

Sicurezza (Prima parte del corso) – Teoria

Indice

1. [Cifrari](#)
 - a. [Tipologie di Cifrari](#)
 - i. [Cifrario Aperto](#)
 - ii. [Cifrario Chiuso](#)
2. [Crittografia Simmetrica](#)
3. [Data Encryption Standard \(DES\)](#)
 - a. [Cifrario di Feistel](#)
 - b. [Specifiche del DES](#)
 - c. [Key Schedule](#)
4. [AES](#)
5. [Double DES e Triple DES](#)
6. [Cifrari a blocchi e cifrari a flusso](#)
 - a. [Electronic Codeblock \(ECB\)](#)
 - b. [Cipher Block Chaining \(CBC\)](#)
 - c. [Cipher Feedback \(CFB\)](#)
 - d. [Output Feedback \(OFB\)](#)
7. [Diffie-Hellman Protocol](#)
 - a. [Problema dello scambio di chiavi condivise](#)
 - b. [Richiami di Aritmetica Modulare](#)
 - i. [Operazione Modulo](#)
 - ii. [Congruenza in Modulo](#)
 - iii. [Proprietà \(identità\)](#)
 - c. [Come funziona Diffie-Hellman](#)
 - d. [Algoritmi Efficienti per Diffie-Hellman](#)
 - i. [Algoritmo efficiente per l'esponente modulare](#)
 - ii. [Algoritmo efficiente per generare un numero primo](#)
 - iii. [Algoritmo efficiente per generare una radice primitiva](#)
 1. [Interi Coprimi e funzione phi di eulero](#)
 2. [Teorema di Eulero](#)
8. [RSA](#)
 - a. [Teorema di Eulero \(moltiplicativo\)](#)
 - b. [Correttezza di RSA](#)
 - c. [Sicurezza di RSA](#)
 - d. [Efficienza di RSA](#)
 - i. [Algoritmo efficiente per l'inverso Moltiplicativo](#)
 1. [Algoritmo di Euclide](#)
 2. [Algoritmo di Euclide Esteso \(EEA\).](#)
9. [Hash](#)
 - a. [Hash Collision](#)
 - b. [Attacco Del Compleanno](#)
 - c. [Più sicurezza in Hash](#)
10. [Autenticazione](#)
 - a. [Autenticazione Simmetrica](#)
 - i. [MAC-CBC](#)
 - ii. [HMAC](#)

- b. Principio di non disconoscibilità
- c. Firma Elettronica
- d. Problema di distribuzione di chiavi pubbliche
- e. Certificato a chiave pubblica

Cifrario

In crittografia, un cifrario è un algoritmo per eseguire la crittografia o la decrittografia: una serie di passaggi ben definiti volte a rendere oscuro, ossia semanticamente non leggibile, un testo di un messaggio in chiaro (plain text) o, al contrario, al ripristino in chiaro di un messaggio precedentemente cifrato.

La differenza tra Cifrario, crittografia e cifratura/decifratura, è che:

- Il Cifrario è l'algoritmo o il processo utilizzato per crittografare i dati (ad es. AES, RSA, ecc.).
- Cifratura/decifratura è il processo di conversione dei dati utilizzando un suddetto cifrario (ad es. Cifro usando AES)
- La Crittografia è una scienza: è la branca dell'informatica che tratta delle "scritture nascoste", ovvero dei metodi per rendere un messaggio non comprensibile/intelligibile a persone non autorizzate a leggerlo, garantendo così, in chiave moderna, il requisito di confidenzialità o riservatezza tipico della sicurezza informatica.

Tipologie di cifrari

Nella storia si sono usati diversi tipi di cifrari, ognuno con le proprie particolarità. È possibile però aggregare i cifrari esistenti in base al fatto che questi siano chiusi o aperti.

Per **cifrario chiuso** si intende che il loro funzionamento è oscuro al pubblico/all'esterno: Questi cifrari adottano la filosofia del **"Security through obscurity"**:

- Questo principio si basa sull'assunto che tenere segreto il funzionamento interno di un sistema o di un componente lo renda più sicuro dato che un eventuale attaccante avrebbe maggiore difficoltà nella scoperta di eventuali falle del sistema stesso.

Il principio di **"Security through obscurity"**, **è stata una filosofia molto contestata nel corso della storia.**

Per la filosofia applicata ai **cifrari aperti** invece, **si utilizza un principio opposto** (detto principio di Kerckhoffs), **che afferma che il progettista di un sistema deve presumere che l'attaccante conosca il sistema alla perfezione, con l'unica eccezione della chiave crittografica.** Un parallelo illuminante può essere quello di una normale serratura meccanica, con azionamento ben conosciuto ma apribile solo con la chiave giusta.

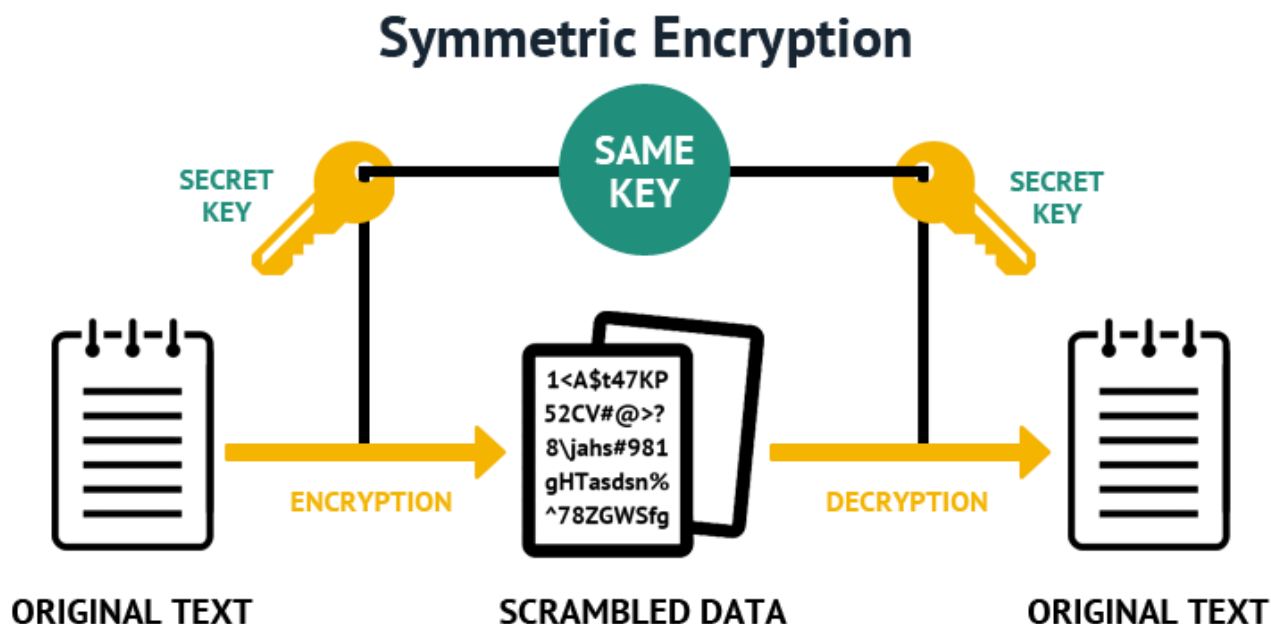
Ciò significa che nei cifrari aperti **tutto il funzionamento è conosciuto** ma, **pur sapendolo, senza la chiave giusta è impossibile risalire al testo/messaggio in chiaro.**

La filosofia dei cifrari aperti si appoggia al conetto dell'open source, la quale sostiene invece che, grazie alla vasta collaborazione della comunità, **le falle possono venir scoperte più facilmente e altrettanto rapidamente corrette**, con il risultato di rendere il sistema intrinsecamente più sicuro. **Questo non accade con i cifrari chiusi.**

La sicurezza tramite segretezza è scoraggiata e non raccomandata dagli enti normativi. Il National Institute of Standards and Technology (NIST) negli Stati Uniti sconsiglia espressamente questa pratica: "La sicurezza del sistema non dovrebbe dipendere dalla segretezza dell'implementazione o dei suoi componenti."

Crittografia Simmetrica

La crittografia simmetrica è una **tecnica di cifratura che utilizza la stessa chiave crittografica sia per la crittografia del testo in chiaro che per la decrittografia del testo cifrato**. Le chiavi rappresentano un segreto condiviso tra due o più parti che può essere utilizzato per mantenere un collegamento di informazioni private.



Di solito si mantiene solo l'iniziale della parola "Key" (K) per indicare che un oggetto è una chiave.

Per **testo in chiaro** (plaintext) si indica il messaggio prima che su di esso venga applicato un qualche algoritmo crittografico.

Per **testo cifrato** (ciphertext) si indica il risultato che si ottiene applicando un algoritmo crittografico su testo in chiaro e quindi reso illeggibile da un essere umano.

Data Encryption Standard (DES)

Il DES è un cifrario che **si basa su un algoritmo crittografico a chiave simmetrica** a 64 bit (56 vengono utilizzati, poiché i restanti 8 sono di controllo), infatti, **con il tempo è stato considerato come insicuro**. Questa insicurezza appunto deriva dalla lunghezza della chiave (Nel gennaio del 1999 distributed.net ed Electronic Frontier Foundation collaborarono per rompere pubblicamente una chiave di crittazione, e ci riuscirono in 22 ore e 15 minuti).

Il DES è l'antenato dei cifrari a blocchi: Un cifrario a blocchi opera su un gruppo di bit di lunghezza finita organizzati in un blocco. A differenza degli algoritmi a flusso che cifrano un singolo elemento alla volta, gli algoritmi a blocco cifrano un blocco di elementi contemporaneamente.

Per spiegare il funzionamento del DES è necessario introdurre il Cifrario di Feistel.

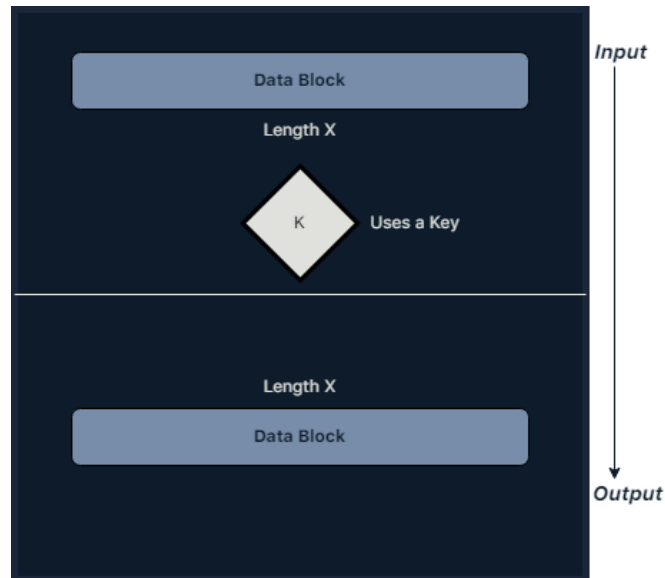
Il Cifrario di Feistel è una struttura di cifratura utilizzata per costruire cifrari a blocchi (Appunto come DES).

Funzionamento del Cifrario di Feistel

Il cifrario accetta due input:

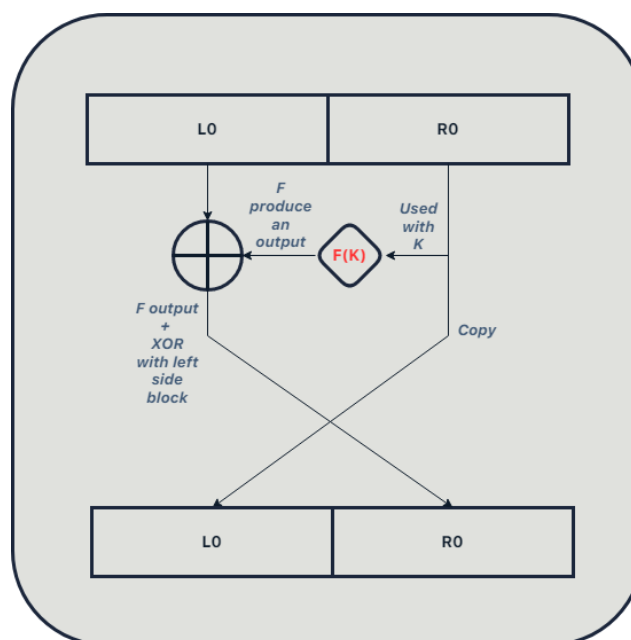
- un blocco dati
- una sottochiave

Restituisce un'uscita della stessa dimensione del blocco dati in ingresso.



Ciò che succede nello specifico è: **Il blocco in input, viene diviso in due parti uguali. La parte più a destra rimane immutata** durante il processo di cifratura **e diventerà poi la parte sinistra del blocco di output**, mentre la parte di sinistra viene manipolata in questo modo:

- Si prende la parte destra del blocco, e si esegue una funzione di cifratura su quel blocco utilizzando la chiave K data in input.
- Il risultato della cifratura eseguita sul blocco destro utilizzando la chiave K viene messa in XOR con la parte sinistra del blocco.
- Il risultato dello XOR applicato alla parte cifrata destra produce il restante blocco dell'output (che sarà ora la parte destra)



Se si vuole decifrare il blocco, è sufficiente conoscere la chiave ed eseguire i passi al contrario: Il blocco L0 del blocco cifrato viene copiato così com'è, mentre il blocco R0 del blocco cifrato viene dato in pasto alla funzione F(K) e messo in XOR con la parte sinistra L0 del blocco cifrato.

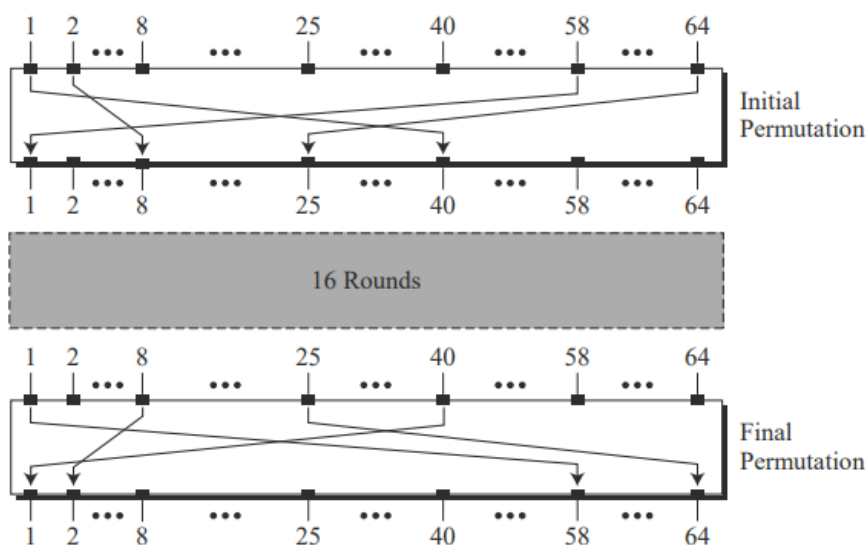
Specifiche del Des

Ora che sappiamo come funziona il Feistel Cipher, possiamo definire il funzionamento del DES.

Il DES utilizza come principio quello del Feistel Cipher ma ripetuto N volte. Nel caso del DES, ci sono 16 fasi identiche di elaborazione (fasi di Feistel), denominate round.

Prima di eseguire i 16 round, viene eseguita una permutazione iniziale dei bit del blocco (IP: Initial Permutation). Successivamente ai 16 round viene seguita anche qui una permutazione dei bit del blocco (FP: Final Permutation). Queste permutazioni sono standard per ogni volta che il DES viene applicato (il bit in posizione 58 viene scambiato sempre con quello in posizione 4).

Queste due permutazioni non hanno significato crittografico in DES a livello di sicurezza. Entrambe le permutazioni sono keyless e predeterminate. Il motivo per cui sono inclusi in DES non è chiaro e non è stato rivelato dai designer di DES. L'ipotesi è che DES sia stato progettato per essere implementato nell'hardware (su chip) e che queste due complesse permutazioni possano ostacolare una simulazione software del meccanismo.



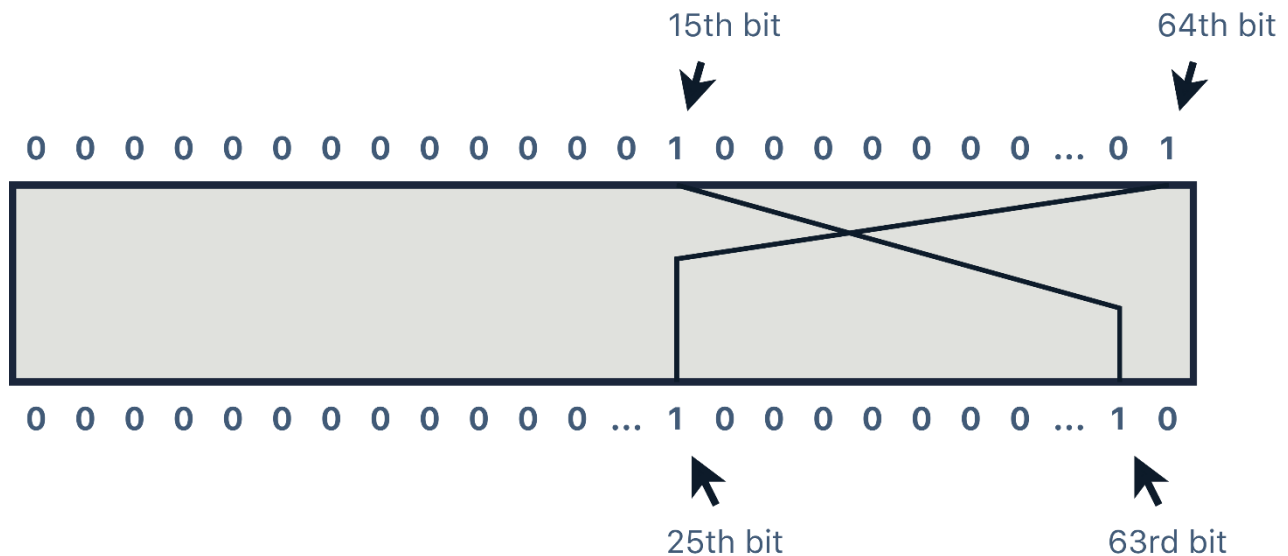
Le permutazioni seguono queste tabelle di permutazione

Initial Permutation							
58	50	42	34	26	18	10	02
60	52	44	36	28	20	12	04
62	54	46	38	30	22	14	06
64	56	48	40	32	24	16	08
57	49	41	33	25	17	09	01
59	51	43	35	27	19	11	03
61	53	45	37	29	21	13	05
63	55	47	39	31	23	15	07

Final Permutation							
40	08	48	16	56	24	64	32
39	07	47	15	55	23	63	31
38	06	46	14	54	22	62	30
37	05	45	13	53	21	61	29
36	04	44	12	52	20	60	28
35	03	43	11	51	19	59	27
34	02	42	10	50	18	58	26
33	01	04	09	49	17	57	25

Le tabelle di permutazione sono array ma per una lettura più compatta si rappresentano in formato tabellare.

