Introduction to Cryptography

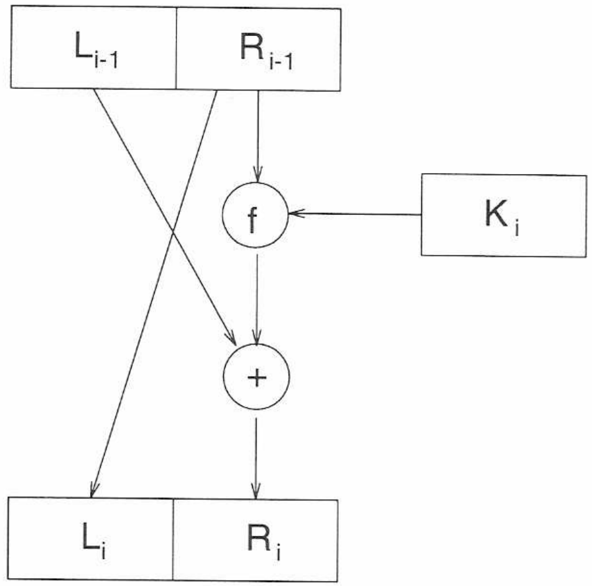
Lecture 4

Benny Pinkas

Feistel Networks

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

- No matter which function is used as F, we obtain a permutation (i.e., F is reversible even if *f* is not).



DES (Data Encryption Standard)

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

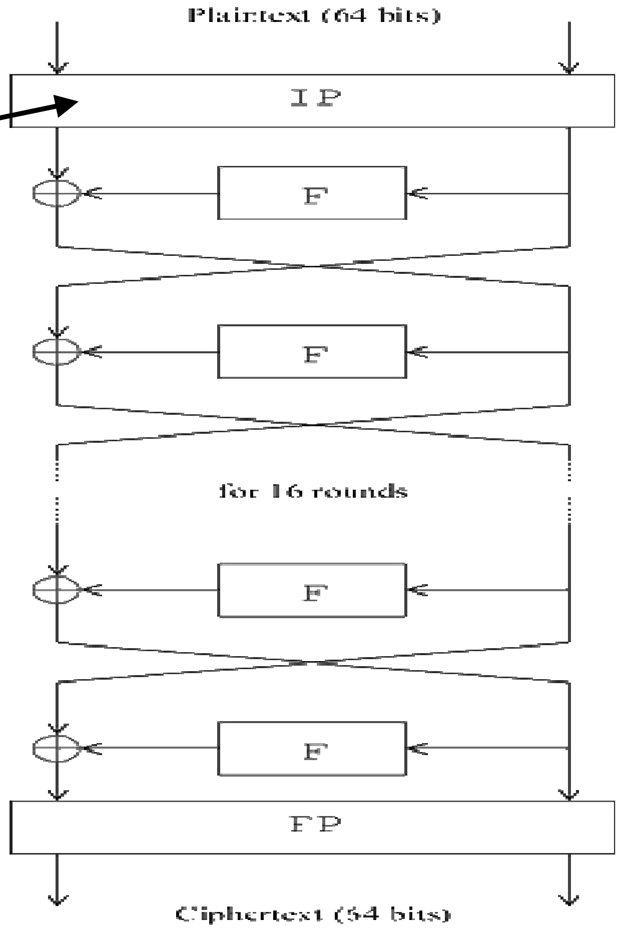
- Throughput ≈ software: 10Mb/sec, hardware: 1Gb/sec (in 1991!).

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- Criticized for unpublished design *decisions* (designers did not want to disclose differential cryptanalysis).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.

