



ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Appunti di "*Fondamenti di Cybersecurity*"

A.A. 2024/25

Ivan De Simone

Introduzione

Con il termine **cybersecurity** ci riferiamo a qualsiasi tecnologia o pratica per prevenire gli attacchi informatici o mitigare l'impatto.

"La cybersecurity mira a proteggere sistemi, applicazioni, dispositivi, dati sensibili e beni finanziari di individui ed organizzazioni da minacce che vanno da semplici virus informatici a sofisticati attacchi ransomware." (IBM)

"La sicurezza informatica è l'impiego di persone, politiche, processi e tecnologie al fine di proteggere le organizzazioni, i loro sistemi critici e le informazioni sensibili dagli attacchi digitali." (Gartner)

La cybersecurity non riguarda la sicurezza dei singoli computer, ma quella dei sistemi informatici, i quali integrano:

- computer
- comunicazioni
- persone (utenti e operatori)

Attacchi informatici (cyber-attacchi)

Un **attacco informatico** è qualsiasi tentativo intenzionale di rubare, esporre, alterare, disabilitare o distruggere dati, applicazioni o altri beni tramite accesso non autorizzato a una rete, un sistema informatico o un dispositivo digitale.

Minacce comuni

Gli attacchi alla sicurezza informatica più comuni e noti includono:

- **Phishing e social engineering:** l'attaccante inganna gli utenti legittimi, inducendoli ad eseguire azioni che aprono le porte a utenti non autorizzati
- **Rischi legati a servizi internet:** incapacità di aziende, partner e fornitori di proteggere adeguatamente i servizi cloud o altri servizi connessi a internet da minacce note
- **Compromissione della password:** l'utente non autorizzato cerca di identificare password comuni, tramite software o hacking, con lo scopo accedere a sistemi o dati riservati
- **Uso improprio di informazioni:** utenti autorizzati diffondono o abusano di informazioni o dati a cui hanno accesso

- **Attacchi in rete e man-in-the-middle**: l'attaccante intercetta, reindirizza o interrompe il traffico di rete non protetto
- **Attacchi alla catena di fornitura**: partner, fornitori o altre risorse di terze parti vengono compromessi, creando uno spiraglio per attaccare sistemi aziendali
- **Denial-of-Service (DoS)**: l'aggressore sovraccarica il sistema causando interruzioni o rallentamenti temporanei. L'attacco **DoS distribuito (DDoS)** fa uguale, utilizzando però una rete di dispositivi attaccanti. *Ogni giorno vengono segnalati migliaia di attacchi DDoS, i quali continuano ad aumentare in complessità, volume e frequenza*
- **Ransomware**: software dannoso infetta il sistema limitando l'accesso a dati e/o strumenti fino al pagamento di un riscatto. Rischio di data-leak se il riscatto non viene pagato

ARPANET

Nel 1969 diventa online, nel 1973 Robert Metcalfe avverte che è insicura, nel 1983 adotta TCP/IP.

Sempre nel 1983 viene coniato il termine "computer virus", da Fred Cohen.

Primo worm (Morris worm)

Nel 1988 Robert Morris rilascia inavvertitamente il primo worm: quando una copia inizia l'esecuzione, la prima cosa che fa è individuare altri host da infettare. Questo avvenne tramite una vulnerabilità nei server mail. L'idea era rilasciarlo come esperimento di ricerca, ma un bug nel codice causò la replicazione incontrollata mandando in crash il 10% dei computer di ARPANET.

Morris venne condannato a 3 anni di libertà vigilata e 10.000 dollari di multa.

Primo utilizzo documentato di un exploit di buffer overflow.

Definizioni

Vulnerabilità → una debolezza che può essere sfruttata per causare danni

Attacco → un metodo per sfruttare una vulnerabilità

Minaccia → un avversario capace e motivato che lancia un attacco

Vulnerabilità zero-day → una vulnerabilità sconosciuta a chi dovrebbe sistemarla

Finestra di opportunità → tempo che intercorre tra l'istante in cui un exploit software diventa attivo per la prima volta ed il momento in cui una patch viene rilasciata ed applicata al sistema interessato

Attacco zero-day → un attacco che si verifica durante la finestra di opportunità

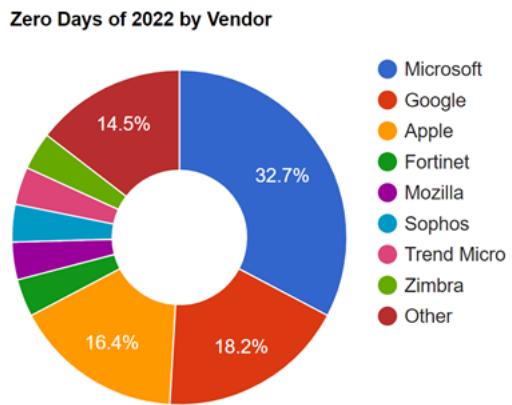
Nel 2005 la durata media di una finestra di opportunità era 54 giorni, nel 2014 è aumentata fino a quasi 12 mesi.



Strategie di prevenzione:

- Identificare e fixare ogni vulnerabilità (solitamente dovuta a bug)
- Analizzare gli attacchi ed eliminare le vulnerabilità sfruttate da essi

Le Big Tech, che sviluppano quasi tutti i sistemi operativi utilizzati in tutto il mondo, sono generalmente i principali portatori di vulnerabilità zero-day.



Prerequisiti

Per affrontare gli argomenti del corso servono alcune conoscenze pregresse.

Probabilità discreta

Chiamiamo **universo** o **spazio campionario** l'insieme finito U dei possibili esiti di un esperimento aleatorio.

Una **distribuzione di probabilità** P su U è una funzione $P : U \rightarrow [0, 1]$ tale che $\sum_{x \in U} P(x) = 1$.

Lancio di un dado

$$U = \{1, 2, 3, 4, 5, 6\}$$

$$P(1) = P(2) = P(3) = P(4) = P(5) = P(6) = 1/6$$

$U = \{0, 1\}^n$ rappresenta l'insieme di tutte le stringhe binarie di lunghezza $n \rightarrow$ esistono 2^n elementi in totale.

Alcune distribuzioni note:

- **Distribuzione uniforme:** $\forall x \in U, P(x) = \frac{1}{|U|}$
- **Distribuzione puntuale** su x_0 : $P(x_0) = 1, P(x) = 0 \quad \forall x \neq x_0$
- Ce ne sono altre: binomiale, geometria, Poisson...

c

Eventi

Un **evento** A è un sottoinsieme dell'universo, $A \subseteq U$.

La **probabilità di** A è $Pr[A] = \sum_{x \in A} P(x)$.

$$A = \text{'esce un dispari sul dado'} = \{1, 3, 5\}$$

$$Pr[A] = 1/6 + 1/6 + 1/6 = 1/2$$

Dati due eventi A_1 e A_2 , $A_1 \cup A_2$ è ancora un evento.

- $Pr[A_1 \cup A_2] = Pr[A_1] + Pr[A_2] - Pr[A_1 \cap A_2]$
- $Pr[A_1 \cup A_2] \leq Pr[A_1] + Pr[A_2]$ (Union bound)
- Se $A_1 \cap A_2 = \emptyset \implies Pr[A_1 \cup A_2] = Pr[A_1] + Pr[A_2]$

Due eventi A e B sono **indipendenti** se $Pr[A \cap B] = Pr[A] \cdot Pr[B]$.

Variabili aleatorie

Una **variabile aleatoria** X è una funzione $X : U \rightarrow V$

In generale si dice che X induce una distribuzione su V , ovvero che X prende valori in V e definisce una distribuzione su V .

$$X : U \rightarrow V = \{ \text{"pari"}, \text{"dispari"} \}$$

$$X(2) = X(4) = X(6) = \text{"pari"}$$

$$X(1) = X(3) = X(5) = \text{"dispari"}$$

$$\Pr[X = \text{"pari"}] = \Pr[X = \text{"dispari"}] = 1/2$$

Sia S un insieme (ad esempio $S = \{0, 1\}^n$).

La notazione $X \leftarrow S$ indica che X è una **variabile aleatoria uniforme** su S , ovvero che $\forall a \in S, \Pr[X = a] = \frac{1}{|S|}$.

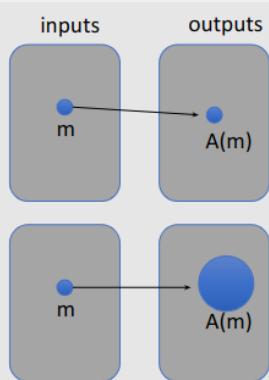
Due variabili aleatorie X e Y che prendono valori in V sono **indipendenti** se

$$\forall a, b \in V, \Pr[X = a \wedge Y = b] = \Pr[X = a] \cdot \Pr[Y = b].$$

Algoritmi randomizzati

Algoritmo deterministico: $y \leftarrow A(m)$

Algoritmo randomizzato: l'output è una variabile aleatoria $y \leftarrow A(m)$



Operazione XOR

Lo **XOR** tra due stringhe in $\{0, 1\}^n$ è la somma bit a bit modulo 2.

La notazione $\text{XOR}(X)$ significa applicare lo XOR a tutti i bit della stringa X .

X	Y	$X \oplus Y$
0	0	0
0	1	1
1	0	1
1	1	0

$$\begin{array}{r} 0110111 \\ 1011010 \\ \hline 1101101 \end{array} \oplus$$

Teorema

Siano X e Y due variabili aleatorie indipendenti tali che X abbia distribuzione uniforme su $\{0, 1\}^n$ e Y abbia distribuzione arbitraria su $\{0, 1\}^n$.

Allora $Z := X \oplus Y$ è una variabile aleatoria uniforme su $\{0, 1\}^n$.

Dimostrazione (per $n = 1$)

$$Pr[Z = 0] =$$

$$Pr[(X, Y) = (0, 0) \vee (X, Y) = (1, 1)] =$$

$$Pr[(X, Y) = (0, 0)] + Pr[(X, Y) = (1, 1)] =$$

$$p_0/2 + p_1/2 = 1/2 \cdot (p_1 + p_2) = 1/2.$$

Perciò $Pr[Z = 1] = 1/2$.

X	$Pr[X]$	Y	$Pr[Y]$	$X \oplus Y$	$Pr[X \oplus Y]$
0	$1/2$	0	p_0	0	$p_0/2$
0	$1/2$	1	p_1	1	$p_1/2$
1	$1/2$	0	p_0	1	$p_0/2$
1	$1/2$	1	p_1	0	$p_1/2$

Paradosso del compleanno

Siano $r_1, \dots, r_n \in U$ variabili aleatorie indipendenti e identicamente distribuite.

Teorema

Quando $n = 1.2 \cdot \sqrt{|U|}$ allora $Pr[\exists i \neq j \mid r_i = r_j] \geq 1/2$.

Sia $U = \{1, 2, \dots, 366\}$

Quando $n = 1.2 \cdot \sqrt{366}$ (circa 23), due persone hanno almeno il 50% di probabilità di avere lo stesso compleanno.

Sia $U = \{0, 1\}^{128}$

Dopo aver prodotto circa 2^{64} messaggi casuali da U , due messaggi sono probabile che saranno uguali.

Aritmetica modulare

Useremo un po' di **Teoria dei Numeri** per costruire protocolli di scambio chiavi, firme digitali, cifratura a chiave pubblica.

Modulo (mod)

Siano a un intero e n un intero positivo.

Definiamo $a \bmod n$ come il resto di a diviso per n .

L'intero n viene detto **modulo**.

Per ogni intero a possiamo scrivere:

$$a = qn + r \quad 0 \leq r \leq n, \quad q = \lfloor a/n \rfloor$$

$$a = \lfloor a/n \rfloor \cdot n + (a \bmod n)$$

Due interi a e b si dicono **congruenti modulo n** se $a \bmod n = b \bmod n$.

Denotiamo la congruenza con $a \equiv b \pmod{n}$.

Proprietà:

1. $a \equiv b \pmod{n}$ se n divide $(a - b)$
2. se $a \equiv b \pmod{n} \implies b \equiv a \pmod{n}$
3. se $a \equiv b \pmod{n}$ e $b \equiv c \pmod{n} \implies a \equiv c \pmod{n}$

Operazioni

Per definizione, l'operatore \pmod{n} mappa tutti gli interi verso l'insieme $\{0, 1, \dots, n-1\}$.

Ne consegue che possiamo effettuare operazioni aritmetiche all'interno di questo insieme → parliamo quindi di **aritmetica modulare**.

Proprietà:

1. $[(a \bmod n) + (b \bmod n)] \bmod n = (a + b) \bmod n$
2. $[(a \bmod n) - (b \bmod n)] \bmod n = (a - b) \bmod n$
3. $[(a \bmod n) \cdot (b \bmod n)] \bmod n = (a \cdot b) \bmod n$

$$11 \bmod 8 = 3, \quad 15 \bmod 8 = 7$$

$$[(11 \bmod 8) + (15 \bmod 8)] \bmod 8 = (11 + 15) \bmod 8 = 26 \bmod 8 = 2$$

$$[(11 \bmod 8) - (15 \bmod 8)] \bmod 8 = (11 - 15) \bmod 8 = -4 \bmod 8 = 4$$

$$[(11 \bmod 8) \cdot (15 \bmod 8)] \bmod 8 = (11 \cdot 15) \bmod 8 = 165 \bmod 8 = 5$$

Sia N un intero positivo, sia p un numero primo.

Definiamo con \mathbb{Z}_N l'insieme degli interi non negativi minori di N .

$$\mathbb{Z}_N = \{0, 1, 2, \dots, n - 1\}$$

All'interno di \mathbb{Z}_N possiamo sommare, sottrarre e moltiplicare modulo N .

Sia $N = 12$, $\mathbb{Z}_{12} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11\}$.

$$-9 + 8 = 17 = 5 \text{ in } \mathbb{Z}_{12}$$

$$-5 - 7 = -2 = 10 \text{ in } \mathbb{Z}_{12}$$

$$-5 \cdot 7 = 35 = 11 \text{ in } \mathbb{Z}_{12}$$

Le proprietà aritmetiche valgono anche in \mathbb{Z}_N .

Calcoliamo le **potenze** come moltiplicazioni consecutive.

Vogliamo calcolare $11^7 \text{ mod } 13$:

$$11^2 = 121 \equiv 4 \pmod{13}$$

$$11^4 = (11^2)^2 \equiv 4^2 \equiv 3 \pmod{13}$$

$$11^7 = 11 \cdot 11^2 \cdot 11^4 = 11 \cdot 4 \cdot 3 = 132 \equiv 2 \pmod{13}$$

GCD

Siano a e b due interi. Il massimo comune divisore, $\text{GCD}(a, b)$, può essere calcolato usando l'algoritmo di Euclide.

Se $\text{GCD}(a, b) = 1 \implies a$ e b sono **primi tra loro**.

Algoritmo di Euclide

1. In partenza dobbiamo sempre avere $a \geq b$, se così non fosse li invertiamo
2. Dividiamo a per b , chiamiamo r il resto
3. Se $r > 0$ sostituiamo a con b e b con r , poi ripartiamo dal punto (2)
4. Se $r \leq 0$ il GCD è il valore finale di b

L'algoritmo di Euclide può essere riformulato sulla base del seguente teorema:

Siano a e b due interi non negativi qualsiasi $\implies \text{GCD}(a, b) = \text{GCD}(b, a \text{ mod } b)$

Algoritmo di Euclide rivisto

```
Euclid(a, b):  
    if (b == 0) then return a;  
    else return Euclid(b, a mod b);
```

Algoritmo di Euclide esteso

Per tutti gli interi a e b esistono due interi x e y tali che $ax + by = \text{GCD}(a, b) = d$. Il valore di x e y si può calcolare tramite l'algoritmo di Euclide esteso (chiaramente x e y avranno segno opposto).

Siano $a = 1759$, $b = 550$.

Calcoliamo inizialmente il GCD:

$$1759 = 550 \cdot 3 + 109 \rightarrow 109 = 1759 - (550 \cdot 3)$$

$$550 = 109 \cdot 5 + 5 \rightarrow 5 = 550 - (109 \cdot 5)$$

$$109 = 5 \cdot 21 + 4 \rightarrow 4 = 109 - (5 \cdot 21)$$

$$5 = 4 \cdot 1 + 1 \rightarrow 1 = 5 - (4 \cdot 1)$$

Ora tramite sostituzioni successive calcoliamo x e y :

$$1 = 5 - 4 \cdot 1 = 5 - (109 - 5 \cdot 21) = 5 \cdot 22 - 109 = (550 - 109 \cdot 5) \cdot 22 - 109 =$$

$$550 \cdot 22 - 109 \cdot 111 = 550 \cdot 22 - (1759 - 550 \cdot 3) \cdot 111 = -1759 \cdot 111 + 550 \cdot 355$$

In conclusione $d = 1$, $x = -111$, $y = 355$.

Inverso modulo N

Lavorando con i numeri razionali, l'inverso moltiplicativo di q è $\frac{1}{q}$.

L'inverso di x in \mathbb{Z}_N è un elemento $y \in \mathbb{Z}_N$ tale che $x \cdot y = 1$.

Denotiamo y come x^{-1} , e lo chiamiamo inverso moltiplicativo di x modulo N .

Lemma

\exists un inverso di x in $\mathbb{Z}_N \iff \text{GCD}(x, N) = 1$

Dimostrazione

$\text{GCD}(x, N) = 1 \implies \exists a, b \mid a \cdot x + b \cdot N = 1 \implies a \cdot x = 1$ in \mathbb{Z}_N .

Pertanto $a = x^{-1}$ in \mathbb{Z}_N

Definiamo con \mathbb{Z}_N^* l'insieme degli elementi invertibili in \mathbb{Z}_N .

$$\mathbb{Z}_N^* = \{ x \in \mathbb{Z}_N \mid \text{GCD}(x, N) = 1 \}$$

$$\mathbb{Z}_{12}^* = \{ 1, 5, 7, 11 \}$$

Per un numero primo p , $\mathbb{Z}_p^* = \mathbb{Z}_p \setminus \{ 0 \} = \{ 1, 2, \dots, p-1 \}$

Per un intero $x \in \mathbb{Z}_N^*$, possiamo calcolare x^{-1} usando l'algoritmo di Euclide esteso.

Teorema di Fermat

Teorema

Sia p un numero primo, x un intero positivo non divisibile per p .

Allora $\forall x \in \mathbb{Z}_p^* \quad x^{p-1} = 1$ in \mathbb{Z}_p .

Questo è un altro modo per calcolare l'inverso moltiplicativo, però è meno efficiente dell'algoritmo di Euclide esteso:

$$x \in \mathbb{Z}_p^* \implies x \cdot x^{p-2} = 1 \implies x^{-1} = x^{p-2} \text{ in } \mathbb{Z}_p.$$

Il teorema di Fermat torna utile per generare numeri primi grandi in modo casuale.

Supponiamo di avere a disposizione 1024 bit.

1. Scegliere un intero casuale $p \in [2^{1024}, 2^{1025} - 1]$
2. Se $2^{p-1} = 1$ in \mathbb{Z}_p allora restituire p , altrimenti ripartire da (1).

Teorema di Eulero

La funzione di Eulero, denotata con $\phi(N)$, è definita come il numero di interi positivi minori di N il cui GCD con N è 1.

$$\phi(N) = |\mathbb{Z}_N^*|$$

$$\phi(1) = 1$$

Sia p un numero primo. $\phi(p) = p - 1$

$$\phi(12) = |\{1, 5, 7, 11\}| = 4$$

Supponiamo di avere 2 numeri primi p e q ($p \neq q$).

$$\phi(p \cdot q) = \phi(p) \cdot \phi(q) = (p - 1) \cdot (q - 1)$$

Teorema (generalizzazione di Fermat)

Siano x e N primi tra loro.

Allora $\forall x \in \mathbb{Z}_N^* \quad x^{\phi(N)} = 1$ in \mathbb{Z}_N .

Crittografia

Introduzione

La **crittografia** è l'arte di usare la matematica per oscurare il significato dei dati applicando trasformazioni impraticabili o impossibili da invertire senza la conoscenza di una chiave (dal greco "scrittura nascosta").

La **crittoanalisi** è l'arte di decifrare codici segreti, recuperando il testo in chiaro dal testo cifrato quando la chiave è sconosciuta.

La **crittologia** combina crittografia e crittoanalisi.

La crittografia si trova praticamente ovunque: nelle comunicazioni sicure, nei file protetti su disco, nell'autenticazione... ovunque servano metodi per memorizzare, elaborare e trasmettere informazioni in maniera sicura in presenza di agenti ostili.

