University of Pisa

Department of Information Engineering

Foundation Of Cybersecurity

Nicolò Mariano Fragale

March 2025

Contents

Information

These notes are intended for educational purposes only and cover essential concepts in the field of data systems and security. The aim is to provide a comprehensive understanding of topics such as system vulnerabilities, protection techniques, and defense strategies in cybersecurity.

This document includes topics related to access control, authentication mechanisms, database security, cryptographic methods, and advanced persistent threats, with a particular focus on practical applications in real-world scenarios.

1 Symmetric cryptography

1.1 Cifratura Perfetta Un cifrario è perfetto se, dato il ciphertext c, è impossibile ottenere il relativo plaintext p.

$$\Pr[P = p] = \Pr[P = p \mid C = c] \Rightarrow \Pr[C = c \mid P = p] = \Pr[C = c]$$

Esempio: se due plaintext diversi cifrati producono lo stesso valore, l'attaccante non saprà distinguerli. Quindi c non deve rivelare alcuna informazione sul relativo p.

Teorema di Shannon - Perfect Secrecy Un cifrario è perfetto se rispetta il seguente teorema:

Perfect Secrecy (Shannon):

- **Unconditional Security**: *Un cifrario è perfetto se:*
  1. $|K| \geq |P|$ (cioè il numero di chiavi deve essere maggiore o uguale al numero di messaggi possibili)
  2. $\forall p, \forall c : \exists k$ tale che $E_k(p) = c$
- **Dimostrazione per assurdo**:
  - Supponiamo $|K| < |M|$ *(contraddizione)*
  - Sia $M(c) = \{m : m = D_k(c)$ per qualche $k \in K\}$
  - Allora $|K| < |M(c)|$ (contraddice il teorema)
  - Se $|K| = |M|$, allora $\Pr[M = m' \mid C = c] = \Pr[M = m']$

Nota bene: La chiave deve essere scelta *randomicamente*.

Diffusion e Confusion

- **Confusion**: tecnica di cifratura che oscura la relazione tra plaintext e ciphertext. È usata in AES, raggiungibile tramite sostituzioni.
- **Diffusion**: tecnica di cifratura che estende la proprietà di un plaintext su molti ciphertext, per nascondere proprietà statistiche. Esempio: permutazioni, usato da AES (MixColumns).

1.2 Random bit generator A Random Bit Generator (RBG) outputs a sequence of *statistically independent* and *unbiased* bits

- *Statistically independent* means that the probability of emitting a bit value (1 or 0) does not depend on the previous bits
- *Unbiased* means that the probability of emitting a bit value (1 or 0) is equal to 0.5

Classes of RBGs:

- **True Random Bit Generators (TRBGs)**: use a physical phenomenon to generate random bits
- **Pseudo Random Bit Generators (PRBGs)**: use a deterministic algorithm to generate random bits
- **Cryptographically Secure Pseudo Random Bit Generators (CSPRBGs)**: use a deterministic algorithm to generate random bits.

True Random Bit Generators (TRBGs)

- Based on a physical phenomenon
- The output can not be reproduced
- Both hw-based and sw-based

Pseudo Random Bit Generators (PRBGs)

- A Pseudo Random Bit Generator is a deterministic algorithm that, given a truly random binary sequence of lenght k (seed), outputs a binary sequence of length L (pseudorandom bit sequence), L ¿¿ k
- **N.b: The length k of the seed is sufficiently large so that it is "infeasible" to search over 2k possible output sequences (necessary condition)**

Cryptographically Secure Pseudo Random Bit Generators (CSPRBGs)

- Informally, a CSPRNG is an unpredictable PRNG;
- More formally, Given a sequence of bits si, si+1, . . . , si+n-1 (a prefix), there exist no polynomial time algorithm that can predict the next bit si+n with better than 50% chance of success

1.3 DES

- DES is a block cipher symmetric encryption.
- DES uses key legth of 64 bits, 56 bits are used for encryption, 8 bits are used for parity check.
- DES is a Feistel cipher, which means that it uses a round function to encrypt the data.
- DES uses 16 rounds of encryption, each round uses a different key generated from the initial key.

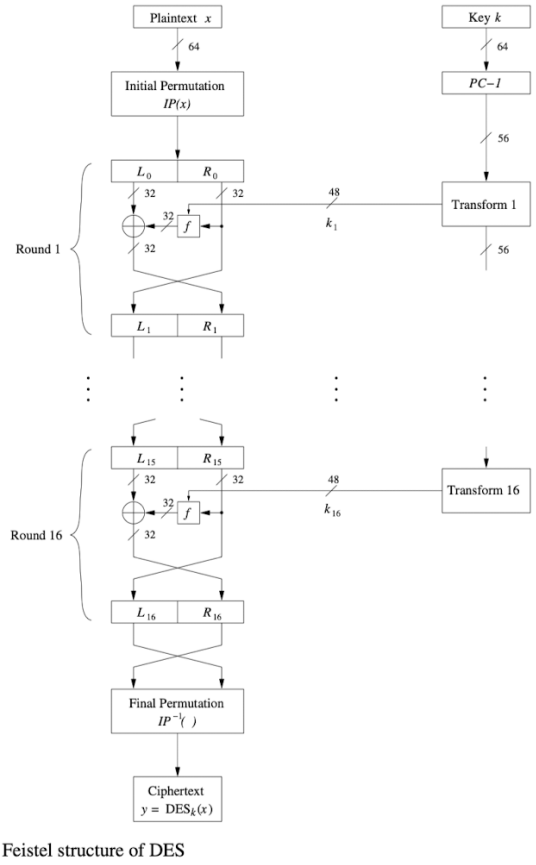


Figure 1: DES algorithm

The key schedule derives 16 round keys ki, each consisting of 48 bits, from the original 56-bit key. Another term for round key is subkey. First, note that the DES input key is often stated as 64-bit, where every eighth bit is used as an odd parity bit over the preceding seven bits. It is not quite clear why DES was specified that way. In any case, the eight parity bits are not actual key bits and do not increase the security. DES is a 56-bit cipher, not a 64-bit one.

N.B The key space is too small, i.e., the algorithm is vulnerable against brute-force attacks.

1.3.1 DES key whitening

Data:

- k : main key
- k1, k2: derived from k
- p : plaintext
- c : ciphertext

Algorithm:

1. p $\oplus$ k1
2. k (p $\oplus$ k1)
3. c = k2 $\oplus$ [k (p $\oplus$ k1)]

key whitening does not strengthen block ciphers against most analytical attacks such as linear and differential cryptanalysis.
key whitening is not a "cure" for inherently weak ciphers.
A variant of DES which uses key whitening is DESX.
most modern block ciphers such as AES already apply key whitening internally by adding a subkey prior to the first round and after the last round.

1.3.2 2DES

2DES (Double DES) significa eseguire l'algoritmo DES due volte, con due chiavi diverse.

$$C(P) = E_{K_2}(E_{K_1}(P))$$

Dove:

- P è il plaintext;
- K_1, K_2 sono due chiavi DES diverse;
- E è la funzione di cifratura DES standard.

Tuttavia, 2DES **non è sicuro** perché è vulnerabile al noto *Meet-in-the-Middle Attack*.

Poiché 2DES fallisce nel suo scopo di rafforzare la sicurezza rispetto a DES, si è passati all'uso di **3DES** (Triple DES), che adotta una struttura più sicura.

