Cryptography and Network Security

Contemporary Simmetric Ciphers

Outline

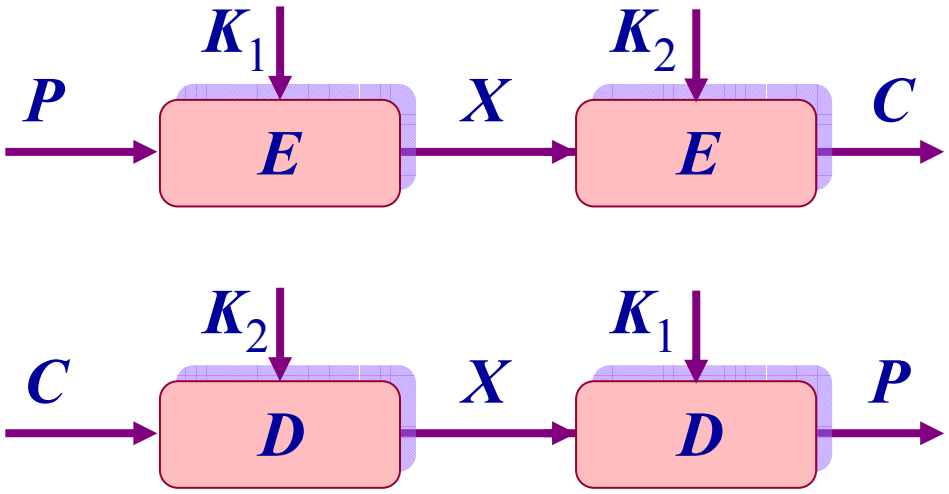
- Double-DES
- Triple-DES
- Modi di funzionamento
  - ECB, CBC, CFB, OFB, CTR
- Stream cipher
- RC4
- A5

Introduzione

- DES suscettibile di attacco a forza bruta
- Serve quindi un cipher migliore
  - Cipher completamente nuovo (AES, etc)
  - Sistemi di encryption multipla con DES
    - Uso di componenti già disponibili

Double-DES

Forma di encryption multipla più semplice due stadii DES



$$C = E\left[K_2, E\left(K_1, P\right)\right]$$

$$P = D\left[K_1, D\left(K_2, C\right)\right]$$

Sembra di usare una chiave di 56 X 2 = 112 bits, ma è vero?

Supponiamo che $E\left[K_2, E\left(K_1, P\right)\right] = E\left[K_3, P\right]$

A prima vista questa relazione non vale

Double-DES (2)

Un'encryption DES è un mapping da un blocco di 64 bit ad un altro

Con una data chiave un blocco di 64 bit diviene un altro ben preciso

Vi sono 2^{64} ingressi possibili quindi vi sono $\left(2^{64}\right)! > 10^{10^{20}}$ mapping diversi

DES definisce un mapping per ogni chiave, quindi $2^{56} < 10^{17}$

Il DES con due chiavi diverse può produrre uno dei mappings
non prodotti da una singola applicazione del DES

Il Double-DES è un miglioramento rispetto al DES

Meet-in-the-Middle Attack

È un tipo di attacco che riduce il tempo necessario per rompere il cipher

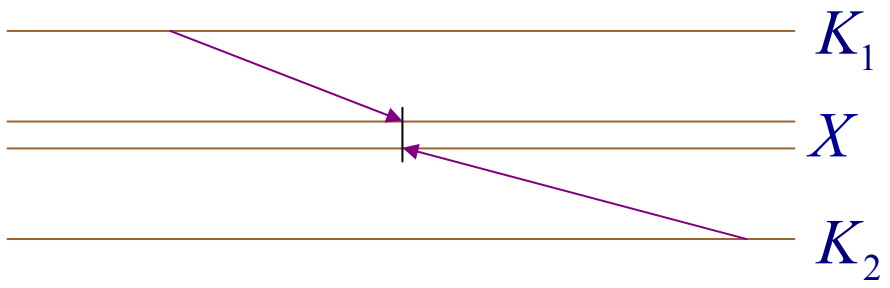
Come al solito, l'attacco mira a trovare la chiave (le 2 chiavi)

Si suppone di conoscere almeno una coppia plaintex-ciphertext

Se abbiamo $C = E\left[K_2, E\left(K_1, P\right)\right]$

allora $X = E\left[K_1, P\right] = D\left[K_2, C\right]$

Se si ottiene un match, la coppia K_1, K_2 è probabilmente quella usata e si controlla con altri P e C



Si sono fatte $2^{56} + 2^{56} = 2^{57}$ prove

Si è però trascurato un fatto

Meet-in-the-Middle Attack (2)

Il Double-DES usa una chiave a 112 bit e quindi le chiavi possibili sono 2^{112}

Non una chiave produce un ciphertext di 64 bit ma bensì $2^{112}/2^{64} = 2^{48}$

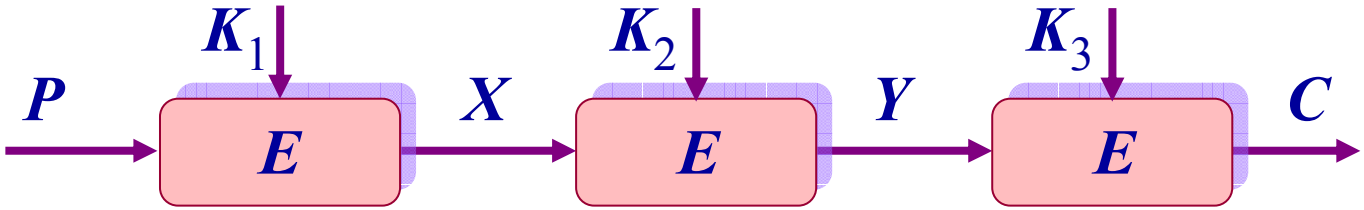
Quindi sulla prima coppia di P, C 2^{48} falsi allarmi

Se si ripete il test su un'altra coppia di P, C è come se il blocco avesse una lunghezza di 128 bits, allora il false alarm rate è $2^{112} - 2^{128} = 2^{-16}$

In pratica con poco sforzo in più di quello richiesto per rompere il DES si può rompere il Double-DES

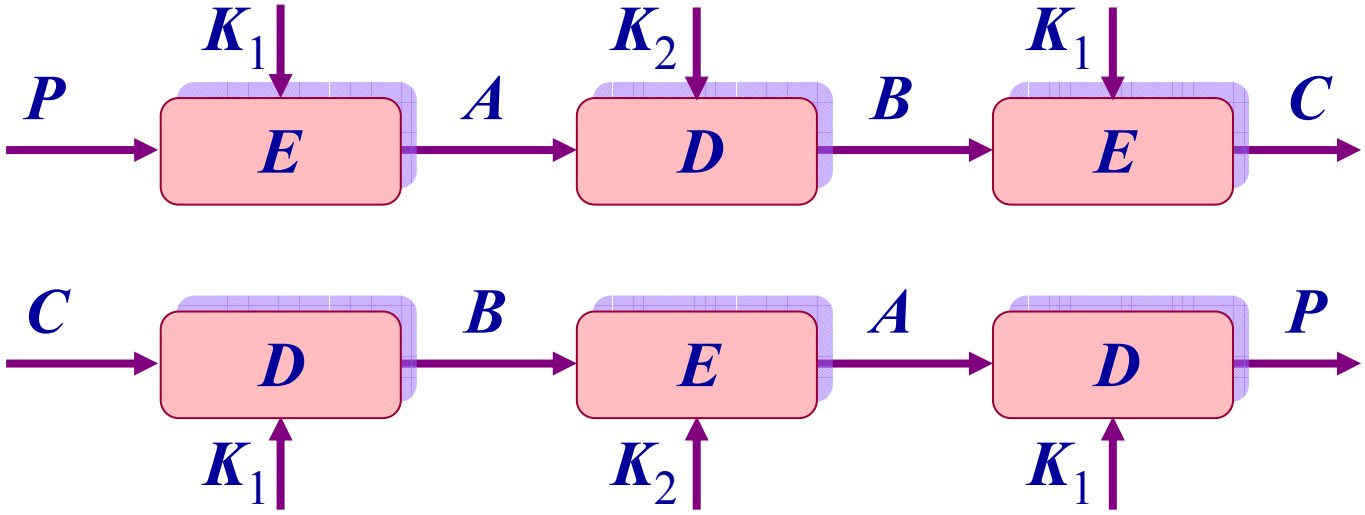
Triple-DES

Possibile aumentare la resistenza all'attacco Meet-in-the-Middle usando tre stadi di criptazione con tre chiavi



Attacco Meet-in-the-Middle comporterebbe $\approx 2^{118}$ test; troppo per essere fattibile ma il cipher sarebbe molto scomodo

Altra possibilità 3 encryption con due chiavi (EDE)



