

Appunti di Cybersecurity

Andrea Bellu

24 ottobre 2025

Questi appunti sono stati compilati con l'aiuto di Gemini. Possono contenere errori.

Indice

1	Introduzione alla Cybersecurity	4
1.1	Perché Cybersecurity? Il Contesto di Industria 4.0	4
1.2	Esempi di CyberThreats	4
1.3	Il fattore umano e la convergenza delle reti	5
2	Reti a Pacchetti e Convergenza	5
2.1	Concetti Base delle Reti a Pacchetti	5
2.2	Evoluzione e Convergenza delle Reti	6
3	Meccanismi e Pilastri della Sicurezza	6
3.1	I Servizi di Sicurezza (Pilastri)	6
3.2	Crittografia	6
3.3	Tipi di Attacchi	7
4	Conclusioni: Relatività e Standard	7
5	Richiami sulle Reti a Pacchetti	7
5.1	Concetti di Base	7
5.2	Pacchettizzazione e Indirizzamento	8
5.3	Stack Protocollore TCP/IP	8
5.4	Modello di Comunicazione: Host vs Router	9

6	Introduzione alla Crittografia	9
6.1	Crittologia: Definizioni	10
6.2	Terminologia: Gli Attori	10
6.3	Terminologia: Notazione e Operatori	10
7	Schema Crittografico Generale	11
7.1	Classi di Algoritmi Crittografici	11
7.1.1	Block Cipher (Cifrario a Blocchi)	11
7.1.2	Stream Cipher (Cifrario a Flusso)	11
8	Crittanalisi e Sicurezza	11
8.1	Ipotesi di Kerckhoffs	11
8.2	Penetrabilità e Tipi di Attacco	12
8.3	Segretezza Perfetta (Shannon)	12
8.3.1	One-Time-Pad (OTP)	12
9	Tipologie Base di Algoritmi Crittografici	12
9.1	Crittografia Simmetrica: Block Cipher	13
9.2	Obiettivi e Tipologie dei Block Cipher	14
9.3	Prodotto di Cifrari: Confusione e Diffusione	14
9.4	Data Encryption Standard (DES)	15
9.5	Advanced Encryption Standard (AES)	18
9.6	International Data Encryption Algorithm (IDEA)	19
9.7	Blowfish (BF)	20
9.8	Stream Cipher	20
9.8.1	Stream Cipher: Synchronous	21
9.8.2	Stream Cipher: Self-Synchronizing	22
9.9	Modi d'uso dei Block Cipher	23
9.10	Definizioni e Tipologie	26
9.11	Proprietà di Sicurezza (MDC)	27
9.12	Algoritmi Notevoli (MDC e MAC)	27

10 Crittografia Asimmetrica	28
10.1 Schema e Concetti Fondamentali	28
10.2 Obiettivi e Basi Teoriche	29
10.3 Richiami: Teorema di Eulero	29
11 Algoritmo RSA (Rivest, Shamir, Adleman)	30
11.1 Generazione delle Chiavi	30
11.2 Utilizzi di RSA	31
11.2.1 1. Confidenzialità	31
11.2.2 2. Firme Digitali	31
11.2.3 3. Instaurazione di Chiavi Effimere	32
11.3 Sicurezza e Uso Pratico di RSA	33
12 Protocollo Diffie-Hellman (DH)	33
12.1 Meccanismo di Base	34
12.2 Sicurezza di DH	34
13 Elliptic Curve Cryptography (ECC)	35
14 Public Key Infrastructure (PKI)	35
14.1 Il Problema dell'Autenticità	35
14.2 Certificati Digitali (X.509)	35
14.3 Processo di Verifica del Certificato	36
14.4 Catene di Certificati e Revoca	37

1 Introduzione alla Cybersecurity

1.1 Perché Cybersecurity? Il Contesto di Industria 4.0

Il bisogno di cybersecurity è cresciuto esponenzialmente con l'arrivo dell'**Industria 4.0** (ora "Impresa 4.0"). Questo paradigma si basa su tecnologie abilitanti che interconnettono l'intera filiera produttiva.

- Questa interconnessione (es. feedback dalle vendite alla produzione) crea **molte nuove possibilità per fare danni**.
- L'elemento fondamentale è la necessità di **meccanismi per garantire la sicurezza nelle comunicazioni**.

1.2 Esempi di CyberThreats

Le minacce informatiche (cyberthreats) dimostrano la necessità di protezione in vari ambiti:

- **Robot Industriali:**
 - Attacco tramite smartphone infetto che, una volta nella rete aziendale, si finge il repository per gli aggiornamenti.
 - Il robot scarica così codice malevolo.
 - **Problema di fondo:** Rete aziendale per i client non separata dalla rete di produzione.
 - **Danno subdolo:** Modificare il movimento di pochi mm, creando migliaia di pezzi difettosi.
- **Stuxnet:**
 - Virus sviluppato da governi (US contro Iran).
 - È entrato nella rete isolata della centrale nucleare di Natanz tramite una **chiavetta USB**.
 - **Obiettivo:** Attaccare i PLC di controllo delle centrifughe (Siemens).
- **Domino's Pizza:**
 - L'app per smartphone gestiva tutto il processing, **senza un double-check lato server**.
 - Era possibile inviare un ordine alla produzione senza aver pagato.
 - **Lezione:** L'app deve essere un'interfaccia, il processing va fatto sul server.

- **Chipset Bluetooth Broadcom:**
 - È stato scoperto un comando di debug via wireless.
 - Esempio di fallimento della "Security by Obscurity".
- **WhatsApp (CVE-2019-3568):**
 - Vulnerabilità di tipo **Buffer Overflow** durante l'instaurazione di una sessione VoIP.
 - Inviando un messaggio "opportuno" (più lungo del buffer), l'attaccante poteva sovrascrivere altre parti di memoria.
 - **Danno:** Installare software simile a Pegasus per spiare l'utente (camera, microfono, ecc.).
- **Virgin Media O2:**
 - Configurazione errata del software IMS (IP Multimedia Subsystem).
 - Informazioni sensibili (Cell ID, IMSI/IMEI) venivano incluse negli **header del protocollo SIP**.
 - **Danno:** Permetteva a un attaccante di mappare la posizione geografica degli utenti.

1.3 Il fattore umano e la convergenza delle reti

Molti problemi di sicurezza sono legati alla "pigrizia" (*laziness*):

- Protocolli senza autenticazione.
- Paradigma "security by obscurity" (es. GSM).
- Password fisse (magari appuntate su una lavagna).
- Mancanza di budget/tempo per aggiornare hw/sw (WannaCry docet).

2 Reti a Pacchetti e Convergenza

2.1 Concetti Base delle Reti a Pacchetti

- Una rete a pacchetto è un insieme di nodi connessi da canali, dove l'informazione è divisa in pacchetti.
- Internet è un'unione di sottoreti tra loro interconnesse.

- **Concetto Chiave:** Nello schema originale di Internet, i nodi intermedi non offrono **nessun servizio di sicurezza**.
- Tutta la sicurezza deve essere gestita **dai nodi terminali** (principio "end-to-end").

2.2 Evoluzione e Convergenza delle Reti

- **Reti LAN:** Si è passati da reti cablate (switch) a reti wireless (Access Point). Questo introduce il rischio di "ascolto" (*eavesdropping*).
- **Reti Industriali:** Si è passati da controlli punto-punto (cablaggio complesso) a Bus di campo (Fieldbus) e infine a **sistemi a pacchetto** (es. Profinet, Wi-Fi).
- **Convergenza:** Oggi, i problemi di sicurezza sono gli stessi ovunque (casa, azienda, produzione) a causa dell'uso delle stesse tecnologie di rete.

3 Meccanismi e Pilastri della Sicurezza

3.1 I Servizi di Sicurezza (Pilastri)

Per proteggere le comunicazioni servono "servizi" specifici:

1. **Autenticazione:** Identificare *chi* partecipa alla comunicazione (utente, nodo, applicazione) chiedendo una "prova".
2. **Controllo di Accesso:** Verificare i *diritti* di un partecipante (già autenticato) ad accedere a una risorsa. È *successivo* all'autenticazione.
3. **Confidenzialità:** Garantire che solo chi è autorizzato possa *leggere* le informazioni (sia in transito che memorizzate).
4. **Integrità (Integrity Protection):** Garantire che chi non è autorizzato non possa *modificare* l'informazione senza essere scoperto.
5. **Non Ripudio:** Garantire che un'entità non possa *negare* in seguito di aver generato un'informazione o partecipato a un processo.

3.2 Crittografia

- La crittografia è il "blocco fondamentale" per ottenere meccanismi di sicurezza "forti".
- Include sistemi a chiave simmetrica (private key), asimmetrica (public key), Hash e MAC.

- **Principio di Kerckhoffs:** La sicurezza di un sistema deve dipendere dalla **segretezza della chiave**, e non dalla segretezza del sistema/algoritmo (come invece si pensava per la macchina Enigma).

3.3 Tipi di Attacchi

- **Attacchi Passivi:** L'attaccante può solo catturare e analizzare i dati (es. intercettazione).
- **Attacchi Attivi:** L'attaccante può ricevere, *modificare* e re-immettere i dati in rete.

4 Conclusioni: Relatività e Standard

- **La Sicurezza Assoluta Non Esiste:** È sempre *relativa* all'ambito applicativo e al *valore* di ciò che si protegge.
- **Compromesso:** Ogni meccanismo è un compromesso tra costo, complessità e livello di protezione.
- **Sicurezza di Sistema vs. di Rete:** La prima riguarda il singolo nodo (HW, SW, OS), la seconda la comunicazione *tra* i nodi (protocolli).
- **Normative (EU):** Dal 2022 in Europa è presente la **NIS2** (Network and Information Security 2). Ha l'obiettivo di aumentare il livello di sicurezza in settori critici (energia, sanità, PA, ecc.) imponendo obblighi di gestione del rischio.
- **Framework:** Il **NIST Cybersecurity Framework (CSF)** è un esempio di approccio sistematico per gestire i rischi (e può essere usato per adeguarsi alla NIS2).

5 Richiami sulle Reti a Pacchetti

5.1 Concetti di Base

Una rete di telecomunicazione è un insieme di nodi connessi da canali di comunicazione, con lo scopo di trasportare informazione da una sorgente a una destinazione. In una **rete a pacchetto**, l'informazione viene suddivisa in pacchetti, che viaggiano attraverso la rete, potenzialmente seguendo strade differenti. **Internet** è un esempio di rete a pacchetto, definita come un'unione di sottoreti tra loro interconnesse.

Esistono due livelli di comunicazione:

- **Comunicazione Logica:** La comunicazione come vista dagli applicativi (es. dalla webcam allo smartphone).
- **Comunicazione Fisica:** Il percorso reale che i dati compiono attraverso la rete, passando di nodo in nodo (switch, router, ecc.) fino alla destinazione.

5.2 Pacchettizzazione e Indirizzamento

Il flusso di dati prodotto da un'applicazione (es. un video compresso) viene "pacchettizzato", ovvero spezzato in pezzetti più piccoli. Questo processo introduce due problematiche fondamentali:

1. **Riordino:** I pacchetti possono viaggiare su percorsi diversi e arrivare fuori ordine. È necessario un *numero di sequenza* per riordinarli a destinazione.
2. **Indirizzamento:** I pacchetti devono essere indirizzati correttamente.

Per gestire l'indirizzamento, vengono aggiunte informazioni (header) a due livelli principali:

- **Header di Trasporto:** Specifica l'applicazione sorgente (N) e l'applicazione destinazione (M) sul nodo. Questi identificativi sono detti **porte**.
- **Header di Rete:** Specifica il nodo sorgente ($addr_{src}$) e il nodo destinazione ($addr_{dst}$) sulla rete globale. Questi sono gli **indirizzi IP**.

5.3 Stack Protocollore TCP/IP

Questo processo di aggiunta di header (incapsulamento) segue un approccio a livelli (o strati), noto come "divide et impera". L'architettura standard di Internet è la **TCP/IP**, formalizzata in 5 livelli.

Livello 5: Applicativo Fornisce i servizi all'utente (es. Web, E-mail, Messaggistica). I dati prodotti qui sono il "payload" per il livello sottostante.