1.3.3 Triple DES (3DES)

Triple DES (3DES) nasce come estensione sicura del DES, per superare le debolezze di DES e 2DES. Utilizza la cifratura DES applicata tre volte, secondo uno schema **Encrypt–Decrypt–Encrypt (EDE)**.

Struttura EDE:

$$C = E_{K_1}(D_{K_2}(E_{K_1}(P)))$$

- P: plaintext
- C: ciphertext
- K_1, K_2: due chiavi DES (56 bit ciascuna)
- E: funzione di cifratura DES
- D: funzione di decifratura DES

Motivazioni dello schema EDE:

- Garantisce la retrocompatibilità: se $K_1 = K_2$, allora $3DES$ si comporta come un singolo DES.
- Impedisce il *Meet-in-the-Middle Attack*, che affligge 2DES.

Varianti di 3DES:

1. **3DES con due chiavi:** usa K_1 e K_2, quindi lunghezza chiave effettiva = 112 bit.
2. **3DES con tre chiavi:** usa K_1, K_2 e K_3, con lunghezza chiave effettiva = 168 bit.

3DES is very efficient in hardware but not particularly in software.
It is popular in financial applications as well as for protecting biometric information in electronic passports.
This surprisingly simple modification makes DES much more resistant against exhaustive key searches.

1.4 Meet In The Middle attack

2DES For double encryption, a plaintext x is first encrypted with a key kL, and the resulting ciphertext is encrypted again using a second key kR.

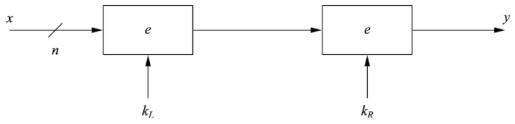


Figure 2: Double encryption

- Time complexity: O(2^k)
- Space complexity: O(k2^k)

In case we would achieve x from y then we have to perform (2^k x 2^k) $\rightarrow$ 2^{2k}.
To avoid this naive brute force attack the attacker perform Meet In The Middle attack.
This is a divide and-conquer attack in which Oscar first brute-force-attacks the encryption on the left-hand side, which requires 2^k cipher operations, and then the right encryption, which again requires 2^k operations. If he succeeds with this attack, the total complexity is $2^k + 2^k = 2\text{x}2^k = 2^{2k+1}$.

- Time complexity: O(2^{56}) **DOABLE! (computationally solvable)**
- Space complexity: O(k2^{56}) (lot of space)

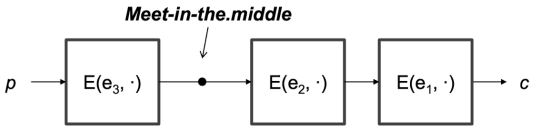


Figure 3: Triple encryption

3DES

- Time complexity: O(2^{112}) **UNdoable! (computationally solvable)**
- Space complexity: O(k2^{56}) (lot of space)

3DES is not safe in software implementation
Key lengths of at least 256-bit are necessary to resist quantum computing attacka.

1.5 AES L'Advanced Encryption Standard (AES) è un algoritmo di cifratura simmetrica a blocchi, standardizzato dal NIST, con blocchi fissi di 128 bit e chiavi di lunghezza variabile: 128, 192 o 256 bit. In questa sezione si analizza in dettaglio il funzionamento di AES-128.

1.5.1 Dati iniziali

AES lavora su:

- **Plaintext**: blocco di 128 bit (16 byte)
- **Chiave**: lunghezza di 128 bit (per AES-128), 192 bit (per AES-192) o 256 bit (per AES-256).
- **State**: rappresentazione interna dei dati come matrice 4×4 di byte
- **Chiavi di round**: Da una singola chiave iniziale vengono generate chiavi di round (una per ogni round + una iniziale)(AES-128: 11 chiavi da 128 bit).

1.5.2 Rappresentazione dello State

I 16 byte del plaintext vengono caricati nella matrice *State* in formato colonna, secondo l'ordine seguente:

$$\text{Plaintext} = [b_0, b_1, \ldots, b_{15}]$$

$$\text{State} = \begin{bmatrix} b_0 & b_4 & b_8 & b_{12} \\ b_1 & b_5 & b_9 & b_{13} \\ b_2 & b_6 & b_{10} & b_{14} \\ b_3 & b_7 & b_{11} & b_{15} \end{bmatrix}$$

Struttura generale di AES-128

AES-128 esegue 10 round (12 round per AES-192)(14 round per AES-256).
Ogni round, ad eccezione dell'ultimo, include le seguenti operazioni:

1. **SubBytes**
2. **ShiftRows**
3. **MixColumns**
4. **AddRoundKey**

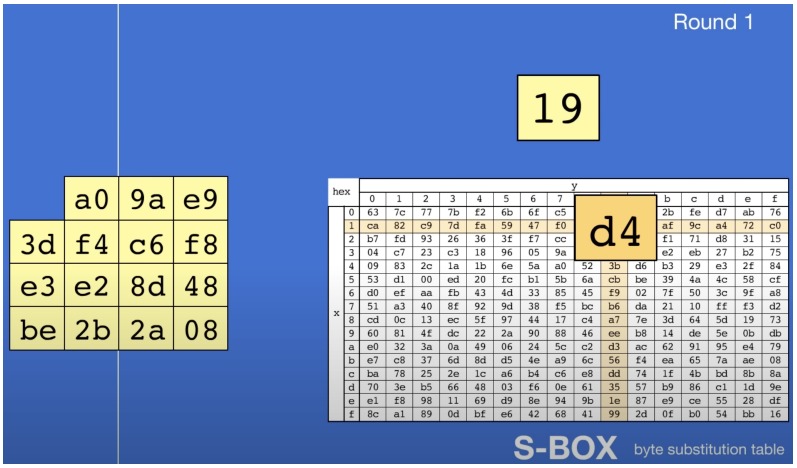
L'ultimo round omette l'operazione di *MixColumns*. Inoltre, vi è un **pre-round** iniziale che esegue solo *AddRoundKey*.

Schema dei Round AES-128

- **Round 0**: AddRoundKey
- **Round 1-9**: SubBytes → ShiftRows → MixColumns → AddRoundKey
- **Round 10**: SubBytes → ShiftRows → AddRoundKey

1.5.3 Operazioni AES

SubBytes Ogni byte dello State viene sostituito con un byte corrispondente in una S-box non lineare, predefinita.
Tale sostituzione è basata sull'inverso moltiplicativo in $GF(2^8)$ seguito da una trasformazione affine.
È una funzione non lineare e invertibile.

