Εισαγωγή στην Κρυπτολογία 2

Ασφάλεια Τηλεπικοινωνιακών Συστημάτων
Κωδικός DIT114
Σταύρος ΝΙΚΟΛΟΠΟΥΛΟΣ

Εμπιστευτικότητα - Κρυπτογραφία

- Συμμετρικοί Αλγόριθμοι Κρυπτογράφησης
  - Αλγόριθμοι ροής – Stream Ciphers
  - Αλγόριθμοι ομάδας – Block Ciphers
- Ασύμμετροι Αλγόριθμοι Κρυπτογράφησης

Συμμετρικοί Αλγόριθμοι Κρυπτογράφησης

Συμμετρικοί αλγόριθμοι Κρυπτογράφησης

Αλγόριθμοι ροής
Stream ciphers

A5, CryptMT, FISH, Grain, HC-256, ISAAC, MUGI, PANAMA, Phelix, Py, Rabbit, RC4, SEAL, SNOW, SOBER, Trivium, VEST

Αλγόριθμοι ομάδας
Block ciphers

Lucifer/DES, IDEA, RC5, Rijndael/AES, Blowfish, Serpent

One-Time Pad

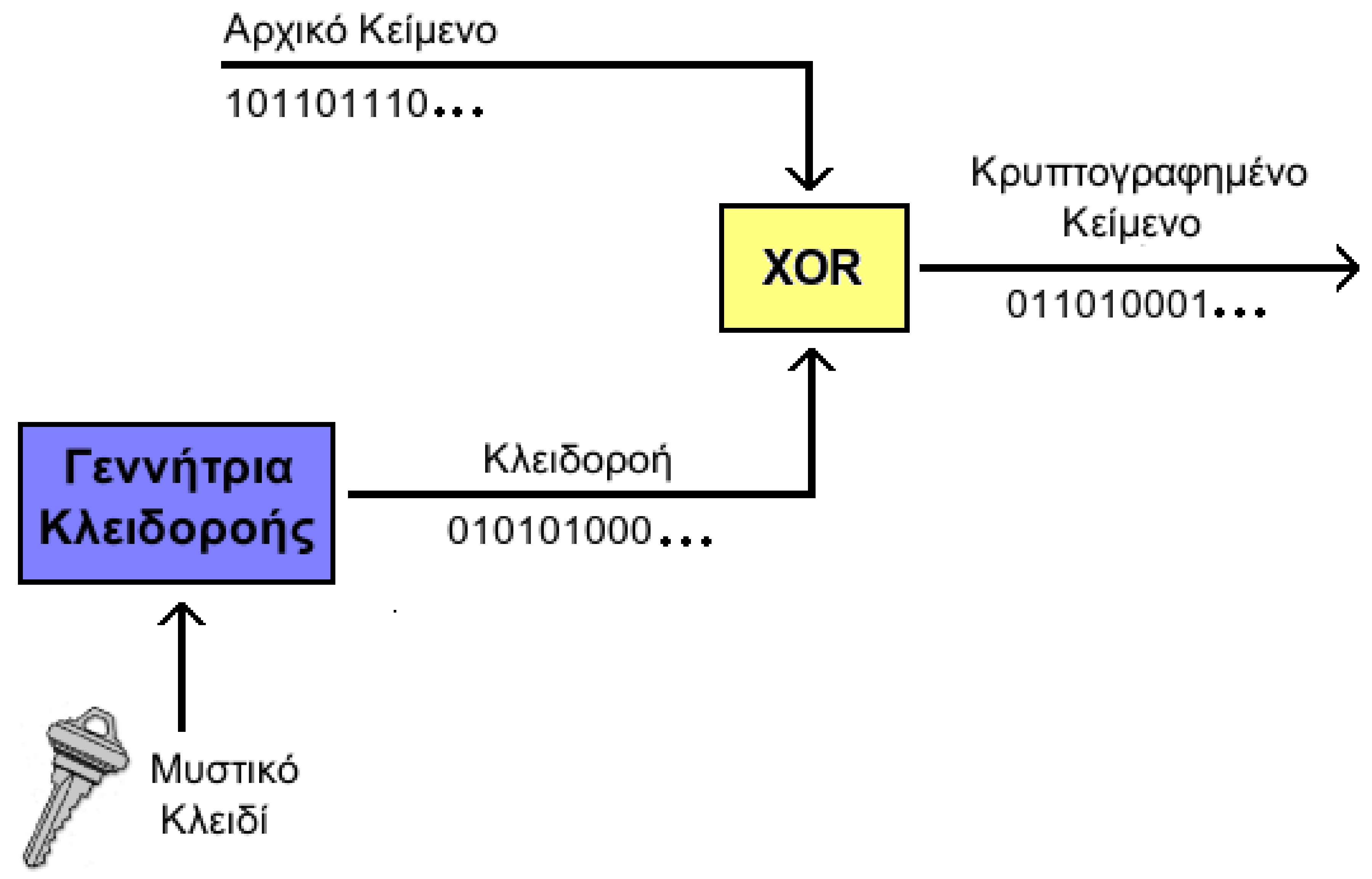
- Developed by Gilbert Vernam in 1918, another name: ***Vernam Cipher***
- The key
  - a truly random sequence of 0's and 1's
  - the same length as the message
  - use one time only
- The encryption
  - adding the key to the message modulo 2, bit by bit.

Encryption $$c_i = m_i \oplus k_i \qquad i = 1, 2, 3, \ldots$$

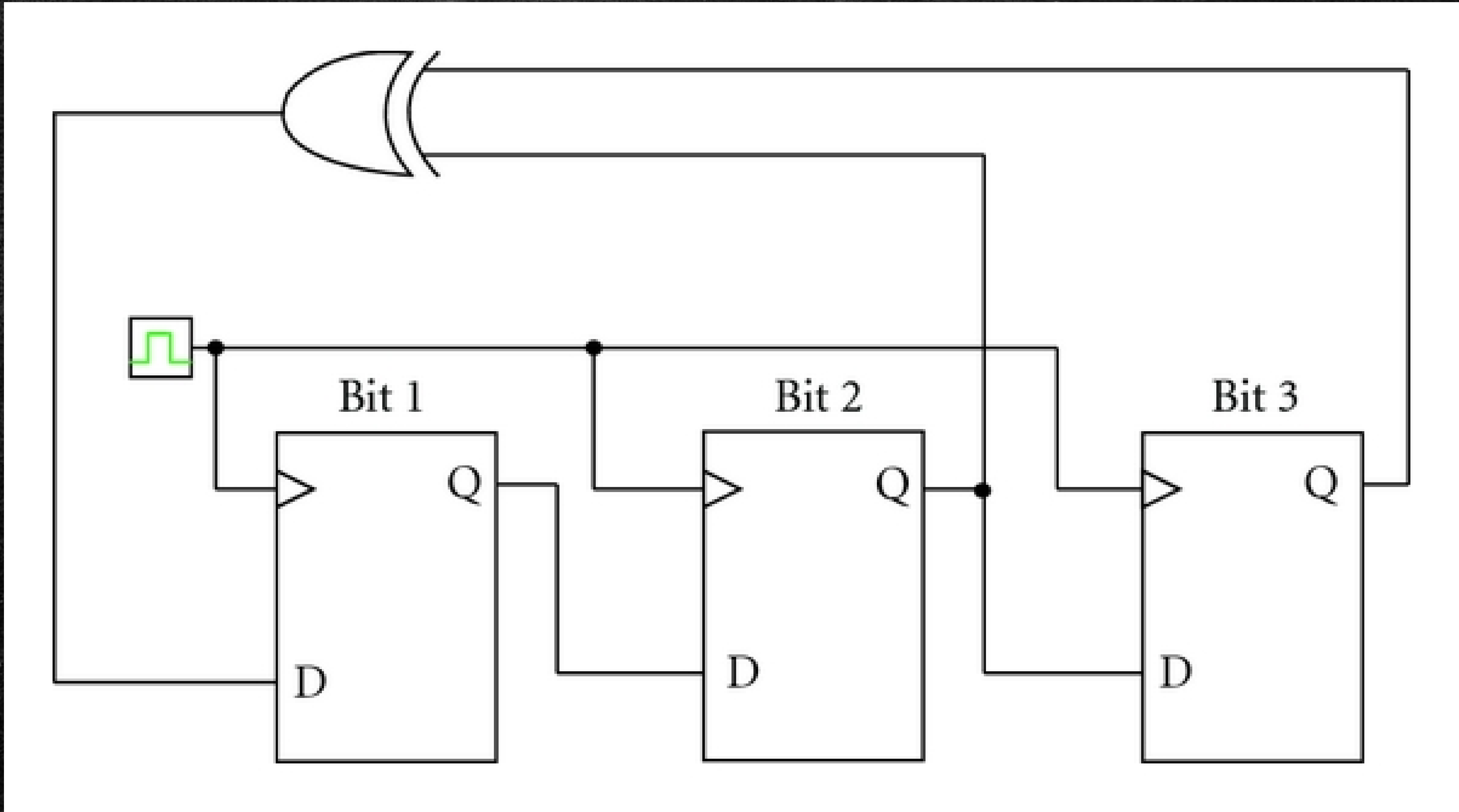
Decryption $$m_i = c_i \oplus k_i \qquad i = 1, 2, 3, \ldots$$

m_i : plain-text bits.
k_i : key (key-stream) bits
c_i : cipher-text bits.

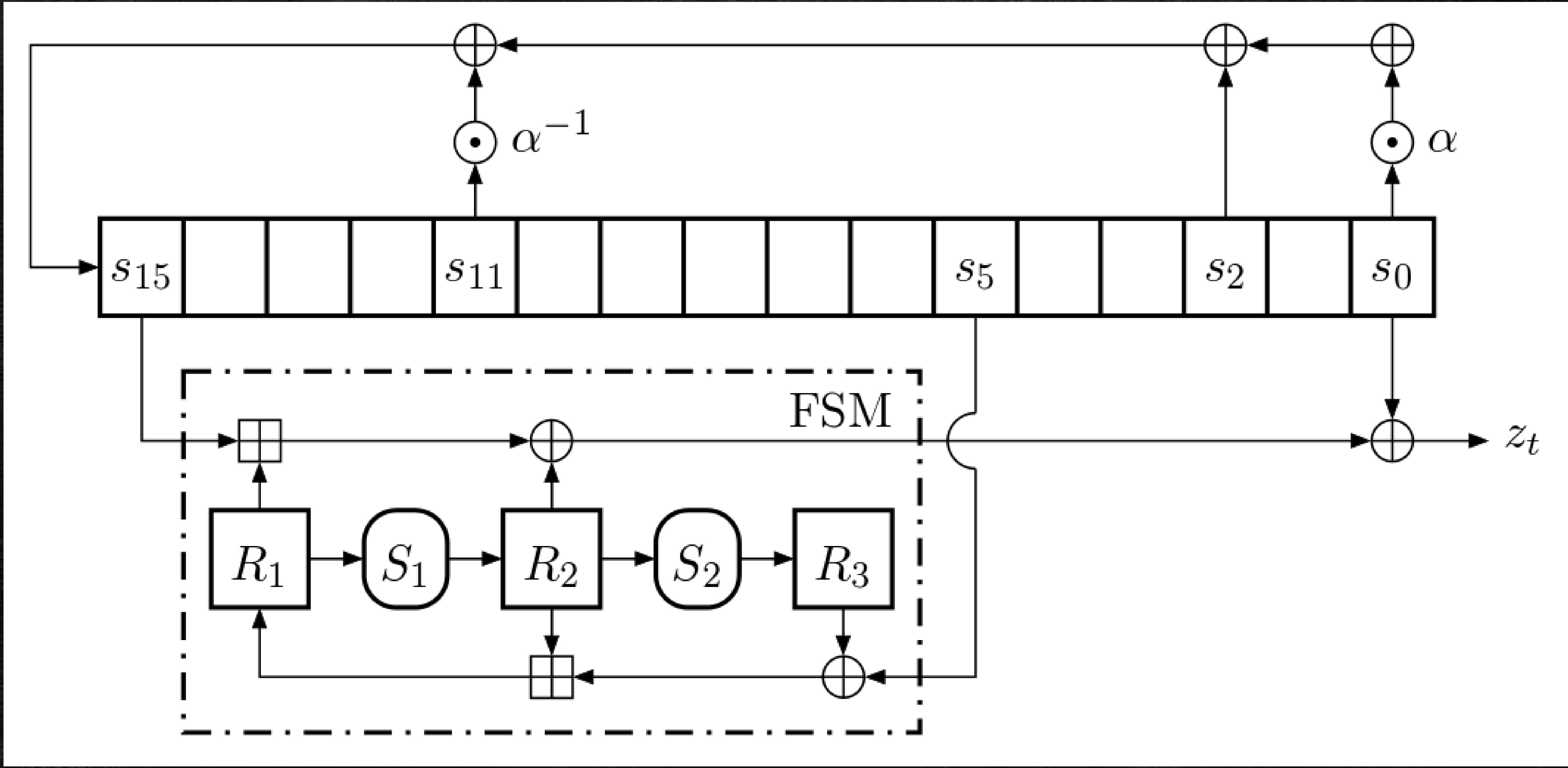
ΑΛΓΟΡΙΘΜΟΙ ΡΟΗΣ - STREAM CIPHERS



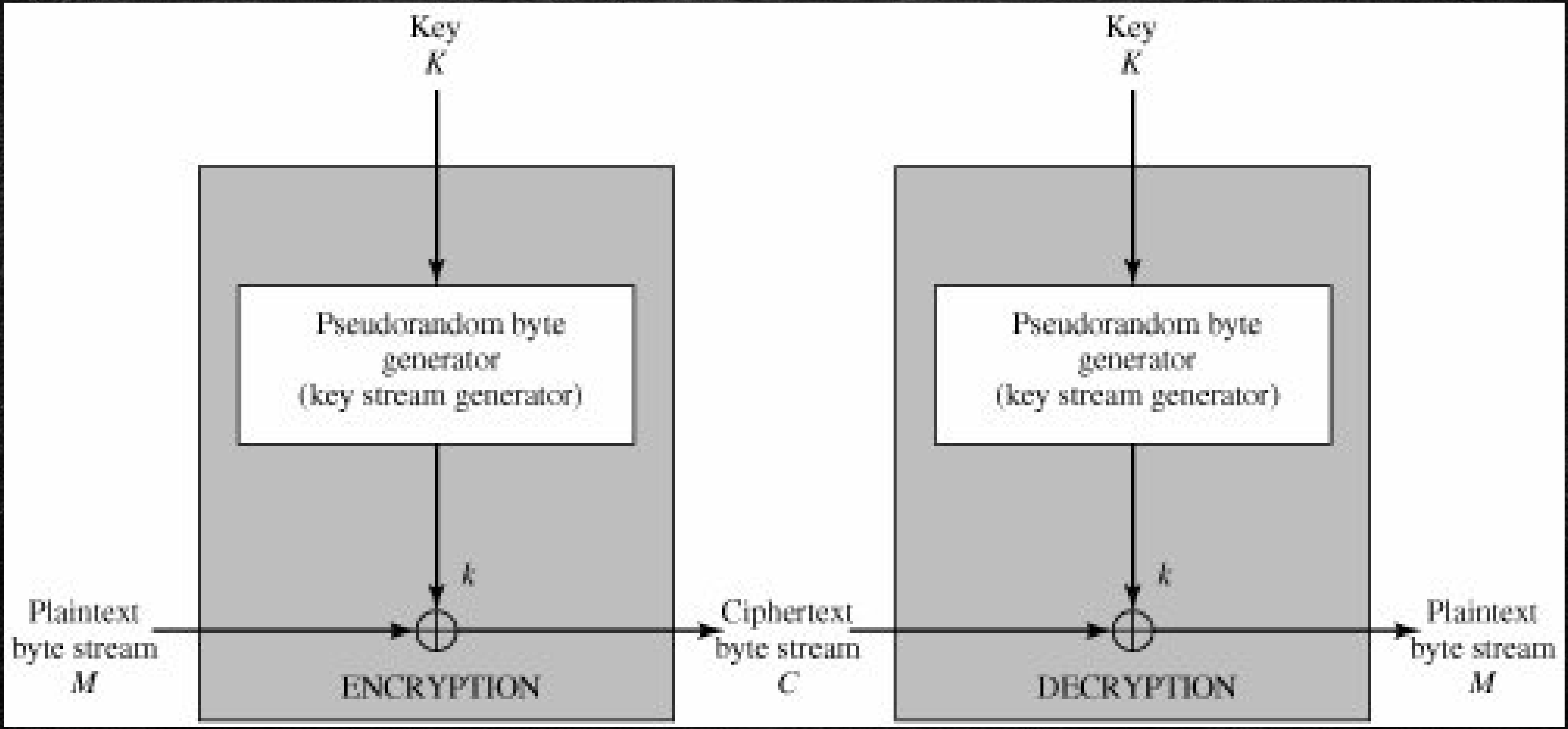
Linear Feedback Shift Registers - LFSR



Non linear Feedback Shift Register - NLFSR



Αλγόριθμος Ροής

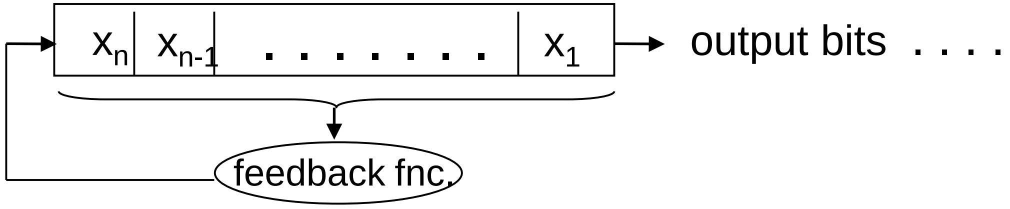


Stream Ciphers

- Generate a *pseudo-random* key stream & xor to the plaintext.
- Key: The seed of the PRNG
- Traditional PRNGs (e.g. those used for simulations) are not secure.
  E.g., the linear congruential generator:
  $$X_i = a X_{i-1} + b \mod m$$
  for some fixed a, b, m.
  It passes the randomness tests, but it is predictible.

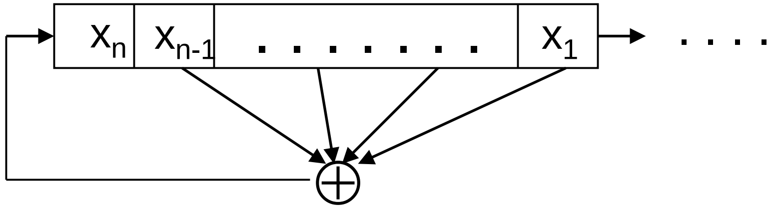
Linear Feedback Shift Registers

Feedback shift register:



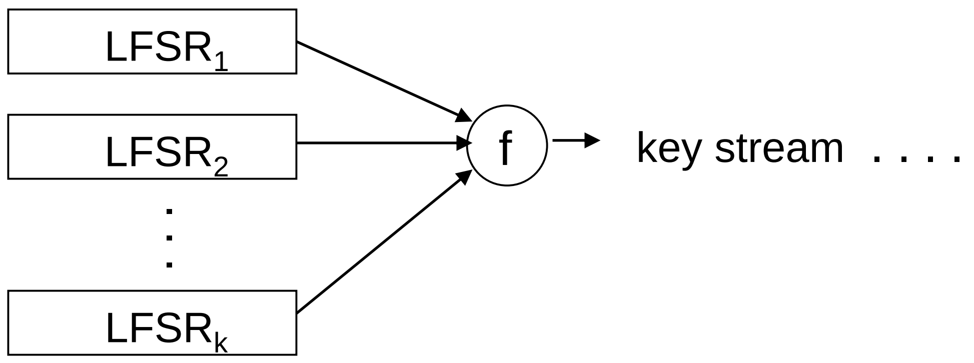
("register", "feedback", "shift")

LFSR: Feedback fnc. is linear over Z_2 (i.e., an xor):



Very compact & efficient in hardware.