Triple-DES con due chiavi

Con chiavi da n bit, schema equivalente a chiave da $2n$ bit

Progettato da IBM per preservare la compatibilità con il DES $\left(K_1 = K_2\right)$

$$C = E\left[K_1, D\left(K_2, E\left(K_1, P\right)\right)\right]$$

N.B. Operazioni encrypt e decrypt equivalenti per la sicurezza

Triple-DES con 2 chiavi non suscettibile all'attacco Meet-in-the-Middle

Standardizzato in ANSI X9.17 & ISO 8732

Non sono noti ad oggi attacchi efficaci

Triple-DES con tre chiavi

Non noti attacchi efficaci verso il Triple-DES a 2 chiavi

Sono però temuti dalla gente

Quindi uso di Triple-DES con 3 chiavi

$$C = E\left[K_1, D\left(K_2, E\left(K_3, P\right)\right)\right]$$

Compatibile con il Triple-DES a 2 chiavi prendendo $K_3 = K_2$ o $K_1 = K_2$

Adottato da diverse applicazioni Internet, p.e. PGP, S/MIME

Modi di funzionamento

Block ciphers criptano blocchi di lunghezza fissa e limitata

p.e. DES cripta blocchi di 64 bit con chiave da 56 bit

Esigenza di un qualche modo per criptare/decriptare quantità arbitrarie di dati

ANSI ha definito in X3.106-1983 quattro modi funzionamento

Modi ripresi dal NIST con il FIPS 81

Successivamente il NIST ha portato questi modi a cinque

Usabili con DES, AES, IDEA, Blowfish, etc

Modi di funzionamento (2)

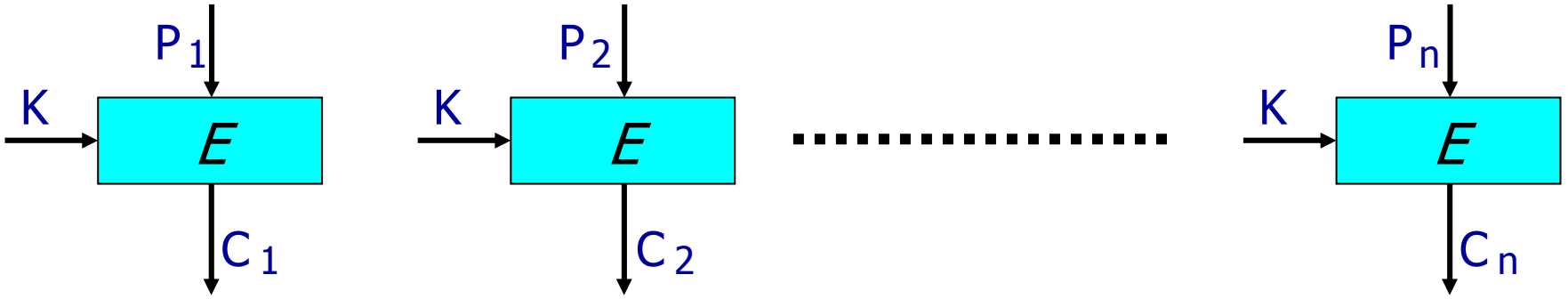
- Electronic CodeBook (ECB)
- Cipher Block Chaining (CBC)
- Cipher FeedBack (CFB)
- Output FeedBack (OFB)
- Counter (CTR)

Electronic CodeBook (ECB)

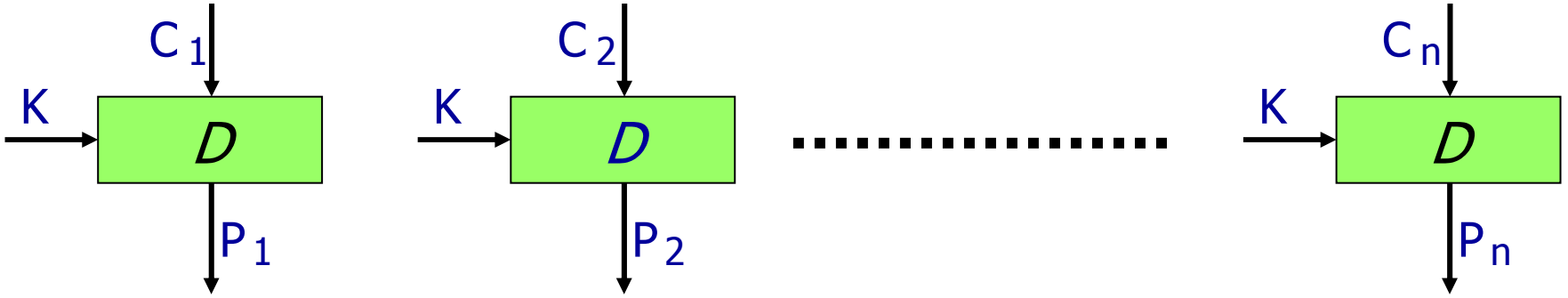
- Il modo di funzionamento più ovvio
- Messaggio spezzato in blocchi indipendenti
- Eventuale padding finale
- Tutti i blocchi encrypted con la stessa chiave
- Ciascun blocco è un valore che è sostituito, come un codebook, da ciò il nome
- Uso: Trasmissione di piccole quantità di dati
- Se un blocco di plaintext si ripete, si ripete anche il blocco di ciphertext –> nascità di regolarità sfruttabili dalla criptanalisi

Electronic CodeBook (ECB)

Encryption



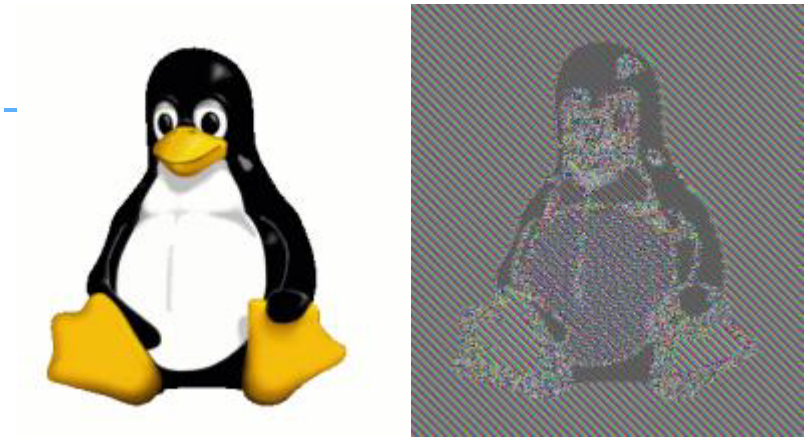
Decryption



Vantaggi e Limiti di ECB

Le ripetizioni dei messaggi possono rilevarsi nel ciphertext se hanno periodicità uguale o multipla alla chiave

Fenomeno pesante con i dati di tipo immagine



Vantaggi e limiti di ECB (2)

Lo stesso accade con i messaggi che presentano soltanto piccole variazioni

No propagazione degli errori. Normalmente l'errore su un bit da luogo a ≈ 50% di bit errati sul plaintext

Un opponent potrebbe sostituire uno o più blocchi

Nel caso di messaggi lunghi si può migliorare la sicurezza inserendo in ogni blocco un numero di padding bit random

Cipher Block Chaining (CBC)

Serve una tecnica con cui blocchi di plaintext ripetuti diano luogo a blocchi di ciphertext differenti

Con CBC ogni blocco di plaintext prima di essere criptato è XOR-ed con il ciphertext del blocco precedente

Problema: Cosa fare con il primo blocco?

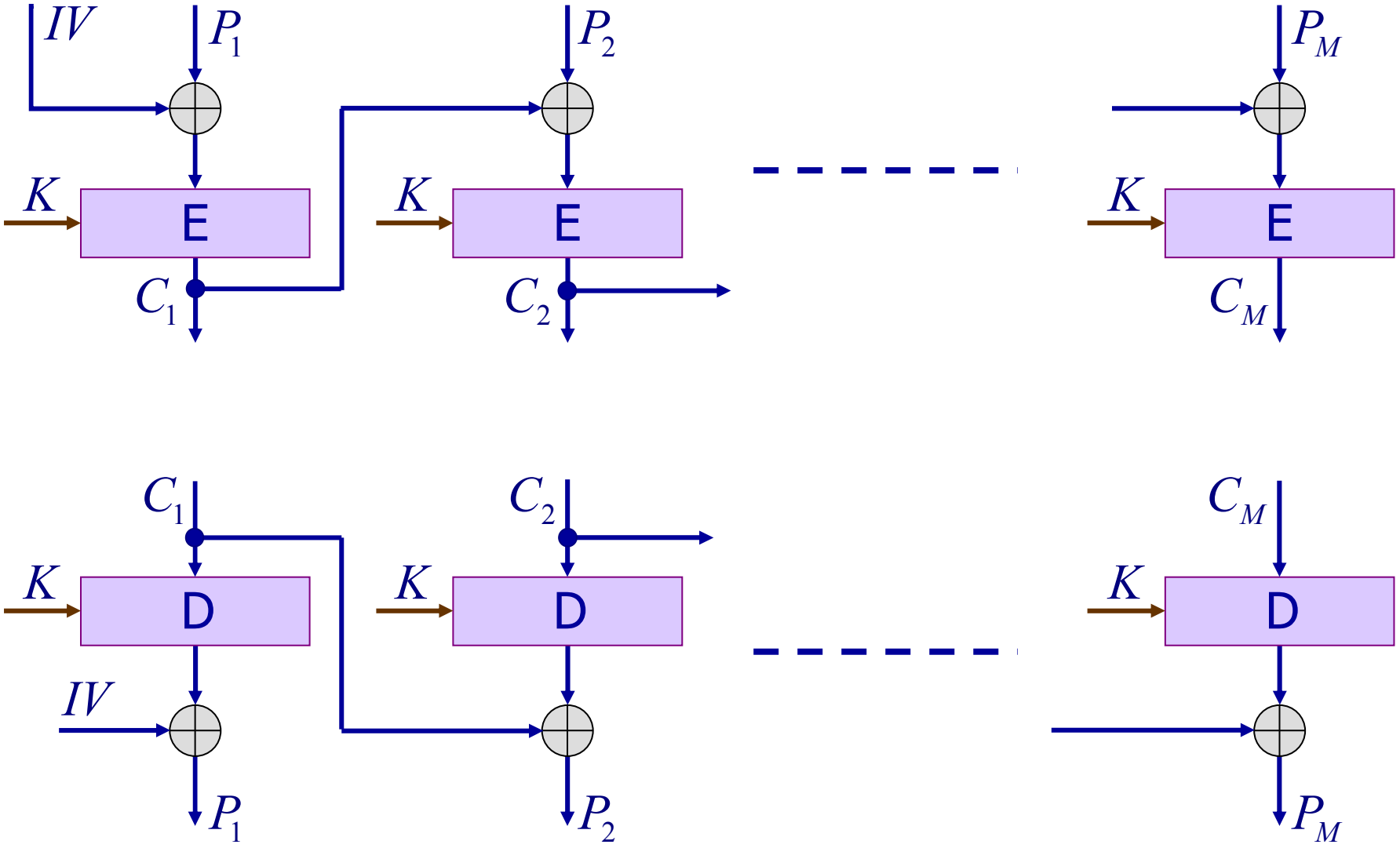
Fare uso di un Initial Vector *(IV)* lungo quanto il blocco