Concetti base

Definire una **comunicazione sicura** vuol dire:

- no eavesdropping, ovvero nessuna intercettazione (confidentiality)
- no tampering, ovvero nessuna alterazione del contenuto del messaggio (integrity)

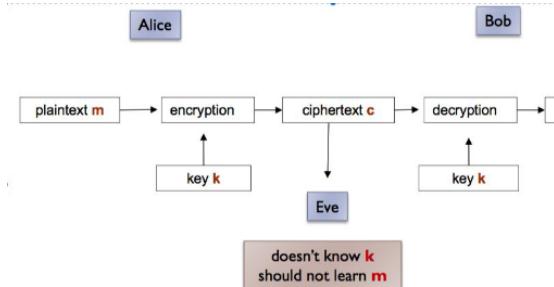
L'obiettivo è impedire ad entità terze di capire cosa viene inviato tra gli interlocutori. Questo si può ottenere con approcci diversi:

- **steganografia** → nascondere l'esistenza di un messaggio (renderlo non visibile)
- **crittografia** → nascondere il significato del messaggio

Terminologia

Alice e Bob sono gli interlocutori, Eve è l'attaccante. Il messaggio in chiaro (o plaintext) è m , il messaggio cifrato (o ciphertext) è c , k è la chiave.

Supponendo che Eve non sappia k , non deve poter risalire a m da c .



Scopi e obiettivi

Lo scopo principale della crittografia è garantire una comunicazione sicura tra le parti che scambiano messaggi su un supporto insicuro.

Obiettivi di sicurezza di base:

- **Privacy** (riservatezza): solo il destinatario previsto può vedere la comunicazione
- **Autenticità**: la comunicazione è generata dal presunto mittente
- **Integrità**: nessuna modifica non autorizzata ai messaggi
- **Non ripudiabilità**: nessuna esclusione di responsabilità, ovvero non si può negare di aver mandato un messaggio in quanto si può risalire al mittente con certezza

Per fornire un protocollo di crittografia è necessario comprendere:

Chi sono le parti coinvolte ed il contesto in cui agiscono?

Quali sono gli obiettivi di sicurezza da garantire?

Cos'è la base di elaborazione attendibile, ovvero cosa è attendibile?

Quali sono le capacità degli attaccanti?

Molto noto e importante è il **principio di Kerckhoffs**: la sicurezza di un protocollo dovrebbe basarsi solo sulla segretezza delle chiavi, mentre gli algoritmi dei protocolli dovrebbero essere resi pubblici.

La security by obscurity non funziona.

Minacce dell'attaccante

Assumiamo che l'attaccante abbia una conoscenza totale dell'algoritmo crittografico.

L'interazione con l'algoritmo ed i messaggi può essere:

- **passiva**, l'attaccante si limita ad osservare ciò che passa sul canale, cercando di decifrare i messaggi. Minaccia solo la confidentiality.
- **attiva**, l'attaccante osserva, modifica, inietta o elimina i messaggi. Minaccia confidentiality, integrity e authenticity.

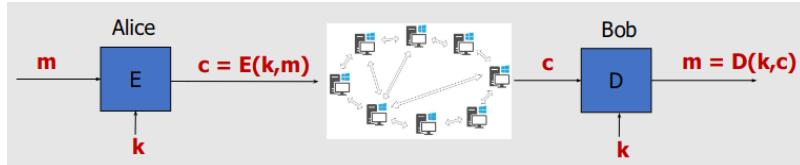
Tipi di attacco al cripto-sistema:

- **Ciphertext-only attack (CA)** = l'attaccante vede solo i messaggi cifrati
- **Chosen-plaintext attack (CPA)** = l'attaccante può scegliere un certo numero di messaggi e ottenerne i testi cifrati
- **Chosen-ciphertext attack (CCA)** = l'attaccante può scegliere un certo numero di testi cifrati e ottenerne i testi in chiaro

Sia gli attacchi CPA che CCA possono essere adattivi: le scelte possono cambiare in base ai risultati dei tentativi precedenti.

Assumiamo che l'attaccante abbia risorse illimitate (di archiviazione e/o elaborazione), in quantità finita. Per calcolare, in genere un tempo di esecuzione polinomiale.

Crittografia simmetrica



Ci troviamo in uno scenario di confidentiality. Altri scenari sono possibili utilizzando la chiave in maniera diversa (es. Message Authentication Code, MAC, per authenticity e integrity).

In questo caso la chiave k è segreta e condivisa tra Alice e Bob.

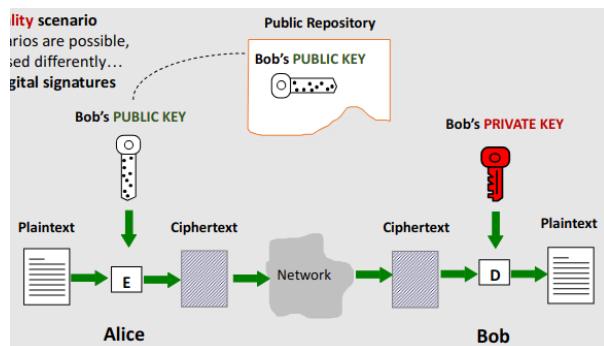
Gli algoritmi sono conosciuti pubblicamente, mai usare un cifrario proprietario.

Casi d'uso

Single-use key (o one-time key): la chiave è usata per cifrare un solo messaggio (es. mail).

Multi-use key (o many-time key): la stessa chiave è usata per cifrare più messaggi (es. file).

Crittografia asimmetrica



Ci troviamo in uno scenario di confidentiality. Altri scenari sono possibili utilizzando la chiave in maniera diversa (es. firma digitale).

Non c'è più il concetto di chiave segreta condivisa. Ci sono due chiavi: la chiave privata deve rimanere segreta, quella pubblica è nota a tutti (in directory ben precise).

Alcune applicazioni

- Comunicazione sicura: condividere una chiave segreta (momento critico su cui si basa l'intero processo) e poi scambiare messaggi in confidenza.
- Comunicazione tra dispositivi mobili
- Firme digitali
- Comunicazione anonima

La crittografia è una scienza rigorosa, che richiede tre passi:

1. Specificare precisamente il modello di minaccia
2. Proporre una costruzione
3. Dimostrare che rompere la costruzione secondo il modello di minaccia risolverebbe un problema difficile sottostante

Storia della crittografia

Vediamo velocemente l'evoluzione dei cifrari a sostituzione (substitution ciphers) nel tempo.

Cifrario di Cesare (shift cipher)

Il più famoso è il **cifrario di Cesare**, in cui la chiave k indica di quante posizioni le lettere dell'alfabeto sono spostate. Per cifrare, ogni lettera del plaintext viene sostituita con la k -esima lettera seguente tramite uno shift a dx, per decifrare si fa lo shift a sx di k posizioni.

Da un punto di vista matematico l'insieme dei possibili plaintext è dato dalle possibili parole costruibili sull'alfabeto, lo spazio delle chiavi k è $\{0, \dots, 25\}$.

Siano $P = C = K = Z_{26}$. Per $k \in K = \{0, \dots, 25\}$:

- $e_k(p) = p + k \bmod 26 \quad p \in P$
- $d_k(c) = c - k \bmod 26 \quad c \in C$

Per un attaccante è molto facile trovare la chiave k : siccome lo spazio delle chiavi è ristretto, basta fare una ricerca esaustiva (crittoanalisi brute-force). Dal momento in cui si trova la chiave, decifrare è estremamente semplice.

Mono-alphabetic substitution cipher

Un altro esempio di cifrario a sostituzione è quello **mono-alfabetico**.

Lo spazio delle chiavi K è dato da tutte le possibili permutazioni dell'alfabeto, ovvero un numero di chiavi pari a $26!$.

Cifrare (data una permutazione π): ogni lettera X del plaintext p è sostituita con $\pi(X)$.

Decifrare (data una chiave π): ogni lettera Y nel ciphertext c è rimpiazzata da $\pi^{-1}(Y)$.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	
π	B	A	D	C	Z	H	W	Y	G	O	Q	X	S	V	T	R	N	M	S	K	J	I	P	E	F	U

BECAUSE → AZDBJSZ

I cifrari a sostituzione sono vulnerabili agli attacchi di analisi della frequenza (frequency analysis).

Ogni lingua ha determinate caratteristiche: frequenza di lettere o gruppi di lettere, e i cifrari a sostituzione preservano tali caratteristiche.

Nel ciphertext viene conteggiato il numero di caratteri o combinazioni per determinarne la frequenza di utilizzo. Il testo viene poi esaminato per individuare schemi, serie ripetute e combinazioni comuni. Si sostituiscono i caratteri del testo cifrato con possibili equivalenti in chiaro, utilizzando le caratteristiche del linguaggio noto.

UKBYBIPOUZBCUFEEBORUKBYBHOBBRFESPVKBWFOPERVNBCVBZPRUBOOPERVNBCVBPCYYFV UFOFEIKNWFRFIKINUPWRFIPOUNVNIPUBRNCKUBEFWWFDNCXCYBOHOPYXPUBNCOYRN VNINWCPOIOFHOPZRVFZIXUBORJRUZRBCNCCBONCHRZSFVNVRJURZRPCYZPKUBZPUN VPWPVCYFZIKUPUNFCPWRVNBCVBRPYNNUNFCPWWJUKBYBIPOUZBCUPOUNVNIPUBRNCHO PYXPUBNCUBOYNRVIWNCPOIOFHOPZRNCRVNBCCUNENVFZIXUNCHPCYVFZIXUPUNFCPWZ PUKBZPUNVR		
B 36 => E	NC 11 => IN	UKB 6 => THE
N 34	PU 10 => AT	RVN 6
U 33 => T	UB 10	FZI 4
P 32 => A	UN 9	trigrams
C 26	diagrams	

Affine cipher

Un altro caso speciale di cifrario a sostituzione è l'**Affine cipher**, in cui la funzione di cifratura è della forma $e(x) = ax + b \pmod{26}$ $a, b \in \mathbb{Z}_{26}$.

Per $a = 1$ otteniamo il cifrario di Cesare (shift cipher).

Perché la decryption abbia un'unica soluzione, la funzione di cifratura deve essere iniettiva, ovvero $\forall y \in \mathbb{Z}_{26}$ la congruenza $ax \equiv y \pmod{26}$ deve avere un'unica soluzione per x . Questo è garantito per ogni y se $\text{GCD}(a, 26) = 1$, ovvero se a e 26 sono primi tra loro. Se così non fosse, avremmo lettere diverse nel plaintext che diventano uguali nel ciphertext.

Vigenère cipher (poly-alphabetic cipher)

La principale debolezza dei cifrari a sostituzione mono-alfabetica è che ogni lettera nel ciphertext corrisponde sempre alla stessa singola lettera nel plaintext. Questo problema viene affrontato con l'**algoritmo di Vigenère**.

Sia m un intero positivo.

$$P = C = \mathbb{Z}_{26}^m \text{ e } k = (k_1, \dots, k_m).$$

Assumiamo che il plaintext sia composto da m simboli. Per cifrare sommiamo in mod26 l' i -esimo simbolo con l' i -esimo valore della chiave, per decifrare facciamo lo stesso sottraendo in mod26.

- Encryption $e_k(p_1, \dots, p_m) = (p_1 + k_1, \dots, p_m + k_m) \pmod{26}$
- Decryption $d_k(c_1, \dots, c_m) = (c_1 - k_1, \dots, c_m - k_m) \pmod{26}$

Per effettuare le operazioni più rapidamente si costruisce una tabella con lettere plain e lettere chiave, e nelle celle la lettera cifrata.

La lunghezza della chiave e quindi della parte di messaggio considerata è $m = 6$.

$k =$	C R Y P T O	O C R Y P T O	C O C R Y P T	(+ mod 26)
$m =$	W H A T A N I C E D A Y T O D A Y			
$c =$	Y Y Y I T B K T C S T M V F B P R			

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

Con la sostituzione poli-alfabetica abbiamo una chiave composta da m sotto-chiavi, quindi non succede più che una lettera nel plaintext corrisponda sempre alla stessa lettera nel ciphertext. C'è comunque una limitazione data da m .

L'algoritmo di Vigenère è un insieme di m shift cipher. Il fatto che una lettera nel ciphertext corrisponda a più di una lettera nel plaintext rende l'analisi della frequenza difficoltosa, ma non impossibile.

L'attaccante sa che, ragionando a blocchi, la lettera in una posizione cifra le lettere nella stessa posizione, quindi ricadiamo nel problema del mono-alfabetismo (questa volta derivante dalla chiave).

È infatti possibile indovinare la lunghezza m della chiave, dividere il ciphertext in blocchi di m lettere e poi usare l'analisi della frequenza sui singoli blocchi.

$$c[i] = m[i] + k[i] \implies k[i] = c[i] - m[i].$$

Macchine rotanti (rotor machines)

L'idea è fare più round di sostituzione. La cifratura quindi consiste nel mappare una lettera numerose volte. Macchine elettriche/meccaniche, composte da cilindri che girano, possono automatizzare questo processo: la lettera premuta viene mappata in un'altra lettera passando per la rotor machine.

La più famosa è la macchina Enigma, con 3 cilindri (poi 5 in seguito).

Cifrari a flusso (stream ciphers)

Un cifrario simmetrico definito su (K, P, C) è una coppia di algoritmi efficienti (E, D) dove

- $E : K \times P \rightarrow C$, da una chiave ed un plaintext produce un ciphertext
 - $D : K \times C \rightarrow P$, da una chiave ed un ciphertext produce un plaintext
- tali che $\forall m \in P, \forall k \in K \quad D(k, E(k, m)) = m$.

Spesso E è randomizzato, in modo da ottenere ciphertext diversi anche usando coppie (k, m) uguali, mentre D è sempre deterministico, in quanto deve garantire che facendo decryption su c con la chiave corretta si ottenga esattamente p . La chiave k appartiene allo spazio delle chiavi K ed è condivisa segretamente tra mittente e destinatario.

One-Time Pad

Primo esempio di cifrario *sicuro*.

- $K = P = C$, sono stringhe binarie di lunghezza n
- $E(k, m) = k \oplus m = c$, cifratura con un semplice XOR (chiave e messaggio lunghi uguali)
- $D(k, c) = k \oplus c$, si decifra rifacendo lo XOR

La chiave k è generata casualmente e viene usata una sola volta. OTP è un cifrario simmetrico:

$$D(k, E(k, m)) = D(k, k \oplus m) = k \oplus (k \oplus m) = (k \oplus k) \oplus m = 0 \oplus m = m$$

Pro

- encryption e decryption molto veloci

Contro

- chiavi molto lunghe (quanto il messaggio): se avessimo un modo per condividere segretamente la chiave così lunga, tanto vale mandare direttamente il messaggio

Cifrario sicuro

Capacità dell'attaccante: ciphertext only attack.

Secondo Shannon un cifrario è sicuro se il ciphertext non rivela nessuna informazione sul messaggio in chiaro.

Information Theoretic Security (Shannon 1949)

Un cifrario (E, D) su (K, P, C) ha la **perfect secrecy** se:

$\forall m_0, m_1 \in P$, con $\text{len}(m_0) = \text{len}(m_1)$ e $\forall c \in C$, $\Pr[E(k, m_0) = c] = \Pr[E(k, m_1) = c]$,
dove k è uniforme in K .

Ovvero se la probabilità che un messaggio venga cifrato in un determinato ciphertext non dipende dal messaggio stesso. Quindi dal ciphertext non si può ricavare nessuna informazione sul plaintext. Non vengono fatte assunzioni computazionali sull'attaccante, ecco perché questa viene anche chiamata *unconditional security* o *perfect security*.

Questo impedisce i ciphertext only attack, però altri attacchi sono possibili...

OTP, godendo della perfect secrecy, è sicuro. Dimostriamolo:

$$\forall m, c \Pr_k[E(k, m) = c] = \frac{\#\text{keys } k \in K \mid E(k, m) = c}{|K|}$$

Quindi se $\forall m, c$ quello che c'è al numeratore è costante \implies il cifrario gode della perfect secrecy. Siccome per OTP esiste una sola chiave per mappare da m a c , la probabilità è sempre $1/|K|$, quindi OTP ha la perfect secrecy.

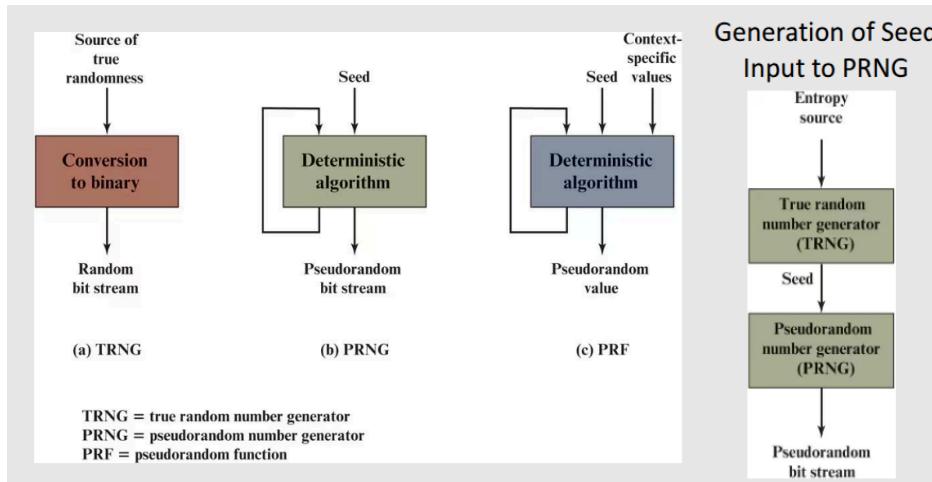
Resta il problema della lunghezza delle chiavi: non è pratico generarle.

Un teorema afferma che la perfect secrecy implica avere chiavi di lunghezza uguale o maggiore alla lunghezza del messaggio da cifrare.

Esiste un'approssimazione per rendere OTP più flessibile ed utilizzabile, ovvero ridurre la lunghezza della chiave OTP ed utilizzarla come seed per generare altre chiavi.

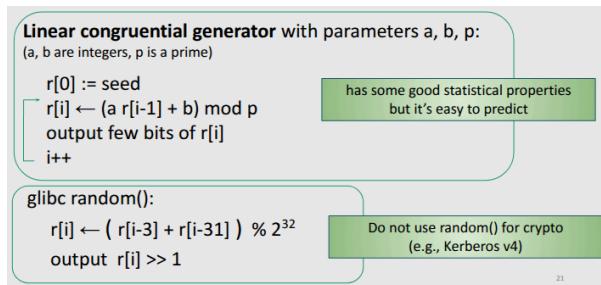
Generatori pseudo-random

Un **generatore di numeri pseudo-random (PRNG)**, Pseudo-Random Number Generator) è un algoritmo utilizzato per generare una sequenza di numeri apparentemente casuale. Tale sequenza in realtà è deterministica, ovvero è prodotta a partire da una specifica formula che riceve in input un **seed**. Un **generatore di numeri veramente-random (TRNG)**, True-Random Number Generator) possiede una sorgente di casualità, che viene successivamente convertita in una sequenza binaria.



La sequenza di bit prodotta da un PRNG sarebbe bene che fosse il più vicino possibile a quelle prodotte da un TRNG. Per ottenere ciò, deve soddisfare i requisiti di casualità e di non predicitività. Essendo il PRNG deterministico, per poterlo applicare alla crittografia il seed deve essere assolutamente casuale (tipicamente generato da un TRNG).

PRNG non sicuri (deboli)



PRNG sicuro

Il generatore **Blum Blum Shub (BBS)** è un approccio popolare per generare numeri pseudo-casuali.

Scegliere due numeri primi "grandi" p e q , tali che abbiano resto 3 quando divisi per 4 (ad esempio 7 e 11). Sia $n = p \cdot q$. Decidere un numero casuale s , tale che s e n siano primi tra loro.

Il BBS genera una sequenza di bit B_i attuando il seguente algoritmo:

$$X_0 = s^2 \bmod n$$

for $i = 1$ to ∞

$$X_i = (X_{i-1})^2 \bmod n$$

$$B_i = X_i \bmod 2$$

Ad ogni iterazione viene preso il bit meno significativo.

Il BBS è definito come un **cryptographically secure pseudo-random bit generator (CSPRNG)**, ovvero un generatore che supera il "next-bit test".

Next-bit test: dati i primi k bit dell'output, non esiste un algoritmo che in tempo polinomiale possa decretare se il prossimo bit sarà 0 o 1 con una probabilità maggiore del 50%.