Stream Ciphers from LFSRs



Desirable properties of f:

– high non-linearity
– long “cycle period” $(\sim 2^{n1+n2+\ldots+nk})$
– low correlation with the input bits

Example LFSR-Based Ciphers

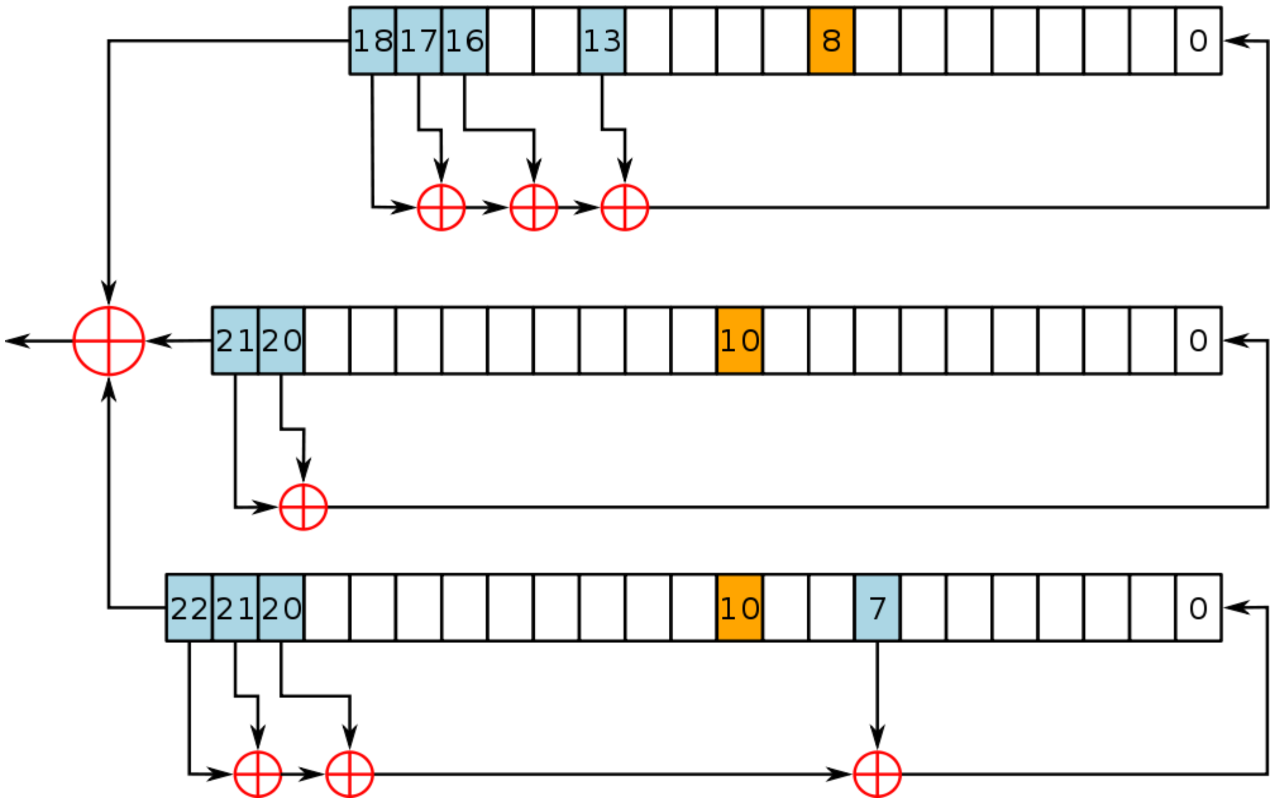
- Geffe Generator:
  - Three LFSRs
  - $LFSR_1$ is used to choose between $LFSR_2$ & $LFSR_3$:

    $$y = (x^{(1)} \wedge x^{(2)}) \oplus (\neg x^{(1)} \wedge x^{(3)})$$
  - Correlation problem: $P(y = x^{(2)}) = 0.75$ (or, $P(y = x^{(3)})$)
- Stop-and-Go Generators:
  - One (or more) LFSR is used to clock the others
  - E.g.: The alternating stop-and-go generator:
    Three LFSRs. If $x^{(1)}$ is 0, $LFSR_2$ is forwarded; otherwise $LFSR_3$. Output is $x^{(2)} \oplus x^{(3)}$.

LFSR-Based Ciphers (cont'd)

- The Shrinking Generator:
  - Two LFSRs
  - If $x^{(1)}$ is 1, output $x^{(2)}$.
    Else, discard both $x^{(1)}$ & $x^{(2)}$; forward the LFSRs.
- A5 (the GSM standard):
  - Three LFSRs; 64 bits in total.
  - Designed secretly. Leaked in 1994.
  - A5/2 is completely broken. (Barkan et al., 2003)
- E0 (Bluetooth's standard encryption)
  - Four LFSRs; 128 bits in total.

GSM A5/1



- The A5/1 stream cipher uses three LFSRs.
- A register is clocked if its clocking bit (orange) agrees with one or both of the clocking bits of the other two registers. (majority match)

Software-Oriented Stream Ciphers

- LFSRs slow in software
- Alternatives:
  - Block ciphers (or hash functions) in CFB, OFB, CTR modes.
  - Stream ciphers designed for software:

    RC4, SEAL, SALSA20, SOSEMANUK…

RC4
(Rivest, 1987)

- Simple, byte-oriented, fast in s/w.
- Popular: Google, MS-Windows, Apple, Oracle Secure SQL, WEP, etc.

Algorithm:

- Works on n-bit words. (typically, $n = 8$)
- State of the cipher: A permutation of $\{0,1,\ldots,N-1\}$, where $N = 2^n$, stored at $S[0,1,\ldots,N-1]$.
- Key schedule: Expands the key (40-256 bits) into the initial state table S.

RC4 (cont'd)

The encryption (i.e., the PRNG) algorithm:

```
i ← 0
j ← 0
loop: {
   i ← i + 1
   j ← j + S[i]
   S[i] ↔ S[j]
   output S[S[i] + S[j]]
}
```

Speed of Software-Oriented Stream Ciphers

(Crypto++ 5.6 benchmarks, 2.2 GHz AMD Opteron 8354. March 2009.)

Algorithm	Speed (MiByte/s.)
3DES / CTR	17
AES-128 / CBC	148
AES-128 / CTR	198
RC4	124
SEAL	447
SOSEMANUK	767
SALSA20	953

RC4 & WEP

WEP: Wired Eqv. Privacy (802.11 encryption prot.)

- RC4 encryption, with 40–104 bit keys.
- 24-bit IV is prepended to the key; RC4(IV || k). IV is changed for each packet.
- Integrity protection: By encrypted CRC-32 checksum.

(What are some obvious problems so far?)

- Key management not specified. (Typically, a key is shared among an AP and all its clients.)
- Design process: Not closed-door, not very public either.

Attacks on WEP

(Borisov, Goldberg, Wagner, 2000)

Obvious problems:

- 24-bit IV too shot; recycles easily. (And in most systems, implemented as a counter starting from 0.)
- CRC is linear; not secure against modifications.
- Even worse: Using CRC with a stream cipher.

Passive decryption attacks:

- Statistical frequency analysis can discover the plaintexts encrypted with the same IV.
- An insider can get the key stream for a packet he sent (i.e., by xoring plaintext and ciphertext); hence can decrypt anyone's packet encrypted with the same IV.

Attacks on WEP (cont'd)

Authentication: challenge-response with RC4

- server sends 128-bit challenge
- client encrypts with RC4 and returns
- server decrypts and compares
- Problem: attacker sees both the challenge & the response; can easily obtain a valid key stream & use it to respond to future challenges.

Attacks on WEP (cont'd)

An active attack:

- Since RC4 is a stream cipher, an attacker can modify the plaintext bits over the ciphertext and fix the CRC checksum accordingly.
- Parts of the plaintext is predictable (e.g., the upper-layer protocol headers).
- Attacker sniffs a packet and changes its IP address to his machine from the ciphertext.
  (If the attacker's machine is outside the firewall, the TCP port number could also be changed, to 80 for example, which most firewalls would not block.)
- Hence, the attacker obtains the decrypted text without breaking the encryption.

Attacks on WEP (cont’d)

A table-based attack:

- An insider generates a packet for each IV.
- Extracts the key stream by xoring the ciphertext with the plaintext.
- Stores all the key streams in a table indexed by the IV. (Requires ~15GB in total.)
- Now he can decrypt any packet sent to that AP.

Note: All these attacks are practical. Some assume a shared key, which is realistic.

Attacks on WEP (cont'd)

- The final nail in the coffin:
  (Fluhrer, Mantin, Shamir, 2001)
  The way RC4 is used in WEP can be broken completely: When IV is known, it is possible to get k in RC4(IV || k).

- WEP2 proposal: 128-bit key, 128-bit IV.
  This can be broken even faster!

Replacements for WEP

- WPA (inc. TKIP)
  - encryption: RC4, but with a complex IV-key mixing
  - integrity: cryptographic checksum (by lightweight Michael algorithm)
  - replay protection: 48-bit seq.no.; also used as IV

- WPA2 (long-term replacement, 802.11i std.)
  - encryption: AES-CTR mode
  - integrity: AES-CBC-MAC

Τύποι Αλγορίθμων Ροής

Ετεροσυγχρονιζόμενοι αλγόριθμοι ροής: η κλείδα αλλάζει, όταν προκύψει πρόβλημα.
Αυτο-συγχρονιζόμενοι αλγόριθμοι ροής: η κλείδα αλλάζει σε τακτά διαστήματα ροής N bits.

Αλγόριθμοι ροής, χρήση

Ταχείς αλγόριθμοι
Απλοί σε υλοποίηση σε hardware
Δεν υπάρχει σταθερό μήκος ανοικτού κειμένου (πχ ασύρματη σύνδεση)
Τελείως ξεχωριστή δομή του κρυπτογραφικού αλγορίθμου
Bulk encryption

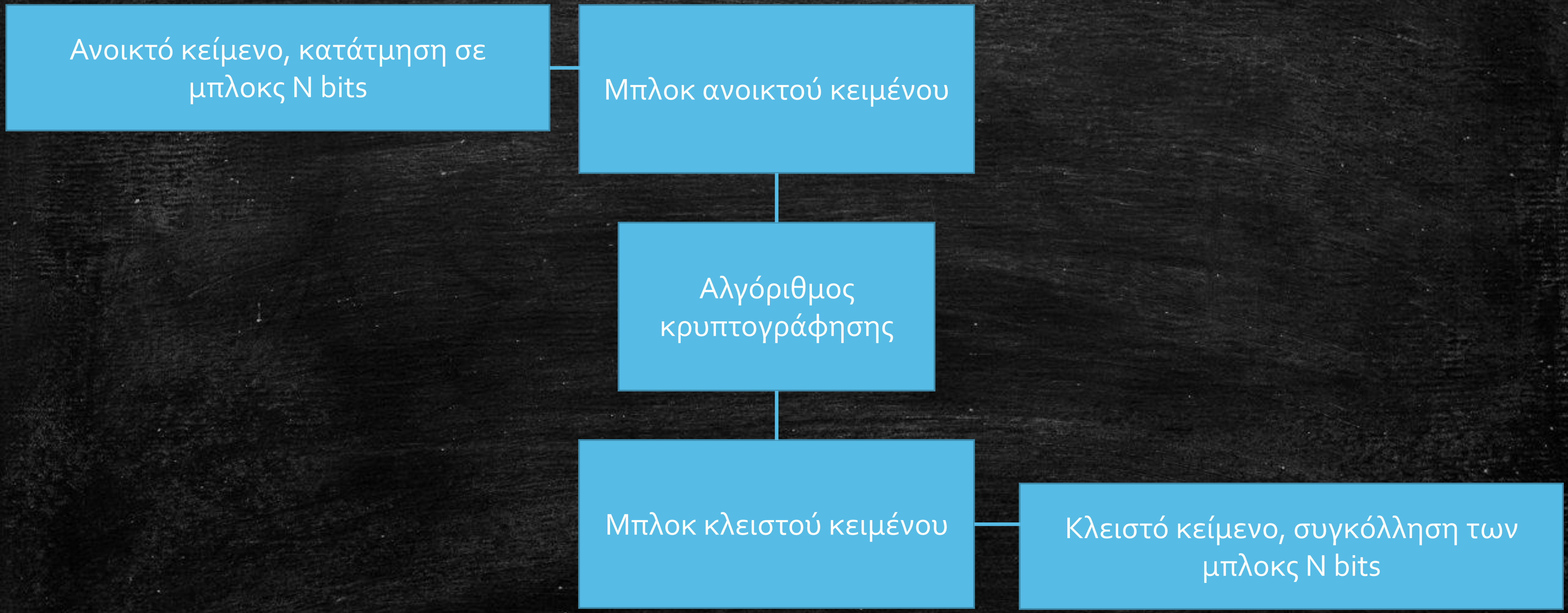
Ασφάλεια Αλγορίθμων ροής

Τυχαία ακολουθία εξόδου (keystream) – άπειρη περίοδο
Η ακολουθία εξόδου πρέπει να είναι ανεξάρτητη από την κλείδα
Η ακολουθία εξόδου πρέπει να είναι ανεξάρτητη από το κρυπτογραφικό σύνθημα (cryptographic nonce).
Να μην υπάρχουν ασθενείς κλείδες (ακόμη και αν ο επιτιθέμενος «μαντέψει» τμήματα του ανοικτού και του κλειστού κειμένου)

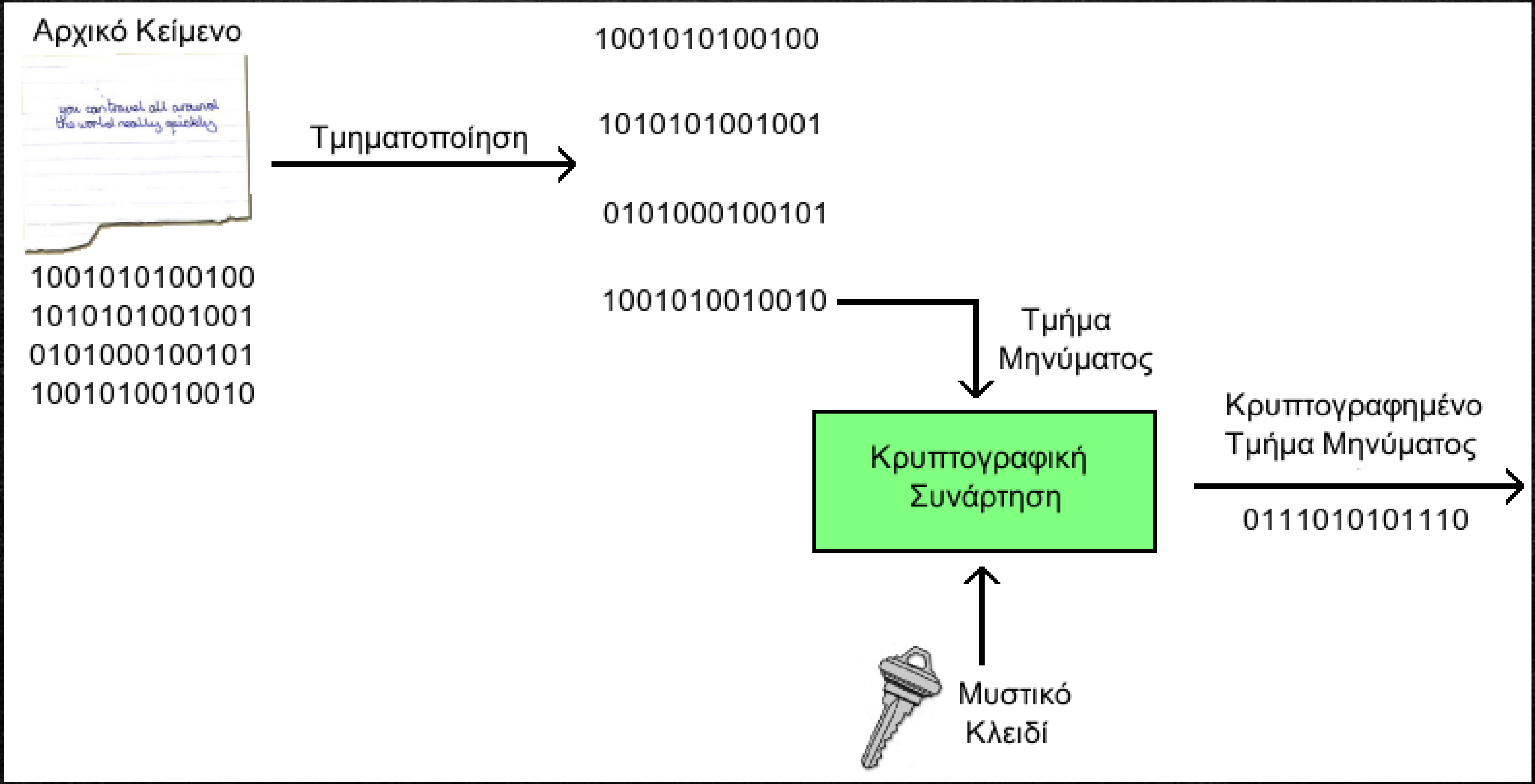
Αλγόριθμοι ροής, προβλήματα

- Διαχείριση κλειδών.
  - ανταλλαγή
  - Διασπορά κινδύνου στην περίπτωση πολλών κατόχων
- Κακό κανάλι επικοινωνίας που απαιτεί συχνό επανασυγχρονισμό
- Ύπαρξη ασθενών κλείδων.