L'Initial Vector deve essere noto solo alle due parti

Deve anche essere protetto contro cambiamenti

Può essere fisso o si può trasmetterlo prima con ECB

Cipher Block Chaining (CBC)



Cipher Block Chaining (CBC)

Trasmettendo più volte un dato plaintext con la stessa chiave e lo stesso *IV* si ottiene lo stesso ciphertext

Si può superare il problema cambiando *IV* o la chiave o meglio usando un primo blocco random che si ripercuote su tutti gli altri

Propagazione dell'errore: l'errore su un bit del blocco C_r influenza la decryption di C_r e di C_{r+1}. Il blocco P'_r sarà totalmente random (percentuale di errori del 50%) mentre il blocco P'_{r+1} avrà errore soltanto in corrispondenza del bit errato di C_r

Un opponent può provocare un cambiamento predicibile di un bit su P_{r+1} alterando il bit corrispondente di C_r

Cipher Block Chaining (CBC)

Il meccanismo di chaining fa sì che il blocco C_r dipenda da P_r e da tutti i blocchi precedenti. Se si altera l'ordine dei blocchi non è possibile effettuare la decryption.

Il CBC è self-synchronizing nel senso che l'errore su un bit o al limite su tutti i bit di un blocco si ripercuote soltanto su quel blocco e su quello successivo. Anche la perdita di uno o più interi blocchi ha effetto limitato

La self-synchronization si presenta soltanto in presenza di errori su bit o della perdita di blocchi. Non si presenta nel caso di perdita di uno o più bit (framing integrity errors)

Cipher Block Chaining (CBC)

Trasmissione dell'IV critica

Opponent potrebbe proporre un IV

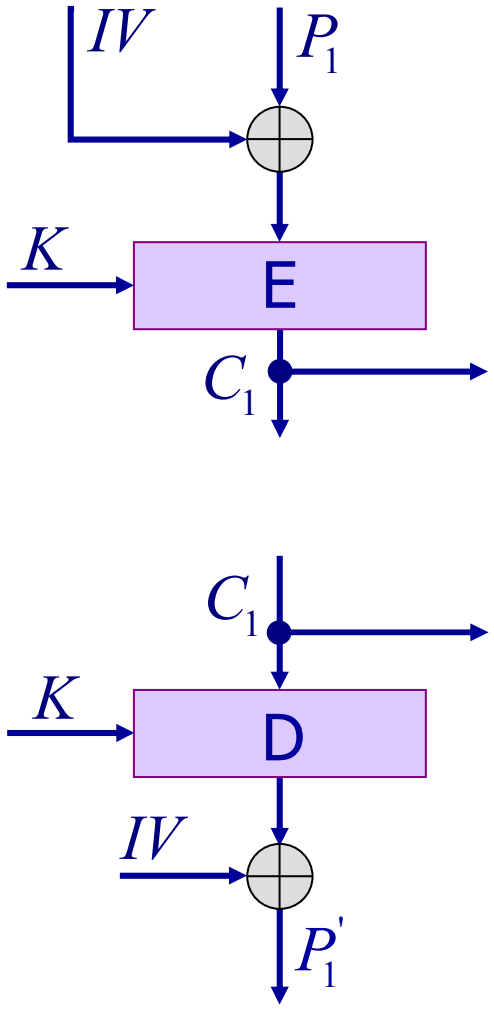
$$C_1 = E\left[K,(IV \oplus P_1)\right]$$

$$P_1 = IV \oplus D\left[K,C_1\right]$$

Considerazione per il bit i-esimo di X

$$P_1(i) = IV(i) \oplus D\left[K,C_1\right](i)$$

IV ancorché non segreto deve essere protetto perché con un suo cambiamento si può produrre un cambiamento predicibile del primo blocco



Message Padding

Alla fine del messaggio possibile presenza di un blocco più breve del blocco del cipher

Possibile padding con valori non-data (p.e. null) noti oppure possibile un padding dell'ultimo blocco con un contatore del padding

p.e. [b1 b2 b3 0 0 0 0 5]

significa avere 3 bytes di dati, quindi 5 bytes di pad+count

Ciò può richiedere un intero blocco aggiuntivo a quelli proprii del messaggio

Esistono altri modi più sofisticati che evitano la nascita del blocco aggiuntivo

Cipher FeedBack (CFB)

- Talvolta è necessario procedere con immediatezza alla criptazione di blocchi piccoli di lunghezza *r* < della lunghezza *b* del block cipher
- Il messaggio è trattato come uno stream di bit
- Aggiunto all'uscita del block cipher
- Cipher FeedBack e Output FeedBack producono uno stream cipher
- Standard consente feedback di qualsiasi numero di bit (1, 8, 64, etc)
- Conosciuti come CFB-1, CFB-8, CFB-64, CFB-128, etc
- Massima efficienza con numero di bit = lunghezza del blocco (64 o 128)

$$C_i = P_i \oplus \mathrm{DES}_K(C_{i-1})$$
$$C_{-1} = IV$$

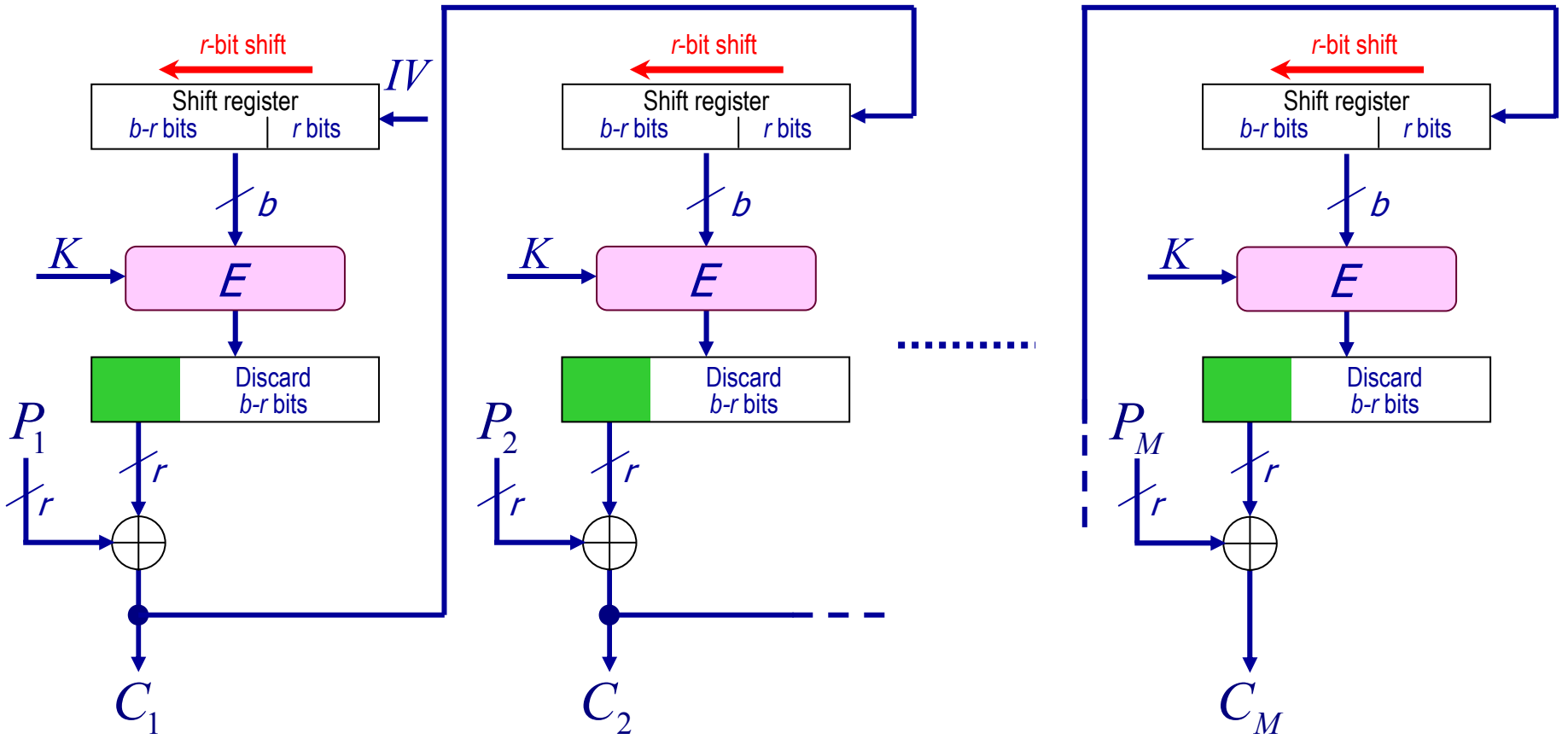
- Usi: Stream data encryption, autenticazione