Livello 4: Trasporto Fornisce un canale di trasporto *end-to-end* (tra le due applicazioni). Introduce l'header di trasporto (con le porte). Offre due protocolli principali:

- **TCP (Transmission Control Protocol):** Orientato alla connessione e *affidabile*. Gestisce il riordino, il recupero degli errori (ritrasmissione) e il controllo di flusso/congestione.
- **UDP (User Datagram Protocol):** *Connectionless* e *non affidabile* (best-effort). Non imposta una connessione e non ritrasmette i pacchetti persi.

Livello 3: Rete Gestisce il trasferimento dei pacchetti tra nodi *su tutta la rete* (inter-networking). Il protocollo è **IP (Internet Protocol)**.

- Aggiunge l'header di rete, che contiene l'indirizzo IP sorgente e destinazione (es. 192.168.0.0/16 per reti private).
- Il servizio IP non è affidabile e non garantisce l'ordine: i pacchetti possono essere persi, duplicati o alterati (best-effort).
- L'instradamento (**routing**) è l'operazione chiave di questo livello. Ogni router prende una decisione locale ("hop-by-hop") basandosi sulla propria tabella di routing per inoltrare il pacchetto al nodo successivo.

Livello 2: Data-link Fornisce un canale di comunicazione affidabile tra macchine *adiacenti* (sullo stesso cavo o AP).

- **Framing:** Incapsula il pacchetto IP in una "trama" (frame), aggiungendo un header (H) e un trailer (T).
- **Indirizzamento Fisico:** Usa un indirizzo fisico univoco detto **MAC** (es. AA:BB:CC:DD:EE:FF) per identificare i nodi sulla LAN (es. Ethernet, Wi-Fi).
- **Controllo d'Errore:** Il trailer contiene un *checksum* per rilevare se la trama è stata corrotta durante la trasmissione.
- **Accesso al Mezzo:** Gestisce la condivisione del canale (es. CSMA/CA per Wi-Fi).

Livello 1: Fisico Gestisce la trasmissione del singolo flusso di bit sul canale fisico (impulsi elettrici su rame, luce su fibra, onde radio).

5.4 Modello di Comunicazione: Host vs Router

Il funzionamento dei livelli è diverso tra un nodo terminale (Host) e un nodo intermedio (Router):

- Un **Host** (sorgente o destinazione) implementa tutti e 5 i livelli dello stack TCP/IP.
- Un **Router** implementa solo i livelli 1, 2 e 3 (Fisico, Data-link, IP). Il suo compito è ricevere una trama L2, estrarre il pacchetto L3 (IP), consultare la tabella di routing, e re-incapsularlo in una nuova trama L2 per inviarlo all'hop successivo.

6 Introduzione alla Crittografia

La crittografia è il "building-block" fondamentale per garantire in forma "forte" le proprietà di sicurezza come autenticazione, confidenzialità, integrità e non ripudio.

6.1 Crittologia: Definizioni

La **Crittologia** è il filone di ricerca che si occupa della segretezza delle comunicazioni. Si divide in tre branche:

- **Crittografia**: lo studio dei metodi matematici (algoritmi) per trasformare i dati in modo da renderli incomprensibili a chi non è autorizzato.
- **Protocolli crittografici**: i meccanismi pratici che usano gli algoritmi crittografici per raggiungere un obiettivo (es. autenticazione).
- **Crittanalisi**: lo studio dei meccanismi per "rompere" (circonvenire) i metodi e i protocolli crittografici.

6.2 Terminologia: Gli Attori

Nei protocolli crittografici, si usano nomi convenzionali per identificare i partecipanti:

- **Alice e Bob**: i due partecipanti legittimi che vogliono comunicare in modo sicuro.
- **Eve** (Eavesdropper): un'intrusa **passiva**. Si limita ad ascoltare il canale di comunicazione.
- **Trudy** (Intruder): un'intrusa **attiva**. Può intercettare, modificare, eliminare o creare nuovi messaggi.

6.3 Terminologia: Notazione e Operatori

m (**plaintext**) È il messaggio in chiaro, appartenente allo spazio dei messaggi M .

c (**ciphertext**) È il messaggio criptato, appartenente allo spazio C .

k (**key**) È la chiave, appartenente allo spazio delle chiavi K .

Nelle reti moderne, si usano quasi esclusivamente alfabeti binari (stringhe di 0 e 1). I dati sono spesso rappresentati in **esadecimale (hex)**.

L'operatore fondamentale è lo **XOR** (scritto come \oplus), che è una somma binaria a 1-bit *senza riporto*.

\oplus	0	1
0	0	1
1	1	0

Tabella 1: Tavola di verità dell'operatore XOR.

7 Schema Crittografico Generale

Un sistema crittografico è definito da due funzioni:

- **Cifratura:** $E_{k1}(m) = c$ (dove $k1$ è la chiave di cifratura).
- **Decifratura:** $D_{k2}(c) = m$ (dove $k2$ è la chiave di decifratura).

La funzione di decifratura D_{k2} deve essere l'inversa di E_{k1} .

7.1 Classi di Algoritmi Crittografici

7.1.1 Block Cipher (Cifrario a Blocchi)

- L'algoritmo opera su blocchi di dati di dimensione fissa b (es. 64 o 128 bit).
- La funzione E_{k1} è **deterministica**: lo stesso blocco di plaintext m_i , se cifrato con la stessa chiave $k1$, produce **sempre** lo stesso blocco di ciphertext c_i .
- **Problema:** Se un blocco di plaintext si ripete nella comunicazione ($m_i = m_j$), anche il blocco di ciphertext si ripeterà ($c_i = c_j$). Questo fa trapelare informazioni (pattern) all'attaccante.

7.1.2 Stream Cipher (Cifrario a Flusso)

- Opera su unità di dati molto piccole (es. 1-8 bit).
- Mantiene uno **stato interno** ($Stato_i$) che viene aggiornato continuamente.
- La funzione di cifratura dipende anche da questo stato: $E_{k1}(m_i, Stato_i) = c_i$.
- **Vantaggio:** Anche se $m_i = m_j$, lo stato interno sarà diverso, quindi $c_i \neq c_j$. Questo nasconde i pattern statistici del plaintext.
- Sono generalmente più veloci e richiedono meno memoria dei block cipher.

8 Crittanalisi e Sicurezza

8.1 Ipotesi di Kerckhoffs

Questo è un principio fondamentale della crittografia moderna: **la sicurezza di un sistema crittografico deve risiedere esclusivamente nella segretezza della chiave (k), non nella segretezza dell'algoritmo (E, D)**. Si deve sempre assumere che l'attaccante (Trudy) conosca perfettamente l'algoritmo che stiamo usando.

8.2 Penetrabilità e Tipi di Attacco

Un algoritmo è **penetrabile** (breakable) se un crittanalista può recuperare m da c (o peggio, k) in un tempo ragionevole. L'obiettivo dei progettisti è fare in modo che l'attacco più veloce sia la **ricerca esaustiva** (brute-force) di tutte le possibili chiavi.

Gli attacchi di crittanalisi si classificano in base all'informazione disponibile all'attaccante:

Ciphertext-only L'attaccante possiede solo messaggi cifrati (c). È l'attacco più difficile.

Known plaintext L'attaccante possiede alcune coppie (m, c) di messaggi in chiaro e cifrati corrispondenti. L'obiettivo è trovare k .

Chosen plaintext L'attaccante può scegliere un m arbitrario e ottenere dall'oracolo (il sistema) il c corrispondente. Un buon algoritmo deve resistere anche a questo scenario.

8.3 Segretezza Perfetta (Shannon)

Un algoritmo offre **segretezza perfetta** se il ciphertext c e il plaintext m sono **statisticamente indipendenti**.

- In pratica: osservare c non dà **alcuna informazione** in più su m .
- **Condizione di Shannon:** La chiave deve essere lunga almeno quanto il messaggio ($H(k) \geq H(m)$).

8.3.1 One-Time-Pad (OTP)

È l'unico algoritmo noto a garantire segretezza perfetta.

- **Algoritmo:** Cifratura $c = m \oplus k$ e Decifratura $m = c \oplus k$.
- **Ipotesi fondamentali:** La chiave k deve essere:
 1. Lunga **esattamente** quanto il messaggio m .
 2. Generata in modo **perfettamente casuale**.
 3. Utilizzata **una sola volta** (da cui il nome "one-time").
- **Problema:** La generazione e la distribuzione sicura di una chiave così lunga (e usarla una sola volta) è estremamente difficile nella pratica.

9 Tipologie Base di Algoritmi Crittografici

Gli algoritmi crittografici si dividono in tre famiglie principali:

Simmetrici (a chiave privata) Si usa una singola chiave segreta condivisa: $k = k_1 = k_2$.

Asimmetrici (a chiave pubblica) Si usa una coppia di chiavi (una pubblica e una privata): $k_1 \neq k_2$.

Algoritmi di Hash Non usano chiavi per la cifratura, ma producono un "fingerprint" dei dati.

9.1 Crittografia Simmetrica: Block Cipher

Lo schema generico della crittografia simmetrica (Symmetric block cipher) prevede che due entità (es. Bob e Alice) condividano la stessa chiave k . Questa chiave deve essere scambiata tramite un canale di comunicazione sicuro.

- Il mittente (Bob) usa un algoritmo di cifratura E per trasformare un blocco di testo in chiaro m (plaintext) in un blocco cifrato c (ciphertext). L'operazione è $c = E_k(m)$.
- Il blocco cifrato c viene inviato sul canale di comunicazione insicuro.
- Il ricevente (Alice) usa la stessa chiave k e un algoritmo di decifratura D per riottenere il blocco in chiaro. L'operazione è $m = D_k(c)$.
- L'algoritmo di decifratura è l'inverso di quello di cifratura: $D_k = E_k^{-1}$.
- m e c sono stringhe di bit di una lunghezza fissa b , mentre k è una stringa di bit di lunghezza l .
- Il block cipher E_k definisce una relazione biunivoca (permutazione) tra i 2^b possibili valori di m e i 2^b possibili valori di c , dipendente dalla chiave k .
- Se il messaggio è più lungo di b bit, viene spezzato in una serie di blocchi m_i .

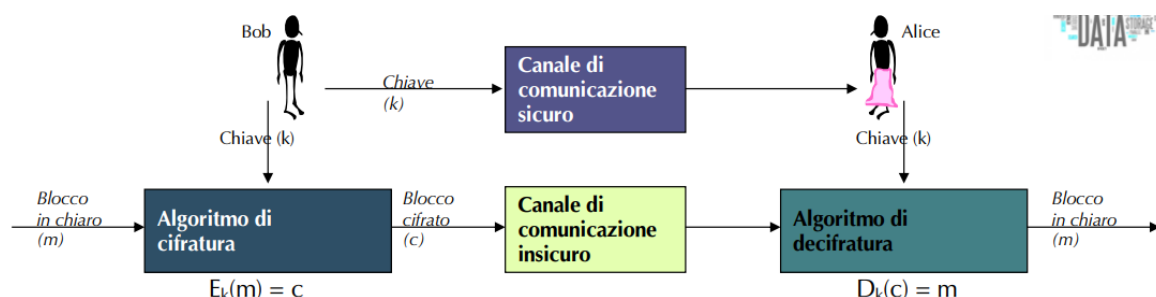


Figura 1: block cypher generico

9.2 Obiettivi e Tipologie dei Block Cipher

Gli obiettivi fondamentali di un block cipher robusto sono:

- **Dipendenza totale:** Ogni bit del blocco cifrato c deve dipendere da tutti i bit del messaggio m e da tutti i bit della chiave k .
- **Occultamento statistico:** Non ci deve essere alcuna relazione statistica evidente tra c e m per chi non conosce k .
- **Effetto valanga (Avalanche effect):** La modifica di un singolo bit in m (o in k) deve portare a una modifica di ciascun bit di c con probabilità 0.5.

Esistono due classi principali di block cipher:

1. **Cifrario a Sostituzione:** Definisce una corrispondenza biunivoca (mappatura) tra l'insieme M dei blocchi in chiaro e se stesso.
 - La chiave k è un indice che seleziona una delle $(2^b!)$ permutazioni possibili.
 - La lunghezza della chiave necessaria, $l \approx b \cdot 2^b$, rende questo approccio impraticabile (es. per $b = 64$, $l \approx 2^{70}$ bit).
 - Se m_i si ripete, anche c_i si ripete, rendendolo vulnerabile all'analisi statistica.
2. **Cifrario a Trasposizione:** Effettua una permutazione (uno "shuffle") dei bit all'interno del blocco m_i .
 - Lo spazio delle chiavi è molto più ridotto: $|K| = b!$. Per $b = 64$, $l \approx 384$ bit.
 - Offre una segretezza ancora minore rispetto alla sostituzione, ma è un blocco costruttivo utile.