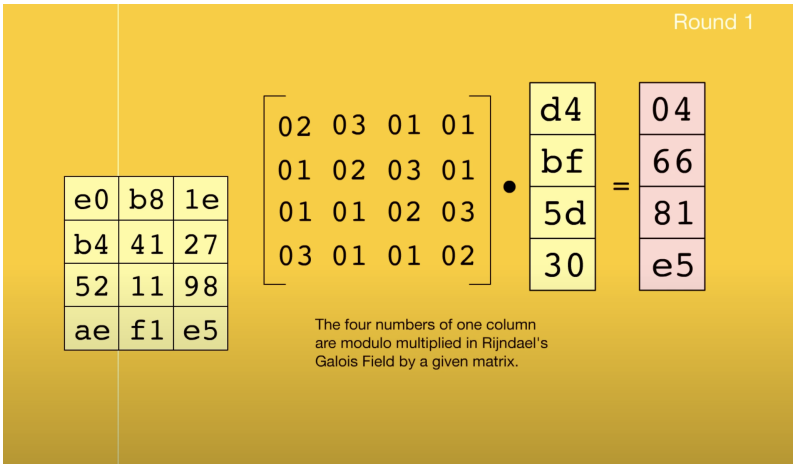


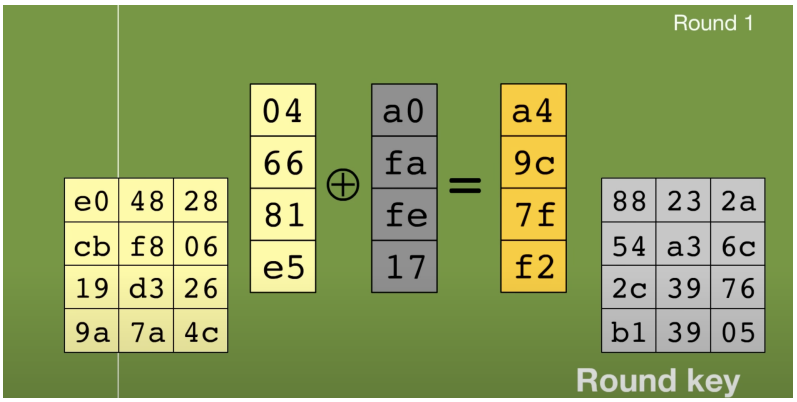
ShiftRows Operazione di permutazione sulle righe dello State:

- Riga 0: nessun shift
- Riga 1: shift di 1 byte a sinistra
- Riga 2: shift di 2 byte a sinistra
- Riga 3: shift di 3 byte a sinistra

MixColumns Operazione lineare su GF(2^8).
Ogni colonna dello state è considerata come un polinomio a 4 componenti.
Viene moltiplicata per una matrice fissa 4x4 in GF(2^8).
È una moltiplicazione polinomiale modulo (x^4) + 1 nel campo finito.

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \cdot \begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix} \quad \text{mod } x^4 + 1$$



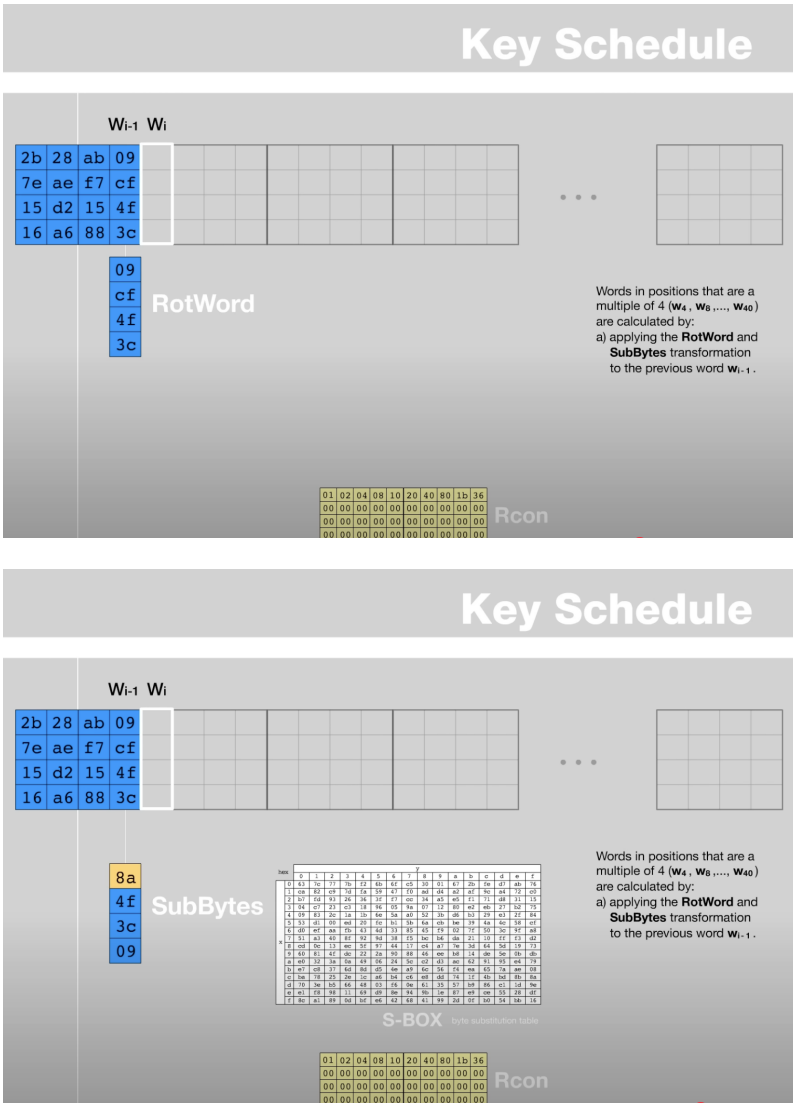


AddRoundKey Ogni byte dello State viene combinato con la Round Key corrispondente tramite XOR.

1.5.4 Key Schedule (Espansione della Chiave)

La chiave di 128 bit viene espansa in 11 parole (una per ogni round più una iniziale). Ogni parola è un vettore di 4 byte. Le operazioni principali sono:

- **RotWord**: rotazione ciclica di 1 byte a sinistra
- **SubWord**: applicazione della S-box ad ogni byte
- **XOR** con la round constant (Rcon)