Cipher FeedBack (CFB) (2)

Cipher con blocco di lunghezza *b*

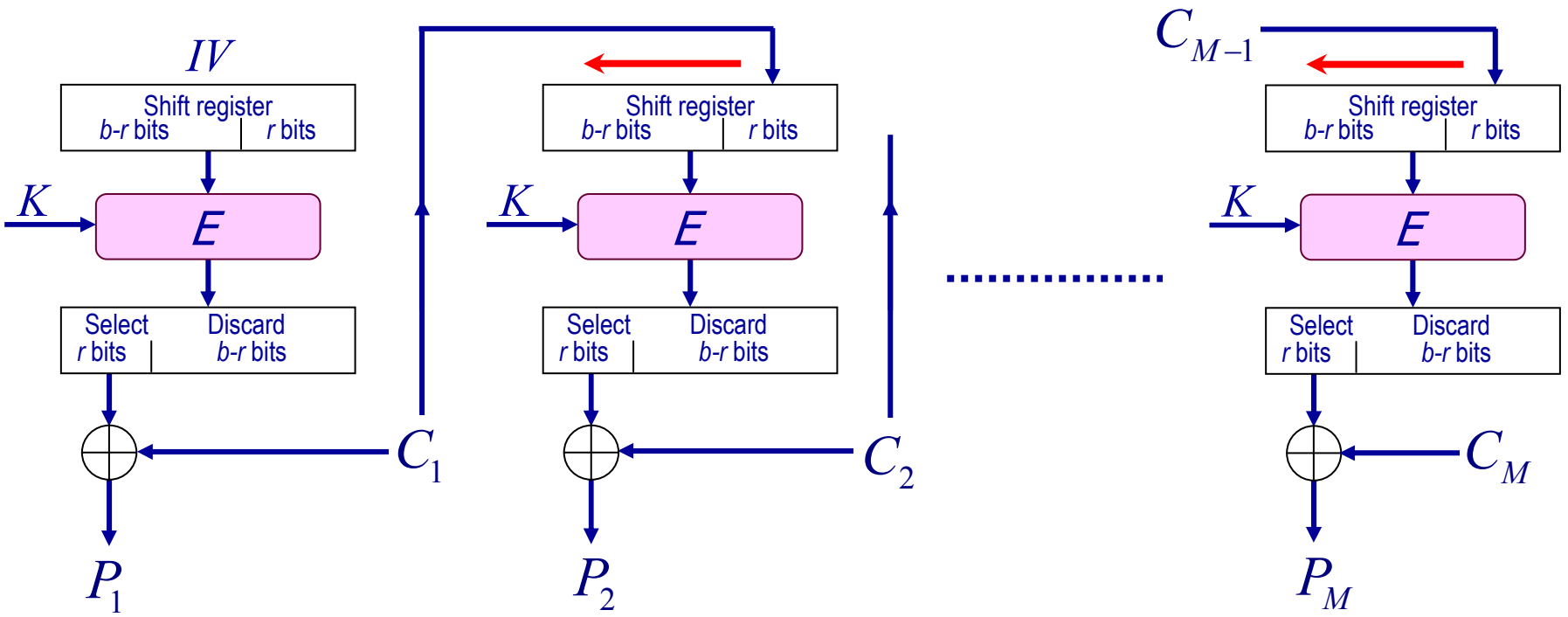
IV lungo *b* bit

Feedback di *r* bit



Uso più frequente con unità di trasmissione $r = 8$

Cipher FeedBack (CFB) (3)



Cipher FeedBack (CFB) (4)

- CFB adatto quando i dati arrivano in unità di bits/bytes
- Stream mode più comune
- Un limite è la necessità di stallo mentre si costruisce un blocco
- Encryption dopo ogni blocco di *r* bit
- Si ha una propagazione degli errori nei blocchi successivi

Output FeedBack (OFB)

È un modo simile al CFB eccetto che il riporto sul blocco successivo non dipende dal plaintext