9.3 Prodotto di Cifrari: Confusione e Diffusione

Per costruire una funzione E_k complessa e sicura, si usano sostituzione e trasposizione come blocchi fondamentali in cascata (prodotto di cifrari).

- Questo approccio è detto **confusione e diffusione**.
- **Confusione (Sostituzione):** Rende complessa la relazione tra c , m e k .
- **Diffusione (Trasposizione):** Spalma l'informazione di un bit di m su molti bit di c , per ottenere l'effetto valanga.
- Un prototipo di block cipher moderno applica ripetutamente (t volte, detti *round*) uno strato di sostituzione (es. n S-box parallele) e uno strato di trasposizione (P-box).

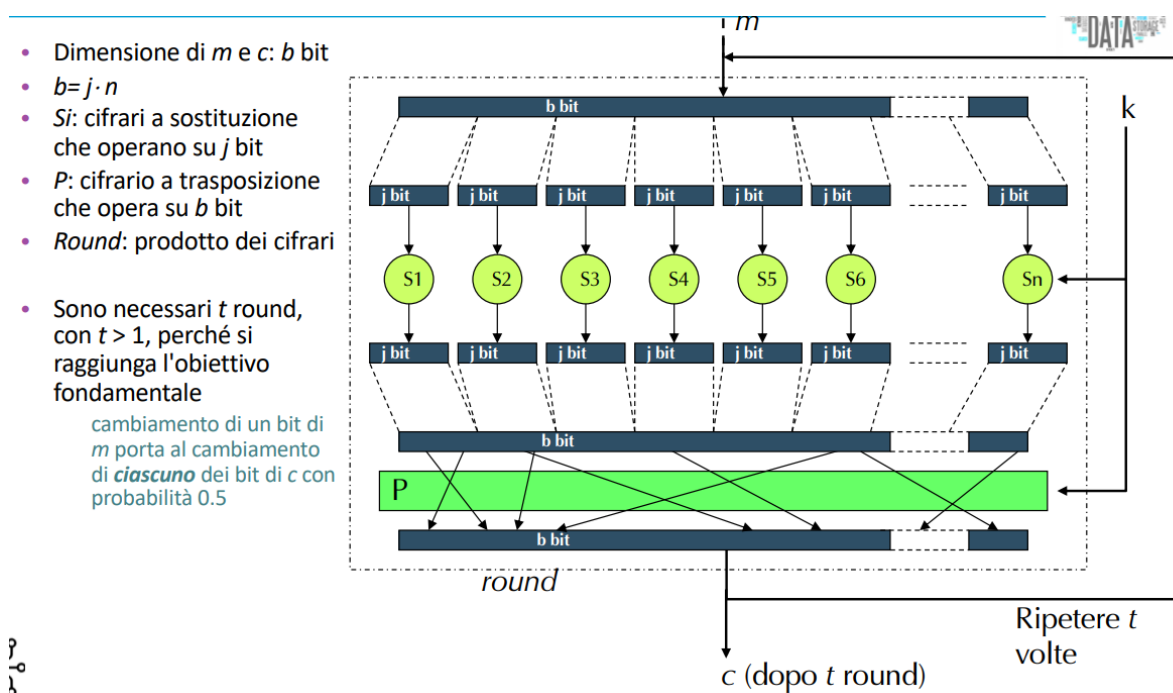


Figura 2: Didascalia

9.4 Data Encryption Standard (DES)

- È un cifrario simmetrico a blocchi sviluppato da IBM su commissione della NSA e standardizzato nel 1977.
- Opera su blocchi di $b = 64$ bit.
- Utilizza una chiave di $l = 56$ bit (sebbene la chiave fornita sia di 64 bit, 8 bit sono di parità e vengono scartati).
- È basato su un **cifrario di tipo Feistel**, che divide il blocco in due metà (Sinistra L e Destra R) e applica 16 round.

- Permutazioni indipendenti dalla chiave
- «Inner function»

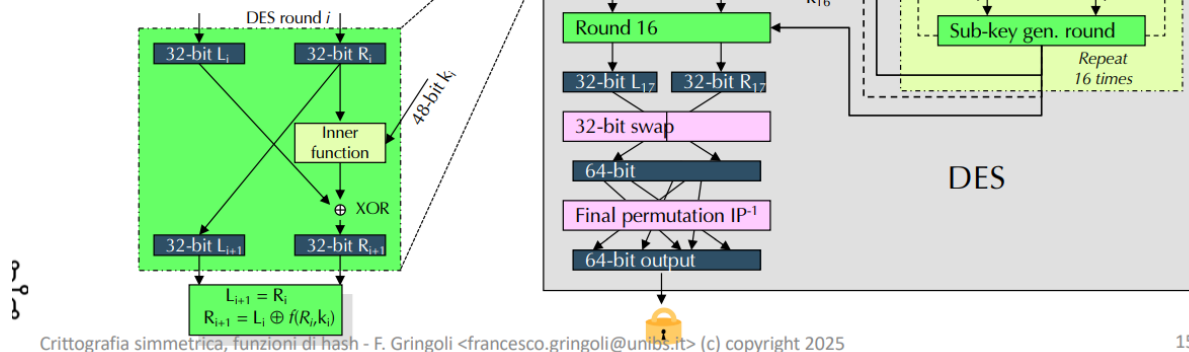
Look-up table, unica parte non lineare

- Prodotto di cifrari:

confusione e diffusione tramite round!

ogni round usa sottochiave k_i

- sottochiavi generate dalla chiave principale
- cifrario di tipo *Feistel* (due metà)
 - sinistra (L) - destra (R)



Crittografia simmetrica, funzioni di hash - F. Gringoli <francesco.gringoli@unibs.it> (c) copyright 2025

15

Figura 3: DES schema ad alto livello

- Lo stesso algoritmo (circuito) si usa per cifrare e decifrare, cambiando solo l'ordine di applicazione delle sottochiavi.

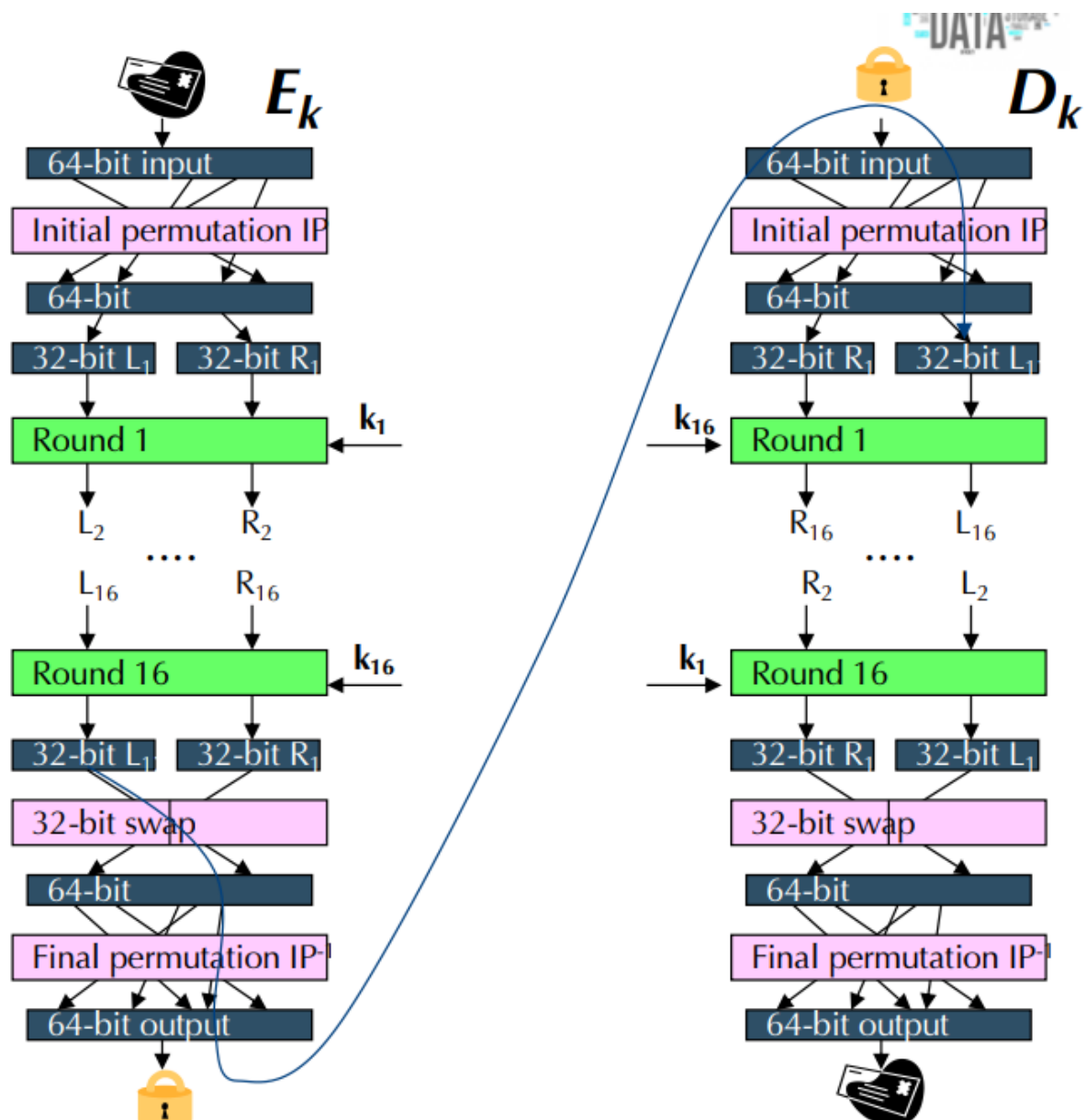


Figura 4: DES: Encryption vs Decryption

- **Analisi:** La chiave a 56 bit (2^{56} iterazioni) è oggi considerata insicura.
 - Già nel 1977 un attacco a forza bruta costava 20M\$ e richiedeva 12 ore.
 - Nel 1993, 3.5 ore per 1M\$.
 - Nel 2007 (COPACOBANA), meno di una settimana con 8K\$.
 - Nel 2022, con 13 GPU (6000\$), si stima un tempo simile.
- **3DES (Triple DES):** Per sopperire alla chiave corta, oggi si usa il 3DES.
 - Applica DES tre volte con due chiavi (k_1, k_2): $c = E_{k_1}(D_{k_2}(E_{k_1}(m)))$.
 - La chiave effettiva diventa di $l = 112$ bit (2^{112} spazio delle chiavi).
 - È retrocompatibile con DES (basta porre $k_1 = k_2$).

