Symmetric Key Cryptography

Model of symmetric key cryptography

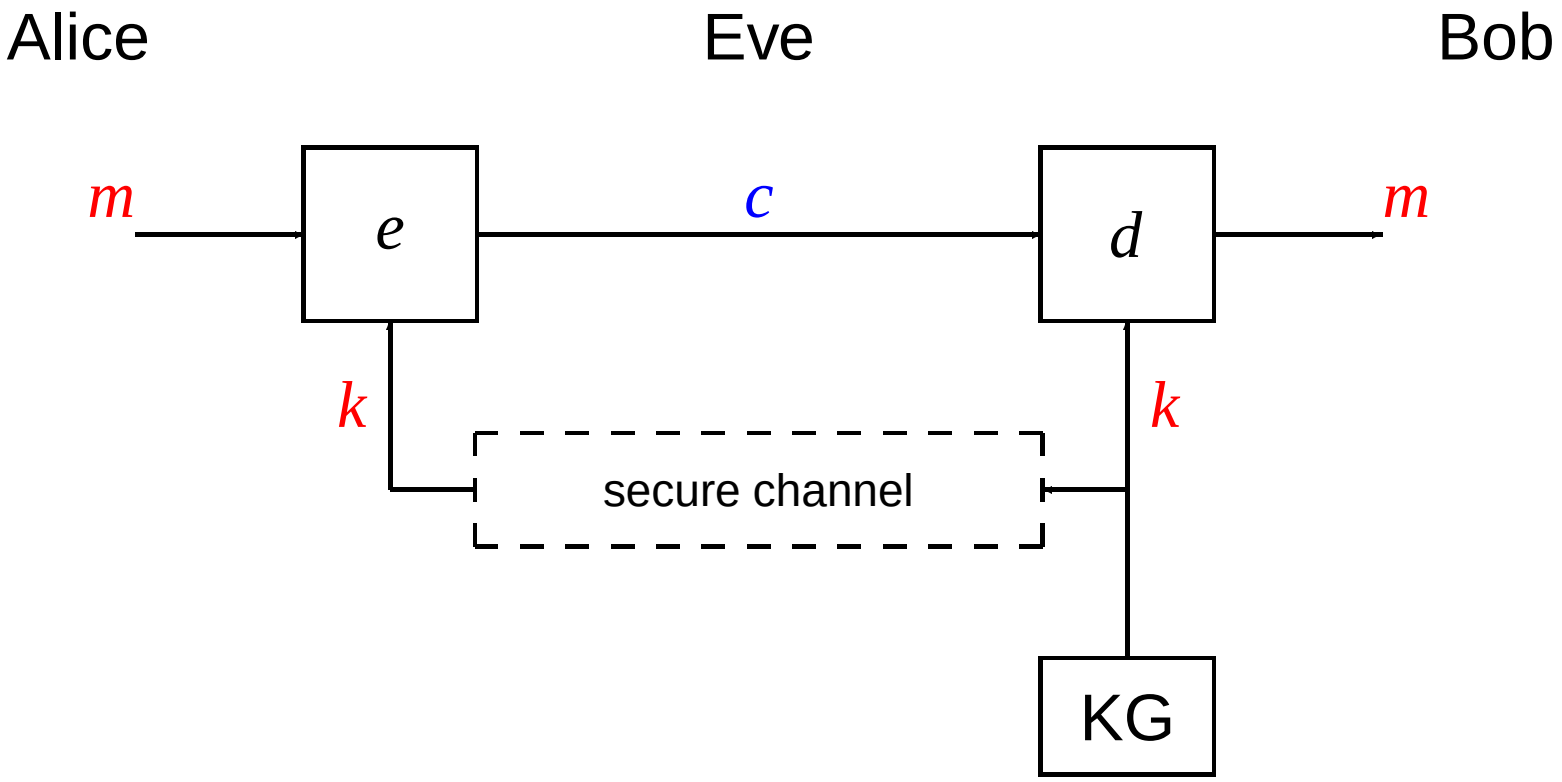
Symmetric Encryption

- Stream ciphers
- Block ciphers

Symmetric Authentication

- Manipulation Detection Codes (MDC)
- Message Authentication Codes (MAC)

Model of Symmetric Key Cryptography



m = plaintext, c = ciphertext, k = key, KG = Key Generator.

Model of Symmetric Key Cryptography

We write $c = e_k(m)$, where

- m is the plaintext,
- e is the encryption function,
- k is the secret key,
- c is the ciphertext.

Decryption is given by $m = d_k(c)$.

Both sides need to know the key k, but k needs to be kept secret.

$\Rightarrow$ secret-key, single-key or one-key algorithms.

Kerckhoffs' principle:

- Cryptanalyst has complete knowledge of $e_\cdot(\cdot)$ and $d_\cdot(\cdot)$.
- Secrecy of message must reside entirely in the secrecy of the key.

Attacks on Symmetric Ciphers

Worst case assumptions: adversary has

- Full knowledge of the encryption algorithm $e_{\cdot}(\cdot)$ and decryption algorithm $d_{\cdot}(\cdot)$.
- Number of plaintext/ciphertext pairs associated to the target key k.

Exhaustive keysearch:

- Given a few plaintext/ciphertext pairs, search through all possible keys until the correct key is found.

$\Rightarrow$ The number of keys must be large enough.

In practice ciphers are used which are believed to be strong, which means that the best attempts of experienced cryptanalysts cannot break them.

Attacks on Symmetric Ciphers

Ciphertext-only attack:

- Adversary only has ciphertext of several messages.
- Recover the plaintext or deduce the keys used during encryption.

Known-plaintext attack:

- Adversary has several ciphertexts and corresponding plaintexts.
- Deduce keys used to encrypt messages or decrypt new messages.

Chosen-plaintext attack:

- Adversary can choose the plaintexts that get encrypted.
- More powerful than known-plaintext attack, because specific plaintexts can be chosen.

Attacks on Protocols

Passive Attack:

- Adversary only eavesdrops on protocol.
- Attempts to break the cryptosystem or to recover the key.
- Traffic analysis.

Defences:

- Passive attacks are difficult to detect, so one has to prevent them.
- Manage keys well.
- Only use secure ciphers and protocols.
- Avoid patterns of traffic.

Attacks on Protocols

Active Attack:

- Adversary inserts, deletes, substitutes, replays messages.
- Much stronger form of attack.

Defences:

- Design against insertion attack

$\Rightarrow$ Adversary should need to break the cipher.

- Design against deletion/substitution/replay attack

$\Rightarrow$ Need to be able to detect and recover.

Types of Symmetric Ciphers

Stream Ciphers: operate on plaintext one bit/byte at a time.

- Synchronous or additive stream ciphers: keystream is independent of the plaintext string, only depends on key.
- Self-synchronizing stream ciphers: keystream is function of key and ciphertext.

Block Ciphers: operate on blocks of data, e.g. 64/128 bits at a time.

- Feistel network structure.
- Substitution-permutation network structure.

Remark: different types have different error propagation and synchronisation problems.

Stream Ciphers

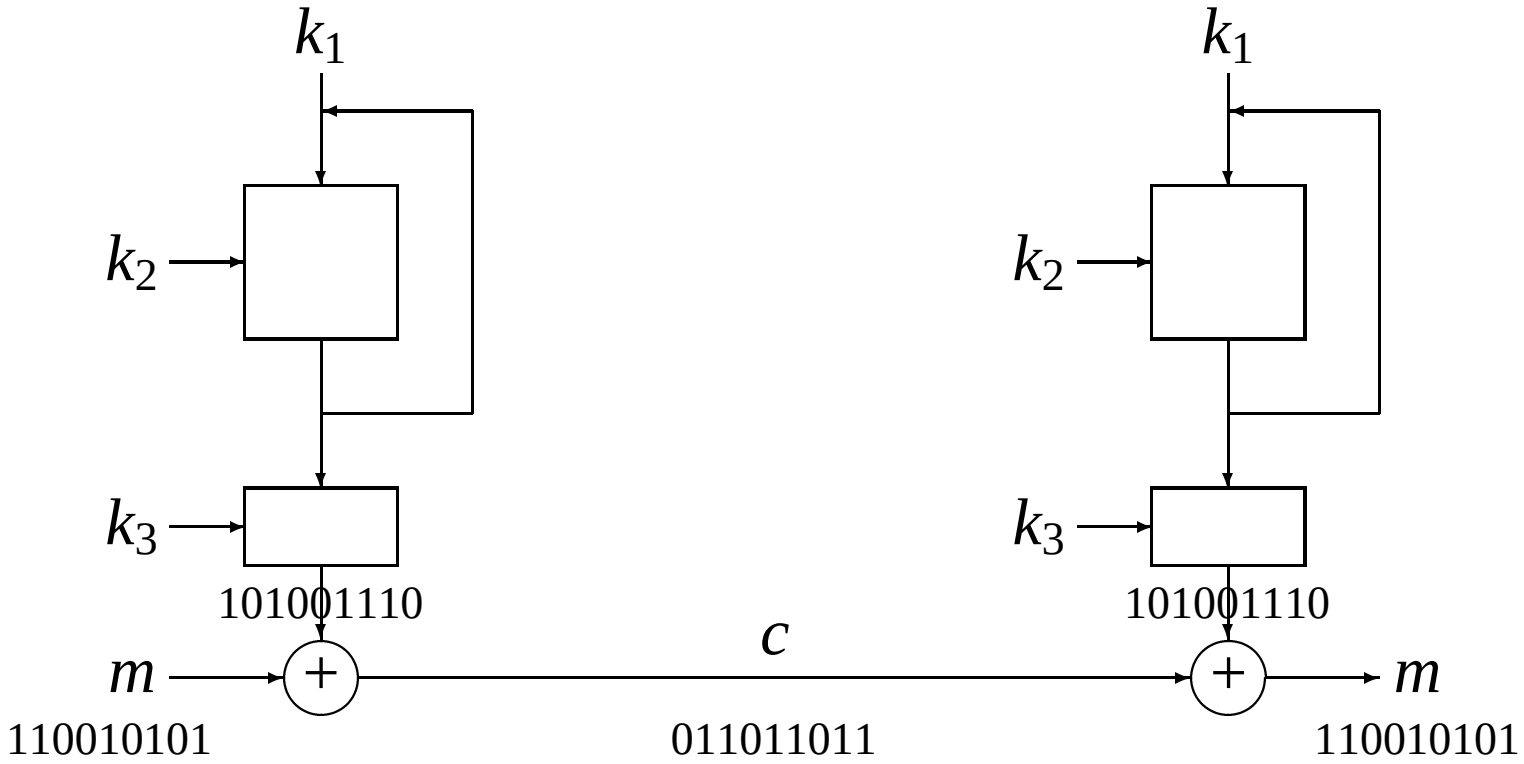
Basic idea: replace the random key in Vernam scheme by a pseudo-random sequence, generated by a *cryptographic* pseudo-random generator that is 'seeded' with the key.

$\Rightarrow$ Definition of synchronous stream cipher

Properties:

- Short key, but only practical security.
- Internal memory in encipherment.
- Encryption in small quantities (bit/byte).
- No error propagation.

Model of Stream Cipher



Finite state machine with output filter.

Keys determine state update (k_1), initial state (k_2) and output filter k_3.

Model of Stream Ciphers

Thus we have $c_i = m_i \oplus k_i$ where

- $m_0, m_1, \ldots$ are the plaintext bits/bytes.
- $k_0, k_1, \ldots$ are the keystream bits/bytes.
- $c_0, c_1, \ldots$ are the ciphertext bits/bytes.

This means $m_i = c_i \oplus k_i$ and thus decryption is the same as encryption.

- Encryption/decryption can be very fast ($>$ 50 MB/s on PC).
- No error propogation: one error in ciphertext gives one error in plaintext.
- Loss of synchronization $\Rightarrow$ decryption fails for remaining ciphertext.
- No protection against message manipulation.
- Same key used twice gives same keystream.

Keystream Requirements

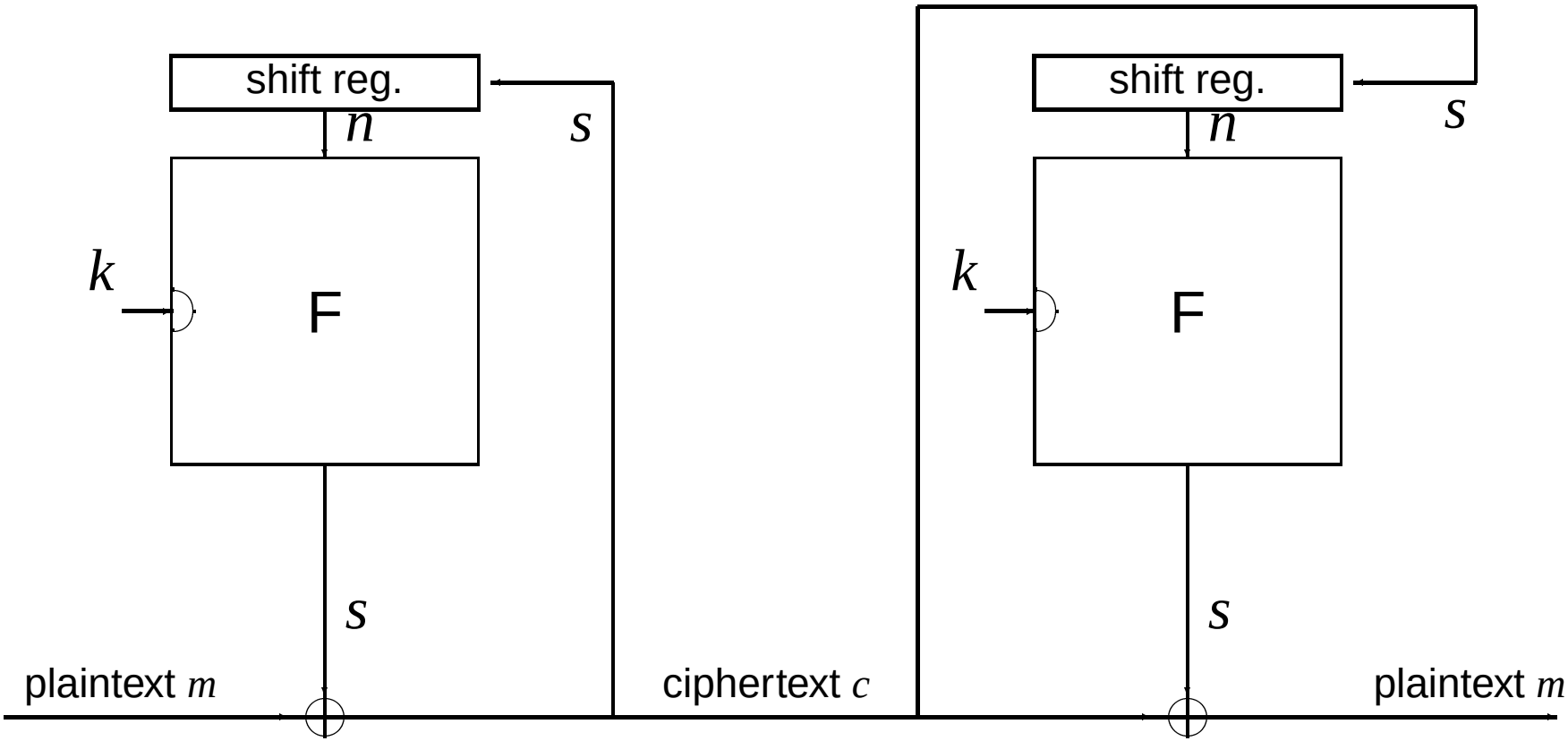
For the stream cipher to be secure, the keystream must

- Look random, i.e. pass pseudo-random tests.
- Be unpredictable, i.e. have a long period.
- Have large linear complexity (see textbooks).
- Have low correlation between key bits and keystream bits.

Furthermore, the keystream should be efficient to generate.

Remark: most stream ciphers are based on a non-linear combination of LFSR (Linear Feedback Shift Register).

Self-synchronizing Stream Ciphers



Self-synchronizing: if s bits/bytes are transmitted correctly, the next bit/byte will be decrypted correctly.

Example: ‘Alleged’ RC4

RC stands for Ron’s Cipher after Ron Rivest (MIT) of RSA fame.

RC4 is NOT a prior version of the block ciphers RC5 and RC6.

RC4 is very, very fast stream cipher and very easy to remember.

