Modulo 4: Cifrari a Blocchi e Modalità di Operazione

Appunti di Crittografia

Indice

1 Introduzione

In questo modulo abbandoniamo l'idea di cifrare bit per bit (*Stream Ciphers*) per passare all'elaborazione di blocchi di dati di dimensione fissa. Analizzeremo perché un algoritmo di cifratura da solo (come AES) non è sufficiente e necessita di una "Modalità di Operazione" per essere sicuro e utile.

2 Fondamenti Teorici: Il Cifrario a Blocchi

Un cifrario a blocchi è una primitiva crittografica che opera su blocchi di testo in chiaro (P) di lunghezza fissa n (es. 128 bit per AES) e produce blocchi di testo cifrato (C) della stessa lunghezza n, utilizzando una chiave K.

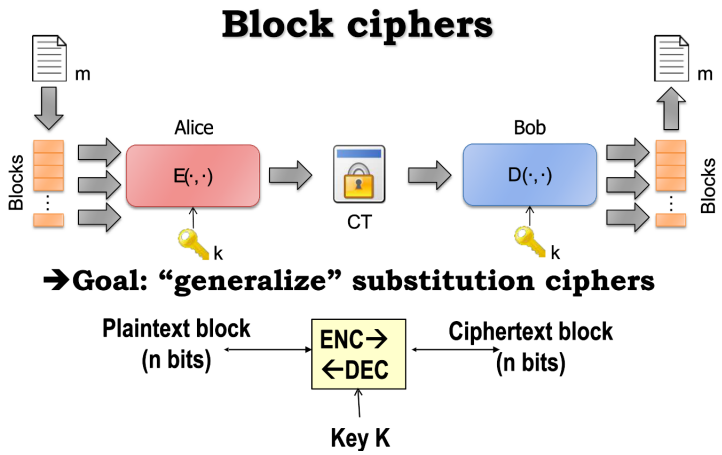


Figura 1: Schema generale di un Cifrario a Blocchi

Il loro goal è quello di andare a "generalizzare" i cifrari a sostituzione.

2.1 Definizione come PRP (Pseudo-Random Permutation)

Dal punto di vista matematico, un cifrario a blocchi ideale modella una **Permutazione Pseudo-Casuale (PRP)**.

- **Permutazione:** È una funzione biiettiva (invertibile) $E : \{0,1\}^n \to \{0,1\}^n$. Mappa ogni possibile input in un unico output e viceversa.
- **Pseudo-Casuale:** La permutazione specifica scelta dalla chiave K deve essere indistinguibile da una permutazione scelta uniformemente a caso dall'insieme di tutte le permutazioni possibili.

Un esempio con $n = 3$ bit:

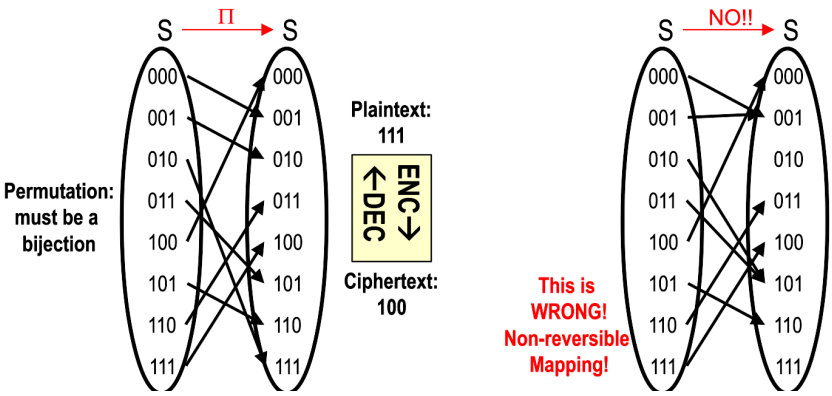


Figura 2: Permutazione con $n = 3$

2.2 Lo Spazio delle Permutazioni (Perché serve la Chiave?)

Se operiamo su blocchi di $n = 3$ bit, l'insieme dei possibili messaggi S ha dimensione $2^3 = 8$. Quante permutazioni possibili esistono su 8 elementi? Sono $8! = 40320$.

Tuttavia, quando n cresce, il numero esplode. Per **AES (Advanced Encryption Standard)** ($n = 128$), il numero di possibili permutazioni è $2^{128}!$ (fattoriale).

Utilizzando l'approssimazione di Stirling, questo numero è circa $2^{2^{135}}$, una cifra inimmaginabilmente grande (una chiave per selezionare una permutazione da questo insieme richiederebbe 10^{40} cifre).

Poiché non possiamo gestire tutte queste permutazioni, la **Chiave** K (es. 128 o 256 bit) serve a selezionare un sottoinsieme molto piccolo ma "ben distribuito" di queste permutazioni. Anche se 2^{256} è molto minuscolo rispetto a $2^{128}!$, è sufficiente per la sicurezza pratica. Le permutazioni di AES sono molto meno del PRP ideale, ma vanno ancora bene dal punto di vista crittografico.

3 Il Fallimento dell'Approccio Ingenuo: ECB

La modalità più semplice e intuitiva per cifrare un messaggio lungo è dividerlo in n blocchi e cifrare ciascuno separatamente con la stessa chiave. Questa modalità è chiamata **Electronic Code Book (ECB)**.