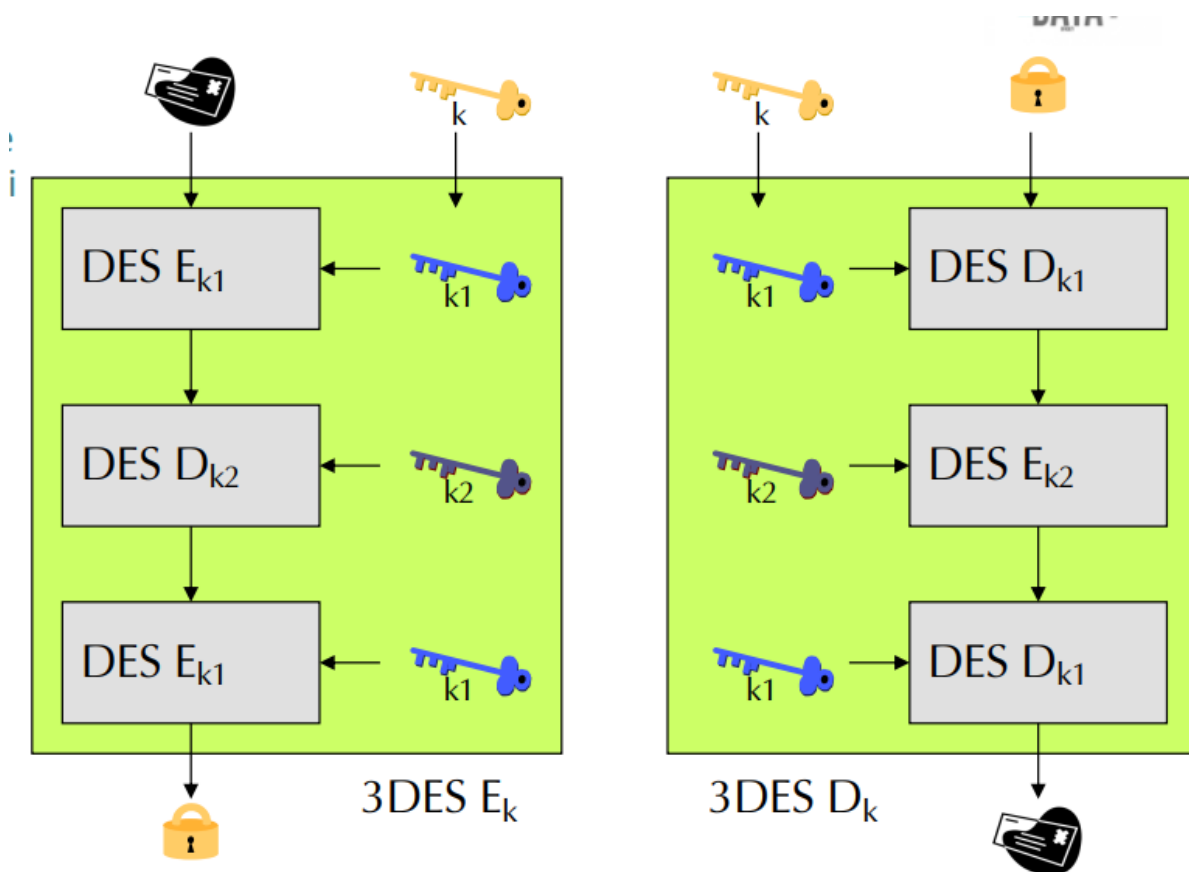


Figura 5: Il DES di oggi: 3DES

9.5 Advanced Encryption Standard (AES)

- È il successore di DES, standardizzato nel 2001. Si basa sull'algoritmo **Rijndael**.
- Dimensione del blocco $b = 128$ bit.
- Lunghezza chiave l variabile: 128, 192 o 256 bit (AES-128, AES-192, AES-256).
- A differenza di DES, non è una rete di Feistel. Opera su "stati", ovvero matrici di 4×4 byte (per AES-128).
- Applica un numero di round che dipende dalla lunghezza della chiave (es. 9 round principali per AES-128).
- Ogni round è composto da 4 operazioni:
 1. **SubBytes:** Sostituzione non lineare di ogni byte usando una S-box.
 2. **ShiftRows:** Rotazione (shift) delle righe dello stato.
 3. **MixColumns:** Moltiplicazione di ogni colonna per una matrice (diffusione). (Assente nel round finale).
 4. **AddRoundKey:** XOR tra lo stato e la sottochiave del round.

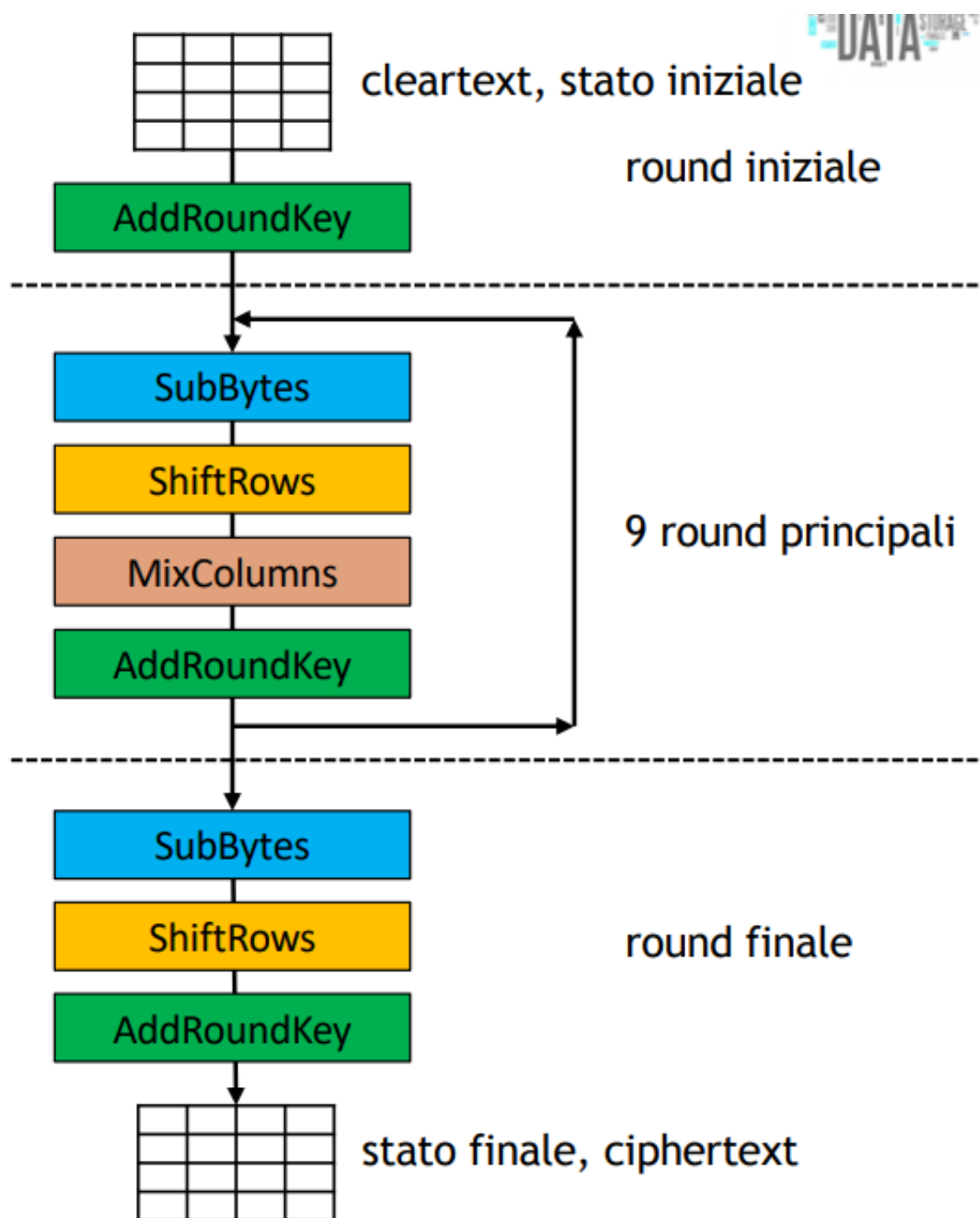


Figura 6: AES: operazioni e round

- **Sicurezza:** Non si conoscono attacchi crittanalitici pratici più efficienti della forza bruta. La NSA lo approva per dati *SECRET* (128 bit) e *TOP-SECRET* (256 bit).
- È molto più veloce di DES/3DES, specialmente in SW.

9.6 International Data Encryption Algorithm (IDEA)

- È un algoritmo a blocchi introdotto nel 1991, coperto da brevetto.

- Ha una dimensione di blocco $b = 64$ bit.
- Utilizza una chiave di lunghezza $l = 128$ bit, ritenuta sufficiente per proteggere da ricerche esaustive.
- È stato progettato per resistere a tecniche crittanalitiche avanzate, come la differential cryptanalysis.
- È progettato per essere efficiente sia in HW (grazie a strutture regolari) sia in SW (utilizzando operazioni orientate a multipli di ottetti, 16 bit, e funzioni base dei processori).
- Ad oggi, non sono stati pubblicati metodi pratici (oltre alla forza bruta) per violare IDEA.
- Viene utilizzato in diversi sistemi commerciali, tra cui PGP e S/MIME.

9.7 Blowfish (BF)

- È un algoritmo a blocchi introdotto nel 1994 ed è royalty-free.
- Ha una dimensione di blocco $b = 64$ bit.
- Si distingue per una lunghezza della chiave l variabile, che può andare da 32 a 448 bit.
- È stato uno dei primi algoritmi a introdurre il concetto di sicurezza variabile, legata alla dimensione della chiave scelta.
- È stato progettato per essere estremamente efficiente in implementazioni software (SW) su processori general-purpose.
- Risulta, invece, molto meno efficace in implementazioni hardware (HW), a causa di un algoritmo di generazione delle sotto-chiavi complesso che richiede "molta" memoria.
- Sebbene abbia ricevuto meno attenzione (in termini di tempo) di IDEA, finora non sono stati trovati problemi seri.

9.8 Stream Cipher

- **Block Cipher (BC):** Processano blocchi "larghi" ($b \geq 64$ bit) e sono *stateless* (la stessa E_k è usata per tutti i blocchi).
- **Stream Cipher (SC):** Processano blocchi "piccoli" ($1 \leq b \leq 8$ bit) e sono *stateful* (la funzione di cifratura varia man mano).

- **One-Time-Pad:** È lo stream cipher perfetto, ma richiede una chiave (keystream) lunga quanto il messaggio e perfettamente casuale.
- Gli SC pratici generano un *keystream pseudocasuale* z_i a partire da una chiave corta k e cifrano tramite XOR: $c_i = m_i \oplus z_i$.
- **Synchronous SC:** Il keystream z_i è generato indipendentemente da m_i e c_i . Sorgente e destinazione devono essere perfettamente sincronizzate. Se si perde un blocco, la decifrazione dei successivi è errata. (Es. RC4, Chacha-20, vedi pagine 32-33 slide symm).

9.8.1 Stream Cipher: Synchronous

- Gli stream cipher processano il plaintext in blocchi piccoli ($1 \leq b \leq 8$ bit) e sono *stateful* (con memoria).
- L'idea è generare un **keystream pseudocasuale** (z_i) a partire da una chiave corta K e cifrare tramite XOR.
- In un cifrario *synchronous*, il keystream è generato indipendentemente dal plaintext e dal ciphertext.
- Il generatore di keystream mantiene uno **stato** interno s_i .
- **Cifratura:** $c_i = m_i \oplus z_i$.
- **Decifratura:** $m_i = c_i \oplus z_i$.
- Il keystream z_i e lo stato successivo s_{i+1} sono generati da due funzioni g e f che dipendono solo dallo stato corrente s_i e dalla chiave K .
- **Proprietà:**
 - * Mittente e destinatario devono essere perfettamente sincronizzati, partendo dallo stesso stato iniziale.
 - * La perdita o l'aggiunta di un singolo blocco durante la trasmissione rompe la sincronizzazione, rendendo errata la decifrazione di tutti i blocchi successivi.
 - * Non c'è propagazione dell'errore per i *bit error*: un errore su un bit di c_i corrompe solo il bit corrispondente in m_i , senza influenzare i blocchi successivi.
- **Sicurezza:** Non offre segretezza perfetta (alla Shannon) perché l'entropia della chiave è molto minore di quella del messaggio ($H(k) \ll H(m)$). La sicurezza si basa sulla imprevedibilità del keystream pseudo-casuale.



Figura 7: Schema Synchronous Stream Cipher

- **Self-synchronizing SC:** Il keystream z_i dipende dalla chiave k e dai t blocchi di *ciphertext* precedenti: $z_i = g(c_{i-1}, \dots, c_{i-t}, K)$. Si auto-risincronizza dopo un errore: la perdita di un c_i corrompe solo un numero limitato di m_j successivi.

9.8.2 Stream Cipher: Self-Synchronizing

- È un cifrario a flusso in cui il keystream z_i dipende dalla chiave K e da un numero finito t di blocchi di **ciphertext** precedenti.
- **Cifratura:** $c_i = m_i \oplus z_i$, dove $z_i = g(c_{i-1}, \dots, c_{i-t}, K)$.
- **Decifratura:** $m_i = c_i \oplus z_i$, dove $z_i = g(c_{i-1}, \dots, c_{i-t}, K)$.
- **Initialization Vector (IV):** Per avviare il processo, è necessaria una serie di t blocchi v_i iniziali e non segreti, noti a mittente e destinatario.
- **Proprietà (Vantaggio):** È **auto-sincronizzante**. Se un blocco di c_i viene perso o aggiunto, la decifrazione sarà errata solo per un numero limitato di blocchi successivi (dipendente da t). Dopodiché, il sistema si riallinea automaticamente.
- **Proprietà (Svantaggio):** **Propagazione dell'errore**. Un singolo errore (bit flip) su un blocco c_i corrompe il plaintext m_i e anche i t blocchi successivi, portando alla corruzione di $t + 1$ blocchi in chiaro.