La sicurezza del BBS si basa sulla difficoltà della fattorizzazione di n , ovvero determinare i fattori primi p e q che lo compongono.

Stream ciphers con PRG

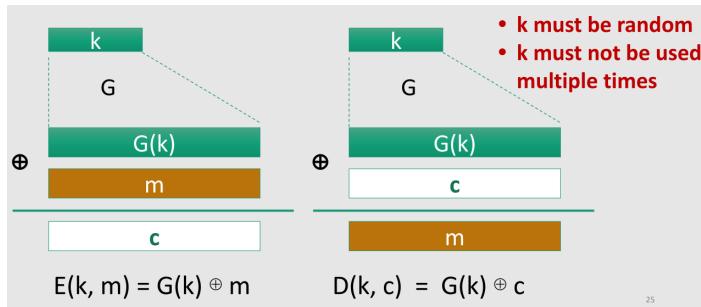
L'idea di base è utilizzare un chiave pseudo-casuale invece di una completamente casuale.

Utilizziamo un **generatore pseudo-casuale (PRG**, Pseudo-Random Generator), ovvero una funzione G che a partire da un seed genera una chiave:

$$G : \{0, 1\}^s \rightarrow \{0, 1\}^n, \text{ con } n \text{ molto maggiore di } s.$$

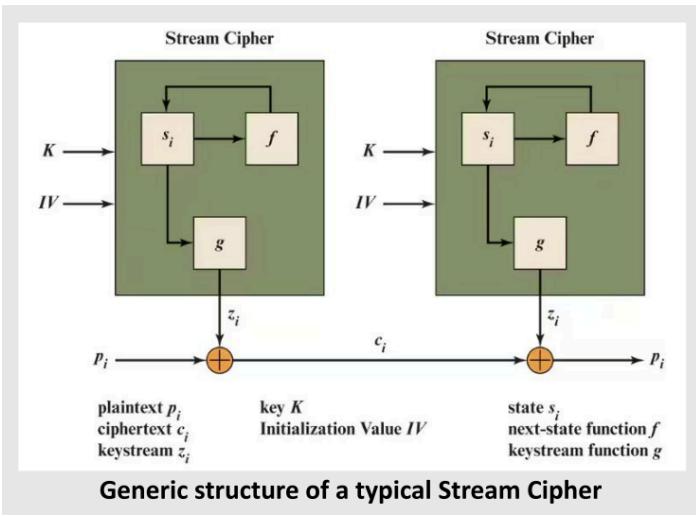
Ciò è calcolabile efficientemente da un algoritmo deterministico.

Perciò da un seed k generiamo una chiave $G(k)$, usata per cifrare e decifrare il messaggio m . Il seed k deve essere casuale ed utilizzato una volta sola.



Struttura generica

La sequenza della chiave z_i viene posta in XOR bitwise con la sequenza del plaintext p_i .



Siano $z_i = 01101100$ e $p_i = 11001100$.

Il ciphertext risultante sarà $c_i = 10100000$.

Uno stream cipher qualunque non può godere della perfect secrecy, in quanto la chiave è più corta del messaggio \implies serve una definizione di sicurezza differente.

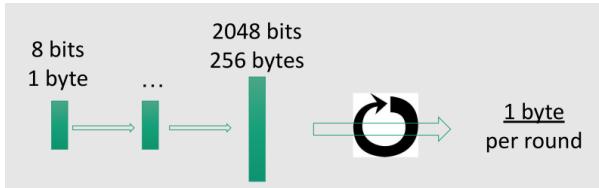
La sicurezza di uno stream cipher dipende fortemente dal PRNG: con un PRNG progettato adeguatamente, uno stream cipher può essere sicuro quanto un block cipher avente lunghezza delle chiavi simile.

Casi d'uso reali

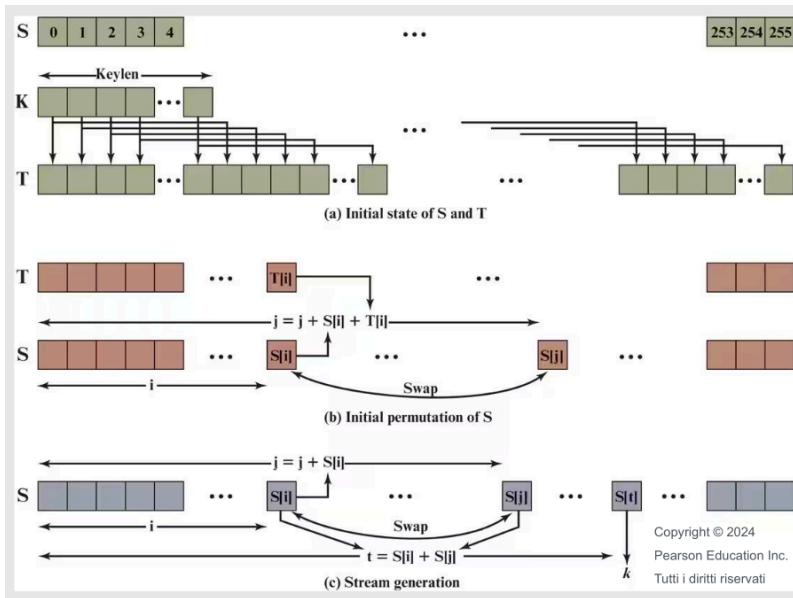
RC4

Rivest Cipher 4, progettato nel 1987, viene utilizzato in:

- SSL/TLS, standard per HTTPS
- WEP e WPA, protocolli di sicurezza per comunicazioni wireless



L'algoritmo RC4 è semplice. Una chiave di lunghezza variabile (da 8 a 2048 bit) è usata per inizializzare un vettore di stato S di 256 byte. In ogni momento, S contiene una permutazione di tutti i numeri a 8 bit da 0 a 255. Per cifrare e decifrare, un byte k è generato da S selezionando una delle 255 voci in modo sistematico. A seguito della generazione di un k , le voci in S vengono permutate nuovamente.



Punti di debolezza

1. Bias nell'output iniziale: supponiamo che l'algoritmo di setup RC4 sia perfetto e generi una permutazione uniforme dall'insieme di tutte le $256!$ permutazioni. Si può dimostrare che l'output di RC4 è vittima di bias, in quanto la probabilità che il secondo byte sia 0 è $\frac{1}{128}$.
2. È stato dimostrato che la probabilità di (0,0) è $\frac{1}{256^2} + \frac{1}{256^3}$.
3. Attacchi con chiave correlata (attacco al WEP).

Stream cipher moderni: eStream

Il PRNG utilizza il seed insieme ad un nonce per generare la chiave.

$$G : \{0, 1\}^s \times r \rightarrow \{0, 1\}^n, \text{ con } n \text{ molto maggiore di } s.$$

Il **nonce** è un valore non ripetuto per una determinata chiave, la quale è una coppia (k, r) che viene usata una volta soltanto, perciò si può riutilizzare la chiave k fintanto che il nonce r cambia. Ricapitolando: $E(k, m, r) = m \oplus PRG(k, r)$

Attacchi ad OTP e stream cipher

Abbiamo quindi degli stream ciphers che rendono OTP pratico utilizzando un PRG: tramite un seed K generano una chiave $G(K)$.

Two Time Pad

Non è sicuro! Mai utilizzare la chiave più di una volta.

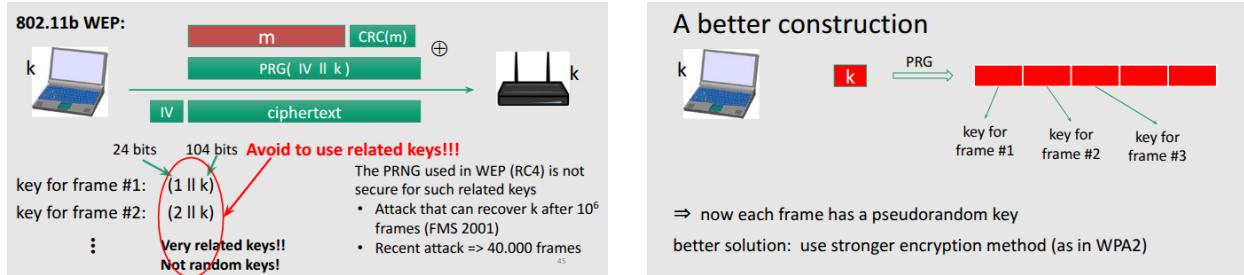
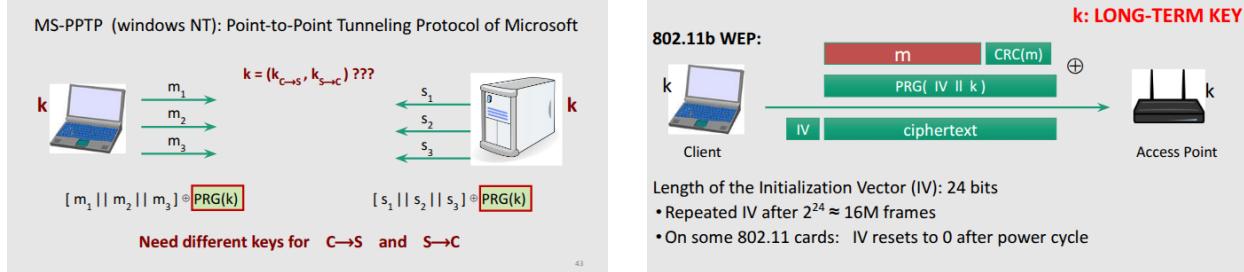
Data una chiave k , cifriamo m_1 e m_2 con lo stesso $PRG(k)$, ottenendo c_1 e c_2 .

Per l'attaccante è sufficiente calcolare $c_1 \oplus c_2$ per ottenere $m_1 \oplus m_2$. C'è abbastanza ridondanza nella codifica inglese e ASCII che $m_1 \oplus m_2 \rightarrow m_1, m_2$.

WEP - RC4

Per il traffico di rete negoziare una nuova chiave ad ogni sessione:

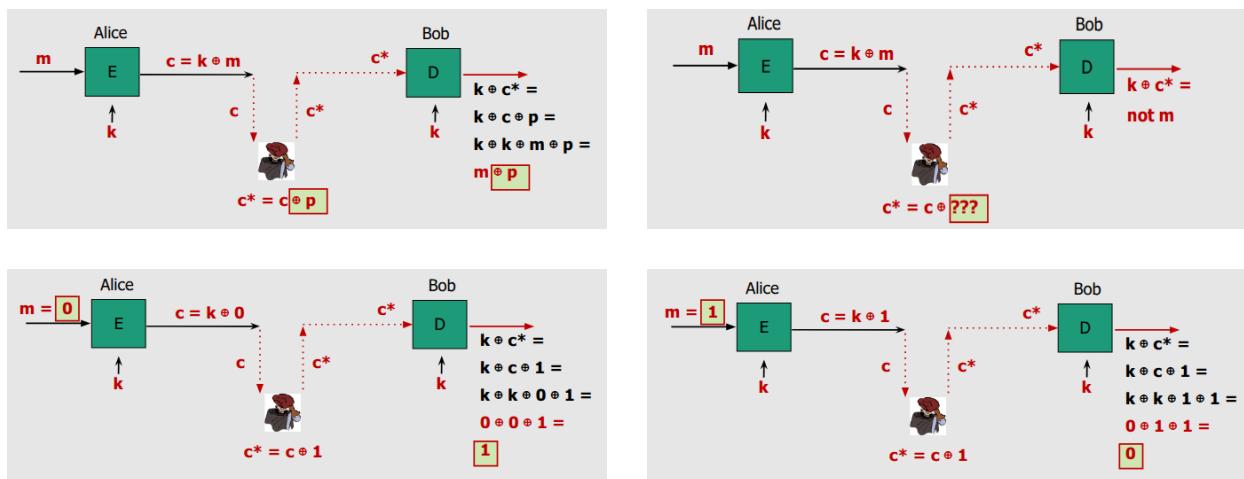
- una chiave (o sottochiave) per la trasmissione da client a server
- una chiave (o sottochiave) per la trasmissione da server a client



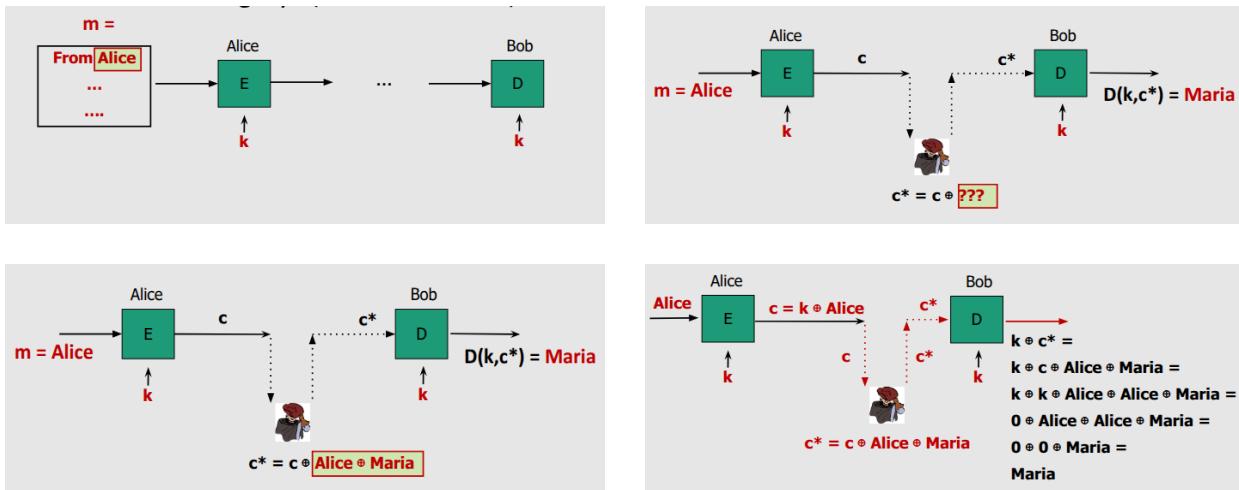
Nessuna garanzia di integrità

OTP è malleabile: cambiamenti al ciphertext non vengono rilevati ed hanno un impatto prevedibile sul plaintext.

Alice deve rispondere 0 o 1 a Bob. Cifrando con OTP, l'attaccante non può sapere il messaggio di Alice, però può invertirlo a prescindere applicando $\oplus 1$ al ciphertext.



Consideriamo un altro caso: l'attaccante vuole trasformare "Alice" in "Maria".

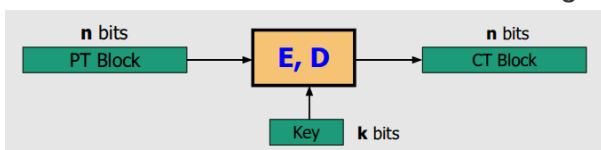


In conclusione, quand'è che un PRNG è sicuro?

Affermare che un PRNG è sicuro equivale a dire che non è predicibile.

Cifrari a blocco (block ciphers)

I cifrari a blocco sono algoritmi di crittografia simmetrica che operano su porzioni di plaintext di lunghezza fissa, trasformandole in blocchi di ciphertext. Ciò avviene tramite una serie di trasformazioni basate su una chiave segreta.

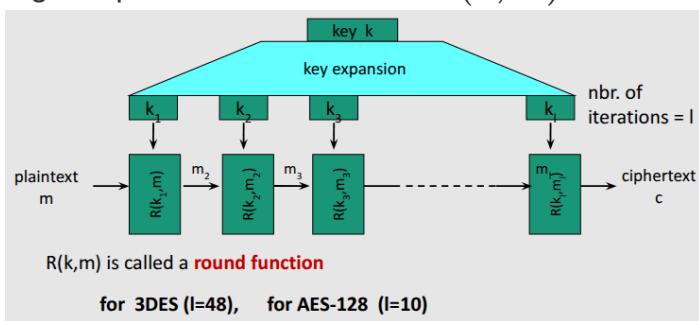


Esempi di cifrari a blocchi:

- **DES (Data Encryption Standard)**: blocchi di 64 bit, chiave di 56 bit
- **3DES (Triple DES)**: blocchi di 64 bit, chiave di 168 bit
- **AES (Advanced Encryption Standard)**: blocchi di 128 bit, chiave di 128, 192 o 256 bit

Tipicamente, i cifrari a blocco applicano iterativamente più **round** di trasformazioni crittografiche.

Ogni round utilizza una porzione della chiave ed esegue una serie di operazioni matematiche e logiche per trasformare i dati. $R(k, m)$ viene detta **round function**.

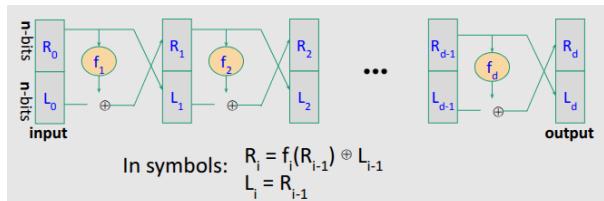


DES (Data Encryption Standard)

Nel 1976 DES diventa lo standard federale, ma nel 1997 viene violato tramite un attacco brute-force a causa delle chiavi troppo corte. Nel 2000 viene sostituito da AES.

Rete di Feistel (Feistel network)

DES è basato sulla **rete di Feistel**, che consente di costruire una funzione crittografica invertibile, anche con funzioni interne non invertibili.

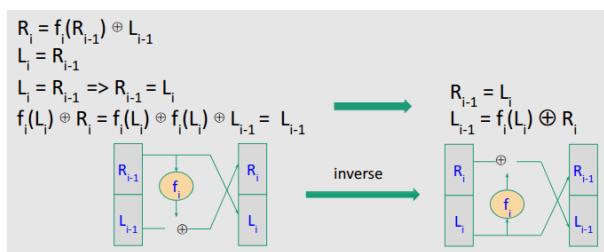


Teorema:

Siano $f_1, \dots, f_d : \{0, 1\}^n \rightarrow \{0, 1\}^n$ funzioni arbitrarie, non necessariamente invertibili.

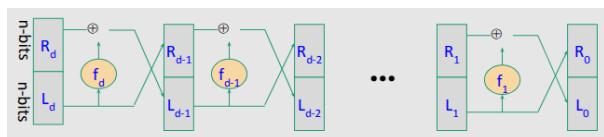
Allora la rete di Feistel $F : \{0, 1\}^{2n} \rightarrow \{0, 1\}^{2n}$ è invertibile.

Dimostrazione: costruiamo l'inverso:



Circuito per decifrare

L'inversione è praticamente lo stesso circuito, con le funzioni f applicate in ordine contrario, ovvero f_d, \dots, f_1 .

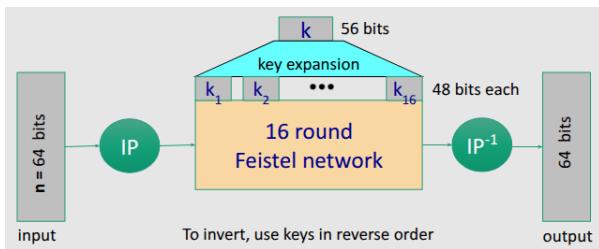


Questo è un metodo generale per costruire funzioni invertibili (cifrari a blocchi) a partire da funzioni arbitrarie. Viene usato in molti block cipher (non in AES).

Struttura di DES

DES utilizza una rete di Feistel a 16 round.

$f_1, \dots, f_{16} : \{0, 1\}^{32} \rightarrow \{0, 1\}^{32}$, $f_i(x) = F(k_i, x)$, dove k_i è una sotto-chiave, ovvero una certa selezione permutata di bit da k .

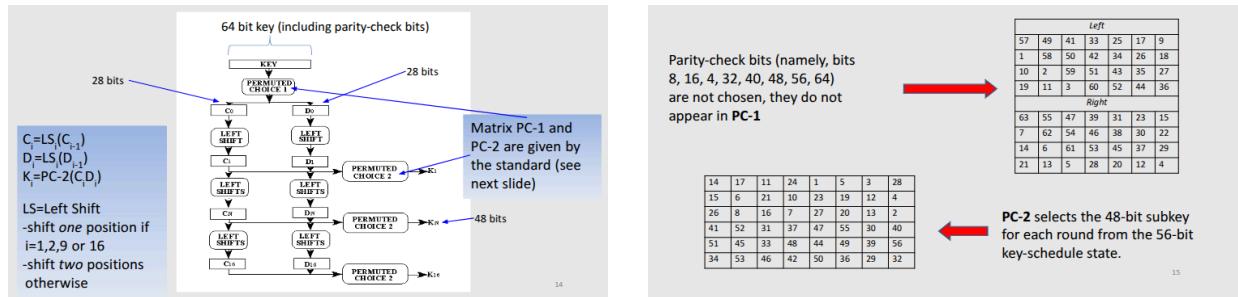


Chiavi e sotto-chiavi

La lunghezza delle chiavi usate da DES è di 64 bit, ovvero 8 byte. In ogni byte, l'ottavo bit è un bit di parità, dato dallo XOR dei precedenti 7 bit.