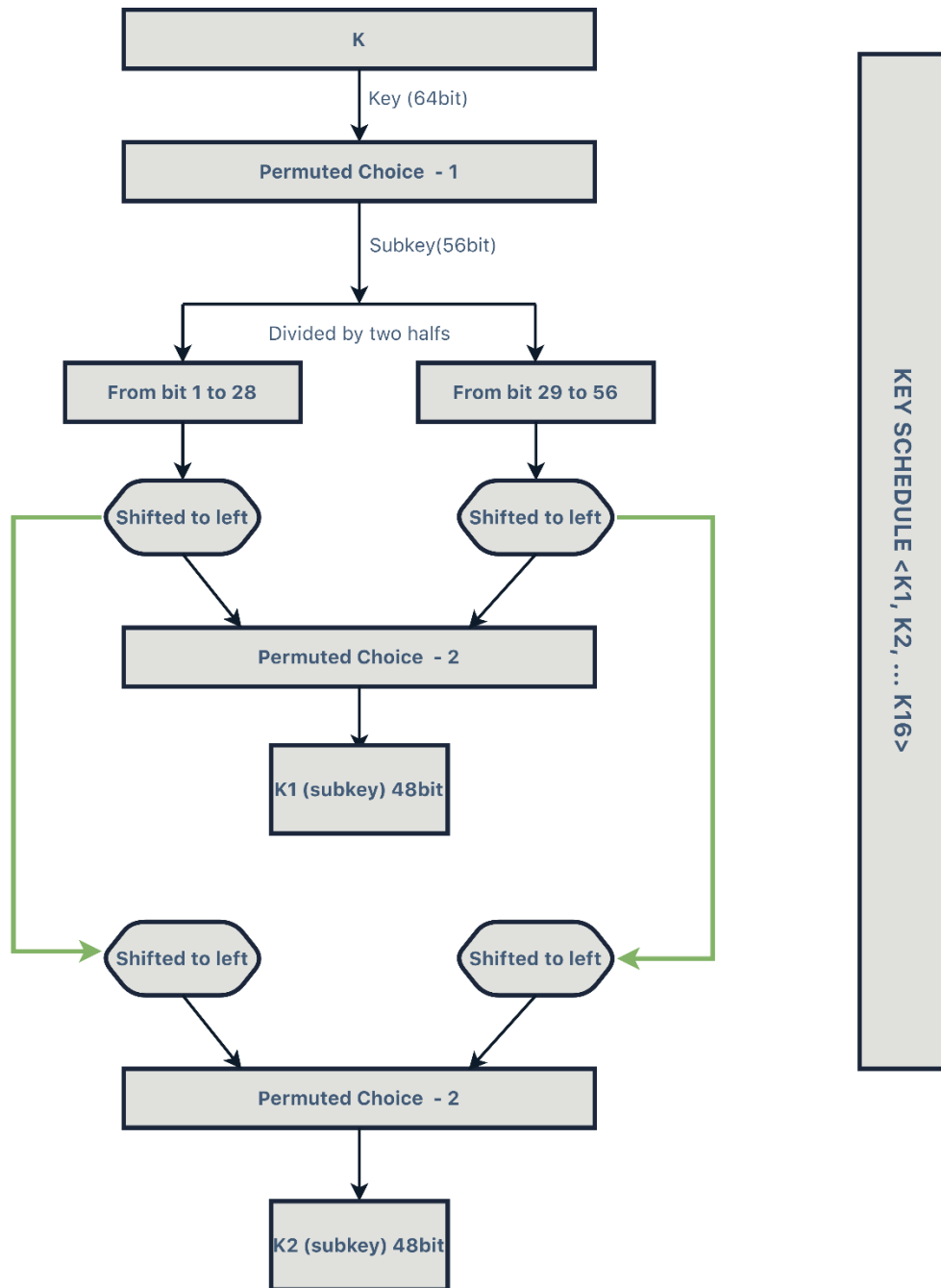
Ad esempio: trovare l'output dell'initial permutation del blocco 0x0002 0000 0000 0001



Il quindicesimo bit (guardando la tabella Final permutation) corrisponde al bit 63 ed il bit 64 corrisponde al bit 25.

Come visto nel Feistel Cipher, c'è bisogno di una chiave K che verrà utilizzata dalla funzione F per cifrare metà blocco. Anche nel DES viene utilizzata una chiave per ogni round ma questa non è uguale per ogni round: il cifrario utilizza una chiave iniziale che poi ad ogni round verrà rielaborata (si parla di **KEY SCHEDULE**):

- La chiave di input viene elaborata secondo la tabella di permutazione PC-1 che produrrà una chiave da 56 bit
- La chiave da 56 bit ottenuta, viene divisa in due metà. Ad ognuna delle metà viene effettuato uno shift circolare a sinistra (di 1 o 2)
- Dopo lo shift, le due metà si riassemblano per essere permutate nuovamente secondo la tabella di permutazione PC2
- L'output di PC-2 è la chiave che verrà usata in quel round di DES
- Per il calcolo della sottochiave successiva, si passano le due parti shiftate (prima dell'applicazione del PC-2). L'output del PC-2 è solo la chiave che viene usata nel round del DES, non viene usata come input per il calcolo della sottochiave successive.



Il passo di permutazione PC-1 genera una chiave di 56 bit a partire dalla chiave principale di 64bit (**questo solo al primo passo, poiché la chiave intera da 64 viene permutata solo al primo round**), seguendo la tabella di permutazione seguente:

57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

Da notare che la tabella è di appunto 56 posizioni.

Si legge come le altre tabelle di permutazione: il bit in posizione 57 della chiave K sarà il bit 1 nella chiave K+, il bit 49 della chiave K sarà il bit in posizione 2 della chiave K+, e così via.

K = 00010011 00110100 01010111 01111001 10011011 10111100 11011111 11110001

K+ = 1111000 0110011 0010101 0101111 0101010 1011001 1001111 0001111

Da notare anche che nella tabella non sono presenti i multipli di 8 (vengono rimossi quindi i bit in posizione 8, 16, 32, 64).

La quantità di shift poi che si effettuano sulle due metà della chiave dipende a che passo ci si trova (dei 16 totali).

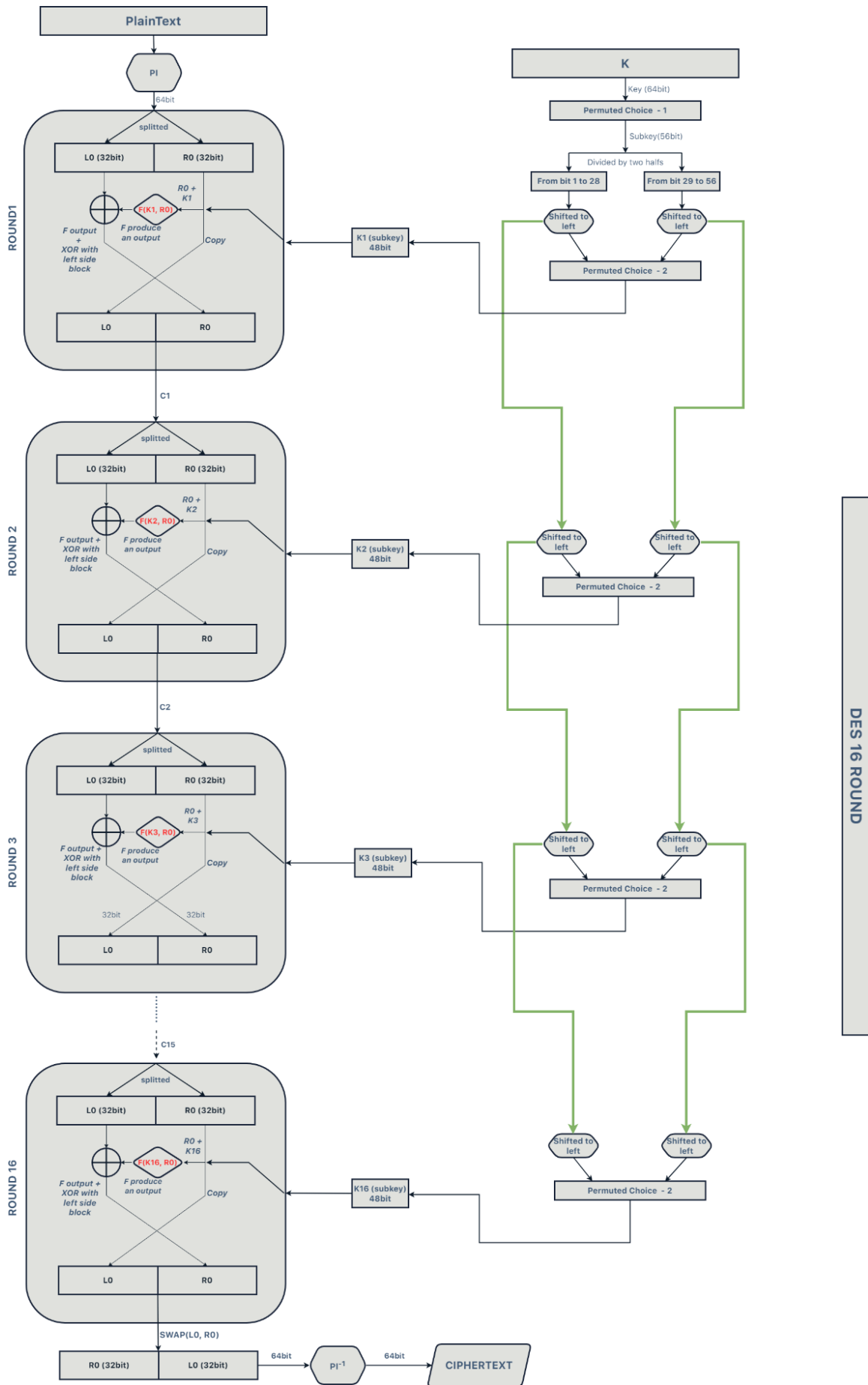
Round	Number of left shift
1	1
2	1
3	2
4	2
5	2
6	2
7	2
8	2
9	1
10	2
11	2
12	2
13	2
14	2
15	2
16	1

Infine, per PC-2 la tabella di permutazione è:

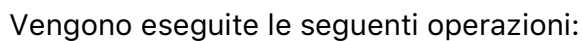
14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

Le tabelle di permutazione sono una cosa in più.. non sono da sapere. Come anche le S-box.

Possiamo riassumere quindi il funzionamento con questo schema:



Tra tutte le cose manca da definire cosa fa questa funzione $F(K_i, R_0)$:



- | | | | | | |
|----|----|----|----|----|----|
| 32 | 1 | 2 | 3 | 4 | 5 |
| 4 | 5 | 6 | 7 | 8 | 9 |
| 8 | 9 | 10 | 11 | 12 | 13 |
| 12 | 13 | 14 | 15 | 16 | 17 |
| 16 | 17 | 18 | 19 | 20 | 21 |
| 20 | 21 | 22 | 23 | 24 | 25 |
| 24 | 25 | 26 | 27 | 28 | 29 |
| 28 | 29 | 30 | 31 | 32 | 1 |

- | S ₅ | | Middle 4 bits of input | | | | | | | | | | | | | | | |
|----------------|----|------------------------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|
| | | 0000 | 0001 | 0010 | 0011 | 0100 | 0101 | 0110 | 0111 | 1000 | 1001 | 1010 | 1011 | 1100 | 1101 | 1110 | 1111 |
| Outer bits | 00 | 0010 | 1100 | 0100 | 0001 | 0111 | 1010 | 1011 | 0110 | 1000 | 0101 | 0011 | 1111 | 1101 | 0000 | 1110 | 1001 |
| | 01 | 1110 | 1011 | 0010 | 1100 | 0100 | 0111 | 1101 | 0001 | 0101 | 0000 | 1111 | 1010 | 0011 | 1001 | 1000 | 0110 |
| | 10 | 0100 | 0010 | 0001 | 1011 | 1010 | 1101 | 0111 | 1000 | 1111 | 1001 | 1100 | 0101 | 0110 | 0011 | 0000 | 1110 |
| | 11 | 1011 | 1000 | 1100 | 0111 | 0001 | 1110 | 0010 | 1101 | 0110 | 1111 | 0000 | 1001 | 1010 | 0100 | 0101 | 0011 |

- Infine, viene effettuata una permutazione

Sebbene siano state pubblicate più informazioni sulla crittoanalisi di DES rispetto a qualsiasi altro cifrario a blocchi, l'attacco più pratico fino ad oggi è ancora un approccio di forza bruta. Il fatto di avere una chiave così ristretta lo rende vulnerabile ad attacchi di forza bruta. Per questo motivo è stato sostituito nel tempo con altri cifrari come **AES**.

L'**Advance Encryption Standard** è un cifrario che utilizza dimensioni di chiave che partono da 128bit e si estende fino a 256bit.

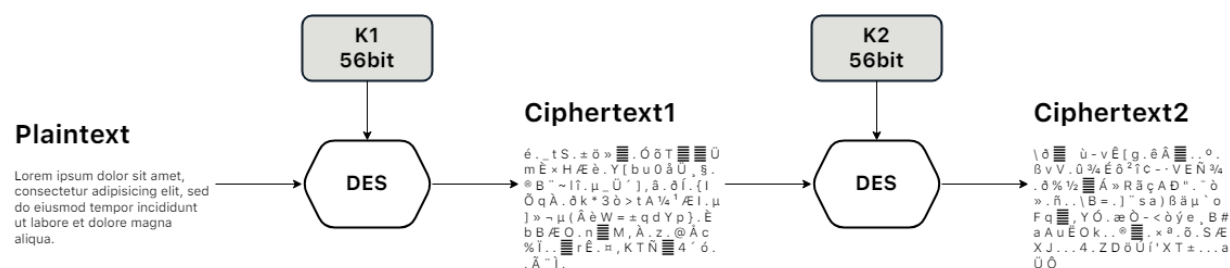
È un cifrario a blocchi anch'esso: blocchi da 128bit.

Non deriva da DES. Questo cifrario è stato adottato come successore del DES ma non deriva da esso. Non utilizza la rete di Feistel, ma una variante del cifrario a blocchi di Rijndael.

Double DES e Triple DES

Il DES applicato una singola volta come si è visto, è molto debole e vulnerabile ad attacchi di forza bruta.

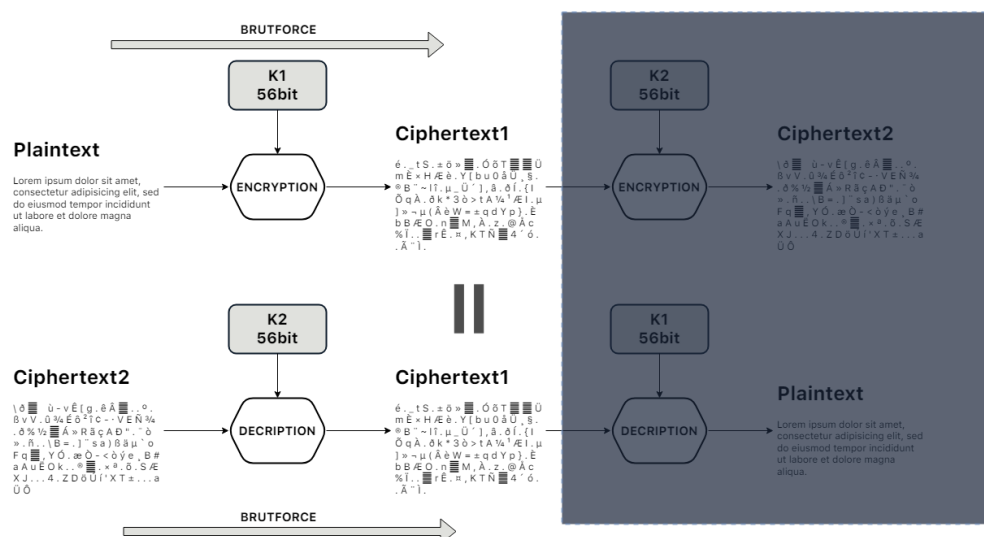
Ciò che si è pensato è di applicare il DES non una volta ma due volte al blocco di dati da cifrare, in modo da raddoppiare la lunghezza della chiave: Utilizzo una chiave per cifrare la prima volta, ed utilizzo un'altra chiave (diversa dalla prima) per cifrare una seconda volta il blocco.



Questo sembrerebbe risolvere il problema del DES. Applicandolo due volte sembra che la sicurezza aumenti e la chiave passi da 56 bit a 112 e che quindi ci sono $2^{56*2} = 2^{112}$ possibili chiavi.

Purtroppo, non è così, in quanto è possibile applicare un attacco noto come "meet in the middle".

Osservando il seguente scenario:



Si osserva che, se partiamo dal Plaintext, utilizzando tutte le possibili chiavi K1, otteniamo una lista di Ciphertext (C1₁).

La stessa cosa viene fatta dal lato opposto, cioè, viene preso il testo cifrato e viene brutforzata la decifratura utilizzando tutte le possibili chiavi K2 ottenendo una lista di Ciphertext (C1₂).

Alla fine del processo se si ottiene una corrispondenza tra (C1₁ == C1₂) avremmo la coppia di chiavi che sta usando il poveraccio per comunicare.

Questo attacco è di tipo "known plaintext". Ciò significa che chi effettua l'attacco deve almeno conoscere qualche coppia <P, C> dove P è il plaintext e C è il testo cifrato.

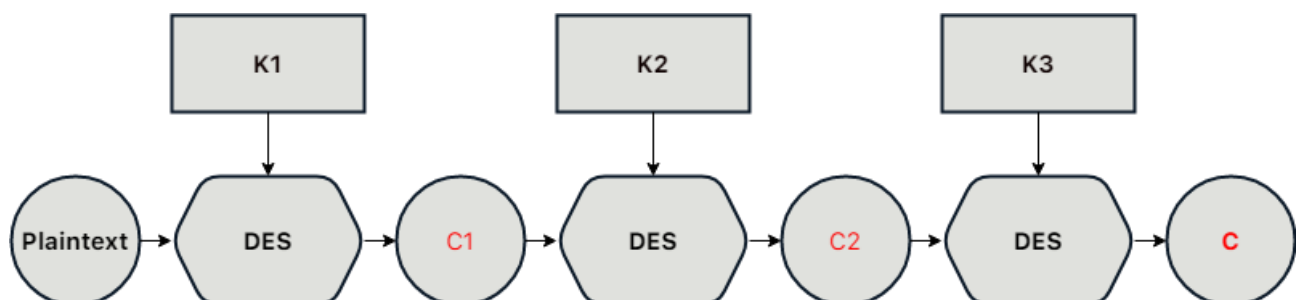
La complessità infine non è 2¹¹² ma, dato che l'attaccante deve solamente eseguire il brutforce su 56bit di chiave 2 volte, la complessità è semplicemente 2 x 2⁵⁶ = 2⁵⁷.

D'altro canto, chi attacca deve avere un bel set di hard-disk per immagazzinare tutte le possibili C1₁ e C1₂ ed eseguire il match (C1₁ == C1₂): In DES un blocco da cifrare è di 64bit (8byte) e abbiamo detto che chi attacca deve prendere il blocco e cifrarlo con tutte le possibili 2⁵⁶ chiavi. Questo produrrà una lista di ciphertext, che per essere memorizzati:

$$8(\text{byte di un blocco}) * 2^{56} = 576.460.752.303.423.488 \text{ Byte} = 576 \text{ Exabyte}$$

Questa è la complessità in termini di spazio solo per la lista del primo tentativo.