Internal state is an array $S[0 \ldots 255]$ which consists of the integers $0, \ldots, 255$ permuted in some key dependent way and which evolves in an unpredictable, non-linear way.

The output of the RC4 algorithm is a keystream of bytes which is XOR-ed with the plaintext byte by byte.

Example: ‘Alleged’ RC4 - Initialisation

Initialise a 256 byte array $K[0 \ldots 255]$ with the key k, repeating the key if necessary to fill the array.

Then execute the following loops:

```
for i=0 to 255 do S[i] = i;
j = 0;
for i=0 to 255 do
  j = (j + S[i] + K[i]) % 256;
  swap (S[i], S[j]);
```

Example: ‘Alleged’ RC4 - Encryption

Initialise with `i=0; j=0;`.

Generation of one byte of keystream Ks:

```
i = (i + 1)%256;
j = (j + S[i])%256;
swap (S[i], S[j]);
t = (S[i] + S[j])%256;
Ks = S[t];
```

Each line needs to be there to make the cipher secure.

Assume the adversary knows Ks and i.

- Therefore she knows $S[t]$.
- But she does not know t since she does not know $j, S[i]$ or $S[j]$.

Example: ‘Alleged’ RC4 - Security

`i=(i + 1)%256`:

- Makes sure every array element is used once after 256 iterations.

`j=(j + S[i])%256`:

- Makes the ouput depend non-linearly on the array.

`swap(S[i], S[j])`:

- Makes sure the array is evolved as the iteration continues.

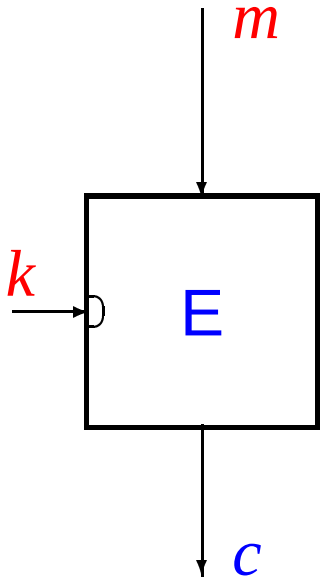
`t=(S[i] + S[j])%256`:

- Makes sure the output sequence reveals little about the the internal state of the array.

Block Ciphers

Plaintext is divided into blocks of fixed length and every block is encrypted one at a time.

A block cipher is a set of ‘Code Books’ and every key produces a different code book. The encryption of a plaintext block is the corresponding ciphertext block entry in the code book.



Block Ciphers

We have $c = e_k(m)$ where

- m is the plaintext block
- k is the secret key
- e is the encryption function
- c is the ciphertext

In addition we have a decryption function, $d_{\cdot}(\cdot)$, such that

$$m = d_k(c)$$

The block size n needs to be reasonably large, $n > 64$ bits, to avoid

- Text dictionary attacks: plaintext-ciphertext pairs for fixed key.
- Matching ciphertext attacks: uncover patterns in plaintext.

Types of Block Ciphers

Two building blocks of block ciphers

- Transposition cipher: permutes a block of n characters of plaintext.
- Substitution cipher: each character is replaced by other symbol.

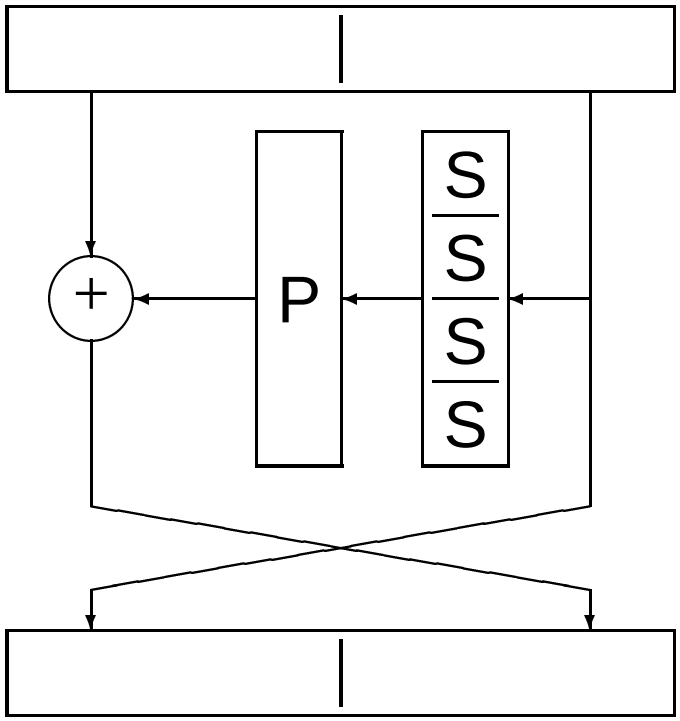
Most block ciphers are repeated application of a round function.

Type determined by combination of transposition/substitution

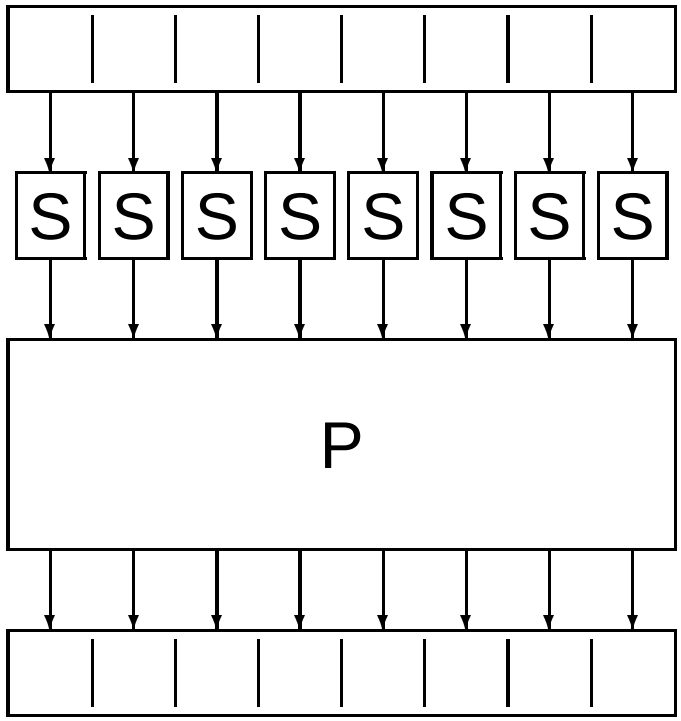
- Feistel cipher: input divided in two halves, right half becomes left half, left half is XOR-ed with function of right half and key.
- Substitution-Permutation network: input divided in blocks which are encrypted with substitution cipher, blocks are mixed using permutation and roundkey is XOR-ed.

Types of Block Ciphers

Different types of round functions:



Feistel Cipher



S/P-Network

Iterated Block Ciphers

An iterated block cipher involves repeated use of a round function.

The round function takes an n-bit block to an n-bit block.

Parameters: number of rounds r, blocksize n, keysize s.

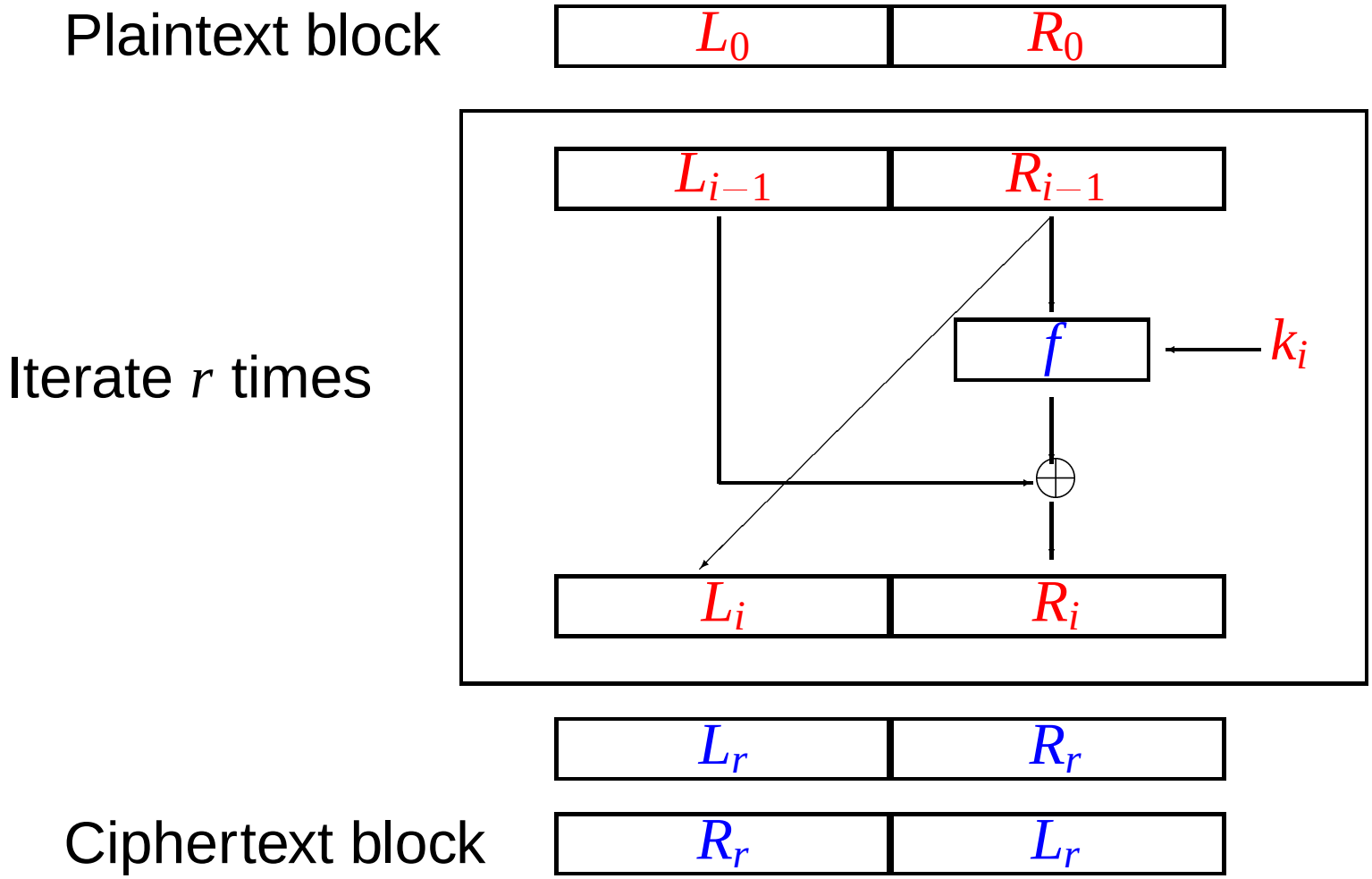
Each use of the round function employs a subkey

$$k_i \text{ for } 1 \leq i \leq r$$

derived from the key k.

For every subkey the round function must be invertible; if not decryption is impossible.

Feistel Cipher



Feistel Cipher

The round function is invertible regardless of the choice of f.

Encryption round is

- $L_i = R_{i-1}$
- $R_i = L_{i-1} \oplus f(k_i, R_{i-1})$

Remark: in last step, the left half and right half are swapped.

- Decryption is achieved using the same r-round process.
- But with round keys used in reverse order, i.e. k_r first and k_1 last !

Data Encryption Standard - DES

- First published in 1977 as a US Federal standard.
- Known as Data Encryption Algorithm DEA (ANSI) or DEA-1 (ISO).
- DES is a de-facto international standard for banking security.
- DES is an example of a Feistel Cipher.
- Most analysed cryptographic algorithm; known to be not a group.
- Short history of DES
  - Work started in early 1970's by IBM.
  - Based on IBM's Lucifer, but amended by NSA.
  - Design criteria were kept secret for more than 20 years.
  - Supposed to be reviewed every 5 years.

Overview of DES Operation

Block length is 64 bits, key length 56 bits.

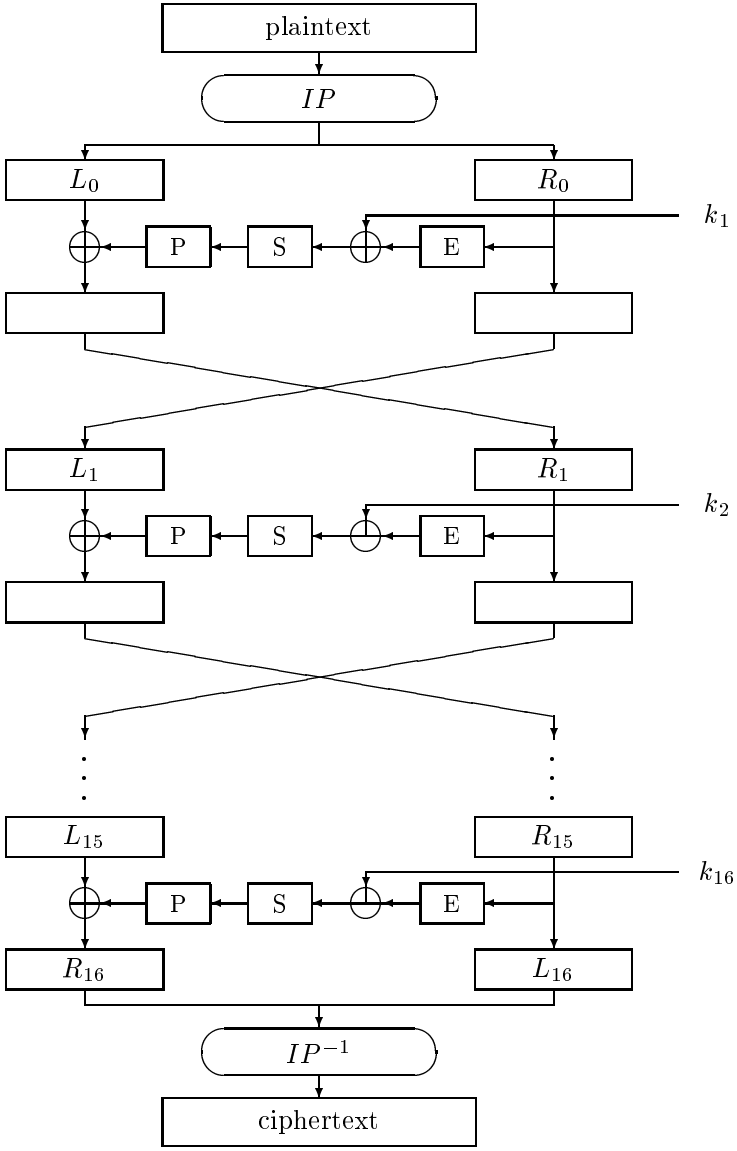
Feistel Cipher

- Initial permutation of bits.
- Split in left and right half.
- 16 rounds of identical operations, depending on round key.
- Inverse initial permutation.

Round transformation

- Linear expansion: 32 bits $\rightarrow$ 48 bits.
- XOR with subkey of 48 bits (key schedule selects 48 bits of key k).
- 8 parallel non-linear S-boxes (6 input bits, 4 output bits).
- Permutation of the 32 bits.

Data Encryption Standard



DES Round

Each DES round consists of the following six stages

1. Expansion Permutation
   - Right half (32 bits) is expanded (and permuted) to 48 bits.
   - Diffuses relationship of input bits to output bits.
   - Means one bit of input affects two substitutions in output.
   - Spreads dependencies.