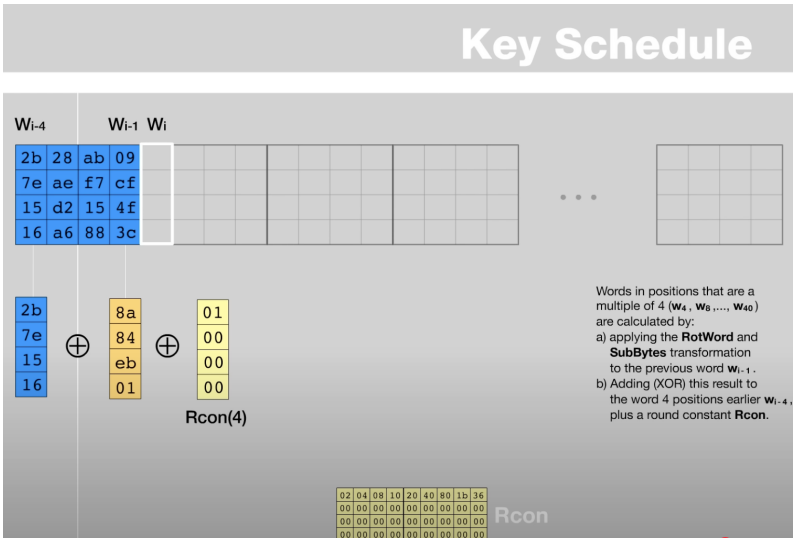


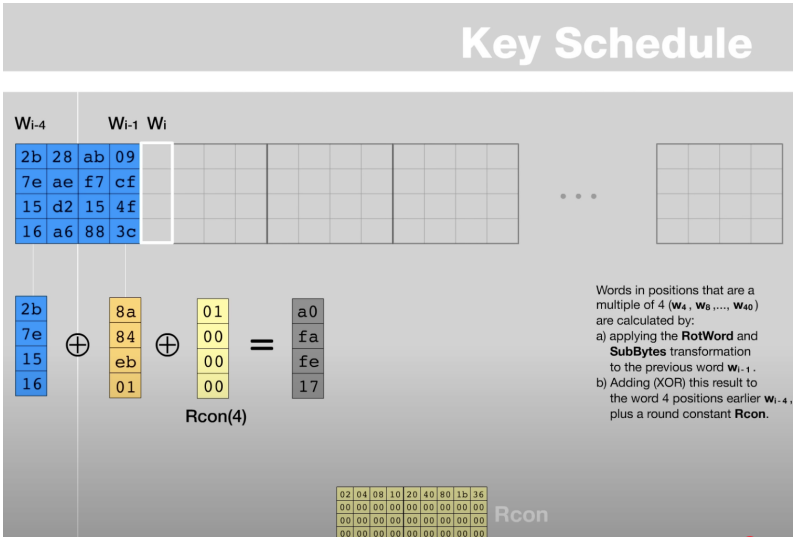
1.6 Stream cipher Uno *stream cipher* è un cifrario simmetrico che cifra il plaintext un bit o un byte alla volta, generando una sequenza pseudo-casuale di bit chiamata **keystream**, la quale viene combinata con il plaintext mediante un'operazione bitwise, solitamente l'XOR.

Il cifrario prende in input una **chiave segreta** (key) e, opzionalmente, un **valore di inizializzazione** (IV), per produrre il keystream. La sicurezza di uno stream cipher dipende dalla **qualità crittografica del keystream**: esso deve essere indistinguibile da una sequenza casuale e non deve rivelare alcuna informazione sulla chiave.

A differenza dei block cipher, gli stream cipher non operano su blocchi di dati fissi, ma sono progettati per operare in **tempo reale**, rendendoli adatti a scenari con requisiti di bassa latenza o dove la quantità di dati non è nota a priori (es. trasmissioni audio/video o protocolli di rete).

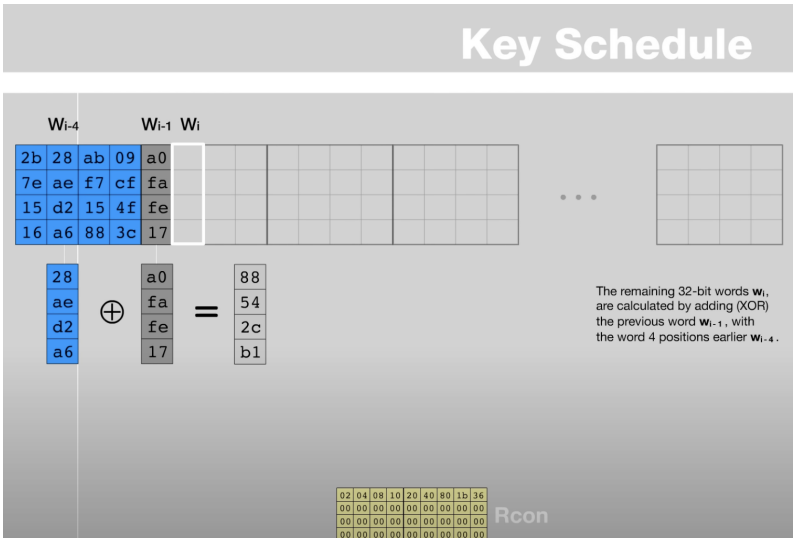
Esistono due categorie principali:

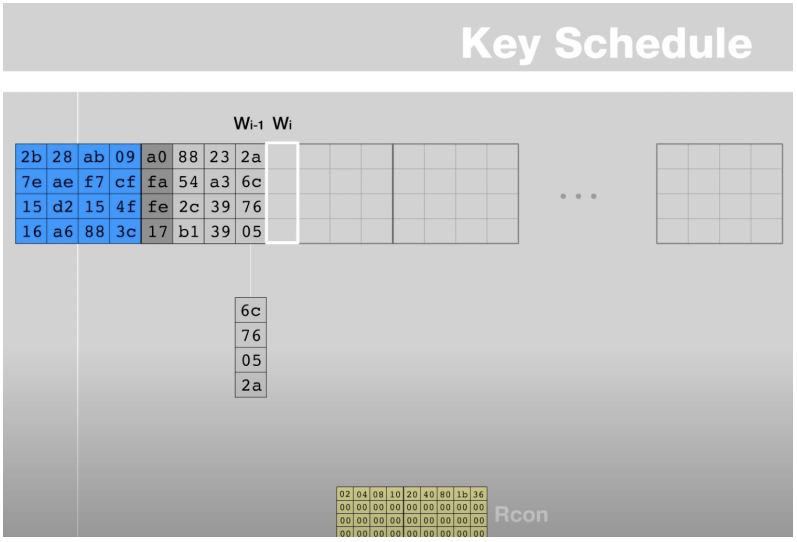




- **Synchronous stream cipher**: il keystream dipende solo dalla chiave (e dall'IV), ed è generato indipendentemente dal plaintext e dal ciphertext. È fondamentale che il mittente e il destinatario siano sincronizzati.
- **Self-synchronizing stream cipher** (o *cipher feedback mode*): il keystream dipende anche da una finestra del ciphertext precedente. In questo modo, può ri-sincronizzarsi automaticamente dopo una perdita di dati.

Un esempio pratico di stream cipher è **RC4** (oggi considerato insicuro), mentre nelle implementazioni moderne si prediligono stream cipher basati su primitive solide come **ChaCha20** o **Salsa20**, utilizzati in protocolli come TLS e VPN.