La soluzione "definitiva" al problema della lunghezza della chiave del DES e al problema del "meet in the middle" del DDES (Double DES), è utilizzare un Triple DES (3DES).



Cifrari a Blocchi

Una volta visto come viene cifrato un blocco di bit, è necessario però analizzare come viene cifrato un intero messaggio (che quindi è composto da un insieme di blocchi).

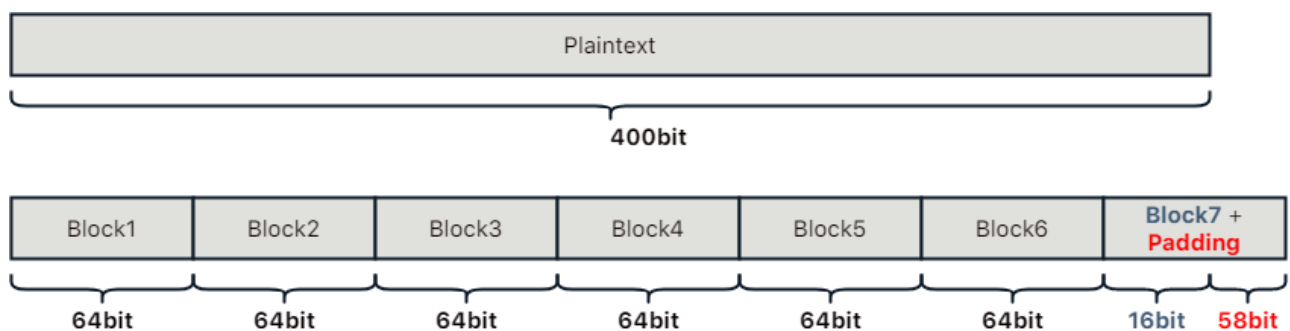
Se il plaintext è di M bit, ed il cifrario che si utilizza suddivide i blocchi in K bit, il numero di blocchi totali sarà:

$$N (\text{blocchi totali}) = M / K + 1$$

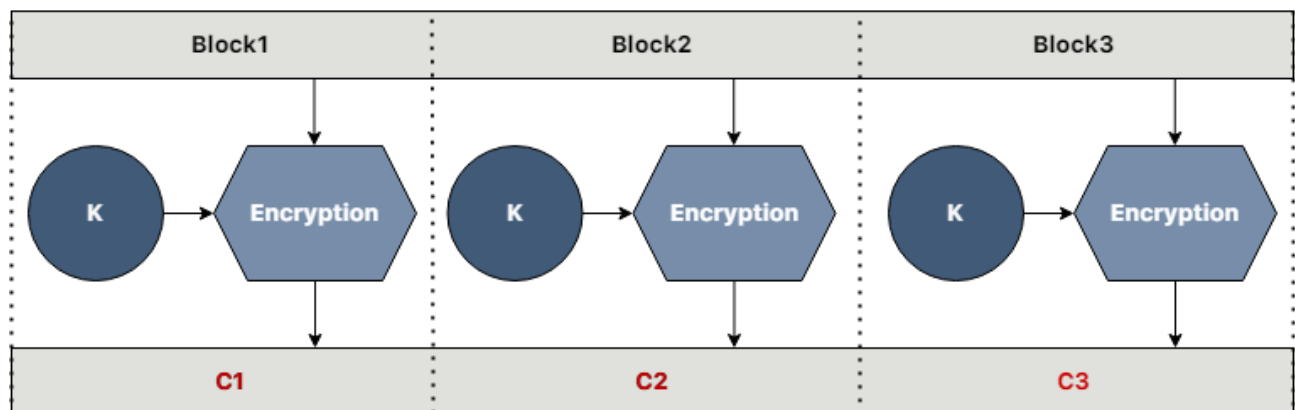
Ad esempio, se usiamo DES, con blocchi da 64 bit, ed abbiamo un plaintext di 400bit, avremmo:

$$N = 400 / 64 = 6,25 = 6 + 1 = 7$$

Quindi se si vuole cifrare un dato di dimensione maggiore di K occorre suddividerlo in blocchi di dimensione K ed eventualmente riempire l'ultimo blocco con tecniche di padding (se la dimensione del dato non è un multiplo esatto di K): L'ultimo blocco si riempie con un insieme di zeri fino a completarlo.



Electronic Codeblock



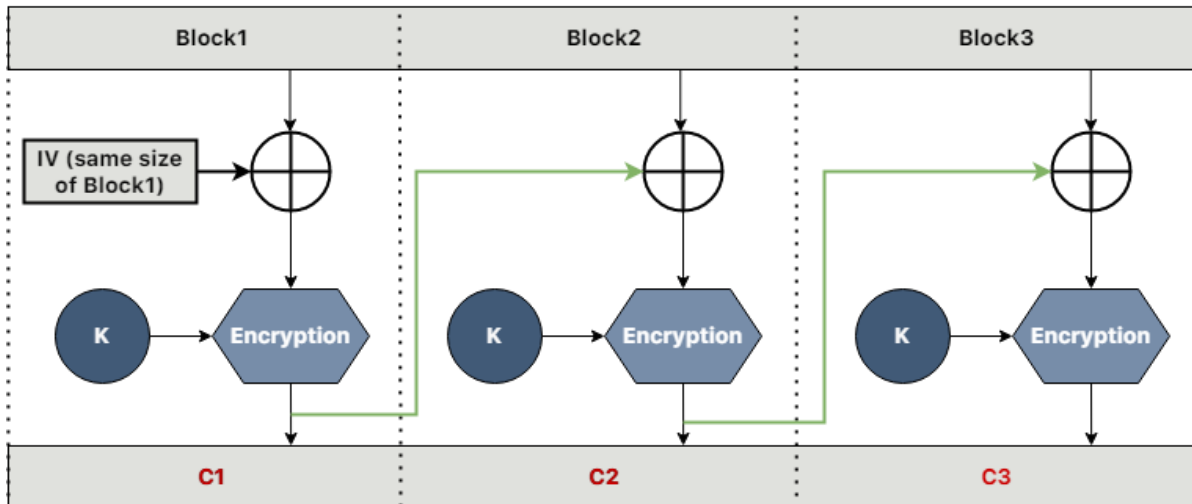
La semplicità di questa tecnica favorisce la velocità di cifratura e decifratura ma è vulnerabile all'attacco di known-plaintext, perché ogni blocco che contiene la stessa sequenza di dati, essendo codificato con la stessa chiave, produce lo stesso blocco cifrato.

Un blocco ripetuto viene cifrato nello stesso modo: è quindi possibile una analisi statistica. In generale, è possibile ottenere informazioni sul testo originario.

Cipher Block Chaining

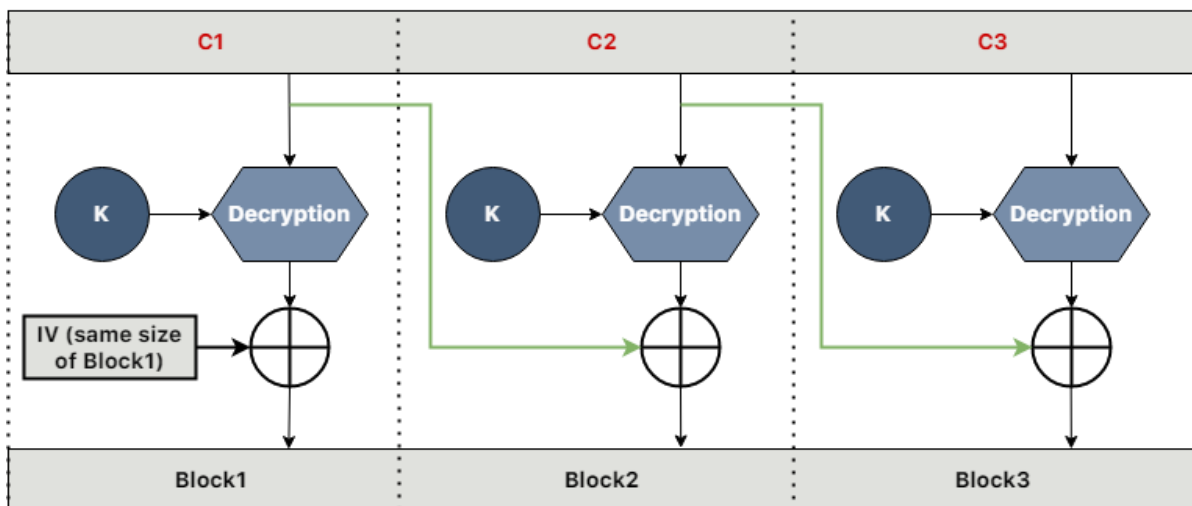
Per superare i limiti di sicurezza di ECB, è necessario l'utilizzo di una tecnica in cui lo stesso blocco di testo in chiaro, se ripetuto, produce blocchi di testo cifrato differenti. Questo è ciò che accade con la modalità CBC, in cui l'input dell'algoritmo di crittografia è il risultato dello XOR tra il blocco di testo in chiaro corrente e il blocco di testo cifrato precedente. Per ciascun blocco viene utilizzata la stessa chiave.

Encryption:



Dato che il primo blocco non ha il blocco ($i - 1$), viene utilizzato un Vettore di inizializzazione (IV – Initialization Vector): un blocco di bit lungo quanto un blocco.

Decryption:



Encryption	Decryption
$C_i = E_k(P_i \oplus C_{i-1})$ $C_0 = IV$	$P_i = D_k(C_i) \oplus C_{i-1}$ $C_0 = IV$
Il blocco cifrato C_i è ottenuto facendo l'encryption del Plaintext P_i messo in XOR con il blocco precedente C_{i-1}	Il Plaintext P_i è ottenuto facendo la decryption del blocco cifrato C_i e poi messo in XOR con il blocco precedente C_{i-1}

Il vettore può essere statico o dinamico, in ogni caso viene comunicato insieme al primo blocco al destinatario del messaggio.

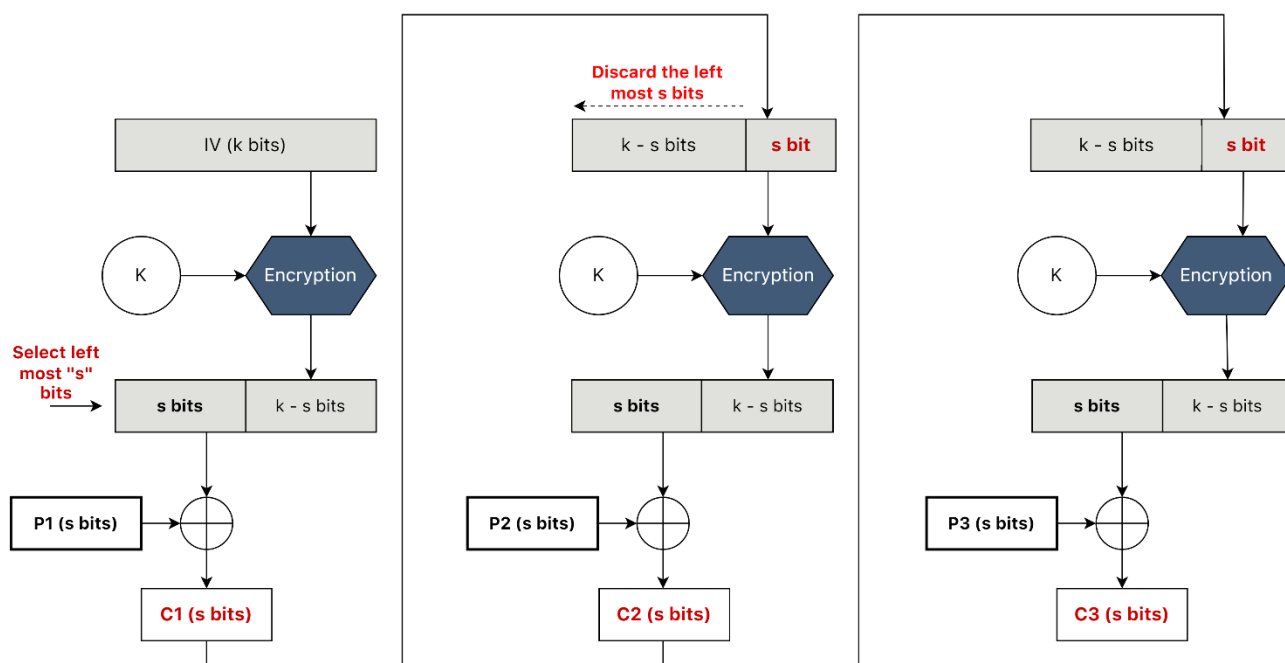
Il vettore di inizializzazione è pubblico ma non è un fattore che mette a rischio la sicurezza dell'algoritmo: il suo scopo è solo quello di non permettere all'attaccante di analizzare il traffico di rete, facendo appunto analisi statistica sui blocchi, poiché ogni blocco (anche se uguale) produrrà un blocco cifrato diverso. Attenzione che questo non accade se l'IV è statico: se si mandano due messaggi identici con IV statico, questo risulterà in blocchi identici. Infatti, per aumentare il livello di sicurezza ed evitare replay attack, si utilizza un IV pseudocasuale per ogni processo di cifratura.

Ampiamente usato con DES (es. 3DES-EDE-CBC).

Cipher Feedback

Per la maggior parte delle applicazioni, utilizzare CBC è la scelta migliore. Ma per una trasmissione di carattere generale orientata al flusso di dati (tipo streaming video) utilizzare CBC è poco efficiente. Ci viene in aiuto il CFB.

La modalità CFB è stata ideata per convertire idealmente una cifratura a blocchi in una cifratura a flusso. La cifratura a flussi non necessita di eseguire riempimenti e può inoltre operare in tempo reale.



Questo cifrario può essere utilizzato come un cifrario a blocchi oppure essere trasformato in un cifrario a flusso poiché la specifica permette di scegliere la dimensione del blocco da cifrare. La specifica richiede appunto un parametro intero, indicato con s , tale che:

$$1 \leq s \leq b \text{ (} b \text{ è la dimensione del blocco completo).}$$

Se ($s == b$) allora diventa un cifrario a blocchi.

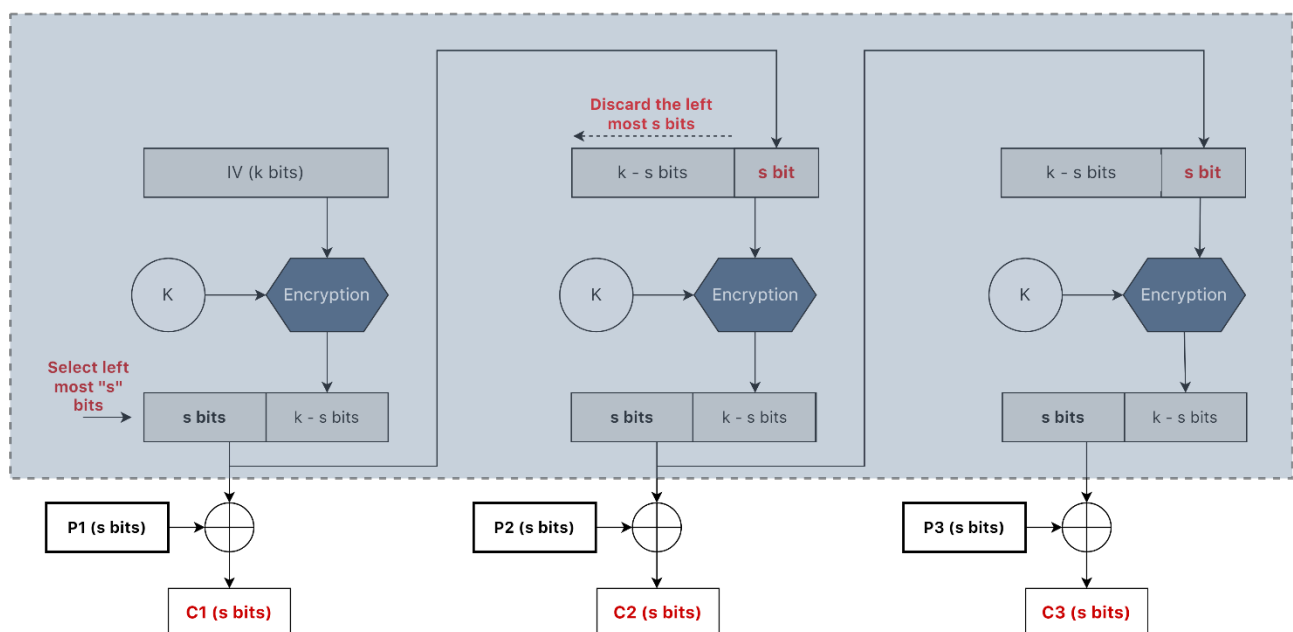
Le modalità più consigliate sono CFB-1, CFB-8, CFB-64, CFB-128.

Encryption	Decryption
$C_i = \text{MSB}_s(E_k(I_i)) \oplus P_i$ $I_i = ((I_{i-1} \ll s) + C_i)$ $I_0 = IV$	$P_i = \text{MSB}_s(E_k(I_{i-1})) \oplus C_i$ $I_i = ((I_{i-1} \ll s) + C_i)$ $I_0 = IV$
<p>La cifratura del blocco C_i è ottenuta prendendo i left most "s" bits dalla (encryption dello shift register (I_i)) e poi messo in xor con il plaintext P_i</p> <p>Lo shift register al passo i, è ottenuto prendendo lo shift register al passo $i - 1$ a cui si shiftano a sinistra "s" bit e si aggiungono a destra C_i bits</p>	

CFB-1 è considerato auto-sincronizzante e resiliente alla perdita di testo cifrato; "Quando viene utilizzata la modalità CFB a 1 bit, la sincronizzazione viene ripristinata automaticamente b + 1 posizioni dopo il bit inserito o cancellato. Per altri valori di s nella modalità CFB e per le altre modalità di riservatezza in questa raccomandazione, la sincronizzazione deve essere ripristinata esternamente". (NIST SP800-38A). Ad esempio, la perdita di 1 bit in un cifrario a blocchi di 128 bit come AES renderà 129 bit non validi prima di emettere bit validi.

Output Feedback

Nel Cipher Feedback ciascuna operazione di cifratura a blocchi dipende da tutte le precedenti e quindi non può essere eseguita in parallelo. Nell'output feedback, poiché il testo in chiaro o il testo cifrato viene utilizzato solo per l'XOR finale, le operazioni di cifratura a blocchi possono essere eseguite in anticipo, consentendo di eseguire il passaggio finale in parallelo una volta che il testo in chiaro o il testo cifrato è disponibile.



Dato che l'IV si riceve al primo blocco, è possibile iniziare a produrre gli "s" bit ad ogni passo che dovranno poi essere messi in XOR con il testo cifrato oppure se siamo nella fase di

cifratura, con il plaintext. Dato che nelle trasmissioni di rete potrebbero esserci problemi di latenza, questo sistema è efficace.

Inoltre, il vantaggio della OFB è che non propaga gli errori di trasmissione dei bit: se c'è un errore di trasmissione, quello produrrà un errore di decifratura solo nel passo in cui quel blocco "manomesso" verrà utilizzato. Il resto dei blocchi è indipendente.