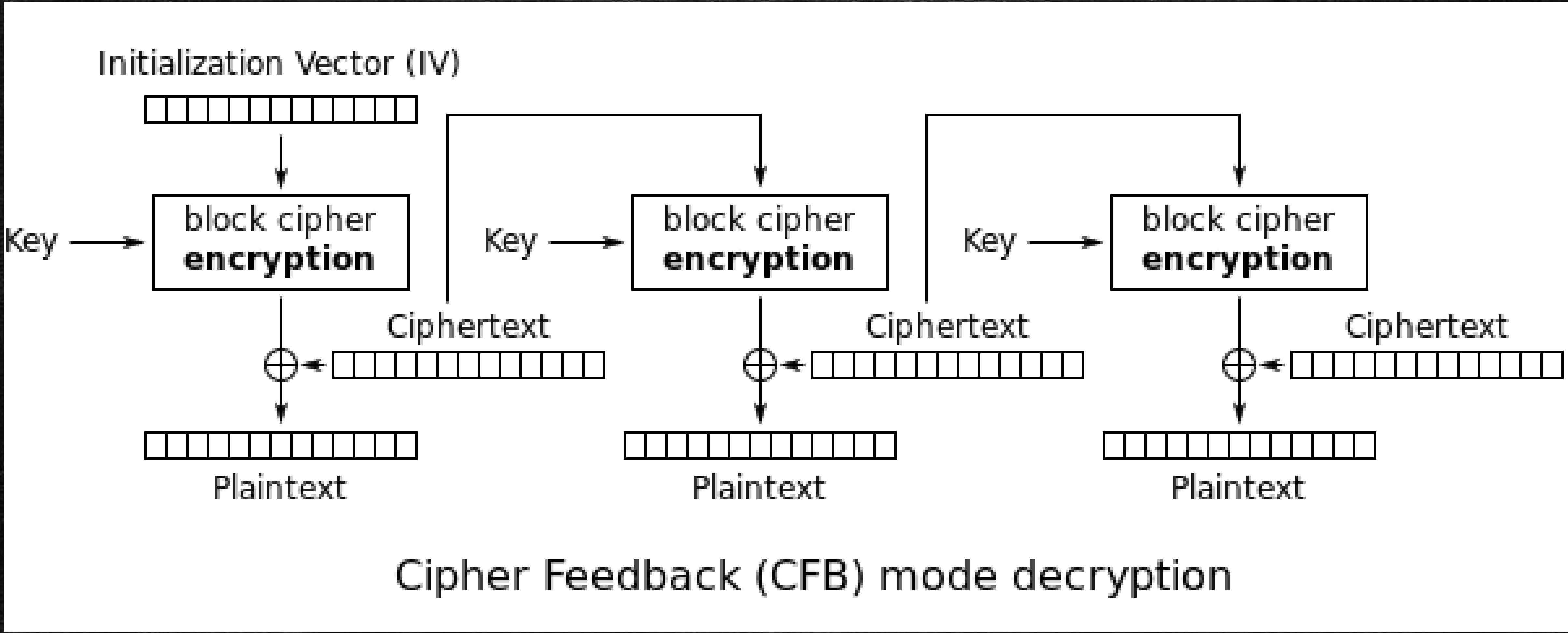
Αλγόριθμοι ομάδας – Block ciphers



Αλγόριθμοι ομάδας – Block ciphers

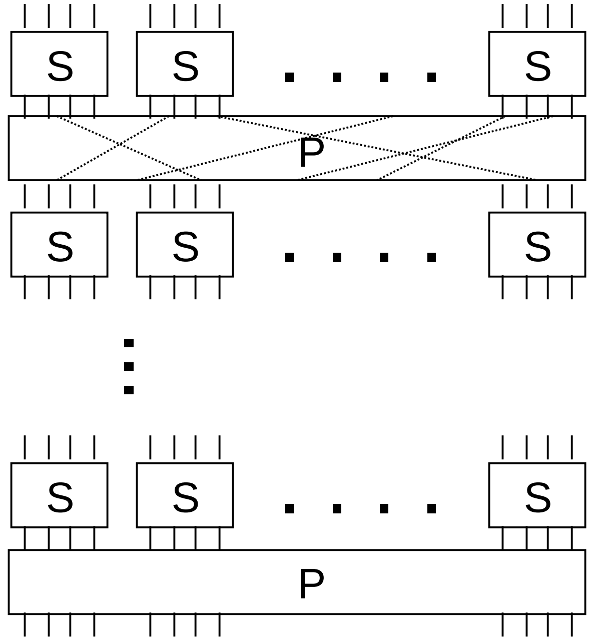


Αλγόριθμοι ομάδας – Block ciphers



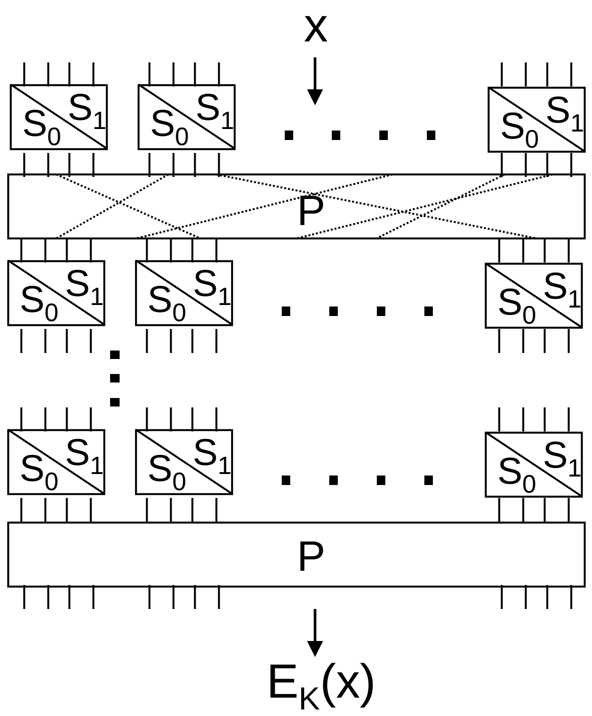
Block Ciphers & S-P Networks

- Block Ciphers: Substitution ciphers with large block size (≥ 64 bits)
- How to define a good substitution for such large blocks?
- "SP Networks" (Shannon, 1949)
  - small, carefully designed substitution boxes ("confusion")
  - their output mixed by a permutation box ("diffusion")
  - iterated a certain number of times



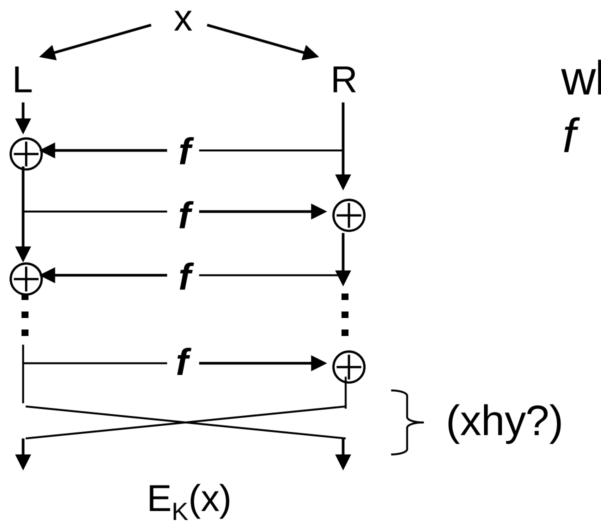
Lucifer

- Early 1970s: First serious needs for civilian encryption (in electronic banking)
- IBM's response: Lucifer, an iterated SP cipher
- Lucifer (v0):
  - Two fixed, 4x4 s-boxes, S_0 & S_1
  - A fixed permutation P
  - Key bits determine which s-box is to be used at each position
  - 8 x 64/4 = 128 key bits (for 64-bit block, 8 rounds)

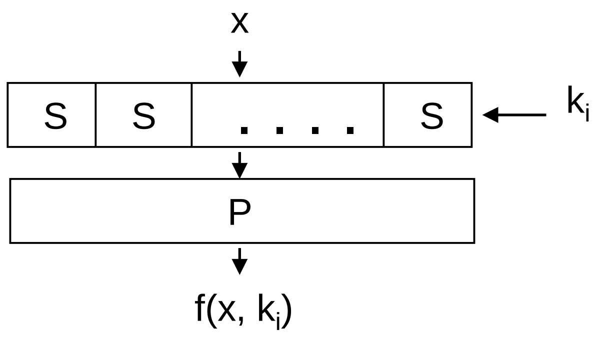


Feistel Ciphers

- A straightforward SP cipher needs twice the hardware: one for encryption (S, P), one for decryption (S^{-1}, P^{-1}).
- Feistel's solution:



where the *f* function is SP:



- Lucifer v1: Feistel SP cipher; 64-bit block, 128-bit key, 16 rounds.

Data Encryption Standard (DES)

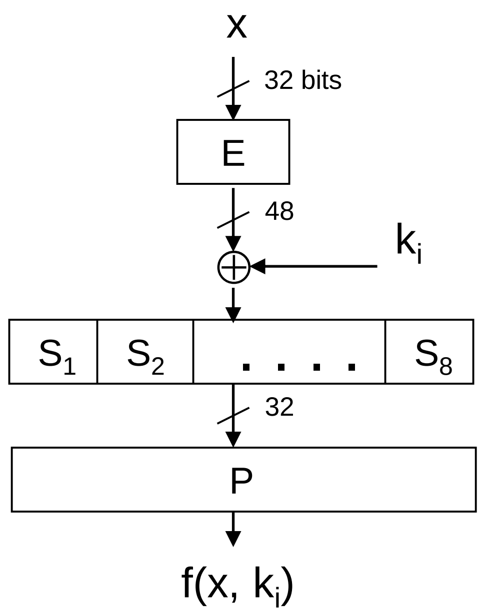
- Need for a standardized cipher to protect computer and communications data

- NBS' request for proposals (1973)

- IBM's submission Lucifer is adopted after a revision by NSA.

From Lucifer to DES

- 8 fixed, 6x4 s-boxes (non-invertible)
- expansion E (simple duplication of 16 bits)
- round keys are used only for xor with the input
- 56-bit key size
- 16 x 48 round key bits are selected from the 56-bit master key by the “key schedule”.



The DES Controversy

- Design process not made public.
  Any hidden trapdoors in the s-boxes?

- 56-bit key length is too short.
  Is it so that NSA can break it?

Strengthening DES

- Multiple DES encryption

  3DES: $E_{K3}(D_{K2}(E_{K1}(x)))$

  – Why not 2DES? (112-bit key not long enough?)

  – Why "D"?

  – Two-key 3DES: K3 = K1

- DES-X (Rivest, 1995)

  $$E_K(x \oplus K1) \oplus K2$$

  – overhead cost minimal

  – construction is provably secure (Rogaway & Killian)

  – Why not

  $$E_K(x) \oplus K2$$

  or

  $$E_K(x \oplus K1)\ ?$$

Τύποι Αλγορίθμων Ομάδας

- Επαναλαμβανόμενοι αλγόριθμοι ομάδας
- Αλγόριθμοι αντικατάστασης – διάχυσης
- Αλγόριθμοι Feistel
- Αλγόριθμοι Lai-Massey

Αλγόριθμοι ομάδας, χρήση

Αξιόπιστοι και ασφαλείς αλγόριθμοι.
Ανθεκτικοί σε επιθέσεις κρυπτανάλυσης
Υπάρχει σταθερό μήκος ανοικτού κειμένου (πχ ενσύρματη σύνδεση)
Πολύπλοκη δομή
Ασύγχρονη επικοινωνία

Ασφάλεια Αλγορίθμων Ομάδας

- Αποδεινύεται με:
  - Στατιστικές μεθόδους
  - Ελέγχους επαναληψιμότητας
  - Κρυπταναλυτικές επιθέσεις
- Η ακολουθία εξόδου πρέπει να είναι ανεξάρτητη από την κλείδα
- Η ακολουθία εξόδου πρέπει να είναι ανεξάρτητη από το κρυπτογραφικό σύνθημα (cryptographic nonce).
- Να μην υπάρχουν ασθενείς κλείδες (ακόμη και αν ο επιτιθέμενος «μαντέψει» τμήματα του ανοικτού και του κλειστού κειμένου)

Αλγόριθμοι ομάδας, προβλήματα

- Διαχείριση κλειδών.
  - ανταλλαγή
  - Διασπορά κινδύνου στην περίπτωση πολλών κατόχων.

Agenda

- **Introduction**
- Block Ciphers
  - Definition
  - Standards Competitions and Requirements
  - Common Building Blocks
  - Examples
  - Modes of Encryption

Introduction

- Intended as an overview
- Practical focus
- Cover many topics instead of a few in-depth
- Examples of ciphers – show variety of designs while using basic building blocks

Uses

- Types of data
- Files, disk, large plaintext
- Not streaming, unless in keystream mode of encryption
- Random number generator: RSA token, VASCO digipass (OTPs)

Symmetric Key Cryptography

- Secret key – one key
- General categories of algorithms
  - Block Ciphers
  - Stream Ciphers
- Heuristics
  - Well analyzed
  - Components based on defined properties
  - But, unlike public key, no formal security proof exists
- Faster than public key algorithms

Why Understand Symmetric Key Cipher Design?

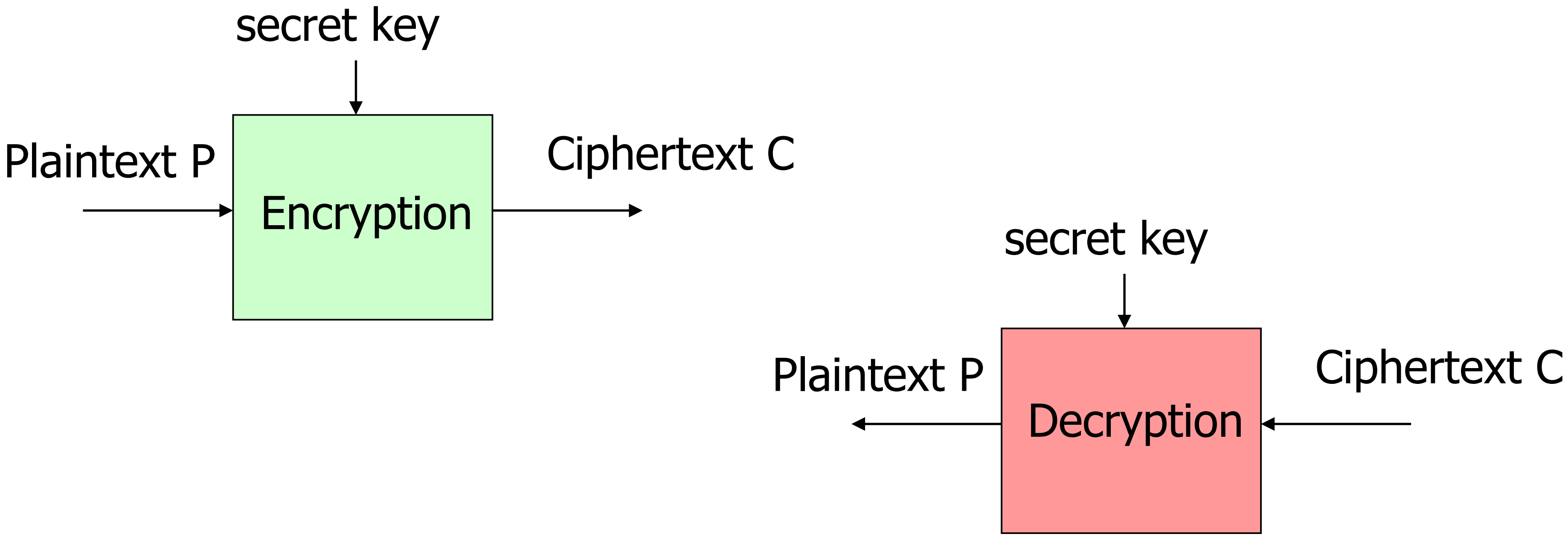
- If develop own library – efficient implementation, need to avoid errors due to misunderstanding or “alterations” to obtain resource savings
- If involve in selecting ciphers for an application, lack of analysis may result in problems later – ex. cellular encryption algorithms
- Using a proprietary cipher is generally not feasible – it will be reversed engineered

Agenda

- Introduction
- Block Ciphers
  - **Definition**
  - Standards Competitions and Requirements
  - Common Building Blocks
  - Examples
  - Modes of Encryption

Block Ciphers

- Input data (plaintext) and a secret key
- Get output (ciphertext)



Block Ciphers - Definition

- A block cipher operating on b-bit inputs is a family of permutations on b bits with the key given to the block cipher used to select the permutation.
- k: q-bit key.
- P: b-bit string denoting a plaintext.
- C: b-bit string denoting a ciphertext.

Block Ciphers - Definition

2 bit block cipher, 2 bit key with encryption function defined by:

Key 00

P	C
00	10
01	11
10	01
11	00

Key 01

P	C
00	11
01	00
10	01
11	10

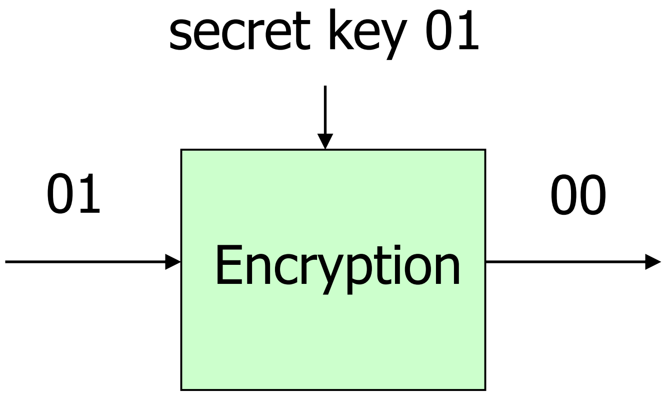
Key 10

P	C
00	11
01	10
10	01
11	00

Key 11

P	C
00	01
01	00
10	11
11	10

In practice, infeasible to store representation of block cipher as tables: example: 2^{128}



Block Ciphers - Definition

- An encryption function: $E = \{E_k\}$ is a family of 2^q permutations on b bits indexed by k, where k is q bits
- A decryption function: $D = \{D_k\}$ is a family of 2^q permutations on b bits indexed by k such that D_k is the inverse of E_k.
- Given a b-bit plaintext, P, and key, k, if $C = E_k(P)$ then $P = D_k(C)$.

Block Ciphers - Definition

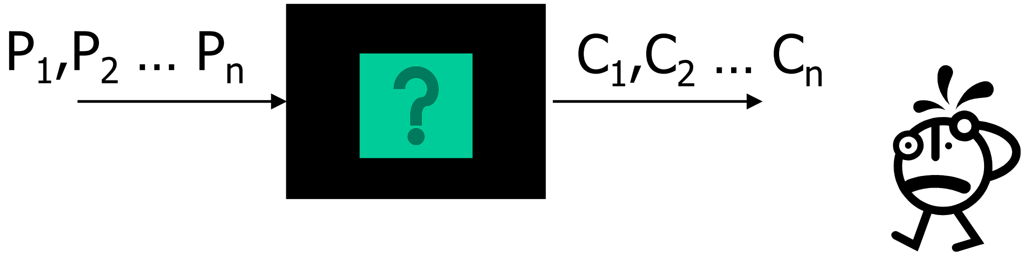
- In practice, a block cipher will take as input a secret key, k, and apply a function, F, called a key schedule, to k that expands k into an expanded key, ek= F(k).
- k is usually 128, 192 or 256 bits and ek is often more than 100 bytes.
- Discuss later – key schedules defined to be computationally efficient at the cost of a lack of randomness in the expanded-key bits.