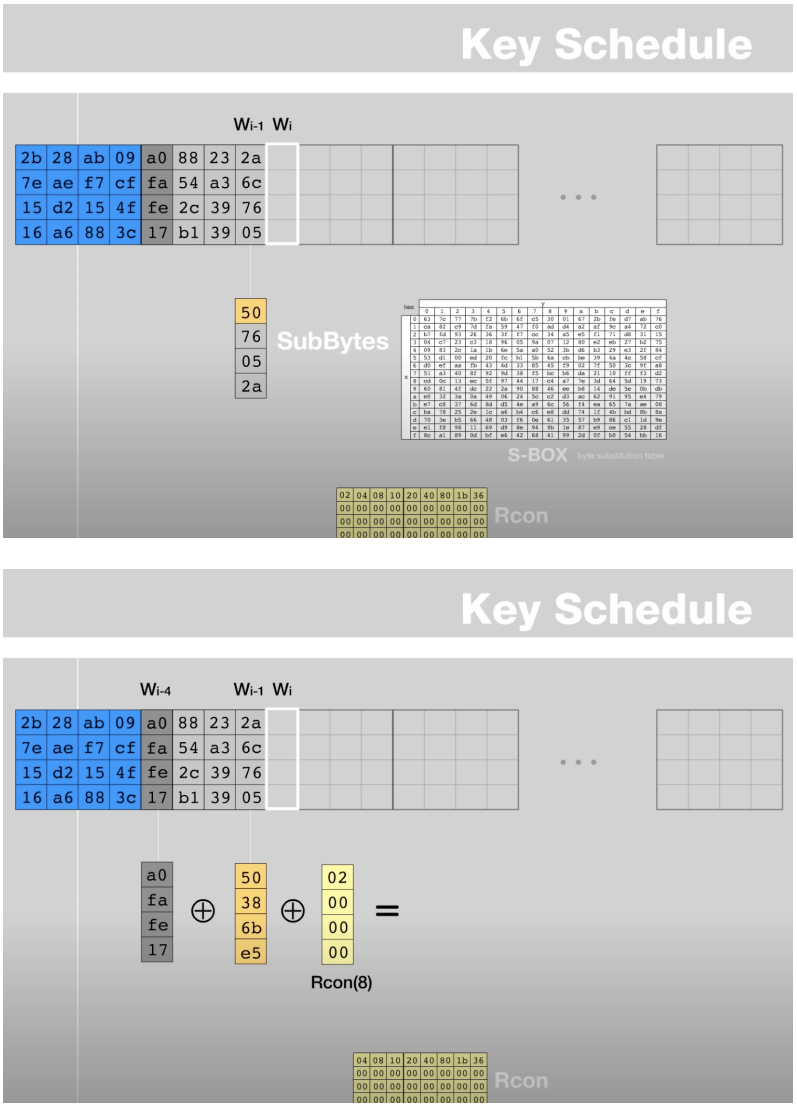


1.7 One-Time Pad (OTP)

OTP è *unconditionally secure* Il One-Time Pad è un cifrario simmetrico che offre una sicurezza incondizionata, ovvero non dipendente dalla potenza computazionale dell'attaccante.

OTP è uno stream cipher con le seguenti proprietà fondamentali:

1. Il **keystream** è generato da un **true random generator** (TRNG), quindi **non** da un generatore pseudo-casuale (PRNG).
2. Il **keystream è conosciuto solo dalle parti legittime** e deve essere condiviso tramite un canale sicuro.
3. Ogni **keystream viene utilizzato una sola volta**.
   - In particolare: $\text{length}_{\text{key}} = \text{length}_{\text{plaintext}}$



Nota importante: Se anche solo una di queste regole viene violata, **l'OTP perde completamente la sua sicurezza**.

Limiti pratici: A livello pratico, l'OTP è **inapplicabile su larga scala** a causa della difficoltà nella generazione, distribuzione e gestione sicura delle chiavi.

Esempi moderni: Stream cipher pratici che offrono buone proprietà di sicurezza includono: ChaCha20, AES-CTR, CTR/CFB, Grain, Trivium.

Nota: è consigliato approfondire il funzionamento dell'OTP passo-passo e consultare dimostrazioni o video esplicativi, ad esempio su YouTube.

1.8 Cypher mode block

2 Asymmetric cryptography

2.1 RSA

2.2 Diffie-Hellman exchange

- **Groups:** è un insieme G dotato di un'operazione binaria $*$ che soddisfa le seguenti proprietà (modulo p omesso negli esempi):
  - **Chiusura**: Per ogni $a, b \in G$, si ha $a * b \in G$.
  - **Associatività**: Per ogni $a, b, c \in G$, vale $(a * b) * c = a * (b * c)$.
  - **Elemento neutro**: Esiste un elemento $e \in G$ tale che per ogni $a \in G$, si ha $a * e = e * a = a$.
  - **Elemento inverso**: Per ogni $a \in G$, esiste un elemento $a^{-1} \in G$ tale che $a * a^{-1} = a^{-1} * a = e$.

  Se inoltre l'operazione è **commutativa** ($a * b = b * a$ per ogni $a, b \in G$), il gruppo è detto **abeliano**.
- **Sub-Groups:** Porzione di gruppo;
- **Finite Groups:** Gruppo con numero finito di elementi;
- **Ciclic Groups:** Un gruppo ciclico è un gruppo che contiene almeno un elemento g (detto generatore) tale che tutte le potenze (o iterazioni dell'operazione del gruppo) di quel generatore generano tutti gli altri elementi del gruppo. $\rightarrow$ G = $\{g^k \mid k \in \mathbb{Z}\}$

$\mathbb{Z}_p^*$: è l'insieme degli interi da 1 a p-1, con l'operazione di moltiplicazione modulo p (escluso 0).

Il protocollo di **Diffie-Hellman Key Exchange** è un protocollo per scambiare una chiave segreta tra due utenti in modo sicuro su un canale non sicuro. Vediamo il processo con un esempio pratico.

Parametri pubblici

Prima di tutto, Alice e Bob scelgono un numero primo pubblico p (modulo) e una base (o generatore) g:

- Numero primo pubblico: $p = 23$
- Generatore pubblico: $g = 5$

Questi valori sono conosciuti da tutti e possono essere intercettati senza problemi.

Scelta delle chiavi private

Alice e Bob scelgono ciascuno una chiave privata segreta:

- Chiave privata di Alice: $a = 6$
- Chiave privata di Bob: $b = 15$

Calcolo delle chiavi pubbliche

Entrambi calcolano le rispettive chiavi pubbliche usando la formula:

$$X = g^a \mod p$$

- Alice calcola: $A = 5^6 \mod 23 = 8$
- Bob calcola: $B = 5^{15} \mod 23 = 19$

Ora Alice e Bob si scambiano pubblicamente A e B.

Calcolo della chiave segreta condivisa

Alice e Bob ora utilizzano il valore pubblico ricevuto per calcolare la chiave segreta condivisa:

- Alice calcola: $S = B^a \mod p = 19^6 \mod 23 = 2$
- Bob calcola: $S = A^b \mod p = 8^{15} \mod 23 = 2$

Entrambi arrivano alla stessa chiave segreta 2, che ora può essere usata per cifrare la comunicazione!

Considerazioni sulla Sicurezza

Un attaccante che intercetta i messaggi vede solo p, g, A e B, ma per calcolare S dovrebbe risolvere il problema del logaritmo discreto, che è computazionalmente difficile se i numeri sono sufficientemente grandi. Per questo motivo è fondamentale usare un numero che sia molto grande (migliaia di bit) e primo.

Man-in-the-middle attack e solution:

Il Man-in-the-Middle (MitM) attack durante lo scambio di chiavi Diffie-Hellman (DH) è un problema serio, poiché il protocollo base non fornisce autenticazione. Un attaccante può intercettare e sostituire le chiavi pubbliche scambiate tra le due parti, instaurando due connessioni separate (una con ciascuna parte) e decrittando i messaggi.
Soluzioni:

- Sfruttare PKI (public key infrastructure) per autenticare le chiavi pubbliche firmante da un autorità certificata.
- Chiavi firmate con RSA.
- Viene autenticato e reso sicuro il canale di comunicazione

Quando si dice che Diffie-Hellman è un protocollo non interattivo, si fa riferimento al fatto che i due partecipanti (ad esempio, Alice e Bob) possono stabilire una chiave segreta condivisa senza la necessità di scambiarsi messaggi diretti in tempo reale o interagire direttamente in ogni passaggio.

2.2.1 DLP(Discrete Logarithm Problem)

:

è un problema matematico che si basa sull'operazione di moltiplicazione in gruppi, in particolare in gruppi ciclici.
In altre parole, dato g (generatore di G) e h (appartente a G), il compito è trovare x,

ovvero il logaritmo discreto, tale che elevando g alla potenza x modulo p si ottiene h.

Supponiamo di avere un gruppo $\mathbb{Z}_p^*$ (ad esempio, $p = 7$), con il generatore $g = 3$, e vogliamo trovare x tale che $3^x \equiv 4 \mod 7$. Il problema del logaritmo discreto è trovare x che soddisfi questa equazione.