2. Use of Round Key: 48 bits are XOR-ed with the round key (48 bits).

3. Splitting: result is split into eight lots of six bit values.

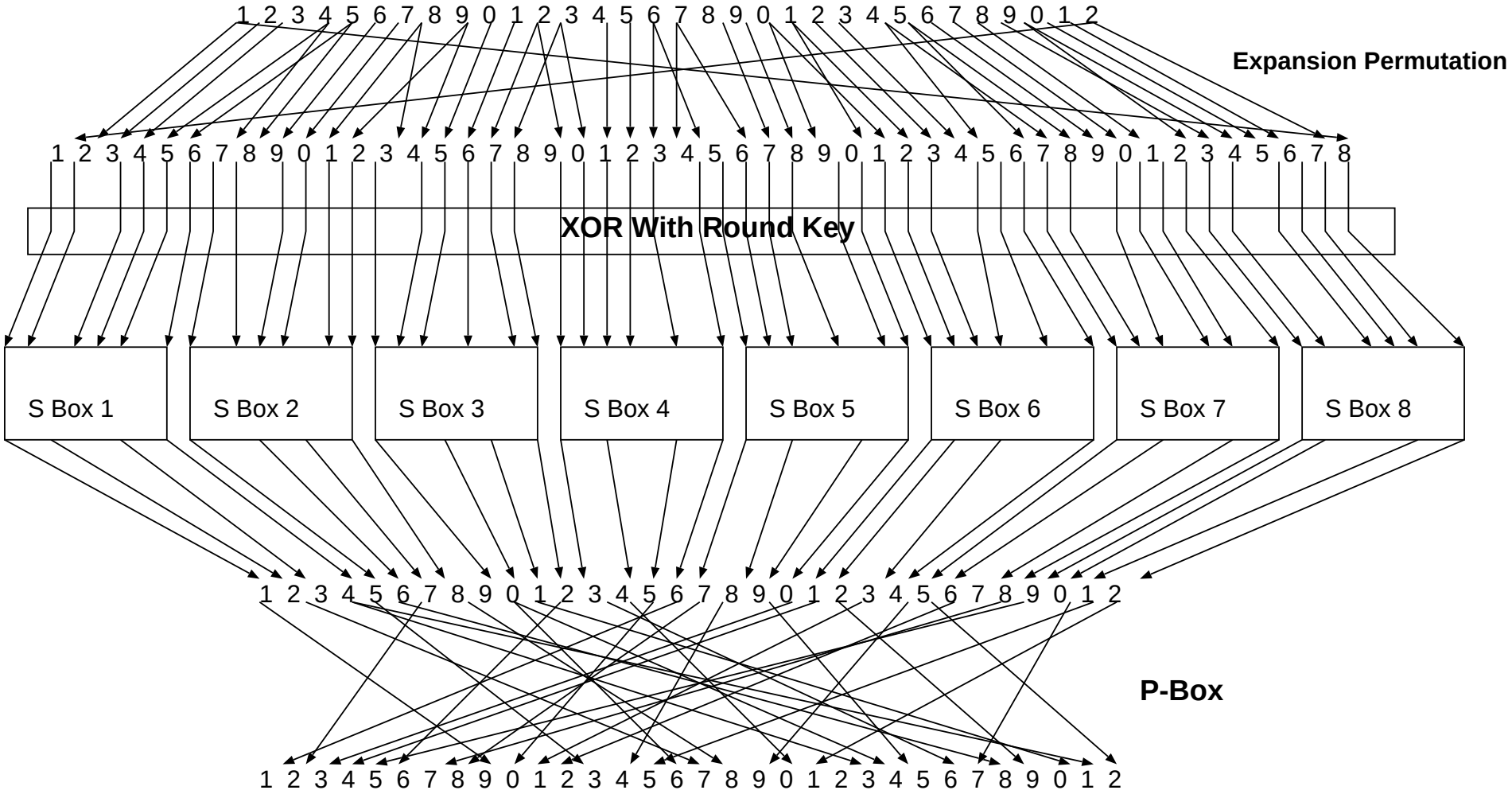
DES Round Continued

4. S-Box
   - Each six bit value is passed into an S-box to produce a four bit result in a non-linear way. (S = Substitution)

5. P-Box
   - 32 bits of output are combined and permuted. (P = Permutation)

6. Feistel Part
   - Output of f is XOR-ed with the left half resulting in new right half.
   - New left half is the old right half.

DES Round Continued



S-Box Operation

S-Boxes represent the non-linear component of DES.

There are eight different S-boxes.

The original S-boxes proposed by IBM were modified by NSA.

Each S-box is a table of 4 rows and 16 columns

- The 6 input bits specify which row and column to use.
- Bits 1 and 6 generate the row.
- Bits 2-5 generate the column.

DES - Decryption

Same algorithm can be used for decryption since we are using the Feistel structure

$$L_{i-1} \oplus f(R_{i-1}, k_i) \oplus f(R_{i-1}, k_i) = L_{i-1}.$$

Remark: for decryption the subkeys are used in the reverse order !

Note that the effect of IP^{-1} is canceled by IP.

Also note that R_{16}, L_{16} is the input of encryption since halves were swapped and that swapping is necessary.

Security of DES

Exhaustive key search (1 or 2 known plaintext/ciphertext pairs).

Number of possibilities for DES is $2^{56} = 7.2 \cdot 10^{16}$.

Software (200 MHz Pentium): 2^{44} encryptions per year.

- 2^{19} keys per second, $2^{16.4}$ seconds per day, $2^{8.5}$ days per year.
- 1 PC: 2000 year, 200 PC's: 10 year, 6 000 PC's: 3 months.

Hardware (FPGA), Cost = 10.000 $.

- Less than 2 year.

Hardware (ASIC), Cost = 250.000 $, 'Deep Crack' (EFF, 1998).

- 1 key in 50 hours, less than 500 $ per key.
- For 1 million $: 1 key in 1/2 hour.

Security of DES Continued

Export from US: 40-bit keys (SSL, Lotus Notes, S/MIME).

- 2 weeks on a PC.
- 2 hours on a LAN.
- 10 minutes on a FPGA.

Moore's 'law':

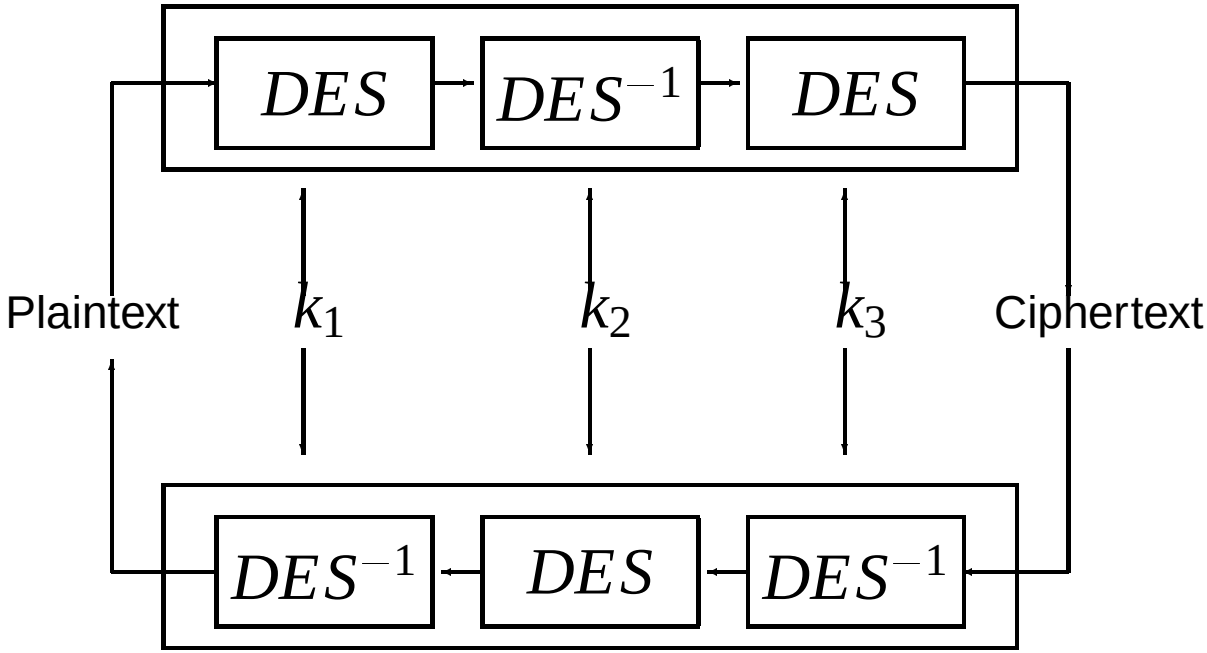
- Computing power doubles every 18 months.
- After 21 years the effective key size is reduced by 14 bits.

Long term: key length and block length of 128 bits.

Triple DES

Invented to get around the problem of a short key.

Involves use of 2 or 3 DES keys.



DES - Weak Keys

The way the subkeys are generated from the original key makes some keys weak:

- The key is split into two halves which are shifted independently.
- If all the bits in one half are the same the corresponding subkeys are identical.

64-bit weak key (including parity bits)	Corresponding 56-bit key
0101 0101 0101 0101	0000000000000000
1F1F 1F1F 1F1F 1F1F	00000000FFFFFFFF
E0E0 E0E0 E0E0 E0E0	FFFFFFFF00000000
FEFE FEFE FEFE FEFE	FFFFFFFFFFFFFFFF

There are a number of other keys which you should not use either, since the encryption key equals the decryption key.

AES - Advanced Encryption Standard

January 1997: NIST call for algorithms to replace DES.

- Block cipher: 128-bit blocks, 128/192/256-bit keys.
- Strength $\approx$ 3-DES, efficiency $\nearrow\nearrow$.
- Documentation, reference C code, optimised C and JAVA code, test vectors.
- Designers give up all intellectual rights.

Open process: public comments, international submissions.

Website: **`http://www.nist.gov/aes/`**

AES - Scope and Significance

- Standard FIPS-197 approved by NIST in November 2001.
- Official scope is limited:
  - US Federal Administration will use AES as Government standard from 26 May 2002.
  - Documents that are ‘sensitive but not classified’.
- Significance is huge: AES is the successor of DES.
- Major factors for quick acceptance:
  - No royalties.
  - High quality.
  - Low resource consumption.

AES - 15 Accepted Submissions

CAST-256	Entrust (CA)
Crypton	Future Systems (KR)
E2	NTT (JP)
Frog	TecApro (CR)
Magenta	Deutsche Telekom (DE)
Mars	IBM (USA)
RC6	RSA (USA)
SAFER+	Cylink (USA)
Twofish	Counterpane (USA)
DEAL	Outerbridge, Knudsen (USA–DK)
DFC	ENS-CNRS (FR)
HPC	Schroeppel (USA)
LOKI97	Brown et al. (AU)
Rijndael	Daemen and Rijmen (BE)
Serpent	Anderson, Biham, Knudsen (UK–IL–DK)

AES - Rijndael

Block + key length: varies between 128 - 256 bits (per 64 bits).

Number of rounds is 10/12/14 depending on block and key length.

Uniform and parallel round transformation, composed of:

- Byte substitution.
- Shift rows.
- Mix columns.
- Round key addition.

Sequential and light-weight key schedule.

No arithmetic operations.

Rijndael - Data Representation

Plaintext block normally is 128 bits or 16 bytes $m_0, \ldots, m_{15}$.

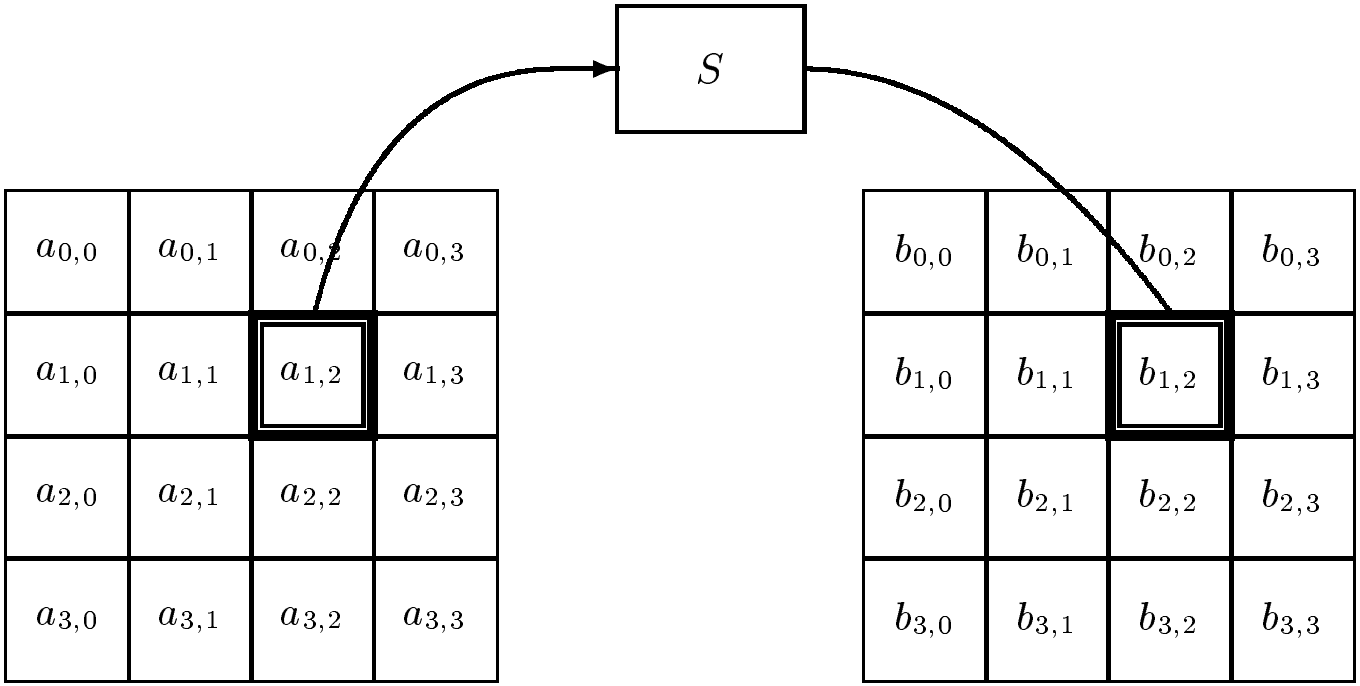
Key normally is 128/192/256 bits or 16/24/32 bytes $k_0, \ldots, k_{31}$.

Both are represented as rectangular array of bytes.

m_0	m_4	m_8	m_{12}
m_1	m_5	m_9	m_{13}
m_2	m_6	m_{10}	m_{14}
m_3	m_7	m_{11}	m_{15}

k_0	k_4	k_8	k_{12}	k_{16}	k_{20}
k_1	k_5	k_9	k_{13}	k_{17}	k_{21}
k_2	k_6	k_{10}	k_{14}	k_{18}	k_{22}
k_3	k_7	k_{11}	k_{15}	k_{19}	k_{23}

Rijndael - Byte Substitution

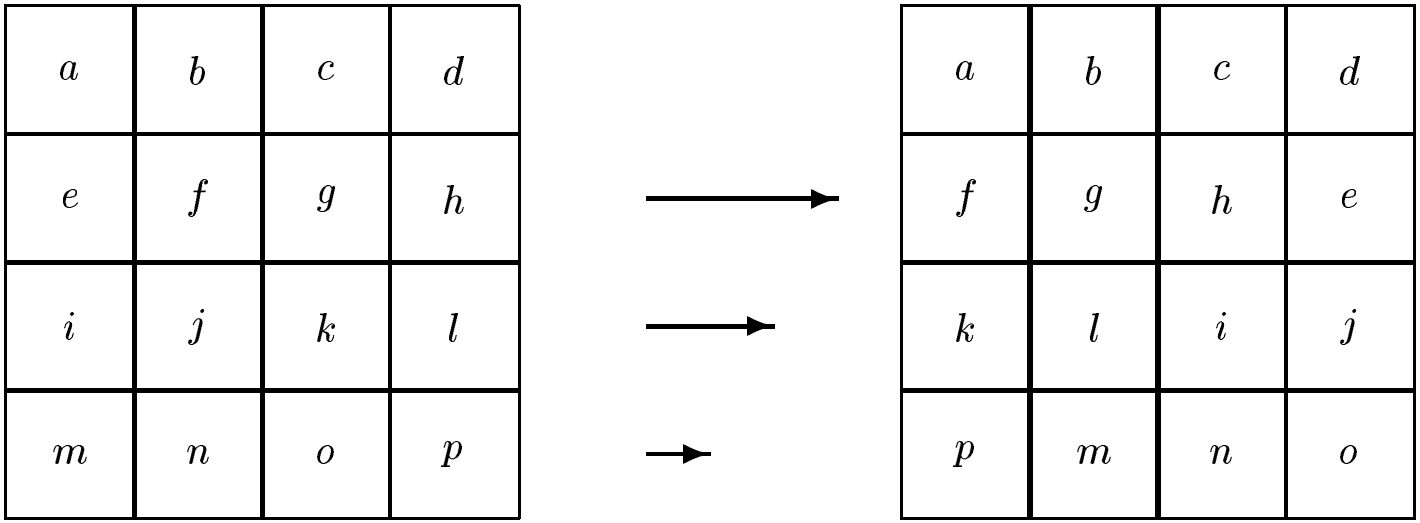


State matrix is transformed byte by byte.

S-box is invertible, else decryption would not work.