Block Ciphers - Definition

- Consider a block cipher with 128 bit plaintext and 128 bit key
  - 2^{128} possible plaintexts
  - $2^{128}!$ possible permutations
- Key is index to permutation to use:
  - Only 2^{128} permutations used by the block cipher

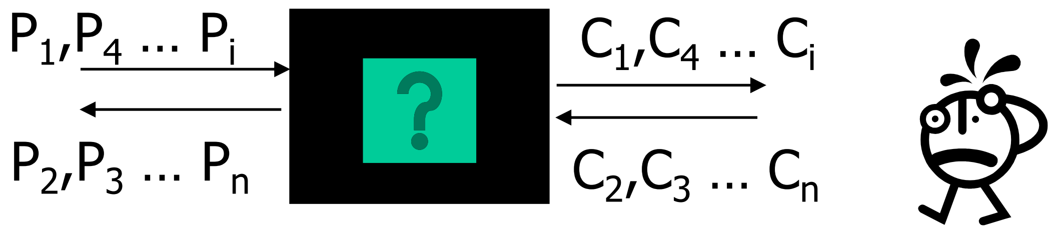
Pseudorandom Permutation Definition

- Property of ideal (in theory) block cipher: strong PRP
- Box contains either the block cipher or a random permutation
- Pseudorandom permutation (PRP): Attacker cannot make polynomial many adaptive chosen plaintext or adaptive chosen ciphertext queries (but not both) and determine contents of box with probability ½ + *e* for non-negligible *e* > 0.



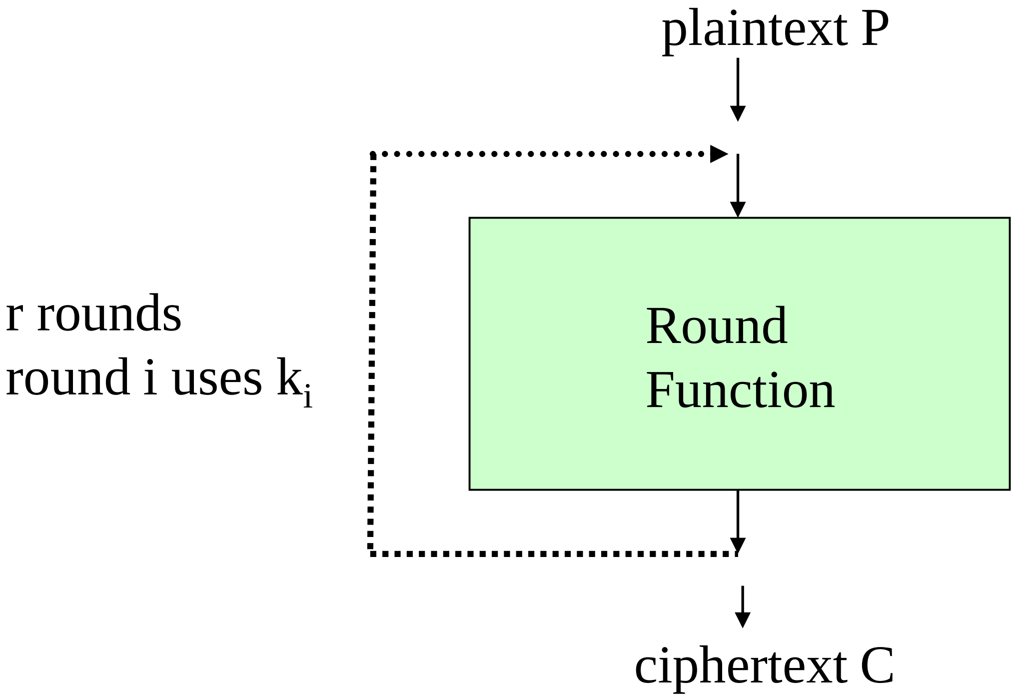
Strong PRP Definition

- Strong PRP (SPRP): same idea as PRP, but can make queries in both directions



Typical Block Cipher Structure

- P,C are fixed length (*e.g.* 128 or 256 bits)
- Secret key, K, expanded via a function called a key schedule to create round keys $k_1, k_2, \ldots k_r$



Parameters

- Block size: 128 bits minimum, 256 bits (64-bit ciphers still in use due to existing implementations – ex. 3DES, Kasumi)
- Key size: 128 typical, 192, 256 bits

Modes of Encryption

- Block cipher is used in a mode of encryption
- Block-by-block encryption (ECB – Electronic Code Book) can result in patterns being detectable
- Common modes presented later

Agenda

- Introduction
- Block Ciphers
  - Definition
  - **Standards Competitions and Requirements**
  - Common Building Blocks
  - Examples
  - Modes of Encryption

Standards Competitions

- NIST Advanced Encryption Standard (AES) – US, November 2001
- New European Schemes for Signatures, Integrity, and Encryption (NESSIE) – European Union, March 2003
- Cryptography Research and Evaluation Committee (Cryptrec) – Japan’s government, August 2003

Standards Competitions

- NIST: AES (Rijndael)
- NESSIE: AES, Camellia
- Cryptrec: AES, Camellia, Hierocrypt-3*, SC2000*

- NIST AES runner-ups: Mars, RC6, Serpent, Twofish
- NESSIE 64-bit: MISTY1
- Cryptrec 64-bit: CIPHERUNICORN
- Other:
  - Kasumi (64-bit block, 128-bit key): 1999 – modified MISTY1, used in 3GPP
  - DES (64-bit block, 64-bit key with 56 bits used – 3DES, NIST standard 1976-2001)

*Also submitted to NESSIE but not selected

Requirements - NIST

- Security:
  - Resistance to cryptanalysis
  - Soundness of the mathematical basis
  - Randomness of the ciphertext
- Costs:
  - System resources (hardware and software) required
  - Monetary costs
- Algorithm and implementation characteristics
  - Use for other cryptographic purposes (hash function, a random bit generator and a stream cipher - such as via CTR mode)
  - Encryption and decryption using the same algorithm
  - Ability to implement the algorithm in both software and hardware
  - Simplicity: reduces implementation errors and impacts costs, such as power consumption, number of hardware gates and execution time

Requirements - NESSIE

"Simplicity and clarity of design are important considerations. Variable parameter sizes are less important."

Selection criteria divided into four areas:

- Security: resistance to cryptanalysis.
- Market requirements: feasibility of implementation from a technical perspective (cost-efficient implementations) and business perspective (free of licensing restrictions).
- Performance and flexibility: range of environments in which the algorithm could efficiently be implemented. Software considerations included 8-bit processors (as found in inexpensive smart cards), 32-bit and 64-bit processors. For hardware, both field-programmable gate arrays (FPGAs) and application-specific integrated circuits (ASICs) were considered.
- Flexibility: use in multiple applications and for multiple purposes

Requirements - NESSIE

Three categories of block ciphers:

- High security: keys $\geq$ 256 bits, block length of 128 bits.
- Normal security: keys $\geq$ 128 bits and a block length of 128 bits.
- Normal legacy: keys $\geq$ 128 bits and a block length of 64 bits.
- In all categories: minimal attack workload must be least $O(2^{80})$ triple DES encryptions

Agenda

- Introduction
- Block Ciphers
  - Definition
  - Standards Competitions and Requirements
  - **Common Building Blocks**
  - Examples
  - Modes of Encryption

Terms

Confusion:

- obscure relationship between plaintext and ciphertext

Diffusion:

- Spread influence of a plaintext bit and/or key bit over ciphertext (avalanche effect)
- Hides statistical relationships between plaintext and ciphertext
- Ideally (not in practice) if a single plaintext bit changes, every ciphertext bit should change with probability ½.

Suppose encrypting plaintext 1111111111111111 produces ciphertext 0110110000101001

Then encrypt 1111111011111111, can't predict anything about ciphertext

Terms

Differential

- Two inputs to a function: P_1, P_2
- Corresponding outputs C_1,C_2
- Differential is $P_1 \oplus P_2$, $C_1 \oplus C_2$

Linear relationship

- Input P, output C, key K
- Linear equation consisting of P_i, C_i, K_i bits that holds with probability ½ + *e* for non-negligible *e*
- Example: $P_1 \oplus K_2 = C_{10}$ with probability ¾

Agenda

- Introduction
- Block Ciphers
  - Definition
  - Standards Competitions and Requirements
  - Common Building Blocks
  - **Examples**
  - Modes of Encryption

Common Building Blocks

Substitution-Permutation Network (SPN)

- General term for sequence of operations that performs substitutions and permutations on bits

Feistel Network (will see example later)

- For input $L_0 \| R_0$ and any function F
- $L_i = R_{i-1}$
- $R_i = L_{i-1} \oplus F(R_{i-1},K_i)$
- K_i = other input to F, (ex. key material)

Whitening

- XOR data with key material ($X \oplus K$)
- Helps break relationship between output of one round and input to next round

Common Building Blocks

Substitution Boxes (S-Box)

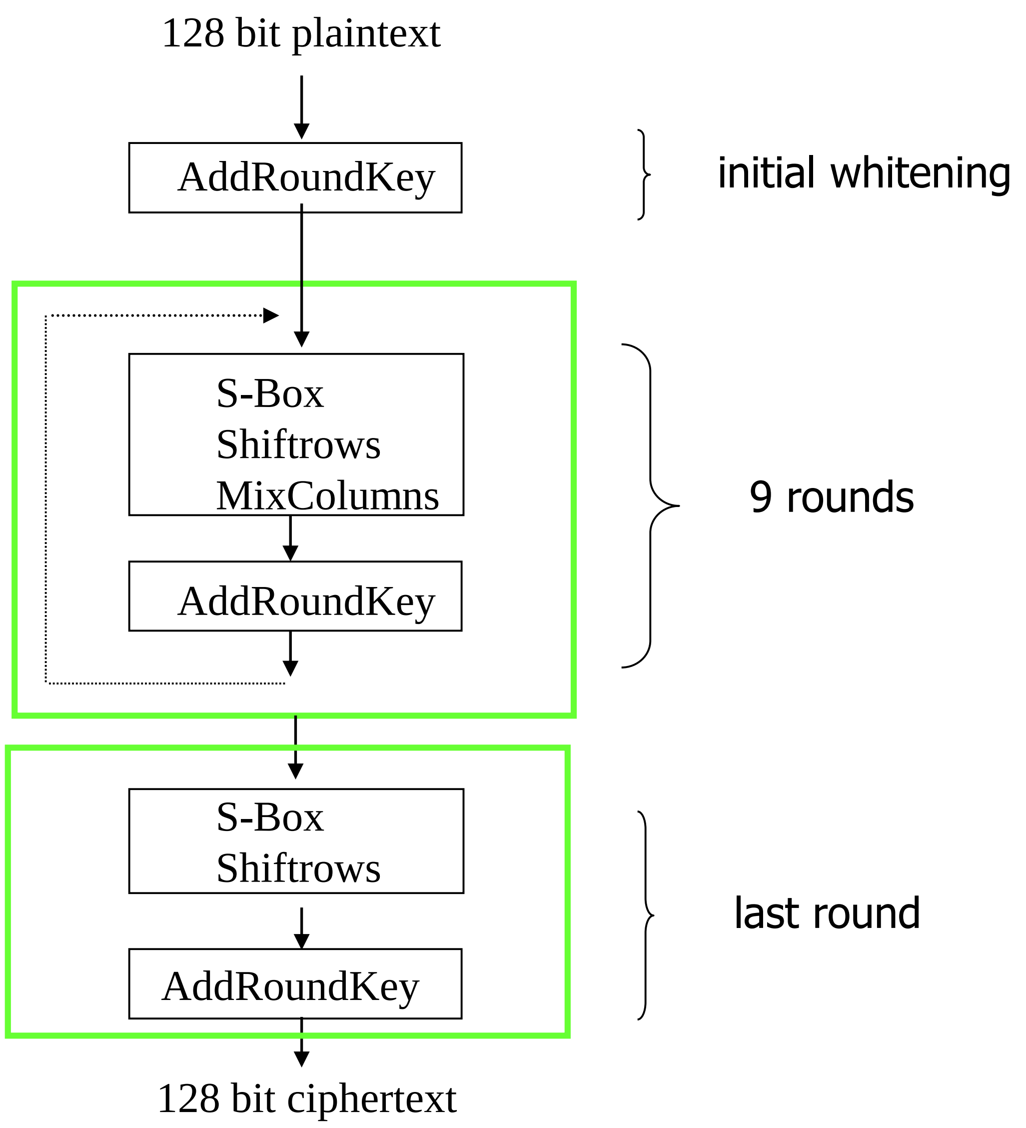
- Based on data (and sometimes key bits), replace data
- Designed to minimize differential and linear relationships

key bits

data bits

	00	01	10	11
00	10	11	01	00
01	11	01	00	10
10	01	00	10	11
11	00	10	11	01

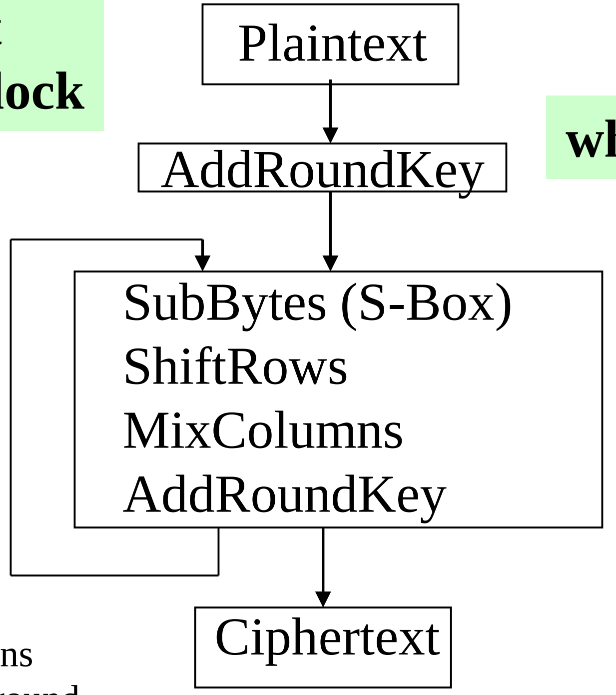
AES – 128 bit block



AES

AES

128 bit data block



whitening

Keyless permutations and substitutions.

⊕ with expanded key bytes

Nr rounds
MixColumns
not in last round

Variable key length and # of rounds.

key length in bits	Nk = # of 32 bit words in key	Nb = # of words in input/output (128 bits)	Nr = # of rounds
128	4	4	10
192	6	4	12
256	8	4	14

Decryption not same as encryption.

AES Round Function Components: Encryption

SubBytes **S-Box (table lookup at byte level, see FIPS197 for table values)**

ShiftRows

A:

sij is a byte

s00	s01	s02	s03
s10	s11	s12	s13
s20	s21	s22	s23
s30	s31	s32	s33

→ Shift row i i positions (i = 0 to 3)

s00	s01	s02	s03
s11	s12	s13	s10
s22	s23	s20	s21
s33	s30	s31	s32

MixColumns

$$A \leftarrow \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} * A$$

(in hex)

Usually implemented as a table lookup
Coefficients of a polynomial

AddRoundKey

$$A \leftarrow \text{round\_key} \oplus A$$

AES Diffusion: Single Byte

Round 1

Input

s00	s01	s02	s03
s10	s11	s12	s13
s20	s21	s22	s23
s30	s31	s32	s33

After ShiftRows

s00	s01	s02	s03
s11	s12	s13	s10
s22	s23	s20	s21
s33	s30	s31	s32

After MixColumns

s’00	s’01	s’02	s’03
s’11	s’12	s’13	s’10
s’22	s’23	s’20	s’21
s’33	s’30	s’31	s’32

Note: AddRoundKey has no impact on diffusion

Round 2

s’00	s’01	s’02	s’03
s’12	s’13	s’10	s’11
s’20	s’21	s’22	s’23
s’32	s’33	s’30	s’31

s”00	S”01	s”02	s”03
s”12	s”13	s”10	s”11
s”20	s”21	s”22	s”23
s”32	s”33	s”30	s”31

AES Round Function

- Can be collapsed to 4 table lookups and 4 XORs using 32-bit values (tables for last round differ – no MixColumns step)
- XOR result with round key

AES Decryption

SubBytes **S-Box inverse**
(see FIPS197 for table values)

ShiftRows **reverse shift**

A:

s00	s01	s02	s03
s10	s11	s12	s13
s20	s21	s22	s23
s30	s31	s32	s33

← Shift row i i positions (i = 0 to 3)

s00	s01	s02	s03
s11	s12	s13	s10
s22	s23	s20	s21
s33	s30	s31	s32

sij is a byte

MixColumns

$$A \leftarrow \begin{pmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{pmatrix} * A$$

(in hex)

AddRoundKey

$$A \leftarrow \text{round\_key} \oplus A$$

AES Key Schedule

w_i = i^th 32 bit word of the expanded key

For 1^st Nk words: w_i = i^th word of key (Nk=4 for 128 bit keys)
i.e. key is used as initial whitening (the first AddRoundKey step)

For remaining words (i = Nk to Nb*(Nr+1) –1) {

if i is not a multiple of Nk
$w_i = w_{i-1} \oplus w_{i-Nk}$

if i is a multiple of Nk and Nk < 8
w_i = (S-Box applied to a rotation of w_{i-1}) $\oplus$ w_{i-Nk} $\oplus$ round constant

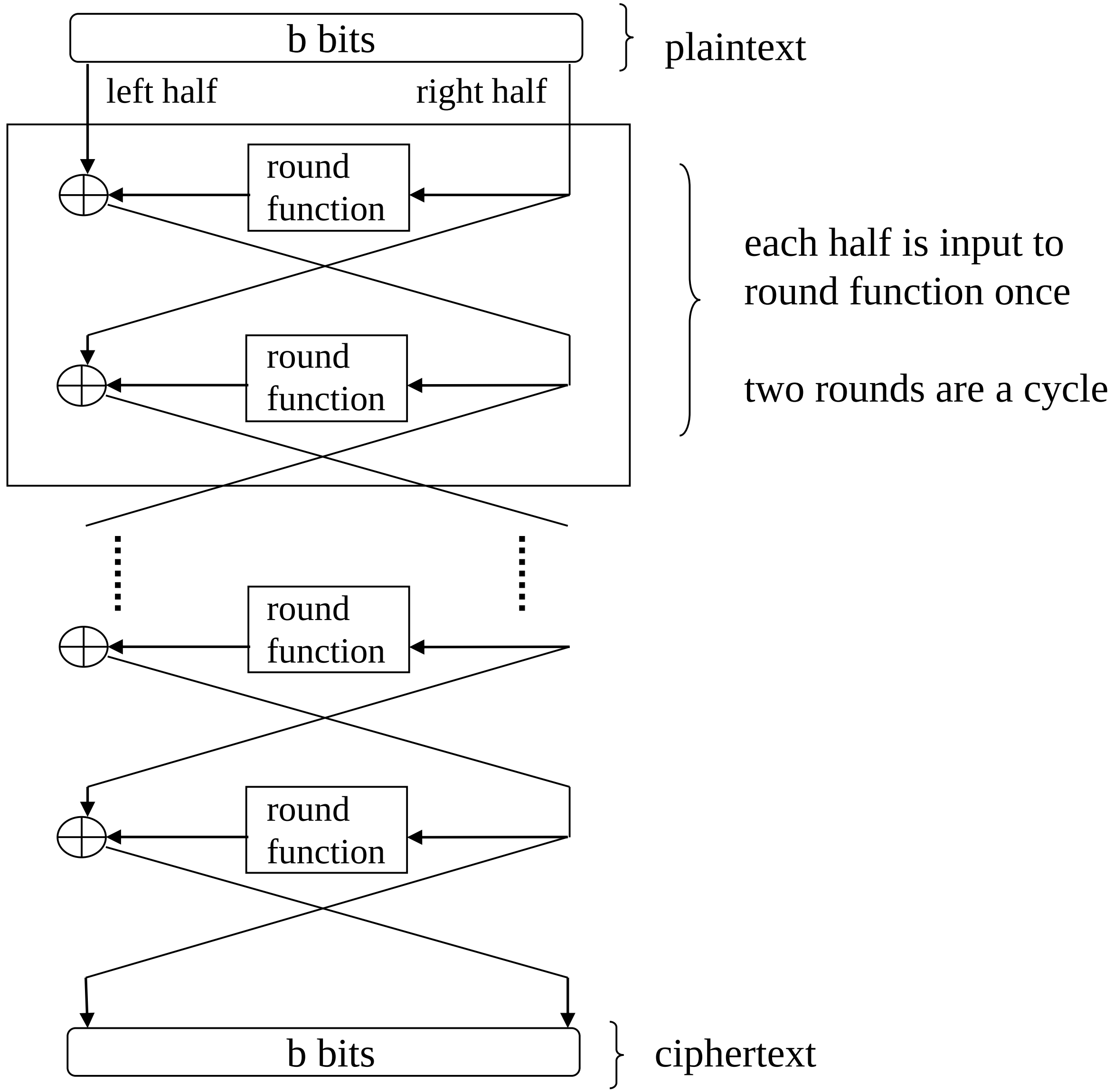
if Nk = 8 and i mod Nk = 4
w_i = (S-Box applied to w_{i-1}) $\oplus$ w_{i-Nk}
}

S-Box and rotations are applied at the byte level.