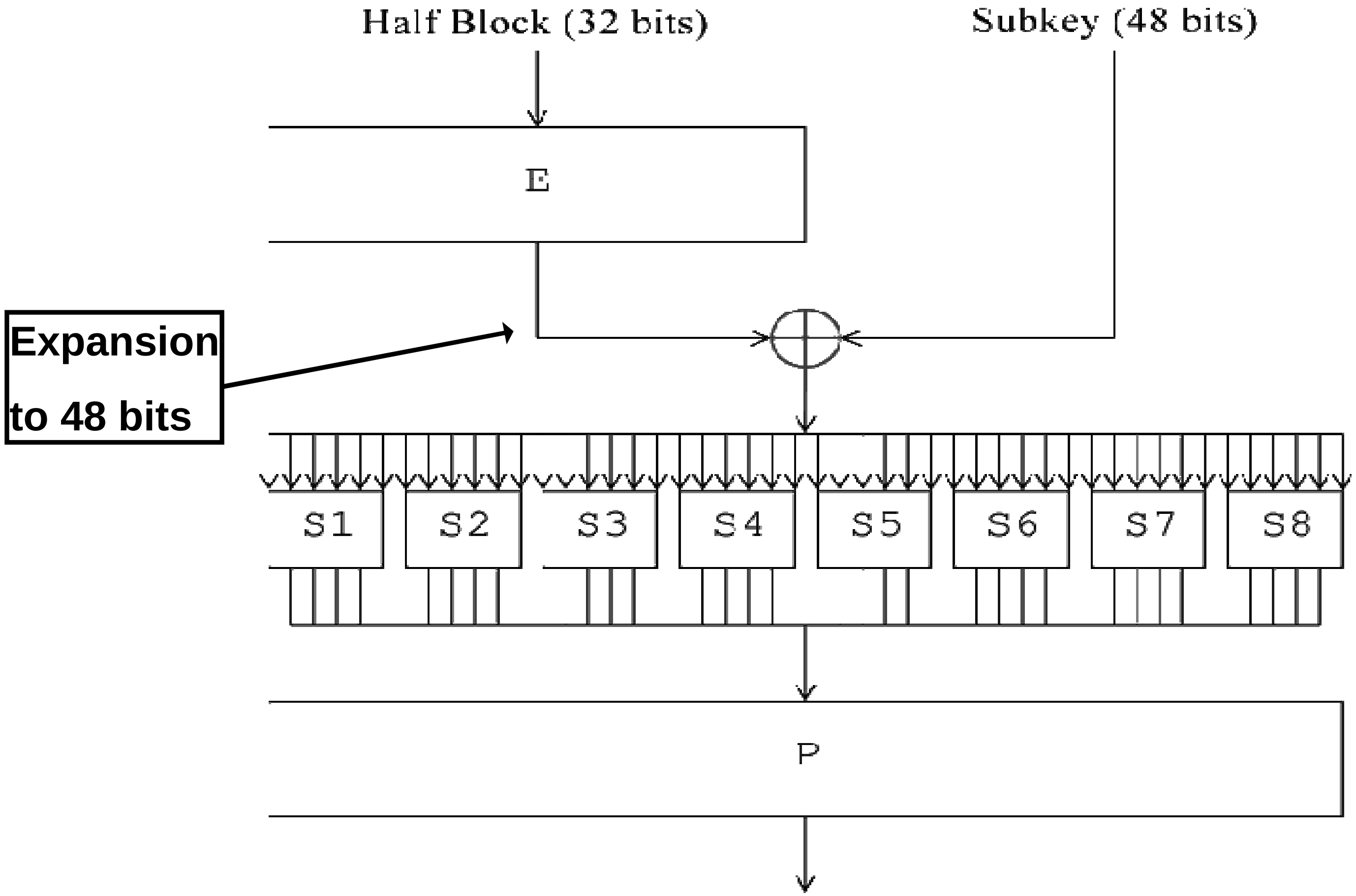
DES diagram (Data Encryption Standard)



Initial permutation of bit locations:

- not secret
- makes implementations in software less efficient

DES F functions



The S-boxes

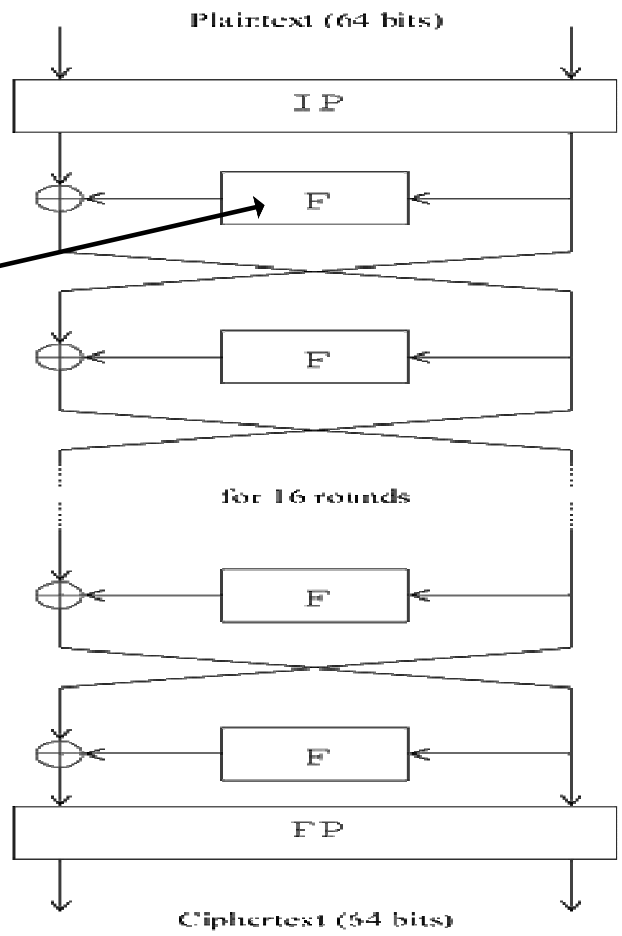
- Very careful design (it is now clear that random choices for the S-boxes result in weak encryption).

- 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 $\Rightarrow$ avalanche effect.

Differential Cryptanalysis of DES

DES diagram:

S-boxes



Differential Cryptanalysis [Biham-Shamir 1990]

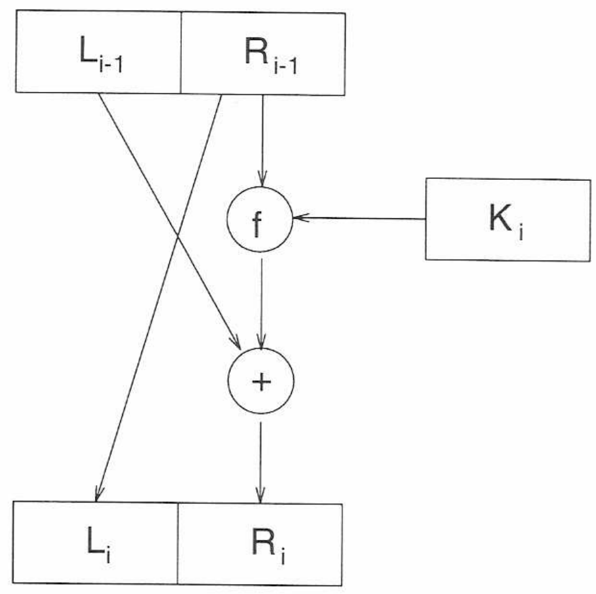
- The first attack to reduce the overhead of breaking DES to below exhaustive search
- Very powerful when applied to other encryption algorithms