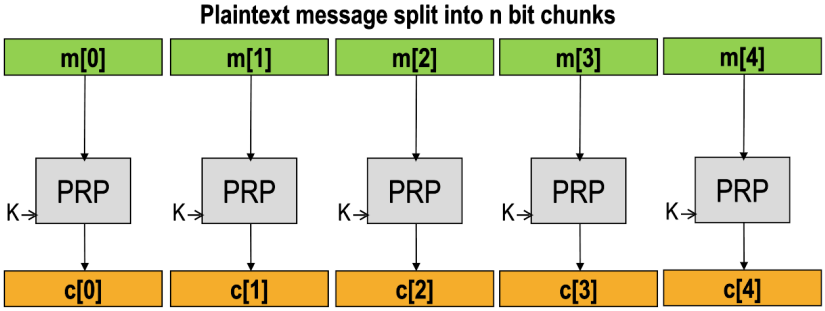


Figura 3: Electronic Code Book (ECB)

Come possiamo vedere in figura, il messaggio originale viene diviso in n chunks. Ogni chunk viene passato alla funzione PRP, con chiave K (notiamo che K è uguale per tutti),

e viene generato il testo cifrato relativo a quel blocco (es. $c[0]$ è il ciphertext del blocco $m[1]$).

Abbiamo quindi una cifratura indipendente per ogni blocco. Questo approccio però porta con sé un problema molto importante.

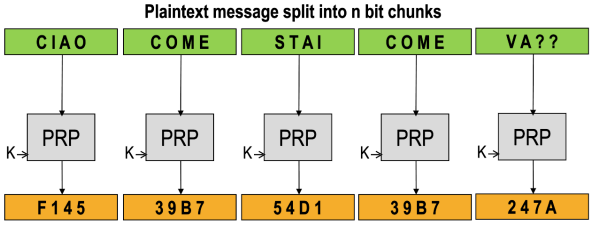
3.1 Funzionamento e Debolezza

Come abbiamo detto, l'operazione che avviene per ogni blocco del messaggio è:

$$C_i = \text{ENC}(K, P_i)$$

Dove sta il problema di ECB? Sembra abbastanza sicuro! Il problema fatale di ECB è che è **deterministico**.

- Se nel messaggio appare due volte lo stesso blocco di testo in chiaro (es. "CIAO" e poi di nuovo "CIAO"), verrà prodotto due volte lo stesso identico blocco di testo cifrato.

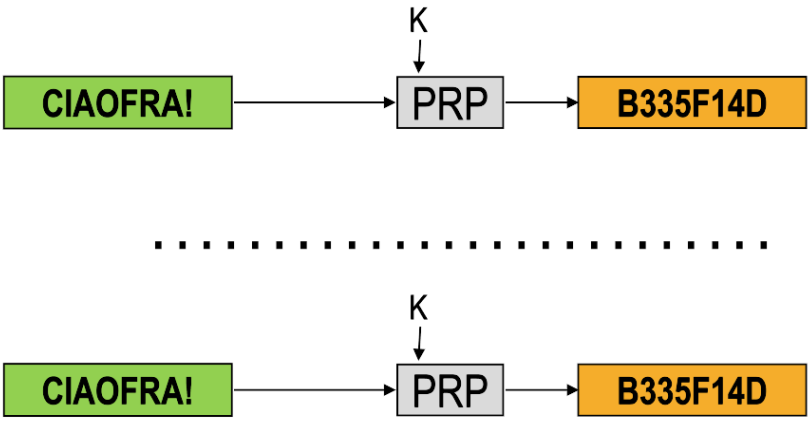


- **Analisi del Traffico:** Un attaccante vede pattern ripetuti nel ciphertext e deduce pattern nel plaintext.

Esempio Visivo (Il Pinguino): Se cifriamo un'immagine bitmap in ECB, le aree di colore uniforme (es. lo sfondo bianco o la pancia del pinguino) generano blocchi cifrati identici. L'immagine cifrata mostra ancora chiaramente i contorni della figura originale. **ECB non garantisce la sicurezza semantica.**

4 Initialization Vector (IV) e Randomizzazione

Per ottenere la sicurezza semantica (IND-CPA), dobbiamo garantire che cifrare due volte lo stesso messaggio produca due ciphertext diversi.

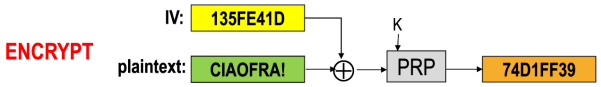


Come visto negli Stream Ciphers, introduciamo un **Initialization Vector (IV)**. L'IV deve essere un valore casuale (nonce) che non si ripete mai per la stessa chiave.

> **Nota**
>
> Ricordare la differenza con SHA-256: qui gli IV sono **CASUALI**, in SHA-256 gli IV sono **COSTANTI**.

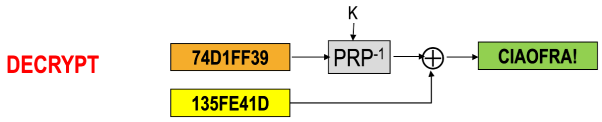
Lo schema di questo approccio è il seguente:

1. Eseguo lo XOR $\oplus$ del mio plaintext con il valore IV, passo poi il risultato alla funzione PRP con chiave K, e ottengo il ciphertext corrispondente.



2. Per trasmettere il messaggio semplicemente inviamo sia il ciphertext appena generato sia il valore IV che abbiamo usato.

3. Per decifrare il messaggio appena ricevuto, prendo il ciphertext e lo passo all'**inversa** della funzione PRP, sempre con chiave K, e del risultato ne faccio lo XOR insieme al valore di IV ricevuto; così facendo riottengo il plaintext originale.



Ora, la cosa su cui dobbiamo prestare attenzione è che gli IV NON si devono ripetere, ma devono essere ANCHE non-predictabili (ovvero un valore random **vero**, che mai si ripete).