Loop 40 times for 128-bit key, 128-bit block

Most expanded key words are $\oplus$ of two previous words

(Balanced) Feistel Network



Note: unbalanced =
b bits divided into
two unequal portions

Feistel Network

Advantages:

- Run network in reverse to decrypt
  - Round function does not have to be invertible
  - Implementation benefit – same code/hardware used for encryption and decryption
- If the round function is pseudorandom permutation (theoretical concept), provable properties about 3 and 4 rounds

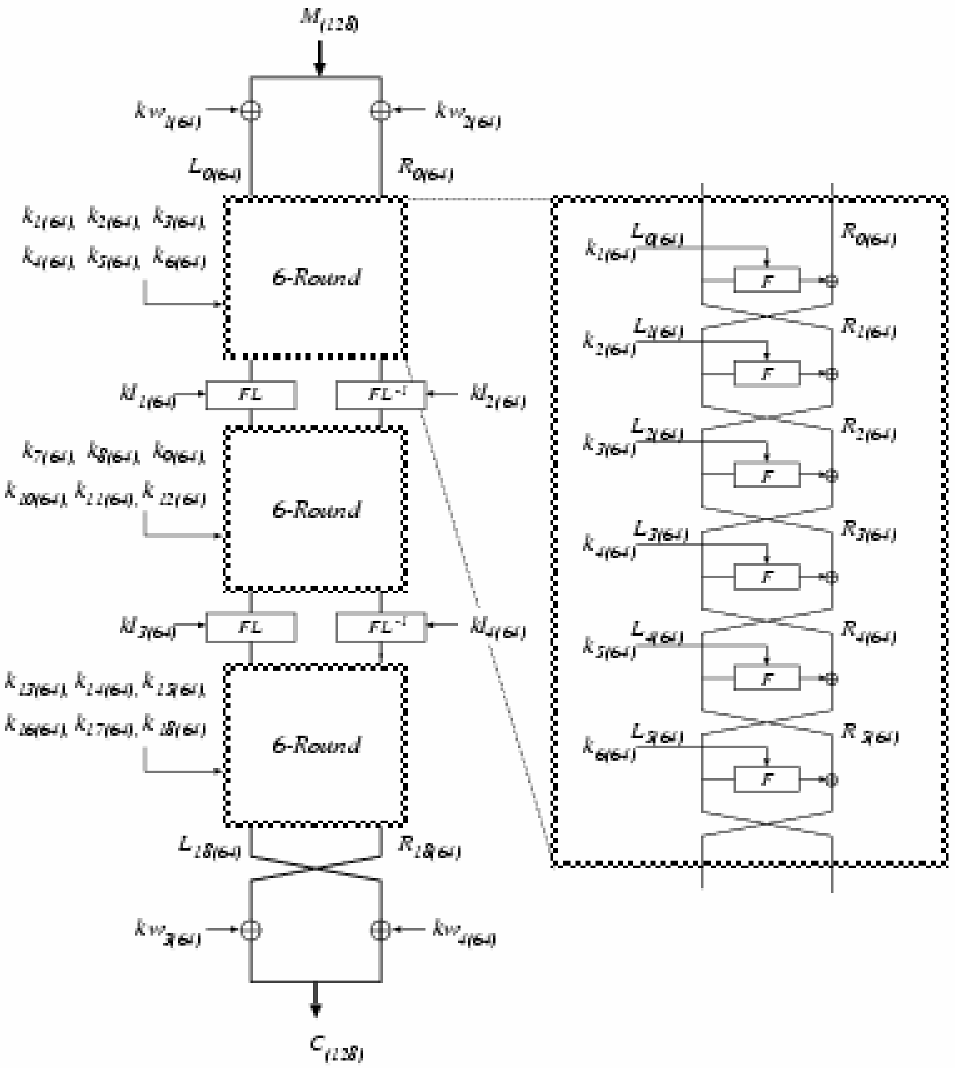
Disadvantages:

- Diffusion can be slow: ½ of bits have no impact in first application of the round function
- One round differential characteristic with probability of 1

PRPs, SPRPs from Feistel

- Round functions independently and randomly chosen PRPs,
  - r rounds and n bit input to round function, randomly select “tables” representing round functions
  - First selection from $2^n!$ tables, then from $2^n! -1$, $2^n!-2$, … $2^n!-r+1$ tables
- 3 round Feistel network is PRP
- 4 round Feistel network is a SPRP
  - Luby-Rackoff,
  - Naor-Reingold

Camellia 128-bit Key and Block



Camellia F Function

F(x,k) = P(S(x ⊕ k)), where S is a S-Box on 8-bytes. P is a function that XORs bytes of its 8-byte input to form an 8-byte output.

P function:

Output Byte : Input Bytes XORed

1 : 1,3,4,6,7,8

2 : 1,2,4,5,7,8

3 : 1,2,3,5,6,8

4 : 2,3,4,5,6,7

5 : 1,2,6,7,8

6 : 2,3,5,7,8

7 : 3,4,5,6,8

8 : 1,4,5,6,7

diffusion

Byte 1: 1,2,5,8

Byte 2: 2,3,4,5,6

Byte 3: 1,3,4,6,7

Byte 4: 1,2,4,7,8

Byte 5: 2,3,4,6,7,8

Byte 6: 1,3,4,5,7,8

Byte 7: 1,2,4,5,6,8

Byte 8: 1,2,3,5,6,7

Camellia F Function

- The substitution performed by S is done by viewing the data as 8 bytes and using one of four S-Boxes, (S1, S2, S3, S4), on each byte.
  - Bytes 1 and 8 have S1 applied
  - Bytes 2 and 5 have S2 applied
  - Bytes 3 and 6 have S3 applied
  - Bytes 4 and 7 have S4 applied
- One table, S represents S1,S2,S3,S4
- Create S1,S2,S3,S4 as follows:

For i = 0 to 255:

S1[i] = S[i]
S2[i] = (S[i] >> 7 ⊕ S[i] << 1) & 0xff
S3[i] = (S[i] >> 1 ⊕ S[i] << 7) & 0xff
S4[i] = S[((i) << 1 ⊕ i >> 7) & 0xff

Camellia F Function

- P function: diffusion amongst bytes
- S-box: Allows for time/memory tradeoff in implementations
  - Can store four tables S1,S2,S3,S4
  - Can store only S and compute values

Camellia FL Function

- The FL function takes a 64-bit input and 64 expanded key bits.
- Let X_L and X_R denote the left and right halves of the input, respectively
- Let Y_L and Y_R denote the left and right halves of the output, respectively.
- Let kl_L and kl_R denote the left and right halves of the 64 key bits.
- FL is defined as:

  $Y_R = ((X_L \cap kl_L) \lll 1) \oplus X_R$
  $Y_L = (Y_R \cup kl_R) \oplus X_L$

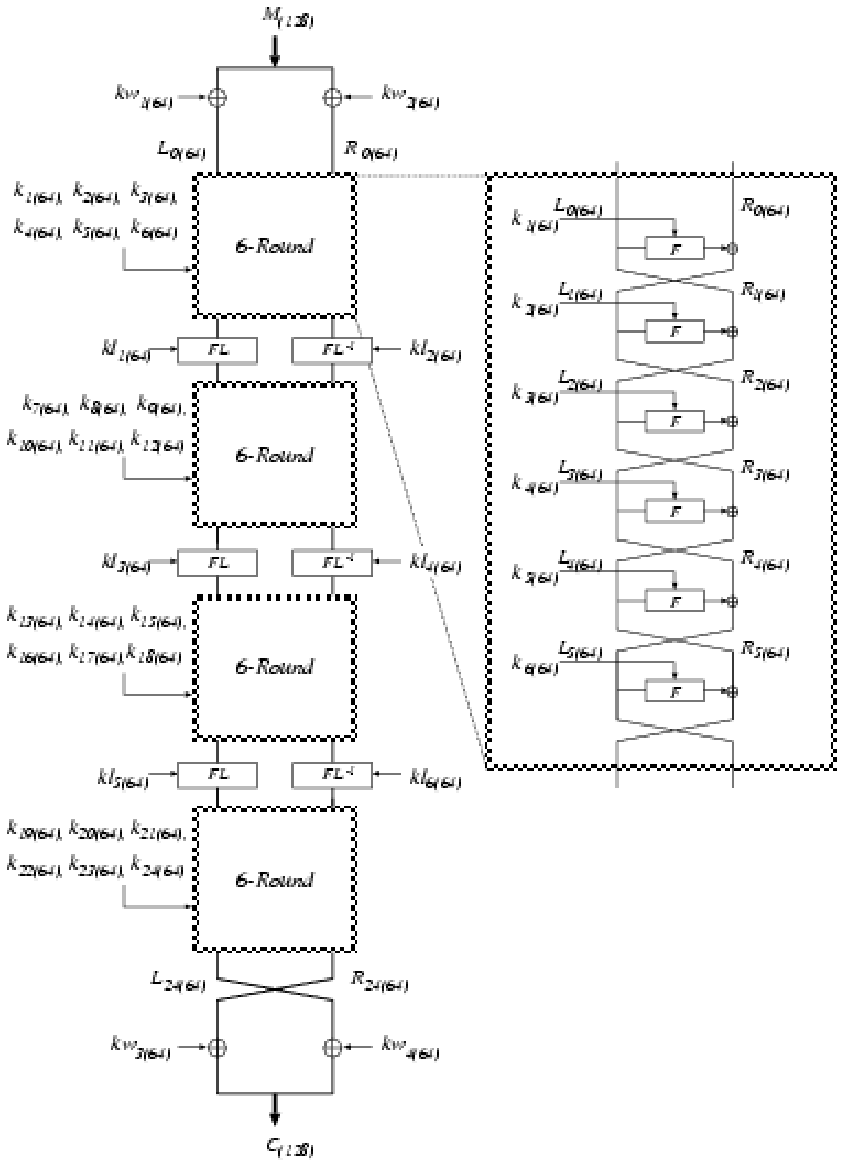
incorporating key bits

- FL^{-1} is:

  $X_L = (Y_R \cup kl_R) \oplus Y_L$
  $X_R = ((X_L \cap kl_L) \lll 1) \oplus Y_R$

$\cup$ is bitwise OR $\cap$ is bitwise AND   <<< is left rotation

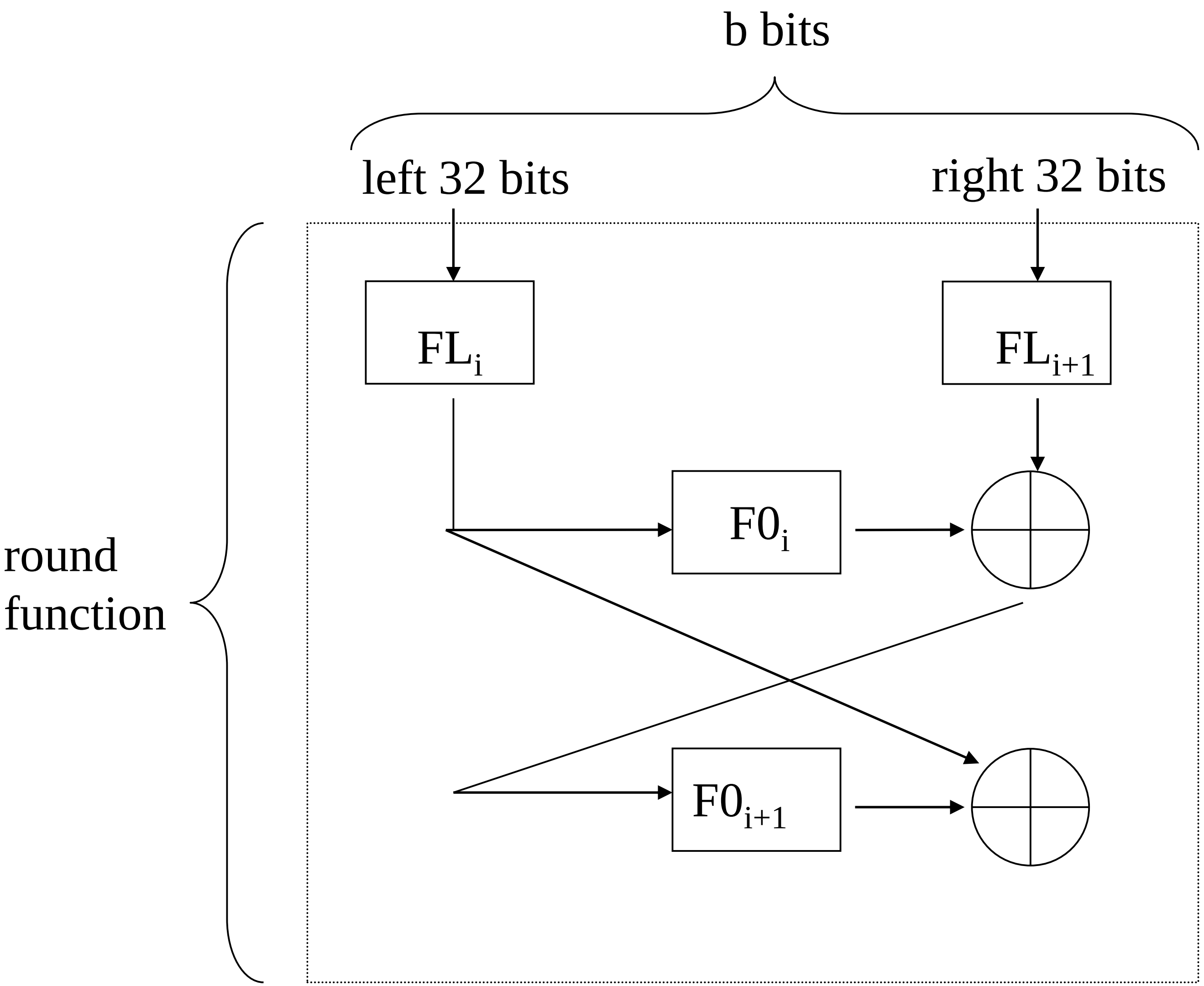
Camellia 192,256-bit Keys



Camellia Key Schedule

- Let K be the key.
- Applies rounds of Camellia with constants for the round keys to K.
- XORs round's output with the K then applies additional rounds.
- Let KA be the final output of the rounds.
- Each round key is part of KA or K rotated.
  - KA, K values used in multiple rounds
  - For example:
    - initial whitening uses K
    - 9^{th} application of F uses the left half of KA rotated 45 bits to the left.

MISTY1



b bits
left 32 bits
right 32 bits
round
function
FLi
FLi+1
F0i
F0i+1

MISTY1 FL Function

The FL function takes a 32-bit input and 32 bits of expanded key bits. Let X_L and X_R denote the left and right halves of the input, respectively. Let KL_{iL} and KL_{iR} denote the left and right halves of the 32 key bits. The index i refers to the component.

$Y_R = (X_L \cap KL_{iL}) \oplus X_R$
$Y_L = (Y_R \cup KL_{iR}) \oplus X_L$
The 32 bit output is $Y_L \,||\, Y_R$

The inverse of FL is used in decryption and is defined by
$X_L = (Y\_{}_R \cup KL_{iR}) \oplus Y_L$
$X_R = (X\_L \cap KL_{iL}) \oplus Y_R$
The 32 bit output is $X_L \,||\, X_R$

Combines key and data bits; some diffusion between two 16-bit data segments

MISTY F0 Function

- A 32-bit input, a 64-bit key and 48-bit key (from expanded key bits).
- Let L_0 and R_0 denote the left and right halves of the input
- Let KO_i be the 64-bit key and KI_i be the 48 bit key.
- KO_i and KI_j are each divided into 16 bit segments. KO_{ij} and KI_{ij} denote the j^{th} 16 bit segment of KO_i and KI_i, respectively.

For (j=1; j $\leq$ 3; ++j) {

    $R_j = FI((L_{j-1} \oplus KO_{ij}), KI_{ij}) \oplus R_{j-1}$

    $L_j = R_{j-1}$

}

The value $(L_3 \oplus KO_{i4}) || R_3$ is returned

Combines key and data bits; some diffusion between two 16-bit data segments

MISTY FI Function

- 16 bit input, X_j, and a 16 bit key, KI_{ij}.
- Let $X_j = L_{0(9)} || R_{0(7)}$ (x) indicates x bits
- Let $KI_{ij} = KI_{ijL(7)} || KI_{ijR(9)}$
- S7 and S9: two S-Boxes mapping 7 and 9-bit inputs to 7 and 9-bit outputs.
  - Refer to the paper on MISTY1 for the table values
  - S-Boxes: each output bit corresponds to the multiplication and XOR of a subset of input bits.
- ZE(x): 7-bit input, x, and adds two 0's as the most significant bits.
- TR(x): 9-bit input, x, and discards the two most significant bits.