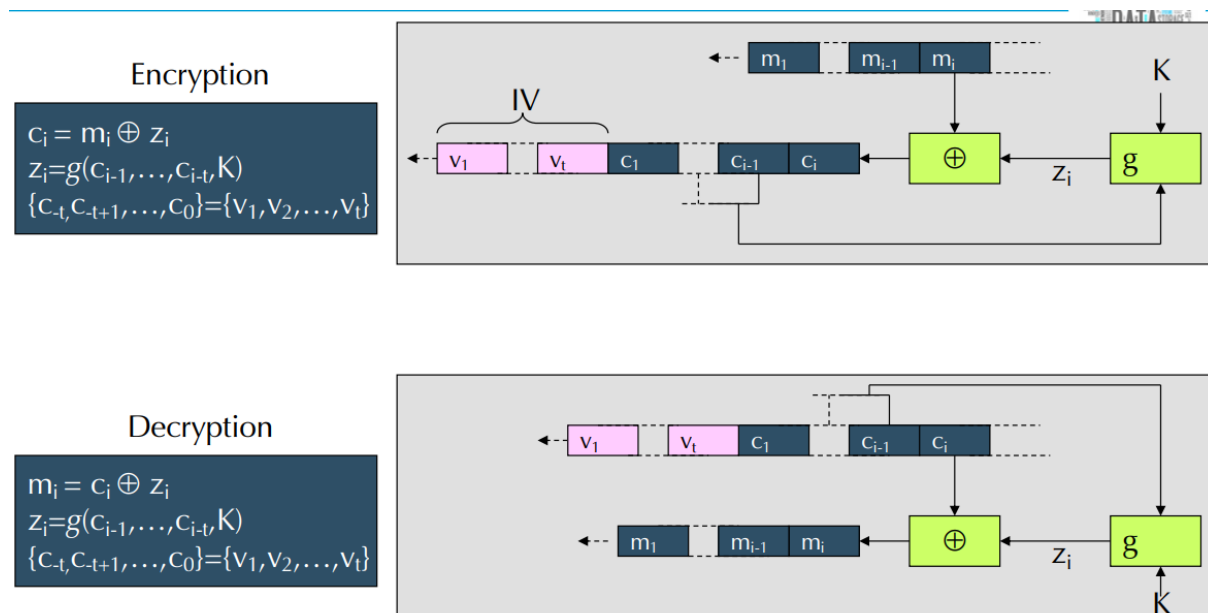


Figura 8: Schema Self-synchronizing Stream Cipher

9.9 Modi d'uso dei Block Cipher

Sono meccanismi che permettono di usare i block cipher (stateless) per cifrare messaggi più lunghi della dimensione del blocco b , trasformandoli di fatto in stream cipher.

- **Electronic Code Book (ECB):**

- È il modo più semplice: ogni blocco m_i è cifrato indipendentemente: $c_i = E_k(m_i)$.
- **Svantaggio:** Blocchi m_i identici producono blocchi c_i identici. Questo non nasconde le relazioni statistiche del plaintext ed è vulnerabile ad attacchi.
- **Non va mai usato.**

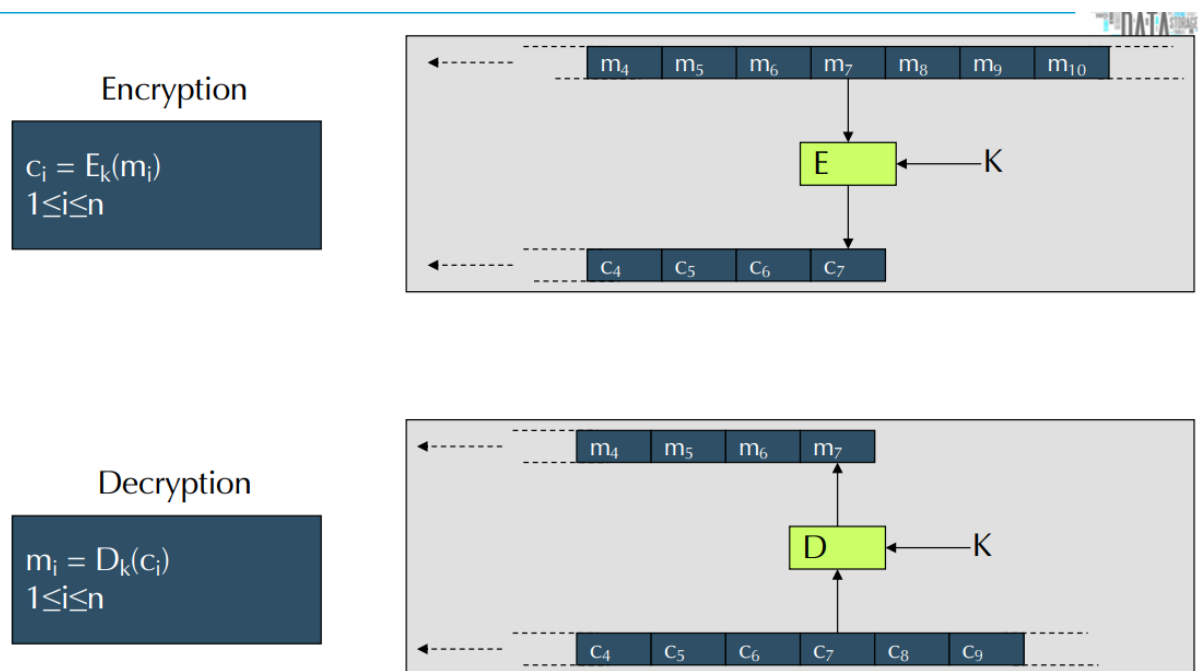


Figura 9: Electronic Code Book Mode

- **Cipher Block Chaining (CBC):**

- Introduce una dipendenza (chaining) tra i blocchi per nascondere le relazioni statistiche del plaintext.
- **Cifratura:** Come mostra l'immagine, il blocco di plaintext m_i viene prima combinato in XOR con il blocco di *ciphertext* precedente c_{i-1} . Il risultato di questo XOR viene poi cifrato con la chiave K .
- **Formula Cifratura:** $c_i = E_k(m_i \oplus c_{i-1})$.
- **Decifratura:** Il blocco c_i viene prima decifrato con la chiave K . Il risultato ($D_k(c_i)$) viene poi combinato in XOR con il blocco di *ciphertext* precedente c_{i-1} per riottenere il plaintext m_i .
- **Formula Decifratura:** $m_i = D_k(c_i) \oplus c_{i-1}$.
- **Initialization Vector (IV):** Poiché m_1 non ha un c_0 precedente, si usa un blocco iniziale IV (che può essere pubblico, ma deve essere casuale e imprevedibile) per avviare la catena: $c_0 = IV$.
- **Vantaggio:** Stessi blocchi m_i producono c_i diversi (a meno che anche i c_{i-1} siano identici, evento improbabile). Resiste molto bene all'analisi della frequenza.
- **Svantaggio (Propagazione dell'Errore):** Un errore (es. un bit flip) in un singolo blocco c_i corrompe la decifrazione di **due** blocchi:
 - * m_i viene completamente corrotto (perché $D_k(c_i)$ sarà incomprensibile).

- * m_{i+1} viene corrotto solo parzialmente, negli stessi bit in cui c_i era errato (perché $D_k(c_{i+1})$ è corretto, ma viene XORato con il c_i errato).
- * L'errore si ferma qui; m_{i+2} sarà decifrato correttamente (il sistema si auto-risincronizza).

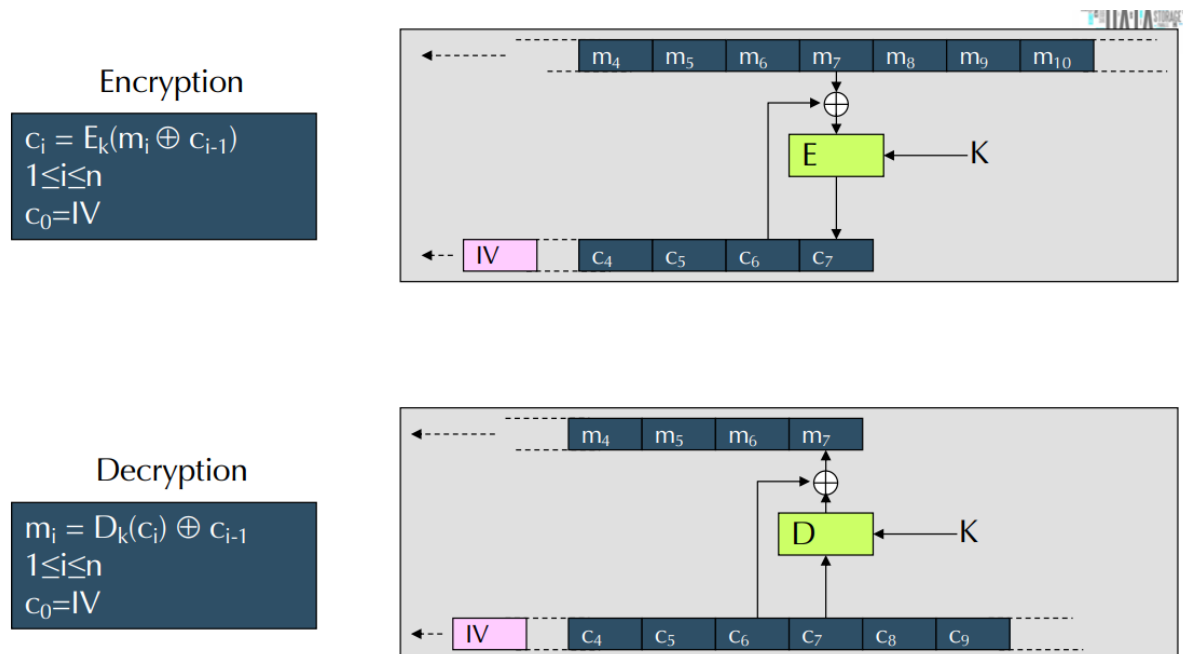


Figura 10: Cipher Block Chaining Mode

- **Counter Mode (CTR):**

- Trasforma un block cipher in uno stream cipher di tipo *synchronous*.
- **Generazione Keystream:** Genera un keystream o_i (detto *output block*) cifrando un **contatore** p_i .
- Il contatore p_i è tipicamente generato combinando un **Initialization Vector (IV)** (o Nonce) con un numero che si incrementa per ogni blocco (es. $p_i = IV + i - 1$).
- **Cifratura:** Il plaintext m_i è combinato in XOR con il keystream: $c_i = o_i \oplus m_i$.
- **Decifratura:** Avviene nello stesso modo. Si genera lo *stesso identico* keystream o_i (cifrando lo stesso contatore p_i) e lo si combina in XOR con il ciphertext c_i .
- **Formula Decifratura:** $m_i = o_i \oplus c_i$.
- **Nota Importante:** Come mostra il diagramma, sia la cifratura che la decifratura usano la funzione di **cifratura** E_k . La funzione di decifratura D_k non è necessaria.
- **Vantaggio (Random Access):** È possibile calcolare il keystream o_j e decifrare m_j (avendo c_j) in modo indipendente, senza dover calcolare i $j - 1$ blocchi

precedenti. Questo permette l'accesso e la decifrazione parallela (molto veloce) ed è ideale per la crittografia di dati ad accesso casuale (file, dischi, ecc.).