Rivedi esempio/esercizio a 0.39.34 Lezione 9

5 Modalità di Operazione Sicure

Abbiamo visto quindi che con i block cipher ci sono 2 grossi problemi: dividere il messaggio quando la sua dimensione supera quella dei blocchi (es. 128 in AES), e garantire che ad ogni cifratura dello stesso messaggio si ottenga un ciphertext diverso.

Per venire in soccorso ai block cipher standard, sono state inventate varie metodologie, chiamate "Modalità di Operazione", che adesso vedremo.

Perché sicure? Perché permettono di garantire la sicurezza semantica, ovvero rispettano e garantiscono la **IND-CPA**.

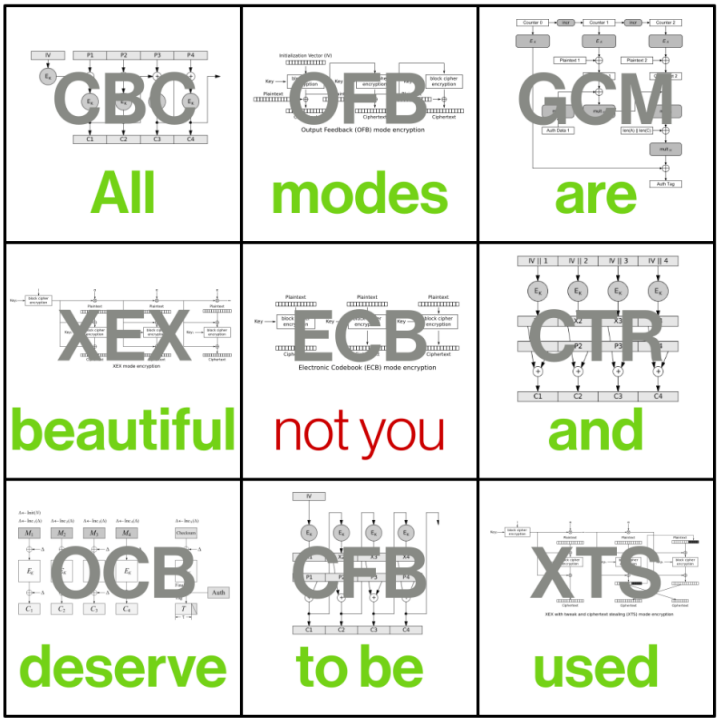


Figura 4: Panoramica Modalità

I più usati sono:

- Cipher Block Chaining (CBC)
- Counter Mode (CTR)

I raccomandati dal NIST, che però non vengono molto usati nella pratica sono:

- Cipher Feedback Mode (CFB)
- Output Feedback Mode (OFB)

Altri che garantiscono altre proprietà più avanzate (quali confidentiality + integrity) sono:

- Galois Counter Mode (GCM)
- Offset Codebook Mode (OCB)
- ecc...

La differenza sostanziale fra le prime due classi e l'ultima è che le prime due garantiscono la IND-CPA, mentre l'ultima garantisce la **IND-CCA** (Chosen Ciphertext Attack), una versione più forte di sicurezza che protegge anche contro attaccanti attivi.

5.1 CBC (Cipher Block Chaining)

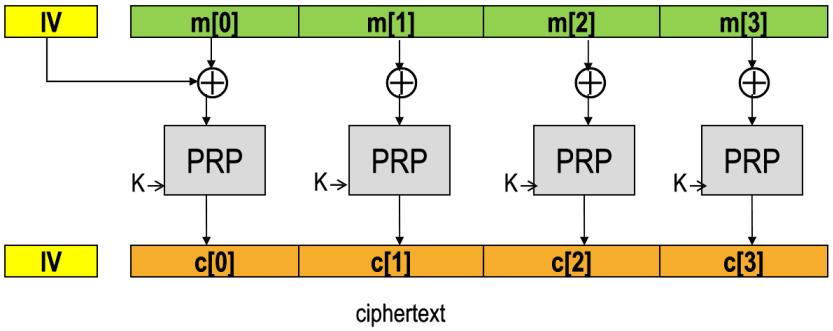
È la modalità storicamente più usata. L'idea è "concatenare" i blocchi in modo che la cifratura del blocco corrente dipenda da tutti i precedenti.

Encryption

Prima di cifrare il blocco P_i, facciamo lo XOR con il testo cifrato precedente C_{i-1}.

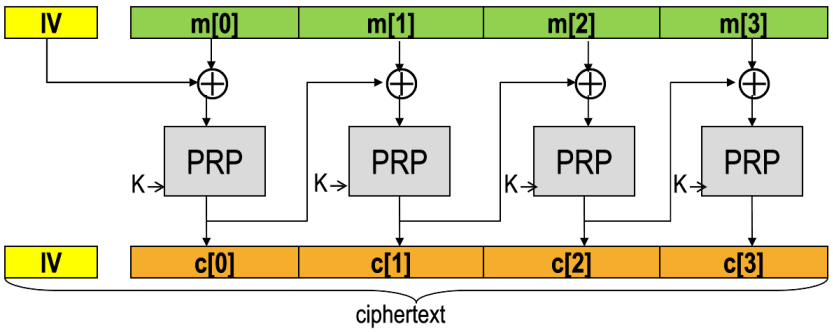
Per il primo blocco vale la seguente operazione:

$$C_0 = \text{ENC}(K, P_0 \oplus IV)$$



Per ogni altro blocco vale questo:

$$C_i = \text{ENC}(K, P_i \oplus C_{i-1})$$

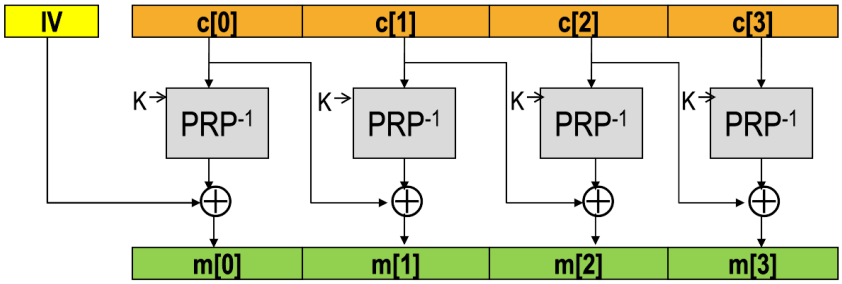


In questo modo, anche se $P_1 = P_2$, l'input al cifrario sarà diverso perché $C_0 \neq C_1$ (grazie all'IV iniziale e alla diffusione).