Generazione delle chiavi ($k_1 - k_{16}$):

DES usa 16 sotto-chiavi di 48 bit ciascuna, generate da una chiave "principale" di 56 bit (64 contando i bit di parità).



Esistono alcune chiavi che sono dette "deboli", in quanto generano la stessa sotto-chiave in più di un round, riducendo la complessità e sicurezza del cifrario. Tali chiavi possono essere evitate in fase di generazione.

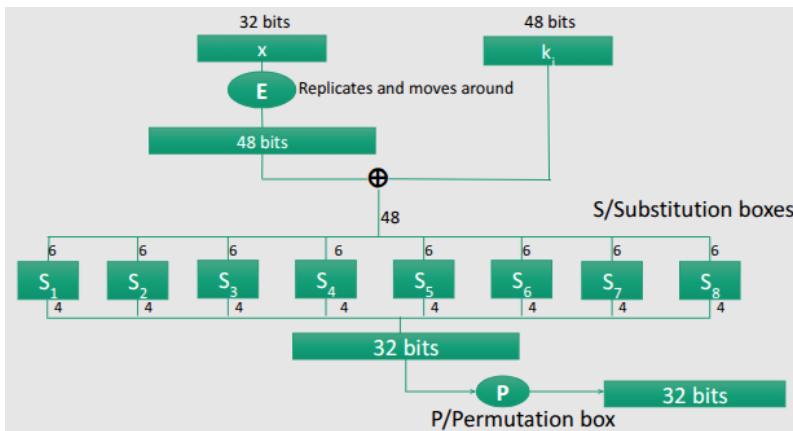
Le chiavi deboli:

- 01010101 01010101
- FEEFEFEFE FEEFEFEFE
- E0E0E0E0 F1F1F1F1
- 1F1F1F1F 0E0E0E0E

Trasformazioni

Il plaintext viene diviso in due metà e sottoposto a 16 round di trasformazioni:

- Espansione e permutazione del blocco
- XOR con una sotto-chiave
- Passaggio attraverso una funzione non lineare (S-Box, implementata come una look-up table)
- Permutazione finale e riunione delle due metà



Initial Permutation (IP)

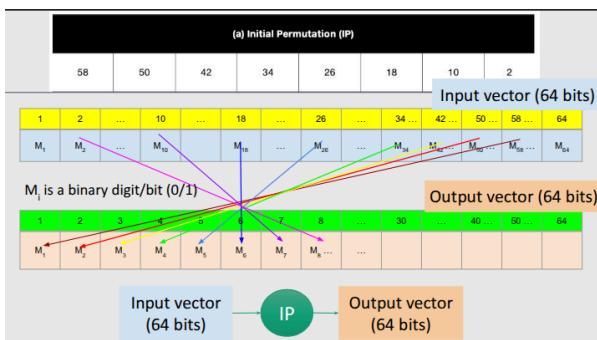
(a) Initial Permutation (IP)

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

Inverse Initial Permutation (IP^{-1})

(b) Inverse Initial Permutation (IP^{-1})

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25



Expansion Permutation (E) and Permutation Function (P)

(c) Expansion Permutation (E)

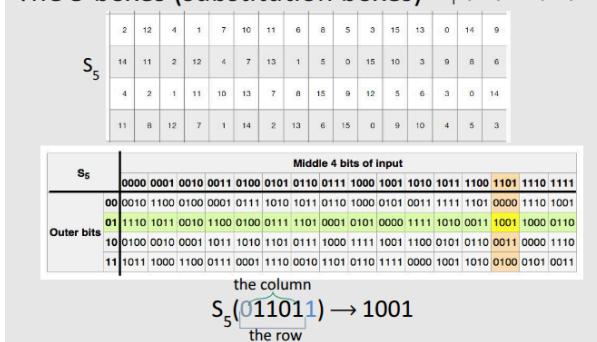
32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

(d) Permutation Function (P)

16	7	23	29	12	26	17
1	15	22	28	5	18	31
2	8	24	14	32	27	3
19	13	30	6	22	11	4

Scegliere le S-box e le P-box casualmente porterebbe ad avere un cifrario non sicuro. Per effettuare la scelta vengono usate numerose regole.

The S-boxes (substitution boxes) $S_i: \{0,1\}^6 \rightarrow \{0,1\}^4$



Effetto valanga

Una proprietà desiderabile per qualsiasi algoritmo di crittografia è che una piccola modifica del plaintext o della chiave produca una modifica significativa del ciphertext.

In particolare, la modifica di un bit nel plaintext o nella chiave dovrebbe produrre una modifica di molti bit nel ciphertext. Questo fenomeno è chiamato **effetto valanga**.

Se la modifica nel ciphertext fosse minima, potrebbe fornire un modo per ridurre la dimensione dello spazio delle chiavi o del plaintext da ricercare.

Attacchi di ricerca esaustiva

La lunghezza della chiave usata da DES (56 bit) è troppo corta e rende tale cifrario vulnerabile agli attacchi di brute-force.

Key Size (bits)	Cipher	Number of Alternative Keys	Time Required at 10^9 Decryptions/s	Time Required at 10^{13} Decryptions/s
56	DES	$2^{56} \approx 7.2 \times 10^{16}$	$2^{55} \text{ ns} = 1.125 \text{ years}$	1 hour
128	AES	$2^{128} \approx 3.4 \times 10^{38}$	$2^{127} \text{ ns} = 5.3 \times 10^{21} \text{ years}$	$5.3 \times 10^{17} \text{ years}$
168	Triple DES	$2^{168} \approx 3.7 \times 10^{50}$	$2^{167} \text{ ns} = 5.8 \times 10^{33} \text{ years}$	$5.8 \times 10^{29} \text{ years}$
192	AES	$2^{192} \approx 6.3 \times 10^{57}$	$2^{191} \text{ ns} = 9.8 \times 10^{40} \text{ years}$	$9.8 \times 10^{36} \text{ years}$
256	AES	$2^{256} \approx 1.2 \times 10^{77}$	$2^{255} \text{ ns} = 1.8 \times 10^{60} \text{ years}$	$1.8 \times 10^{56} \text{ years}$
26 characters (permutation)	Monoalphabetic	$26! = 4 \times 10^{26}$	$2 \times 10^{26} \text{ ns} = 6.3 \times 10^9 \text{ years}$	$6.3 \times 10^6 \text{ years}$

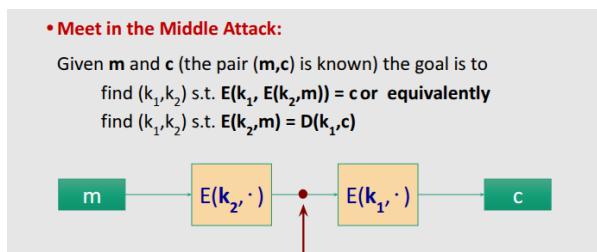
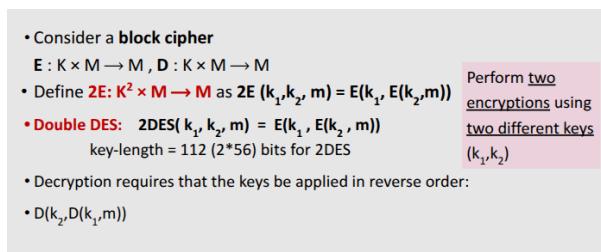
Per cercare di porre rimedio a questo problema sono state pensate diverse soluzioni:

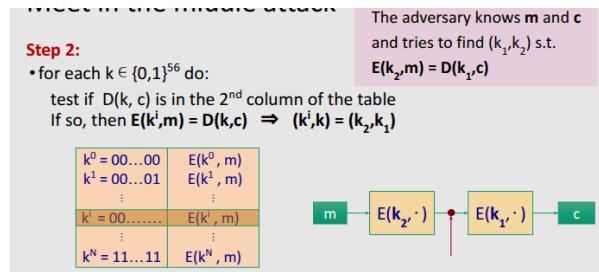
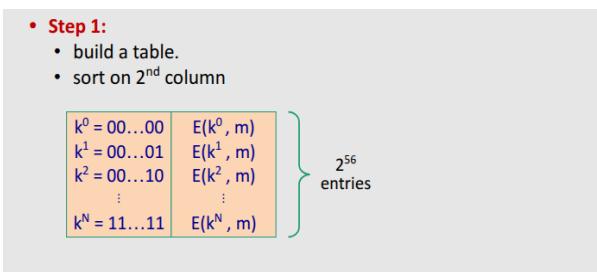
- Cifrare più volte usando chiavi differenti
- Costruzione generale applicabile ai block cipher

Double DES

Due cifrature consecutive utilizzando due chiavi k_1 e k_2 . Per decifrare ci applicano in ordine opposto.

Questo approccio non migliora di molto la sicurezza: tramite l'attacco *Meet in the Middle*, conoscendo plaintext e ciphertext, l'attaccante è in grado di ridurre lo spazio di ricerca a poco più dello spazio delle chiavi (2^{63} vs 2^{56}). In breve, si costruisce una tabella con tutte le possibili cifrature di un plaintext e poi si cerca una corrispondenza con le decifrature di un ciphertext.

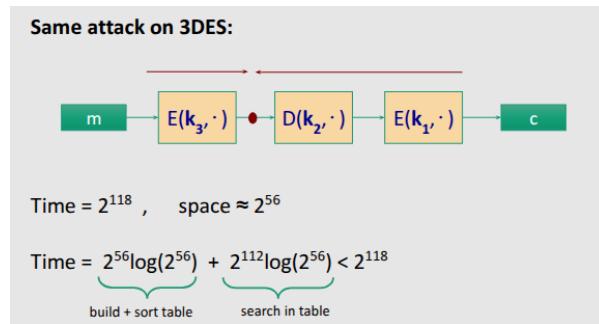
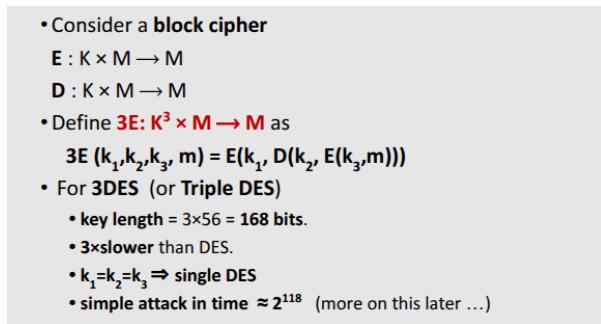




Triple DES (3DES)

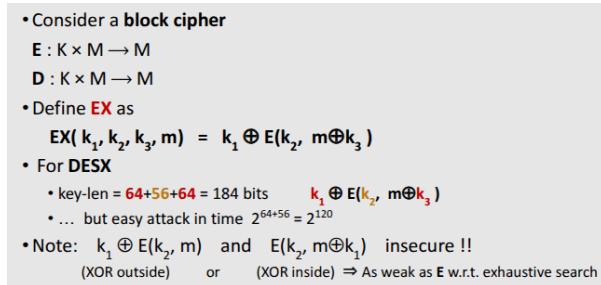
In questo caso vengono usate tre chiavi distinte k_1 , k_2 e k_3 per effettuare: cifratura con k_3 , "decifrare" con k_2 , cifratura con k_1 . Se usassimo tre chiavi uguali otterremmo DES classico. 3DES è tre volte più lento di DES.

Applicando lo stesso attacco *Meet in the Middle* su 3DES lo spazio di ricerca si attesta su una dimensione elevata, circa 2^{118} .



DESX

Introduce operazioni di XOR con chiavi aggiuntive prima e dopo DES. Protegge da attacchi brute-force, ma può essere vulnerabile ad attacchi avanzati.



Altri attacchi possibili

Crittoanalisi differenziale

È un CPA (Chosen Plaintext Attack), quindi si assume che l'attaccante conosca alcune coppie (plaintext, ciphertext).

La crittoanalisi differenziale compara lo XOR di due plaintext con lo XOR dei rispettivi ciphertext. La distribuzione di queste analisi potrebbe rivelare informazioni sulla chiave (alcuni bit). Si prosegue con un attacco brute-force per trovare i bit rimanenti della chiave.

DES non ne è vulnerabile in quanto durante il suo sviluppo tale attacco era già noto, quindi le S-box sono state progettate per resistervi.

Crittoanalisi lineare

Un attacco che cerca di replicare un'approssimazione più semplice (lineare) del cifrario nella sua interezza, al fine di ridurre la complessità dell'attacco. È un tipo di attacco applicabile a cifrari iterativi (tra i più usati contro i block cipher).

Non è pratico per violare DES in quanto richiede un numero di coppie di partenza troppo elevato.

Attacchi all'implementazione

- Side channel attacks: misurare il tempo e/o la potenza usati per cifrare e decifrare, al fine di capire in quale fase si trova l'algoritmo e cercare di carpire delle informazioni
 - Fault attacks: cercano di sfruttare errori nel calcolo che espongano la chiave k
- Per evitare questi attacchi, mai implementare primitive di crittografia "fai da te".

AES (Advanced Encryption Standard)

Progettato per sostituire il cifrario DES ormai vulnerabile, AES è lo standard di crittografia simmetrica, progettato per essere efficiente su numerose piattaforme hardware e/o software.

AES è un cifrario a blocchi che lavora iterativamente con:

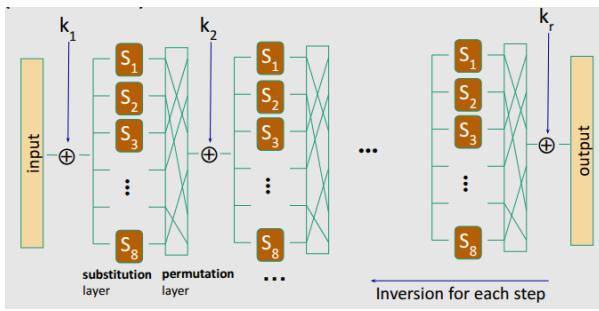
- blocchi di 128 bit (anche 192 o 256 volendo)
- chiavi di lunghezza 128, 192 o 256 bit
- 10, 12 o 14 round in base alla lunghezza della chiave
- 44, 52 o 60 sotto-chiavi di lunghezza 32 bit

1 word corrisponde a 4 byte, ossia 32 bit, quindi un blocco classico è di 4 word.

- Block size: 128 bits (Nb=4 words)
- 1 word = 32 bit

	Key Length (Nk words)	Block Size (Nb words)	Number of Rounds (Nr)
AES-128	4	4	10
AES-192	6	4	12
AES-256	8	4	14

AES non si basa sulla rete di Feistel, bensì è basato su una Substitution-Permutation Network (SPN).



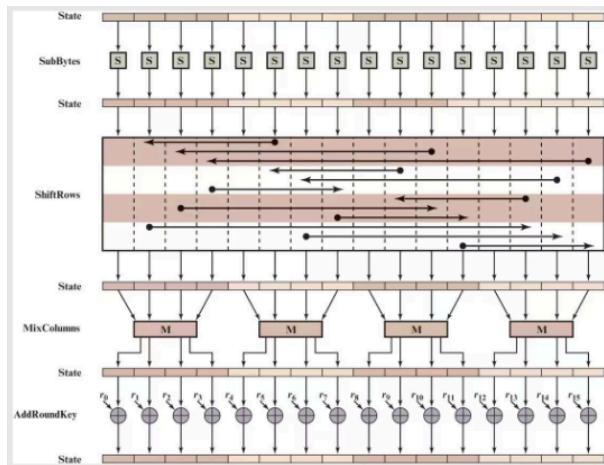
Funzionamento

Inizialmente la chiave viene espansa, generando 44 sotto-chiavi di 4 byte ciascuna. Il plaintext viene messo nell'*input state* e gli viene applicata una *initial transformation*. Da qui si genera la *state matrix*, che rappresenta l'input per il primo round.

Ogni round di AES, con input la *state matrix* e la sotto-chiave di round, include quattro operazioni fondamentali:

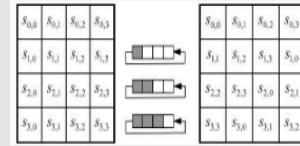
- SubBytes:** sostituzione di ogni byte attraverso una S-Box
- ShiftRows:** permutazione, spostamento ciclico delle righe della matrice
- MixColumns:** operazioni matematiche basate sui campi di Galois
- AddRoundKey:** XOR bitwise con la sotto-chiave del round

L'*output matrix* risultante sarà l'input per il round successivo.

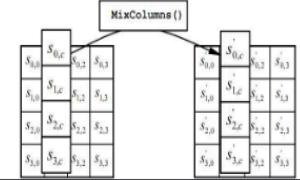


• **ByteSub:** a 1 byte S-box. 256 byte table (easily computable)
 • Apply S-box to each byte of the 4x4 input A, i.e., $A[i,j] = S[A[i,j]]$, for $1 \leq i,j \leq 4$

• **ShiftRows:**



• **MixColumns:**



AES è considerato sicuro per almeno altri 20 anni, anche contro gli attacchi crittografici più avanzati (best key recovery attack, related key attack). È immune al brute-force per la lunghezza delle chiavi e per il numero di chiavi possibili:

- AES-128: 2^{128} chiavi = $3.4 \cdot 10^{38}$
- AES-192: 2^{192} chiavi = $6.2 \cdot 10^{57}$
- AES-256: 2^{256} chiavi = $1.1 \cdot 10^{77}$

Modalità di operazione

Le **modalità di operazione** definiscono il modo in cui un cifrario a blocchi viene utilizzato per cifrare messaggi più lunghi della dimensione del blocco. Queste modalità influenzano la sicurezza e le proprietà del cifrario.

Nella tabella seguente sono riportate le principali modalità:

Modalità	Descrizione	Applicazioni tipiche
Electronic Code Book (ECB)	Ogni blocco di bit del plaintext è codificato indipendentemente usando la stessa chiave	Trasmissione sicura di valori singoli (una chiave di cifratura)
Cipher Block Chaining (CBC)	L'input dell'algoritmo di cifratura è lo XOR del blocco successivo di plaintext e del precedente di ciphertext	Trasmissioni block-oriented generiche, autenticazione
Cipher Feedback (CFB)	L'input è processato s bit alla volta. Il ciphertext precedente è usato come input per l'algoritmo di cifratura per produrre un output pseudorandom, il quale viene passato in XOR con il plaintext per produrre il prossimo ciphertext	Trasmissioni stream-oriented generiche, autenticazione
Output Feedback (OFB)	Simile al CFB, tranne che l'input dell'algoritmo di cifratura è il precedente ciphertext, usando i blocchi interi	Trasmissioni stream-oriented su canali rumorosi (comunicazione satellitare)
Counter (CTR)	Ogni blocco di plaintext viene passato in XOR con un contatore cifrato, il quale è incrementato ogni blocco successivo	Trasmissioni block-oriented generiche, utile quando si richiede alta velocità

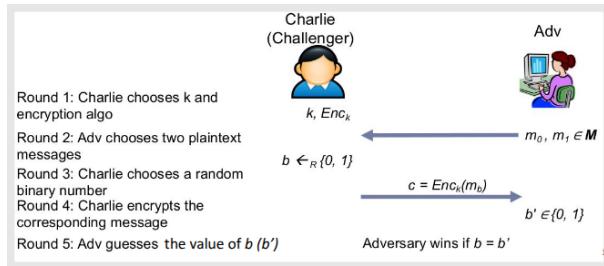
Nozioni di sicurezza

Vediamo alcune nozioni di sicurezza ed il loro significato.

Eavesdropping security, IND-EAV

Indistinguibilità del ciphertext per un intercettatore.

L'attaccante effettua un Ciphertext-only Attack (CA): osserva i ciphertext cercando di carpire informazioni sul plaintext.

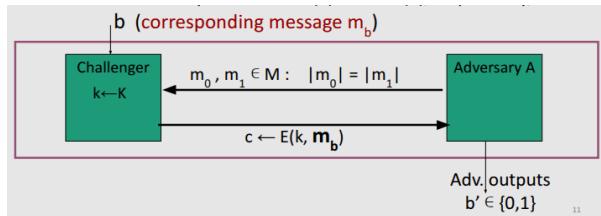


Sicurezza semantica

Chiave usata una sola volta

Definiamo un gioco di attacco per un cifrario $Q = (E, D)$ ed un avversario A . A mi manda due messaggi m_0 ed m_1 di uguale lunghezza. Io restituisco uno dei due cfrato e A deve capire se è lo 0 o l'1.