- 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

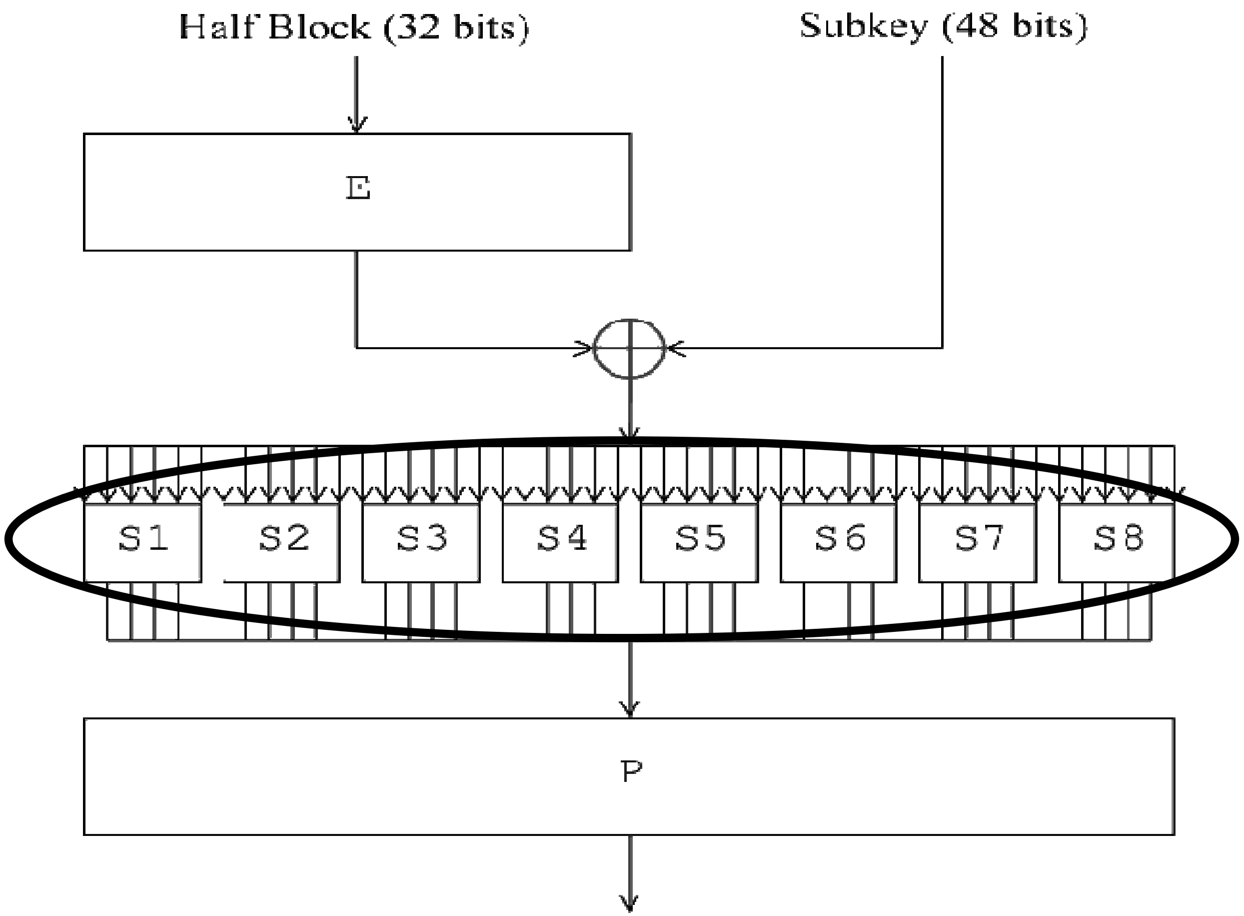
A Linear F in a Feistel Network?

- 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



Source of non-linearity

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 $dX\neq0$, $X\neq X^*$ 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

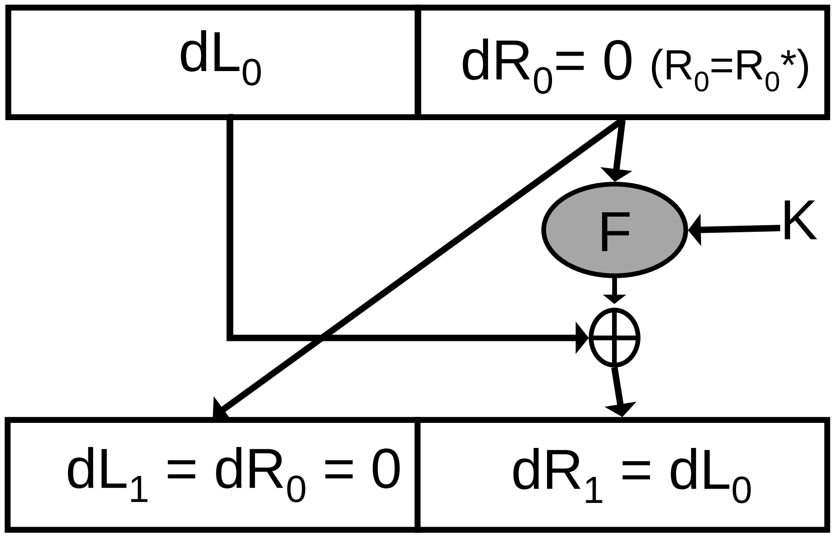
Differential Probabilities

- 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.