MISTY FI Function

$L_{1(7)} = R_{0(7)}$
$R_{1(9)} = S9(L_{0(9)}) \oplus ZE(R_{0(7)})$
$L_{2(9)} = R_{1(9)} \oplus KI_{ijR(9)}$
$R_{2(7)} = S7(L_{1(7)}) \oplus TR(R_{1(9)}) \oplus KI_{ijL(7)}$
$L_{3(7)} = R_{2(7)}$
$R_{3(9)} = S9(L_{2(9)}) \oplus ZE(R_{2(7)})$
FI returns $L_{3(7)} \| R_{3(9)}$

Combines key and data bits;
“shifts” bits so 16-bit halves used in F,
F0 functions are altered – helps diffusion
between two 16-bit data segments

MISTY1 Key Schedule

One 128-bit key is divided into eight 16 bit values.

Let K_i be the i^{th} 16 bit portion.

Note: i = i-8 for i > 8

Create eight 16 bit values using the K_i's and the FI function:

$K'_i = FI(K_i, K_{i+1})$

$KO_{i1} = K_i$

$KO_{i2} = K_{i+2}$

$KO_{i3} = K_{i+7}$

$KO_{i4} = K_{i+4}$

$KI_{i1} = K'_{i+5}$

$KI_{i2} = K'_{i+1}$

$KI_{i3} = K'_{i+3}$

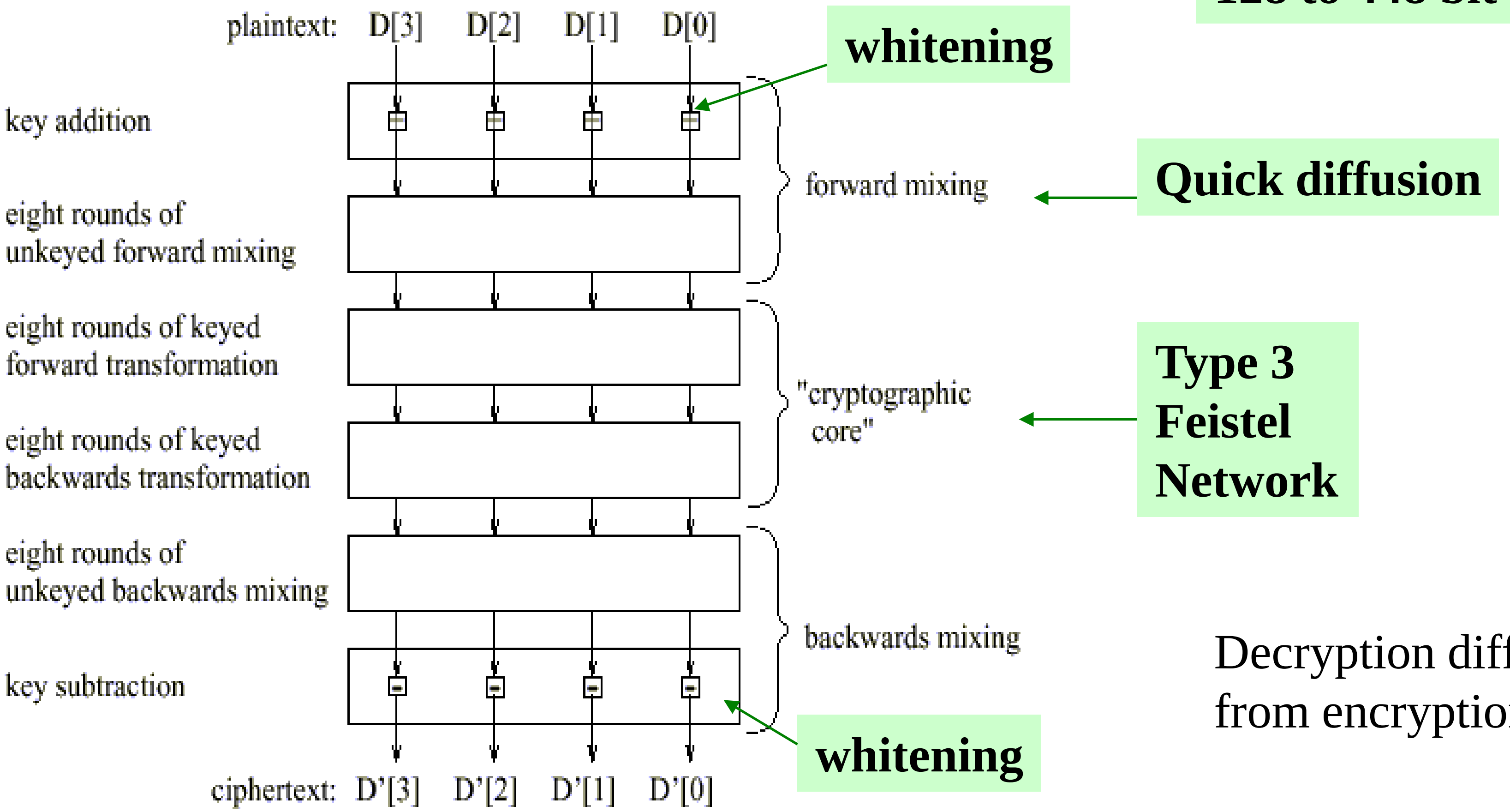
$KL_{iL} = K'_{(i+1)/2}$ when i is odd and $K'_{i/2+2}$ when i is even

$KL_{iR} = K'_{(i+1)/2+6}$ when i is odd and $K_{i/2+4}$ when i is even

MARS

128 bit data block

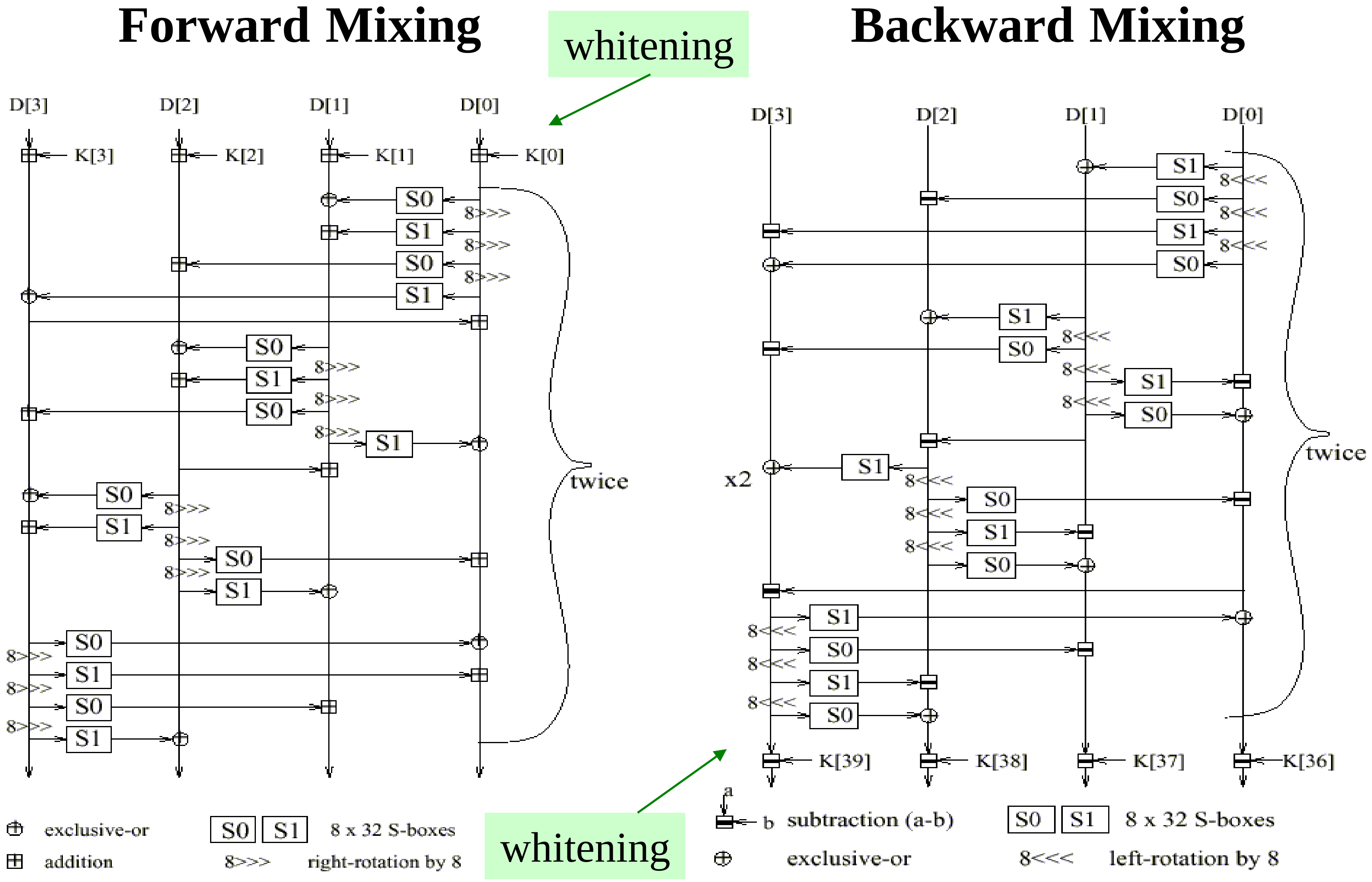
3 main stages
128 to 448 bit keys



Decryption differs from encryption.

Images downloaded from http://islab.oregonstate.edu/koc/ece575/00Project/Galli/MARSReport.html, original source unknown.

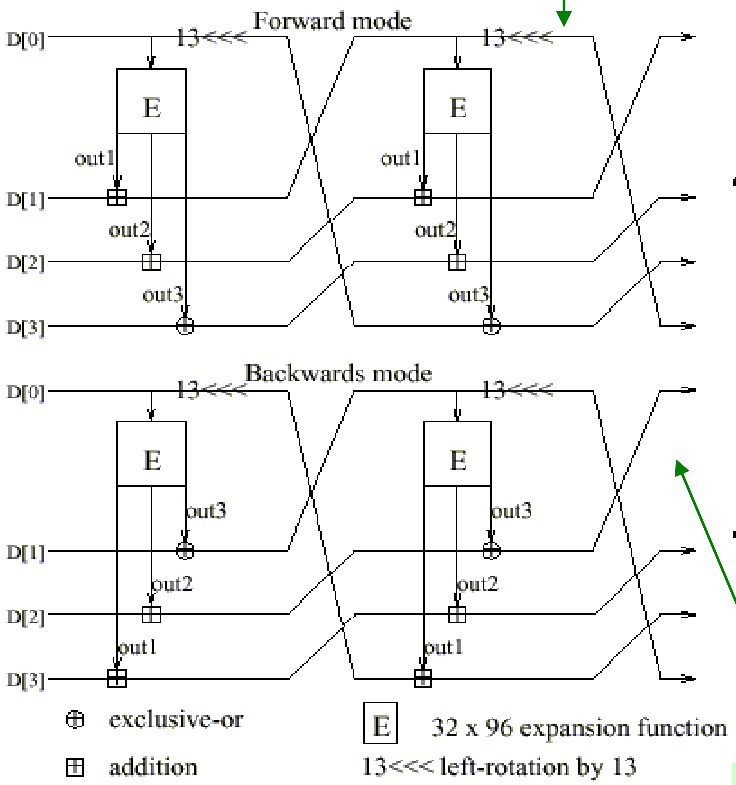
MARS - Details



MARS - Details

Core

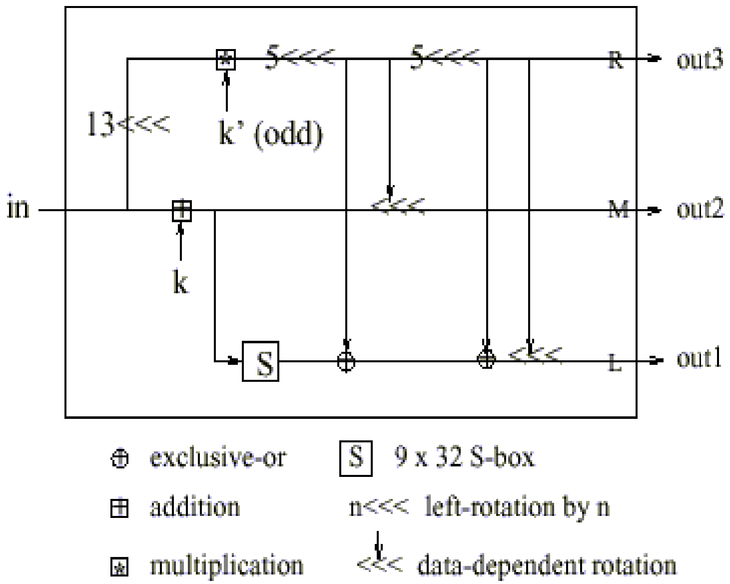
Odd bit rotations



16 rounds: 8 each of
forward and backward mode.

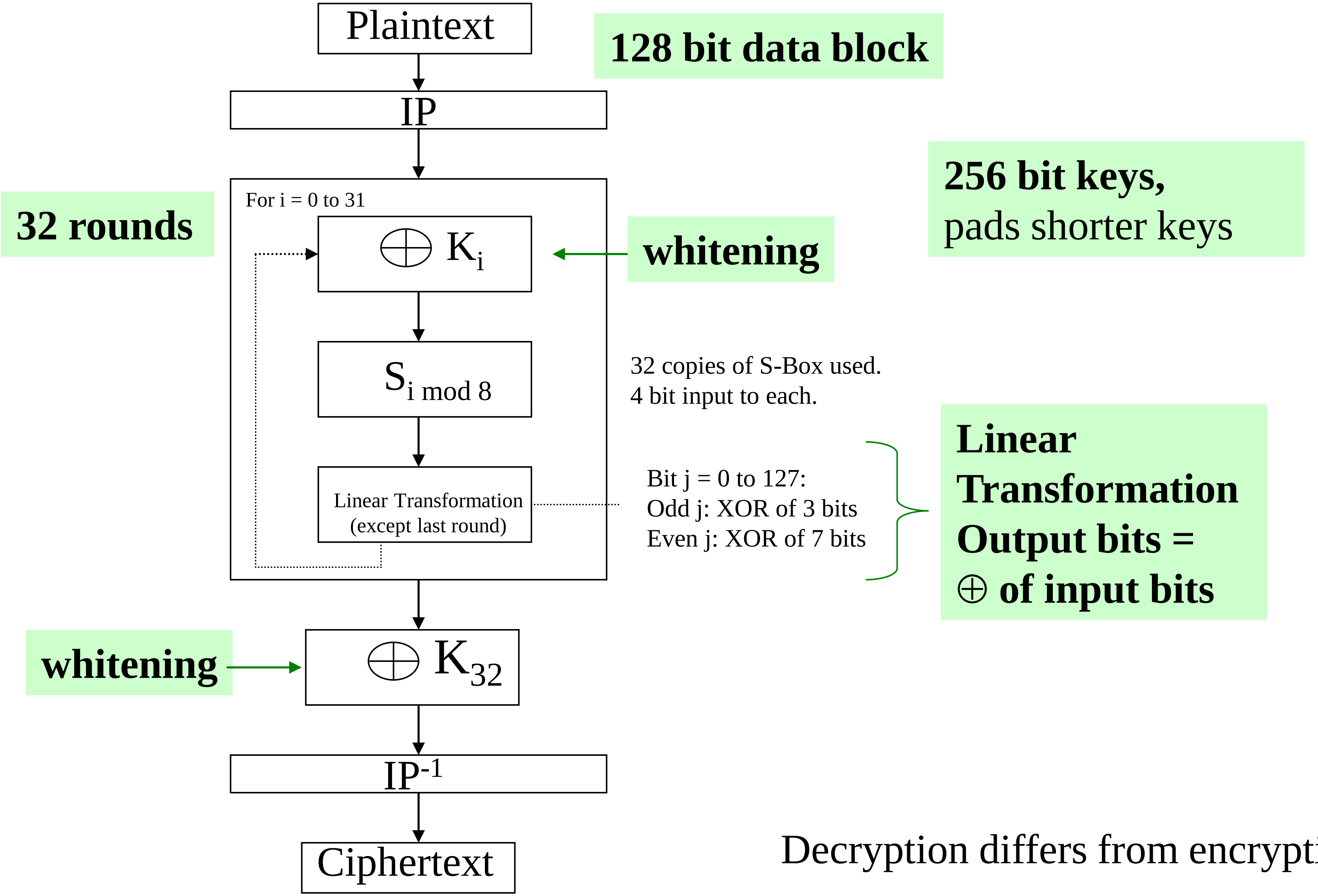
Alternate blocks entering E.

E Function



Data dependent rotation
Odd bit rotations
Multiplication
S-Box, addition

Serpent



Decryption differs from encryption

Twofish

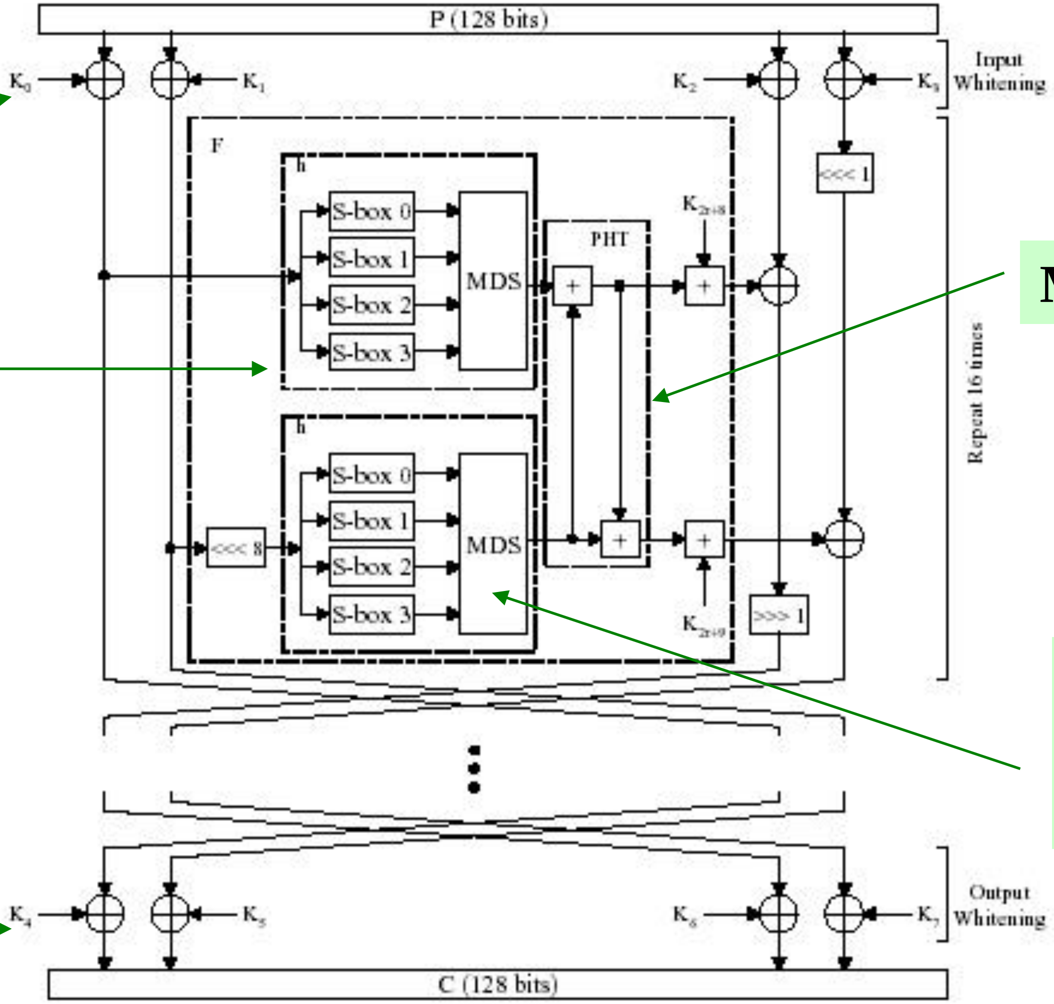
128 bit data

whitening

4 key dependent S-Boxes

16 rounds
(Not Feistel –
1 bit rotations.)

whitening



128,192,256 bit keys
pads shorter keys

Mix bits

Maximize difference in outputs

Decryption differs from encryption.