In $\text{EXP}(0)$ cifriamo m_0 , in $\text{EXP}(1)$ cifriamo m_1 .



Il vantaggio dell'avversario (Adv_{SS}) è la differenza in valore assoluto di due probabilità:

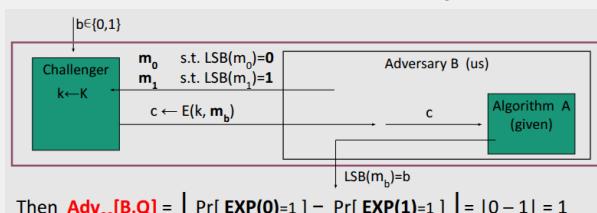
- $\Pr[\text{EXP}(0) = 1]$: prob. che A risponda 1 all'esperimento 0, quindi che A perda
- $\Pr[\text{EXP}(1) = 1]$: prob. che A risponda 1 all'esperimento 1, quindi che A vinca

$$\text{Adv}_{SS}[A, Q] = | \Pr[\text{EXP}(0) = 1] - \Pr[\text{EXP}(1) = 1] |$$

Q è **semanticamente sicuro** se per ogni A "efficiente", Adv_{SS} è trascurabile.

Questo vuol dire che l'avversario non è in grado di distinguere il messaggio ricevuto dal ciphertext.

Supponiamo che un A efficiente sia sempre in grado di dedurre il bit meno significativo del plaintext dal ciphertext $\Rightarrow Q$ non è semanticamente sicuro.

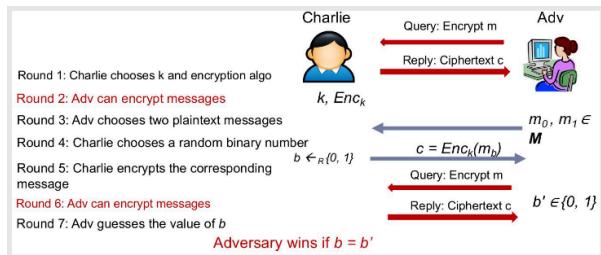


IND-CPA

La sicurezza semantica è equivalente ad un'altra definizione di sicurezza chiamata indistinguibilità del ciphertext per un CPA.

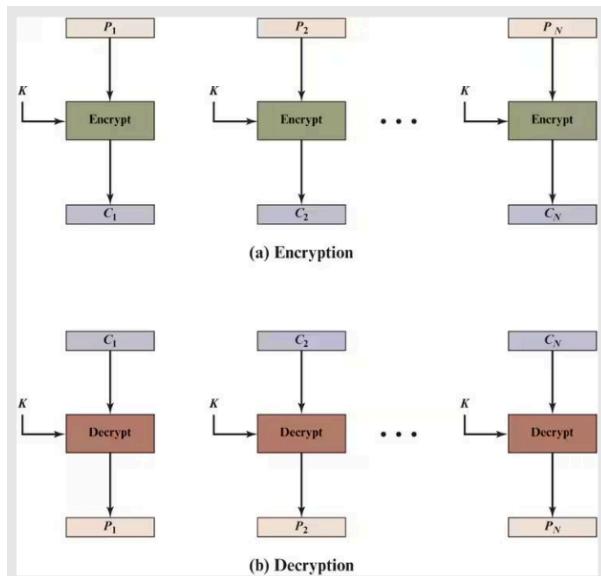
Chiave usata più volte

L'avversario osserva molti ciphertext cifrati con la stessa chiave. Le sue capacità sono quelle di un Chosen-Plaintext Attack (CPA), quindi può ottenere il ciphertext di messaggi arbitrari a sua scelta (similmente al mondo reale). Il suo obiettivo è violare la sicurezza semantica.

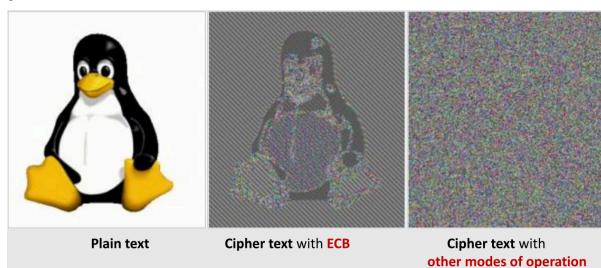


Electronic Code Book (ECB)

Il messaggio originale viene suddiviso in blocchi indipendenti p_1, \dots, p_N . La stessa chiave K è usata per cifrare e decifrare ciascun blocco separatamente.



Questa modalità presenta diverse problematiche. La prima è il determinismo, ovvero lo stesso blocco di plaintext genera sempre lo stesso blocco di ciphertext. Questo causa l'esposizione di pattern nei dati cifrati, rendendo facile identificare informazioni ripetute. Visivamente:



La modalità ECB non è semanticamente sicura per messaggi che contengono più di un blocco. Non usare l'ECB nella pratica.



Soluzione 1: cifratura randomizzata

Sia $E(k, m)$ un algoritmo randomizzato.

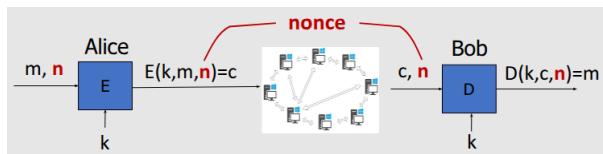
Allora cifrare lo stesso messaggio due volte, con alta probabilità, produrrà due ciphertext distinti.

Il ciphertext deve essere più lungo del plaintext, quindi si aggiungono un certo numero di bit random prima di cifrare.

Soluzione 2: cifratura basata sul nonce

Il termine **nonce** indica un numero che ha un utilizzo unico e quindi non ripetuto. Un nonce potrebbe essere composto da un timestamp, un contatore crescente di numeri oppure una combinazione dei due (non solo questo). Il nonce deve cambiare ad ogni operazione di cifratura.

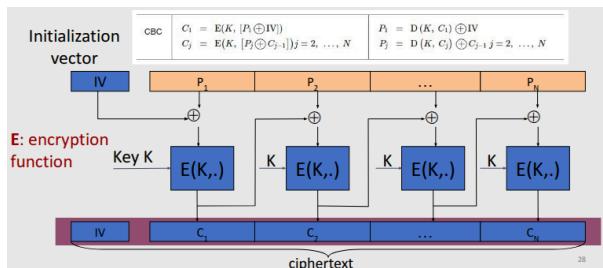
Nel caso di ECB, avere un nonce n che varia da messaggio a messaggio consente di produrre ciphertext diversi per uno stesso plaintext. Per ottenere ciò, la coppia (chiave, nonce) non deve mai essere utilizzata più di una volta. Il nonce n non occorre che sia segreto o casuale.



Cipher Block Chaining (CBC)

In questa modalità, ogni blocco di plaintext è combinato con il blocco di ciphertext precedente, prima della cifratura. Quindi ogni ciphertext dipende dal suo precedente, per questa ragione serve un Initialization Vector (IV) per il primo blocco.

Cifrare



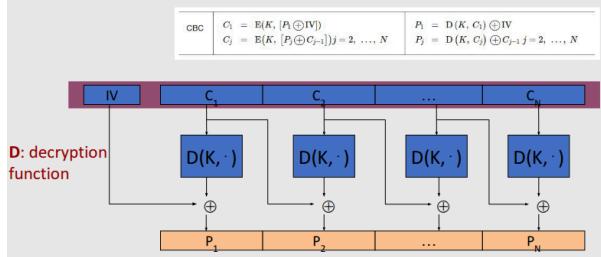
L'Initialization Vector deve essere noto sia al mittente che al destinatario, ma deve essere casuale e imprevedibile per terzi, altrimenti CBC diventa vulnerabile. L'IV dovrebbe essere protetto contro modifiche non autorizzate.

Ci sono due metodi per generare un IV:

1. Generare un blocco di dati casuale utilizzando un generatore di numeri casuali.
2. Cifrare un nonce con la stessa chiave usata per il plaintext. Il nonce deve essere un blocco di dati univoco per ogni esecuzione (contatore, timestamp, ...).

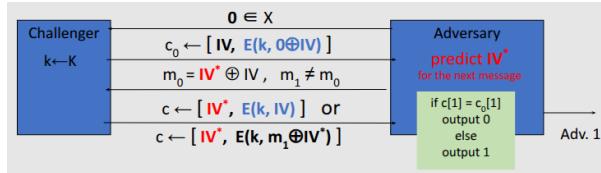
Se l'IV non è casuale bisogna sempre cifrarlo prima dell'uso, altrimenti non è CPA-sicuro!

Decifrare



Una modalità CBC in cui l'avversario può predire l'Initialization Vector non è al sicuro dai CPA: se l'IV è prevedibile, un attaccante può manipolare il primo blocco del messaggio cifrato.

Supponiamo, dato $c \leftarrow E_{CBC}(k, m)$, che l'avversario possa predire l'IV per il prossimo messaggio:

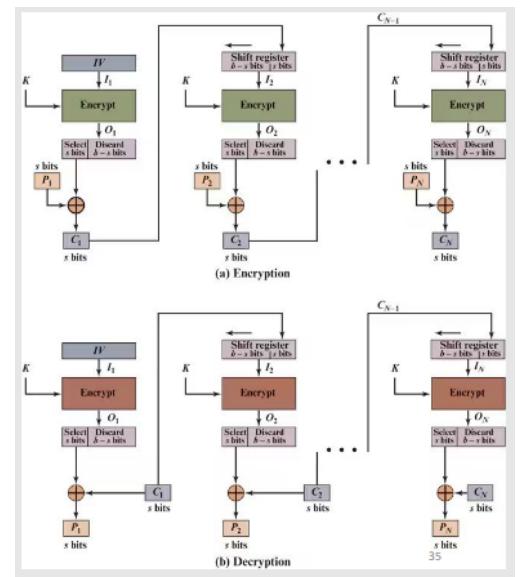


In conclusione, CBC ha una sicurezza maggiore di ECB, in quanto lo stesso blocco di plaintext produce blocchi di ciphertext diversi. Tuttavia, se l'IV non è casuale, CBC è vulnerabile agli attacchi CPA.

Cipher FeedBack (CFB)

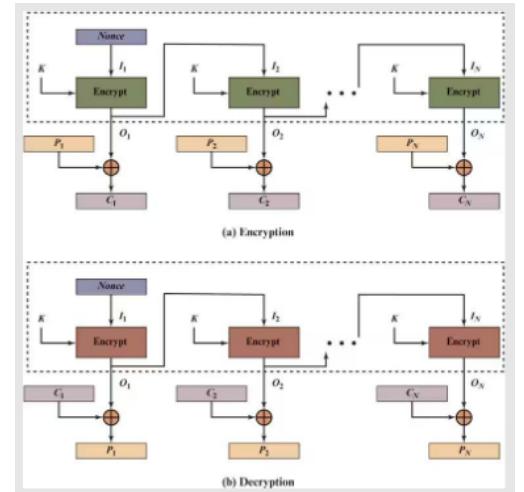
Utilizzando i block cipher, la cifratura viene effettuata su blocchi di n bit (DES = 64, AES = 128). È possibile "convertire" un block cipher in uno stream cipher usando le modalità CFB, OFB oppure CTR. Uno stream cipher elimina la necessità di suddividere precisamente il messaggio in blocchi. Inoltre, può operare in tempo reale, cifrando e trasmettendo i caratteri senza dover aspettare il blocco intero.

CFB converte un cifrario a blocchi in un cifrario a flusso, permettendo di cifrare dati di lunghezza arbitraria senza padding. Ogni ciphertext viene usato come input per la cifratura del blocco successivo. Uno svantaggio è che un bit errato nel testo cifrato si propaga nei blocchi successivi.



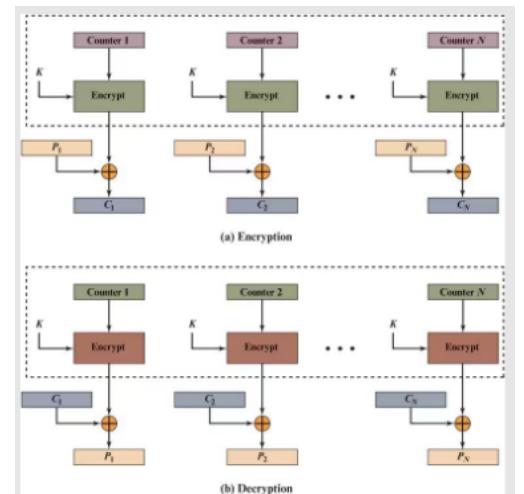
Output FeedBack (OFB)

OFB è simile a CFB, ma il valore di feedback è calcolato prima della cifratura. In questo caso l'IV deve essere un nonce, univoco per ogni esecuzione della cifratura. Un vantaggio di OFB è che gli errori di trasmissione non si propagano.



Counter (CTR)

CTR trasforma un cifrario a blocchi in un cifrario a flusso, usando un contatore che cambia valore ad ogni blocco di plaintext. Il valore del contatore viene cifrato ed il risultato viene messo in XOR con il plaintext. CTR non richiede padding ed è parallelizzabile, ovvero ogni blocco può essere cifrato indipendentemente.



XTS-AES

Modalità standardizzata per dispositivi di archiviazione, come HD e SSD. Protegge i dati anche in caso di accesso fisico da parte di un attaccante. Ampiamente utilizzata in sistemi di crittografia dei dischi.

Scambio di chiavi

Lo **scambio di chiavi** è il procedimento con cui due parti comunicanti A e B cooperano per ottenere una chiave crittografica condivisa e segreta.

È assolutamente necessario per la crittografia simmetrica, dove entrambi gli estremi devono conoscere la stessa chiave, mantenendola segreta verso terzi per prevenire attacchi.

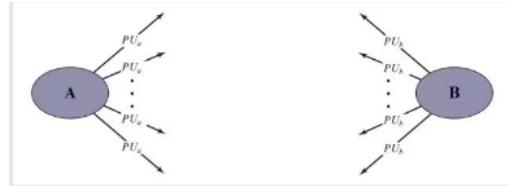
È preferibile cambiare frequentemente la chiave, per limitare la quantità di informazioni compromesse se un attaccante dovesse arrivare a scoprirla.

Distribuzione delle chiavi pubbliche e certificati

Per garantire comunicazioni sicure, è fondamentale distribuire correttamente le chiavi pubbliche.

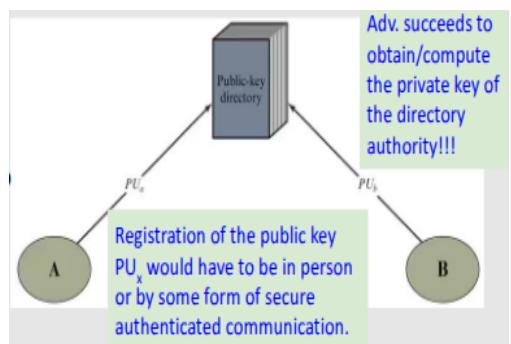
Annuncio pubblico

Metodo più banale (e ingenuo): l'utente diffonde la propria chiave pubblica. Il problema è che chiunque potrebbe fingere di essere un altro utente.



Directory pubblica

Una directory centralizzata, dinamica e pubblicamente disponibile, mantiene le chiavi pubbliche. Il mantenimento e la distribuzione della directory dovrebbe essere responsabilità di un qualche ente fidato oppure di una Central Authority (CA). Tale directory può essere vulnerabile a compromissione.

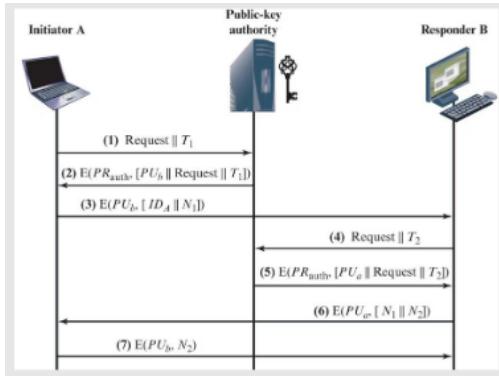


Public-key authority

Supponiamo che una CA mantenga una directory delle chiavi pubbliche di tutti i partecipanti. Ogni partecipante conosce una chiave pubblica affidabile per l'authority.

A manda un messaggio con timestamp alla public-key authority. L'authority risponde con un messaggio contenente la chiave pubblica di *B* (PU_b), cifrato usando la propria chiave privata (PR_{auth}). *A* salva PU_b e la usa per cifrare un messaggio verso *B* contenente un identificativo ID_A ed un nonce N_1 . *B* effettua gli stessi passaggi di *A*. I nonce N_1 e N_2 vengono usati per identificare univocamente la transazione tra *A* e *B*.

Il principale svantaggio di questo scenario è che la public-key authority potrebbe diventare un collo di bottiglia, in quanto un utente deve chiedere la chiave pubblica di ogni altro utente che desidera contattare. Inoltre, resta il problema della directory vulnerabile a compromissioni.



Certificati

Essenzialmente, un **certificato** è composto da:

- una chiave pubblica
- un identificativo del proprietario della chiave
- la firma di un ente terzo fidato

Tipicamente, l'ente terzo è un'**Autorità di Certificazione (CA, Certificate Authority)**, come un'agenzia governativa o un'istituzione finanziaria, di cui la comunità di utenti si fida. La CA emette certificati digitali attestanti l'autenticità delle chiavi pubbliche.

Ciascun partecipante si rivolge alla CA, fornendo una chiave pubblica e richiedendo un certificato. L'applicazione deve essere svolta di persona oppure tramite una comunicazione sicura ed autenticata.

Per un partecipante *A*, l'autorità fornisce un certificato della forma:

$$C_A = E(PR_{auth}, [T || ID_A || PU_a]), \text{ dove } T \text{ è il timestamp e } ID_A \text{ l'identificativo.}$$

Qualsiasi partecipante riceva il certificato da *A*, può verificarlo nel seguente modo:

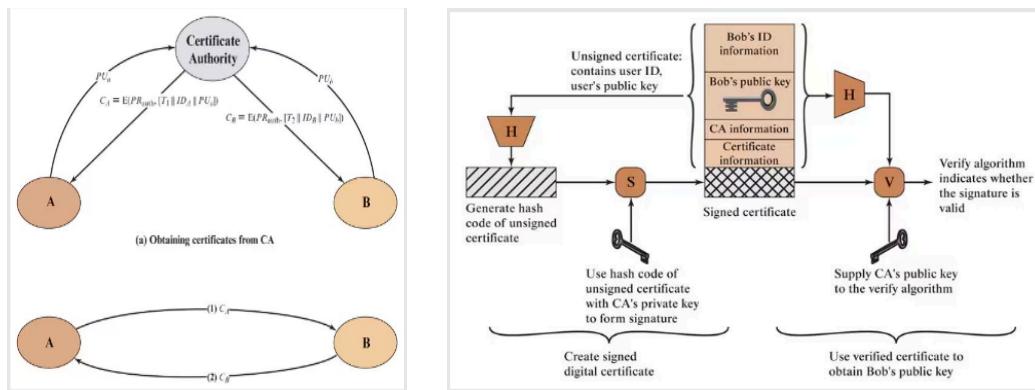
$$D(PU_{auth}, C_A) = D(PU_{auth}, E(PR_{auth}, [T || ID_A || PU_a])) = (T || ID_A || PU_a)$$

I certificati devono soddisfare dei requisiti:

- Ogni partecipante deve poter leggere il certificato per determinare il nome e la chiave pubblica del suo proprietario
- Ogni partecipante deve poter verificare che il certificato provenga da una CA e che non sia contraffatto
- Solo la CA può generare ed aggiornare i certificati
- Ogni partecipante deve poter verificare la validità temporale del certificato

Certificati X.509

Esiste uno schema universalmente accettato per la formattazione dei certificati di chiave pubblica: lo standard **X.509**. I certificati X.509 sono usati nella maggior parte dei protocolli di sicurezza, come TLS.