Only 1 S-box for the whole cipher (simplicity).

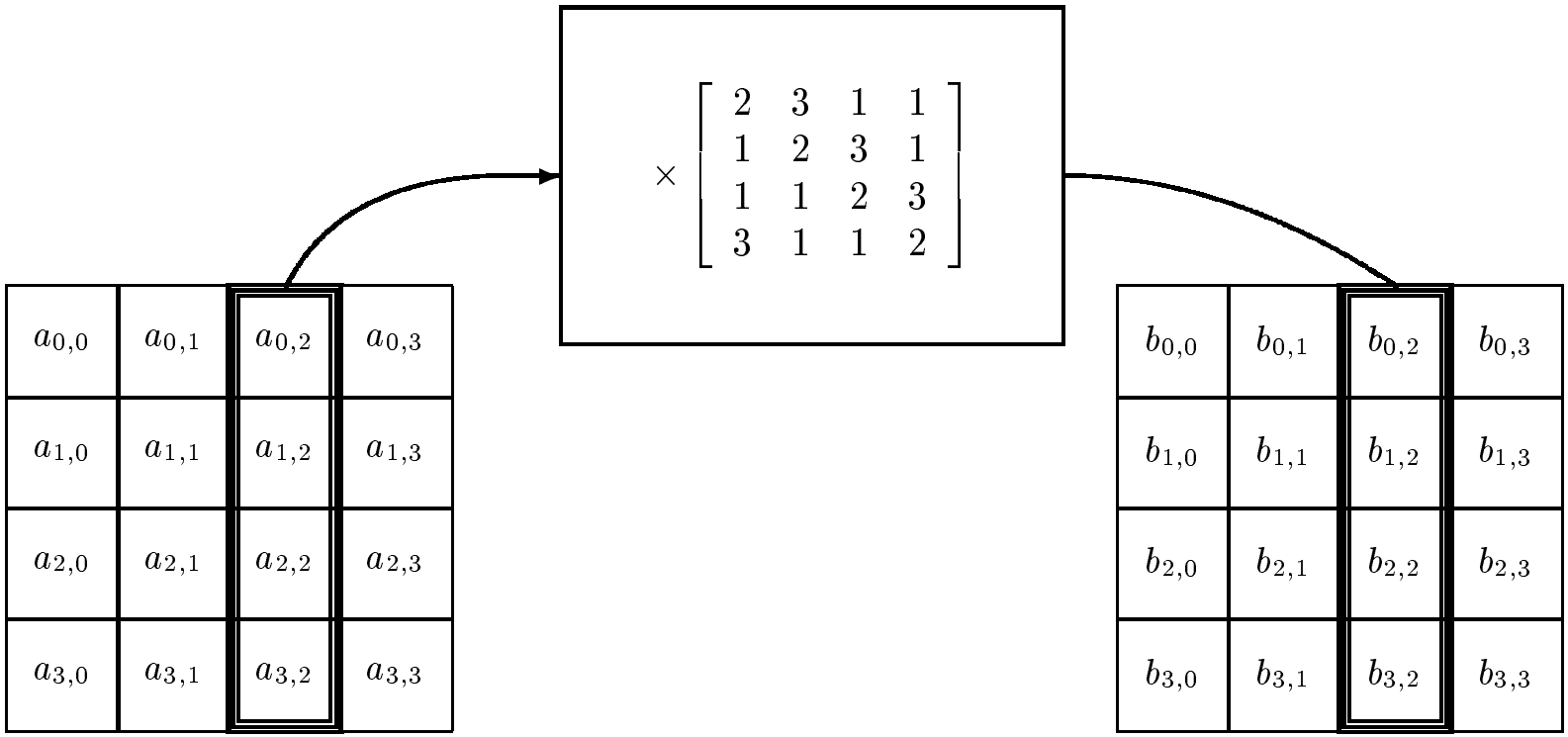
Rijndael - Shift Rows



Rows shifted over different offsets (depending on block length).

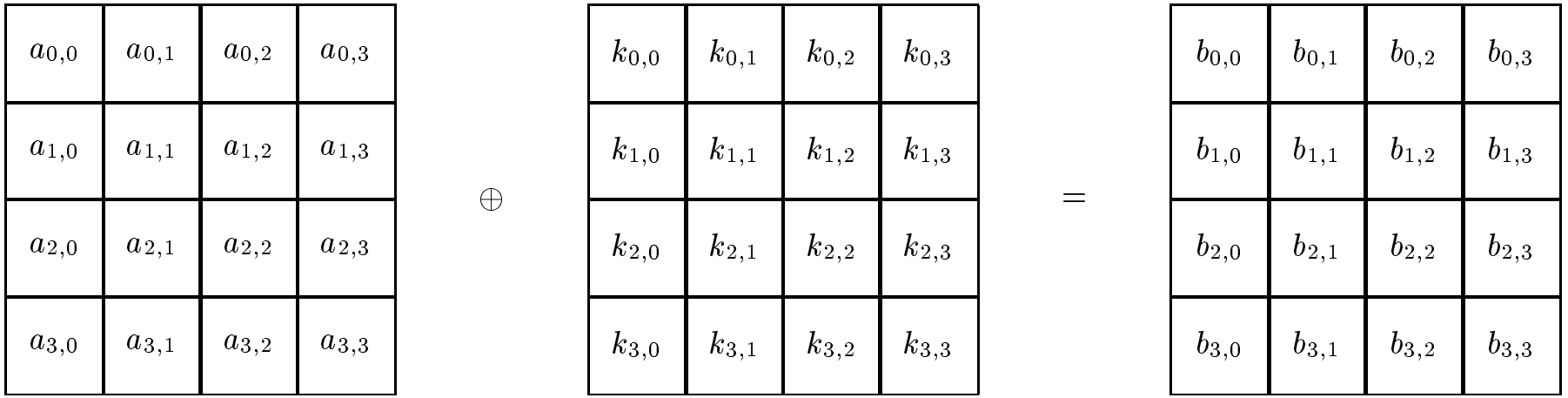
Purpose: diffusion over the columns.

Rijndael - Mix Columns



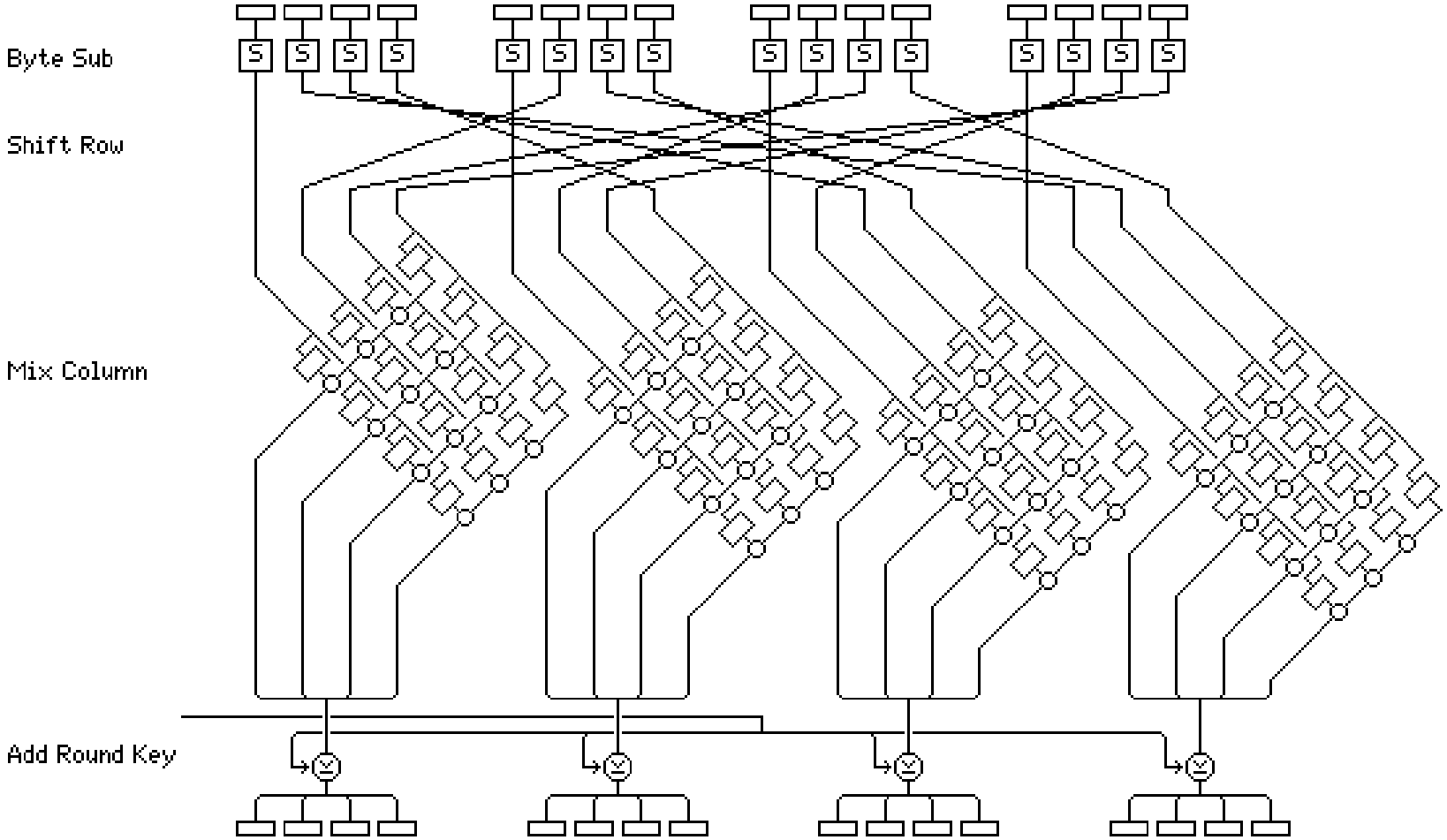
- Operations over finite field $GF(2^8)$.
- Bytes in columns are combined linearly.
- Good diffusion properties over rows.
- Based on maximal distance codes.

Rijndael - Round Key Addition



Round key is simply XOR-ed with state matrix.

Rijndael - Round function



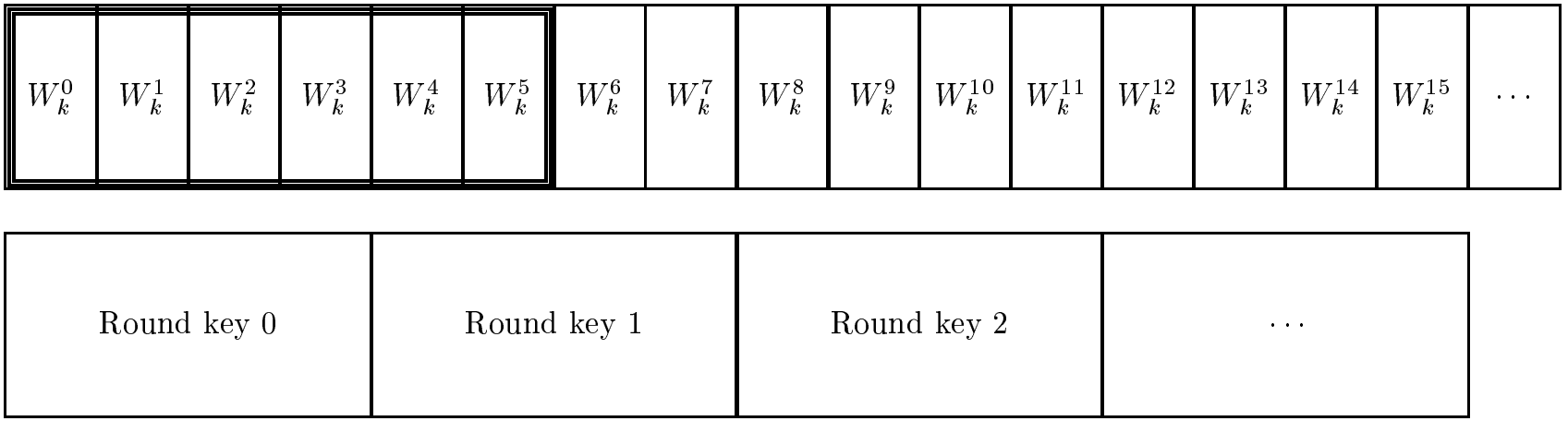
Rijndael - Pseudo-code

Rijndael with 10 rounds is described by the following code:

```
AddRoundKey(S,K[0]);
for (i = 1; i <= 9; i++)
  {
     SubBytes(S);
     ShiftRows(S);
     MixColumns(S);
     AddRoundKey(S,K[i]);
  }
SubBytes(S);
ShiftRows(S);
AddRoundKey(S,K[10]);
```

Rijndael - Key Schedule

Example: key of 192 bits or 6 words of 32 bits.



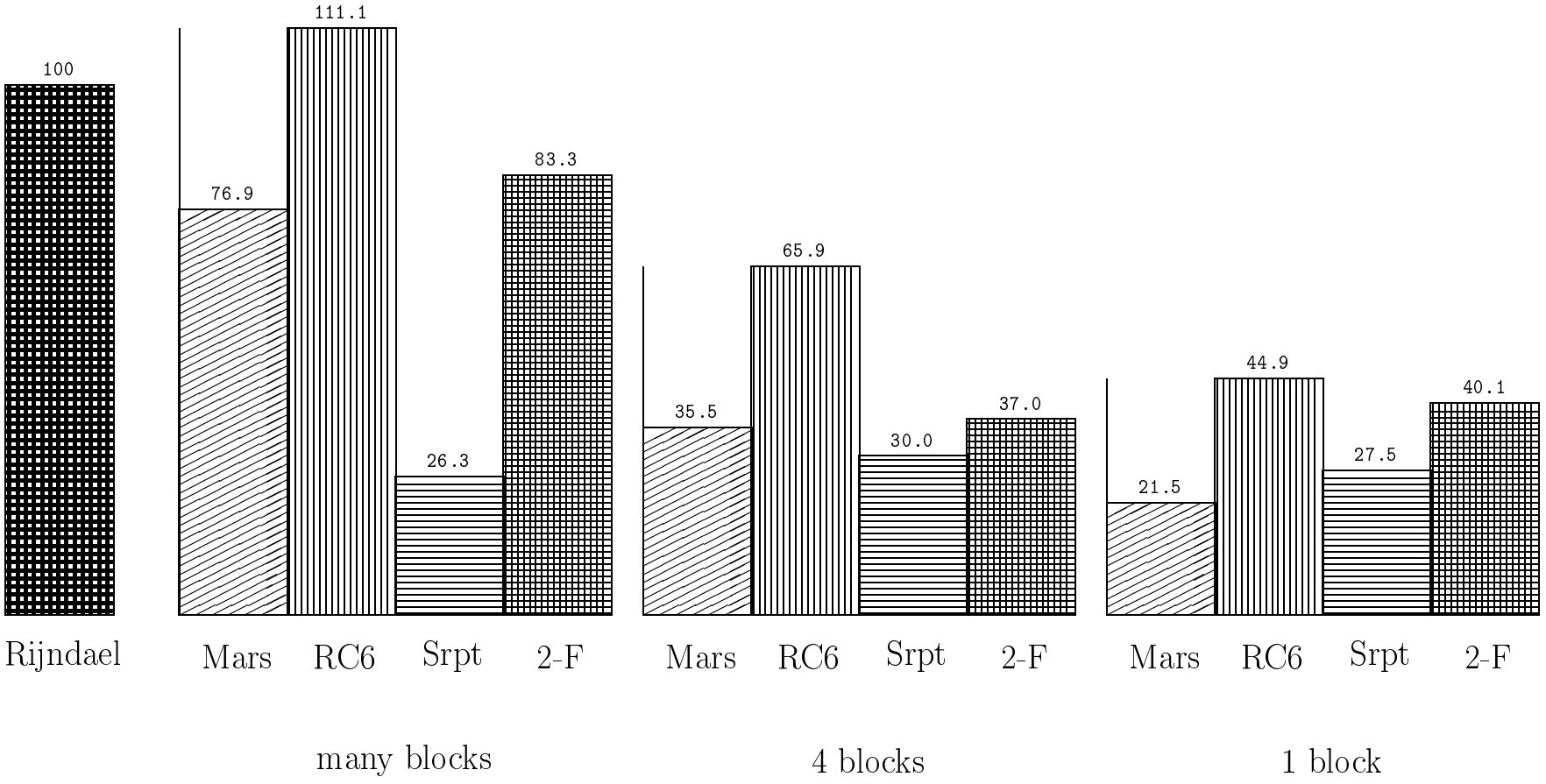
$$W_k^{6n} = W_k^{6n-6} \oplus f(W_k^{6n-1})$$
$$W_k^i = W_k^{i-6} \oplus W_k^{i-1}$$

Cipher key expansion can be done just-in-time.