Decryption

Per ogni blocco vale la seguente operazione:

$$P_i = \text{DEC}(K, C_i) \oplus C_{i-1}$$



Pro e Contro di CBC

- (+) **Sicurezza:** Standard robusto se l'IV è casuale e imprevedibile. Altrimenti, poter predire gli IV porta ad attacchi di tipo CPA, come è avvenuto in TLS (TLS Beast Attack).
- **(-) Sequenziale:** La cifratura **NON** è parallelizzabile. Devi aver calcolato C_1 prima di calcolare C_2. Questo rallenta l'hardware ad alte prestazioni.
- **(-) Padding:** Richiede che il messaggio sia multiplo della dimensione del blocco. Serve uno schema di padding (es. PKCS#7). Questo espone al *Padding Oracle Attack*.
- **(+-) Overhead:** Abbiamo solamente l'IV iniziale che è in più.
- Un altro problema è che cifratura e decifratura richiedono due circuiti diversi (Enc = PRP, Dec = PRP^{-1}). Le altre modalità, come CTR, usano PRP sia per Enc che per Dec.
- **Nota:** La decifratura è parallelizzabile (poiché conosci già tutti i C_i).

5.2 CTR (Counter Mode)

La modalità CTR trasforma un cifrario a blocchi in uno stream cipher. È la modalità preferita nei protocolli moderni (es. AES-GCM in TLS 1.3).

Funzionamento

Non cifriamo direttamente il messaggio. Cifriamo un "Contatore" (che include l'IV) per generare un keystream, che poi viene messo in XOR col messaggio.
I passaggi sono quindi:

- 1) Inizializziamo un contatore ctr, e lo incrementiamo ad ogni blocco
- 2) "cifriamo" il contatore con il block cipher (indipendente dal plaintext, può essere precomputato)
- 3) Effettuiamo lo XOR con il plaintext

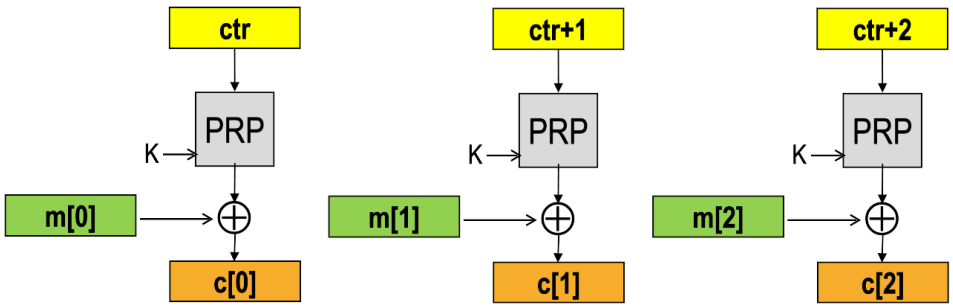


Figura 5: Schema CTR

Nella pratica quindi stiamo costruendo un keystream pseudo-random partendo da un blocco PRP.
Per questa modalità, la sicurezza è dimostrabile: infatti se PRP è sicuro allora anche PRNG è sicuro, e questo basta

$$\text{Keystream}_i = \text{ENC}(K, \text{Nonce}||\text{Counter}_i)$$
$$C_i = P_i \oplus \text{Keystream}_i$$

Il contatore viene incrementato per ogni blocco.
In RFC3689 hanno standardizzato alcuni dettagli su ICB (Initial Counter Block):

- i primi 96 bits sono gli IV
- i successivi 32 bits sono il counter, che parte da 1

AES-CTR è lo standard "de facto" se si vuole fare ***SOLAMENTE*** cifratura Se si vuole fare anche Autenticazione allora bisogna usare AES-GCM (più avanti)

Pro e Contro di CTR

- (+) **Unione:** Questa procedura combina i vantaggi sia di CFB che di OFB in un'unica procedura, e inoltre trasforma il blockcipher in uno streamcipher
- (+) **Performance:** Sia cifratura che decifratura sono **completamente parallelizzabili**. Possiamo pre-calcolare il keystream prima ancora che arrivi il messaggio.
- (+) **Accesso Casuale:** Possiamo decifrare l'ultimo blocco del file senza decifrare quelli prima.
- (+) **Niente Padding:** Poiché agisce come uno stream cipher (XOR), non serve padding; il ciphertext ha la stessa lunghezza del plaintext.
- (+) **Sicurezza:** I contatori (se propriamente usati) non si ripetono, di conseguenza la predicibilità ***NON*** è un problema; Garantisce inoltre che non avvengano problemi di cicli corti

Attenzione

Attenzione: Mai riutilizzare la coppia $(Key, Nonce)$. Come per l'OTP, se il contatore si ripete, la sicurezza crolla catastroficamente.

5.3 CFB (Cipher Feedback) e OFB (Output Feedback)

Queste modalità trasformano il block cipher in stream cipher usando un registro a scorrimento.

- **OFB:** L'output del cifrario diventa l'input per il blocco successivo. Il keystream è indipendente dal messaggio. Genera un flusso sincrono.

– *Problema:* Se l'IV è sfortunato, si può entrare in un "ciclo corto" (short cycle), ripetendo il keystream troppo presto.

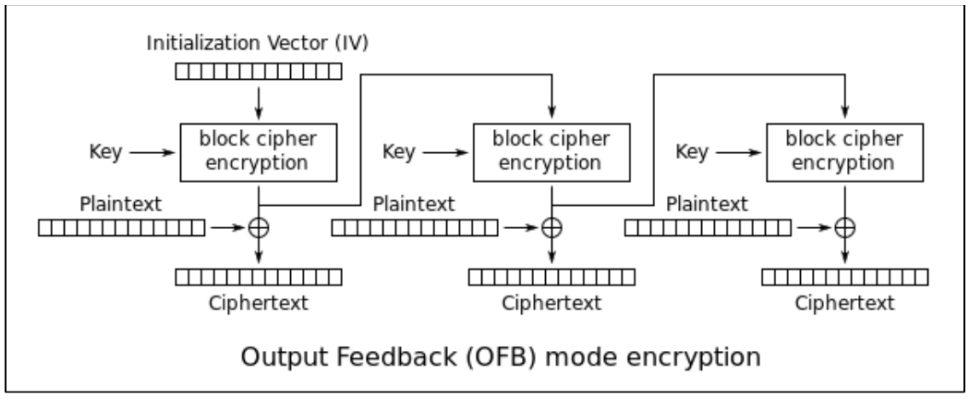


Figura 6: Schema modalità OFB (Output Feedback Mode)

– La concatenazione di tutti i block cipher crea la nostra keystream, e quindi possiamo semplicemente scordarci i plaintext; infatti se rimuoviamo tutti i plaintext, possiamo prendere le chiavi e i block cipeher e creare una **catena di cose** che altro non è che una **sequenza pseudo-random**

- **CFB:** Il *ciphertext* precedente diventa l'input per il blocco successivo. È "auto-sincronizzante" (se si perde un pezzo di ciphertext, l'errore si propaga solo per pochi blocchi poi si ristabilisce).

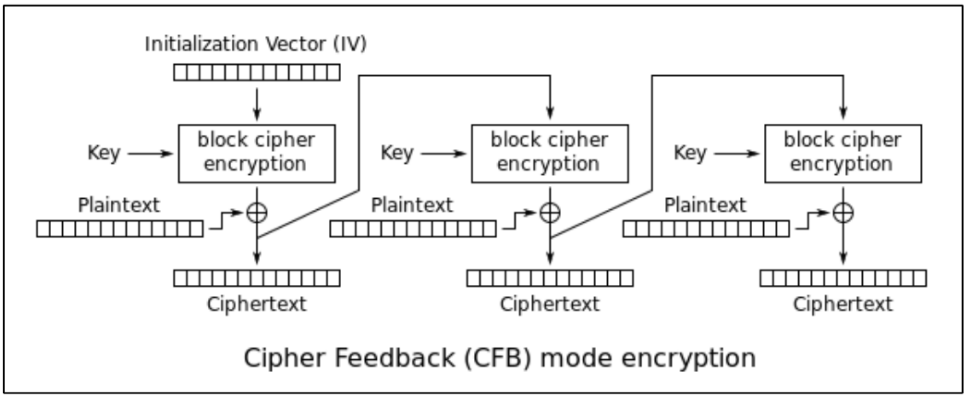


Figura 7: Schema modalità CFB (Cipher Feedback Mode)

CFB, così come CBC, dipende dai ciphetexts precedenti; mentre il keystream di OFB dipende solamente dagli IV
Analisi CFB:

- Così come CBC, non è paralellizabile in encryption, ma è parellalizzabile in decryption

- Ma così come CBC; CFB dipende dai ciphertexts precedenti

Analisi OFB:

- OFB non può essere parallelizzato ne in encryption, ne in decryption (la procedura di decryption è quindi seriale, e bisogna passare per ogni blocco, a differenza di CBC dove si può parallelizzare tutto)
- I dati però possono essere precomputati in fase di ecnryption, accellerando di fatto la procedura

Vediamo la modalità di decifratura:

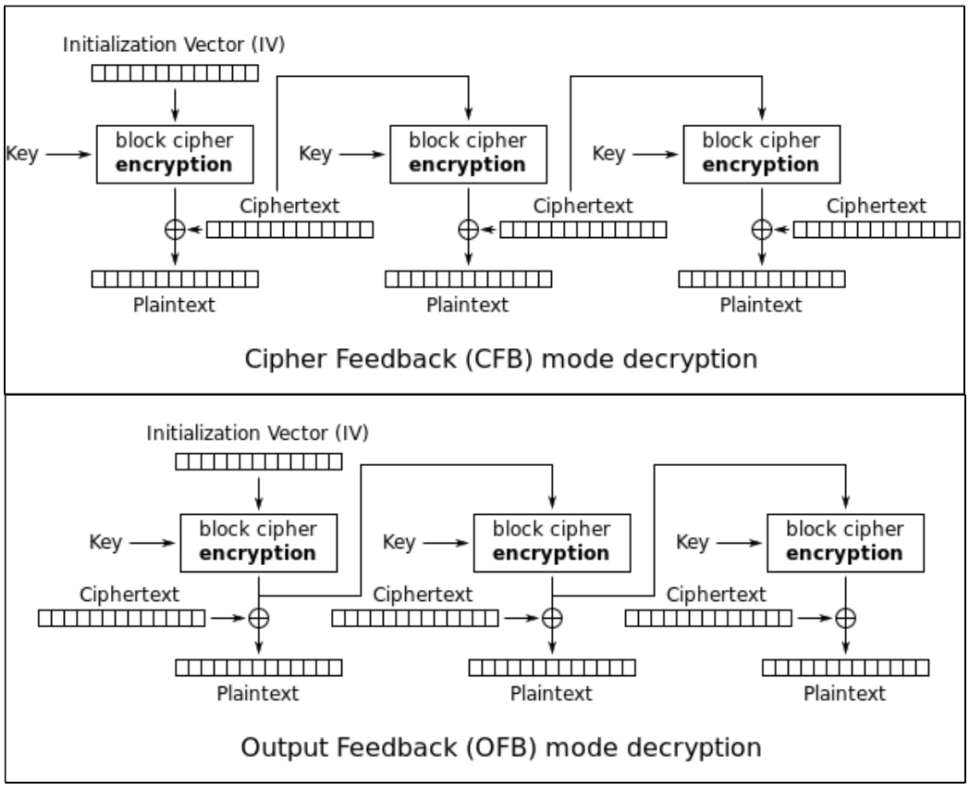


Figura 8: CFB e OFB in modalità Decifratura

Possiamo notare un'altro vantaggio di CFB rispetto a OFB in fase di decifratura, infatti vediamo che CFB non ha bisogno di un blocco a parte solo per la decifratura, e di conseguenza non ha bisogno di usare l'inversa di PRP.
Possiamo quindi effettuare le operazioni di enc e dec usando sempre gli stessi blocchi!! Supponiamo che il nostro sistema sia improntato ad avere errori sul canale, e che ogni blocco sia un pacchetto, che modalità preferiresti usare? CFB o OFB? e perchè?

Risposta:

- se avviene un errore, ad un certo punto della cifratura usando CFB, l'errore si propagherà per tutti i blocchi successivi, annullando di fatto l'itero messaggio
- se invece avviene un errore usando OFB, non c'è nessun problema di propagazione - non possiamo recuperare il blocco, ma fintanto che gli IV sono corretti possiamo comuque continuare a generare tutta la cifratura restante

Di conseguenza, la modalità migliore in questo caso è la OFB

Attenzione

Attenzione: Per le modalità CBC,CFB e OFB non esiste un teorema che dimostra la sicurezza; per queste modalità non esiste quindi una dimostrazione di sicurezza effettiva.

6 Autenticazione con BlockCipher: Che modalità uso?

Fino ad ora abbiamo utilizzato i cifrari a blocchi per garantire la *Confidenzialità*.
Tuttavia, è possibile utilizzare queste primitive anche per l'autenticazione, tipicamente in schemi *Challenge-Response* (come il protocollo CHAP).
Il problema fondamentale è: come faccio a provare a un interlocutore (Authenticatore) che conosco un segreto K, senza rivelare il segreto stesso?
L'approccio standard prevede che l'Authenticatore invii una "Challenge" (una stringa casuale, un nonce) e il dispositivo debba rispondere cifrando quella challenge con la chiave segreta condivisa.

$$\text{Response} = \text{ENC}_K(\text{Challenge})$$

Secondo i manuali di crittografia (es. Menezes, cap 10.3.2), questo approccio è sicuro purché la challenge sia casuale e non si ripeta mai.
Tuttavia, la scelta della Modalità di Operazione è critica e controintuitiva rispetto a quanto visto per la cifratura dati.

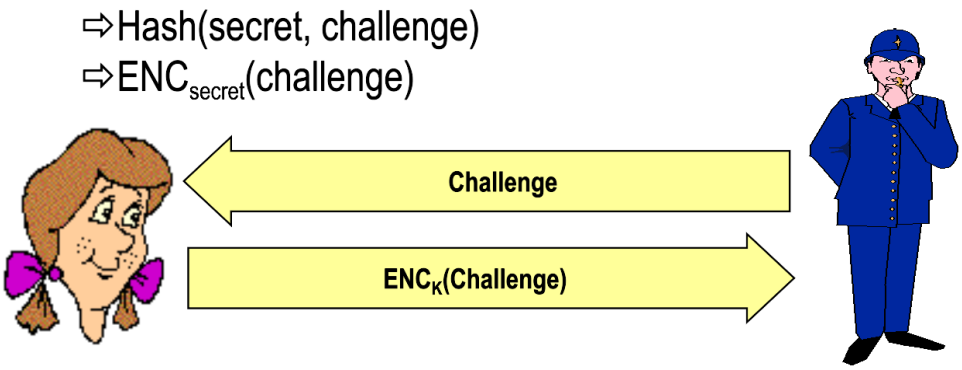


Figura 9: Schema Autenticazione con CHAP

6.1 Il Caso di Studio WEP e la Vulnerabilità degli Stream Ciphers

Per capire quale modalità scegliere, analizziamo il fallimento del protocollo WEP (WiFi), che utilizzava l'algoritmo RC4 (uno Stream Cipher) per l'autenticazione.

In uno Stream Cipher (o in modalità come AES-CTR), la cifratura avviene tramite XOR:

$$C = P \oplus \text{Keystream}$$

Nel contesto dell'autenticazione:

- P è la Challenge (pubblica, inviata in chiaro dall'Authenticatore).

- C è la Response (intercettabile dall'attaccante).

Un attaccante che osserva lo scambio si trova in una situazione di *Known Plaintext Attack*. Poiché conosce sia P che C, può calcolare banalmente il Keystream:

$$\text{Keystream} = P \oplus C$$

Una volta posseduto il keystream (assumendo che l'IV venga riutilizzato o possa essere forzato dall'attaccante), l'attaccante può forgiare una risposta valida per qualsiasi **nuova** challenge futura, senza conoscere la chiave K.

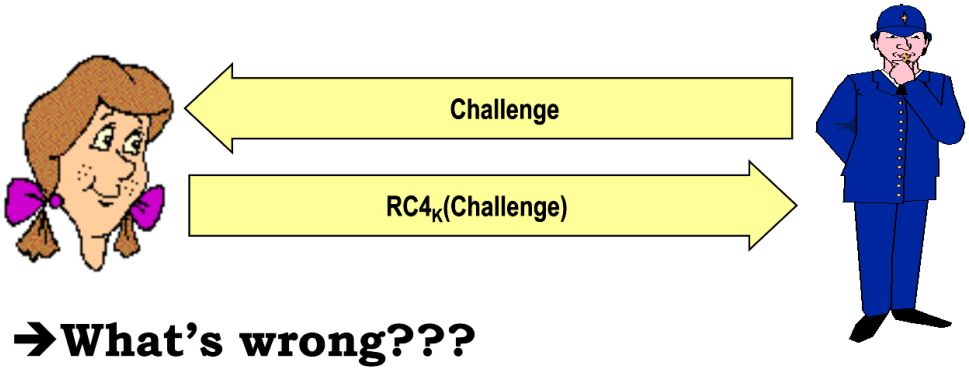


Figura 10: Recap Schema in WEP

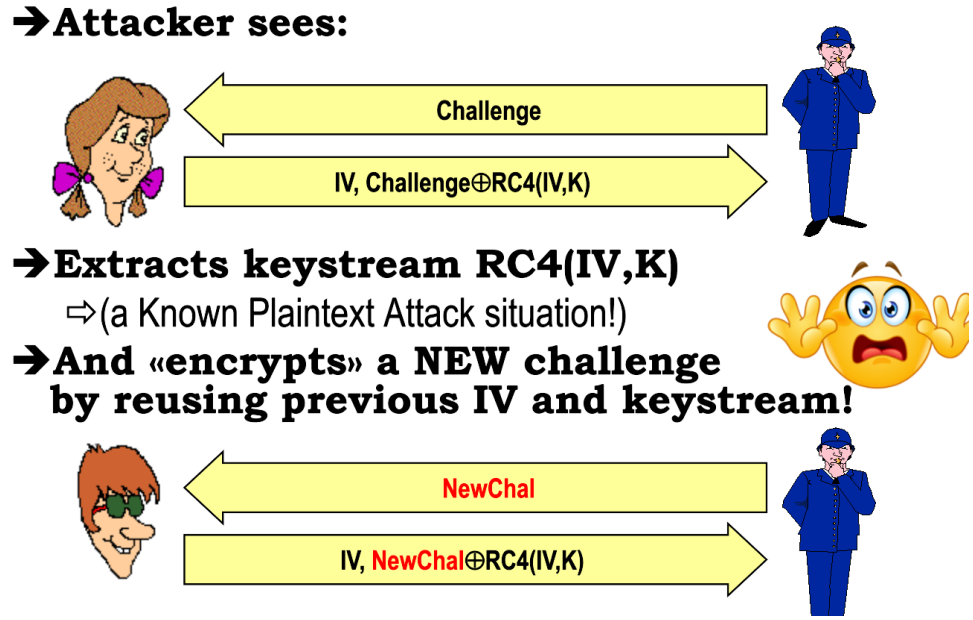


Figura 11: Problema di WEP

6.2 Analisi delle Modalità AES per l'Autenticazione

Se decidiamo di sostituire l'insicuro RC4 con AES, quale modalità dovremmo usare?

6.2.1 Il Problema di AES-CTR (e CBC)

Si potrebbe pensare che AES-CTR, essendo la modalità migliore per la confidenzialità, sia ottima anche qui. In realtà, AES-CTR trasforma il cifrario a blocchi in uno stream cipher basato su XOR.

$$\text{Response} = \text{Challenge} \oplus \text{AES}_K(\text{Counter})$$

Se il protocollo permette al dispositivo di scegliere l'IV o il contatore, o se questi vengono riutilizzati, si ricade esattamente nello stesso attacco del WEP:

1. L'Authenticatore invia la Challenge C_1 (es. "123456").
2. Il dispositivo risponde con $R_1 = C_1 \oplus K_{stream}$ (es. "nctuth").
3. L'attaccante calcola $K_{stream} = C_1 \oplus R_1$.
4. L'Authenticatore invia una nuova Challenge C_2 (es. "789012").
5. L'attaccante non conosce K, ma può forgiare la risposta R_2:

$$R_2 = C_2 \oplus K_{stream} = C_2 \oplus (C_1 \oplus R_1)$$

6. L'Authenticatore decifra e valida la risposta come corretta.

Lo stesso problema di "malleabilità" affligge anche AES-CBC se l'IV è manipolabile.