Output del cipher aggiunto al blocco di messaggio

L'output del cipher è pure riportata al blocco successivo

Possibile precalcolare gli output dei cipher e quindi rapidità

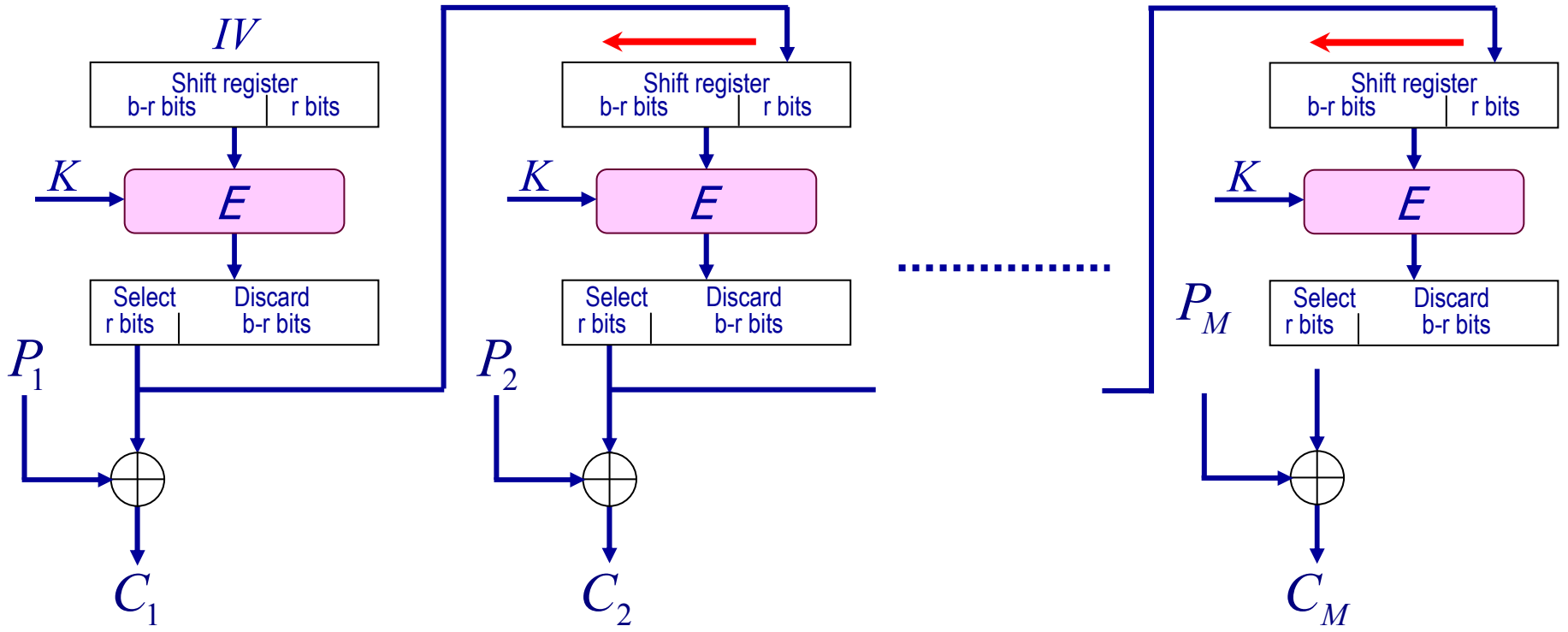
$$C_i = P_i \oplus O_i$$

$$O_i = DES_K\left(O_{i-1}\right)$$

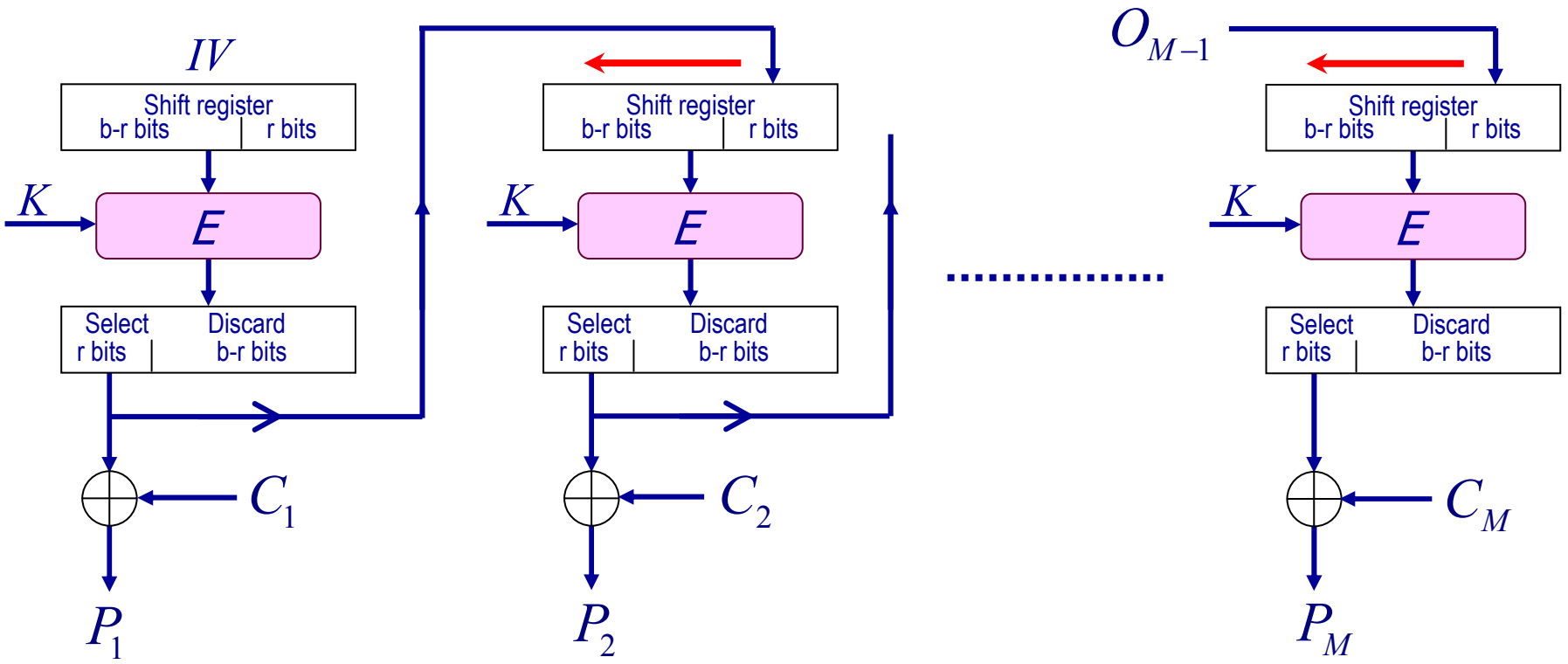
$$O_{-1} = IV$$

Uso: stream encryption su canali rumorosi

Output FeedBack (OFB)



Output FeedBack (OFB)



Vantaggi e limiti di OFB

Assenza di propagazione degli errori

Più vulnerabile alla modifica del flusso del messaggio

In sostanza una variazione sul cifrario di Vernam, quindi non si deve mai riusare lo stesso insieme (key + IV)

Il sender ed il receiver devono rimanere in sincronismo

Inizialmente definito con un feedback generico di m bits

Successivamente si è notato che si deve effettuare soltanto un feedback lungo quanto il blocco(OFB-64 e OFB-128)

Counter (CTR)

Un modo entrato in uso recentemente anche se vecchio

Simile ad OFB ma piuttosto che criptare valori di feedback

lo fa con i valori di un counter

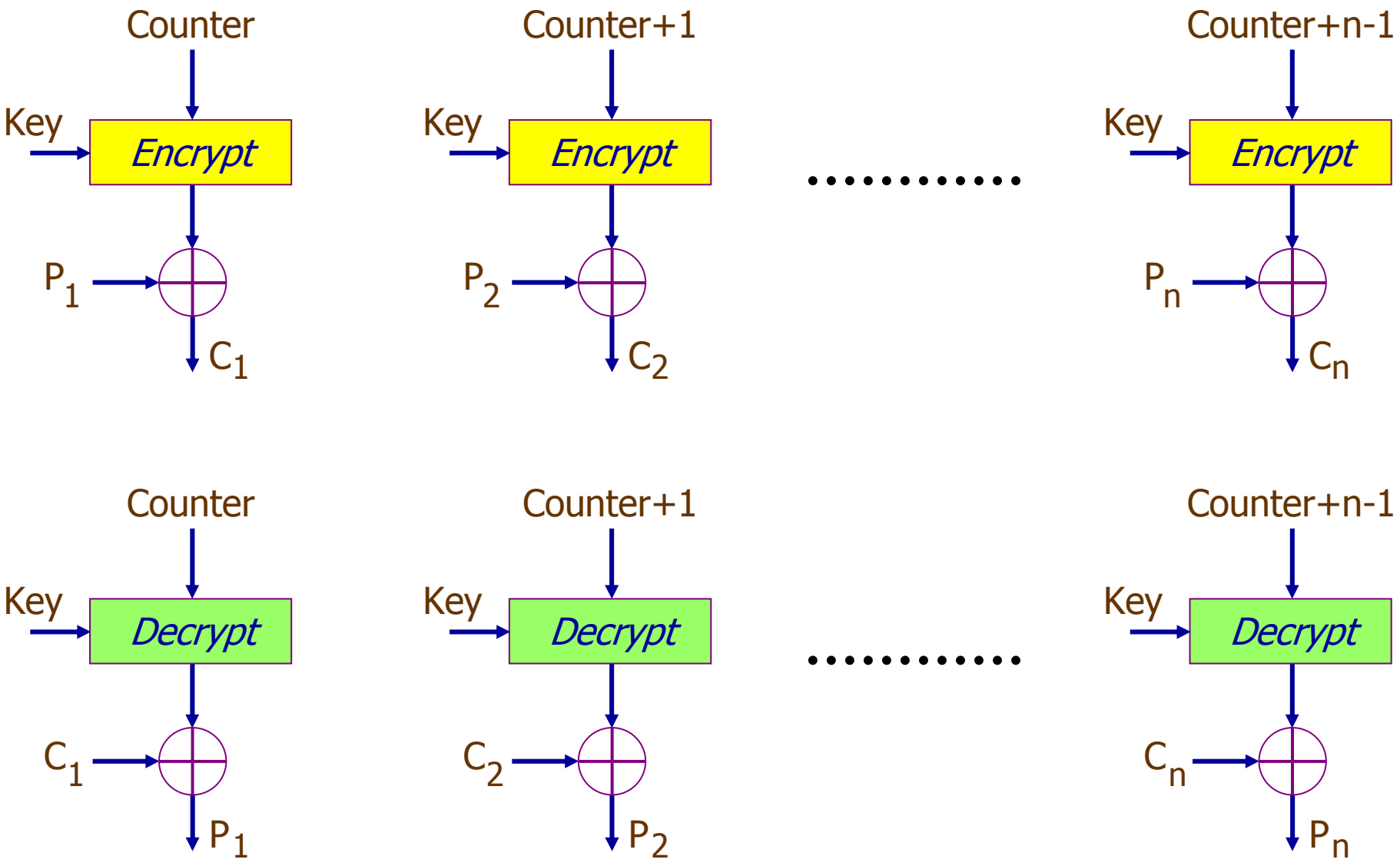
Bisogna non riutilizzare il valore del counter

$$C_i = P_i \oplus O_i$$

$$O_i = DES_K(i)$$

Uso: Encryption nelle reti ad alta velocità

Counter (CTR)


Counter
Key
Encrypt
P1
C1
Counter+1
Key
Encrypt
P2
C2
Counter+n-1
Key
Encrypt
Pn
Cn
Counter
Key
Decrypt
C1
P1
Counter+1
Key
Decrypt
C2
P2
Counter+n-1
Key
Decrypt
Cn
Pn

Vantaggi e limiti di CTR

Efficienza

Possibile effettuare encryptions in parallelo in h/w o s/w

Possibile pre-elaborare i counter

Buono per linee rumorose ad alta velocità

Possibile accedere in modo random ai diversi blocchi criptati

Di provata sicurezza (come gli altri modi)

Bisogna essere certi del non riuso del sistema key/counter

Stream Ciphers

Conversione da plaintext a ciphertext un bit (o byte) alla volta

Una possibilità costituita da un processo di somma modulo 2

Allo stream del messaggio si somma un keystream random

Con un keystream di lunghezza ∞ si ha il cipher Vernam

La casualità del keystream distrugge completamente le proprietà statistiche del messaggio

Uno stream cipher può essere un cipher Vernam con chiave di lunghezza finita che si ripete

In altri termini un cipher Vernam con keystream pseudorandom

Stream Ciphers (2)

- Considerazioni di progetto:
  - Serve un keystream con periodo lungo
  - Serve una casualità statistica
  - Dipendenza da chiavi sufficientemente lunghe
- Ben progettato altrettanto sicuro di un block cipher con la stessa dimensione di chiave
- Normalmente più semplice e veloce

Tipi di Stream Cipher

In uno stream cipher un Keystream Generator (KG) genera un Keystream viene X-ored al flusso del messaggio

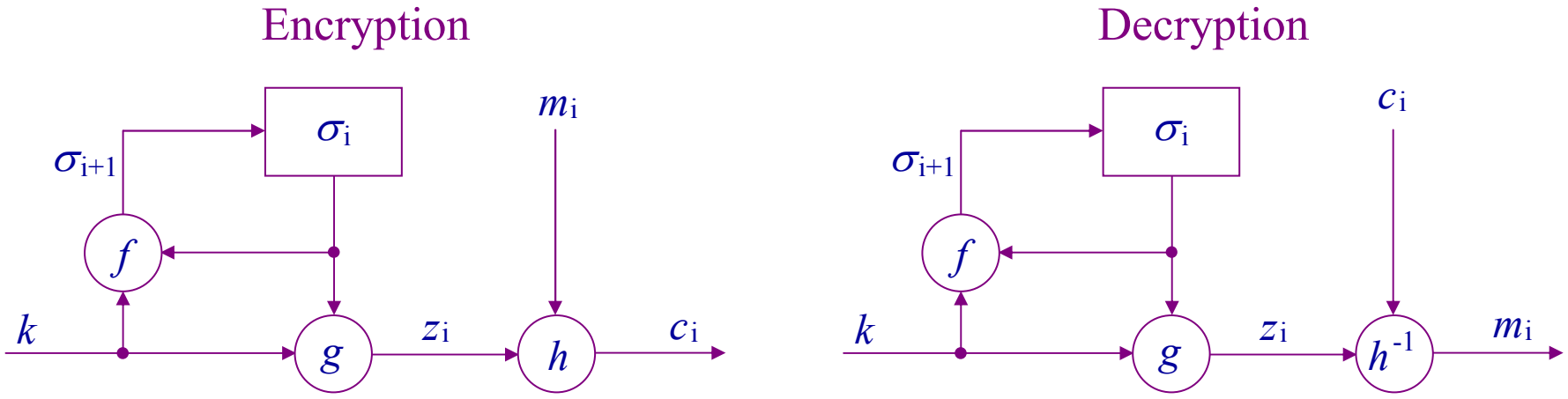
Possibile classificazione in base al keystream generator

Synchronous KG

Self-synchronising KG

Un synchronous stream cipher genera un keystream in modo indipendente dal plaintext e dal ciphertext

Synchronous Stream Cipher



Necessità di sincronizzazione

In caso di perdita o inserimento di un bit serve ri-sincronizzazione

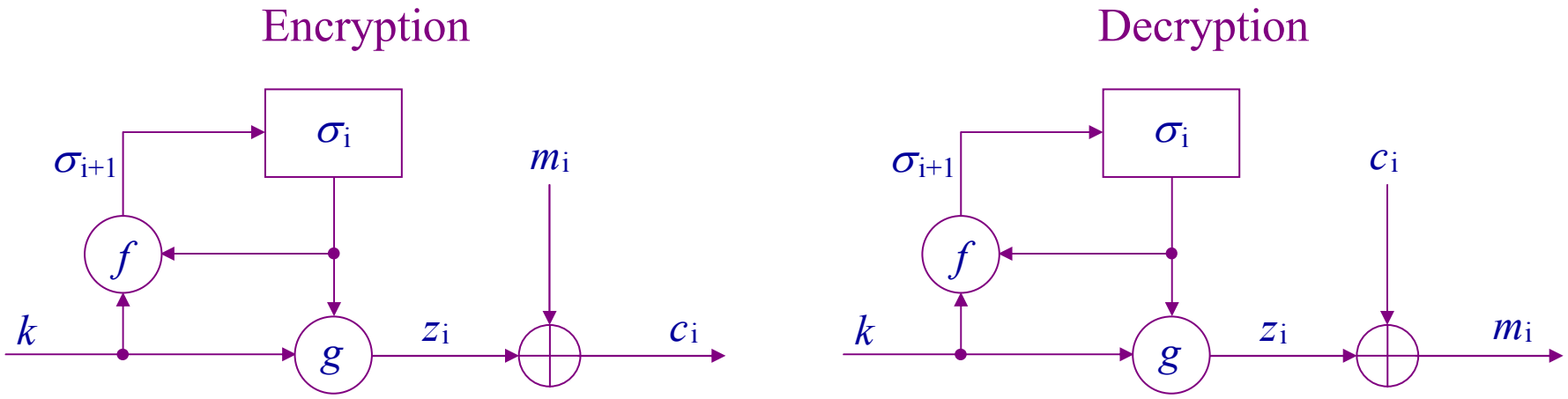
Un errore di trasmissione su un bit, da luogo in output a un solo bit errato

Aspetto negativo: questi cipher sono suscettibili agli attacchi attivi
- Se un attacker può cambiare un bit nel ciphertext, può cambiare in modo prevedibile il bit corrispondente del plaintext

Synchronous Stream Cipher

La funzione h è generica

Se la funzione h è uno XOR si parla di binary additive stream cipher



Self-Synchronizing Stream Cipher

Il keystream è una funzione della chiave e di un definito numero di bit precedenti del ciphertext

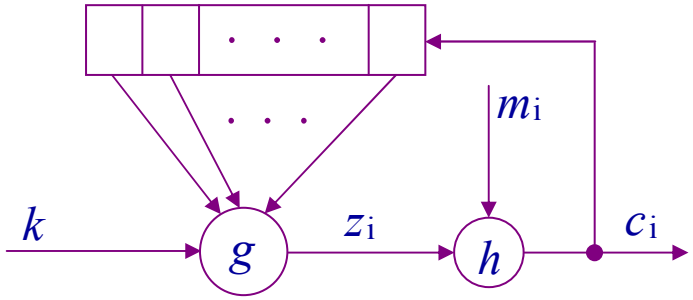
$$\sigma_i = (c_{i-t}, c_{i-t+1}, \ldots, c_{i-1})$$
$$z_i = g(\sigma_i, k)$$
$$c_i = h(z_i, m_i)$$

$\sigma_0 = (c_{-t}, c_{-t+1}, \ldots, c_{-1})$ Stato iniziale non segreto

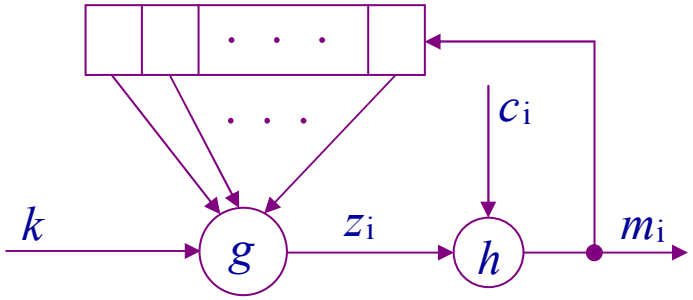
I Self-Synchronizing Stream Cipher più usati sono basati sui block cipher operanti con un feedback di un bit

Self-Synchronizing Stream Cipher

Encryption



Decryption



In caso di inserimento di digit spuri o di perdita possibile self-synchronization

Limitata propagazione degli errori

Meno soggetti agli attacchi attivi (più facilmente rilevabili)

Diffusione delle proprietà statistiche del plaintext

RC4

- Noto pure come ARC4 o ARCFOUR
- Binary Additive Stream Cipher proprietario della RSA
- Progetto semplice ma efficiente
- Funzionamento byte-oriented
- Lunghezza di chiave variabile
- Largamente usato: WEB SSL/TLS, WLAN WEP e WPA
- Chiave genera permutazione casuale di tutti i valori a 8 bit
- Usa quella permutazione per l'elaborazione dell'informazione
- Fino al 1994 trade secret ma da allora noto

RC4 (2)

L'algoritmo funziona in OFB

Il keystream è indipendente dal plaintext

Inizialmente array S di numeri da 0 a 255

S costituisce l'**internal state** del keystream generator

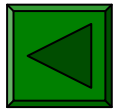
Le varie entries sono permutazioni dei numeri da 0 a 255

La permutazione è una funzione della chiave (di lunghezza variabile)

Due counters i e j inizializzati a 0

Per un byte da cifrare viene selezionato un byte dello state

Dopo di ciò nuova permutazione



RC4 (3)

In primo luogo inizializzazione dell'array S

Riempimento dell'array in modo lineare

```
for i = 0 to 255 do
S_i = i
```

Riempimento di un array T con la chiave K (lunga $keylen$)

```
for i = 0 to 255 do
T_i = K(i mod keylen)
```

RC4 (4)

Permutazione dell'array S in funzione dell'array T

```
j = 0
for i = 0 to 255 do
j = (j + S_i + T_j) mod 256
swap S_i, S_j
```