Distribuzione delle chiavi segrete

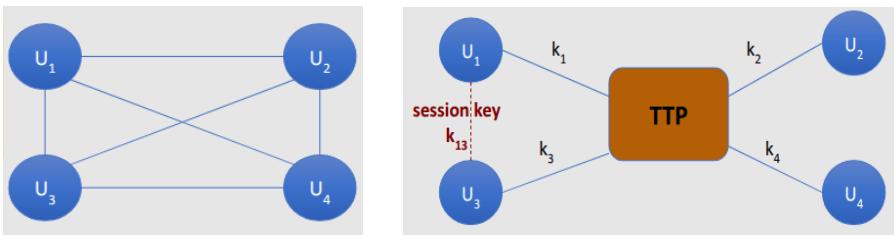
Per due estremi A e B , lo scambio di chiavi segrete (crittografia simmetrica) può avvenire in diversi modi:

- A crea o sceglie una chiave e la consegna fisicamente a B
- Un terzo fidato distribuisce fisicamente la chiave ad A e B
- Se A e B hanno recentemente usato una chiave, A può trasmettere la nuova chiave a B , cifrandola con quella vecchia
- Se A e B hanno entrambi una connessione sicura ad un terzo C , C può trasmettere una chiave ad A e B tramite il canale cifrato.

Terzi fidati (TTP, Trusted Third Parties)

Avendo una rete di n utenti, diventa difficoltoso memorizzare le chiavi segrete. Ogni utente avrebbe $O(n)$ chiavi, portando il numero totale di chiavi nel sistema a $O(n^2)$.

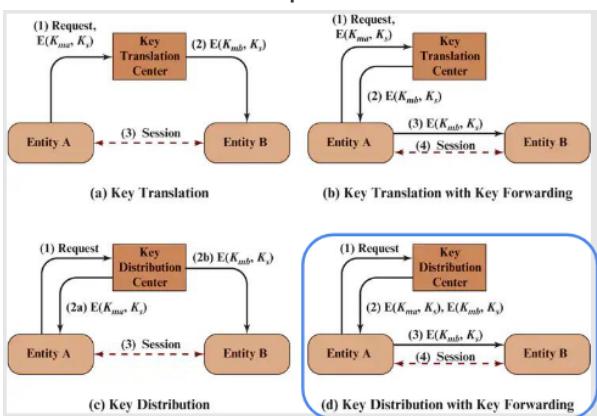
Un approccio più conveniente è avere un server fidato, il TTP, che memorizza le chiavi segrete e all'occorrenza le fornisce. Ogni utente memorizza soltanto la chiave segreta condivisa con il TTP.



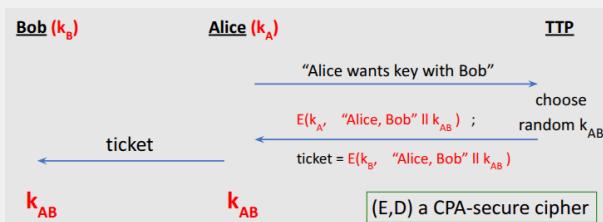
Distribuzione della chiave usando un TTP

A e B possiedono entrambi una chiave segreta principale (K_{ma} e K_{mb}) condivisa con il TTP coinvolto nella distribuzione. Il TTP genera una session key temporanea (K_s) che durerà per l'arco della connessione logica tra A e B , consentendo una comunicazione sicura.

Se il TTP viene compromesso, tutte le chiavi sono a rischio.



Alice vuole una chiave condivisa con Bob, in uno scenario di eavesdropping security soltanto.



Alice chiede la chiave al TTP, il quale ne genera una casuale (k_{AB}). Il TTP risponde ad Alice con due messaggi contenenti la chiave condivisa, uno cifrato con la chiave di Alice (k_A) e l'altro con quella di Bob (k_B). Alice decifra il suo e spedisce l'altro a Bob.

Siccome l'intercettatore non può distinguere i plaintext avendo il ciphertext, la chiave condivisa è al sicuro.

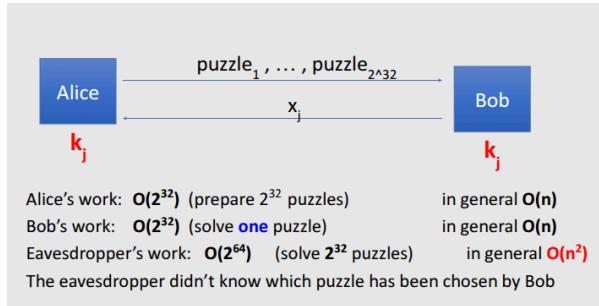
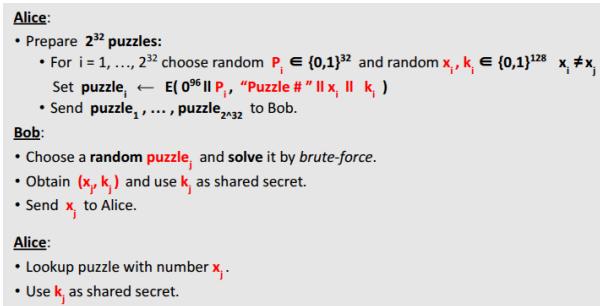
Questo protocollo non è sicuro contro gli attacchi attivi, ad esempio un replay attack.

Merkle puzzles

Esistono metodi per scambiare chiavi in modo sicuro senza un TTP.

Supponiamo che Alice e Bob vogliano una chiave condivisa, in uno scenario di eavesdropping security soltanto (no manomissione, no message injection).

I **Merkle puzzles** sono un modo, computazionalmente inefficiente, per arrivare ad un segreto condiviso comune.



Alice crea 2^{32} **puzzle crittografici** contenenti un identificativo (x_i) ed una chiave segreta (k_i), li cifra con una chiave casuale (P_i) e poi li trasmette a Bob. Bob sceglie random un puzzle j e lo risolve tramite brute-force. La chiave k_j sarà la chiave segreta condivisa, mentre x_j viene trasmesso ad Alice che così troverà la chiave corrispondente. Un eventuale intercettatore dovrebbe risolvere tramite brute-force tutti i puzzle per trovare la chiave.

Protocollo di Diffie-Hellman

Premessa: Alice e Bob non hanno nessun segreto condiviso inizialmente.

Diffie-Hellman è il primo vero protocollo di scambio di chiavi senza terzi, il quale si basa sulla difficoltà del problema del **logaritmo discreto**.

Procedimento

Sia p un numero primo enorme, fissato. Sia g un intero tra 2 e $p - 2$.

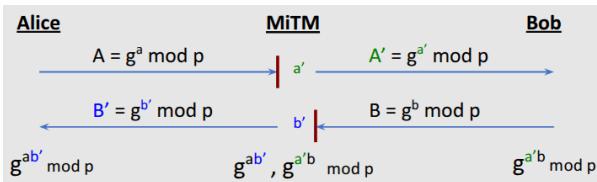
1. Alice sceglie un numero a e calcola $A = g^a \pmod p$
2. Bob sceglie un numero b e calcola $B = g^b \pmod p$
3. Alice e Bob si scambiano A e B
4. Alice calcola la chiave segreta $K = B^a \pmod p$
5. Bob calcola la chiave segreta $K = A^b \pmod p$

A questo punto, Alice e Bob hanno una chiave segreta condivisa $K = g^{ab} \pmod p$.

Un eventuale intercettatore vedrebbe passare tutti i parametri sul canale, ma non potrebbe calcolare la chiave perché richiederebbe tempo esponenziale nella dimensione di p .

Vulnerabilità

Il protocollo di Diffie-Hellman non è sicuro contro gli attacchi Man-in-the-Middle (MiTM).



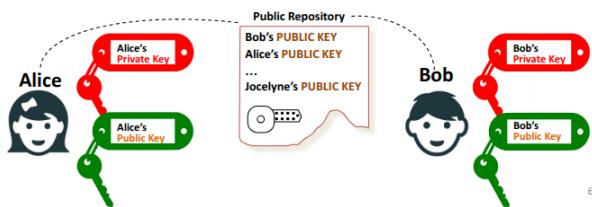
In generale, il protocollo non è sicuro contro gli attacchi attivi, come l'alterazione dei messaggi.

Crittografia asimmetrica

Per poter comunicare sicuramente utilizzando un algoritmo di crittografia simmetrica, come 3DES o AES, i due estremi della comunicazione devono possedere una chiave segreta comune. È problematico scambiarla quando il canale non è sicuro. Inoltre, ogni utente dovrebbe mantenere una chiave per ciascuna controparte della comunicazione.

La **crittografia asimmetrica** fornisce delle soluzioni in merito. Essenzialmente:

1. Ogni utente genera una coppia di chiavi da usare per cifrare e decifrare i messaggi
2. La **chiave pubblica** viene resa disponibile agli altri utenti tramite un registro pubblico
3. La **chiave privata** rimane segreta (la conosce solo il proprietario)
4. Ciascun utente mantiene una collezione delle chiavi pubbliche degli utenti con cui ha comunicato



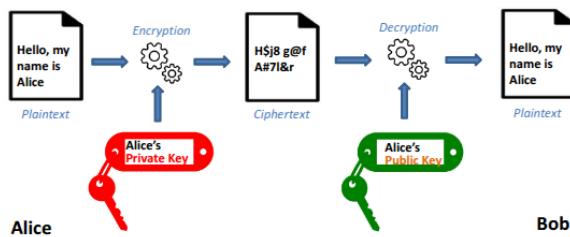
Tra chiave pubblica e privata c'è una relazione. Siccome la chiave pubblica è disponibile a tutti, deve essere estremamente complesso risalire alla chiave privata da quella pubblica.

La crittografia asimmetrica funziona "in entrambe le direzioni":

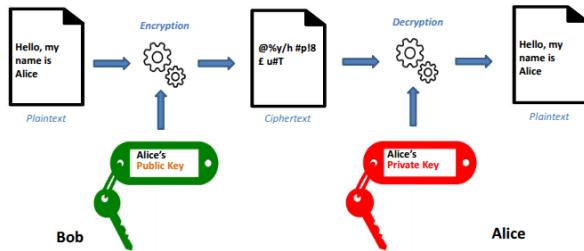
- cifrando un plaintext con la chiave pubblica si ottiene un ciphertext, decifrabile soltanto con la chiave privata
- cifrando un plaintext con la chiave privata si ottiene un ciphertext, decifrabile soltanto con la chiave pubblica

Casi d'uso

- Autenticazione, firma digitale: Alice può cifrare i dati con la sua chiave privata, in modo che chiunque, tramite la sua chiave pubblica, possa accertarsi della loro autenticità e integrità



- Confidenzialità: Bob può cifrare il messaggio utilizzando la chiave pubblica di Alice, in modo che solo essa possa decifrarlo, utilizzando la sua chiave privata



- Scambio di chiavi: i due estremi cooperano per stabilire una chiave di sessione simmetrica, che ha validità temporanea

Quindi a differenza della crittografia simmetrica, con quella asimmetrica non è necessario avere una chiave segreta condivisa a priori, e ogni utente necessita di sole due chiavi, pubblica e privata. Per ottenere la chiave pubblica di altri utenti ci si rivolge ad una CA, che ne garantisce la validità.

Definizioni formali

Un **sistema di crittografia a chiave pubblica** è una tripla di algoritmi (G, E, D) in cui:

- $G()$ è un algoritmo randomizzato che genera una coppia di chiavi pubblica e privata (p_k, s_k)
- $E(p_k, m)$ è un algoritmo randomizzato che prende un messaggio m e produce il suo ciphertext c
- $D(s_k, c)$ è un algoritmo deterministico che ricava il messaggio m dal ciphertext c

Consistenza per ogni coppia (p_k, s_k) generata da G : $\forall m \quad D(s_k, E(p_k, m)) = m$.

Trapdoor one-way functions (TDF)

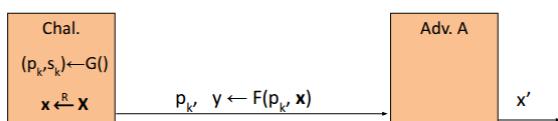
Una **TDF** è una funzione facile da calcolare in una direzione, ma difficilissima da invertire senza informazioni aggiuntive. Per poterla invertire in tempo polinomiale serve conoscere un'informazione segreta (la chiave), altrimenti è computazionalmente infattibile.

Una TDF $X \rightarrow Y$ è una tripla di algoritmi efficienti (G, F, F^{-1}) , in cui:

- $G()$ è un algoritmo randomizzato che genera una coppia di chiavi (p_k, s_k)
- $F(p_k, \cdot)$ è un algoritmo deterministico che definisce una funzione $X \rightarrow Y$
- $F^{-1}(s_k, \cdot)$ definisce una funzione $Y \rightarrow X$ che inverte $F(p_k, \cdot)$

Per ogni coppia (p_k, s_k) generata da G , $\forall x \quad F^{-1}(s_k, F(p_k, x)) = x$.

Informalmente, possiamo dire che una TDF è sicura se $F(p_k, \cdot)$ è una one-way function che può essere calcolata, ma non invertita senza s_k .



(G, F, F^{-1}) è una **TDF sicura (TDFs)** se per ogni A efficiente la quantità

$Adv_{OW}[A, F] = Pr[x = x']$ è trascurabile.

Ciò significa che l'avversario ha una probabilità pressoché nulla di risalire al dato iniziale.

Algoritmi di hash one-way

Un algoritmo di hash one-way trasforma un documento in un output condensato di lunghezza definita. Gli algoritmi di hash sono esempi di TDF.

Sia H un algoritmo di hash one-way, sia m una stringa binaria in input di lunghezza arbitraria. L'output $H(m)$ è una stringa binaria di L bit, detta *hash di m sotto H*, dove L è fissato per la funzione H .

Un buon algoritmo di hash dovrebbe avere le seguenti proprietà:

- Il valore hash deve essere efficiente da calcolare
- Deve essere computazionalmente infattibile invertire il valore hash per risalire al documento originale
- Deve essere computazionalmente infattibile trovare due documenti aventi lo stesso valore hash
- Una piccola modifica nell'input deve causare un grosso cambiamento nel valore hash

Mai sfruttare una TDF per cifrare un messaggio applicando direttamente F al plaintext. Essendo una funzione deterministica, non è semanticamente sicura.

RSA

Un algoritmo di crittografia a chiave pubblica basato su risultati della Teoria dei Numeri, in particolare sulla difficoltà della fattorizzazione di grandi numeri primi.

Trapdoor permutation

Una **trapdoor permutation** è una tripla di algoritmi (G, F, F^{-1}) , in cui:

- $G()$ genera una coppia di chiavi (p_k, s_k)
- p_k definisce una funzione $F(p_k, \cdot) : X \rightarrow X$
- $F(p_k, x)$ calcola la funzione su x
- $F^{-1}(s_k, y)$ inverte la funzione F su y utilizzando s_k

Una trapdoor permutation è **sicura** se la funzione $F(p_k, \cdot)$ senza la trapdoor s_k è una funzione one-way.

Funzionamento e utilizzo

G sceglie due numeri primi casuali p e q di circa 1024 bit. Sia $N = p \cdot q$. Ora sceglie due interi e e d , dove $1 < e < \phi(N)$ e $\gcd(e, \phi(N)) = 1$, tali che $e \cdot d \equiv 1 \pmod{\phi(N)}$. In output restituisce $p_k = (N, e)$, ovvero la chiave pubblica, e $s_k = (N, d)$, ovvero la chiave privata. $e \cdot d \pmod{\phi(N)}$ è la relazione tra chiave privata e chiave pubblica.

$$F(p_k, x) : \mathbb{Z}_N^* \rightarrow \mathbb{Z}_N^* = RSA(x) = x^e = y \quad (\text{in } \mathbb{Z}_N).$$

$F^{-1}(s_k, y) = y^d$, dove $y^d = (RSA(x))^d = (x^e)^d = x^{ed} = x^{k\phi(N)+1} = (x^{\phi(N)})^k \cdot x = x$.
 x è invertibile in \mathbb{Z}_N , quindi $x^{\phi(N)}$ fa 1, elevato alla k per x fa x .

Bob genera le chiavi:

- sceglie due numeri primi $p = 5$ e $q = 11$
 - $N = p \cdot q = 55$, $\phi(N) = (p-1)(q-1) = 40$
 - sceglie un intero $e = 3$ tale che $\gcd(e, 40) = 1$
 - calcola $d = 27$ in modo da soddisfare $(3 \cdot d) \pmod{40} = 1$
- Chiave pubblica = (3, 55), chiave privata = (27, 55)

Alice cifra il messaggio e lo spedisce a Bob:

- il messaggio da inviare è $m = 13$
- trova la chiave pubblica di Bob (3, 55)
- calcola $c = m^e \pmod{n} = 13^3 \pmod{55} = 2197 \pmod{55} = 52$
- manda il ciphertext $c = 52$ a Bob

Bob riceve il messaggio e lo decifra

- riceve il ciphertext $c = 52$ da Alice
- usa la sua chiave privata (27, 55) per calcolare $m = 52^{27} \pmod{55} = 13$

Sicurezza di RSA

È davvero così difficile invertire RSA senza conoscere la trapdoor?

Per invertire la one-way function di RSA, senza conoscere la chiave privata d , un attaccante dovrebbe ricavare x da $c = x^e \pmod{N}$. L'algoritmo attualmente migliore per calcolare la e -esima radice modulo N implica fattorizzare N , il che è computazionalmente inaffrontabile.

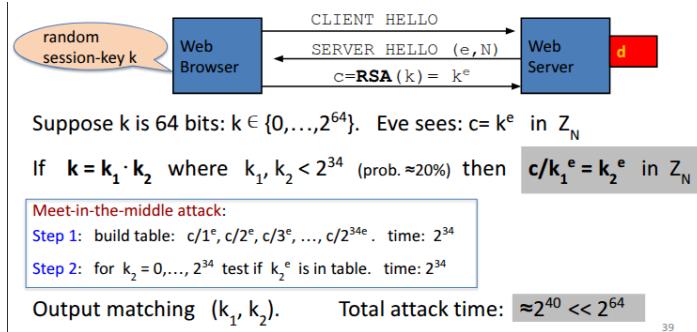
Per questo motivo, RSA è una one-way permutation, ovvero:

per ogni algoritmo A efficiente, la probabilità $Pr[A(N, e, y) = y^{1/e}]$ è trascurabile.

Plain/textbook RSA

Quando si parla di **plain/textbook RSA** si intende cifrare e decifrare utilizzando semplicemente l'elevamento a potenza seguito dal modulo (come fatto nell'esempio per intenderci).
 Mai usare textbook RSA come metodo di cifratura diretta! Non è semanticamente sicuro ed esistono numerosi attacchi, questo perché la trapdoor permutation di RSA non è uno schema di cifratura.

Esempio di attacco al textbook RSA



RSA in pratica

Sia (E_s, D_s) uno schema di crittografia simmetrica. Sia H una funzione tale che $H : \mathbb{Z}_N \rightarrow K$, dove K è lo spazio delle chiavi di (E_s, D_s) .

- $G()$: genera le chiavi di RSA, $p_k = (N, e)$ e $s_k = (N, d)$
- $E(p_k, m)$: sceglie casualmente un $x \in \mathbb{Z}_N$. Calcola $y = RSA(x) = x^e$ e $k = H(x)$. Restituisce in output $(y, E_s(k, m))$
- $D(s_k, (y, c))$: calcola $m = D_s(H(RSA^{-1}(y)), c)$ e lo restituisce in output

Per decifrare più velocemente, usare una chiave privata piccola (d circa 2^{128}). Se $d < N^{0.292}$ allora RSA non è sicuro, nel senso che la chiave privata d può essere ricavata dalla chiave pubblica (N, e) .

Per cifrare più velocemente, usare una chiave pubblica e piccola. Il valore minimo è $e = 3$, il valore raccomandato è $e = 65537 = 2^{16} + 1$.

La sicurezza di un sistema a chiave pubblica dovrebbe essere dell'ordine della sicurezza dei sistemi simmetrici:

Lunghezza della chiave segreta	Dimensione del modulo (N)
80 bit	1024 bit
128 bit	3072 bit
256 bit (AES)	15360 bit

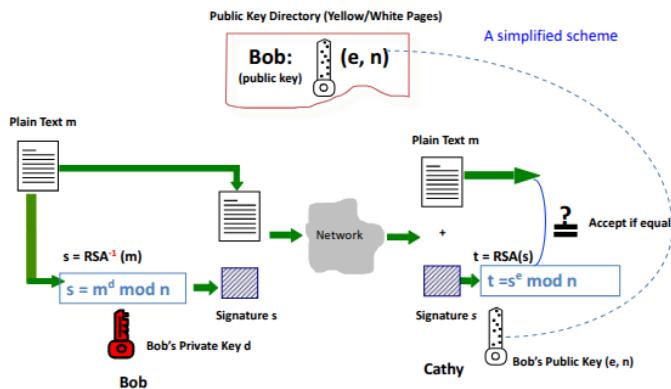
Attacchi

RSA potrebbe essere vulnerabile ad alcuni attacchi:

- Timing attack: il tempo necessario per decifrare il ciphertext, calcolando $c^d \pmod{N}$, può esporre d
- Power attack: il consumo di potenza di una smartcard per calcolare $c^d \pmod{N}$ può esporre d
- Faults attack: un errore del computer durante il calcolo $c^d \pmod{N}$ può esporre d
- Key generation trouble: se la generazione delle chiavi non è sufficientemente casuale, si può risalire ai fattori di N

Firme digitali con RSA

Scenario: Bob vuole inviare un messaggio M ad Alice. Sebbene non sia importante la segretezza del messaggio, Bob vuole che Alice sia certa che provenga da lui. Bob usa una funzione hash per generare un valore hash h per il messaggio M . Il valore hash h , cifrato con la chiave privata di Bob, funge da firma digitale S . Bob invia M con la firma S allegata. Alice userà la chiave pubblica di Bob per accettare la provenienza.