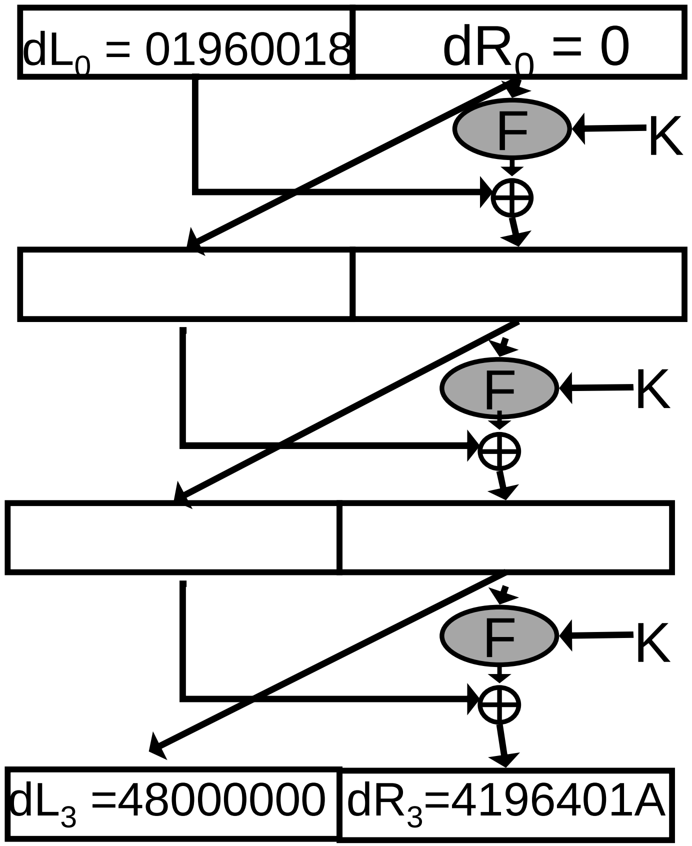
- Differential cryptanalysis uses entries with large values
  - dX=0 $\Rightarrow$ dY=0
  - Entries with value 16/64
  - (Recall that the values in the S-box are uniformly distributed, so the attacker gains a lot by looking at diffs.)