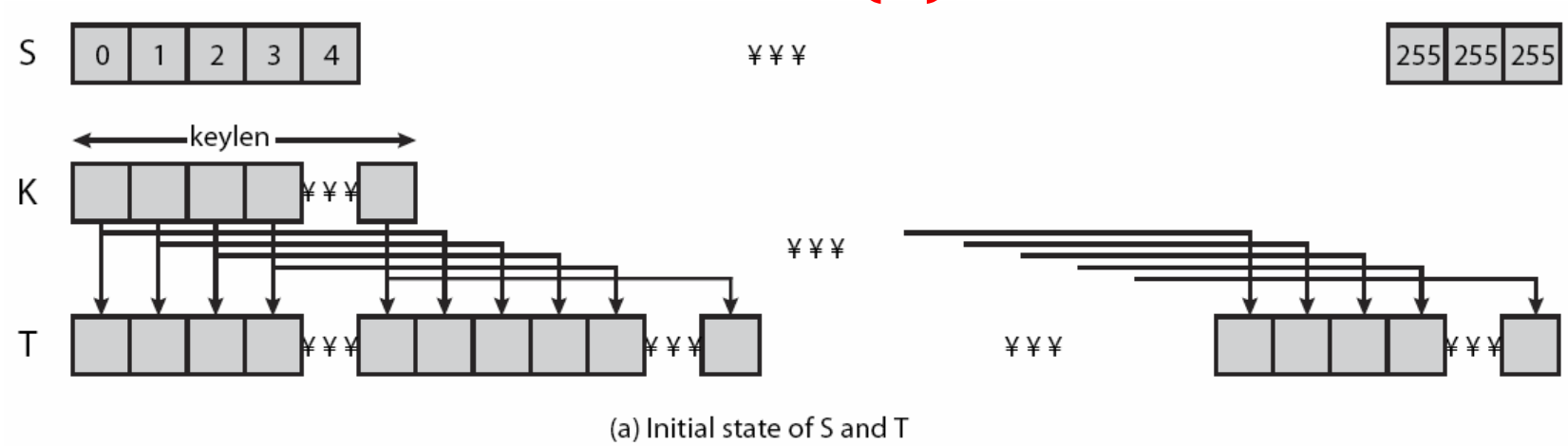
RC4 (5)

Generazione di un byte K casuale:

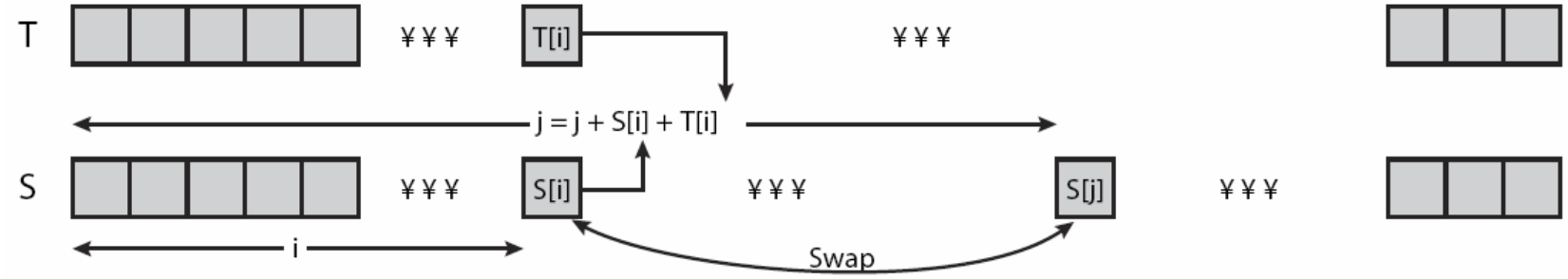
```
i = (i + 1) mod 256
j = (j + S_j) mod 256
swap S_i, S_j
t = (S_i + S_j) mod 256
K = S_t
```

Il byte K è XORed con un byte di plaintext per produrre un byte di ciphertext o con uno di ciphertext per produrne uno di plaintext

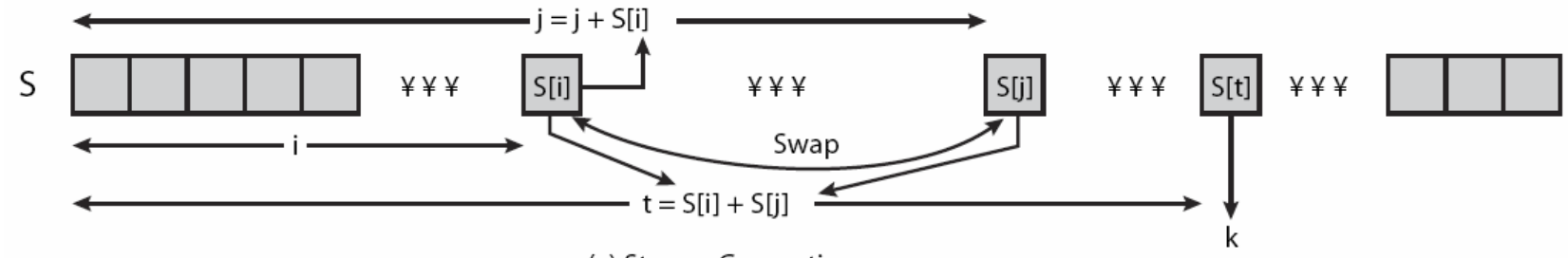
RC4 (6)



(a) Initial state of S and T



(b) Initial permutation of S



(c) Stream Generation

RC4 (7)

È dichiarato sicuro nei riguardi di tutti gli attacchi

L'S-box si può trovare in 256! x 2^{256} possibili stati

L'S-box evolve con l'uso

- i assicura il cambiamento di ciascun elemento
- j assicura che il cambiamento è random

Vi sono alcune analisi ma nessuna con dati pratici

Il risultato finale è fortemente non lineare

RC4 è uno stream cipher, quindi non si deve riusare la chiave

Esiste un problema noto con il WEP, ma è dovuto al sistema di gestione della chiave e non ad RC4

RC4 è implementabile in modo veloce tramite software

SEAL

Software-optimized Encryption ALgorithm

Come dice il nome algoritmo adatto per implementazione software

Patented dalla IBM

Ottimizzato per CPU a 32 bit

Funzionamento ad alta velocità

Binary additive stream cipher

Cripta il data stream XOR-ing con una PRS

Tre versioni SEAL 1.0, SEAL 2.0 e SEAL 3.0

Chiave di 160 bit

SEAL non è un vero stream cipher quanto una

Length-increasing pseudorandom function

che mappa una sequenza n di 32 bit in una stringa $K(n)$ di L bit

SEAL (2)

SEAL preprocessa la chiave ottenendo grandi tabelle

Data la chiave k da 160 bit e un intero n a 32 bit SEAL ottiene

una stringa $k(n)$ di lunghezza L < 64 Kbyte

Dopo la preelaborazione della chiave la generazione del keystream

richiede solo 5 istruzioni di macchina per byte

Con k random, allora $k(n)$ è computazionalmente indistinguibile

da una funzione random a L bit

Alla fine il flusso del messaggio è XOR-ed con la pseudo random string

SEAL (3)

Vantaggio dei sistemi con pseudo-random function family:

Con i sistemi classici di keystream generation per ottenere un bit di una sequenza bisogna calcolare tutti quelli precedenti

Con questi sistemi possiamo ottenere un elemento a caso

Utilità per l'encryption di hard disk

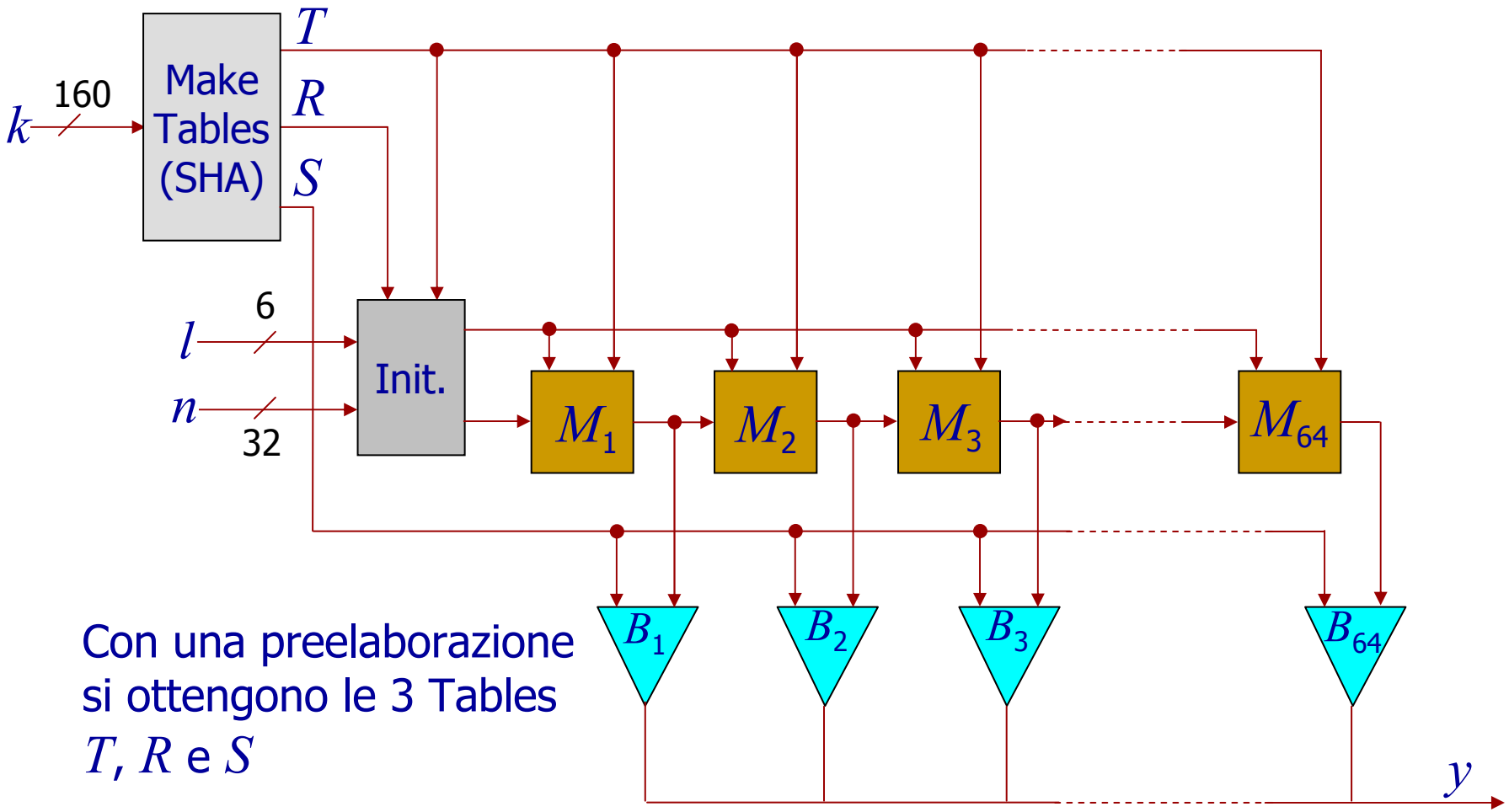
Possibile generare la PRS per il settore *n* senza aver generato quelle per i precedenti valori di *n*