1. $3^1 = 3 \mod 7$
2. $3^2 = 9 \mod 7 = 2$
3. $3^3 = 6 \mod 7$
4. $3^4 = 18 \mod 7 = 4$

Quindi, $x = 4$ è la soluzione, perché $3^4 \equiv 4 \mod 7$.

Non esiste un algoritmo efficiente (polinomiale) per risolvere il DLP in generale, rendendo la sicurezza di molti algoritmi crittografici basati su DLP molto solida.

If DLP can be easily solved, then DHP can be easily solved.
DLP can be applied for any cyclic group.
DLP id indipendent of the generator.

DLP è usato in:

- Diffie-Hellman Key Exchange
- DSA (Digital Signature Algorithm)
- ElGamal Encryption
- ECDSA (Elliptic Curve Digital Signature Algorithm)
- Lattice-based Cryptography
- Schnorr signature

In $\mathbb{Z}_p^*$, to achieve 80-bit security, the prime p must be at least 1024 bit long, it is more efficiente than $GF(2^m)$.

2.2.2 Galois Field

$GF(2^m)$ è un campo finito che contiene 2^m elementi.

Come esempio pratico, m = 256 è una dimensione comunemente utilizzata per garantire un livello di sicurezza simile a 2048 bit in un gruppo basato su $\mathbb{Z}_p^*$.

Sebbene si possa definire un DLP in un campo finito come $GF(2^m)$, non è ideale come base per la crittografia. Questo perché il problema risulta relativamente più facile da risolvere rispetto al DLP in $\mathbb{Z}_p^*$. Sebbene il DLP in $GF(2^m)$ non sia ideale, le curve ellittiche su campi finiti come $GF(2^m)$ sono molto sicure e efficienti. Le curve ellittiche offrono una sicurezza elevata con chiavi più corte e sono ampiamente utilizzate in crittografia moderna, grazie alla loro resistenza a determinati attacchi e alle loro operazioni efficienti.

In sintesi:

- Per il DLP, $\mathbb{Z}_p^*$ è la scelta migliore.
- Per la sicurezza avanzata e l'efficienza, le curve ellittiche su $GF(2^m)$ sono preferite, ma non sono utilizzate per il DLP tradizionale.

2.2.3 DLP in cyclic subgroup

Usare il DLP in un sottogruppo ciclico di ordine primo grande è più sicuro, più efficiente e resistente agli attacchi.
Sottogruppo: Un sottogruppo è semplicemente una parte di un gruppo che segue le stesse regole del gruppo originale.
Il teorema di Lagrange dice che l'ordine di un sottogruppo deve essere un divisore dell'ordine del gruppo. */rightarrow* Se G ha ordine 12, i possibili sottogruppi avranno ordini 1, 2, 3, 4, 6, 12 (tutti i divisori di 12).
Se G è ciclico, ogni sottogruppo è anch'esso ciclico ed ha esattamente un sottogruppo per ogni divisore di n. */rightarrow* Se G ha ordine 12 allora esistono esattamente 6 sottogruppi per ogni divisore.

N.B! DH si applica tipicamente a un sottogruppo ciclico di ordine primo (che ha come ordine (ordine: cardinalità del gruppo) un numero primo) grande (se g cade in un sottogruppo piccolo allora il DLP diventa facile da risolvere).
Come si sceglie un sottogruppo ciclico di ordine primo grande?

- Scegliere un numero primo p grande (2048 bit) tale che $p = 2q + 1$ con q primo.
- Scegliere un generatore g che appartiene a q
- Si verifica che
$$g^q \equiv 1 \mod p$$
, assicurandosi che g appartenga davvero al sottogruppo di ordine q.

2.3 ElGamal Cryptosystem ElGamal estende l'idea di Diffie-Hellman per creare un cifrario a chiave pubblica. Bob genera una coppia di chiavi pubblica e privata.

- Sceglie un **numero primo** p.
- Sceglie un **generatore** g del gruppo ciclico $\mathbb{Z}_p^*$.
- Sceglie una **chiave privata** x (segreta).
- Calcola la **chiave pubblica** y:
$$y = g^x \mod p$$
- Pubblica (p, g, y) e tiene segreta x.

Cifratura
Alice vuole inviare un messaggio m a Bob.

- Sceglie un **valore casuale** k (nonce segreto).
- Calcola il primo valore cifrato:

$$c_1 = g^k \mod p$$

- Calcola il secondo valore cifrato:

$$c_2 = m \cdot y^k \mod p$$

- Invia la coppia cifrata (c_1, c_2) a Bob.

Decifratura
Bob riceve (c_1, c_2) e usa la sua chiave privata x per decifrare.

- Calcola il valore condiviso segreto s:

$$s = {c_1}^x \mod p$$

- Calcola l'**inverso moltiplicativo** di s modulo p, denotato come s^{-1}.
- Recupera il messaggio originale:

$$m = c_2 \cdot s^{-1} \mod p$$

Esempio pratico:
Supponiamo di lavorare in $\mathbb{Z}_{23}^*$ con i seguenti valori:

- **Primo**: $p = 23$
- **Generatore**: $g = 5$
- **Chiave privata di Bob**: $x = 6$
- **Chiave pubblica di Bob**:

$$y = g^x \mod p = 5^6 \mod 23 = 8$$

Cifratura di un messaggio: Alice vuole inviare il messaggio $m = 10$.

- Sceglie un valore casuale $k = 3$.
- Calcola il primo valore cifrato:

$$c_1 = g^k \mod p = 5^3 \mod 23 = 10$$

- Calcola il secondo valore cifrato:

$$c_2 = m \cdot y^k \mod p = 10 \times 8^3 \mod 23 = 10 \times 12 \mod 23 = 5$$

- Invia $(c_1, c_2) = (10, 5)$ a Bob.

Decifratura di Bob: Bob riceve $(10, 5)$.

- Calcola il valore segreto condiviso:

$$s = {c_1}^x \mod p = 10^6 \mod 23 = 12$$

- Trova l'inverso moltiplicativo:

$$s^{-1} = 12^{-1} \mod 23 = 2$$

- Recupera il messaggio:

$$m = c_2 \times s^{-1} \mod 23 = 5 \times 2 \mod 23 = 10$$

Risultato: Bob ha decifrato correttamente il messaggio $m = 10$!

2.4 Combinazione di Diffie-Hellman e ElGamal Fase 1: Scambio di Chiavi Diffie-Hellman

Alice e Bob vogliono stabilire una chiave condivisa.

- Scelgono un **numero primo** p e un **generatore** g del gruppo ciclico $\mathbb{Z}_p^*$.
- Alice sceglie un **segreto** a e calcola:

$$A = g^a \mod p$$

- Bob sceglie un **segreto** b e calcola:

$$B = g^b \mod p$$

- Si scambiano pubblicamente A e B.
- Alice calcola la chiave condivisa:

$$s = B^a \mod p$$

- Bob calcola la stessa chiave:

$$s = A^b \mod p$$

Risultato: Entrambi ottengono la stessa chiave segreta condivisa s, che useranno per cifrare i messaggi.

Fase 2: Cifratura con ElGamal

Ora, Alice vuole inviare un messaggio m a Bob, usando la chiave condivisa s.