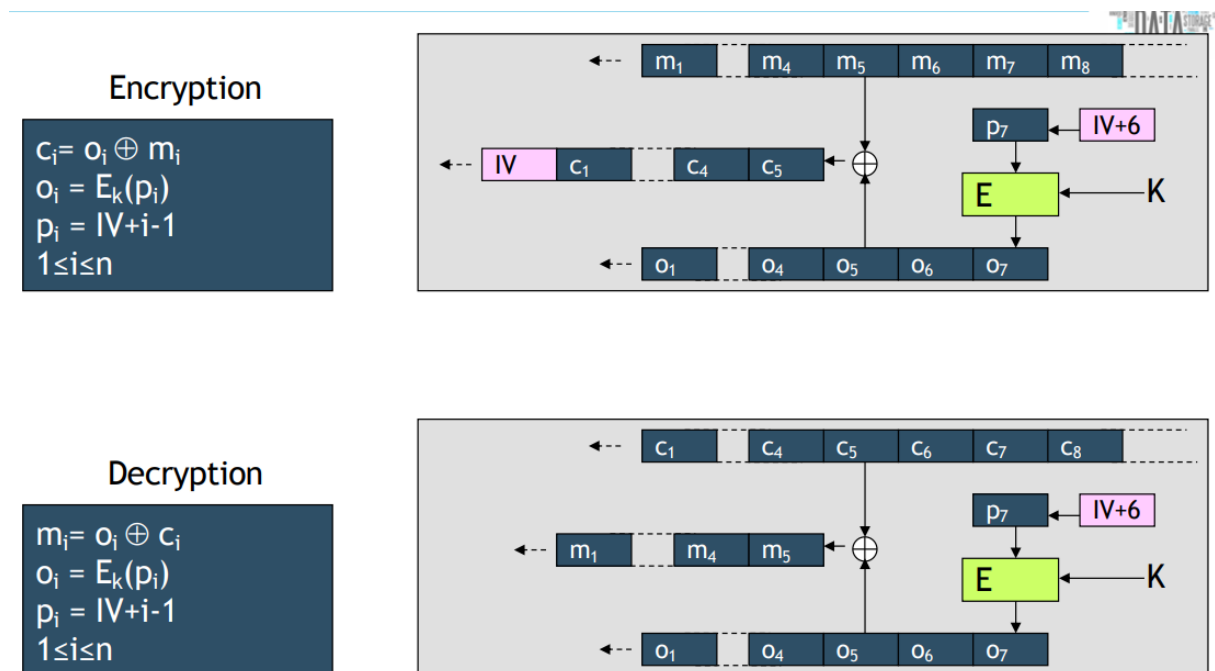


Figura 11: Schema modo Counter Mode

9.10 Definizioni e Tipologie

- Una **funzione di hash** (o *message digest function*) h è una funzione che mappa una stringa binaria m di lunghezza arbitraria in una stringa binaria d (l'hash) di lunghezza n fissa.
- L'hash $d = h(m)$ è una "impronta digitale" (fingerprint) o rappresentazione compatta di m .
- **MDC (Modification Detection Code):**
 - È una funzione di hash *senza chiave*: $d = h(m)$.
 - **Obiettivo:** Garantire l'integrità di m . Se l'hash d (ottenuto separatamente, es. da un sito web) è integro, si può verificare che m non sia stato modificato.
- **MAC (Message Authentication Code):**
 - È una funzione di hash *con chiave* (segreta): $d = h_k(m)$.
 - **Obiettivo:** Garantire sia l'integrità (nessuno ha modificato m) sia l'autenticazione (solo chi conosce k può aver generato d).

- Fase preparatoria

Si concatena il messaggio originale con un numero binario che indica la lunghezza del messaggio stesso

Padding, per ottenere, da una lunghezza originaria del messaggio arbitraria (j), un multiplo del blocco base q sul quale opera la funzione di compressione f

- La funzione di compressione f accetta come input un blocco di q bit (blocco del messaggio originale) più una variabile di stato di n bit, e produce un valore di n bit
- t iterazioni della funzione di compressione, con il risultato intermedio $r_i = f(x_i, r_{i-1})$, $r_0 = IV$
IV può essere fisso, funzione di m , o casuale (nel qual caso dev'essere trasmesso con m)
- La trasformata finale g è opzionale

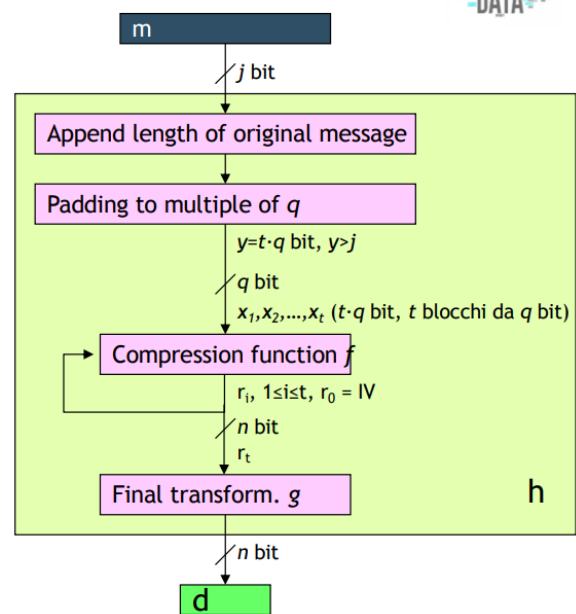


Figura 12: Modello MDC (funzione di hash iterativa)

9.11 Proprietà di Sicurezza (MDC)

Una funzione di hash crittografica (MDC) deve avere le seguenti proprietà:

- **Efficienza:** Deve essere semplice calcolare $d = h(m)$.
- **Preimage Resistance (Non invertibilità):** Dato d , deve essere computazionalmente difficile risalire a m tale che $h(m) = d$.
- **2nd Preimage Resistance (Weak Collision Resistance):** Dato x , deve essere computazionalmente difficile trovare un $y \neq x$ tale che $h(y) = h(x)$.
- **Strong Collision Resistance:** Deve essere computazionalmente difficile trovare *qualsiasi* coppia di messaggi distinti (x, y) tali che $h(x) = h(y)$.
- **Paradosso del Compleanno (Birthday Problem):** A causa del paradosso del compleanno, la probabilità di trovare una collisione (strong collision) è 0.5 dopo aver calcolato solo $O(2^{n/2})$ hash, dove n è la dimensione dell'hash.
- Un MDC è sicuro se trovare collisioni richiede un lavoro $O(2^{n/2})$ e la preimage resistance è garantita.

9.12 Algoritmi Notevoli (MDC e MAC)

- **MD5 (Message Digest 5):**
 - Introdotto nel 1991, produce un hash $n = 128$ bit.

- È molto veloce.
- **È considerato insicuro:** Dal 1996 è stata dimostrata una crescente insicurezza. Nel 2005 è stato dimostrato che bastano 8 ore per trovare due messaggi che collidono (violazione della *strong collision resistance*).
- **SHA-1 (Secure Hashing Algorithm 1):**
 - Introdotto nel 1993, produce un hash $n = 160$ bit.
 - Produce un digest più lungo di MD5 (160 vs 128) per resistere meglio al birthday attack.
 - **È considerato insicuro:** Nel 2005 è stata mostrata la sua insicurezza teorica (trovare collisioni richiede $O(2^{69})$ operazioni invece del $O(2^{80})$ ottimale).
- **HMAC (Hash-based MAC):**
 - È un metodo standard per costruire un MAC sicuro a partire da un qualsiasi MDC (es. HMAC-SHA1).
 - La formula è: $MAC_k(m) = MDC((k \oplus c_1) | MDC((k \oplus c_2) | m))$ (dove k è paddato e c_1, c_2 sono costanti).
 - **Vantaggio:** È un meccanismo la cui sicurezza è dimostrata. L'applicazione di HMAC "risolve" i problemi di bassa collision-resistance degli MDC sottostanti (come MD5). Per questo, **usate HMAC**.

10 Crittografia Asimmetrica

10.1 Schema e Concetti Fondamentali

La crittografia asimmetrica, o a chiave pubblica, si basa sull'uso di una coppia di chiavi (keypair) per ogni partecipante.

- **Chiave Pubblica (k_1 o K_{pub}):** Può essere distribuita liberamente, anche su canali insicuri. Viene usata per cifrare i messaggi destinati al proprietario della coppia di chiavi.
- **Chiave Privata (k_2 o K_{priv}):** Deve essere mantenuta segreta dal suo proprietario. Viene usata per decifrare i messaggi ricevuti.

Lo schema generico per la **confidenzialità** è il seguente:

1. Bob vuole inviare un messaggio m ad Alice.
2. Bob ottiene la chiave pubblica di Alice, K_{pubA} (es. tramite un canale insicuro).
3. Bob cifra il messaggio usando la chiave pubblica di Alice: $c = E_{K_{pubA}}(m)$.

4. Alice riceve il messaggio cifrato c .
5. Alice usa la sua chiave privata, K_{privA} , per decifrare il messaggio: $m = D_{K_{privA}}(c)$.

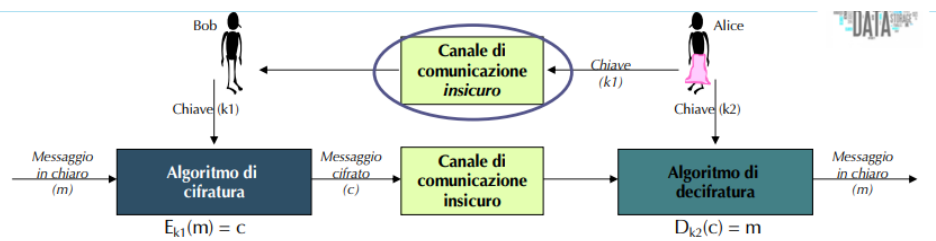


Figura 13: Schema generico crittografia asimmetrica

La proprietà fondamentale è che, data la chiave pubblica k_1 , deve essere computazionalmente impossibile (molto difficile) ricavare la corrispondente chiave privata k_2 . Mentre questo risolve il problema della distribuzione sicura della chiave di cifratura, introduce un nuovo problema: garantire l'**autenticità** della chiave pubblica k_1 .

10.2 Obiettivi e Basi Teoriche

I meccanismi asimmetrici servono a tre scopi principali:

1. **Confidenzialità:** Proteggere il contenuto del messaggio (come visto sopra).
2. **Firme Digitali:** Fornire autenticazione del mittente, integrità del messaggio e non ripudio.
3. **Instaurazione di Chiavi Effimere:** Scambiare in modo sicuro una chiave di sessione (es. simmetrica), come fa il protocollo Diffie-Hellman.

Questi meccanismi basano la loro sicurezza sulla difficoltà (ad oggi) nel risolvere alcuni problemi della teoria dei numeri. Si usano operazioni "unidirezionali":

- **Semplici da calcolare:** Es. $n = p \cdot q$ (moltiplicazione di due primi).
- **Difficili da invertire:** Es. trovare p e q dato n (fattorizzazione di interi).

La sicurezza non è provata in assoluto, ma si basa sul fatto che i problemi matematici sottostanti sono studiati da secoli e ancora non si conoscono algoritmi efficienti per risolverli (es. in tempo polinomiale).

10.3 Richiami: Teorema di Eulero

L'algoritmo RSA si basa su concetti di aritmetica modulare, in particolare sul Teorema di Eulero.

- Si definisce Z_n come l'insieme dei numeri modulo n (i resti della divisione per n).
- Si definisce Z_n^* come il sottoinsieme di Z_n che contiene i numeri primi relativi a n (cioè $\text{mcd}(x, n) = 1$).
- La **Funzione $\phi(n)$ di Eulero** conta quanti numeri ci sono in Z_n^* (la sua cardinalità: $\phi(n) = |Z_n^*|$).
- Se n è un numero primo, $\phi(n) = n - 1$.
- Se $n = p \cdot q$, con p e q primi distinti, allora $\phi(n) = (p - 1)(q - 1)$.
- **Corollario del Teorema di Eulero (base di RSA):** Dati p, q primi, $n = pq$, per ogni $m \in Z_n$ e $k \geq 0$ vale:

$$m^{k \cdot \phi(n) + 1} = m \pmod{n}$$

Questo corollario ci dice che se troviamo due numeri e e d tali che $e \cdot d = 1 \pmod{\phi(n)}$ (cioè $e \cdot d = k \cdot \phi(n) + 1$ per qualche k), allora l'operazione di "elevamento a e " è l'inversa dell'operazione di "elevamento a d ". Infatti: $(m^e)^d \pmod{n} = m^{ed} \pmod{n} = m^{k \cdot \phi(n) + 1} \pmod{n} = m \pmod{n}$.

11 Algoritmo RSA (Rivest, Shamir, Adleman)

RSA (1977) è il primo e più noto meccanismo asimmetrico "tuttofare", utilizzabile per tutti e tre gli obiettivi (confidenzialità, firme, scambio chiavi). Basa la sua sicurezza sulla difficoltà di fattorizzare n .

11.1 Generazione delle Chiavi

I parametri di RSA sono:

1. Parametri Privati (segreti):

- Si scelgono due numeri primi p e q molto grandi (es. 512+ bit).
- Si calcola $\phi(n) = (p - 1)(q - 1)$.
- Si sceglie e (esponente pubblico) e si calcola d (esponente privato) tale che $d = e^{-1} \pmod{\phi(n)}$ (cioè $e \cdot d = 1 \pmod{\phi(n)}$).