Facilitazione dei problemi di sincronizzazione:

Possibile inviare n insieme al messaggio n-esimo

SEAL (4)

Le Tables T, R e S richiedono ≅ 3 Kbyte e servono ≅ 200 calcoli di SHA



Con una preelaborazione si ottengono le 3 Tables T, R e S

A5

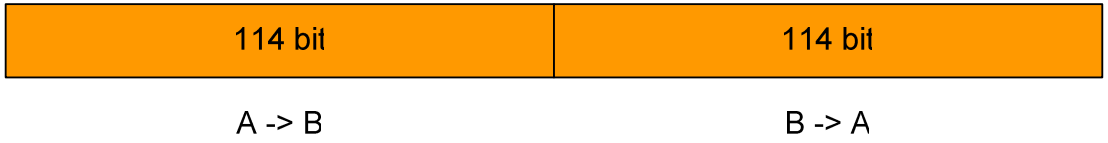
Forse il più diffuso stream cipher

Usato dal GSM per la encryption della voce/dati

Mai reso pubblico, quindi noto solo in modo non ufficiale

Due varianti di A5: ad alta sicurezza A5/1 a bassa sicurezza A5/2

GSM trasmette un frame ogni 4,6 ms, formato da 228 bit
114 in un senso e 114 in quello opposto



Una conversazione è criptata con una chiave di sessione K
di 64 bit

10 bit sono uguali a 0, sicché la chiave effettiva è di 54 bit

A5 (2)

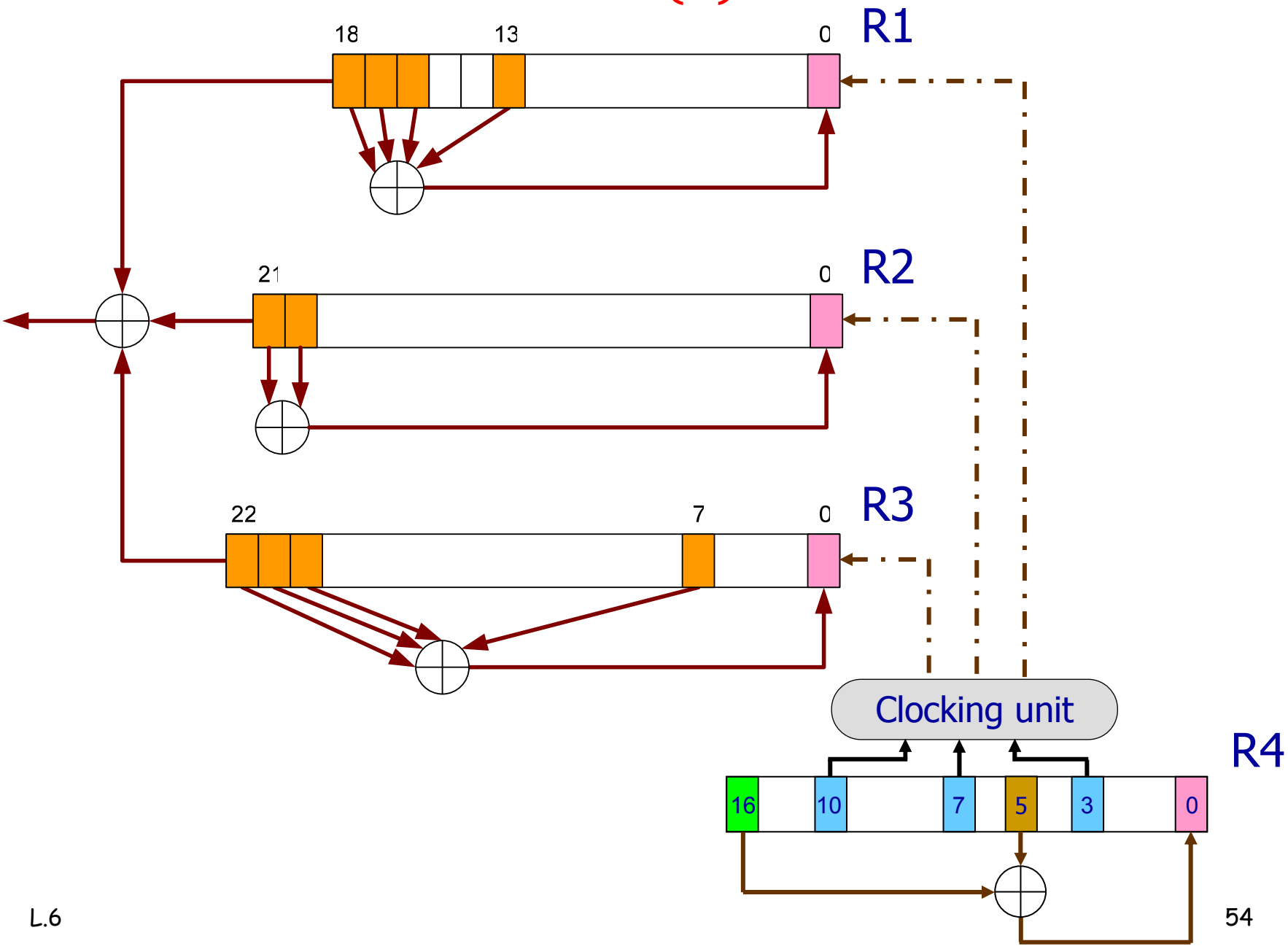
La chiave K è unita ad un frame counter Fn di 22 bit noto al pubblico e il risultato è lo stato iniziale del generatore di un keystream con unità di 228 bit

L'unità di keystream di 228 bit è XORed dalle due parti A e B con i 114 + 114 bit di plaintext per produrre i 114 + 114 bit di ciphertext

L'unità di keystream di 228 bit è generata usando tre Linear Feedback Shift Register (LFSR) R1 (19 bit), R2 (22 bit) e R3 (23 bit), più un quarto LSFR R4 (17)

I registri sono di tipo Maximal Length

A5 (3)



A5 (4)

Funzionamento del meccanismo di clocking

- R4 controlla il clocking di R1, R2 e R3
- Dovendo effettuare il clocking di R1, R2 e R3 si iniettano nella Clocking unit i bit R4[3], R4[7] e R4[10]
- La Clocking unit calcola la majority function su questi 3 bit

$$maj(a,b,c) = a \cdot b \oplus b \cdot c \oplus c \cdot a$$

- R1, R2 e R3 sono clocked, se e solo se, rispettivamente R4[1], R4[3] e R4[7] è uguale alla majority
- Dopo queste operazioni R4 è clocked

A5 (5)

Senza funzionamento stop/go, il periodo della somma dei tre LSFR è dato da

$$(2^{19}-1)(2^{22}-1)(2^{23}-1)$$

Tenendo conto dello stop/go si si ha un periodo di circa $4\cdot(2^{23}-1)/3$

All'avvio i registri sono inizializzati a 0

Per 64 cicli di clock si provvede ad aggiungere la chiave

A5 (6)

Nel ciclo i si aggiunge al bit meno significativo di ciascun registro il bit i-esimo della chiave usando un'XOR

$$R(0) = R(0) \oplus K(i)$$

Dopo l'XOR si invia il clock

Similmente in 22 cicli di clock si aggiunge il frame number

Si effettuano 100 cicli di clock a vuoto, ossia senza prendere l'uscita

A questo punto il keystream generator è pronto

Stream Cipher Asimmetrici

Possono esistere anche stream cipher asimmetrici

(con chiave privata e pubblica)