- Alice sceglie un valore casuale k.
- Calcola il primo valore cifrato:

$$c_1 = g^k \mod p$$

- Calcola il secondo valore cifrato usando la chiave condivisa s:

$$c_2 = m \cdot s^k \mod p$$

- Invia (c_1, c_2) a Bob.

Fase 3: Decifratura con ElGamal

Bob riceve (c_1, c_2) e recupera il messaggio.

- Calcola il valore segreto condiviso:

$$s_k = {c_1}^b \mod p$$

- Trova l'inverso moltiplicativo di s_k, denotato come s_k^{-1}.
- Recupera il messaggio:

$$m = c_2 \cdot s_k^{-1} \mod p$$

Risultato: Bob ha recuperato il messaggio originale m!

2.5 Digital Signature Phase

1. Key generation: private key x and public key y.
2. Signature generation: Alice signs a message m using her private key x to create a signature s.
3. Signature verification: Bob verifies the signature s using Alice's public key y and the message m.

Security properties implies:

1. Integrity: The message m has not been altered.
2. Authenticity: The signature s is valid and was created by Alice.
3. Non-repudiation: Alice cannot deny having signed the message m.

4. Confidentiality: The message m is kept secret from unauthorized parties.

In comparison with MAC: MAC with symmetric key, autentica l'integrità del messaggio e dimostra che proviene da una fonte che possiede la chiave segreta condivisa. Tuttavia, il MAC non prova l'identità del mittente in modo univoco, poiché entrambe le parti (mittente e destinatario) devono avere la stessa chiave segreta. Se una terza parte conosce la chiave segreta, potrebbe generare un MAC valido. Garantisce l'integrità del messaggio, ma non l'autenticità in modo definitivo (chiave hackerata), e anche Non Ripudio. La crittografia simmetrica è più veloce e meno intensiva rispetto alla crittografia asimmetrica.

2.6 RSA

Il **RSA Signature Scheme** è uno dei più noti e utilizzati algoritmi per la creazione di firme digitali. Si basa sull'algoritmo **RSA** (Rivest-Shamir-Adleman), che è un sistema di crittografia asimmetrica, cioè utilizza una coppia di chiavi: una pubblica e una privata.

Il **RSA Signature Scheme** sfrutta la crittografia asimmetrica per generare e verificare una firma digitale, garantendo **autenticità**, **integrità** e **non ripudio**. Ecco come funziona in dettaglio:

2.6.1 Generazione delle chiavi

Per utilizzare RSA per la firma digitale, bisogna prima generare una coppia di chiavi:

- **Chiave privata** (d, n): utilizzata per firmare i messaggi.
- **Chiave pubblica** (e, n): utilizzata per verificare la firma.

Processo di generazione:

1. **Selezione di due numeri primi**: Scegli due numeri primi p e q abbastanza grandi.
2. **Calcolo di** n: Calcola il prodotto $n = p \cdot q$. Questo valore è usato come parte sia della chiave privata che della chiave pubblica.
3. **Calcolo di** $\varphi(n)$: Calcola $\varphi(n) = (p-1) \cdot (q-1)$, che è la funzione di Eulero di n.
4. **Selezione di** e: Scegli un numero e che sia coprimo con $\varphi(n)$ (ovvero, che non abbia fattori comuni con $\varphi(n)$). Di solito si sceglie $e = 65537$, che è un valore comune per e.
5. **Calcolo di** d: Calcola d, che è l'inverso moltiplicativo di e modulo $\varphi(n)$. In altre parole, d è il numero che soddisfa la congruenza $e \cdot d \equiv 1 \mod \varphi(n)$.

La coppia di chiavi sarà quindi:

- **Chiave privata**: (d, n).
- **Chiave pubblica**: (e, n).

2.6.2 Generazione della firma

Quando una persona (il mittente) vuole firmare digitalmente un messaggio, esegue i seguenti passaggi:

1. **Creazione dell'hash del messaggio**: Prima di firmare il messaggio, si calcola l'hash del messaggio (solitamente usando una funzione di hash sicura come SHA-256). Questo passo serve a ridurre la dimensione del messaggio, rendendo più efficiente la firma.
2. **Firma del messaggio**: La firma digitale è una cifra del valore dell'hash usando la chiave privata. In termini matematici, la firma S del messaggio m (che è rappresentato come il suo hash $H(m)$) è calcolata come:

$$S = H(m)^d \mod n$$

   Dove:

   - $H(m)$ è l'hash del messaggio m.
   - d è la chiave privata.
   - n è il modulo della chiave.

   La **firma** è il risultato di questa operazione e viene inviata insieme al messaggio.

2.6.3 Verifica della firma

Per verificare la firma, il destinatario (il verificatore) deve avere la chiave pubblica del firmatario.

Nel caso del RSA Signature Scheme, la funzione di hash viene utilizzata per ridurre la lunghezza del messaggio prima che venga firmato, rendendo la firma più gestibile (perché RSA firma numeri, non messaggi di lunghezza arbitraria). L'algoritmo di firma digitale RSA firma solo l'hash del messaggio, non il messaggio intero.

Se la funzione di hash utilizzata non è collision-resistant, un attaccante potrebbe essere in grado di trovare due messaggi distinti con lo stesso hash e "sostituire" il messaggio firmato con un altro. Questo potrebbe portare a un attacco di collisione.

La collision resistance è una proprietà importante di una funzione di hash crittografica. Si riferisce alla difficoltà di trovare due input distinti che producono lo stesso hash. In altre parole, una funzione di hash è considerata collision-resistant se è difficile, anche con una grande quantità di calcoli, trovare due messaggi distinti m1 e m2 tali che H(m1) = H(m2), dove H è la funzione di hash.

In a **Blind signature scheme**, il firmatario firma un messaggio senza conoscere il contenuto del messaggio stesso. Questo è utile in scenari in cui la privacy è fondamentale, come nei sistemi di voto elettronico o nei pagamenti anonimi.

2.7 ElGamal Signature Scheme Per firmare un messaggio, l'utente esegue i seguenti passaggi:

1. **Creazione dell'hash del messaggio**: Calcola l'hash del messaggio, denotato con $H(m)$, utilizzando una funzione di hash sicura (ad esempio, SHA-256).
2. **Scelta di un numero casuale** k: Scegli un numero casuale k, con $1 \leq k \leq p-2$, che deve essere mantenuto segreto e non riutilizzato per altre firme.
3. **Calcolo dei valori della firma**: La firma digitale consiste di due valori r e s, calcolati come segue:

$$r = g^k \mod p$$

$$s = k^{-1} \cdot (H(m) - x \cdot r) \mod (p-1)$$

Dove:

- k^{-1} è l'inverso di k modulo $p-1$,
- $H(m)$ è l'hash del messaggio m,
- x è la chiave privata,
- r è il primo valore della firma,
- s è il secondo valore della firma.

La firma risultante è il paio di valori (r, s).

2.8 Generazione della Firma con DSA Per firmare un messaggio, l'utente esegue i seguenti passaggi:

1. **Creazione dell'hash del messaggio**: Calcola l'hash del messaggio, denotato con $H(m)$, utilizzando una funzione di hash sicura (ad esempio, SHA-1 o SHA-256).
2. **Scelta di un numero casuale** k: Scegli un numero casuale k, con $1 \leq k \leq q-1$, dove q è un numero primo di lunghezza fissa associato al parametro della chiave pubblica. k deve essere scelto in modo tale che non venga mai riutilizzato per altre firme, poiché se k viene riutilizzato, la chiave privata potrebbe essere compromessa.
3. **Calcolo dei valori della firma**: La firma digitale consiste di due valori r e s, calcolati come segue:

$$r = (g^k \mod p) \mod q$$

$$s = k^{-1} \cdot (H(m) + x \cdot r) \mod q$$

Dove:

- g è il generatore, una parte della chiave pubblica,
- p è un numero primo,

- q è un altro numero primo associato a p,
- x è la chiave privata,
- k^{-1} è l'inverso di k modulo q,
- r è il primo valore della firma,
- s è il secondo valore della firma.