Warmup

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

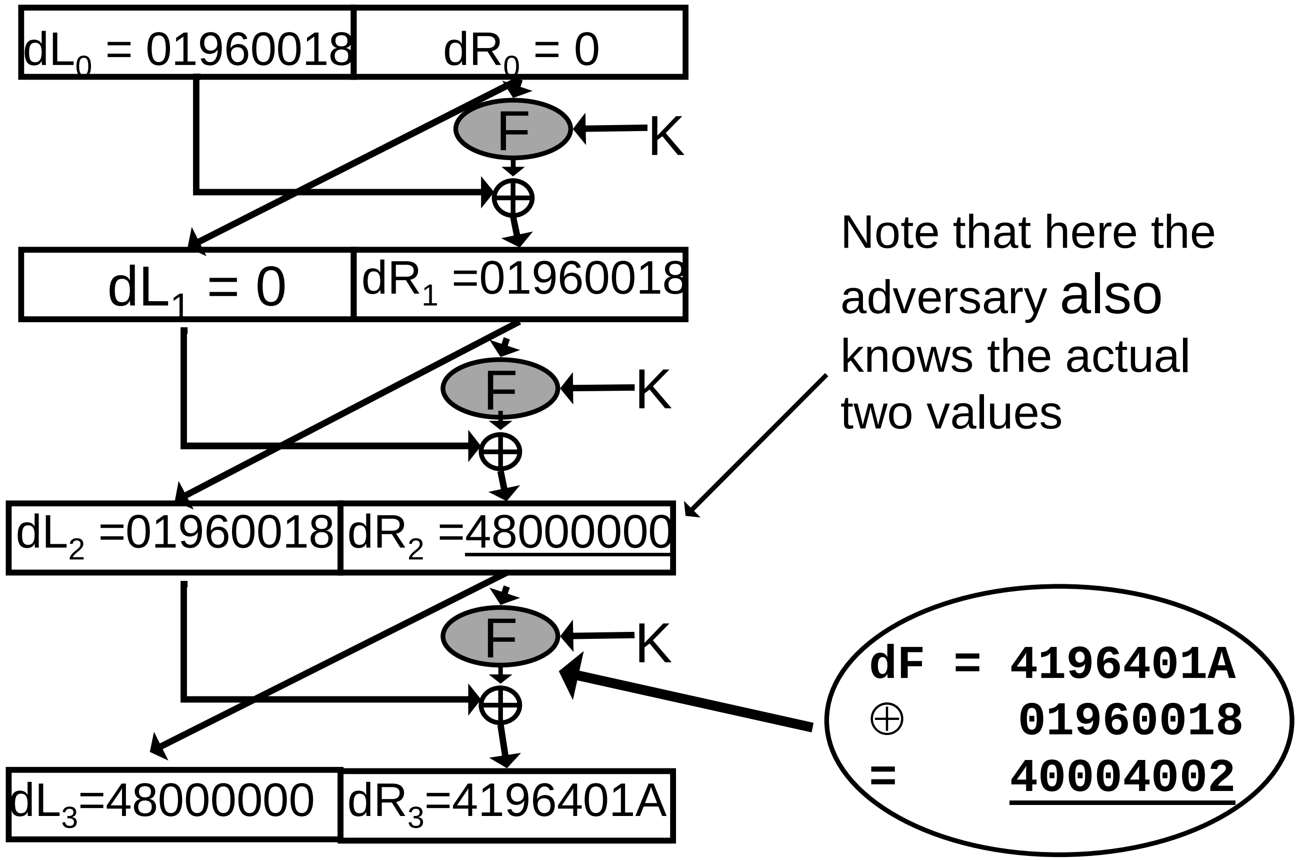


3 Round DES

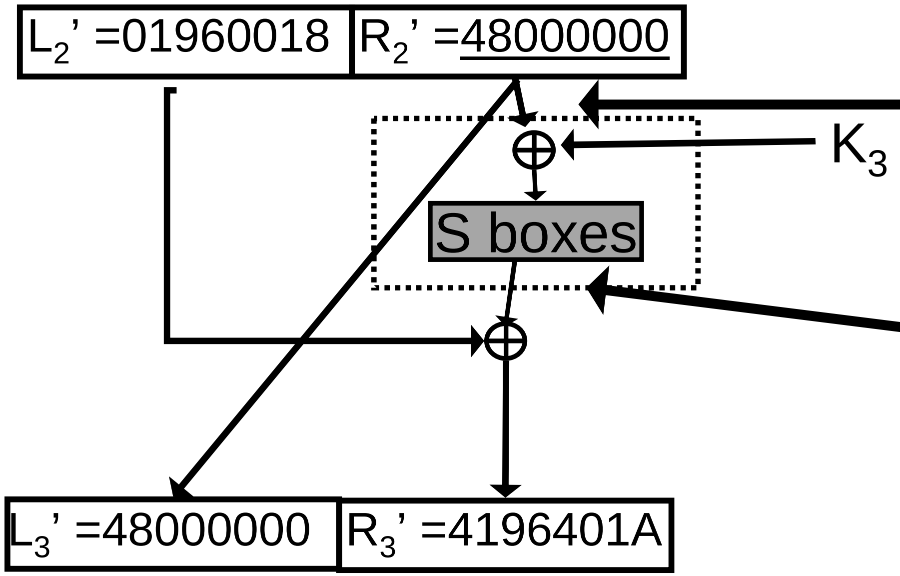


The attacker knows the two plaintext/ciphertext pairs, and therefore also their differences

Intermediate differences equal to plaintext/ciphertext differences



Finding K



The actual two inputs to F are known

Output xor of F (i.e., S boxes) is 40004002

⇒Table enumerates options for the pairs of inputs to S box

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

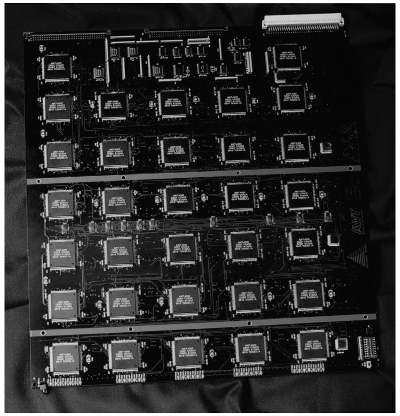
DES with more than 3 rounds

- 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, 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 satisfies 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.

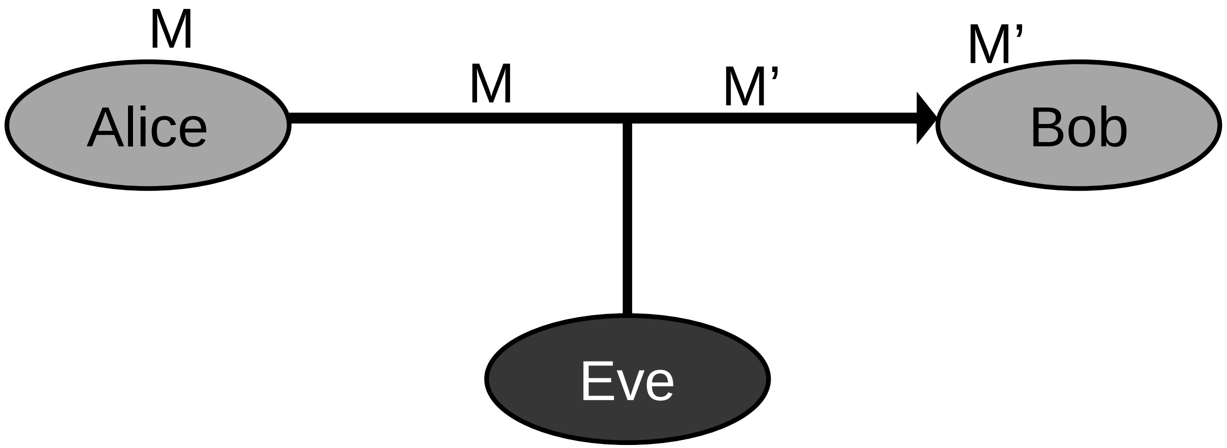
- Suppose that we are given one pairs (m,c), (m',c')
  - The correct values of k1,k2 satisfies both equalities
  - There are 2^{112} (actually $2^{112}-1$) other values for k_1,k_2.
  - Each one of these satisfies the equalities with probability 2^{-128}
  - We therefore expect to have $2^{112-128}<1$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

Triple DES

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

Data Integrity, Message Authentication

- Risk: an *active* adversary might change messages exchanged between Alice and Bob



• Authentication is orthogonal to secrecy. A relevant challenge regardless of whether encryption is applied.

One Time Pad

- OTP is a perfect cipher, yet provides no authentication
  - Plaintext $x_1x_2\ldots x_n$
  - Key $k_{1k2}\ldots k_n$
  - Ciphertext $c_1=x_1\oplus k_{1,}$ $c_2=x_2\oplus k_2,\ldots,c_n=x_n\oplus k_n$

- Adversary changes, e.g., c_2 to $1\oplus c_2$
- User decrypts $1\oplus x_2$

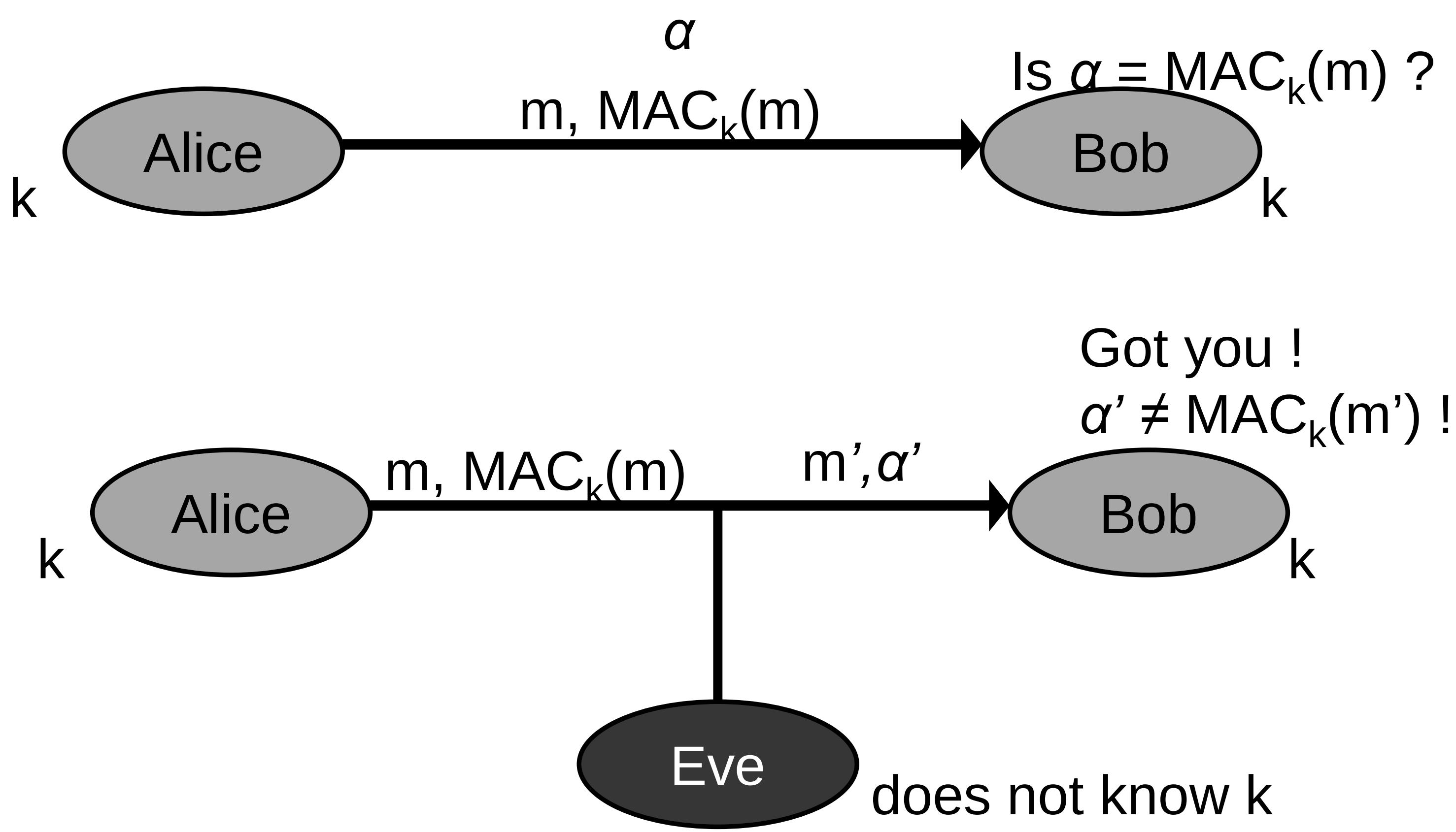
- Error-detection codes are insufficient. (For example, linear codes can be changed by the adversary, even if encrypted.)
  - They were not designed to withstand adversarial behavior.

Definitions

- Scenario: Alice and Bob share a secret key K.
- Authentication algorithm:
  - Compute a Message Authentication Code: $\alpha = MAC_K(m)$.
  - Send m and α
- Verification algorithm: $V_K(m, \alpha)$.
  - $V_K(m, MAC_K(m)) = accept.$
  - For $\alpha \neq MAC_K(m)$, $V_K(m, \alpha) = reject.$

- How does $V_k(m)$ work?
  - Receiver knows k. Receives m and α.
  - Receiver uses k to compute $MAC_K(m)$.
  - $V_K(m, \alpha) = 1$ iff $MAC_K(m) = \alpha$.

Common Usage of MACs for message authentication



Requirements

- Security: The adversary,
  - Knows the MAC algorithm (but not K).
  - Is given many pairs $(m_i, MAC_K(m_i))$, where the m_i values might also be chosen by the adversary (chosen plaintext).
  - Cannot compute $(m, MAC_K(m))$ for any new m $(\forall i\ m \neq m_i)$.
  - The adversary must not be able to compute $MAC_K(m)$ *even* for a message m which is “meaningless” (since we don’t know the context of the attack).