RC6

break input into 4 words

Consists of ⊕, +, *

```
RC6_encrypt(A,B,C,D) {
    B = B + S[0];
    D = D + S[1];
    for (i=0; i < r; ++i) {
        t = (B*(2B+1)) <<< log2(w);
        u = (D*(2D+1)) <<< log2(w);
        A = ((A ⊕ t) <<< u) + S[2i];
        C = ((C ⊕ u) <<< t) + S[2i+1];
        (A,B,C,D) = (B,C,D,A);
    }
    A = A + S[2r+2];
    C = C +  S[2r+3];
    return (A,B,C,D);
}
```

whitening

modify half of data,
⊕ with other half, shift
whitening
swap "halves"

whitening

r = # of rounds
S = expanded key (2r+3 words)
w = word size
* = multiplication mod 2^w
+ = addition mod 2^w
<<< = left rotate

Decryption use: >>>, -

RC6 Key Schedule

Key schedule for RC6-$w/r/b$

Input: User-supplied b byte key preloaded into the c-word array $L[0, \ldots, c-1]$
Number r of rounds

Output: w-bit round keys $S[0, \ldots, 2r+3]$

Procedure: $S[0] = P_w$

for $i = 1$ **to** $2r+3$ **do**
$\quad S[i] = S[i-1] + Q_w$

$A = B = i = j = 0$

$v = 3 \times \max\{c, 2r+4\}$
for $s = 1$ **to** v **do**
{
$\quad A = S[i] = (S[i] + A + B) \lll 3$
$\quad B = L[j] = (L[j] + A + B) \lll (A + B)$
$\quad i = (i+1) \bmod (2r+4)$
$\quad j = (j+1) \bmod c$
}

P_{32} = B7E15163 Q_{32} = 9E3779B9
Constants really are arbitrary and can be changed.

RC6

Decryption with RC6-$w/r/b$

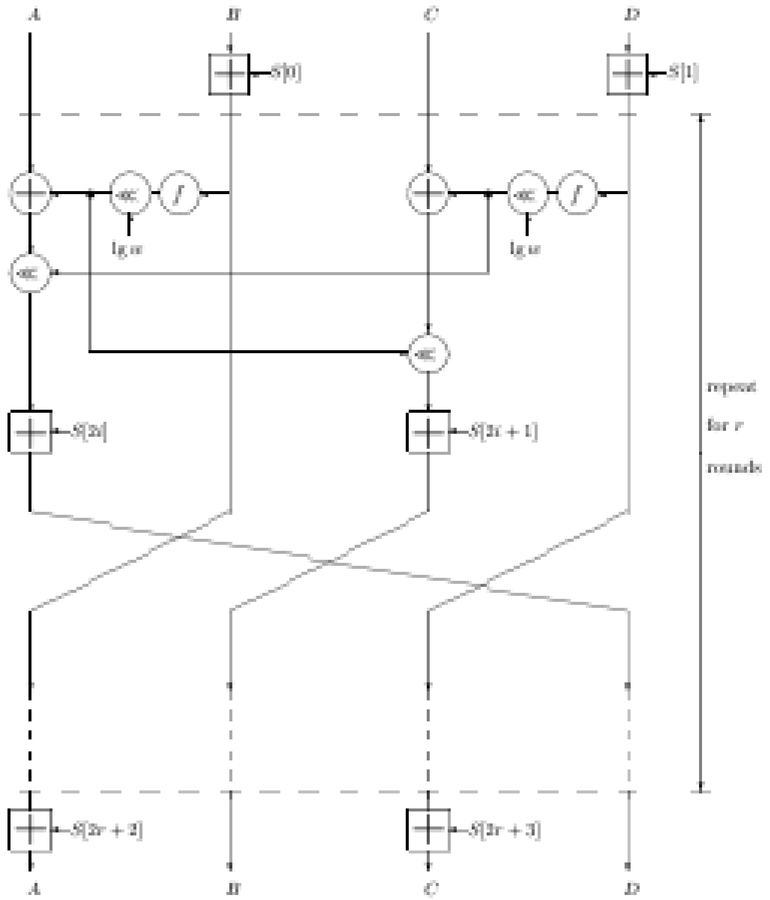
Input: Ciphertext stored in four w-bit input registers A, B, C, D
Number r of rounds
w-bit round keys $S[0, \ldots, 2r+3]$

Output: Plaintext stored in A, B, C, D

Procedure:

$C = C - S[2r+3]$
$A = A - S[2r+2]$
for $i = r$ **downto** 1 **do**
{
$(A, B, C, D) = (D, A, B, C)$
$u = (D \times (2D+1)) \lll \lg w$
$t = (B \times (2B+1)) \lll \lg w$
$C = ((C - S[2i+1]) \ggg t) \oplus u$
$A = ((A - S[2i]) \ggg u) \oplus t$
}
$D = D - S[1]$
$B = B - S[0]$

RC6 Encryption



Key Schedules

- Ideal key schedule
  - pseudorandom expanded key bits
  - efficient
- Existing key schedules
  - Unique per block cipher
  - Lack of randomness/independence
  - Contributes to attacks – if find few expanded key bits can plug into key schedule
  - Design for efficiency
- Suggestion: Use a generic key schedule
  - Generate as many expanded key bits as needed
  - Single implementation
  - Increase randomness compared to existing key schedules

Key Schedules – Existing

- AES:
  - 11 128-bit strings created each as 4 32-bit words (11 whitening steps)
  - The 128-bit key is split into four 32-bit words. Additional 128-bit strings are formed by:
    - 1st word: a table lookup on a previous word then XOR it with a constant and a previous word.
    - 2nd to 4th words: XORing two previous words
- Camellia, MISTY1: expanded key bits used in multiple locations
- RC6: more complex relationship between expanded key bits

Example: Use of a Block Cipher to Create Random Bits

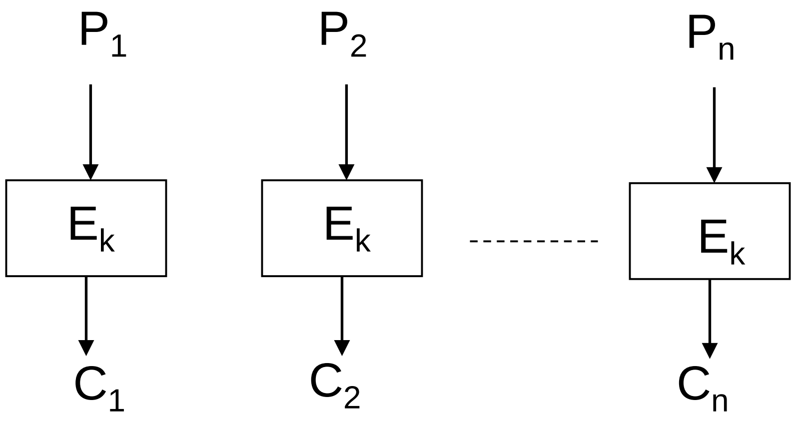
RSA SecurID®

- Provides a one time password
- Previous version used proprietary algorithm that was reversed engineered.
- Current version uses AES as a hash function
- Algorithm to handle timing issues

Agenda

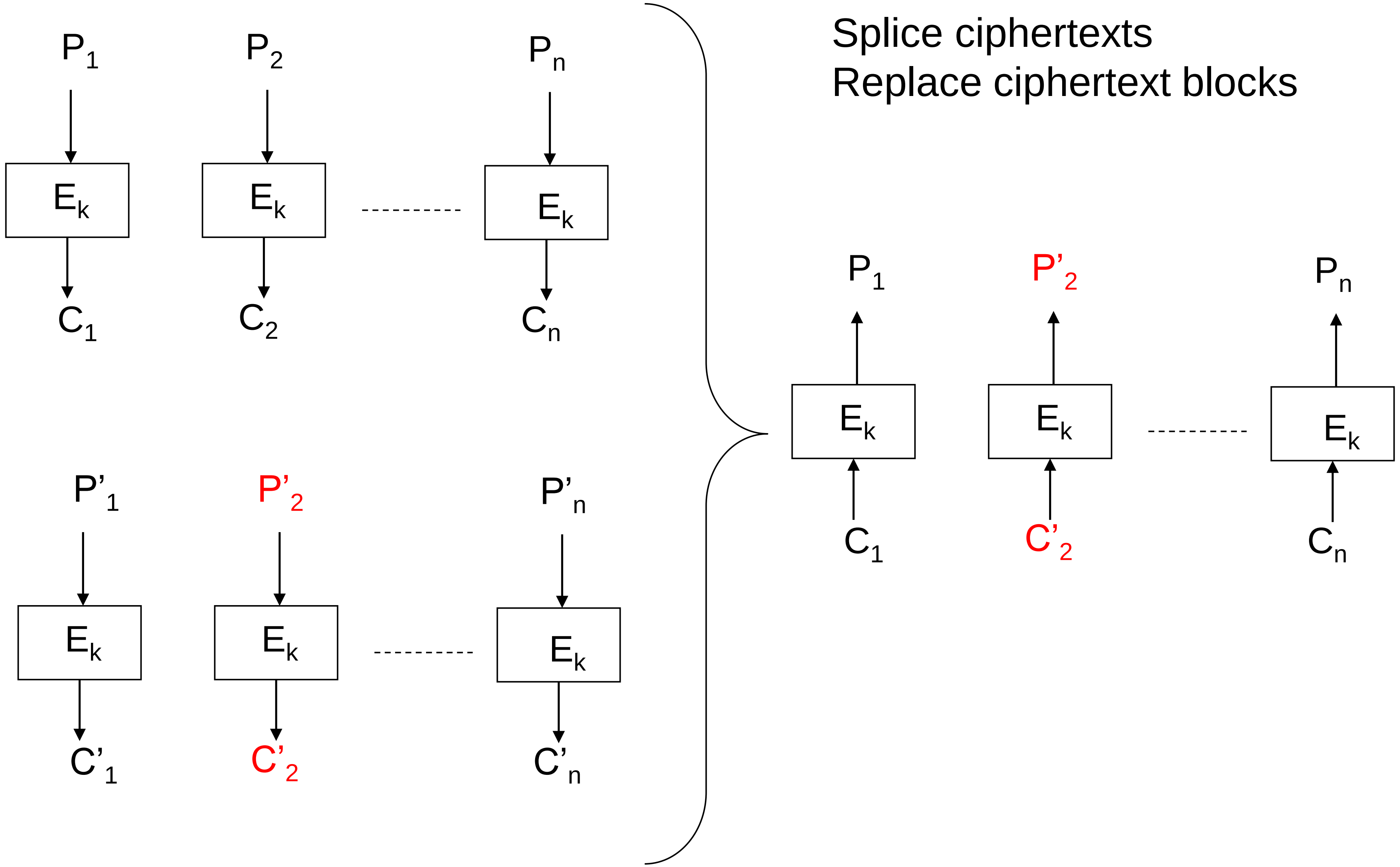
- Introduction
- Block Ciphers
  - Definition
  - Standards Competitions and Requirements
  - Common Building Blocks
  - Examples
  - **Modes of Encryption**

ECB Mode



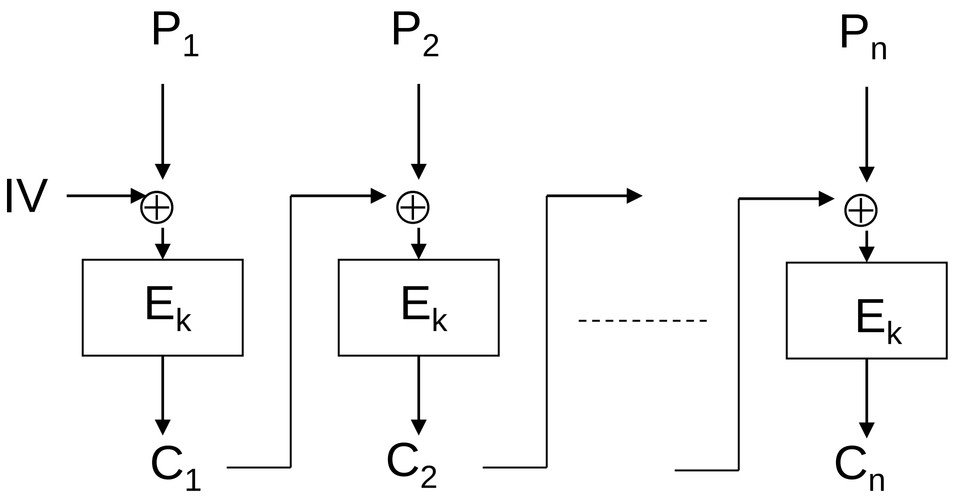
- **Identical plaintext blocks produce identical ciphertext block:** pattern detection
- Patterns not likely in normal text – newspaper, book – due to need to align on block boundary
- Patterns likely in structured text – log files

ECB Mode

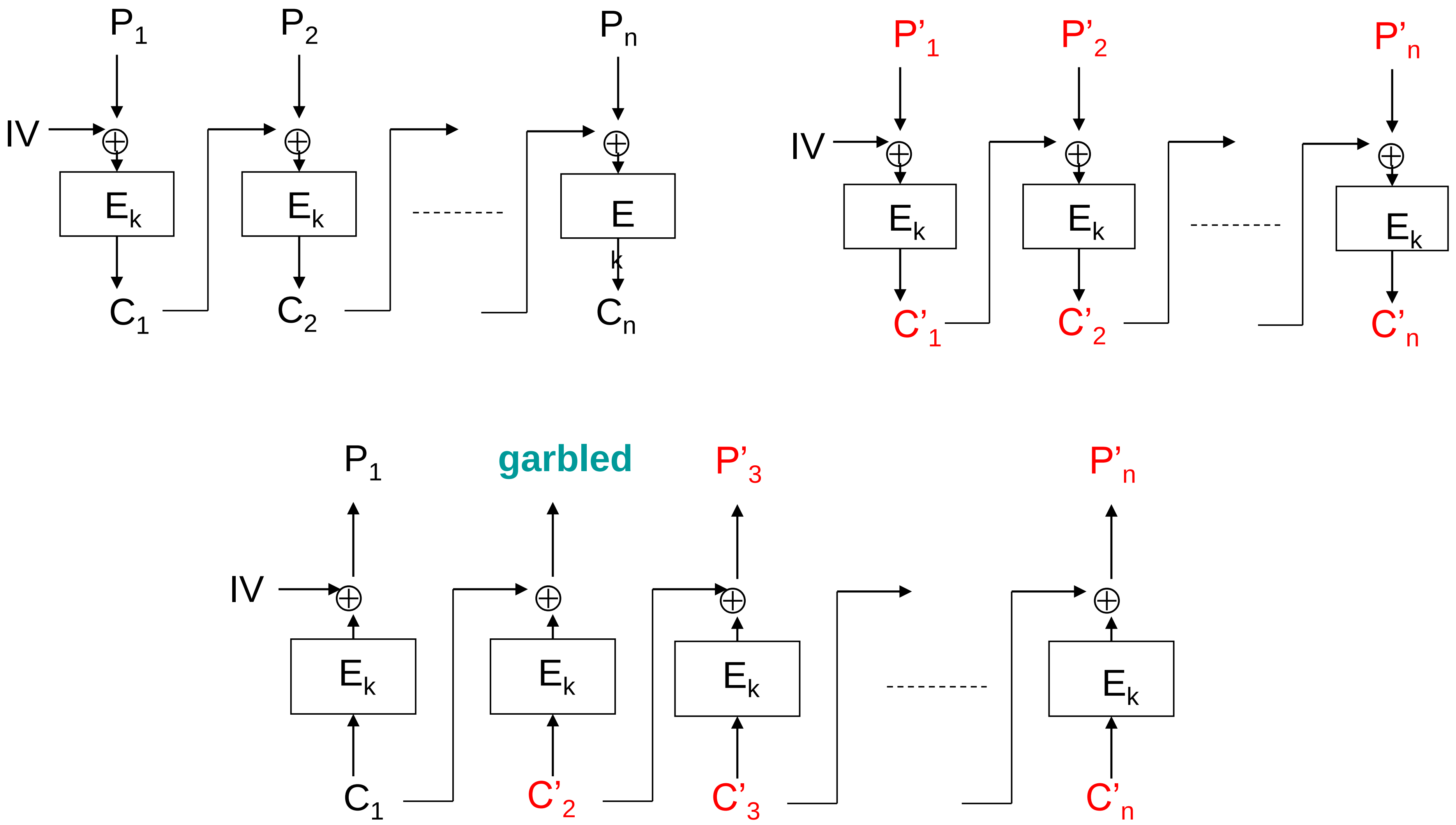


Splice ciphertexts
Replace ciphertext blocks

CBC Mode

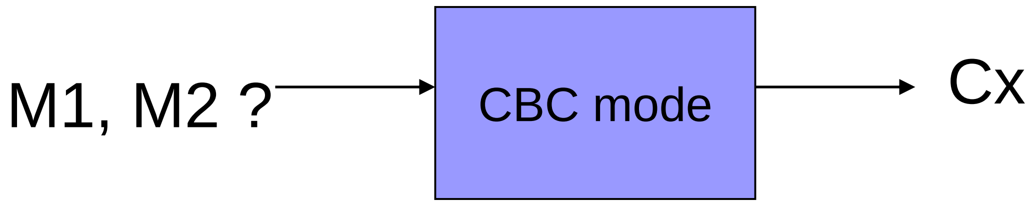


CBC Mode - Splicing



Blockwise Adaptive

- Consider a block cipher and CBC mode
- Environment where see ciphertext from plaintext block i before having to input plaintext block i+1
- M1,M2,M3 are three distinct 2b-bit plaintexts.
- Know one of M1 and M2 was encrypted. Ciphertext, Cx



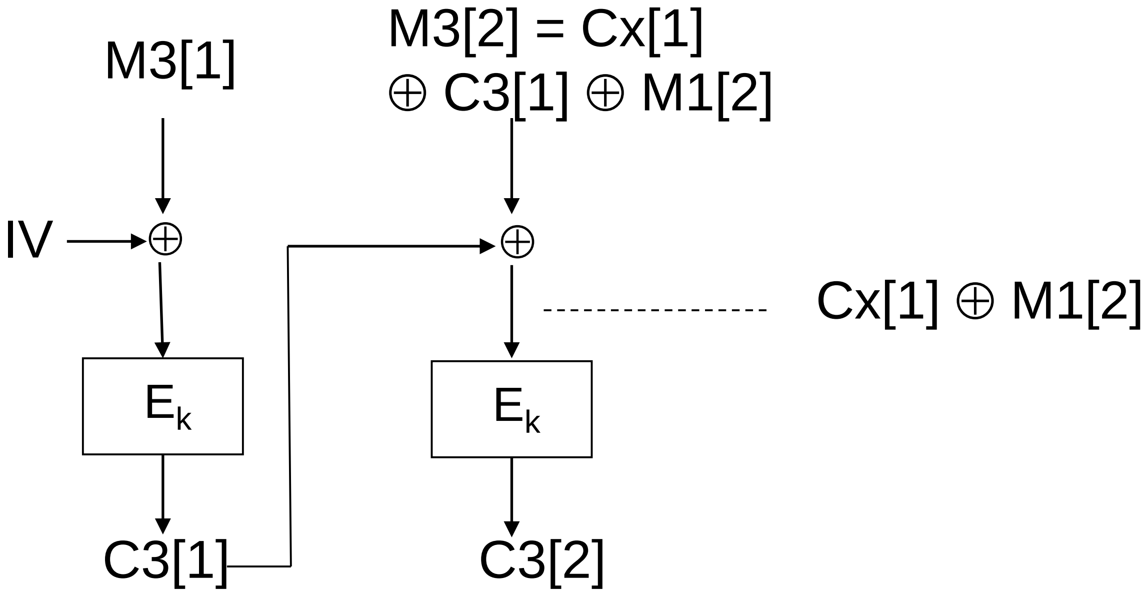
- Can form M3 to determine if it is M1 or M2.