Protocollo Diffie Hellman

Pur esistendo un cifrario a chiave condivisa (o cifrario simmetrico) sufficientemente sicuro per permettere a due nodi della rete di comunicare (come DED o AES), rimane il problema che i due nodi dovranno scambiarsi questa chiave condivisa in una rete che, nella maggior parte dei casi, non è sicura.

Problema dello scambio di chiavi condivise

Su una rete non sicura esistono tecniche per analizzare il traffico di rete ed ottenere quindi le informazioni che i nodi interni alla rete si scambiano, questa tecnica viene chiamata Packet Sniffing o Packet Analyzer. Quindi è anche possibile sniffare la chiave condivisa.

Per uno scambio "sicuro" di una chiave condivisa, su una rete pubblica, si utilizza quindi il protocollo crittografico di Diffie Hellman.

Richiami di aritmetica modulare

Per capire come funziona il protocollo è necessario conoscere alcune proprietà dell'aritmetica modulare:

- **Operazione Modulo**
 - la funzione modulo, indicato con "mod", è quella funzione che dà come risultato il resto della divisione euclidea del primo numero per il secondo

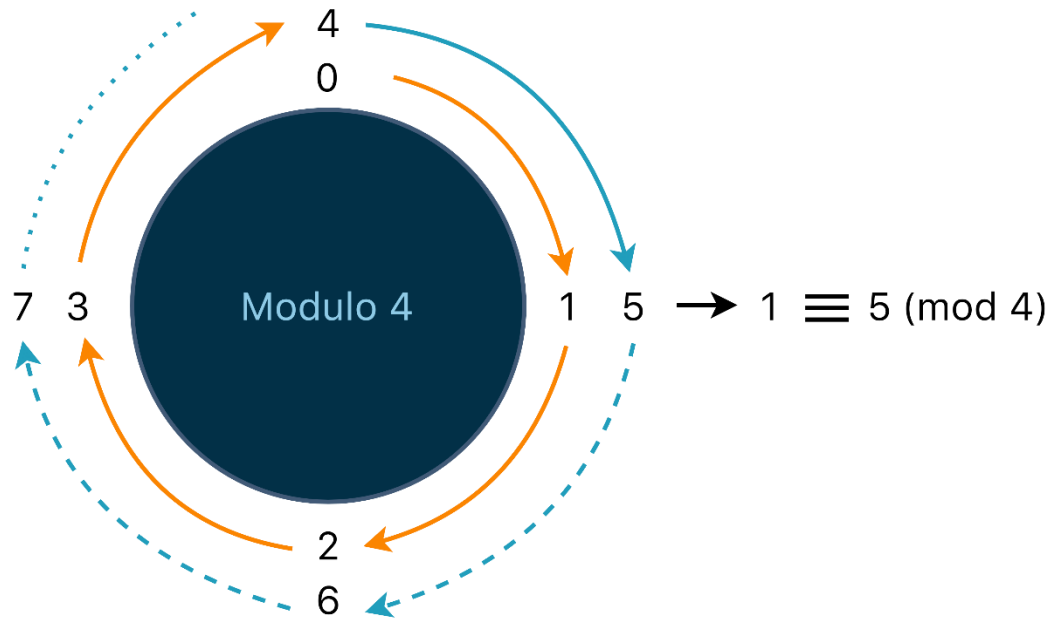
$$\lfloor 13 / 3 \rfloor = 4$$

$$13 - (3 * 4) = 1$$

- **Congruenza in modulo**
 - La relazione di congruenza dice che, presi tre numeri (a, b, n) a e b sono congruenti modulo n, oppure che a è congruo a b modulo n, se la differenza (a - b) è un multiplo di n.
 - Nel caso entrambi i numeri siano positivi, si può anche dire che a e b sono congruenti modulo n se hanno lo stesso resto nella divisione per n.

$$a \equiv b \pmod{n}$$

$$\begin{array}{lcl} 1 \equiv 5 \pmod{4} & \rightarrow & 5 - 1 = 4 \\ 9 \equiv 1 \pmod{4} & \rightarrow & 9 - 1 = 8 \end{array} \quad \left. \vphantom{\begin{array}{l} 5 - 1 = 4 \\ 9 - 1 = 8 \end{array}} \right\} \begin{array}{l} 4 \text{ e } 8 \text{ sono multipli di } 4 \end{array}$$



- Proprietà (Identità)

- Identità

- $(a \bmod n) \bmod n = a \bmod n$

(palese, è come scrivere $((4/1)/1)$)

- $n^x \bmod n = 0$, per qualsiasi x positivo.

(guardare il cerchio in modulo 4, ad esempio $4^2 = 16 \rightarrow 16 \bmod 4 = 0$)

- Distributiva

- $a * b \bmod n = [(a \bmod n) * (b \bmod n)] \bmod n$

(Si può distribuire il modulo sul prodotto dei singoli fattori).

- $(a^b) \bmod M = (a \bmod M)^b \bmod M$

Come funziona Diffie Hellman

I passi per produrre una chiave condivisa secondo il protocollo Diffie Hellman sono i seguenti:

Abbiamo due nodi A e B sulla rete che vogliono condividere la chiave:

1. A e B concordano pubblicamente su due valori comuni ad entrambi i nodi:
 - a. Un valore 'q' che è un numero primo
 - b. Una radice primitiva 'g' di 'q'
2. Il nodo A sceglie un intero positivo (segreto) 'S_a' (con S_a < q), ed invia al nodo B una chiave pubblica P_a = g^{S_a} mod q
3. Il nodo B sceglie un intero positivo (segreto) 'S_b' (con S_b < q), ed invia al nodo A una chiave pubblica P_b = g^{S_b} mod q
4. Il nodo A calcola la chiave condivisa: K = P_b^{S_a} mod q
5. Il nodo B calcola la chiave condivisa: K = P_a^{S_b} mod q
6. Ora i due nodi condividono un segreto (K)

Le due chiavi K sono uguali, e si dimostra con una semplice **proprietà delle potenze**:

$$(X^y)^k = (X^k)^y$$

La chiave condivisa calcolata dal Nodo A risulta essere:

$$K = P_b^{S_a} \text{ mod } q$$

Dove P_b è

$$P_b = g^{S_b} \text{ mod } q$$

La chiave calcolata dal Nodo A si può riscrivere nel modo seguente:

$$K = (g^{S_b})^{S_a} \text{ mod } q$$

Analogamente, la chiave calcolata dal Nodo B:

$$K = (g^{S_a})^{S_b} \text{ mod } q$$

Per le **proprietà delle potenze**:

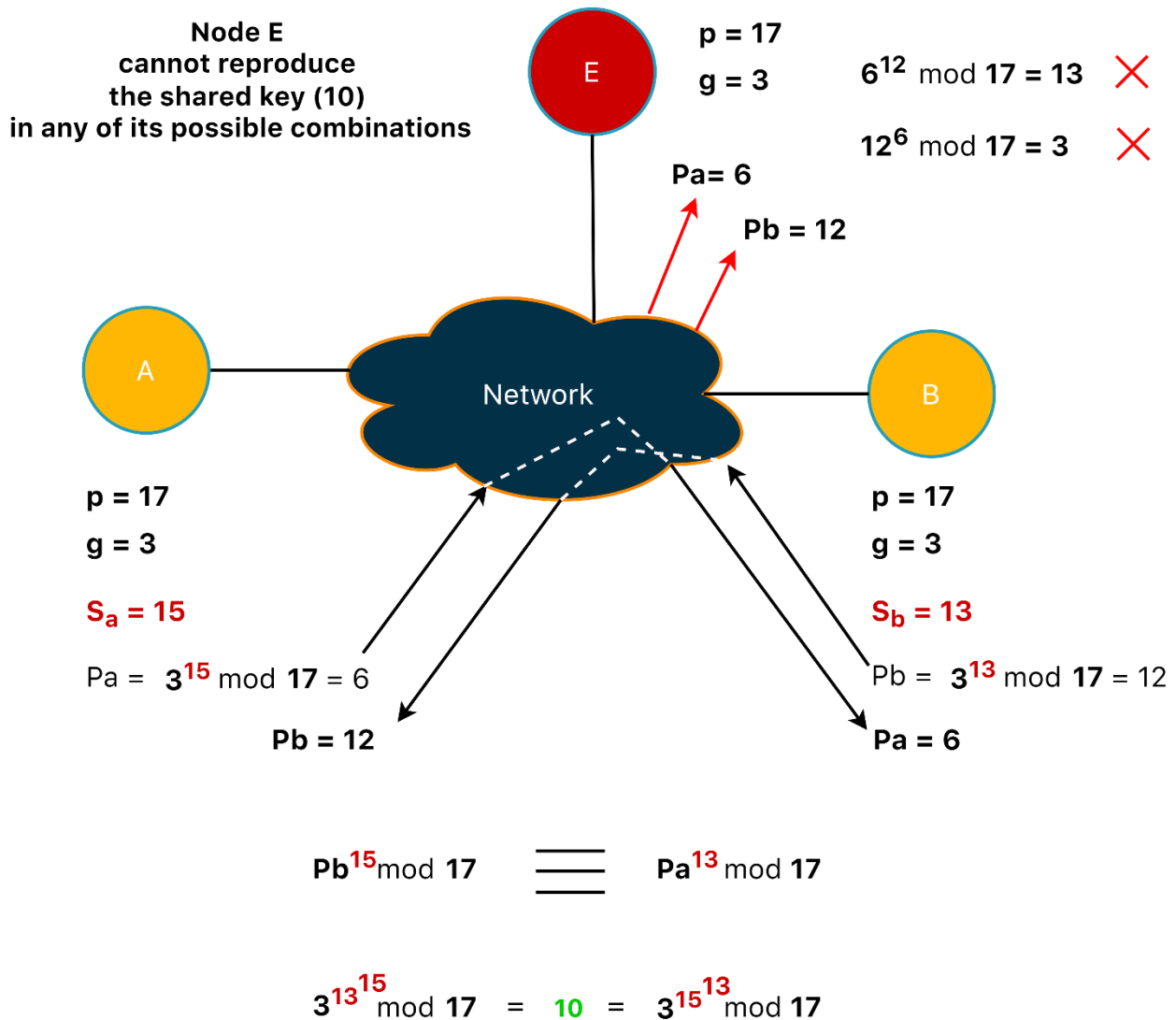
$$K = (g^{S_b})^{S_a} \text{ mod } q = (g^{S_a})^{S_b} \text{ mod } q$$

L'unico modo per un altro nodo sulla rete di riuscire a riprodurre la chiave condivisa tra il nodo A ed il nodo B è di risalire ad una delle due chiavi private a partire dalle chiavi pubbliche inviate sulla rete:

$$P_b = g^{S_b} \text{ mod } q$$

P_b (chiave pubblica del nodo B) ad esempio è un'informazione pubblica. L'unico modo per un nodo malintenzionato di recuperare **la chiave privata** da quella pubblica è ricavare **S_b** utilizzando il logaritmo discreto: Chi genera la chiave pubblica applica **l'esponente modulare** all'operazione **g^{S_b} mod q**, questa operazione è di complessità polinomiale (P), **il logaritmo discreto invece è un problema di tipo NP**.

Schematizzato:



Algoritmi efficienti per DH

Una volta analizzato il processo, c'è bisogno di algoritmi efficienti per generare i parametri di cui il protocollo DH necessita:

Come calcolare in modo efficiente l'esponente modulare ($a^b \bmod q$)

Iniziamo ad analizzare un metodo poco efficiente

```
1. int expMod(int a, int b, int q)
2. {
3.     int s = a;
4.     for (int i = 1; i < b - 1; i++)
5.     {
6.         s = (a * s) % q;
7.     }
8.     return s % q;
9. }
```

La complessità in questo caso è esponenziale rispetto al numero di bit di 'b'. Essendo che si ripete il ciclo for b volte:

$$O(b) = O(2^n) \mid \text{dove } n \text{ è il numero di bit di 'b'}$$

Il secondo metodo che si propone è anch'esso ingenuo ma utilizza la ricorsione:

```
1. int expmod(int a, int b, int q)
2. {
3.     // restituisce un valore minore di q
4.     if (b == 0)
5.         return 1; //  $a^0 \bmod q = 0$ 
6.     if (b % 2 == 0)
7.         return sq(expmod(a, b / 2, q)) % q; // b pari
8.     else
9.         return (a * expmod(a, b - 1, q)) % q; // b dispari
10. }
```

Questo metodo ha complessità logaritmica rispetto al numero di bit di 'b', pertanto:

$$O(2 \log_2(b)) = O(2 \log_2(2^n)) = O(2n) = O(n)$$

Dato che b è dell'ordine di 2^n abbiamo una complessità lineare.

Ovviamente applicando questo metodo si va in contro a problemi come quello di avere uno stack solo di attivazioni di metodi expmod(), e quindi andare a saturare lo stack.

La soluzione definitiva sarebbe quella di utilizzare il metodo iterativo:

```
1. int expMod(int a, int b, int q)
2. {
3.     //  $b = b_k \dots b_2 b_1 b_0$ 
4.     // c non serve al calcolo, solo per la dimostrazione di correttezza
5.     int c = 0;
6.     int d = 1;
7.     for (int i = k; i >= 0; i--)
8.     {
9.         c = c * 2;
10.        d = (d * d) % q;
11.        if (b_i == 1)
12.        {
13.            c = c + 1;
14.            d = (d * a) % q;
15.        }
16.    }
17.    return d;
18. }
```

Mi rifiuto di dimostrare la correttezza sta roba.

Ad ogni modo, per il metodo iterativo, la complessità è lineare rispetto al numero di bit di b: $O(k)$ | dove k è il numero di bit di b.

Come generare in modo efficiente un numero "q" primo

Prima di tutto c'è da porsi la domanda: Esistono infiniti numeri primi? A questo enigma viene in aiuto il teorema di Euclide. Dimostriamo per assurdo che ci sono infiniti numeri primi.

Teorema di Euclide per infiniti numeri primi

Dimostrazione per assurdo

Ipotesi

Esiste un numero P maggiore di tutti gli altri numeri primi

Prendiamo un numero N come prodotto di tutti i numeri primi, successivamente addizioniamo 1 ottenendo N + 1

$$\begin{array}{c} (2 \times 3 \times 5 \times \dots \times P) + 1 \\ \underbrace{\hspace{1.5cm}} \\ N \\ \underbrace{\hspace{1.5cm}} \\ N + 1 \end{array}$$

Per il teorema fondamentale dell'aritmetica

Per ogni numero naturale maggiore di 1 ha un divisore primo

N+1 non è divisibile per 2 perché lo sarebbe N, e quindi ha resto 1. Non è divisibile per 3, per lo stesso motivo, e così via.

A questo punto, N + 1 o è un numero primo, oppure non essendo primo, è il prodotto di numeri primi che non possono figurare tra gli N ipotizzati

In entrambi i casi si perviene alla conclusione che (per assurdo) non può non esistere un numero primo più grande di N, dunque i numeri primi sono infiniti.

Esempio

$$\begin{array}{c} (2 \times 3 \times 5 \times 11 \times 13) + 1 \\ \underbrace{\hspace{1.5cm}} \\ 30030 \\ \underbrace{\hspace{1.5cm}} \\ 30031 \end{array}$$

30031 non è un numero primo ma è composto da 59×509

La seconda domanda è: con che frequenza troviamo un numero primo?

Oppure: quanti numeri primi ci sono in un intervallo che va da 0 ad un dato N?

Il teorema dei numeri primi ci viene in aiuto:

Teorema dei numeri primi

Enunciato

Per ogni numero reale positivo x, si definisca la funzione:

$$\pi(x)$$

Che restituisce il numero di primi minori o uguali a x

Il teorema afferma che

$$\pi(x) \sim \frac{x}{\ln(x)}$$

La funzione $\pi(x)$ si può approssimare (è equivalente a) all'infinito con $x/\ln(x)$

$$\lim_{x \rightarrow +\infty} \frac{\pi(x)}{\frac{x}{\ln(x)}} = 1$$

Esempio

$$\pi(10)$$

$$\frac{10}{\ln(10)} \approx 4$$

Il primo approccio potrebbe essere quello un po' più diretto:

```
1. M = generateRandomNumber();
2. for (i = 2; i * i <= M; i++)
3.     if (M % i == 0) goto 1 // M non primo
4. return M; // M primo
```

In questo caso generiamo un numero random M e controlliamo che ogni numero (partendo da 2) fino alla radice di M non sia un divisore di M. **Questo approccio però è troppo lento:** infatti la complessità è **esponenziale** rispetto del numero di bit di M è:

$$M^{(1/2)} = 2^{(k/2)}$$

Esempio:

$$k = 512 \rightarrow 2^{(512/2)} = 2^{(256)} = 1,2e+77 \text{ (numero di divisioni da effettuare)}$$

Un metodo più efficace è quello probabilistico. Utilizzando questo metodo, si effettuano dei test chiamati di primalità sul numero random generato, per verificare che quest'ultimo possa essere primo o meno.

Negli anni sono stati presentati diversi test di primalità, **quello che analizzeremo sarà il Test di Miller Rabin.**

L'algoritmo che utilizza il test di primalità è il seguente:

```
1. M = generateRandomNumber();
2. for (i = 2; i < T; i++) {
3.     a = generateRandomLessThanM();
4.     if (MillerRabinTest(a, M)) goto 1
5. }
6. return M; // M primo
7.
```

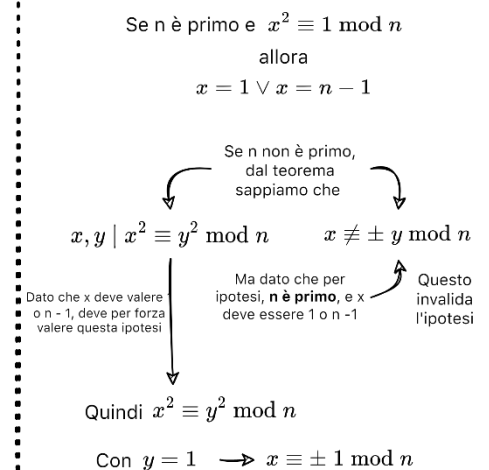
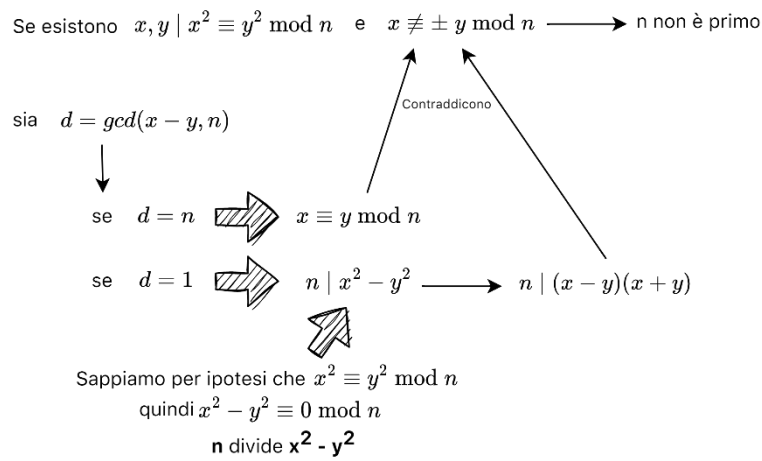
Dove MillerRabinTest è una funzione così definita:

```
1. MillerRabinTest(a, M) {
2.     // M-1 = bk ... b2 b1 b0: bit di M
3.     d = 1;
4.     for (i = k; i >= 0; i--) {
5.         x = d;
6.         d = (x * x) % M; // d ≡ x² (mod M)
7.         if (d == 1 && x != 1 && x != M - 1)
8.             return (TRUE); // 1 ≡ x² (mod M)
9.         if (bi == 1)
10.            d = (d * a) % M;
11.     } // d = aM-1 mod M = 1
12.     if (d != 1)
13.         return TRUE;
14.     else
15.         return (FALSE);
16. }
```

All'interno del test di Miller Rabin si fa uso di due teoremi:

1. Piccolo Teorema di Fermat: se M è primo e $a < M$ allora $a^{M-1} \bmod M = 1$

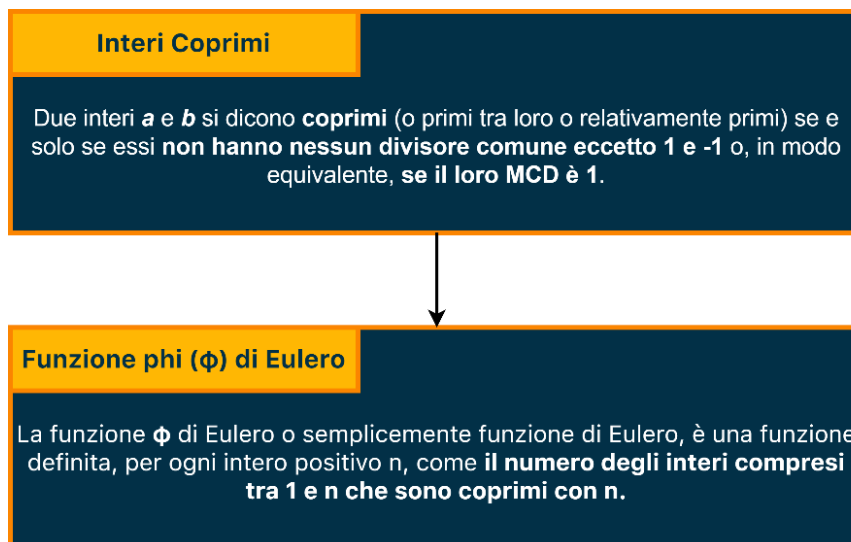
2. Principio fondamentale:



Come calcolare in modo efficiente una radice primitiva "g"

Prima di spiegare come si genera un generatore di q , è necessario spiegare il Teorema di Eulero. È un teorema che sia per utile in Diffie Hellman che per l'algoritmo RSA che vedremo.