- Efficiency: output must be of fixed length, and as short as possible.
  - $\Rightarrow$ The MAC function is not 1-to-1.
  - $\Rightarrow$ An n bit MAC can be broken with prob. of at least 2^{-n}.

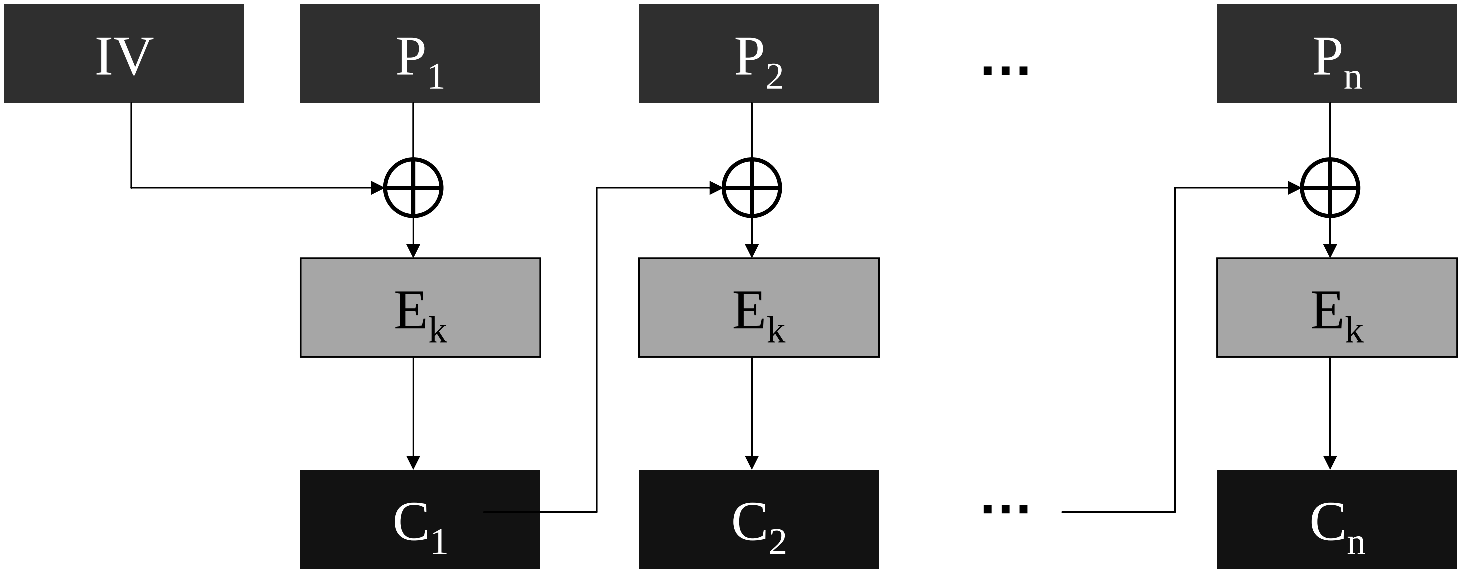
Constructing MACs

- Based on block ciphers (CBC-MAC)

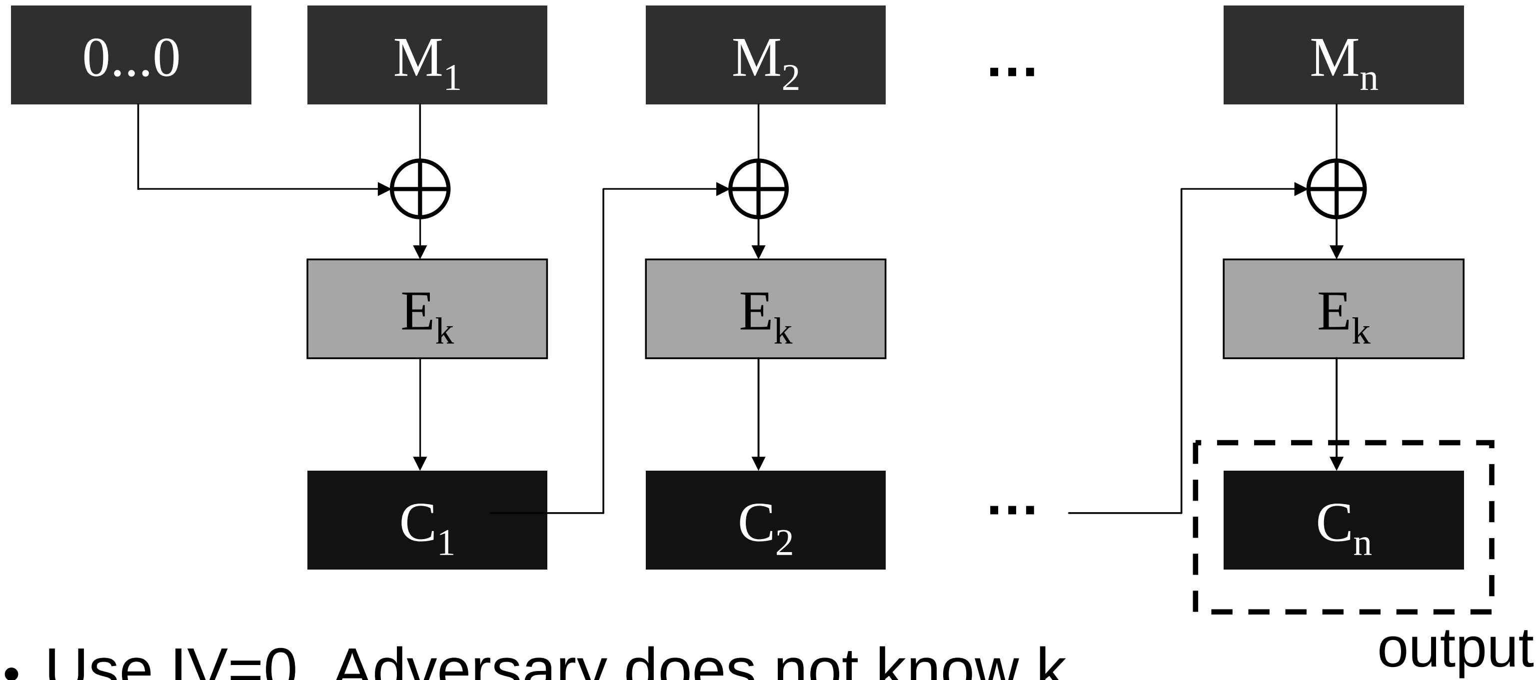
  or,
- Based on hash functions
  - More efficient
  - At the time, encryption technology was controlled (export restricted) and it was preferable to use other means when possible.