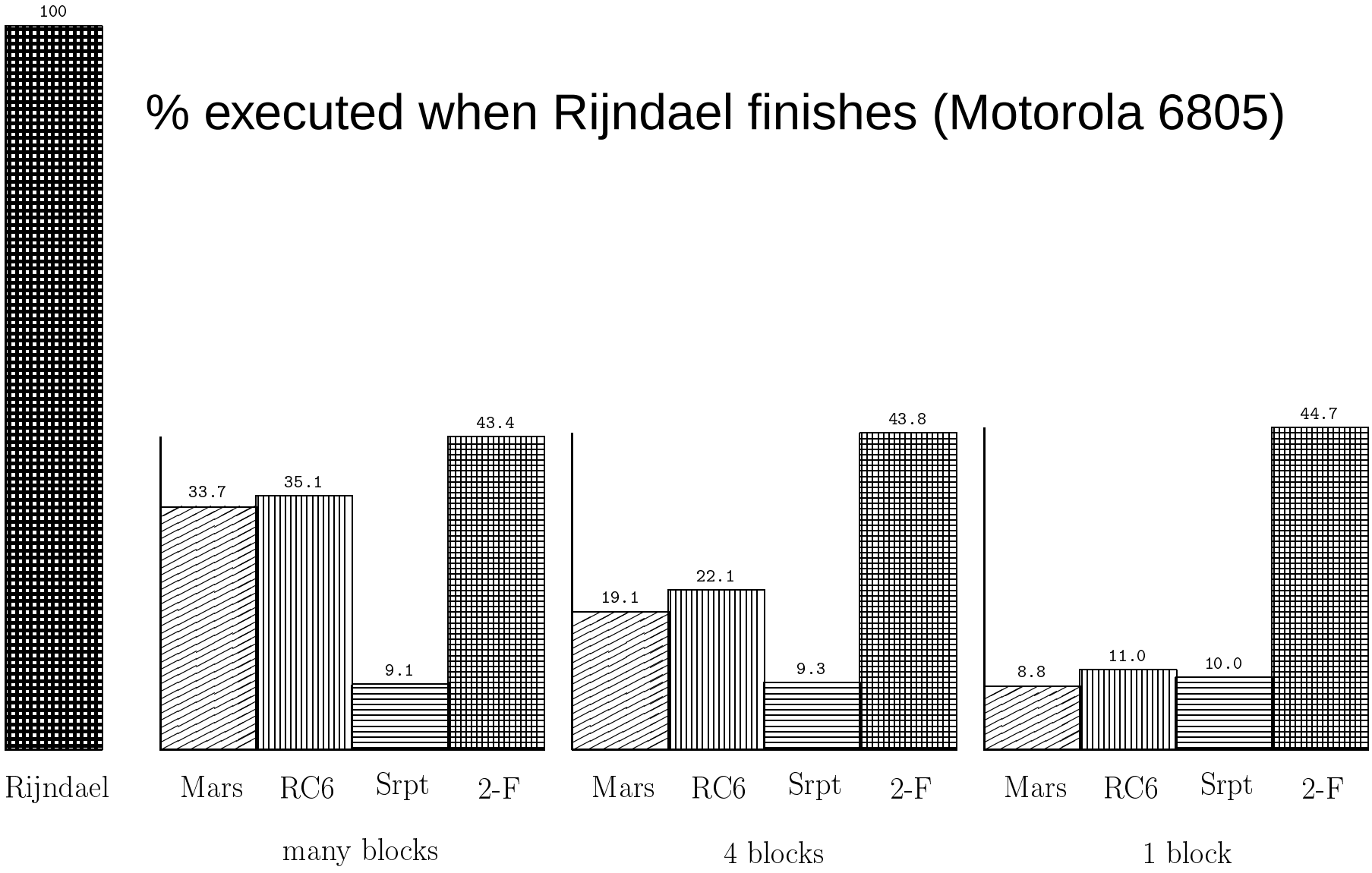
No extra storage required.

Rijndael - Performance PC

% executed when Rijndael finishes (Pentium Pro II)



Rijndael - Performance Smartcard



Modes of Operation

If message is longer than blocksize, block cipher can be used in a variety of ways to encrypt the plaintext.

Soon after DES was made a US Federal standard, another US standard appeared giving four recommended ways of using DES for data encryption.

These modes of operation have since been standardised internationally and can be used with any block cipher.

ECB Mode

ECB = Electronic Code Book

Simplest approach to use a block cipher.

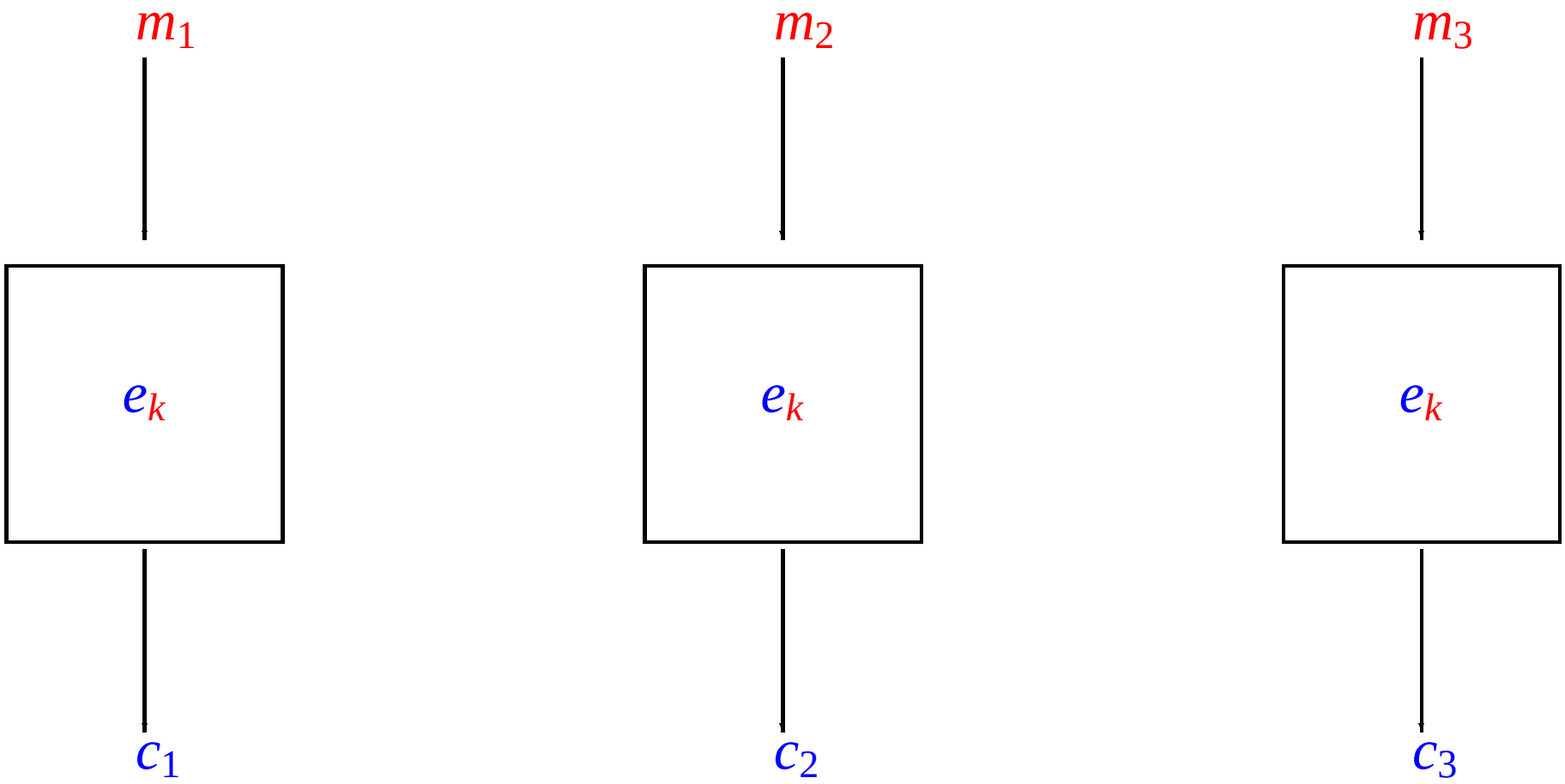
Plaintext m is divided into t blocks of n bits $m_1, m_2, \ldots, m_t$.
The last block is padded if necessary.

Ciphertext blocks $c_1, \ldots, c_t$ are defined as follows

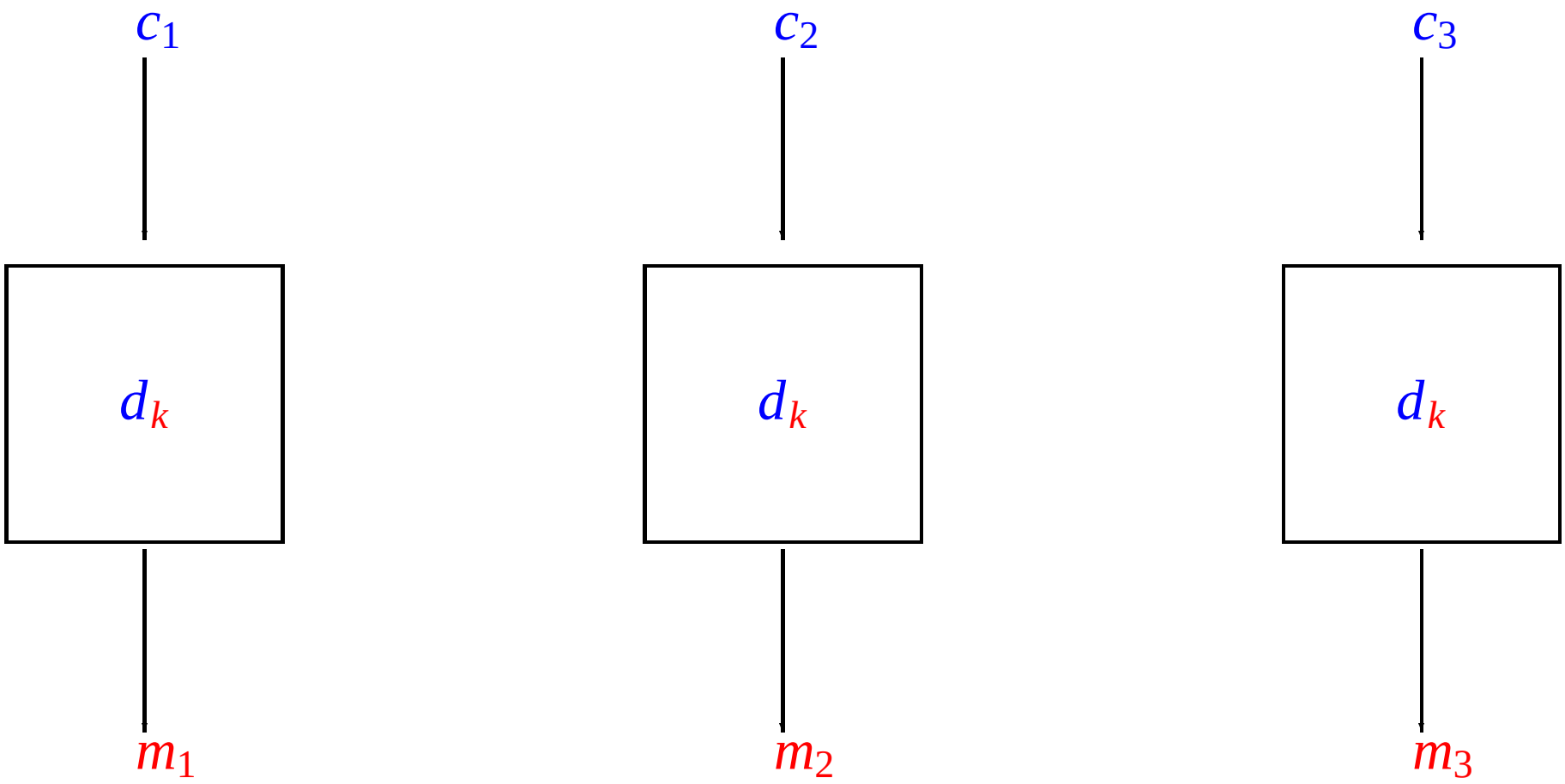
$$c_i = e_k(m_i).$$

Note that if $m_i = m_j$ then we have $c_i = c_j$, thus patterns in plaintext reappear in ciphertext.

ECB Encipherment



ECB Decipherment



ECB Mode - Properties

- Blocks are independent.
- Does not hide patterns or repetitions.
- Error propagation: expansion within one block.
- Reordering of blocks is possible without too much distortion.
- Stereotyped beginning/ends of messages are common.
- Susceptible to block replay.

Block replay

- Extracting ciphertext corresponding to a known piece of plain text.
- Amending other transactions to contain this known block of text.

Countermeasures against block replay

- Extra checksums over a number of plaintext blocks.
- Chaining the cipher, this adds context to a block.

CBC Mode

CBC = Cipher Block Chaining

Plaintext m is divided into t blocks of n bits $m_1, m_2, \ldots, m_t$.
The last block is padded if necessary.

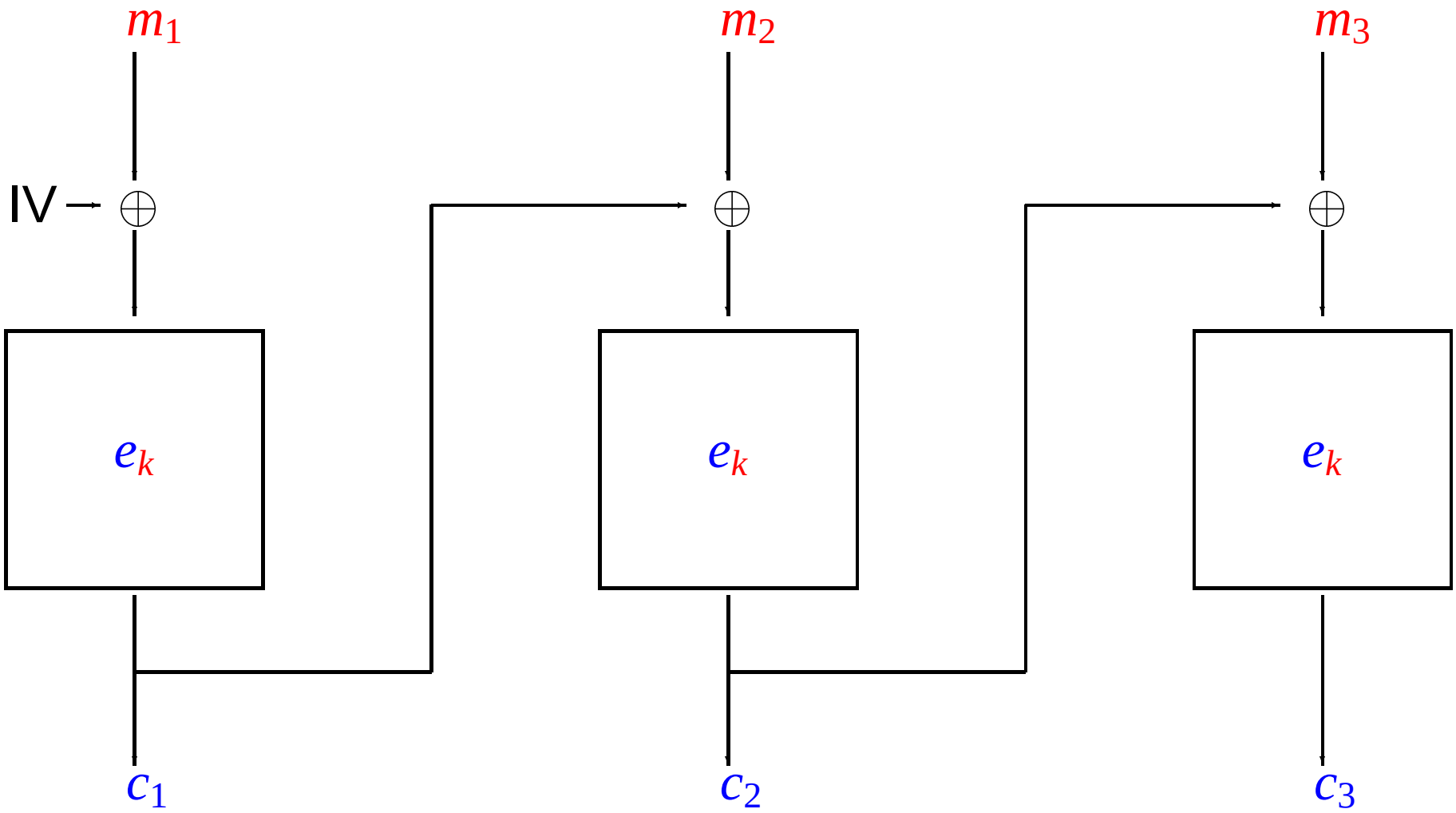
Encryption

- $c_1 = e_k(m_1 \oplus IV)$.
- $c_i = e_k(m_i \oplus c_{i-1})$ for $i > 1$.

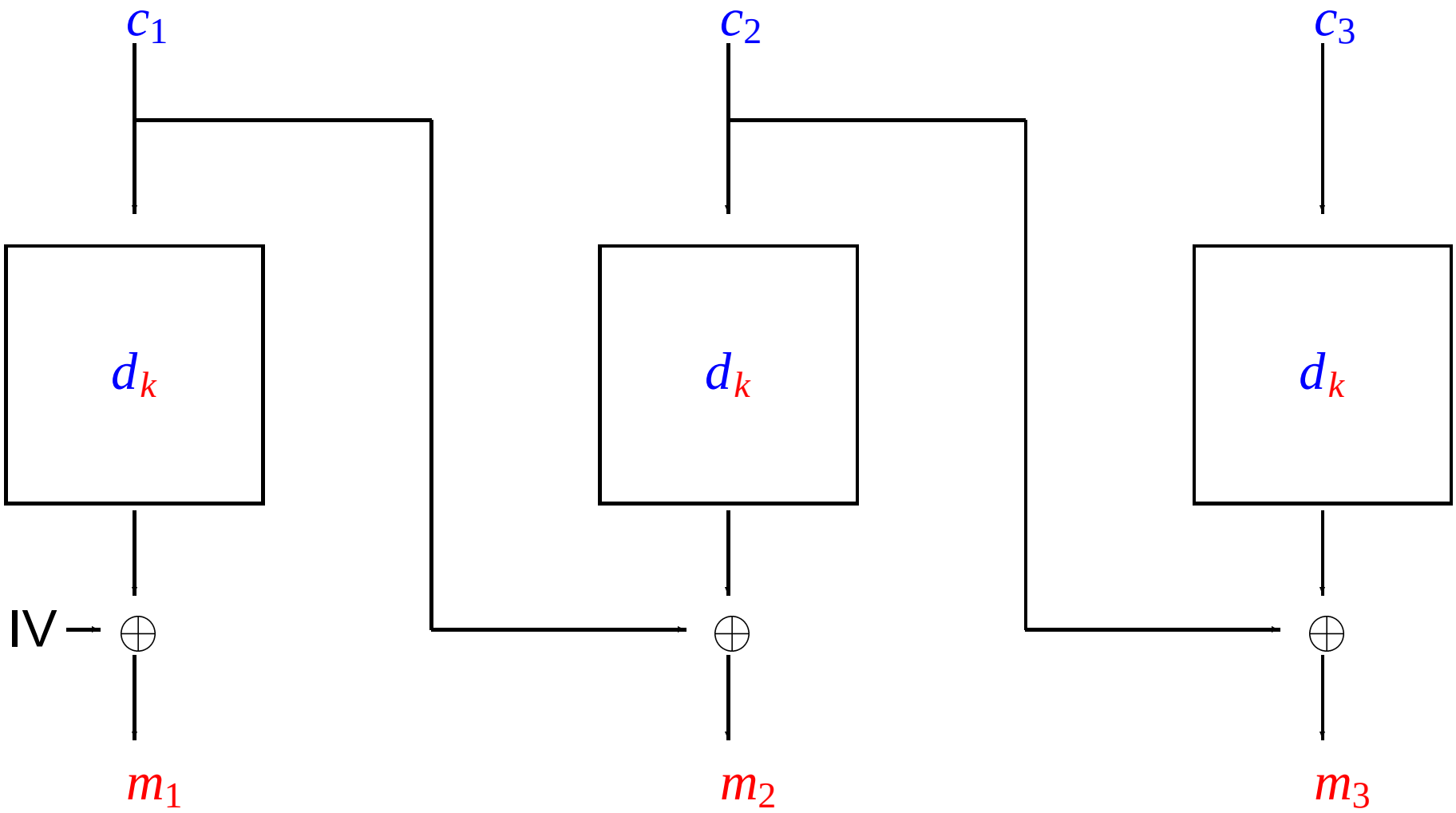
Decryption

- $m_1 = d_k(c_1) \oplus IV$.
- $m_i = d_k(c_i) \oplus c_{i-1}$ for $i > 1$.

CBC Encipherment



CBC Decipherment



CBC Mode - Properties

- Ciphertext depends on all previous plaintext blocks (internal memory).
- Different IV hides patterns and repetitions.
- Error propagation: expansion in one block, copied in next block.
- Decryption of one block requires only ciphertext of previous block, so CBC is self-synchronizing.
- Rearanging order of blocks affects decryption (not if previous ciphertext block is correct).
- Default mode to use.

ECB and CBC - Padding

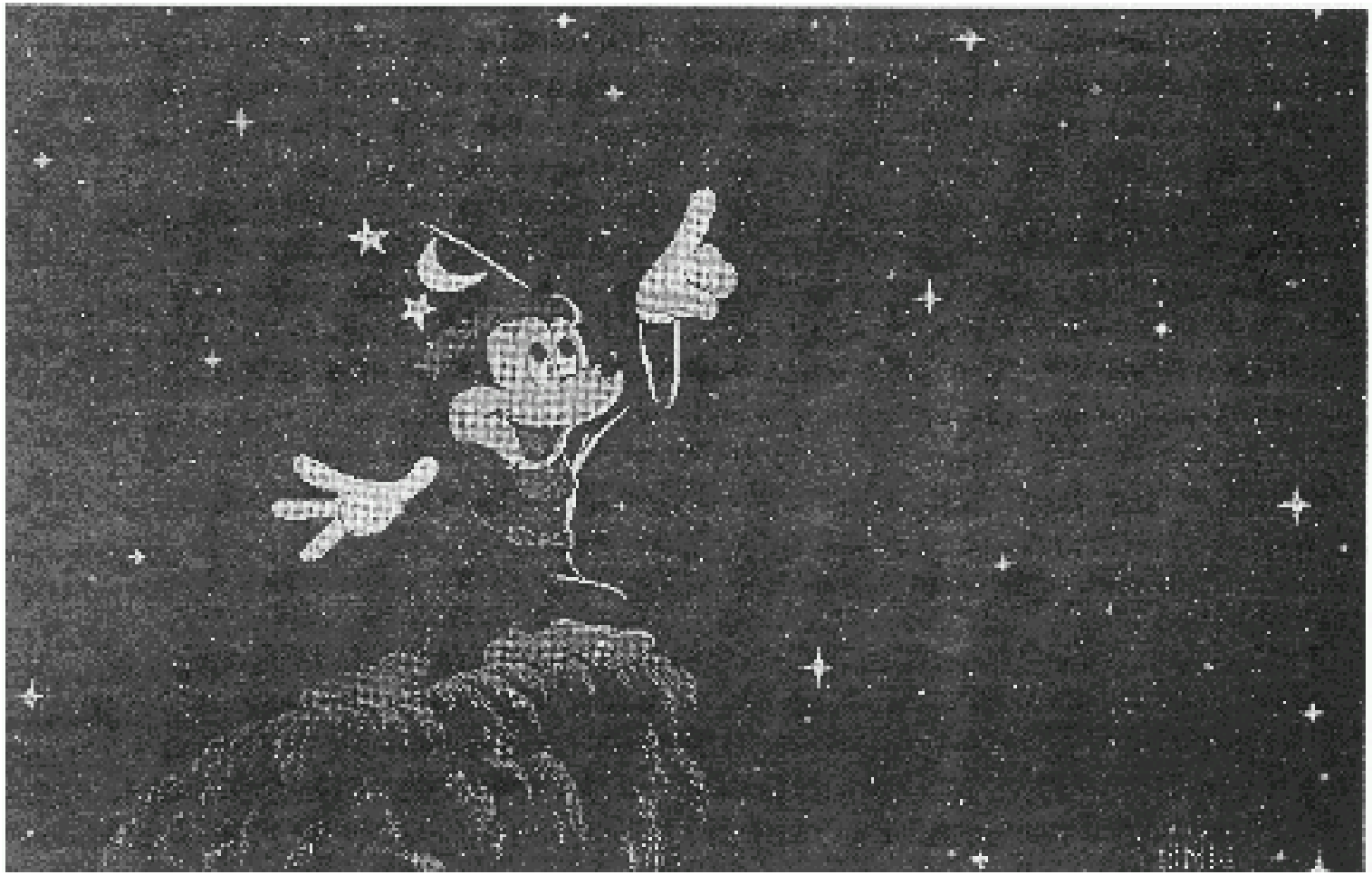
Problem: suppose length of plaintext is not multiple of block length.

$\Rightarrow$ Last block m_t only contains $l < n$ bits.

Padding schemes:

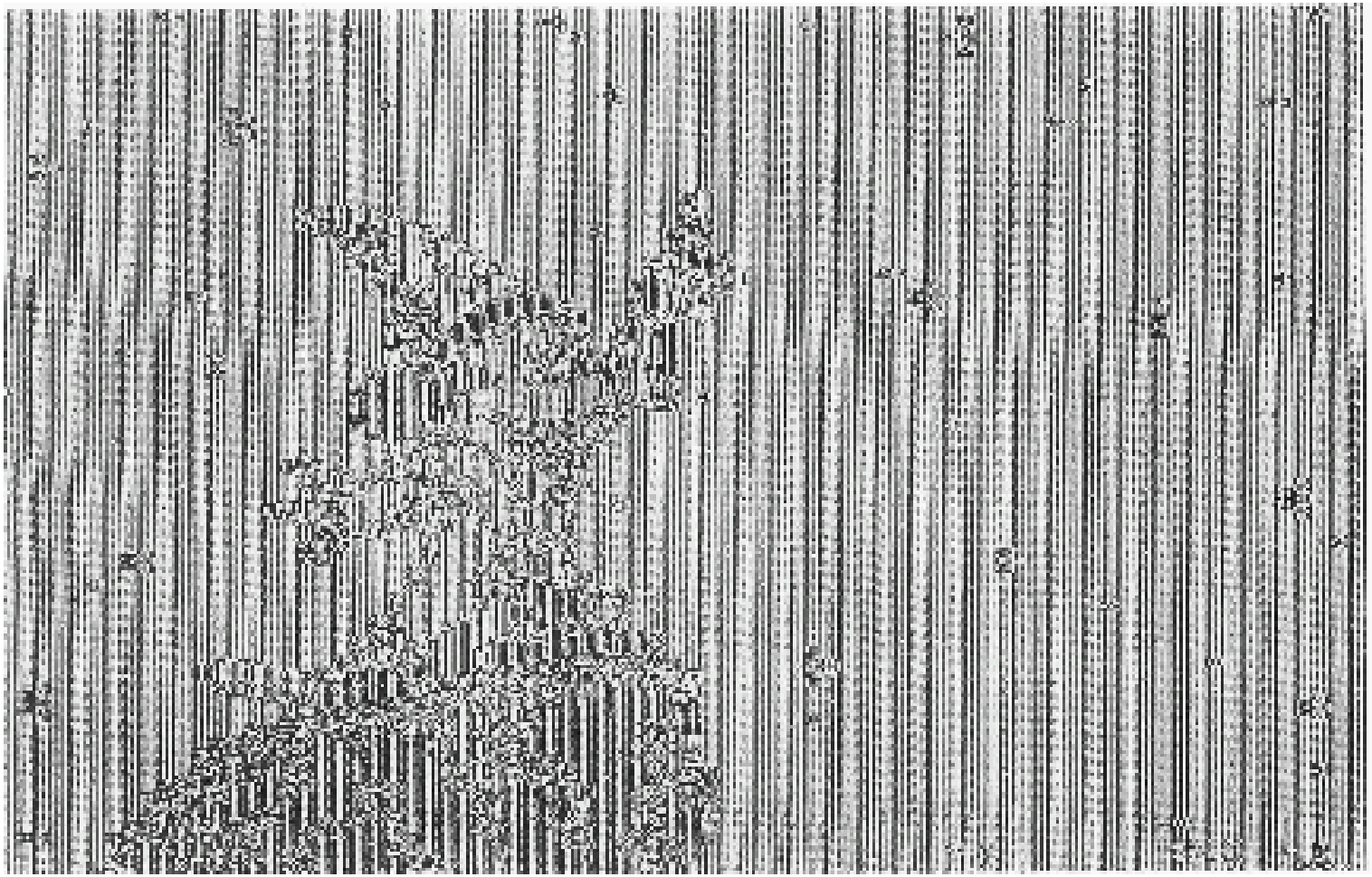
- Append $n - l$ zeros to last block. Problem is that trailing 0-bits of plaintext cannot be distinguished from padding.
- Append 1 and $n - l - 1$ 0-bits. If $l = n$, then create extra block starting with 1 and remaining $n - 1$ bits 0.
- Append $n - l$ bits of ciphertext c_{t-1}.