Per poterlo discutere c'è bisogno di introdurre la nozione di **Interi coprimi e la funzione phi di Eulero**:



Esempio

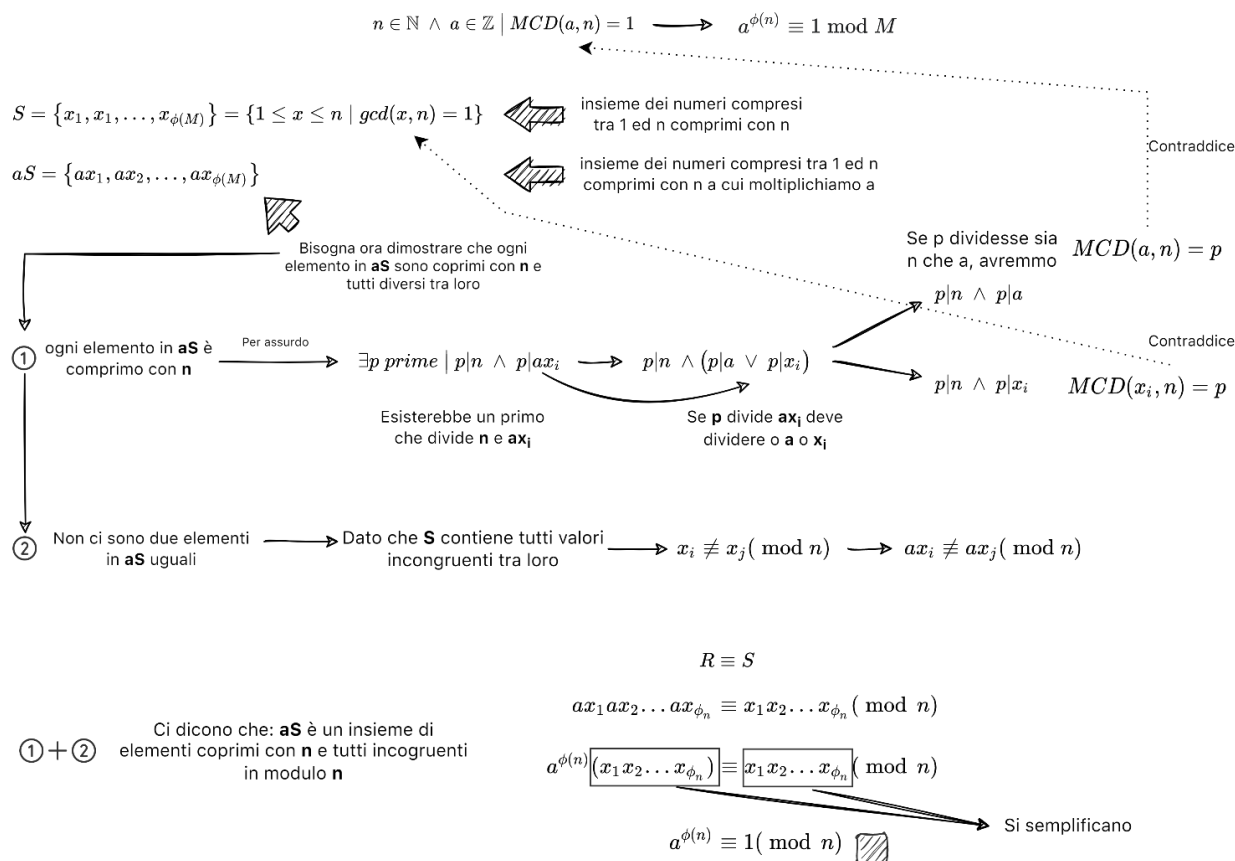
$$\text{MCD}(4, 3) = 1$$

$$\text{MCD}(3, 14) = 1$$

$$\phi 8 = 4$$

Sapendo ora cosa sono due interi coprimi e la funzione di Eulero, introduciamo il **teorema di Eulero**:

Teorema di Eulero

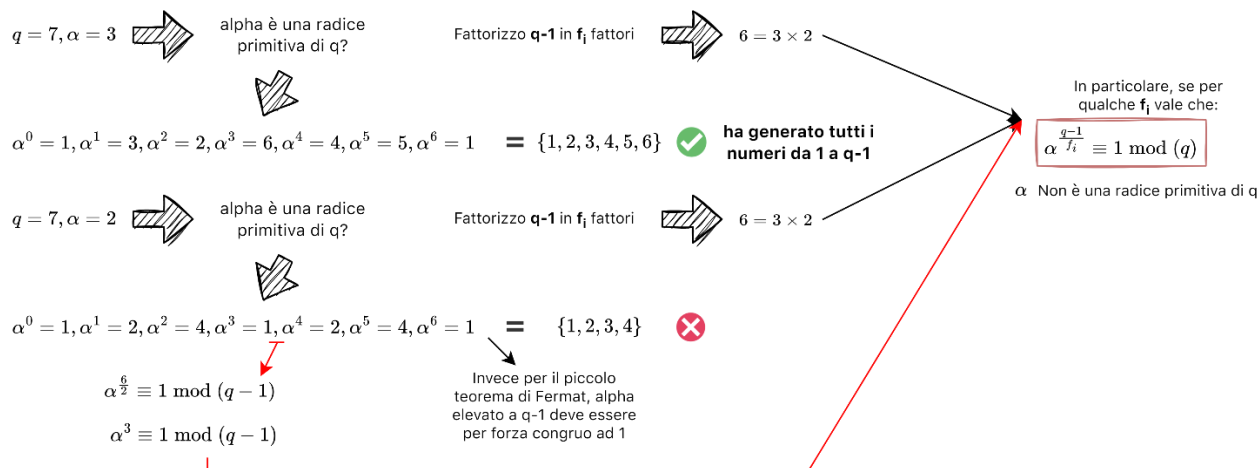


Come ottenere ora una radice primitiva?

Definizione di radice primitiva:

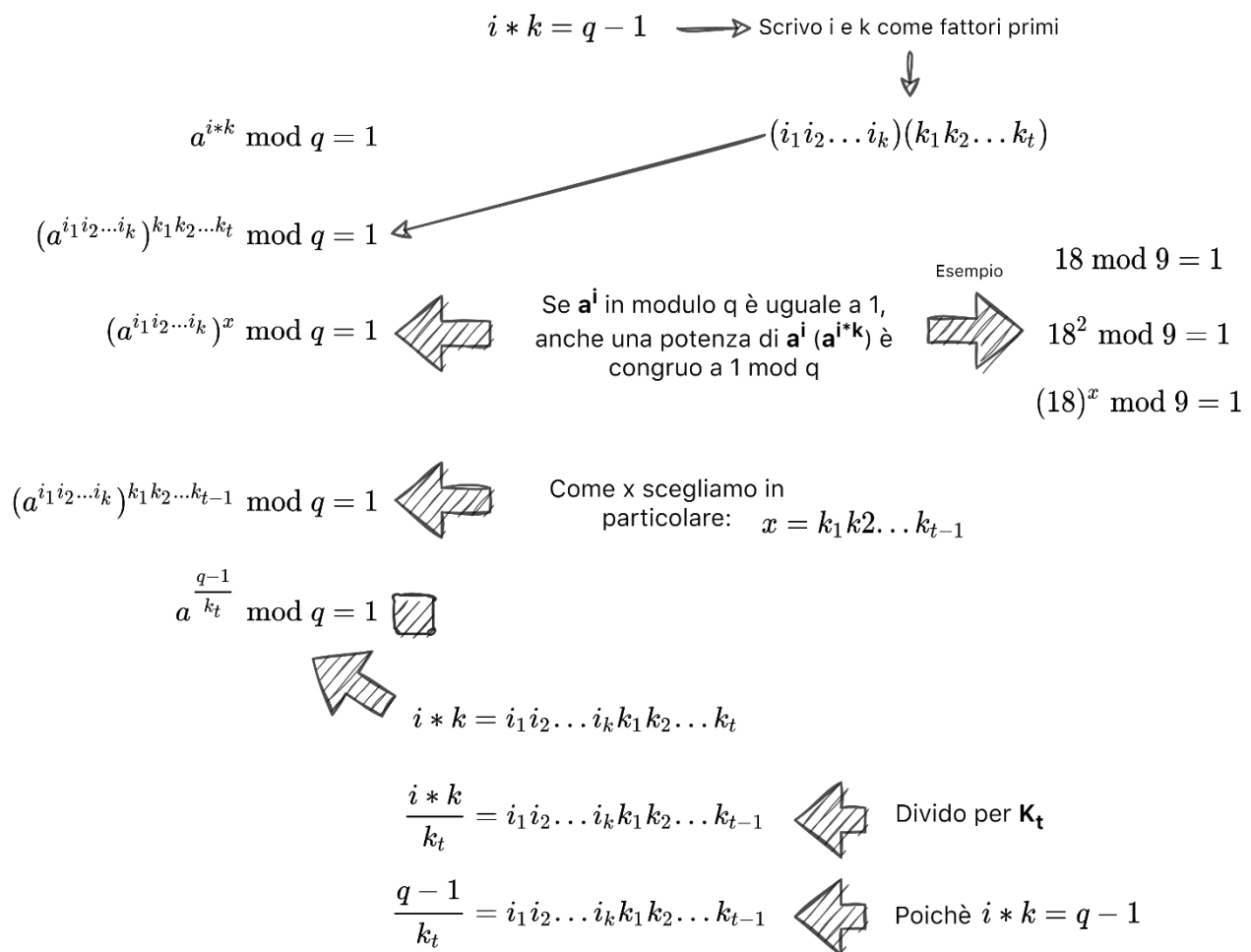
$$S = \{1, 2, \dots, q-1\} \mid \forall b \in S, \exists i \mid b = \alpha^i \pmod{q}$$

Alpha è una radice primitiva di q se è possibile ottenere tutti i possibili valori di S (quindi tutti i valori da 1 a q-1) elevando alpha ad un qualsiasi i.



Dimostriamo l'algoritmo per verificare che alpha sia una radice primitiva di q:

Se α non è una radice primitiva, allora: $\exists i < q-1 \mid \alpha^i \bmod q = 1$ inoltre $\exists k \mid i * k = q-1$



Algoritmo efficiente per generare alpha generatore di q:

```
1. // Genero alpha minore di q, tale che alpha sia relativamente primo con q
2. // Fattorizzo q-1 in {f1, f2, ..., fj} fattori.
3. // Se esiste fi tale che alpha^{q-1/fi} mod q = 1
4. // Torno al passo 1
5. // Altrimenti restituisci Alpha}
```

Al passo uno, possiamo generare un alpha che sia primo con Miller Rabin così è sicuramente relativamente primo con q.

Il passo due potrebbe far pensare che l'algoritmo sia NP (Non esistono algoritmi che fattorizzano in tempi polinomiali). Questo è possibile aggirarlo: invece di generare q a caso e poi fattorizzare, genero j fattori a caso, li moltiplico tra loro, aggiungo 1 e faccio il test di primalità.

Il passo tre è un esponente modulare, quindi $O(n)$.

RSA

Algoritmo per la generazione di chiavi:

- Scelgo due numeri 'p' e 'q' primi.
 - Complessità: Vista con Diffie-Hellman per la generazione di numeri primi
- Calcolo il modulo 'n' come: $n = p \cdot q$
- Scelgo $e < (p-1)(q-1)$ tale che $\text{MCD}(e, (p-1)(q-1)) = 1$
- Calcolo la chiave privata come $d = e^{-1} \bmod (p-1)(q-1)$

A questo punto avremo le chiavi

- Pubblica: $K^+ = \langle e, n \rangle$
- Privata: $K^- = \langle d, n \rangle$

Per cifrare un messaggio useremo la chiave pubblica:

$$c = m^e \bmod n$$

Per decifrare useremo la chiave privata:


$$m = c^d \bmod n$$

Funzione di Eulero (moltiplicativa)


La funzione di Eulero è moltiplicativa e, in particolare:

$$\text{if } p, q \text{ prime} \rightarrow \phi(n) = \phi(pq) = \phi(p)\phi(q) = (p-1)(q-1)$$

Se p e q sono primi, la funzione phi di Eulero restituisce (p-1)(q-1)

$$\phi(n) = \phi(pq) = \#\{1 < x < n-1 \mid \gcd(x, p) = 1 \vee \gcd(x, q) = 1\}$$


numeri da 1 a n-1 che non sono multipli di p o di q

$$\begin{aligned}\phi(n) &= n - 1 - |\{p, 2p, \dots, (q-1)p, q, 2q, \dots, (p-1)q\}| = \\ &= n - 1 - [(q-1) + (p-1)] = \\ &= n - q - p + 1 = \\ &= pq - p - q + 1 = (p-1)(q-1)\end{aligned}$$


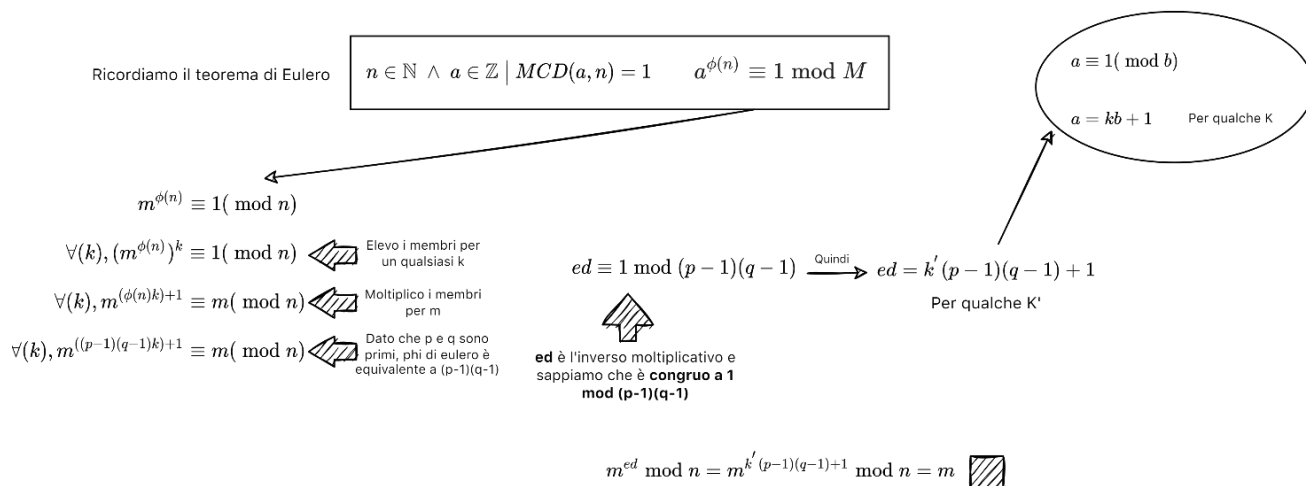
Correttezza RSA

Abbiamo detto che per cifrare utilizziamo la chiave pubblica e per decifrare utilizziamo quella privata. Ora bisogna dimostrare quindi, che il passaggio da testo in chiaro a testo cifrato e viceversa funzionano.

In particolare, dobbiamo dimostrare che:

$$m = (m^e \bmod n)^d \bmod n$$

$$m = m^{ed} \bmod n$$



Sicurezza per RSA

RSA è ritenuto sicuro in quanto (la precedente implica la successiva):

- Si ritiene computazionalmente difficile calcolare p e q a partire da n, per n grande (RSA utilizza almeno 1024bit di modulo per ritenersi sicuro): è difficile fattorizzare n in p*q.
- Si ritiene computazionalmente difficile calcolare d (la chiave privata) a partire da e (chiave pubblica) ed n senza conoscere p e q.
- È computazionalmente difficile decifrare un messaggio con RSA senza conoscere la corrispondente chiave privata d

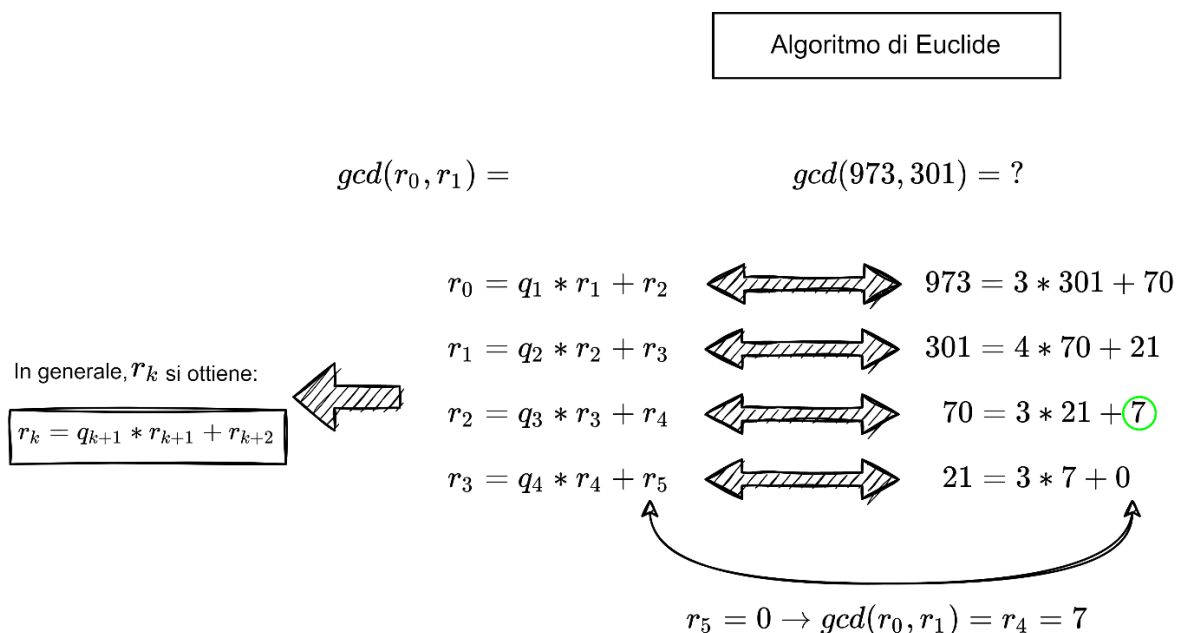
Efficienza per RSA

Chi cifra utilizza per lo più esponenti modulari per la generazione delle chiavi e la cifratura del plaintext; quindi, la complessità è logaritmica rispetto al numero di bit di n.