CBC

- Reminder: CBC encryption
- Plaintext block is xored with previous ciphertext block



CBC MAC



- Use IV=0. Adversary does not know k.
- Encrypt M in CBC mode, using the MAC key. Discard $C_1,\ldots,C_{n-1}$ and define $MAC_K(M_1,\ldots,M_n)=C_n$.

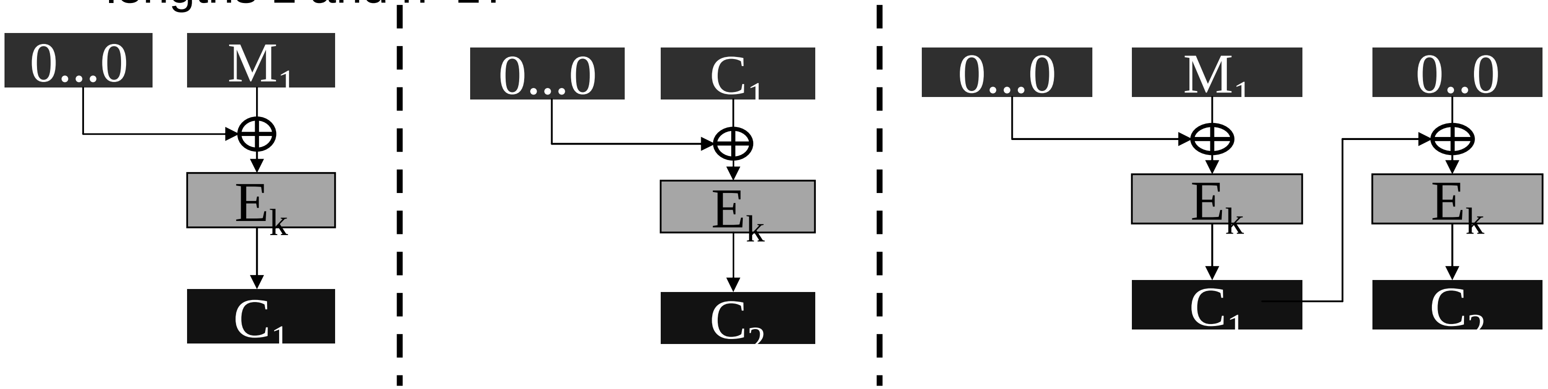
Security of CBC-MAC

- Claim: if E_K is pseudo-random then CBC-MAC, applied to *fixed length messages*, is a pseudo-random function, and is therefore resilient to forgery.

- But, insecure if variable lengths messages are allowed

Security of CBC-MAC

- Insecurity of CBC-MAC when applied to messages of variable length:
  - Get $C_1 = \text{CBC-MAC}_K(M_1) = E_K(0 \oplus M_1)$
  - Ask for MAC of C_1, i.e., $C_2 = \text{CBC-MAC}_K(C_1) = E_K(0 \oplus C_1)$
  - But, $E_K(C_1 \oplus 0) = E_K(E_K(0 \oplus M_1) \oplus 0) = \text{CBC-MAC}_K(M_1 \mid 0)$
    - It's known that CBC-MAC is secure if message space is prefix-free.
    - Can you show, for every n, a collision between two messages of lengths 1 and n+1?



CBC-MAC for variable length messages

- Solution 1: The first block of the message is set to be its length. I.e., to authenticate $M_1,\ldots,M_n$, apply CBC-MAC to $(n,M_1,\ldots,M_n)$.
  – Works since now message space is prefix-free.
  – Drawback: The message length (n) must be known in advance.
- "Solution 2": apply CBC-MAC to $(M_1,\ldots,M_n,n)$
  – Message length does not have to be known is advance
  – But, this scheme is broken (see, M. Bellare, J. Kilian, P. Rogaway, The Security of Cipher Block Chaining, 1984)
- Solution 3: (preferable)
  – Use a second key K'.
  – Compute $MAC_{K,K'}(M_1,\ldots,M_n) = E_{K'}(MAC_K(M_1,\ldots,M_n))$
  – Essentially the same overhead as CBC-MAC