La firma risultante è il paio di valori (r, s).

2.9 Crittografia con Curve Ellittiche La **Crittografia con Curve Ellittiche** (Elliptic Curve Cryptography, ECC) è una tecnica di crittografia asimmetrica che si basa sulla matematica delle **curve ellittiche**. ECC è considerata una delle soluzioni più efficienti rispetto ad altri algoritmi di crittografia asimmetrica, come RSA, poiché offre lo stesso livello di sicurezza con chiavi di dimensioni molto più piccole. Questo la rende particolarmente utile in contesti con risorse limitate, come dispositivi mobili o smartcard.

2.9.1 Cos'è una Curva Ellittica?

Una **curva ellittica** è una curva algebrica definita da un'equazione di secondo grado in due variabili. La forma generale dell'equazione di una curva ellittica è:

$$y^2 = x^3 + ax + b$$

dove a e b sono coefficienti che determinano la forma della curva, e la curva stessa è definita su un campo finito $\mathbb{F}_p$ o $\mathbb{F}_2$ (un campo di numeri interi modulo un numero primo p, o il campo binario).

2.9.2 Proprietà delle Curve Ellittiche

Le curve ellittiche possiedono alcune proprietà matematiche particolari che le rendono adatte per la crittografia:

- **Gruppo abeliano**: I punti sulla curva formano un gruppo abeliano, il che significa che possiamo sommare due punti sulla curva per ottenere un altro punto della curva. La somma di due punti è definita tramite una operazione geometrica, che è la base dell'algoritmo di cifratura.
- **Operazione di punto**: Se abbiamo due punti P e Q sulla curva, possiamo sommarli per ottenere un nuovo punto $R = P+Q$. Allo stesso modo, possiamo moltiplicare un punto per uno scalare k, che significa aggiungere il punto P a sé stesso k volte: $kP = P + P + \cdots + P$.
- **Problema del logaritmo discreto**: L'elemento di sicurezza chiave in ECC è il **problema del logaritmo discreto**. Data una curva ellittica e un punto P, è computazionalmente difficile determinare k dato kP. Questo è simile

al problema del logaritmo discreto in altri contesti (come il caso di Diffie-Hellman o DSA), ma con curve ellittiche il problema è molto più difficile da risolvere con chiavi di dimensioni inferiori.

2.9.3 Come Funziona ECC nella Crittografia Asimmetrica?

ECC può essere utilizzata per vari scopi, come la **cifratura** e la **firma digitale**. La parte più comune della crittografia ECC si basa sull'uso delle curve ellittiche per generare una coppia di chiavi pubblica e privata.

2.9.4 Generazione delle Chiavi

1. **Scegliere una curva ellittica**: Si sceglie una curva ellittica definita su un campo finito $\mathbb{F}_p$. La scelta della curva e dei parametri (come il generatore G) è fondamentale per la sicurezza del sistema.

2. **Generare la chiave privata**: La chiave privata è un numero intero casuale d, che è scelto da una distribuzione uniforme in un intervallo definito. La chiave privata deve essere mantenuta segreta.

3. **Generare la chiave pubblica**: La chiave pubblica è il punto Q sulla curva che si ottiene moltiplicando il punto generatore G per la chiave privata d:

$$Q = d \cdot G$$

Il punto Q è la chiave pubblica. Poiché d è segreto, solo la persona che possiede la chiave privata può generare la corrispondente chiave pubblica.

2.9.5 Cifratura con ECC (Elliptic Curve Integrated Encryption Scheme, ECIES)

L'algoritmo ECIES è uno degli algoritmi di crittografia simmetrica utilizzato con le curve ellittiche. La cifratura funziona come segue:

1. **Generare una chiave segreta condivisa**: Un mittente e un destinatario utilizzano la chiave pubblica e privata per generare una chiave segreta condivisa. Ad esempio, il mittente calcola:

$$K = d_{\text{mittente}} \cdot Q_{\text{destinatario}}$$

   e il destinatario calcola:

$$K = d_{\text{destinatario}} \cdot Q_{\text{mittente}}$$

   Poiché $Q_{\text{destinatario}} = d_{\text{destinatario}} \cdot G$ e $Q_{\text{mittente}} = d_{\text{mittente}} \cdot G$, entrambi ottengono lo stesso punto K sulla curva, che è utilizzato per derivare una chiave simmetrica per la cifratura dei dati.

2. **Cifratura del messaggio**: Il messaggio viene cifrato utilizzando la chiave simmetrica derivata dalla chiave segreta K.

3. **Decifratura del messaggio**: Il destinatario usa la sua chiave privata per calcolare la stessa chiave segreta K e decifra il messaggio cifrato.

2.9.6 Firma Digitale con ECC (ECDSA)

Il **Elliptic Curve Digital Signature Algorithm (ECDSA)** è un algoritmo di firma digitale basato su ECC. La generazione della firma funziona come segue:

1. **Creare un hash del messaggio**: Prima di firmare un messaggio, viene calcolato un hash del messaggio, $H(m)$.
2. **Generare la firma**: Sia d la chiave privata e G il generatore della curva, la firma è costituita da due valori r e s:

$$r = (g^k \mod p) \mod q$$

$$s = k^{-1} \cdot (H(m) + r \cdot d) \mod q$$

   dove k è un numero casuale scelto durante la firma.
3. **Verifica della firma**: La firma viene verificata utilizzando la chiave pubblica Q (calcolata come $Q = d \cdot G$).

2.9.7 Vantaggi di ECC

- **Chiavi più corte e più sicure**: ECC offre lo stesso livello di sicurezza di algoritmi come RSA, ma con chiavi molto più corte. Ad esempio, una chiave ECC di 256 bit fornisce un livello di sicurezza simile a una chiave RSA di 3072 bit. Questo riduce i requisiti di memoria e aumenta la velocità di elaborazione.
- **Efficienza computazionale**: Le operazioni con curve ellittiche sono molto più veloci rispetto agli algoritmi basati su RSA, specialmente quando si usano dispositivi con risorse limitate (come smartphone, IoT, etc.).
- **Ampia applicabilità**: ECC è ampiamente utilizzato in applicazioni moderne di crittografia, come HTTPS (SSL/TLS), Bitcoin, e altre criptovalute, oltre che in protocolli di autenticazione.

2.9.8 Conclusioni

La **crittografia a curve ellittiche (ECC)** è una delle tecnologie di crittografia più avanzate ed efficienti. Fornisce una sicurezza elevata utilizzando chiavi molto più corte rispetto agli algoritmi tradizionali come RSA, il che la rende particolarmente utile in ambienti con risorse limitate. ECC è ampiamente adottata in molte applicazioni moderne, soprattutto per la firma digitale e la cifratura in ambienti mobili e di blockchain.

3 Digital Signature

4 Certificate

5 Hash