Per contro sono inefficienti rispetto a cifrari simmetrici.

Algoritmo efficiente per calcolare l'inverso moltiplicativo

Ricordiamo l'algoritmo di Euclide per trovare il massimo comun divisore di due numeri:



Ora che abbiamo ricordato come funziona l'algoritmo Euclide, introduciamo l'algoritmo di Euclide Esteso, che è l'algoritmo su cui si basa il calcolo dell'inverso moltiplicativo.

L'algoritmo di Euclide esteso, oltre a ritornare il massimo comun divisore tra i due input, afferma che: il massimo comun divisore tra due numeri è rappresentabile secondo una combinazione lineare dei due input:

$$\exists x, y \mid \gcd(a, b) = ax + by$$

Ci avvaliamo comunque di Euclide standard	Euclide Esteso
$973 = 3 * 301 + 70$	$70 = [1] * 973 + [-3] * 301$
$301 = 4 * 70 + 21$	$21 = 301 - 4 * 70 = 301 - 4 * (973 - 3 * 301)$ $= [-4] * 973 + [13] * 301$
$70 = 3 * 21 + 7$	$7 = 70 - 3 * 21 = (973 - 3 * 301) - 3(-4 * 973 + 13 * 301)$
$21 = 3 * 7 + 0$	$= [13] * 973 + [-42] * 301$
Ad ogni passo si esplicita il resto	$\gcd(a, b) = ax + by$

In generale, il resto r_i è rappresentabile come:

$$\begin{aligned}
 r_{i-2} &= s_{i-2}r_0 + t_{i-2}r_1 \\
 r_{i-1} &= s_{i-1}r_0 + t_{i-1}r_1 \\
 r_{i-2} &= q_{i-1}r_{i-1} + r_i \\
 r_i &= r_{i-2} - q_{i-1}r_{i-1} \\
 &= (s_{i-2}r_0 + t_{i-2}r_1) - q_{i-1}(s_{i-1}r_0 + t_{i-1}r_1) \\
 &= [s_{i-2} - q_{i-1}s_{i-1}]r_0 + [t_{i-2} - q_{i-1}t_{i-1}]r_1 \\
 &= [s_i]r_0 + [t_i]r_1
 \end{aligned}$$

Ora applichiamo l'algoritmo di Euclide Esteso al problema di trovare l'inverso modulare:

Ricordo i passaggi per RSA:

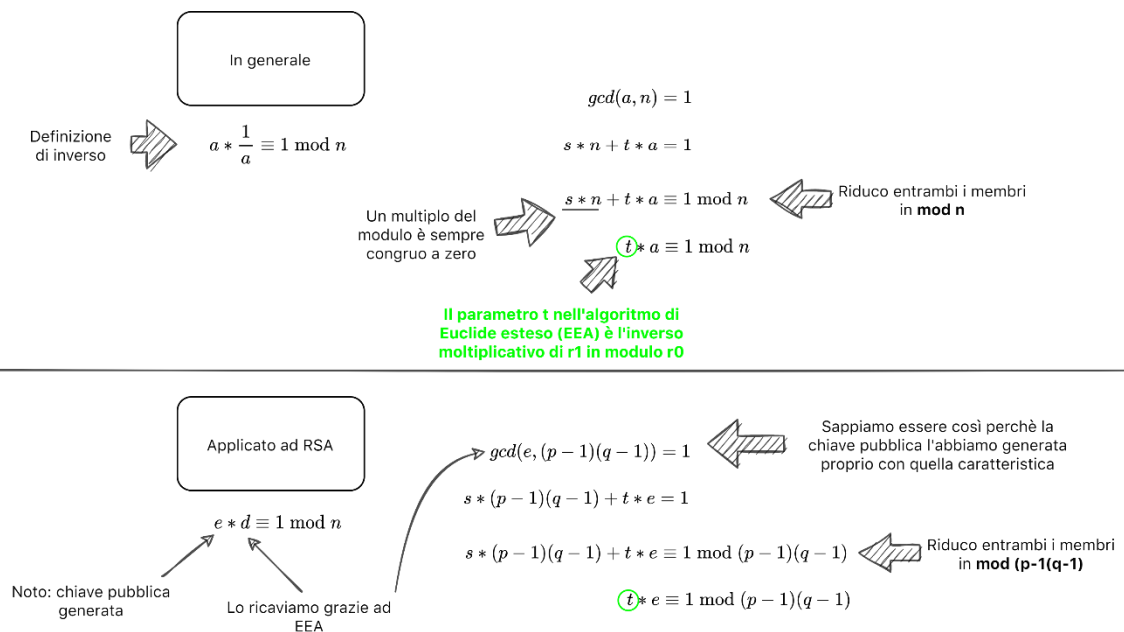
Algoritmo per la generazione di chiavi:

- Scelgo due numeri 'p' e 'q' primi.
- Calcolo il modulo 'n' come: $n = p * q$
- Scelgo $e < (p-1)(q-1)$ tale che $\text{MCD}(e, (p-1)(q-1)) = 1$
- Calcolo la chiave privata come $d = e^{-1} \text{ mod } (p-1)(q-1)$

A questo punto avremo le chiavi

- Pubblica: $K^+ = \langle e, n \rangle$
- Privata: $K^- = \langle d, n \rangle$

Nell'ultimo passaggio, bisogna calcolare la chiave privata, che è l'inverso moltiplicativo della chiave pubblica in modulo $(p-1)(q-1)$:



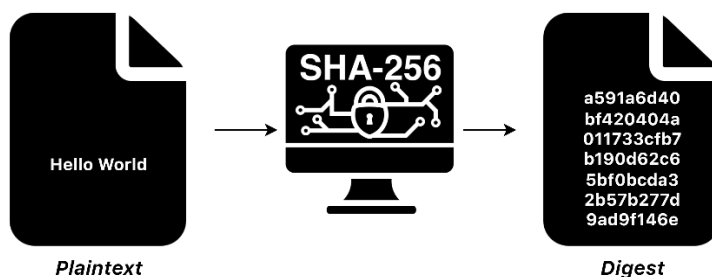
Hash

Fino ad ora abbiamo trattato solo uno degli obiettivi crittografici, la riservatezza: un messaggio cifrato è un messaggio che è illeggibile (riservato). La riservatezza è appunto uno degli obiettivi di sicurezza che rende il messaggio riservato ai soli destinatari.

Il prossimo obiettivo di sicurezza che tratteremo sarà l'Integrità. L'integrità si occupa di verificare che il messaggio, durante il suo cammino sulla rete, non venga modificato. Uno dei meccanismi per implementare questo livello di sicurezza è utilizzando una delle principali primitive crittografiche più diffusa: le funzioni di Hash.

Una funzione hash è qualsiasi funzione che può essere utilizzata per mappare dati di dimensioni arbitrarie su valori di dimensioni fisse. I valori restituiti da una funzione hash sono chiamati valori hash, codici hash, digest o semplicemente hash.

Vengono utilizzate in diversi campi dell'informatica, come le basi di dati, sistemi operativi (password utente), e anche nella sicurezza informatica.



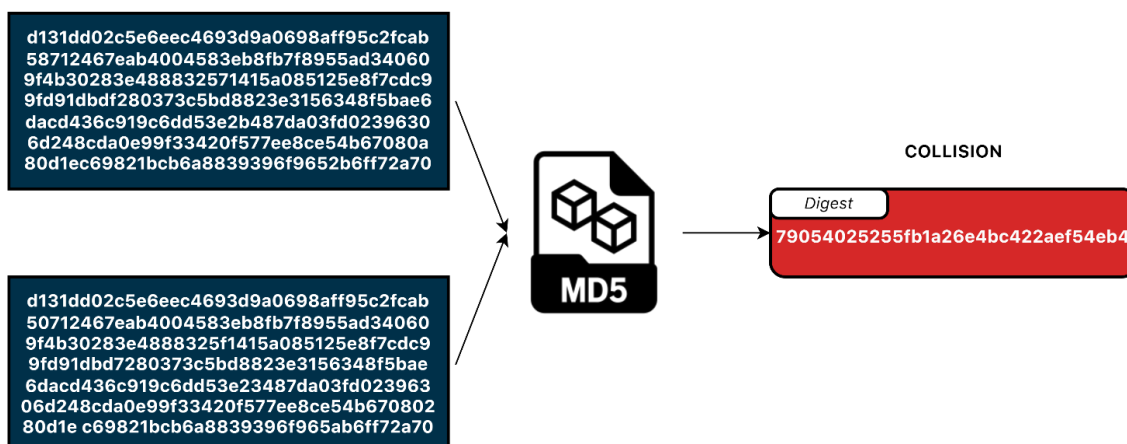
Produce quindi un codice (digest) che, nella maggior parte dei casi, è rappresentato nel formato esadecimale. In sostanza si genera una sorta di impronta digitale per il plaintext (stesso input, stesso digest).

Questa procedura è unidirezionale (one way function), poiché, non è possibile partendo dal digest, ottenere il messaggio originario.

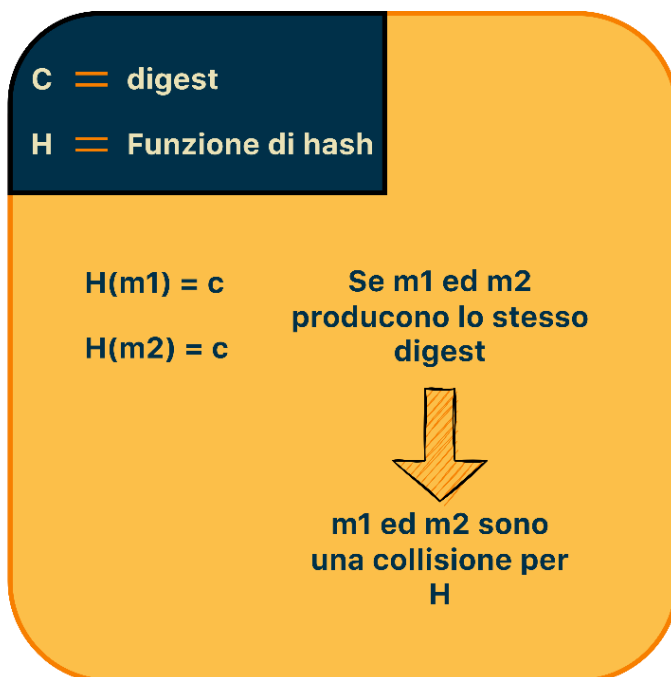
Inoltre, l'output delle funzioni di hash è sempre della stessa lunghezza indipendentemente dalla lunghezza dell'input.

Hash Collision

Una collisione hash si verifica quando due input diversi condividono lo stesso valore hash. Ad esempio, con l'algoritmo MD5 è stata trovata una collisione nel 1996 con questi due valori di input:



Più formalmente:



Una funzione di hash può essere più o meno resistente alle collisioni, qui elencati i livelli di sicurezza alle collisioni:

1. Non invertibile: dato c (il digest), è difficile trovare m (è difficile a partire da c , trovare i messaggi che generano quel digest).
È quindi difficile trovare $H^{-1}(c)$ = insieme dei messaggi m tale che $H(m)=c$
2. Fortemente non invertibile: dato $m1$, è difficile trovare $m2$ tale che $H(m1) = H(m2)$
3. Resistente alle collisioni: è difficile trovare $m1$ e $m2$ tali che $H(m1) = H(m2)$

Attacco del compleanno

Un attacco di compleanno è un **tipo di attacco crittografico che sfrutta la matematica alla base del problema del compleanno nella teoria della probabilità, per trovare una collisione di una funzione hash.**

L'attacco si basa quindi sul paradosso del compleanno: è un paradosso di teoria della probabilità che afferma che la probabilità che almeno due persone in un gruppo compiano gli anni lo stesso giorno è largamente superiore a quanto potrebbe dire l'intuito: infatti già in un gruppo di 23 persone la probabilità è circa 0,51 (51%).

Si può calcolare usando la probabilità di eventi classica (casi favorevoli su casi possibili):

$$\frac{\text{Casi favorevoli}}{\text{Casi possibili}} = \frac{(365 * 364 * \dots * (365 - 22))}{365^{23}} = 0.4927$$

Vogliamo ora rendere il calcolo più generico, e quindi

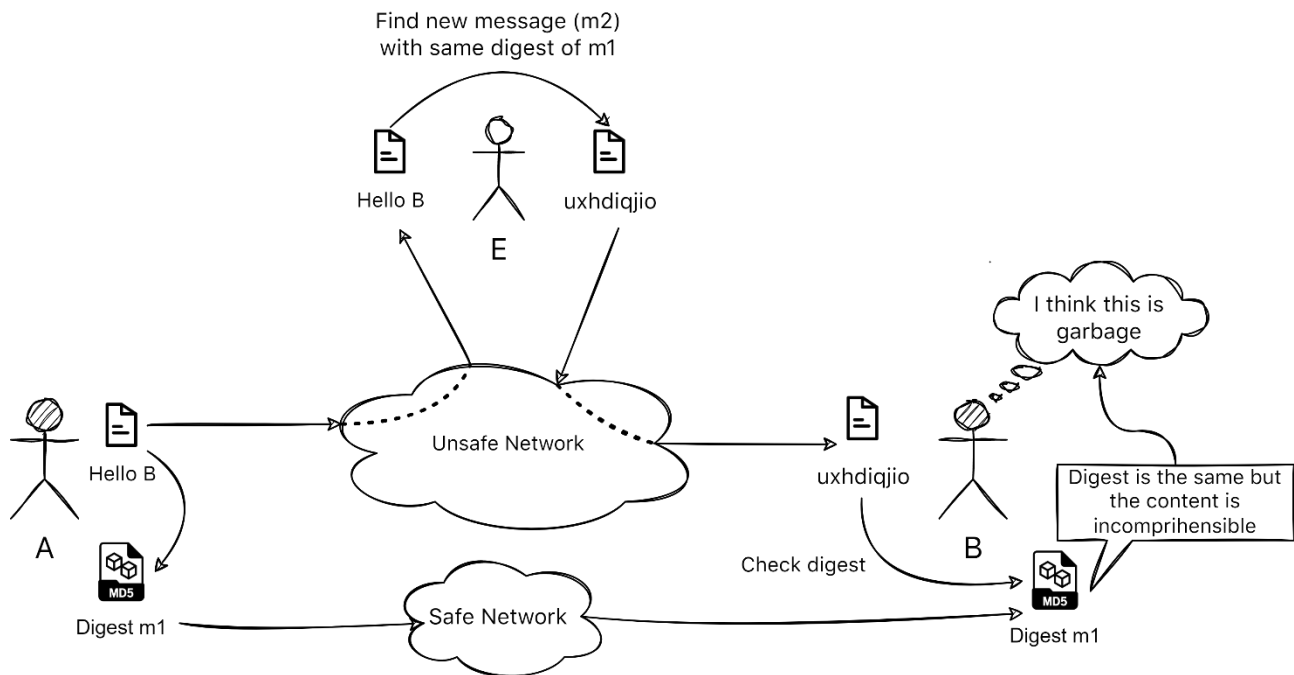
$$P(n,k) = P(\text{almeno una ripetizione in un insieme di } k \text{ elementi scelti tra } n)$$

$$\begin{aligned}
 P(\text{almeno una ripetizione in un insieme di } k \text{ elementi in } n) &= 1 - \frac{n * (n-1) * \dots * (n-k+1)}{n^k} = \\
 &= 1 - \frac{n}{n} * \frac{n-1}{n} * \dots * \frac{n-k+1}{n} = \\
 &= 1 - \left\{ 1 * \frac{n}{n} - \frac{1}{n} * \dots * \frac{n}{n} - \frac{k-1}{n} \right\} = \\
 e^{-x} > 1 - x &\quad \Rightarrow \quad 1 - \left\{ 1 - \frac{1}{n} * \dots * 1 - \frac{k-1}{n} \right\} = \\
 &= 1 - \left\{ e^{-\frac{1}{n}} * \dots * e^{-\frac{k-1}{n}} \right\} = \\
 \sum_{k=0}^{\infty} a_k = a_0 + a_1 + a_3 + \dots = \frac{k(k+1)}{2} &\quad \Rightarrow \quad 1 - \left\{ e^{-\frac{1+\dots+k+1}{n}} \right\} = \\
 \text{fattore è una serie} &= 1 - e^{-\frac{k^2}{2n}}
 \end{aligned}$$

$$\mathbb{P}(n, k) > 0.5 \quad \text{quando} \quad k > 1.18 * \sqrt{n}$$

$\mathbb{P}(n, k) > \frac{1}{2}$
 $1 - e^{-\frac{k^2}{2n}} > \frac{1}{2}$
 Sposto il -1 a destra $-e^{-\frac{k^2}{2n}} > -\frac{1}{2}$
 multiplico per -1 $e^{-\frac{k^2}{2n}} < \frac{1}{2}$
 $x^{-1} < \frac{1}{y}$
 $x > y$
 $e^{\frac{k^2}{2n}} > 2$
 $\frac{k^2}{2n} > \ln(2)$
 applico il logaritmo naturale ad entrambi i membri
 $k^2 > 2(\ln(2)n)$
 porto 2n a destra
 $k > \sqrt{2(\ln(2)n)}$
 $k > \sqrt{\ln(2^2)} * \sqrt{n}$
 $k > 1.18 * \sqrt{n}$

C'è da fare una considerazione: il problema sopra esposto prendeva due valori dall'insieme dei messaggi possibile, che avessero lo stesso digest e che quindi generassero una collisione. Ma considerando questo scenario:



Se il nodo E usasse questo metodo, e quindi di produrre un messaggio m_2 che genera collisione, non riuscirebbe comunque a controllarne il contenuto.