2. Parametri Pubblici (condivisibili):

- Si calcola il modulo $n = p \cdot q$.
- Si pubblica e (spesso un numero piccolo e fisso, come $e = 65537$).

La chiave pubblica è $K_{pub} = \{e, n\}$ e la chiave privata è $K_{priv} = \{d, n\}$.

11.2 Utilizzi di RSA

11.2.1 1. Confidenzialità

Come nello schema generale: Bob cifra m per Alice usando la K_{pub} di Alice.

- **Cifratura (Bob):** $c = m^e \bmod n$
- **Decifratura (Alice):** $m = c^d \bmod n$

L'operazione funziona per il teorema di Eulero, purché $m < n$.

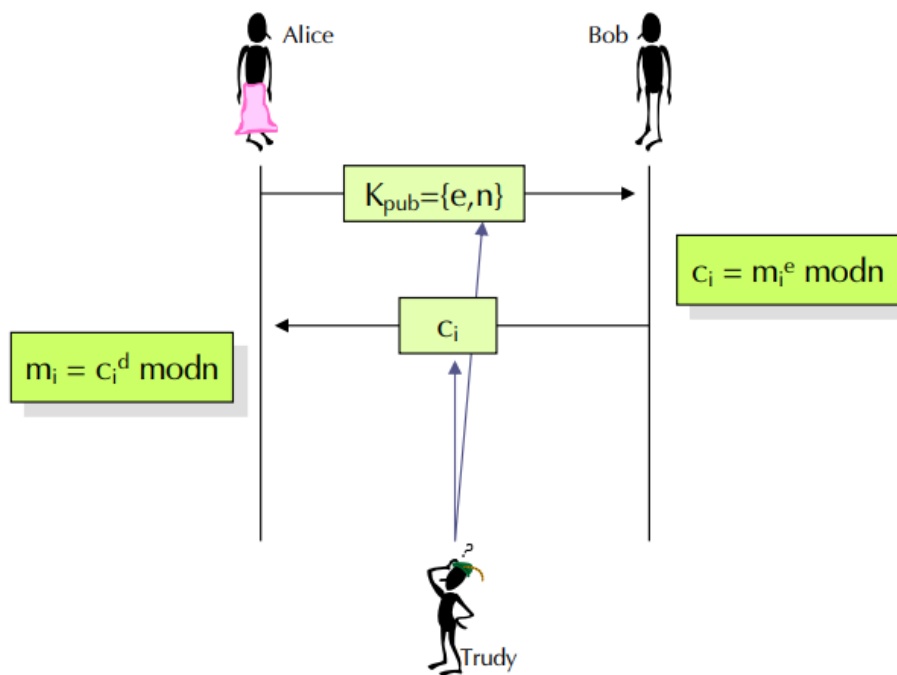


Figura 14: Schema RSA per Confidenzialità

11.2.2 2. Firme Digitali

Per firmare, Alice usa la sua **chiave privata** (per dimostrare di essere lei). Bob verifica la firma usando la **chiave pubblica** di Alice.

- **Firma (Alice):** $f = m^d \bmod n$ (dove m è il messaggio o, più realisticamente, l'hash del messaggio).
- Alice invia a Bob la coppia $\{m, f\}$.
- **Verifica (Bob):** Bob calcola $m' = f^e \bmod n$. Se $m' = m$, la firma è valida.

Questo garantisce non ripudio: solo Alice (che possiede d) può aver generato f .

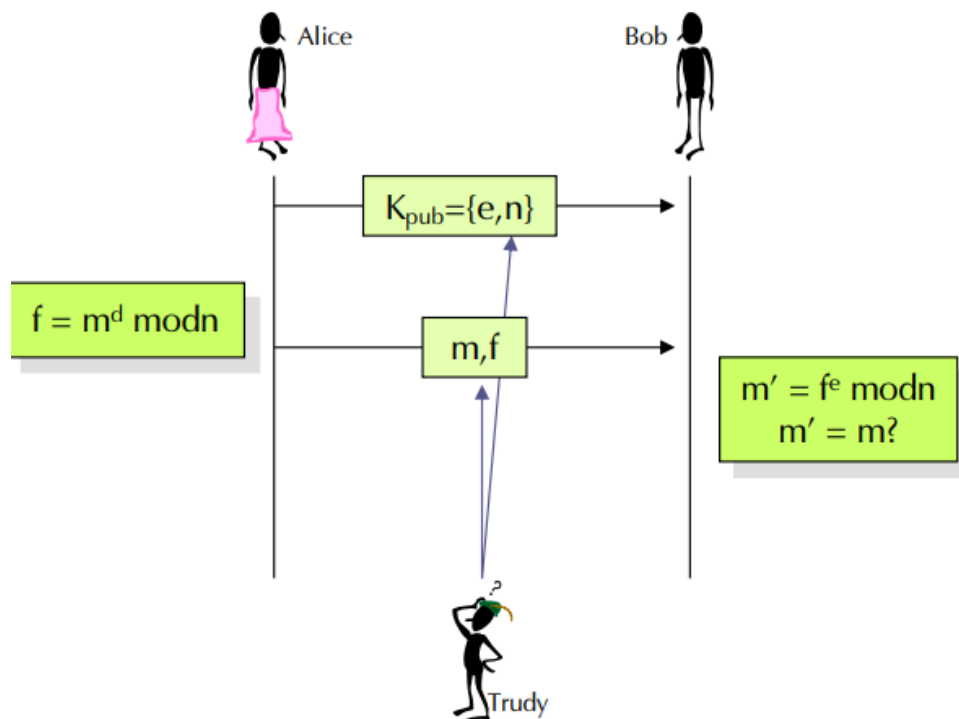


Figura 15: Schema RSA per Firme Digitali

11.2.3 3. Instaurazione di Chiavi Effimere

Poiché RSA è computazionalmente lento, non si usa per cifrare messaggi lunghi. Si usa invece un approccio ibrido:

1. Bob genera una chiave simmetrica r (es. per DES o AES), detta chiave effimera o di sessione.
2. Bob cifra r usando la K_{pub} di Alice (RSA): $c = r^e \bmod n$.
3. Bob invia c ad Alice.
4. Alice decifra c con la sua K_{priv} (RSA) e ottiene r : $r = c^d \bmod n$.
5. Ora Bob e Alice condividono la chiave simmetrica r e la usano per cifrare il resto della comunicazione (es. $c_i = DES_r(m_i)$), che è molto più efficiente.

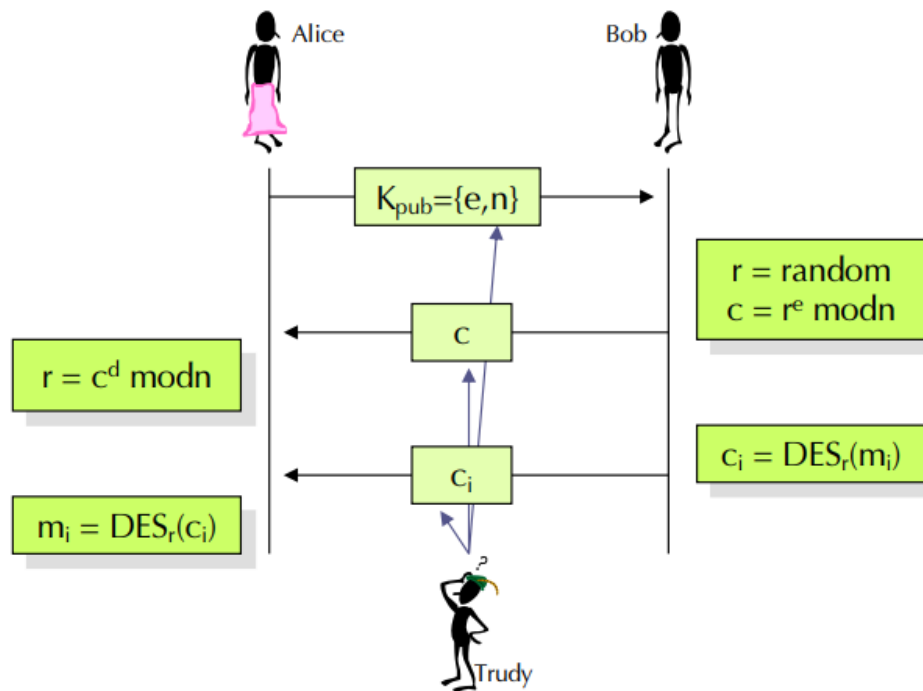


Figura 16: Schema RSA per Scambio Chiavi

11.3 Sicurezza e Uso Pratico di RSA

- **Padding:** Non si deve mai usare RSA "puro" (textbook RSA). Se m è piccolo (es. $m^e < n$), un attaccante può calcolare $m = \sqrt[e]{c}$. Per evitare questo e altri attacchi, si aggiunge un **padding** (riempimento, spesso casuale) al messaggio m prima di cifrarlo. Gli standard PKCS (Public Key Cryptography Standard) definiscono come applicare correttamente il padding.
- **Lunghezza Chiave:** La sicurezza dipende dalla grandezza di n . Oggi, chiavi n più corte di 1024 o 2048 bit non sono considerate sicure.
- **Efficienza:** La generazione delle chiavi è lenta. La cifratura (con e piccolo) è veloce, la decifratura (con d grande) è lenta.

12 Protocollo Diffie-Hellman (DH)

DH (1976) è stato il primo meccanismo asimmetrico pubblicato. Il suo unico obiettivo è lo **scambio sicuro di una chiave effimera** K . Basa la sua sicurezza sulla difficoltà del **problema del logaritmo discreto**: è semplice calcolare $b = a^i \bmod p$, ma è difficile calcolare i (il logaritmo) conoscendo a, b, p .

12.1 Meccanismo di Base

1. Alice e Bob si accordano pubblicamente su un numero primo p e un generatore g (di Z_p^*).
2. **Alice (privato):** Sceglie un numero segreto Sa (random, $1 \leq Sa < p$).
3. **Alice (pubblico):** Calcola $T_a = g^{Sa} \mod p$ e lo invia a Bob.
4. **Bob (privato):** Sceglie un numero segreto Sb (random, $1 \leq Sb < p$).
5. **Bob (pubblico):** Calcola $T_b = g^{Sb} \mod p$ e lo invia ad Alice.
6. **Calcolo Chiave (Alice):** Alice calcola $K = (T_b)^{Sa} \mod p = (g^{Sb})^{Sa} \mod p = g^{Sb \cdot Sa} \mod p$.
7. **Calcolo Chiave (Bob):** Bob calcola $K = (T_a)^{Sb} \mod p = (g^{Sa})^{Sb} \mod p = g^{Sa \cdot Sb} \mod p$.

Alice e Bob hanno ottenuto la stessa chiave segreta K senza mai trasmetterla. Un attaccante (Trudy) che osserva p, g, T_a, T_b non può calcolare K perché non conosce Sa o Sb , e non può ricavarli a causa della difficoltà del logaritmo discreto.

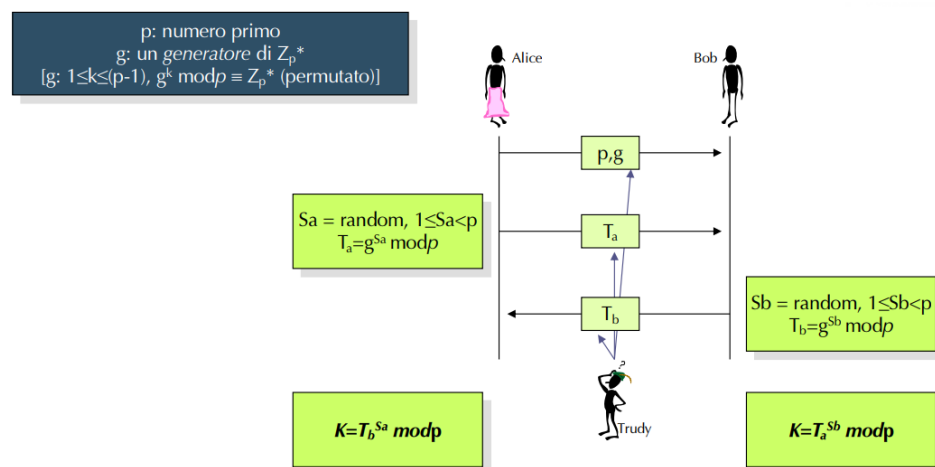


Figura 17: Schema protocollo Diffie Hellman

12.2 Sicurezza di DH

- **Attacchi Passivi:** DH è sicuro contro l'intercettazione (come visto sopra).
- **Attacchi Attivi (Man-in-the-Middle):** DH è vulnerabile all'attacco "Man-in-the-Middle" (MitM) perché lo scambio non è autenticato. Trudy può fingersi Bob con Alice e Alice con Bob, negoziando due chiavi separate e inoltrando i messaggi. Per prevenire ciò, DH viene quasi sempre usato insieme a meccanismi di autenticazione (es. firme digitali).

13 Elliptic Curve Cryptography (ECC)

ECC è un'evoluzione degli algoritmi asimmetrici. Si basa su problemi matematici (legati alle curve ellittiche) che sono considerati (ad oggi) ancora più difficili da risolvere rispetto alla fattorizzazione (RSA) o al logaritmo discreto (DH). Il vantaggio principale è che ECC può offrire lo **stesso grado di sicurezza di RSA con chiavi molto più piccole** (es. 1/10 della dimensione). Questo rende le operazioni molto più veloci ed efficienti, aspetto cruciale per server o dispositivi con poche risorse.

14 Public Key Infrastructure (PKI)

14.1 Il Problema dell'Autenticità

Come detto, la crittografia asimmetrica richiede che un utente (Bob) ottenga una copia autentica della chiave pubblica di un altro utente (Alice). Se non c'è garanzia di autenticità, Bob è vulnerabile a un attacco MitM.

La soluzione è introdurre una **Certification Authority (CA)**, ovvero una terza parte fidata. L'idea è: "Se Bob si fida della CA, e la CA garantisce (firma) che una certa chiave pubblica appartiene ad Alice, allora Bob può fidarsi di quella chiave pubblica".

14.2 Certificati Digitali (X.509)

Una PKI è l'infrastruttura (protocolli, policy, CA) che gestisce questo processo. Lo strumento operativo è il **certificato digitale** (lo standard più diffuso è X.509).

Un certificato X.509 è un documento digitale che lega un'identità a una chiave pubblica. I suoi campi principali sono:

- **Identità dell'Utente (Subject):** Chi possiede la chiave (es. CN=argo.ing.unibs.it).
- **Chiave Pubblica dell'Utente:** La chiave K_{pubA} che si sta certificando.
- **Identità della CA (Issuer):** Chi ha emesso e firmato il certificato (es. CN=Test CA).
- **Validità:** Un periodo (data di inizio e fine) durante il quale il certificato è valido.
- **Firma della CA:** L'hash di tutti i campi precedenti, cifrato con la chiave *privata* della CA (K_{privCA}).

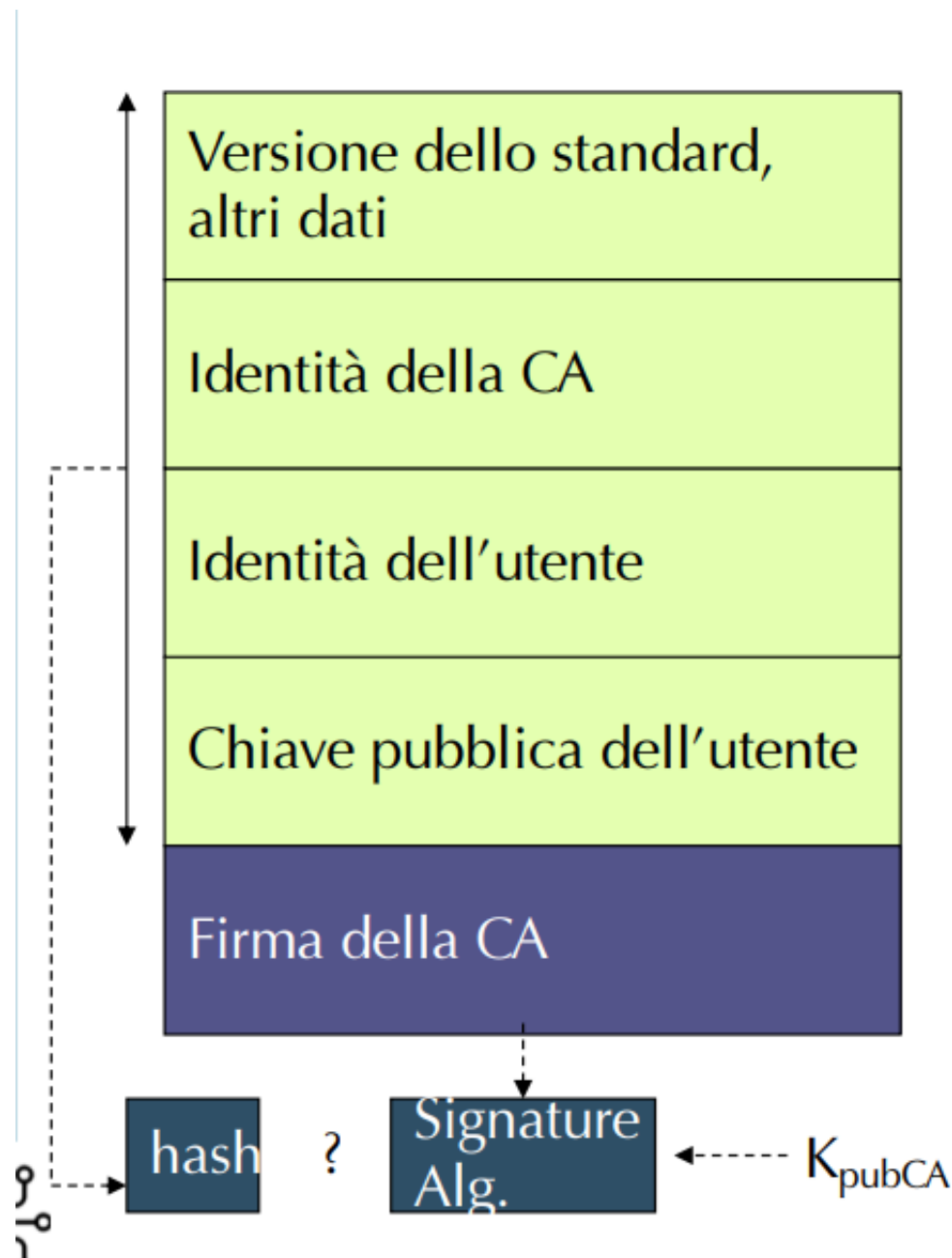


Figura 18: Schema campi certificato X.509

14.3 Processo di Verifica del Certificato

1. Alice invia a Bob il suo certificato, $CERT_A$ (firmato da CA_{xz12a}).
2. Bob riceve $CERT_A$. Per fidarsi, deve verificare la firma.
3. Bob estrae l'identità dell'Issuer (CA_{xz12a}).
4. Bob cerca CA_{xz12a} nella sua lista di **Root CA** attendibili (pre-installate nel suo OS o browser).
5. Se la trova, estrae la K_{pubCA} (che era nel certificato della CA stessa).

6. Bob usa K_{pubCA} per verificare la firma $S_{CA,xz12a}$ presente sul $CERT_A$ di Alice.
7. Se la verifica ha successo, Bob sa che $CERT_A$ è autentico e integro. Ora può fidarsi della K_{pubA} contenuta al suo interno per comunicare con Alice.

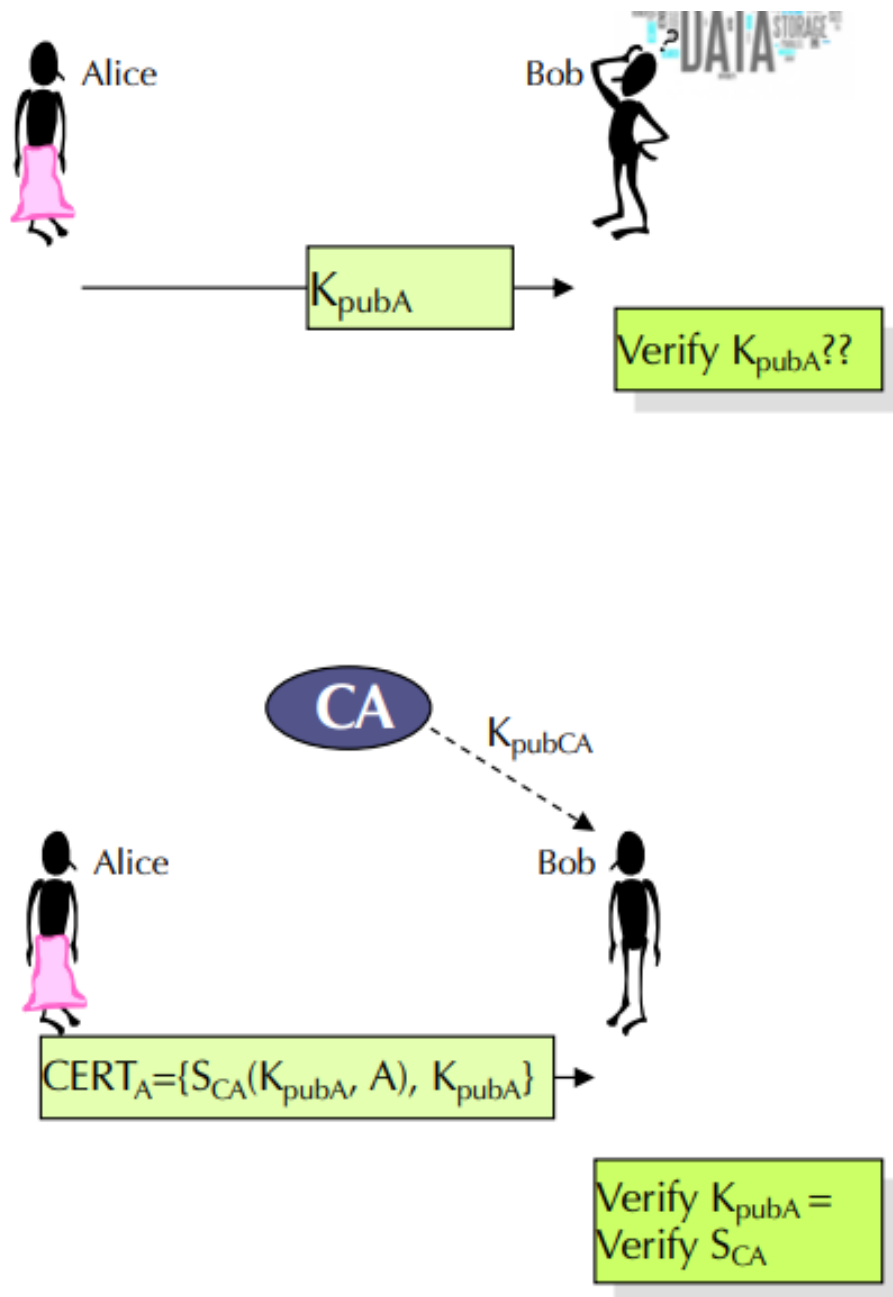


Figura 19: Schema verifica certificato

14.4 Catene di Certificati e Revoca

- **Root CA:** Le CA di cui ci si fida implicitamente (i loro certificati sono pre-installati) sono dette "Root CA". I loro certificati sono *self-signed* (Issuer == Subject), poiché non c'è un'autorità superiore che li firmi.

- **Intermediate CA:** Per ragioni di sicurezza, le Root CA non firmano direttamente i certificati degli utenti finali. Firmano i certificati di "CA Intermedie" (CA_1), che a loro volta firmano i certificati degli utenti (Alice).
- **Catena (Chain of Trust):** Alice invia a Bob sia il suo certificato (firmato da CA_1) sia il certificato di CA_1 (firmato dalla CA_{root}). Bob verifica prima CA_1 usando la CA_{root} , e poi verifica Alice usando CA_1 .
- **Revoca (CRL):** Se una chiave privata viene compromessa, il certificato corrispondente deve essere revocato prima della sua data di scadenza. Le CA pubblicano periodicamente una **Certificate Revocation List (CRL)**, una lista firmata dei numeri seriali dei certificati revocati. Prima di fidarsi di un certificato, Bob dovrebbe anche controllare che non sia presente nell'ultima CRL della CA.