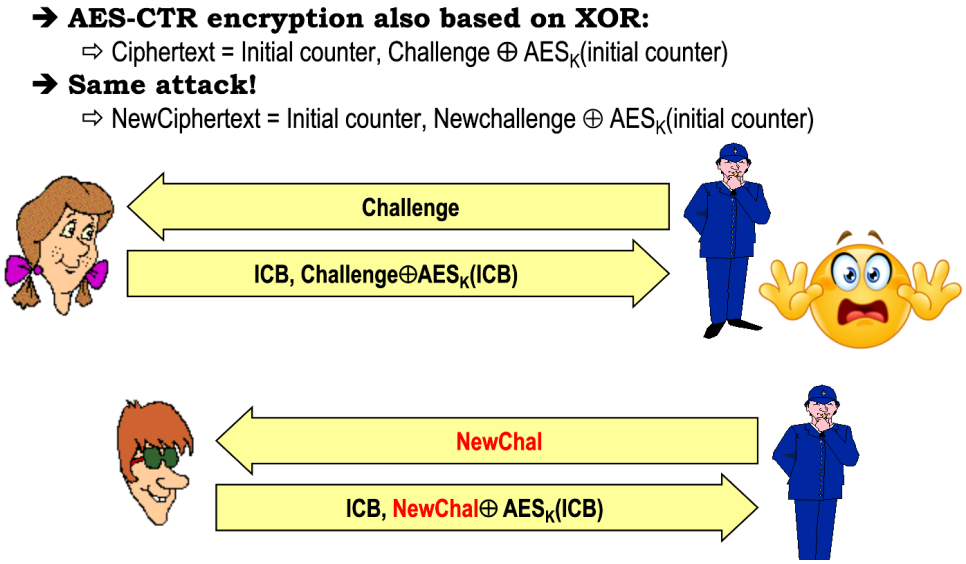


Figura 12: Problema con AES-CTR

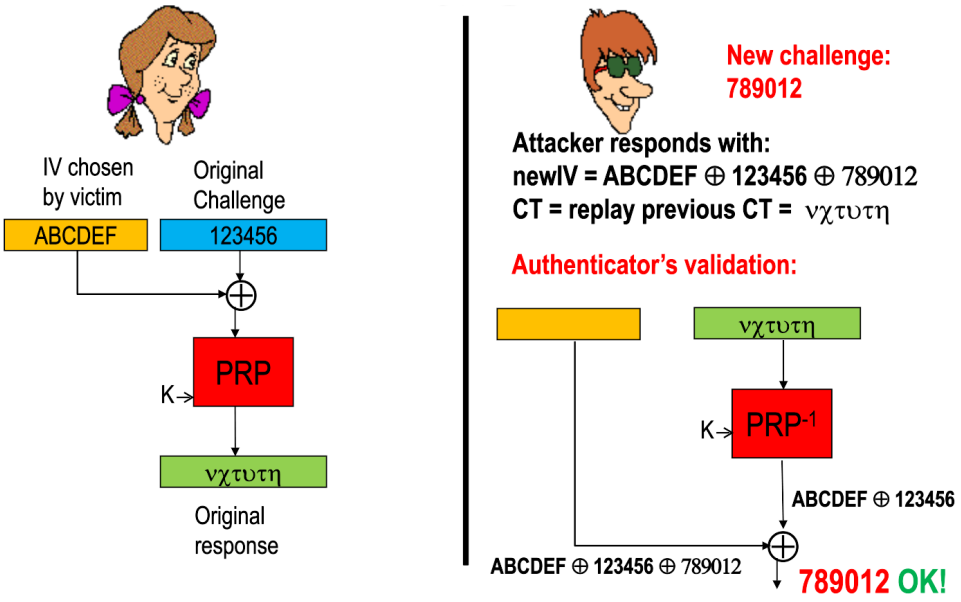


Figura 13: Stesso problema con AES-CBC

6.2.2 Il Paradosso di AES-ECB

Sorprendentemente, la modalità considerata "peggiore" e insicura per la cifratura dei dati, **AES-ECB**, diventa la scelta più sicura per questo specifico tipo di autenticazione Challenge-Response.

Perché AES-ECB funziona qui?

- È deterministico.
- Non c'è IV che l'attaccante possa manipolare o riutilizzare.
- Non è basato su XOR lineare come gli stream cipher.

Fintanto che la Challenge è casuale e non si ripete mai (garantito dall'Authenticatore), l'attaccante non può usare risposte passate per forgiare risposte future, poiché $ENC_K(C_1)$ non aiuta a calcolare $ENC_K(C_2)$ senza la chiave.

6.3 La Regola d'Oro dell'Autenticazione

Da questa analisi deriva una regola fondamentale per la progettazione di sistemi di autenticazione sicuri:

Lasciare ZERO gradi di libertà al dispositivo che si autentica.

Il nonce (Challenge) deve essere deciso interamente dall'Authenticatore. Il dispositivo deve solo eseguire l'operazione crittografica deterministica (come AES-ECB) sulla challenge ricevuta, senza aggiungere parametri extra come IV o contatori che potrebbero essere sfruttati per attacchi di replay o manipolazione del keystream.

7 Padding (Riempimento)

Poiché i cifrari a blocchi (in modalità ECB e CBC) richiedono input di lunghezza esatta (es. multipli di 16 byte), se il messaggio non raggiunge tale lunghezza, dobbiamo aggiungere dati extra.

PKCS#7 Padding È lo standard più comune. Se mancano N byte per completare il blocco, aggiungiamo N byte, ciascuno di valore N.

- Esempio (mancano 4 byte): `... DATI 04 04 04 04`
- Se il messaggio è già allineato, si aggiunge un intero blocco di padding (es. 16 byte di valore 16) per evitare ambiguità in decifratura.

8 Sintesi Comparativa

Modalità	Parallelismo (Enc)	Sicurezza Semantica (IND-CPA)?	Integrità?	Problema Principale
ECB	Sì	**NO**	No	Rivela pattern (Pinguino). Insicuro. **NON USARE MAI**
CBC	No	Sì (con IV random)	No	Lento (sequenziale), Padding Oracle, IV malleabile.
CTR	**Sì**	Sì (con Nonce unico)	No	Catastrofico se il Nonce si ripete. Malleabilità bit-flip.
GCM	Sì	Sì	**Sì**	È CTR + Autenticazione. Lo standard de-facto oggi.

Tabella 1: Confronto delle Modalità di Operazione

Conclusione

Per la sola confidenzialità ad alte prestazioni, **CTR** è la scelta eccellente. Se serve compatibilità legacy, CBC è accettabile (con IV corretti). Per sistemi moderni, si preferiscono modalità autenticate come **GCM** (Galois Counter Mode) che combinano CTR con un MAC per garantire sia confidenzialità che integrità.