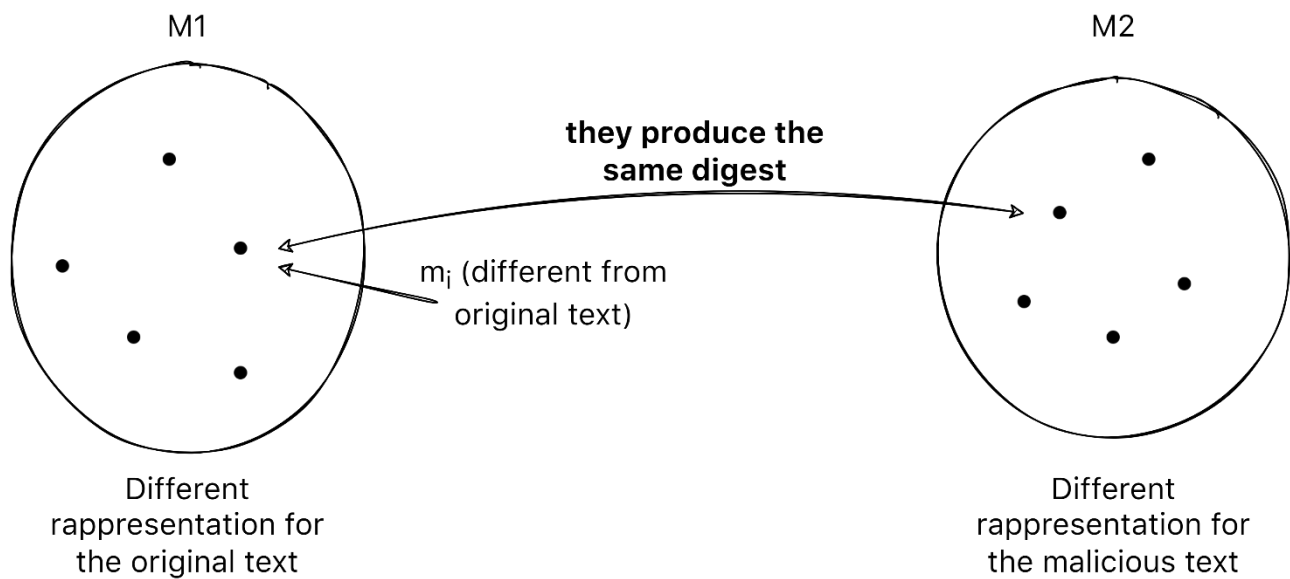
In sostanza, nell'esempio sopra, il nodo E non potrebbe far funzionare la cosa perché il fatto che lui trovi un m_i che generi collisione (m_1 e m_i diversi) non toglie il fatto che m_i non ha alcun significato (è incomprensibile). Quindi noi non vogliamo un messaggio casuale che generi la collisione, ma un caso specifico di messaggio (comprensibile) per ingannare il nodo B.

Se volessi ad esempio falsificare il messaggio: "Pin: 4839" con "Pin: 0000" ma il digest dei due messaggi fosse diverso?

Questo è possibile farlo, in quanto, l'attacco del compleanno è di tipo Chosen Plaintext Attack (CPA): è un modello di attacco per la crittoanalisi che presume che l'attaccante possa ottenere i testi cifrati per testi in chiaro arbitrari. L'obiettivo dell'attacco è ottenere informazioni che riducano la sicurezza dello schema di crittografia.

L'attaccante, quindi, produrrà diverse versioni del messaggio "Pin: 4839" e diverse versioni del messaggio "Pin: 0000". Avremo quindi due insiemi:

- Uno contenente tutte le possibili versioni del messaggio originario (diversi ma accettabili)
- Uno contenente tutte le possibili versioni del messaggio malevolo (diversi ma accettabili e comprensibili).




Nello schema sopra, m_i è una rappresentazione diversa per il messaggio originale ma che produce un digest identico ad uno dei possibili messaggi malevoli. Dato che il modello di attacco è di tipo CPA, è possibile a questo punto, da parte del nodo E, di comunicare al nodo A di rimandare il messaggio m_i , a quel punto intercettarlo e mandare il messaggio malevolo che produce lo stesso hash di m_i .

Qual è, a questo punto, la probabilità di trovare due messaggi con lo stesso digest, presi da due insiemi diversi?


\mathbb{P} (almeno un elemento comune in due insiemi di k elementi scelti tra n)

$X = \{x_1, x_2, \dots, x_n\}$  Assunzione: sono tutti elementi diversi

$Y = \{y_1, y_2, \dots, y_n\}$  Assunzione: sono tutti elementi diversi

$\mathbb{P}(x_1 = y_1) = \frac{1}{n}$  La probabilità che x_1 sia uguale ad un elemento in Y

$\mathbb{P}(x_1 \neq y_2) = 1 - \frac{1}{n}$  La probabilità che x_1 sia diverso da un elemento in Y

$\mathbb{P}(x_1 \neq y_i) = (1 - \frac{1}{n})^k$  La probabilità che x_1 non sia uguale a nessun elemento in Y (x_1 non è contenuto in Y)
 $\forall y_i \in Y$

$$\mathbb{P}(n, k) = 1 - ((1 - \frac{1}{n})^k)^k > 1 - ((e^{-\frac{1}{n}})^k)^k > \boxed{1 - e^{-\frac{k^2}{n}}}$$

 $e^{-x} > 1 - x$

La probabilità è quella indicata in verde.

Una volta trovata la formula per il calcolo della probabilità si può dimostrare che:

$$\mathbb{P}(n, k) > \frac{1}{2} \mid k > 0.83 * \sqrt{n}$$

La probabilità è almeno 0.5 quando k è almeno maggiore di 0.83 per la radice di n.

$$\mathbb{P}(n, k) > \frac{1}{2} \mid k > 0.83\sqrt{n}$$

$$\begin{array}{ll} \mathbb{P}(n, k) > \frac{1}{2} & \xrightarrow{\text{applico il logaritmo naturale ad entrambi i membri}} \frac{k^2}{n} > \ln(2) \\ 1 - e^{-\frac{k^2}{n}} > \frac{1}{2} & \xrightarrow{\text{porto n a destra}} k^2 > \ln(2)n \\ \text{Sposto il -1 a destra} \quad -e^{-\frac{k^2}{n}} > -\frac{1}{2} & k > \sqrt{\ln(2)n} \\ \text{moltiplico per -1} \quad e^{-\frac{k^2}{n}} < \frac{1}{2} & k > \sqrt{\ln(2)}\sqrt{n} \\ x^{-1} < \frac{1}{y} \quad e^{\frac{k^2}{n}} > 2 & k > 0.83\sqrt{n} \end{array}$$

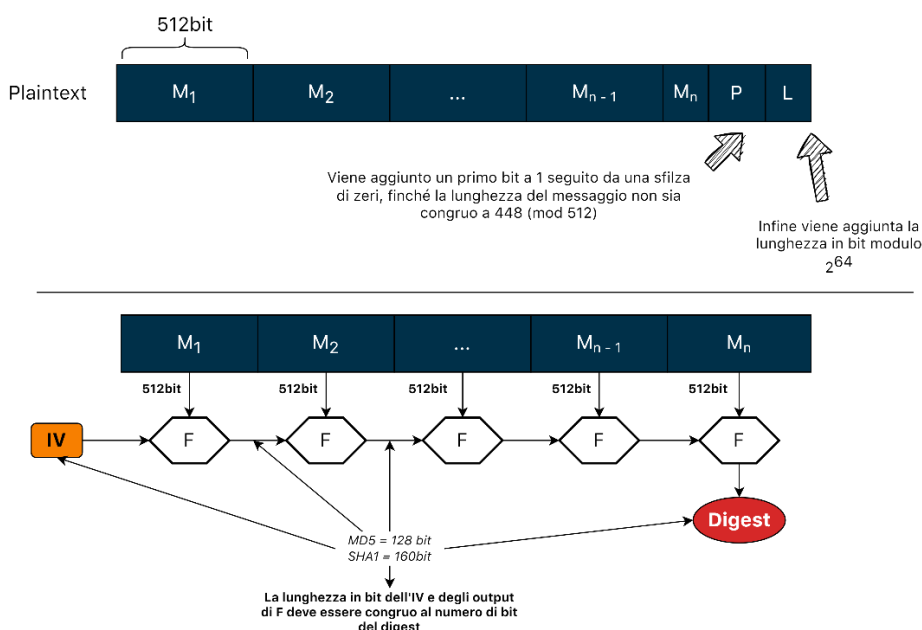
$x > y$

Questo dimostra che presi due insiemi, è comunque molto probabile trovare due messaggi che collidono sullo stesso digest. **Ciò impone ai sistemi crittografici di scegliere dei digest con numeri di bit almeno di 256bit** per rendere la funzione di hash un minimo sicura.

Più sicurezza in Hash

Avere un numero di bit sufficientemente grande non implica che la funzione di hash sia sufficientemente sicura, è necessario anche che la funzione operi sui bit/blocchi del plaintext, "mescolandoli" in modo tale da rendere la funzione più sicura e rendere più difficile trovare metodologie per la generazione di collisioni.

Questo è uno degli schemi più usati dagli algoritmi di hash per generare dei digest sicuri:



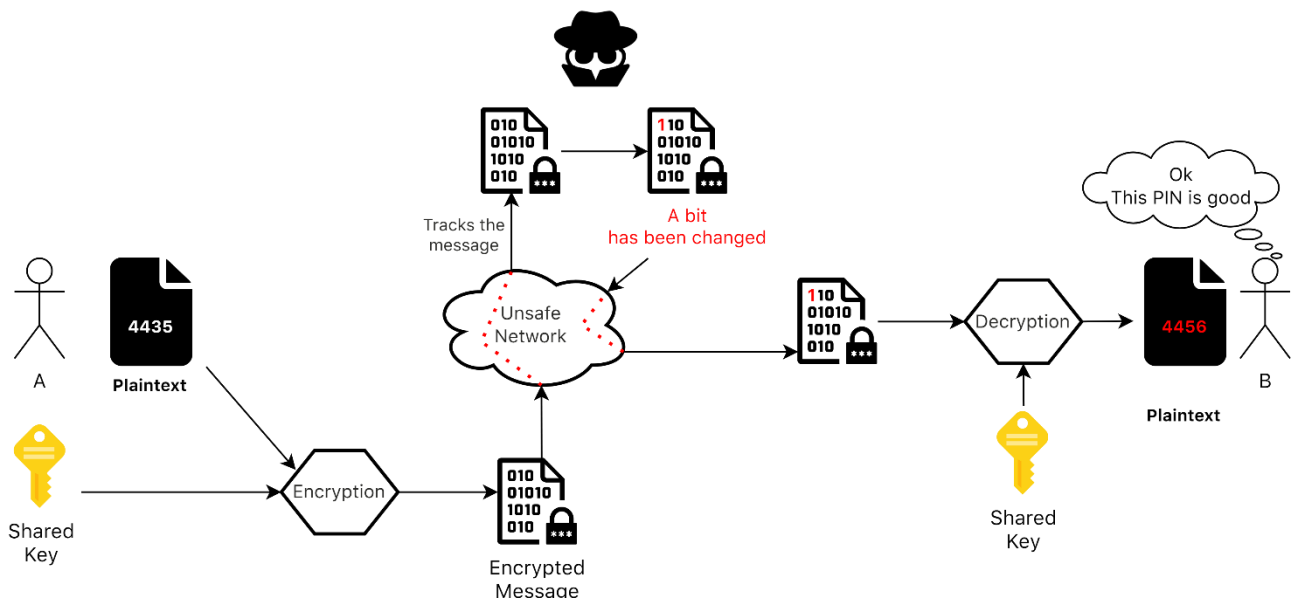
Autenticazione

Dopo aver trattato l'integrità, andremo a trattare un altro degli obiettivi di sicurezza: l'autenticazione. Con l'autenticazione è possibile avere la certezza che il destinatario, a fronte di un messaggio ricevuto, verifichi che provenga dal mittente desiderato.

L'autenticazione può avvenire sia in un contesto simmetrico che asimmetrico:

- L'autenticazione simmetrica si basa su cifrari simmetrici, utilizzando la chiave condivisa.
- L'autenticazione asimmetrica (o firma elettronica) si basa su cifrari asimmetrici, utilizzando in fase di firma la chiave privata di chi firma.

Attenzione che: un messaggio cifrato non implica che quest'ultimo sia autenticato. Questo concetto si è dato per scontato per anni nel mondo della sicurezza informatica; si pensava al fatto che bastasse cifrare il plaintext per aggiungere l'autenticazione come livello di sicurezza (oltre alla riservatezza). Questo però è dimostrabile non essere vero.

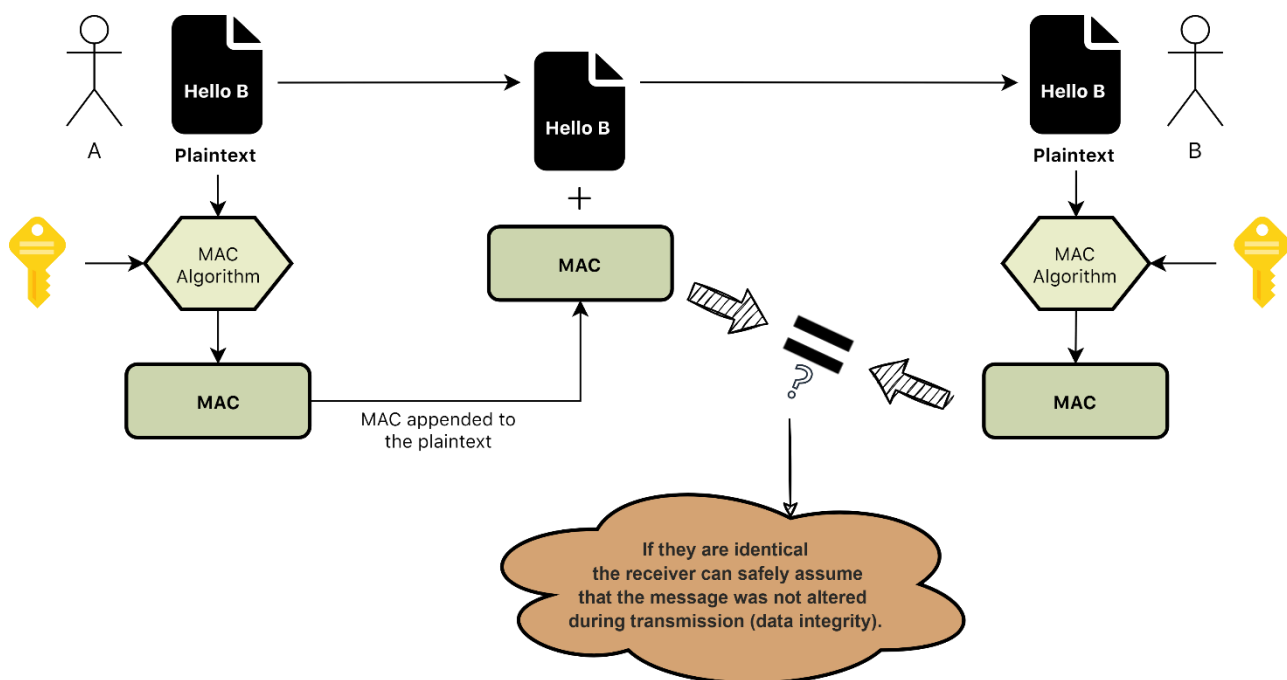


L'esempio sopra è uno dei possibili scenari che dimostrano che solo la cifratura del plaintext non è sufficiente a garantire autenticazione ed integrità: l'attaccante intercetta il messaggio, cambiando un bit del messaggio cifrato. Se fosse un messaggio di senso compiuto, scritto in qualche lingua esistente, probabilmente dopo aver cambiato il bit, quel messaggio non avrebbe più senso per l'effetto cascata dei cifrari a blocchi. In questo caso il plaintext è un codice/sequenza di numeri. In questo caso, cambiando un singolo bit, è possibile che si arrivi ad avere ancora una sequenza di codici/numeri ma che non corrispondono al messaggio originale.

Con l'effetto cascata avremo comunque un messaggio totalmente diverso (bit parlando) ma essendo anch'esso una sequenza di numeri/codici, per chi riceve il messaggio è come se non fosse successo nulla.

Autenticazione Simmetrica

In un contesto simmetrico (con chiave condivisa), l'autenticazione di messaggio si basa sul MAC (Message Authentication Code): noto anche come tag, è una breve informazione utilizzata (insieme al plaintext o ciphertext) per autenticare un messaggio.



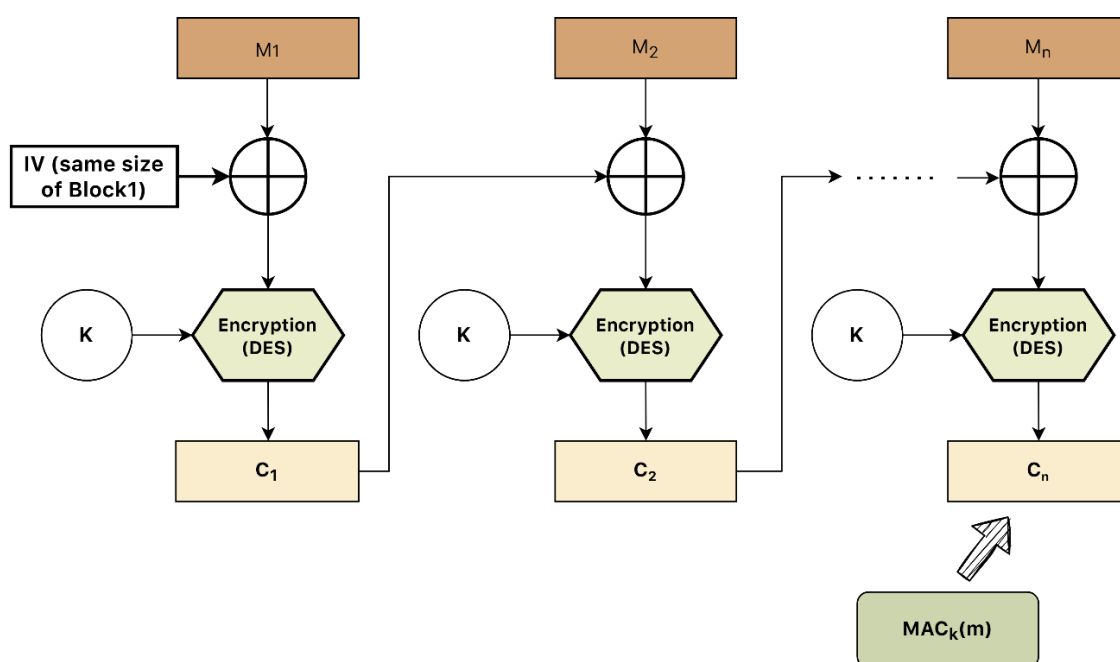
I valore MAC, quindi, protegge l'integrità dei dati di un messaggio, nonché la sua autenticità, consentendo a chi verifica (quindi chi possiede la chiave simmetrica) di rilevare eventuali modifiche al contenuto del messaggio.

Le funzioni MAC condividono somiglianze con le funzioni hash crittografiche, tuttavia soddisfano requisiti di sicurezza diversi. Lo scopo di un MAC è autenticare l'origine di un messaggio e la sua integrità. A differenza di un hash, il MAC può essere generato solo da chi possiede la chiave condivisa.

MAC-CBC

Uno dei modi per generare il tag è attraverso l'uso di cifrari a blocchi.