ECB and CBC - Example



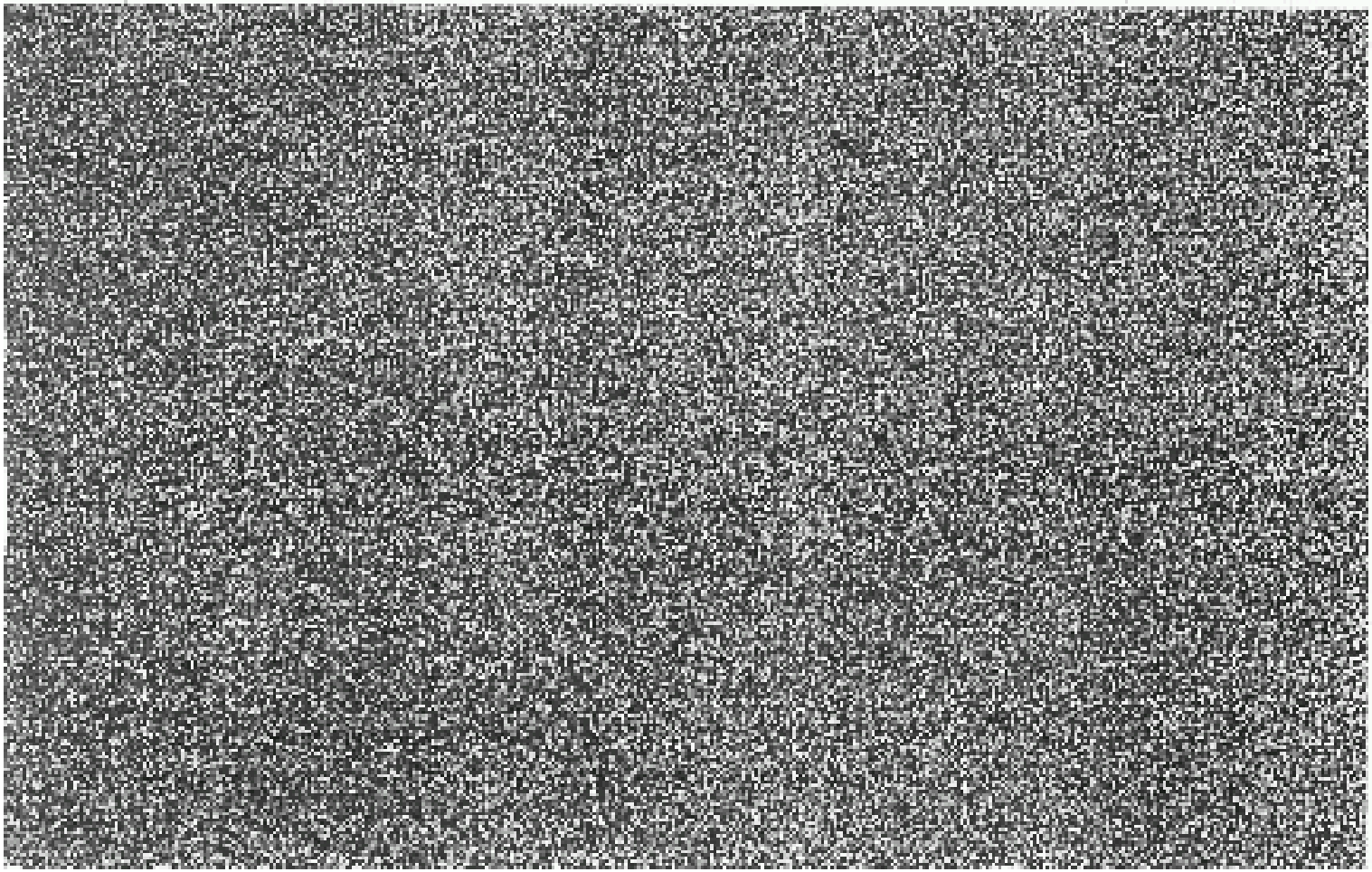
Plaintext : original picture

ECB and CBC - Example



Ciphertext: ECB Encryption

ECB and CBC - Example



Ciphertext: CBC Encryption

OFB Mode

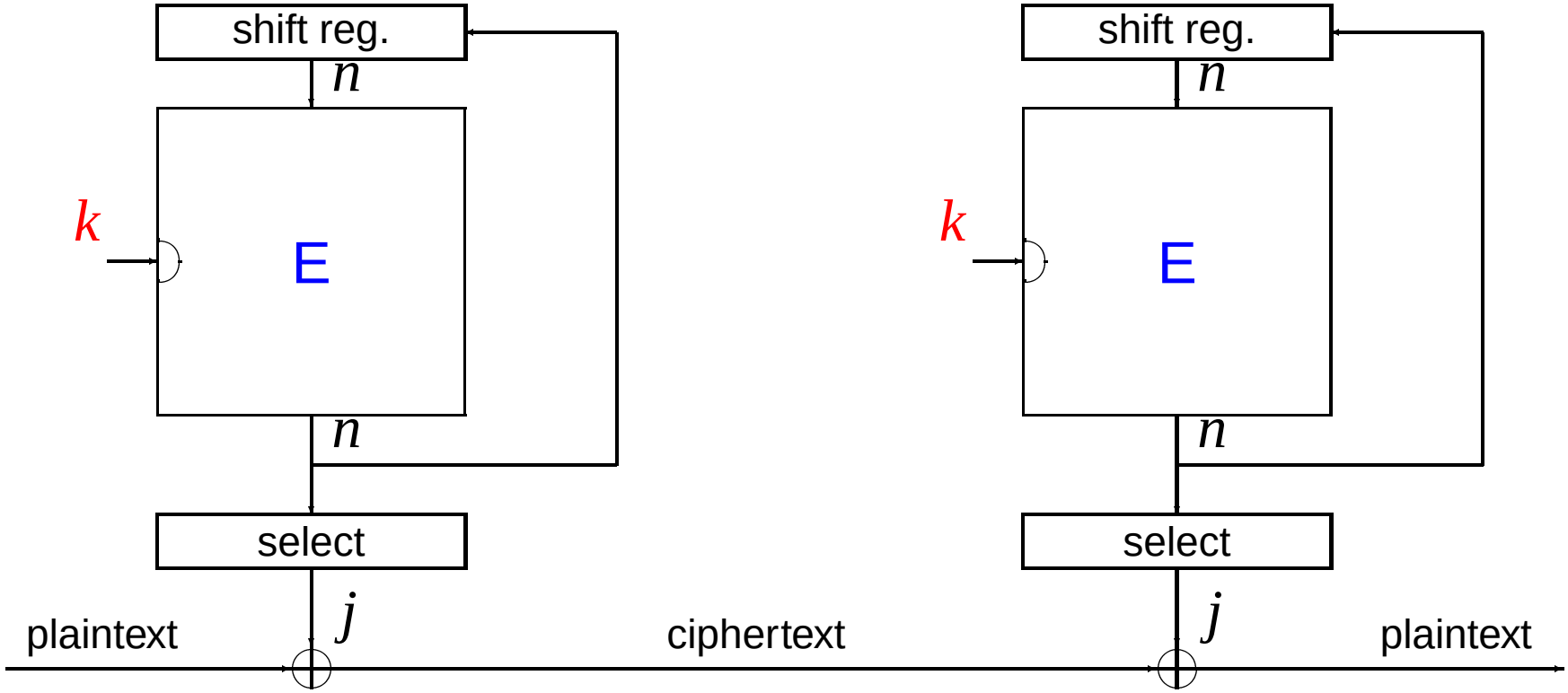
OFB = Output FeedBack

This mode enables a block cipher to be used as a stream cipher

$\Rightarrow$ The block cipher creates the keystream.

- Block length of block cipher is n.
- One can choose to use keystream blocks of $j \leq n$ bits only.
- Divide plaintext into a series of j-bit blocks $m_1, \ldots, m_t$.
- Encryption: $c_i = m_i \oplus e_i$, where e_i selection of j bits of 'ciphertext' generated by block cipher, starting with IV in shift register.

OFB Mode


shift reg.
n
k
E
n
select
j
plaintext
ciphertext
shift reg.
n
k
E
n
select
j
plaintext

OFB Mode - Properties

- Synchronous stream cipher.
- No linking between subsequent blocks.
- Different IV necessary; otherwise insecure.
- Only uses encryption (no decryption algorithm necessary).
- If $j < n$: more effort per bit.
- Key stream independent of plaintext: can be precomputed.
- No error propagation: errors are only copied.

CFB Mode

CFB = Cipher FeedBack

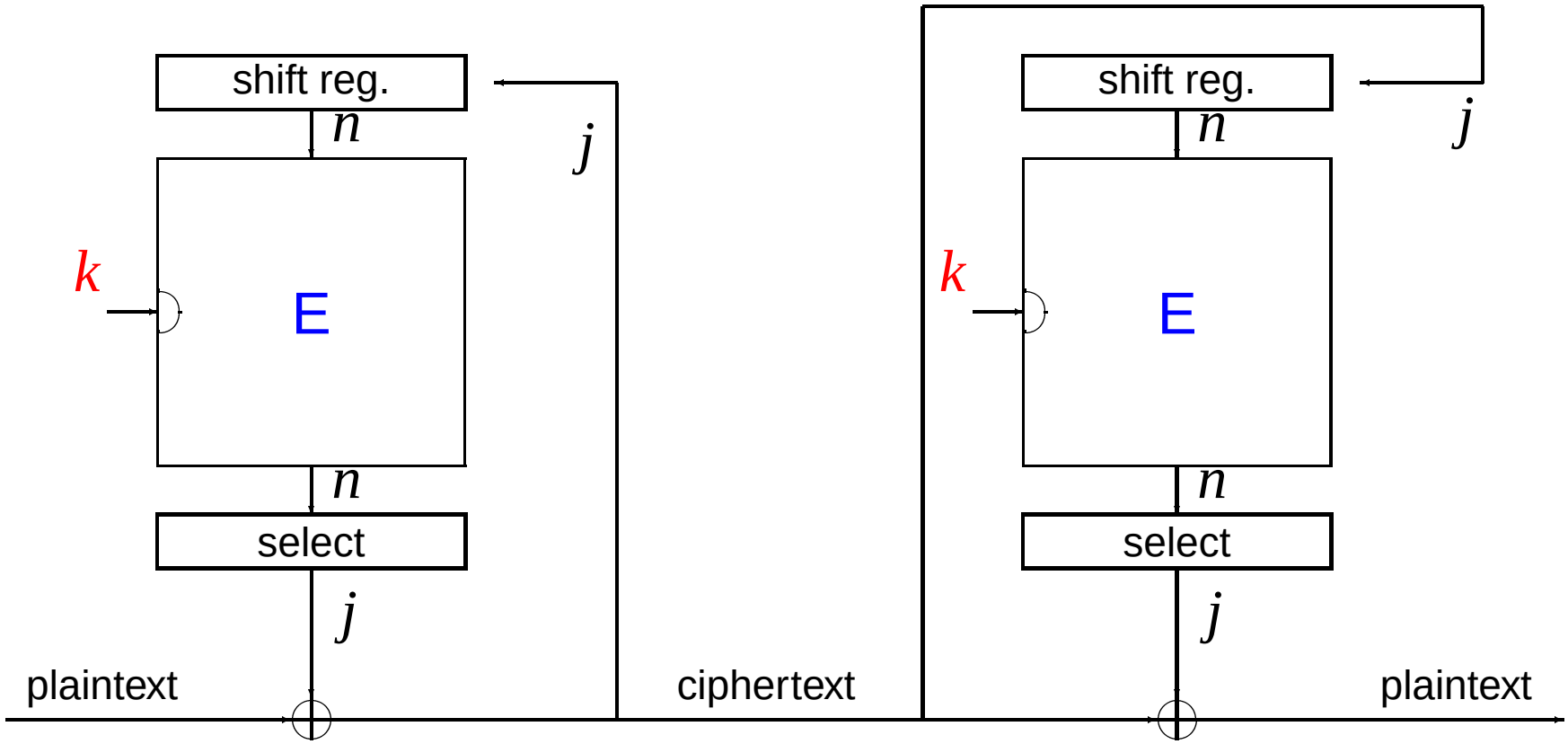
In OFB Mode the keystream is generated by

- Encrypting the IV.
- Encrypting the output from this encryption.

In CFB Mode the keystream is generated by

- Encrypting the IV.
- Encrypting n bits of ciphertext.

CFB Mode


shift reg.
n
j
k
E
n
select
j
plaintext
ciphertext
shift reg.
n
j
k
E
n
select
j
plaintext

CFB Mode - Properties

- Self-synchronizing stream cipher.
- Ciphertext depends on all previous plaintext blocks (internal memory).
- Different IV hides patterns and repetitions.
- Only uses encryption (no decryption algorithm necessary).
- If $j < n$: more effort per bit.
- Error propagation: propagates over $\lceil n/j \rceil + 1$ blocks.
- No synchronization needed between sender and receiver, synchronization if n bits have been received correctly.
- Often used with $j = 1, 8$ because of synchronization.

Mode Summary

	ECB	CBC	OFB	CFB
Patterns	–	+	+	+
Repetitions	–	IV	IV	IV
Block length	n	n	j	j
Error prop.	1 block	1 block + 1 bit	1 bit	n bits + 1 bit
Synch.	block	block	exact	j bits
Application	key enc.	default	no error prop.	synch.

Block or Stream Cipher ?

Which is best ?

- Block ciphers:
  - More general, can be used as stream cipher.
  - Standardisation: DES and AES + modes of operation.
  - Easier to implement in software (no bit manipulations).
- Stream ciphers:
  - Stream ciphers easier to do the maths.
  - Either makes them easier to break or easier to study.
  - Very fast in hardware (less flexible).
  - Not very suitable for software since works 1 bit at a time.
  - Better regarding synchronization and error propagation.

Data Integrity

Encryption provides confidentiality.

Encryption does not necessarily provide integrity of data !

Counterexamples:

- Changing order in ECB mode.
- Encryption of a compressed file, i.e. without redundancy.
- Encryption of a random key.

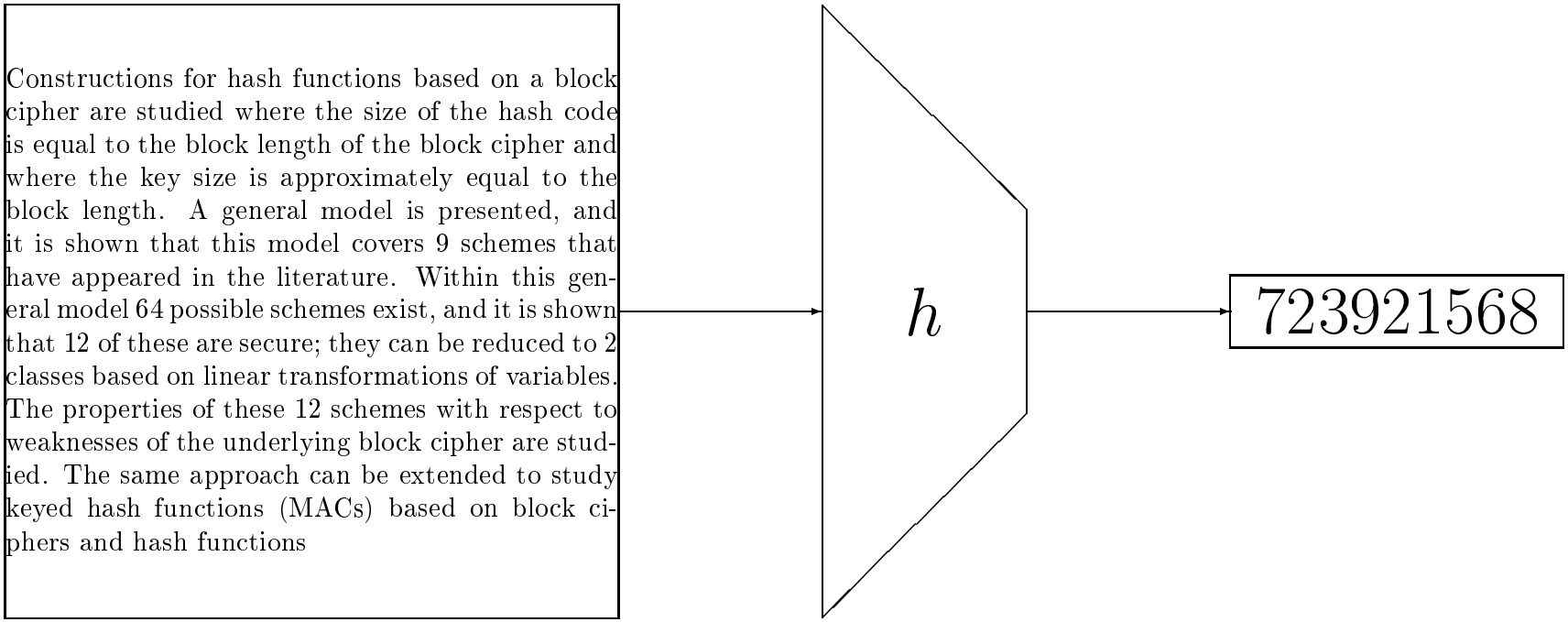
Use cryptographic function to get a check-value and send it with data.

Two types:

- Manipulation Detection Codes (MDC).
- Message Authentication Codes (MAC).

Hash Function Model

Hash function = efficient function mapping binary strings of arbitrary length to binary strings of fixed length, called the hash-value.



Manipulation Detection Code (MDC)

MDC: hash function without key.

- Compression to fixed length.
- Preimage resistant: given $c = h(m)$, hard to find m' with $h(m') = c$.
- 2nd preimage resistant: given m and $h(m)$, hard to find $m' \neq m$ with $h(m) = h(m')$.
- Collision resistant: hard to find m, $m' \neq m$ with $h(m) = h(m')$.