Bob genera le chiavi:

- sceglie due numeri primi $p = 5$ e $q = 11$
 - $N = p \cdot q = 55$, $\phi(N) = (p - 1)(q - 1) = 40$
 - sceglie un intero $e = 3$ tale che $gcd(e, 40) = 1$
 - calcola $d = 27$ in modo da soddisfare $(3 \cdot d) \pmod{40} = 1$
- Chiave pubblica = $(3, 55)$, chiave privata = $(27, 55)$

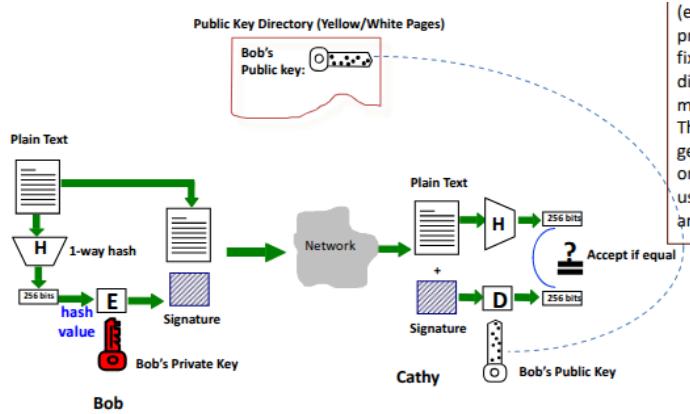
Bob firma e invia il documento:

- il documento da inviare è $m = 19$
- calcola la firma digitale $s = m^d \pmod{N} = 19^{27} \pmod{55} = 24$
- invia messaggio e firma $(m, s) = (19, 24)$

Alice riceve e verifica la firma:

- riceve $(m, s) = (19, 24)$
- richiede la chiave pubblica di Bob $(e, N) = (3, 55)$
- calcola la firma $t = s^e \pmod{N} = 24^3 \pmod{55} = 19$
- $t = m$ quindi la firma è di Bob

In caso di documenti molto lunghi, si calcola il valore hash del documento e poi si firma quello, che certifica inoltre l'integrità del documento.



Le firme digitali vengono utilizzate in quanto sono impossibili da contraffare. Inoltre:

- garantiscono la non ripudiabilità della firma
- garantiscono l'autenticità ed integrità del documento
- sono universalmente verificabili
- cambiano da un documento all'altro

Nota: chi verifica la firma non deve necessariamente possedere chiavi proprie.

Confronto finale

Simmetrico	Asimmetrico
Una chiave per cifrare e decifrare	Chiave pubblica/privata
Necessità di scambio sicuro della chiave	Più flessibile (no scambio segreto iniziale)
Più veloce	Più lento

How-to esercizi con RSA

Generato da ChatGPT

Generazione delle chiavi (pk, sk)

1. **Scegli due primi** p, q (per esercizi manuali usa numeri piccoli)
2. **Calcola** $n = p \cdot q$ e $\phi(n) = (p - 1)(q - 1)$
3. **Scegli** un esponente pubblico e con $1 < e < \phi(n)$ e $\gcd(e, \phi(n)) = 1$
4. **Trova** l'esponente privato d come **inverso modulare** di e modulo $\phi(n)$:
$$d \equiv e^{-1} \pmod{\phi(n)}$$
, cioè $e \cdot d \equiv 1 \pmod{\phi(n)}$

Output: chiave pubblica $pk = (n, e)$, chiave privata $sk = (n, d)$.

Come trovare l'inverso d a mano

Utilizzare l'algoritmo di Euclide esteso.

Per $3d \equiv 1 \pmod{40}$:

- dividi $40 = 3 \cdot 13 + 1 \Rightarrow 1 = 40 - 3 \cdot 13$
- quindi $3^{-1} \equiv -13 \equiv 27 \pmod{40}$

Cifratura

Per un messaggio m con $0 \leq m < n$:

$$c \equiv m^e \pmod{n}$$

Suggerimento pratico: usare “square-and-multiply”

Per calcolare $m^e \pmod{n}$ senza numeri giganteschi:

1. Scrivi e in binario
2. Parti da 1; scansiona i bit: **per ogni bit** fai “quadra” (eleva al quadrato) e riduci mod n ; se il bit è 1, **moltiplica** per m e riduci mod n . Riduci **sempre** modulo n dopo ogni operazione per tenere i numeri piccoli

Decifratura

Per decifrare il ciphertext c :

$$m \equiv c^d \pmod{n}$$

Usa sempre “square-and-multiply”.

Perché funziona: per i m coprimi con n , segue da $ed \equiv 1 \pmod{\phi(n)}$ ed **Euler** che $(m^e)^d \equiv m \pmod{n}$.

Firma e verifica

- **Firma** di m : $s \equiv m^d \pmod{n}$
- **Verifica**: calcola $t \equiv s^e \pmod{n}$;
- **accetta** se $t = m$.

Per **messaggi lunghi** si firma l'**hash** del messaggio, non il messaggio intero.

Trucchi di calcolo a mano

- **Riduzioni frequenti**: dopo ogni moltiplicazione o quadratura, riduci mod n
- **Spezzare l'esponente**: usa la scomposizione binaria (es. $27 = 16 + 8 + 2 + 1$) e pre-calcola $c, c^2, c^4, c^8, \dots \pmod{n}$
- **Controlli veloci**: verifica che $\gcd(e, \phi(n)) = 1$; se non lo è, cambia e

Mini-procedura riassuntiva per un esercizio

1. Scegli p, q piccoli; calcola n e $\phi(n)$
2. Prendi e con $\gcd(e, \phi(n)) = 1$
3. Trova d con Euclide esteso
4. **Cifra**: $c = m^e \pmod{n}$ (square-and-multiply)
5. **Decifra**: $m = c^d \pmod{n}$ (square-and-multiply)
6. **Firma/Verifica**: $s = m^d \pmod{n}$; verifica $s^e \pmod{n} = m$

Sicurezza informatica

Sicurezza di rete

Definizioni, sicurezza su internet e anonimato.

Principi di sicurezza

La sicurezza di rete ha le sue fondamenta in due principi fondamentali:

- **AAA**
 - Authentication (autenticazione)
 - Authorization (autorizzazione)
 - Accounting (accreditamento)
- **CIA**
 - Confidentiality (Confidenzialità)
 - Integrity (Integrità)
 - Availability (Disponibilità)

AAA

Authentication (autenticazione)

La proprietà di **autenticazione** è la capacità di un sistema di garantire che un utente possa essere identificato tramite informazioni in suo possesso. Queste informazioni possono essere di tre tipi (utilizzabili da soli o combinati):

- Qualcosa che si ha (badge, smartcard,...) → anche detti token
- Qualcosa che si sa (password, PIN, risposte a domande prestabilite)
- Qualcosa che si è (impronte digitali, retina e viso)

Se queste informazioni vengono combinate si ottiene la cosiddetta autenticazione multifattore. Attenzione però, più meccanismi dello stesso tipo non aumentano la sicurezza.

Authorization (autorizzazione)

L'**autorizzazione** indica che azioni può effettuare un determinato utente.

Accounting (accreditamento)

L'**accounting** è la procedura con cui si associano determinate operazioni effettuate a un account (logging).

CIA

Questi concetti sono indipendenti l'uno dall'altro. La sicurezza di un sistema si può misurare in base a questi tre fattori (che devono essere sempre presenti).

Confidentiality (Confidenzialità)

Un messaggio si definisce **confidenziale** se può essere letto solo dal destinatario pre-designato.

Attacchi alla confidenzialità

- Man-in-the-Middle: l'attaccante si mette nel mezzo di una comunicazione, anche spacciandosi per il destinatario con il mittente o per il mittente con il destinatario
- Eavesdropping: un intercettatore tenta di leggere o ascoltare un messaggio destinato ad altri

Integrity (Integrità)

Un messaggio si definisce **integro** se il destinatario è certo del suo mittente e che il contenuto del messaggio non è stato alterato.

Attacchi all'integrità

- Manipulation: l'attaccante modifica il messaggio prima che venga recapitato
- User impersonation/spoofing: l'attaccante si spaccia per un altro utente

Availability (Disponibilità)

La **disponibilità** è la capacità di un sistema di rispondere a determinate richieste di entità autorizzate.

Attacchi alla disponibilità

- Denial of Service (DoS): un'azione che impedisce o compromette l'uso autorizzato di reti, sistemi o applicazioni tramite esaurimento delle risorse come CPU, memoria, larghezza di banda e spazio su disco. L'attenzione si concentra generalmente sui servizi di rete, che vengono attaccati tramite la loro connessione.

È un DoS anche se l'attaccante elimina i messaggi, impedendogli di giungere a destinazione.

- Distributed Denial of Service (DDoS): un DoS in cui gli attacchi arrivano da più sorgenti (bot o zombie)

Anonimato

L'**anonimato** garantisce che un utente possa utilizzare una risorsa o un servizio senza rivelare la propria identità. È una proprietà non presente in CIA perché non rappresenta la sicurezza di un sistema, bensì un modo di eludere la proprietà di accounting.

Spoiler: navigando online non si è anonimi, si è soggetti a profilazione da parte di ISP (chi vi fornisce la rete), social network, cookies...

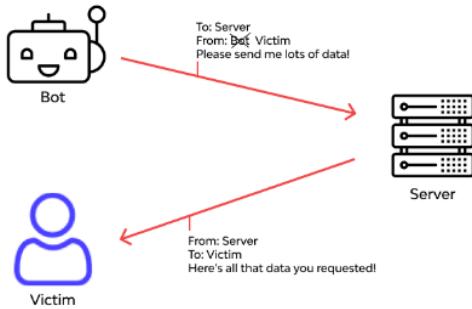
Attacchi in rete

IP

Internet Protocol (IP) è il protocollo usato a livello globale per identificare un dispositivo host o server. IPv4 prevede indirizzi a 32 bit, con IPv6 gli indirizzi sono lunghi 128 bit.

Il server deve conoscere il vostro indirizzo pubblico (IP address) per rispondere, quindi non c'è anonimato.

IP spoofing: gli indirizzi IP non sono protetti da una forma di integrità, perciò chiunque può cambiare il proprio IP sorgente fingendosi qualcun altro.



Routing

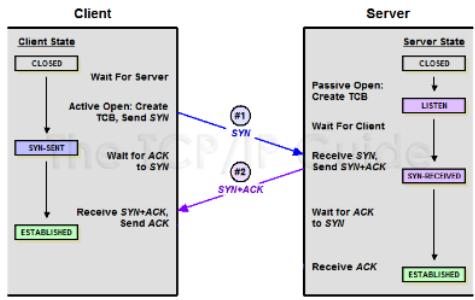
Non entreremo nei dettagli del routing. Ci limitiamo a dire che il pacchetto viene fatto passare attraverso un'infrastruttura di router, i quali sono per definizione Man in the Middle.

TCP

A livello trasporto, dove opera il protocollo **Transmission Control Protocol (TCP)**, i servizi vengono identificati tramite quella che viene indicata come *porta*.

TCP prevede il concetto di connessione, a differenza di altri protocolli dello stesso livello (UDP). Per effettuare questa operazione, TCP invia un pacchetto con una segnalazione speciale, chiamata SYN. Si aspetta di ricevere un pacchetto con segnalazione SYN-ACK e per concludere deve inviare un pacchetto con segnalazione ACK.

SYN flooding: assumendo che un server possa accettare al massimo n connessioni, l'attaccante manda continue segnalazioni SYN a cui poi non fa seguire gli ACK finali. Dopo n "chiamate" è in atto un DoS.



SYN spoofing: Utilizzando lo spoofing, l'attaccante può anche direzionare il DoS verso un altro utente, che verrà sommerso di segnalazioni SYN-ACK.

DNS

Il mondo dei servizi online non ragiona con gli indirizzi. Per questo motivo esiste un registro online distribuito chiamato **Domain Name System (DNS)**, che traduce indirizzi in nomi di dominio.

Typosquatting: registrare nomi di dominio simili a nomi legittimi, con lo scopo di ingannare gli utenti che sbagliano a scrivere il nome. Usato per ottenere informazioni private senza l'uso di social engineering.

Sniffing

Lo **sniffing** è una forma di eavesdropping (intercettazione). Può essere fatto (e normalmente viene messo in atto) da chiunque lungo il percorso verso la vostra destinazione.

In tutti i protocolli che abbiamo elencato è possibile fare sniffing:

- sniffing DNS rivela a che siti state facendo richiesta
- sniffing TCP rivela cosa state chiedendo al server
- ...

Wireshark è un analizzatore di protocolli di rete gratuito e open source. Permette di vedere cosa succede sulla rete a livello microscopico. È lo standard de facto in molti settori (istruzione compresa).

Anonimizzazione

Prendiamo in considerazione un caso semplice: l'invio di mail anonime.

Un anonymous remailer è un server che ricevuta una mail, con le informazioni sul destinatario, la inoltra al destinatario rimuovendo le informazioni del mittente. È il concetto alla base di VPN come NordVPN.

Un anonymous remailer è un esempio di Proxy. Il problema dei proxy è che hanno informazioni su di noi, quindi possono agire da eavesdropper.

Mix-based system

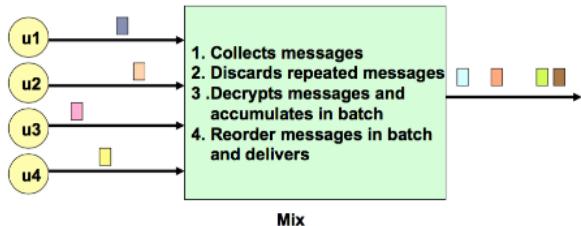
Inizialmente introdotto per e-mail anonime, è stato poi generalizzato al traffico TCP. Utilizza server relay (MIX) per comunicazioni anonime. Gli obiettivi di questo sistema sono l'anonimato del mittente e la non-linkability contro intercettatori globali. L'idea è che i messaggi dal mittente abbiano un contenuto diverso dai messaggi al destinatario.

Questo concetto ha avuto un impatto su altre idee come: onion routing, mixing del traffico, traffico dummy/fittizio (noto anche come traffico di copertura).

Un mix è un relay store-and-forward, che esegue due operazioni di base:

- Batching: raccogliere messaggi di lunghezza fissa da diverse sorgenti, fino ad accumulare un batch di n messaggi.
- Mixing: trasformare tramite crittografia i messaggi raccolti e inoltrare i messaggi ai loro destinatari in ordine casuale.

Ogni mix ha una chiave pubblica. Ogni mittente cifra il suo messaggio utilizzando la chiave pubblica del mix.



Questo sistema presenta alcuni svantaggi:

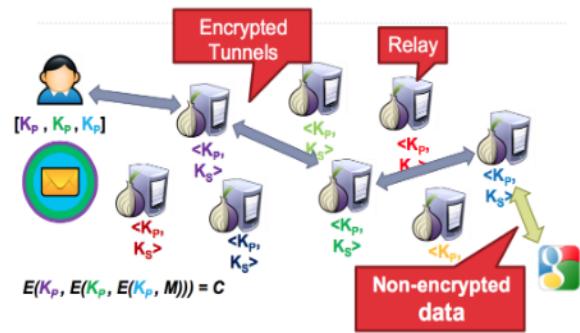
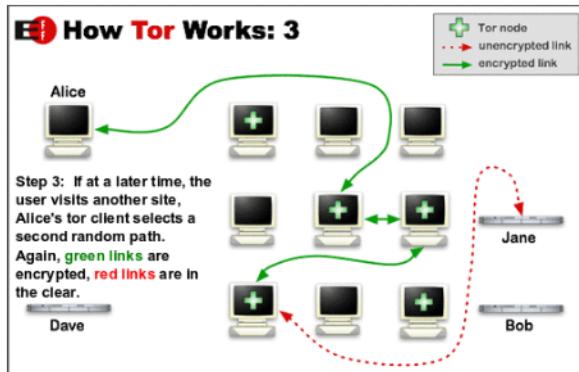
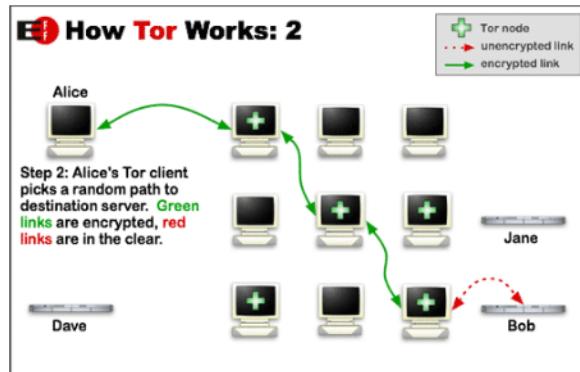
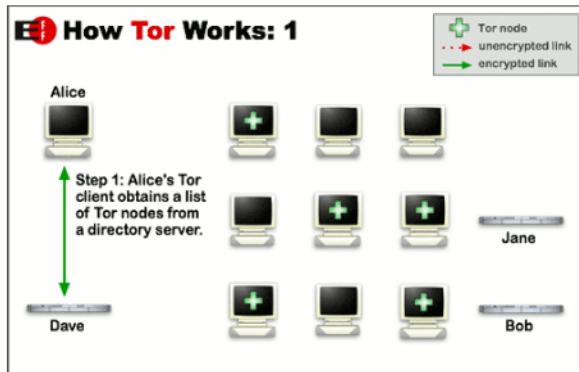
- La cifratura e decifratura a chiave pubblica ad ogni mix è computazionalmente costosa
- I Mixnet di base hanno un'elevata latenza, va bene per la posta elettronica ma non per la navigazione Web anonima

È possibile avere una rete anonima a bassa latenza?

1. Utilizzare la crittografia a chiave pubblica per stabilire un circuito con chiavi simmetriche a coppie tra i salti sul circuito
 2. Utilizzare la decifratura e ri-cifratura simmetrica per spostare i messaggi lungo i circuiti stabiliti
- Ogni nodo si comporta come un mix, quindi l'anonimato viene preservato anche se alcuni nodi sono compromessi.

Routing a cipolla

Per anonimizzare completamente il traffico è possibile usare il cosiddetto **Routing “a Cipolla” (Onion Routing)**. Il software Tor (The Onion Router) utilizza questi concetti per anonimizzare il traffico. I relay formano un circuito anonimo, in cui tutto il traffico è protetto con svariati livelli di cifratura.



Tor viene denominata come **rete overlay**. Una rete chiusa, al cui interno vengono distribuiti dati in forma anonima. Questo è il principio dei servizi onion.

Esistono alcuni servizi su Tor in grado di farvi uscire sulla rete "normale". Vengono chiamati **Exit Node**. Chi mantiene l'exit node vede tutto il vostro traffico. Inoltre, essendo pubblici sono normalmente bloccati da ogni servizio.

Illecità

A causa dell'anonymato offerto dai servizi onion, sia di chi visita che di chi fa hosting, è inevitabile lo sviluppo di servizi illegali al loro interno (compravendita di droga e armi, servizi di hacking, pornografia illegale...)

Ovviamente l'affidabilità di questi servizi è sempre in dubbio: quanti di questi possono essere esche? Questo non rende l'uso di Tor illegale, né automaticamente illegale ogni servizio al suo interno.

Data leak

Un comportamento molto presente sui servizi onion, soprattutto grazie alla compravendita illegale, è quello dei data leak. Un **data leak** è un rilascio di informazioni private di aziende, persone o oggetti.

Un data leak di tipo **Fullz** è una collezione di informazioni personali almeno contenenti il minimo indispensabile per creare conti correnti e/o pagare con carte di credito. Tramite questi leak è possibile attuare truffe molto più facilmente.

I leak più comuni rimangono quelli di account e password. In questo caso una lista di account viene messa online, sperabilmente (dal punto di vista dell'attaccante) con password in chiaro.