Uno dei cifrari più utilizzati per generare il MAC è utilizzando CBC (CBC-MAC):



Il blocco C_n del CBC sarà anche il tag del messaggio ($MAC_k(M)$).

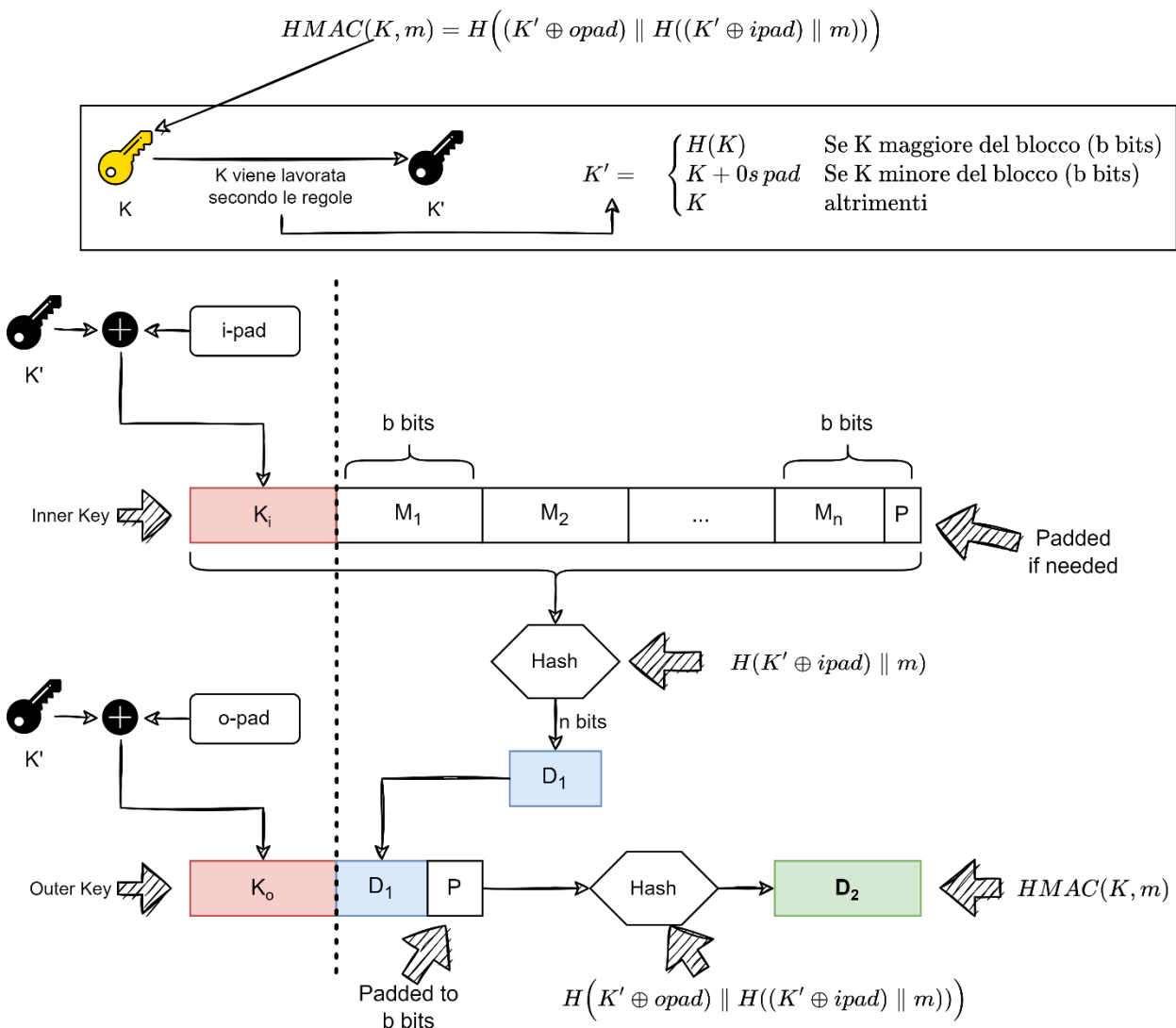
Questo rende il messaggio protetto da integrità ed autenticazione: se anche solo un bit venisse alterato, per l'effetto cascata del CBC, anche il blocco C_n risulterà alterato e di conseguenza anche il MAC.

Inoltre, non è possibile generare messaggi falsi, perché solo chi conosce la chiave genererà MAC validi per gli altri interlocutori con cui condivide la chiave. Di conseguenza, anche l'attacco del compleanno non può essere efficace utilizzando MAC-CBC: questo perché l'attaccante non riuscirebbe a generare le collisioni non conoscendo la chiave condivisa.

HMAC

Un altro modo per generare il MAC è utilizzare come primitiva crittografica l'hashing.

Qualsiasi funzione di hash, come SHA-2 o SHA-3, può essere utilizzata nel calcolo di un HMAC. L'algoritmo MAC risultante è chiamato HMAC-X, dove X è la funzione hash utilizzata (es. HMAC-SHA256 o HMAC-SHA3-512). La robustezza dell'HMAC dipende dalla forza crittografica della funzione hash sottostante, dalla dimensione del suo output hash e dalla dimensione e dalla qualità della chiave.



Questo è lo schema per la generazione di un **keyed-hash message authentication code** (HMAC), i passi sono semplici:

1. Partendo dalla chiave in input, si calcola la chiave K' secondo le regole indicate
2. Si suddivide il messaggio in b bits (b è in funzione dell'algoritmo che si sta utilizzando: se si usa SHA-1, ad esempio, i blocchi saranno di 512bit).
3. Si antepone al messaggio la chiave interna K_i :
 - a. Questa si calcola prendendo la chiave K' precedentemente calcolata e mettendola in XOR con un blocco chiamato "i-pad" (internal padding), che è una sequenza di 8 bit (0x36) ripetuti fino a raggiungere la lunghezza di un blocco (b bit).
4. Si crea il primo digest (D_1) che sarà di n bit (esempio SHA-1: 160bit).
5. Si antepone ora la chiave esterna K_o al primo digest D_1
 - a. Questa si calcola prendendo la chiave K' precedentemente calcolata e mettendola in XOR con un blocco chiamato "o-pad" (outer padding), che è una sequenza di 8 bit (0x5C) ripetuti fino a raggiungere la lunghezza di un blocco (b bit).
6. Il Digest D_1 viene paddato per raggiungere la lunghezza di un blocco (b bit)
7. Infine, si riesegue l'algoritmo di hash creando il secondo digest: questo sarà il nostro tag (HMAC).

Come per il MAC-CBC è immune ad attacchi di tipo CPA perché l'attaccante non conosce la chiave condivisa.

Inoltre, è più efficiente di MAC-CBC: è efficiente tanto quanto la funzione di HASH sottostante che si utilizza (H viene chiamata due volte, ma la seconda volta con un argomento lungo solo $2b$).

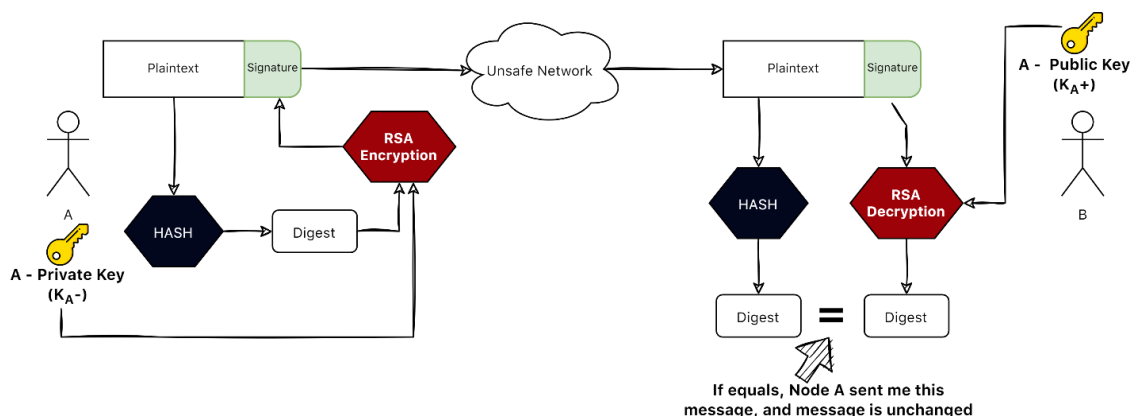
Principio di non disconoscibilità

Con i costrutti spiegati fino ad ora (MAC, HMAC) non è possibile identificare con precisione l'autore del tag: dato che siamo in simmetrica, la chiave condivisa può essere posseduta da più membri del sistema. Chiunque può essere in grado di produrre un tag partendo dalla chiave condivisa e concatenarlo al plaintext. Quindi in questo caso il messaggio è disconoscibile (se mando un messaggio potrò dire di non essere stato io). La non disconoscibilità viene aggiunta nel contesto asimmetrico con le chiavi private.

Firma Elettronica

La firma elettronica è il dato che si utilizza per verificare integrità e autenticazione di messaggio in un contesto asimmetrico.

A differenza del MAC, un messaggio autenticato con firma elettronica non è disconoscibile: se si cifra con la propria chiave privata il messaggio, non si potrà negare di non aver firmato quel messaggio (come nella vita reale).



La firma viene aggiunta in coda al messaggio e poi spedito in rete.

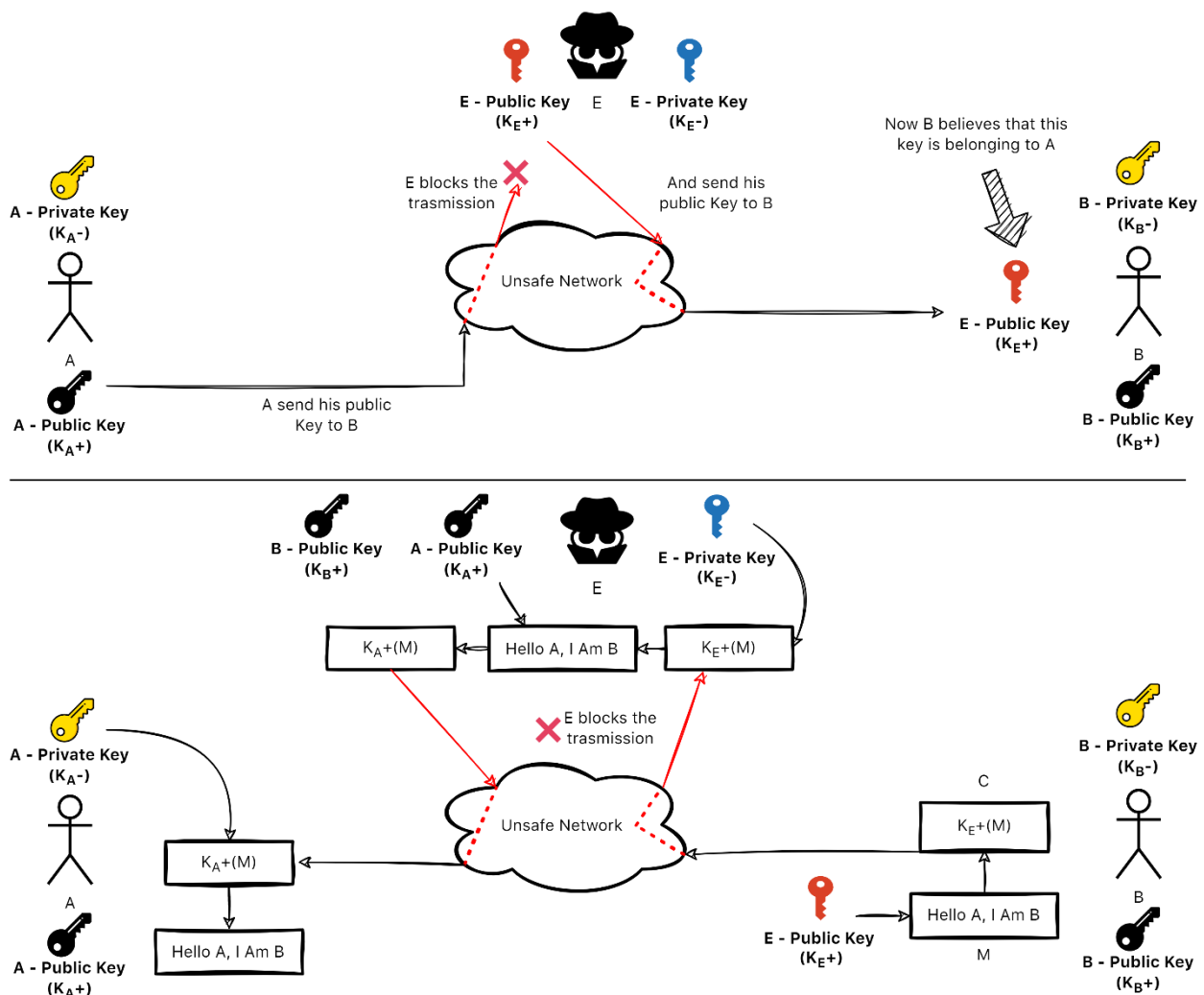
Da non confondere quali contesti si usa la chiave pubblica e in quali quella privata:

- Per creare il signature di un messaggio si usa la chiave privata di chi firma
- Per cifrare un messaggio invece si usa la chiave pubblica di chi dovrà ricevere il messaggio.

Problema di distribuzione della chiave pubblica

Nella crittografia simmetrica, lo scambio della chiave condivisa (come abbiamo visto) è risolto utilizzando Diffie-Hellman.

Nello scenario asimmetrico si ha lo stesso problema: Non è consigliato scambiare chiavi pubbliche su una rete non sicura, in quanto è possibile essere soggetti a MITM (Man in the middle attack):



A manda la sua chiave pubblica a B. L'attaccante interrompe la trasmissione ed inoltra a B quella che invece è la chiave pubblica di E. **B sarà sicuro a quel punto che la chiave pubblica ricevuta sia quella di A**, ed userà quella chiave per cifrare la conversazione.

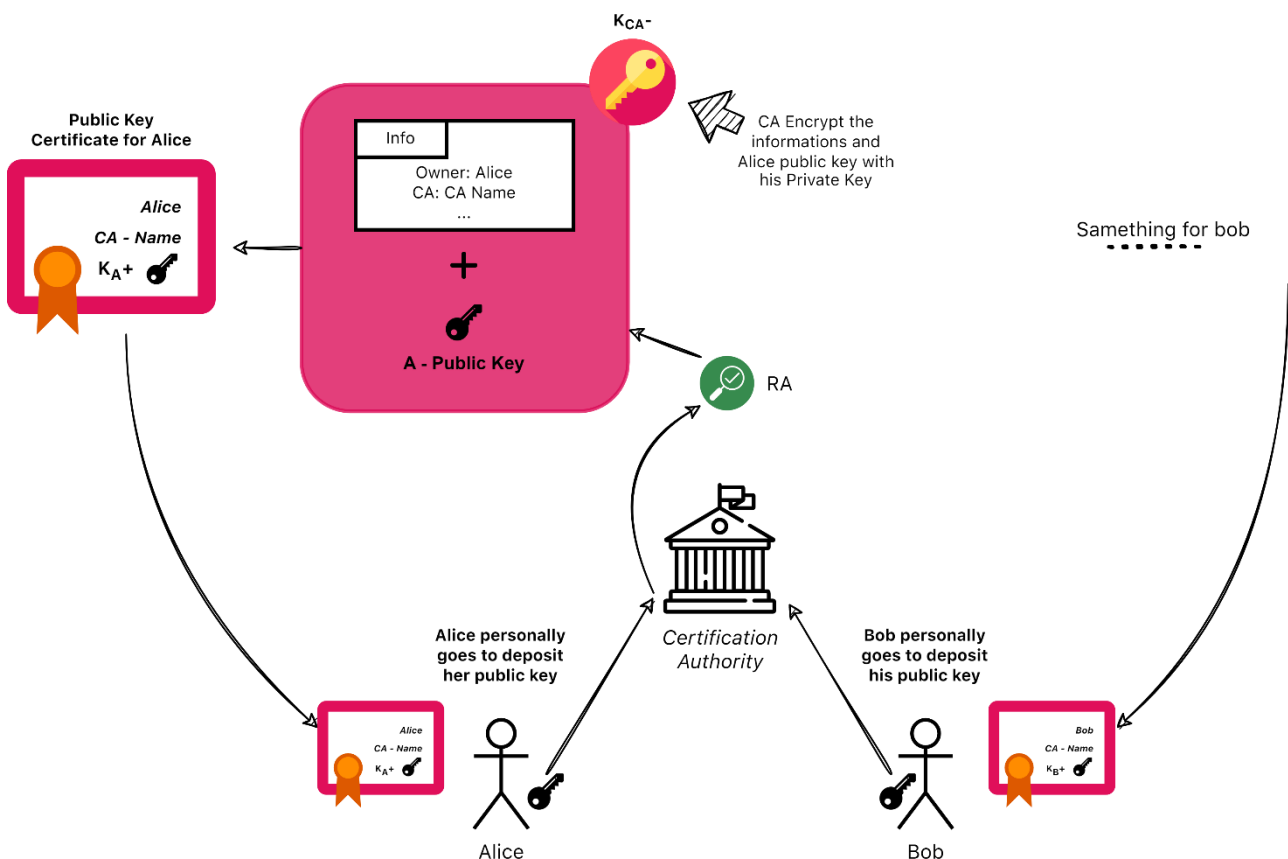
Il nodo B inoltra un messaggio ad A utilizzando quella che crede sia la chiave pubblica di A. L'attaccante (E) intercetta il messaggio, decifra il messaggio e lo cifra con la chiave pubblica di A. il nodo A, quel punto, riceve il messaggio ignaro del pericolo. **Il nodo E sniffa tutto il traffico riuscendo a comprendere anche il contenuto.**

Ovviamente la procedura è effettuata anche al contrario, quindi quando B manderà la sua chiave pubblica ad A, e quindi il gioco è fatto per l'attaccante.

A questo punto i due interlocutori devono trovare un metodo per scambiarsi le chiavi pubbliche così da evitare casi di MITM: i due nodi potrebbero incontrarsi dal vivo ma questo sarebbe poco efficiente. **Nella vita reale abbiamo adottato invece un sistema chiamato certificato di chiave pubblica.**

Certificato di chiave pubblica

Il certificato di chiave pubblica è un documento elettronico che attesta l'associazione univoca tra una chiave pubblica e l'identità di un soggetto.



Il certificato viene rilasciato da un ente di certificazione (CA – Certification Authority), di cui tutti si fidano. L'ente di certificazione appena riceve una chiave da certificare di Alice, verifica che Alice sia veramente chi dice di essere (questa fase è detta RA – Registration Authority). Una volta che la CA approva l'identità di Alice, cripta la chiave pubblica di Alice con la chiave privata della CA stessa, insieme ad alcune informazioni (Owner della chiave, nome della CA che ha rilasciato il certificato, etc..).

Anche Bob farà la cosa specularmente rispetto ad Alice.

Più in generale, **chiunque voglia comunicare in rete accertandosi che l'interlocutore sia chi dice di essere senza andare in contro ad attacchi MITM, verificherà che la firma sia valida e, qualora lo sia, può utilizzare tale chiave (contenuta nel certificato) per comunicare in modo sicuro con il soggetto del certificato.**

L'attacco MITM con l'utilizzo dei certificati a chiave pubblica non è appunto possibile:

