  - Let X and X* be two intermediate values, for P and P*, respectively, in the encryption process.
  - Their difference is $dX = X \oplus X^*$
    - Namely, dX is always the result of two inputs

The advantage of looking at XORs

- It's easy to predict the difference of the results of linear operations
- Unary operations, (e.g. P is a permutation of the order of the bits of X)
  - $dP(x) = P(x) \oplus P(x^*) = P(x \oplus x^*) = P(dx)$
- XOR
  - $d(x \oplus y) = (x \oplus y) \oplus (x^* \oplus y^*) = (x \oplus x^*) \oplus (y \oplus y^*) = dx \oplus dy$
- Mixing the key
  - $d(x \oplus k) = (x \oplus k) \oplus (x^* \oplus k) = x \oplus x^* = dx$
  - The result here is key independent (the key disappears)

Differences and S-boxes

- S-box: a function (table) from 6 bit inputs to 4 bit output

- X and X* are inputs to the same S-box, and we know their difference $dX = X \oplus X^*$.

- $Y = S(X)$
- When dX=0, X=X*, and therefore $Y=S(X)=S(X^*)=Y^*$, and dY=0.
- When $dX \neq 0$, $X \neq X^*$ and we don't know dY for sure, but we can investigate its distribution.

- For example,

Distribution of Y' for S1

- dX=110100
- 2^6=64 input pairs, { (000000,110100), (000001,110101),…}
- For each pair compute xor of outputs of S1
- E.g., S1(000000)=1110, S1(110100)=1001. dY=0111.
- Table of frequencies of each dY:

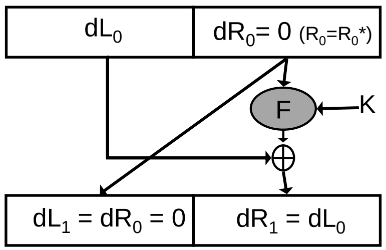
0000	0001	0010	0011	0100	0101	0110	0111
0	8	16	6	2	0	0	12
1000	1001	1010	1011	1100	1101	1110	1111
6	0	0	0	0	8	0	6

Differential Probabilities

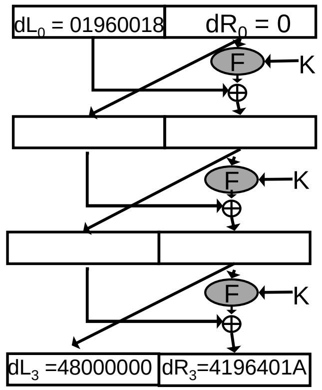
- The probability of dX $\Rightarrow$ dY is the probability that a pair of difference dX results in a pair of difference dY (for a given S-box).
- Namely, for dX=110100 these are the entries in the table divided by 64.
- Differential cryptanalysis uses entries with large values
  - dX=0 $\Rightarrow$ dY=0
  - Entries with value 16/64
  - (Recall that the values in the S-box are uniformly distributed, so the attacker gains a lot by looking at diffs.)

Warmup

Inputs: L_0R_0, $L_0^*R_0^*$, s.t. $R_0=R_0^*$.
Namely, inputs whose xor is dL_0 0

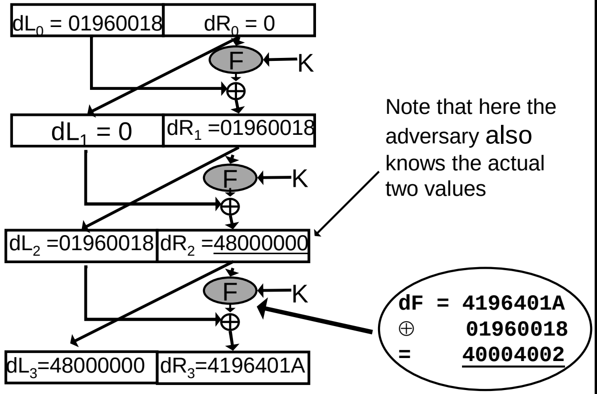


3 Round DES

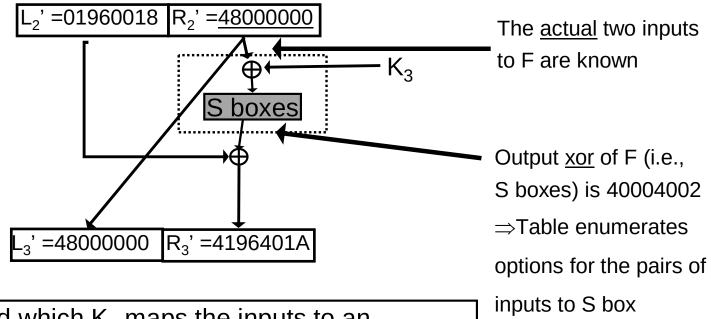


The attacker knows the two plaintext/ciphertext pairs, and therefore also their differences

Intermediate differences equal to plaintext/ciphertext differences



Finding K



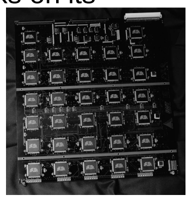
Find which K_3 maps the inputs to an s-box input pair that results in the output pair!

DES with more than 3 rounds

- Carefully choose pairs of plaintexts with specific xor, and determine xor of pairs of intermediate values at various rounds.
- E.g., if dL_0=40080000$_x$, dR_0=04000000$_x$
  Then, with probability ¼, dL_3=04000000$_x$, dR_3=4008000$_x$
- 8 round DES is broken given 2^{14} chosen plaintexts.
- 16 round DES is broken given 2^{47} chosen plaintexts...

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, check if $D_{k2}(c)$ is in the table
  - Might obtain several options for (k_1,k_2). Check them or repeat the process again with a new *(m,c)* pair.
- The attack is applicable to any iterated cipher

Meet-in-the-middle attack

- The plaintext and the ciphertext are 64 bits long
- The key is 56 bits long
- Suppose that we are given two plaintext-ciphertext pairs (m,c) (m',c')
- The attack looks for k1,k2, such that $D_{k2}(c) = E_{k1}(m)$ and $D_{k2}(c') = E_{k1}(m')$
- The correct value of k1,k2 satisfies both equalities
- There are 2^{112} (actually $2^{112}-1$) other values for k1,k2.
- Each one of these satisfies the equalities with probability 2^{-128}
- The probability that there exists one or more of these other pairs of keys, which satisfy both equalities, is bounded from above by $2^{112-128} = 2^{-16}$.

Triple DES

- $3DES_{k1,k2} = E_{k1}(D_{k2}(E_{k1}(m)))$
- Why use Enc(Dec(Enc( ))) ?
  - Backward compatibility: setting $k_1=k_2$ is compatible with single key DES
- Only two keys
  - Effective key length is 112 bits
  - Why not use three keys? There is a meet-in-the-middle attack with 2^{112} operations
- 3DES provides good security. Widely used. Less efficient.