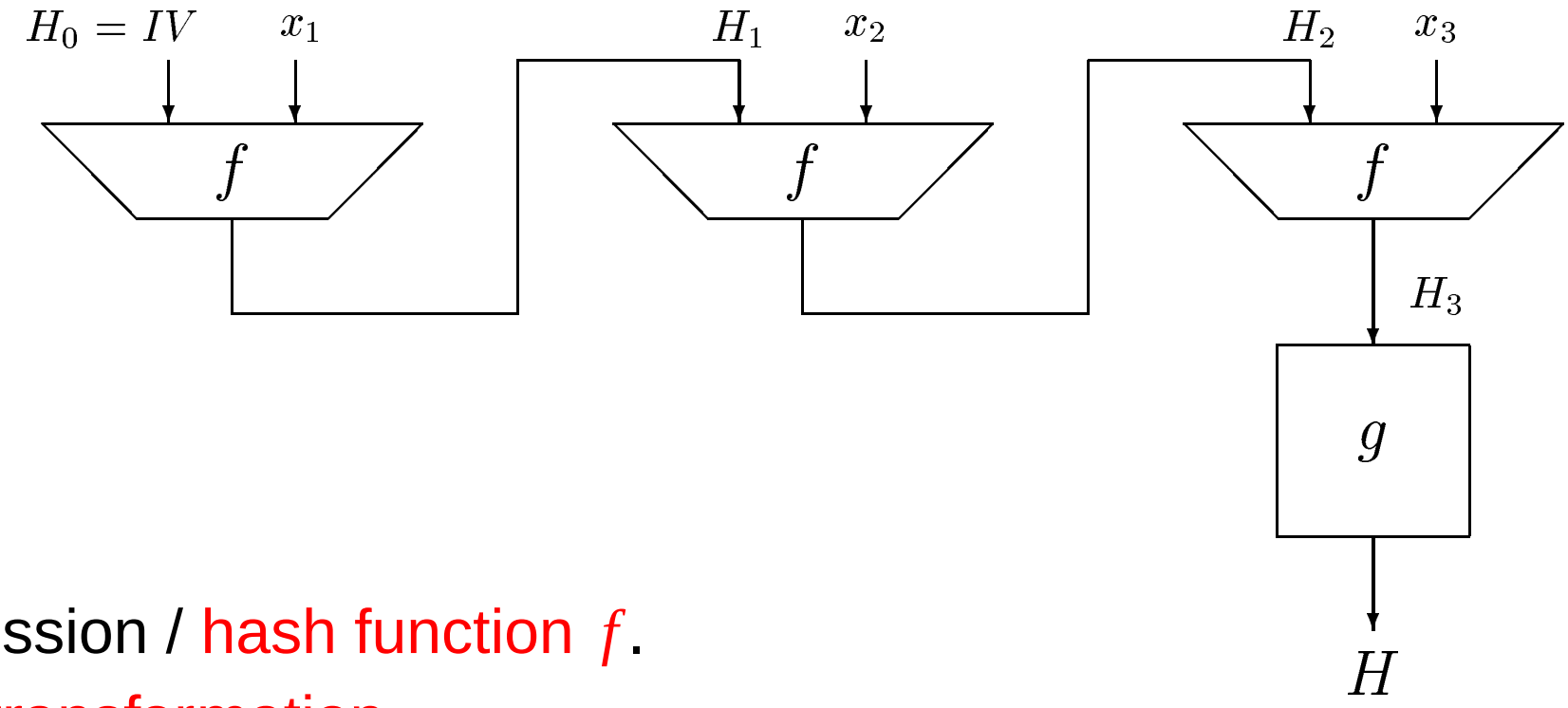
The MDC is concatenated with the data and then the combination is encrypted/signed (to stop tampering with the MDC).

Two types of MDC:

- MDCs based on block ciphers (12 variants).
- Customized hash functions.

Construction of MDCs

Most MDC are constructed as iterated hash function.



Compression / hash function f.

Output transformation g.

Unambiguous padding needed if length is not multiple of block length.

Practical Hash Functions

ISO/IEC 10118-3 contains:

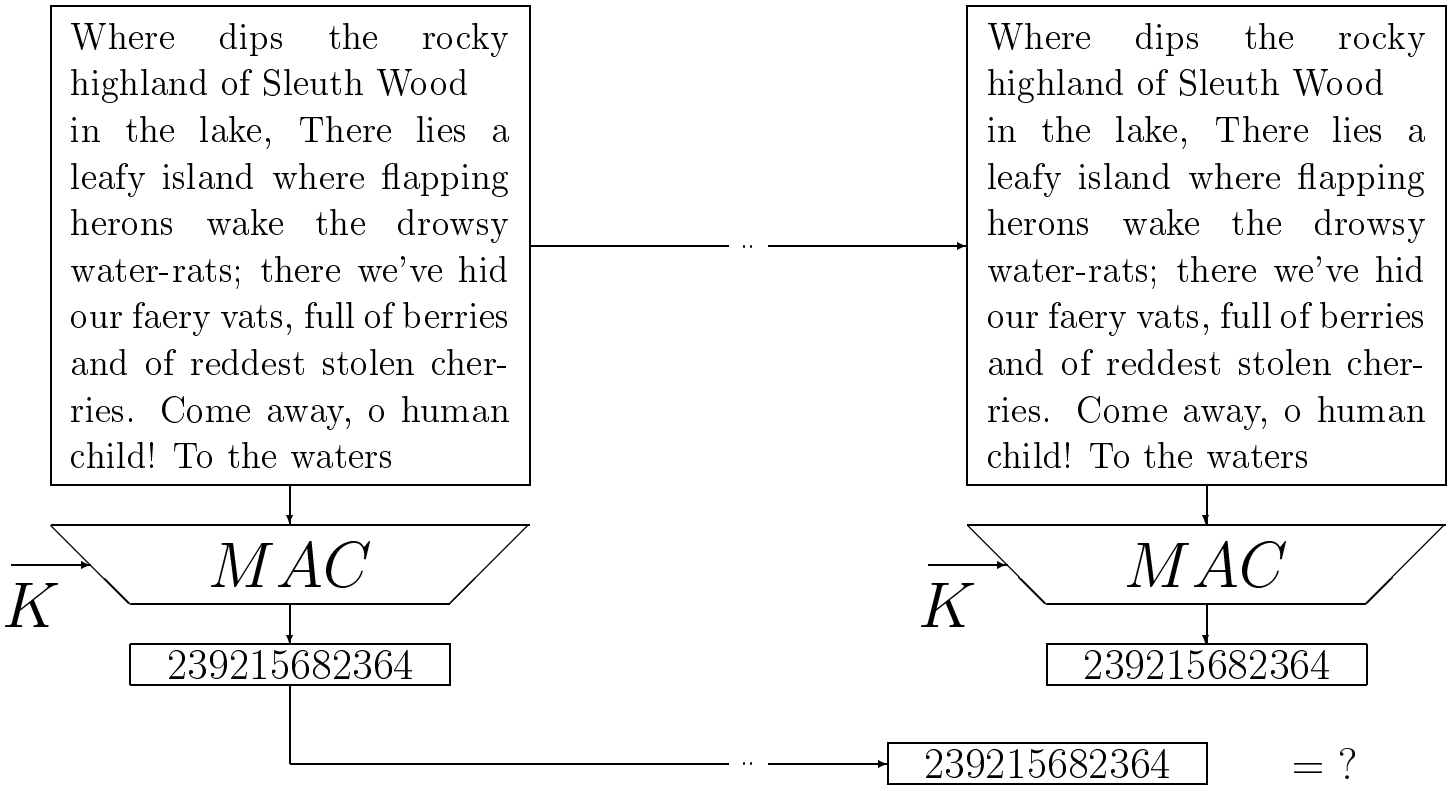
- SHA-1: US government standard.
- RIPEMD-160: European design (K.U.Leuven and BSI).

Properties:

- Processes data in blocks of 512 bits.
- 160-bit result.
- Uses 32-bit arithmetic (exor, and, or, add, rotate).
- 30 MB/s on a 850 Mhz PC (similar to Rijndael).

Message Authentication Code (MAC)

MAC = hash function with secret key.



Message Authentication Code (MAC)

Message authentication code = hash function with secret key.

$MAC = h_k(m)$, where

- h is the hash function.
- k is the secret key.
- m is the message.

Transmit $m||MAC$, where $||$ denotes concatenation of data items.

Description of hash function is public.

Maps string of abitrary length to string of fixed length (32-160 bits).

Computing $h_k(m)$ easy given m and k.

Computing $h_k(m)$ given m, but not k should be very difficult, even if a large number of pairs $\{m_i, h_k(m_i)\}$ are known.

MAC Mechanisms

There are various types of MAC schemes

- MACs based on block ciphers in CBC mode.
- MACs based on MDCs.
- Customized MACs.

Best known and most widely used by far are the CBC-MACs.

CBC-MACs are the subject of various international standards:

- US Banking standards ANSI X9.9, ANSI X9.19.
- Specify CBC-MACs, date back to early 1980s.
- The ISO version is ISO 8731-1: 1987.

Above standards specify DES in CBC mode to produce a MAC.

CBC-MAC - Overview of Operation

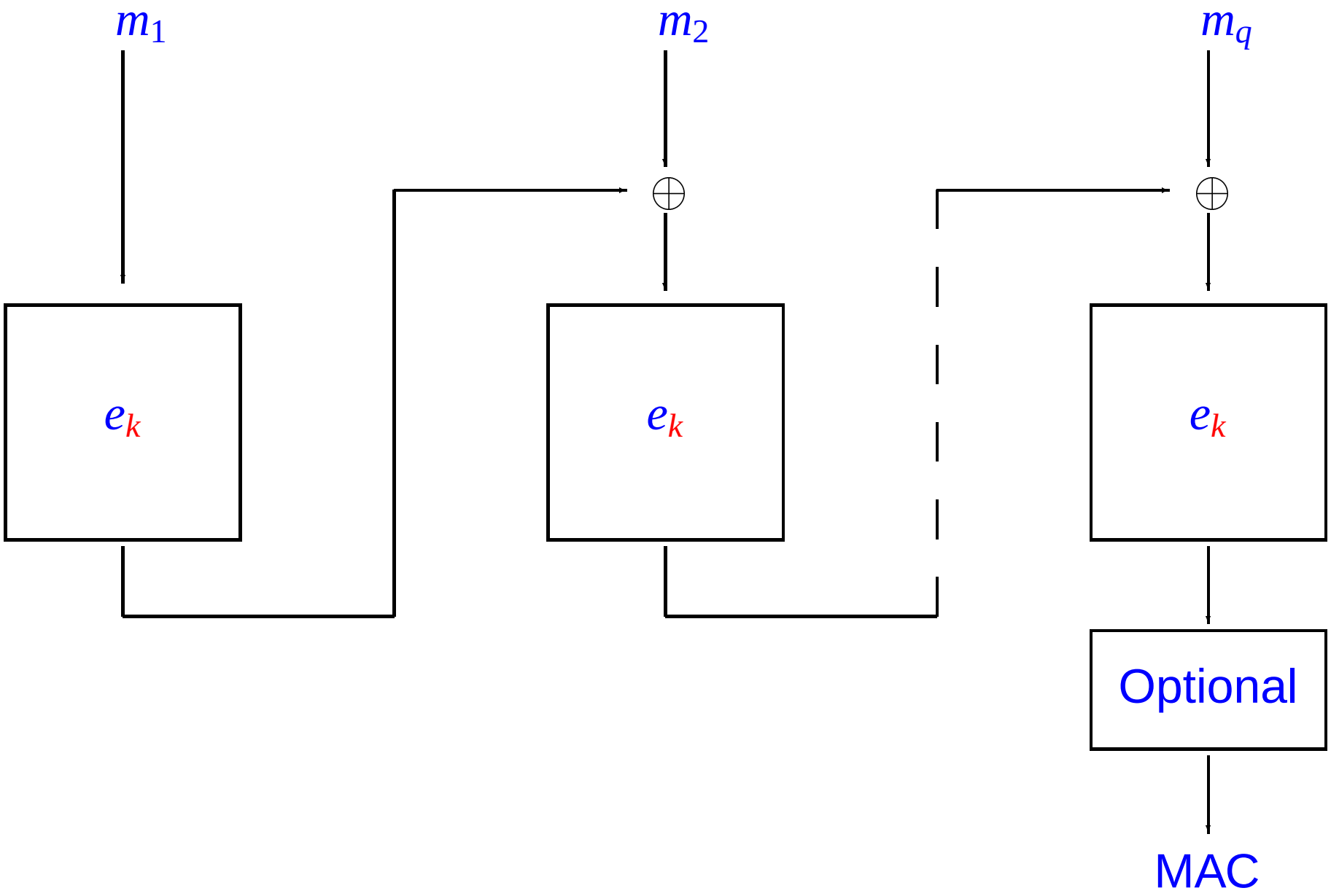
Given an n-bit block cipher, one constructs an m-bit MAC ($m \leq n$) as

- Encipher the blocks using CBC mode (with padding if necessary).
- Last block is the MAC, after optional post-processing and truncation if $m < n$.

If the n-bit data blocks are $m_1, m_2, \ldots, m_q$ then the MAC is computed by

- Put $I_1 = m_1$ and $O_1 = e_k(I_1)$.
- Perform the following for $i = 2, 3, \ldots, q$
  - $I_i = m_i \oplus O_{i-1}$.
  - $O_i = e_k(I_i)$.
- O_q is then subject to an optional post-processing.
- The result is truncated to m bits to give the final MAC.

CBC-MAC - Flow Diagram



CBC-MAC : Padding

There are three possible padding methods proposed in the standards

- Method 1 : Add as many zeros as necessary to make a whole number of blocks.
- Method 2 : Add a single one followed by as many zeros as necessary to make a whole number of blocks.
- Method 3 : As (1) but also add an extra block containing the length of the unpadded message.

The first method does not allow detection of additional or deletion of trailing zeroes.

Unless message length is known by the recipient.

Optional Post-processing

Two specified optional post-processes:

- Choose a key k_1 and compute

$$O_q = e_k\left(d_{k_1}(O_q)\right).$$

- Choose a key k_1 and compute

$$O_q = e_{k_1}(O_q).$$

The optional process can make it more difficult for a cryptanalyst to do an exhaustive key search for the key k.

MACs based on MDCs

Given a key k, how do you transform an MDC h in a MAC ?

Secret prefix method: $MAC_k(m) = h(k||m)$

- Possible to compute $MAC(m||m') = h(k||m||m')$ without knowing k.

Secret suffix method: $MAC_k(m) = h(m||k)$

- Off-line attacks possible to find a collision in the hash function.

Envelope method with padding: $MAC_k(m) = h(k||p||m||k)$

- p is a string used to pad k to the length of one block.

None of these is very secure, better to use HMAC:

$$HMAC_k(m) = h(k||p_1||h(k||p_2||m))$$

with p_1, p_2 fixed strings used to pad k to full block.

MACs versus MDCs

Data integrity without confidentiality:

- MAC: one computes $MAC_k(m)$ and sends $m||MAC_k(m)$.
- MDC: one sends m and computes $MDC(m)$, which needs to be sent over an authenticated channel.

Data integrity with confidentiality:

- MAC: needs two different keys k_1 and k_2.
  - One for encryption and one for MAC.
  - One computes $MAC_{k_1}(m)$ and sends $c = e_{k_2}(m||MAC_{k_1}(m))$.
- MDC: only needs one key k for encryption.
  - One computes $MDC(m)$ and sends $c = e_k(m||MDC(m))$.

Cryptanalysis of Ciphers

Some general methods:

- Exhaustive key search.
- Use of pre-computed tables.
- Divide and conquer.
- Chosen plaintext attack (choose special data which may reveal properties of the key).

In general cryptanalysis one needs a combination of mathematical and puzzle solving skills, plus luck.

Differential and Linear Cryptanalysis

Differential Cryptanalysis

- Look at ciphertext pairs, where the plaintext has a particular difference.
- Probabilistic analysis, yielding structure of the keys.

Linear Cryptanalysis

- Approximates behaviour of non-linear components with linear functions.
- Again uses probabilistic analysis.

Attacks are successful against some ciphers (not DES / Rijndael).

Timing Attacks and Power Analysis

Timing attacks:

- Can be used if execution time of algorithm depends on the key.
- Carefully measuring execution time reveals information about key.

$\Rightarrow$ Solution: make all operations take exactly same amount of time, e.g. by introducing no-ops.

Power Analysis (SPA and DPA):

- Power consumption of device during one encryption reveals info about key.
- If consumption pattern of hardware depends on the instruction, then attacker can deduce the sequence of instructions and parts of key.

$\Rightarrow$ Solution: load balancing, operand masking, desynchronization.

Differential Fault Analysis (DFA)

Basic idea: Use mistakes made (or induced) in the encryption of known plaintext to get information about the key.

Assumes that the attacker has possession of a “tamperproof” device which contains the key, e.g. smartcard.

As an example let us assume a DES box where we know one bit of one intermediate value is flipped.

Operation of DFA

Step 1: identify the round in which the fault occured.

Example:

- If it's in the right half of round 16 then only one bit of the ciphertext is different from what it should be.
- If it's in the left half then the difference is in the output of one or two S-boxes.

Step 2: use the difference distribution table of the S-boxes to guess the 6-bit subkey to the S-box.

DFA - Results

Biham-Shamir: recovered 48-bit last subkey after 50-200 ciphertexts.

To recover the remaining 8 bits of key

- Do brute force attack on the last 8 bits.
- Peel off the last round and repeat the attack.

Works even against triple DES with key of 168 bits.

Can be used against any block cipher and even stream ciphers.

Furthermore, the details of the algorithm do not have to be known, e.g. SkipJack designed by the NSA.