Blockwise Adaptive

- M3: for first block send an arbitrary b-bit bits, receive the ciphertext, C3[1]
- Generate the next b bits of M3 by XORing the first block from Cx, C3[1] and M1[2]

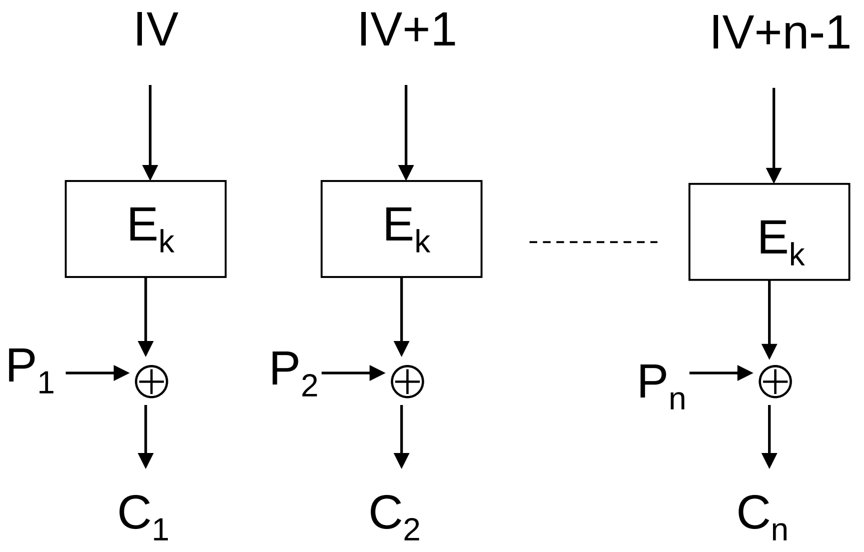
Notation: X[i] = i^{th} block of X

Blockwise Adaptive



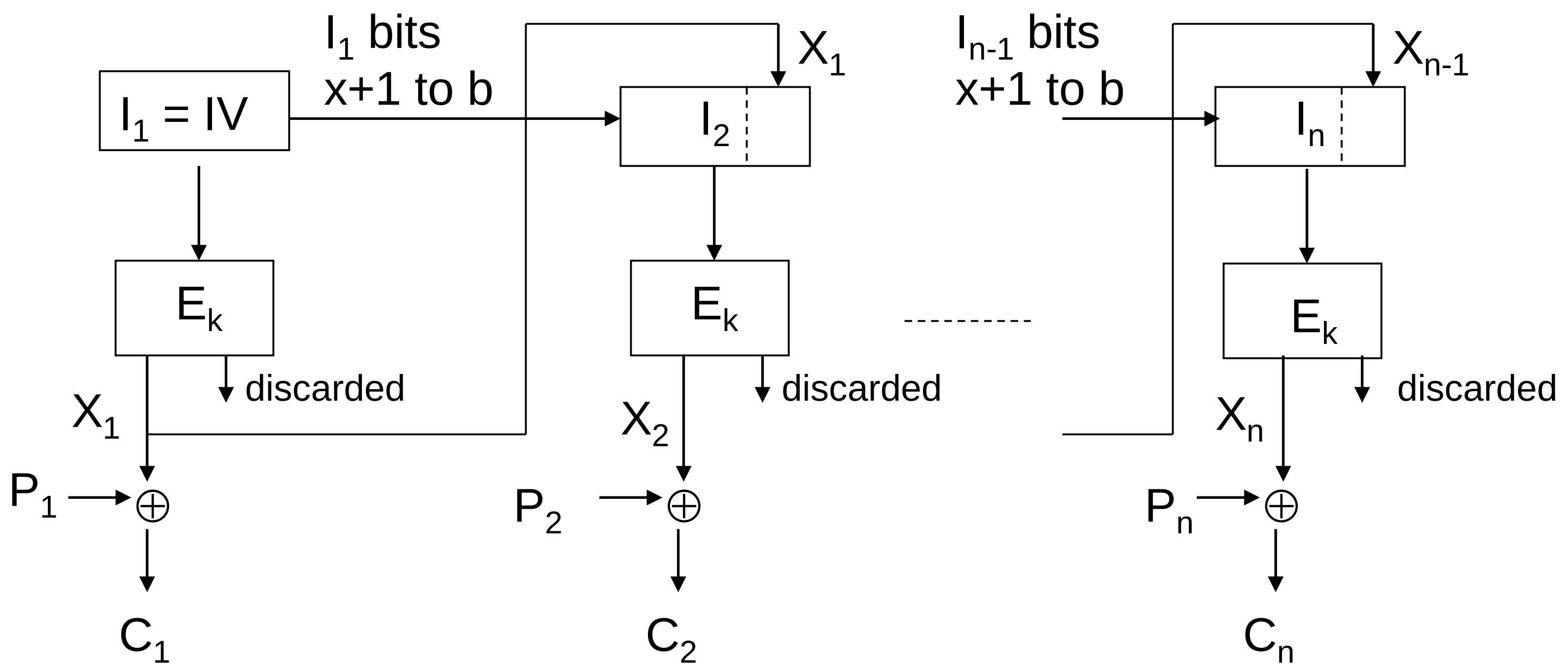
C3[2] = Cx[2] if Cx is the encryption of M1
C3[2] ≠ Cx[2] if Cx is the encryption of M2.

CTR Mode



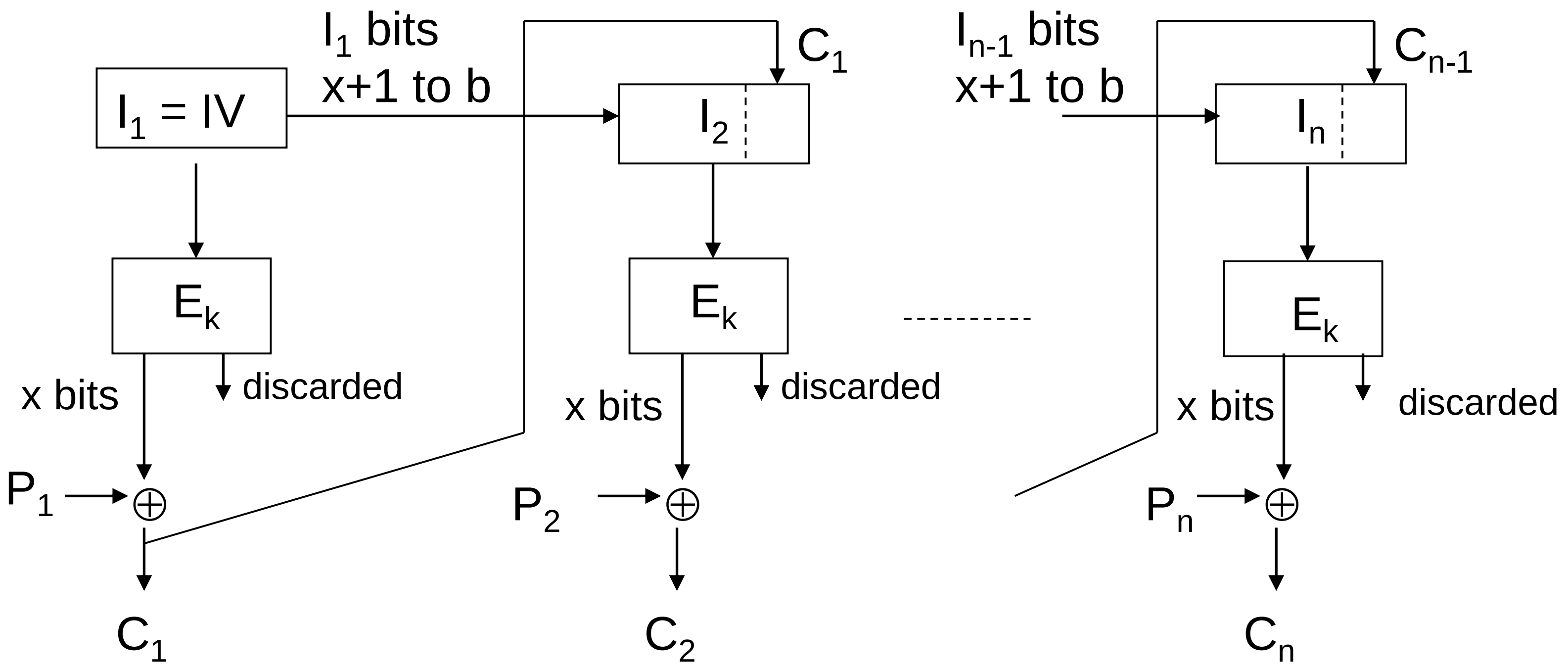
Creates key stream and XORs with plaintext
Need to avoid reusing key and IV+i value combination

OFB Mode



X_j = leftmost x bits of the b bit output from the cipher
P_j is x bits
$I_j = I_{j-1}$ bits x+1 to b || X_{j-1}

CFB Mode



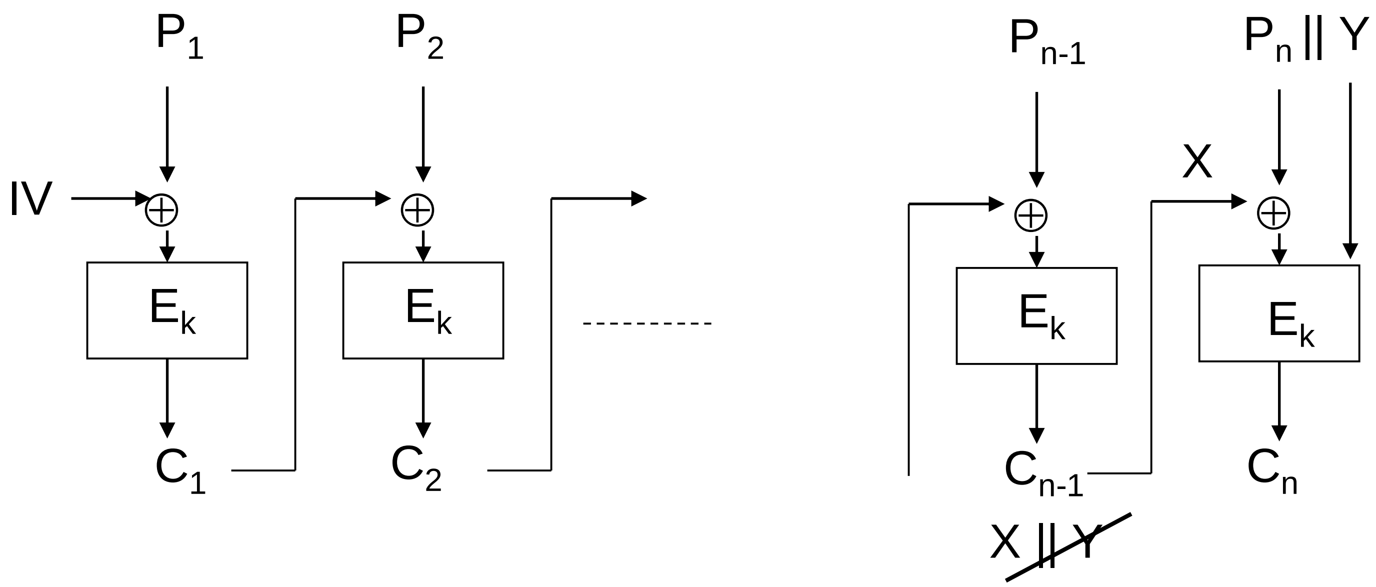
Cipher outputs b bits, the rightmost b-x bits are discarded.

P_j is x bits

$I_j = I_{j-1}$ bits x+1 to b || C_{j-1}

Ciphertext Stealing

Example using CBC mode



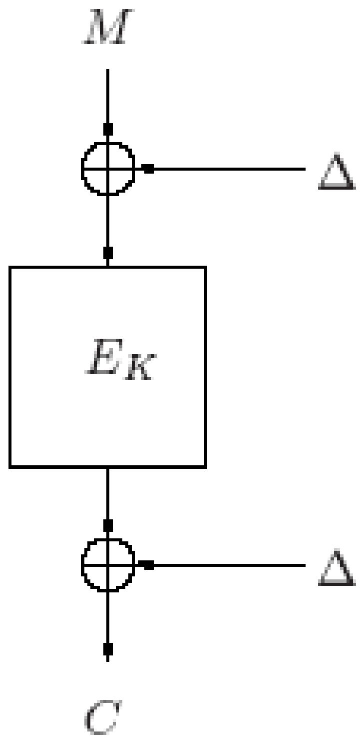
Length preserving

Use bits from next to last block of ciphertext to pad last plaintext block

Disk Encryption

- Modes seen so far process block, move on
  - no backward diffusion
  - can easily distinguish output from random by encrypting a few plaintexts
  - ex. If P1 = P2 in first x blocks, encrypt with same key then first x blocks of ciphertext are identical
- Tweakable modes:
  - narrow-block encryption modes: LRW, XEX, XTS
  - wide-block encryption: CMC, EME
  - designed to securely encrypt sectors of a disk

XEX



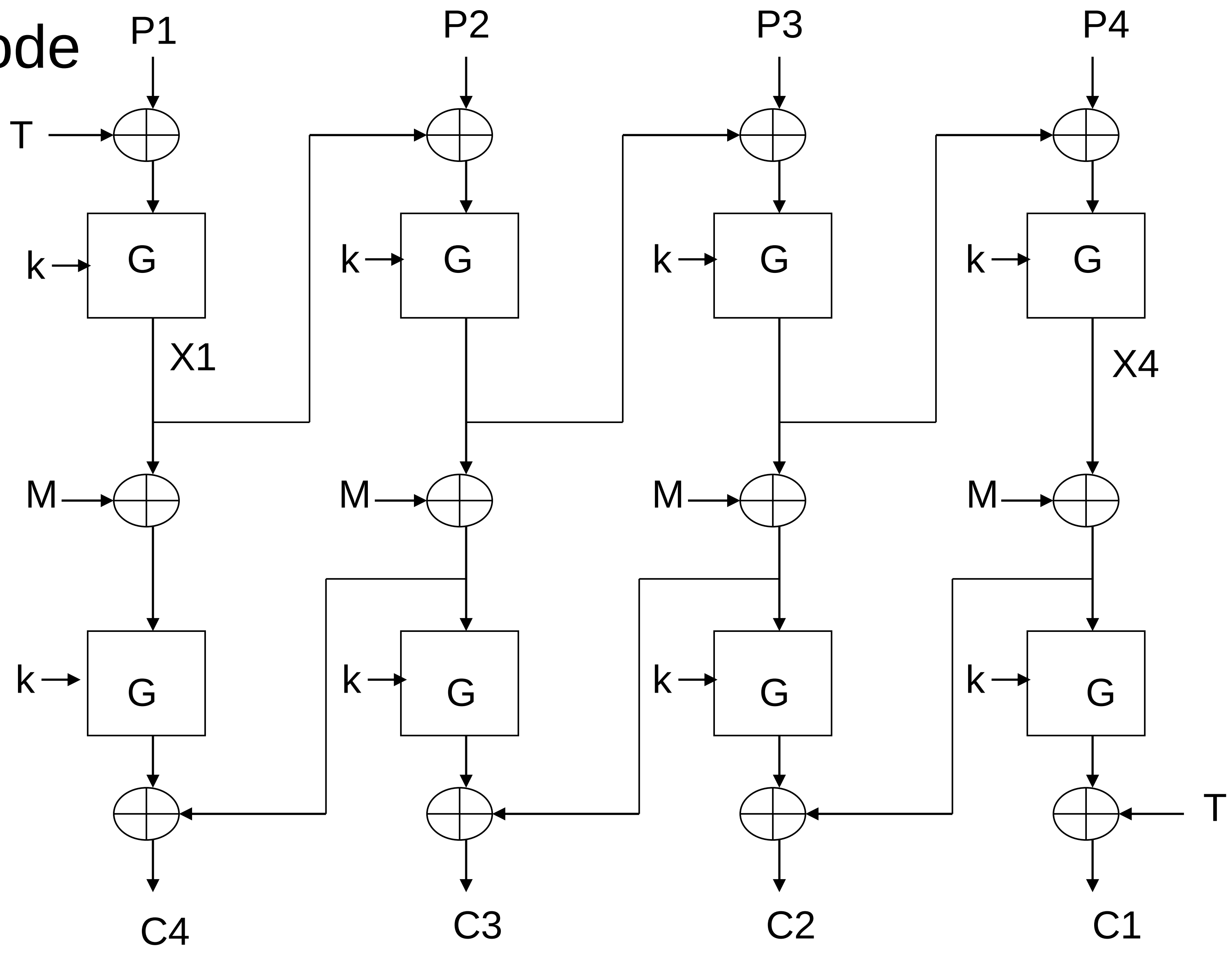
$$\Delta = \alpha_1^{i_1}\alpha_2^{i_2}\cdots\alpha_k^{i_k}\ \mathsf{N}$$

$$\mathsf{N} = E_K(N)$$

Disk encryption:
N = sector index
I = $i_1 i_2 \ldots i_k$ = block index

XTS is XEX-based Tweaked CodeBook mode (TCB)
with CipherText Stealing (CTS)

CMC Mode



T = G(tweak) using key k, T = 0 if no tweak
M = 2(X1 ⊕ X4)

Halevi and Rogaway

EME mode

- EME: ECB-mask-ECB
- Mask is different from that of CMC mode
- CMC creates PRP/SPRP in theory on m blocks
- EME does not
  - Flaw – authors stated in CMC paper not fixable
- Patented
- Used for disk encryption in practice