Esistono sistemi online in grado di allertarvi quando viene pubblicato un leak contenente il vostro account. Il più famoso è HavelBeenPwned.

Mentre HavelBeenPwned è gestito da un'organizzazione esterna, è possibile utilizzare alcuni software in grado di scandagliare la rete (crawler/spider) alla ricerca di leak contenenti determinate stringhe. Questi software sono in grado di scandagliare anche i principali mercati neri tramite insider.

Leak di anonimato

Tor non è perfetto. La rete è principalmente contraria all'anonimato. Esistono alcune problematiche che possono rivelare l'IP di chi sta accedendo a un determinato servizio. La prima su tutte è la geolocalizzazione.

Anche la richiesta di informazioni al DNS (soggetta a sniffing o controllata dal gestore) è in grado di rivelare a cosa state cercando di accedere.

Supponiamo di voler accedere ad un sito illecito dall'account Unibo. Una richiesta DNS per richiedere il sito potrebbe essere inviata fuori da Tor, quindi gli sniffer saprebbero che state cercando di accedere a quel sito.

Le informazioni rivelate non devono per forza essere direttamente collegate a vostre richieste, ma possono essere informazioni "lateralì" (side channel) a cui non avete pensato.

Rilasciare più informazioni del necessario o poter utilizzare più strumenti rispetto a quelli strettamente necessari è un problema concettuale di sicurezza. Dovreste limitare le capacità di un software al minimo indispensabile richiesto.

Per evitare queste problematiche esiste una versione di firefox modificata, già configurata per non rilasciare più informazioni del necessario (privilegio minimo). Tuttavia, queste problematiche possono sussistere anche a livello di sistema operativo. Esiste una distribuzione Linux chiamata Tails, la quale vi predispone il sistema per essere il più anonimo possibile.

Una distribuzione Linux è un insieme di software, configurato per un determinato scopo. Linux non è un sistema operativo, è un Kernel.

Sicurezza di sistema

Permessi, controllo degli accessi e Linux.

Controllo degli accessi

Una volta autenticati i soggetti, il problema successivo da affrontare è l'autorizzazione o il controllo degli accessi. Il **controllo degli accessi** è un insieme di politiche e meccanismi che servono per decidere se ad un particolare soggetto è consentito eseguire determinate operazioni su determinati oggetti. Gli obiettivi sono:

- impedire a utenti non autorizzati di accedere alle risorse
- impedire agli utenti legittimi di accedere alle risorse in modo non autorizzato
- consentire agli utenti legittimi di accedere alle risorse in modo autorizzato

La politica di controllo degli accessi stabilisce quali tipi di accesso sono consentiti, in quali circostanze e da chi. Gli elementi base del controllo degli accessi sono:

- **soggetto**: un'entità in grado di accedere agli oggetti
- **oggetto**: una risorsa di cui è necessario controllare l'accesso
- **diritto di accesso**: descrive il modo in cui un soggetto può accedere ad un oggetto

Sia S l'insieme dei soggetti, O l'insieme degli oggetti (che possono essere attivi ed agire da soggetti) e α l'insieme dei diritti di accesso che i soggetti hanno sugli oggetti.

Un **dominio di protezione** è un insieme di oggetti e diritti di accesso ad essi associati.

Formalmente è un insieme di tuple <oggetto, insieme-di-diritti>.

I soggetti operano all'interno di un determinato dominio di protezione.

L'associazione soggetto-dominio può essere statica o dinamica.

In un sistema operativo, *kernel mode* e *user mode* sono due domini di protezione che controllano l'accesso alla memoria principale. Normalmente i processi operano in *user mode*. Quando eseguono una system call passano in *kernel mode*, ottenendo i privilegi necessari per completare la chiamata. In questo esempio l'associazione soggetto-dominio è dinamica.

Le politiche per il controllo degli accessi possono essere di tre tipi:

1. Controllo discrezionale dell'accesso (Discretionary Access Control): DAC
2. Controllo di accesso obbligatorio (Mandatory Access Control): MAC
3. Controllo degli accessi basato sui ruoli (Role-Based Access Control): RBAC

Controllo discrezionale dell'accesso (DAC)

L'accesso è basato sull'identità dei soggetti e su regole di accesso, che stabiliscono cosa i soggetti possono (o non possono) fare su quali oggetti.

Discrezionale perché sono i soggetti che decidono di concedere (o negare) l'accesso ad altri soggetti.

Un approccio generale al DAC è quello di una **matrice di accesso**. Si tratta di una matrice M con i domini (a cui sono associati i soggetti) sulle righe e gli oggetti sulle colonne. Ogni cella $M(i, j)$ contiene l'insieme dei diritti di accesso α che il dominio D_i consente sull'oggetto O_j . Quando viene creato un nuovo oggetto si aggiunge una colonna alla matrice, il cui contenuto viene deciso dal creatore dell'oggetto.

		OBJECTS				
		File 1	File 2	File 3	File 4	
SUBJECTS		User A	Own Read Write		Own Read Write	
		User B	Read	Own Read Write	Write	Read
		User C	Read Write	Read		Own Read Write

Problema: due utenti A e B , nei rispettivi domini, modificano file diversi ($File_2$ e $File_3$) ma usano entrambi $File_4$, il quale è protetto e non deve essere copiato.

Soluzione: introdurre un dominio separato D_4 , in cui $File_4$ può solo essere letto, aggiungendo un diritto di accesso speciale chiamato *switch*. Lo *switch* cambia dominio e copia i diritti dal dominio sorgente a quello di destinazione, creando più istanze del dominio quando più utenti vi accedono.

	object domain	$File_1$	$File_2$	$File_3$	$File_4$	
A	D_1	read				
	D_2		read write	read	read	
	D_3			read write		

	object domain	$File_1$	$File_2$	$File_3$	$File_4$	D_4
A	D_1	read				
	D_2		read write	read		switch
	D_3			read write		switch

	object domain	$File_1$	$File_2$	$File_3$	$File_4$	D_4
A	D_1	read				
	D_2		read write	read		switch
	D_3			read write		switch

Questo meccanismo si implementa come una tabella globale, tramite una matrice 2D contenente . Il vantaggio è la semplicità, lo svantaggio è che può diventare enorme e difficile da mantenere in un sistema dinamico dove domini, oggetti e diritti cambiano frequentemente.

Controllo di accesso obbligatorio (MAC)

L'accesso è basato sul confronto delle etichette di sicurezza (security labels), che indicano quanto sono sensibili o critici gli oggetti, con le autorizzazioni di sicurezza (security clearances) dei soggetti.

Obbligatorio perché le etichette e i nulli di sicurezza sono impostati dal sistema e non possono essere modificati dai soggetti. I primi sistemi MAC erano basati sul modello Bell-LaPadula.

Controllo degli accessi basato sui ruoli (RBAC)

L'accesso è basato sui ruoli che i soggetti hanno all'interno del sistema e su regole che stabiliscono quali accessi sono consentiti ai soggetti in determinati ruoli.

Agli utenti vengono assegnati ruoli diversi, in modo statico o dinamico, in base alle loro responsabilità.

Possiamo usare una matrice per associare i singoli utenti ai ruoli. Una cella della matrice contrassegnata indica che l'utente è assegnato al rispettivo ruolo. In generale, gli utenti sono molti più dei ruoli. A un singolo utente possono essere assegnati più ruoli, più utenti possono essere assegnati ad uno stesso ruolo.

	R ₁	R ₂	• • •	R _n
U ₁	X			
U ₂	X			
U ₃		X	X	
U ₄			X	
U ₅			X	
U ₆			X	
⋮				
U _m	X			

Principi fondamentali per le politiche di sicurezza

- **Open design:** la sicurezza del sistema non deve basarsi sulla segretezza della sua progettazione o implementazione.
- **Economy of mechanism (as simple as possible):** questo principio semplifica la progettazione e l'implementazione dei meccanismi di sicurezza. Se tali procedure sono semplici, esistono meno possibilità di errori. Il processo di controllo e test è meno complesso. Le interfacce tra i moduli di sicurezza sono aree sospette e dovrebbero essere il più semplici possibile.
- **Fail-safe defaults:** di default, i soggetti non hanno privilegi di accesso su nessun oggetto. Questo principio afferma che ad un soggetto dovrebbe essere negato l'accesso ad un determinato oggetto, a meno che non gli sia stato concesso esplicitamente (l'accesso).
- **Complete mediation (reference monitor):** non è possibile accedere direttamente agli oggetti, tutti gli accessi devono essere controllati.
- **Least privilege:** i soggetti dispongono dei privilegi di accesso minimi, necessari per effettuare le operazioni in una determinata fase di esecuzione. Questo significa che ogni soggetto dovrebbe operare utilizzando il set minimo di privilegi (diritti di accesso) necessari per svolgere una sua task/operazione.
Questo principio limita i danni che possono derivare da un incidente o da un errore, limita il numero di programmi privilegiati, aiuta nel debug, aumenta la sicurezza, consente l'isolamento dei sotto-sistemi critici.

Il privilegio minimo viene imposto attraverso un **reference monitor** che implementa una completa mediazione: ogni accesso a ogni oggetto viene controllato.

Con RBAC ogni ruolo dovrebbe disporre del set minimo di diritti di accesso necessari per quel ruolo. A un utente/soggetto viene assegnato un ruolo che gli consente di svolgere solo le attività necessarie. Più utenti assegnati allo stesso ruolo godono dello stesso set minimo di diritti di accesso.

Sicurezza dei sistemi

La sicurezza dei sistemi differisce da quella delle reti:

- possibilità di avere un ente certificato nel mezzo (SO)
- i sistemi possono essere singolo o multi-utente
- possibilità di divisione dei privilegi

Le minacce per la sicurezza dei sistemi provengono dai malware locali.

Si definisce **privilege escalation** la possibilità di ottenere più privilegi sul sistema, come la possibilità di installare programmi, leggere informazioni private o modificare configurazioni del sistema. Solitamente avviene tramite lo sfruttamento di una falla, di un errore di configurazione di un software applicativo o di un sistema operativo.

Il primo passo per il privilege escalation è l'**esecuzione arbitraria di codice** (per poter effettuare attacchi per ottenere privilegi più elevati). Questo può essere ottenuto installando o eseguendo software malevolo, sfruttando vulnerabilità in grado di dirottare il funzionamento dei programmi, ecc. In UNIX tutto è un file (più o meno). Questo vuol dire che avendo controllo su un file si può ottenere il controllo su ciò che rappresenta.

Access Control List (ACL)

Il sistema mantiene le informazioni di accesso ai file tramite un meccanismo denominato **Access Control List (ACL)**. Queste liste vengono rappresentate sulle colonne della tabella (matrice), dove ad ogni oggetto è associata una lista di diritti di accesso per i rispettivi domini.

Le ACL sono mantenute dagli oggetti!!!

ACL for o ₂			
	O ₁	O ₂	O ₃
S ₁	RW	RX	
S ₂	R	RWX	RW
S ₃		RWX	

Alcune ottimizzazioni permettono di ridurre la lunghezza delle liste:

- includere solo i domini che hanno diritti di accesso diversi da quelli default
- raggruppare i domini in piccoli insiemi e definire i diritti di accesso su di essi

UNIX/Linux (senza le estensioni Posix ACL), dichiara 3 soggetti (domini):

- **user**: il proprietario di un file
- **group**: il gruppo proprietario di un file
- **other**: tutti gli altri

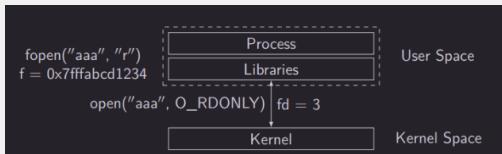
I soggetti sono identificati tramite un file (`/etc/passwd`), il quale contiene un'associazione nome utente:user id. Lo user id viene quindi salvato nel file-system, associato ad ogni file. Analogamente, per i gruppi, esiste un file (`/etc/group`) e un group id per gruppo.

Normalmente in UNIX (sistemi multi-utente) è possibile "regalare" l'accesso a file ad altri utenti (con `chmod`).

Capabilities

Una **capability** è un token o un ticket in grado di rappresentare un determinato oggetto sul quale si ha l'accesso. In questo caso si ragiona per riga, ovvero per soggetto/dominio: ogni capability specifica per un oggetto le operazioni autorizzate per un particolare dominio. Ogni utente dispone di un numero di ticket e può essere autorizzato a prestarli o cederli ad altri. Le capabilities sono mantenute dall'utente, e vengono presentate dai processi utente per ottenere l'accesso agli oggetti. L'integrità della capability deve essere protetta e garantita, solitamente dal sistema operativo. In particolare, il ticket deve essere inconfondibile, quindi la capability viene firmata con la chiave privata dell'oggetto per attestarne l'integrità e la validità.

Un esempio di capability sono i file aperti. A seguito di una system call `open`, viene ritornato un identificativo univoco (file descriptor), che può essere riconosciuto dal sistema operativo per indicare il file. Questo file descriptor (che non può essere alterato) può essere usato per effettuare operazioni sul file, presentandolo come una capability, ovvero una garanzia di possesso dei diritti di accesso.



UNIX utilizza un sistema ibrido tra ACL (file non aperti) e capabilities (file aperti, socket, ...), al fine di ottenere i vantaggi di entrambi:

- controllo degli accessi basato sull'identità (ACL)
- facilità di revoca (ACL)
- efficienza negli accessi (capabilities)

Access Control Lists (ACLs)
Encode columns of an access control matrix

	O1	O2	O3
S1	RW	RX	
S2	R	RWX	RW
S3		RWX	

Capabilities
Encode rows of an access control matrix

	O1	O2	O3
s1	RW	RX	
s2	R	RWX	RW
s3		RWX	

Revoca dei diritti di accesso

La revoca dei diritti può essere:

- immediata o ritardata
- selettiva o generale
- parziale o totale
- temporanea o permanente

Nei sistemi basati su ACL è facile revocare i diritti, basta modificare la lista associata all'oggetto. Nei sistemi basati su capability è più complesso, poiché i diritti sono distribuiti ai processi e non centralizzati, quindi bisogna prima localizzarli. Alcune soluzioni per questo problema:

- capability a tempo limitato: ogni capability ha una scadenza, al termine della quale si rinnova oppure si considera revocata
- capability indirette: puntano a una tabella intermedia anziché direttamente agli oggetti, quindi modificando la tabella è possibile revocare

Modelli di scambio informazioni

Se volessimo evitare il regalo degli accessi da parte dell'utente *owner* potremmo utilizzare un sistema di Mandatory Access Control (MAC).

Bell-LaPadula (BLP)

BLP è stato sviluppato come modello formale per il controllo degli accessi. Un modello formale mira a dimostrare, logicamente o matematicamente, che un particolare design/progetto soddisfa un insieme stabilito di requisiti di sicurezza e che l'implementazione di quel progetto è fedelmente conforme alle sue specifiche.

Nel modello BLP, ad ogni soggetto e ad ogni oggetto viene assegnata una classe di sicurezza. Nella formulazione più semplice, le classi di sicurezza formano una rigida gerarchia e vengono chiamate livelli di sicurezza.

Esempio (schema di classificazione militare statunitense):

top secret > secret > confidential > restricted > unclassified

Esistono diversi modelli di flusso delle informazioni. Supponiamo di avere diversi file a diversi livelli di segretezza. Il modello Bell-LaPadula mette in atto le seguenti regole (Write-Up-Read-Down):

- Ogni soggetto e ogni oggetto hanno un livello di sicurezza
- Un soggetto può leggere oggetti a livelli di sicurezza minori o uguali al suo (no read up)
- Un soggetto può scrivere oggetti a livelli di sicurezza pari o maggiori del suo (no write down)

Biba

Biba è un modello duale a Bell-LaPadula (Write-Down-Read-Up):

- Ogni soggetto e ogni oggetto hanno un livello di sicurezza
- Un soggetto può scrivere oggetti a livelli di sicurezza minori o uguali al suo
- Un soggetto può leggere oggetti a livelli di sicurezza pari o maggiori del suo

Usato per l'integrità dei dati. Nel mondo Windows prende il nome di Integrity Level (IL).

Filesystem

Nei sistemi operativi, il sistema per lo scambio dei dati locali è il **filesystem**. Su Linux si compone di una radice (/) dalla quale il sistema si dirama come un albero.

```
/      The root filesystem
/proc  (pseudo) The process virtual infrastructure (e.g. /proc/1/cmdline).
/sys   (pseudo) The system virtual infrastructure (e.g. /sys/class/leds).
/dev   (pseudo) Device drivers file interface (e.g. /dev/sda).
/run   (pseudo) Contains run-time information.
/etc   Configuration directory.
/tmp   Temporary directory, volatile, usually mounted in RAM.
/root  Root's home.
/bin   Binaries.
/lib   Libraries.
/sbin  System Binaries.
/usr   User binaries.
/var   Log, cache, and structured personal files (e.g. mailboxes).
/home  Users' directories.
/mnt   External mountpoint.
/opt   Optionals softwares.
```

Permessi

Il sistema in modalità DAC, senza meccanismi MAC, analizza in ordine UID processo e proprietario del file, GID processo e gruppo del file, ...

In ambienti con molti utenti e risorse, gestire manualmente i permessi può diventare complesso e soggetto a errori. Questo può generare complicanze contro-intuitive.

Il super-utente privilegiato del sistema è quello che può svolgere ogni operazione a partire dalla radice (root) del filesystem. Questo utente prende il nome di **root** (SYSTEM in Windows).

È possibile cambiare il proprietario o il gruppo di appartenenza di un file, tramite i comandi `chown` e `chgrp` rispettivamente. Solo root (normalmente) può "regalare" i file agli altri.

È possibile anche modificare i permessi associati a un file, attraverso il comando `chmod`.

Altri comandi interessanti sono `getfacl` (get file access control list) e `setfacl` (set file access control list).

Sulle cartelle (directories) i permessi hanno un significato lievemente diverso:

- execute = possibilità di entrare in una cartella
- write = possibilità di eliminare i file contenuti in una cartella (anche quelli non nostri su cui non abbiamo permesso di scrittura)
- read = possibilità di leggere il contenuto di una cartella

Un utente può appartenere a più gruppi. Questi vengono usati di solito (ma non solo) per controllare accessi a file di device driver (/dev).

Permessi speciali

Esistono 3 permessi speciali, salvati nei bit più significativi dei metadati del file:

- setuid
- setgid
- sticky

Una directory con permesso *setuid* verrà ignorata, mentre *setgid* assegnerà ai file creati al suo interno il gruppo d'appartenenza della directory. Questo è utile per mantenere all'interno di una directory il gruppo corretto per tutti i file. Lo *sticky bit* su una directory invece permetterà l'eliminazione dei file al suo interno solo al suo owner (anche se la directory ha permesso di scrittura per tutti).

Lo *sticky bit* su un file è ignorato e deprecato. Un file *setuid* verrà eseguito come se fosse eseguito dal suo proprietario. Analogamente, un file *setgid* eseguirà con il gruppo di appartenenza.

Quindi *setuid* e *setgid* consentono a un utente di eseguire un programma con i privilegi del proprietario/gruppo del file, cambiando dinamicamente il dominio di protezione per la sola durata dell'esecuzione.

Se un file eseguibile di proprietà di root è stato configurato con il bit *setuid* o *setgid*, un altro utente potrebbe abusarne per ottenere l'accesso all'utente root. Molti degli eseguibili con questo bit di autorizzazione impostato possono essere facilmente sfruttati per ottenere l'accesso all'utente root.
ESTREMAMENTE PERICOLOSO!

Cosa succede, ad esempio, se usiamo il seguente codice C?

```
system("whoami");
```

Il sistema eseguirà il comando `whoami` con i privilegi dell'utente indicato. `whoami` è un comando shell, che può quindi essere dirottato cambiandogli il "nome" (`PATH=. ./vulnprogram`).

In questo modo funziona il comando `sudo`: esegue con privilegi elevati e controlla la password dell'utente (normalmente tramite un framework chiamato PAM).

`setuid` è un ovvio problema di sicurezza: per ottenere uno dei privilegi di root, otteniamo l'intero insieme dei suoi privilegi. Per questo motivo sono stati spezzettati i privilegi di root in vari sotto privilegi (capabilities).

Proprietà di file

Gli utenti e i gruppi sono identificati tramite degli interi memorizzati nel file delle password.

Ogni processo creato dall'utente (per eseguire comandi) eredita il suo *user-id* e *group-id* rispettivamente in *real-user-id* e *real-group-id*. Quando un processo crea un nuovo file, il suo utente e gruppo proprietari sono impostati al *real-user-id* e *real-group-id* del processo creante.

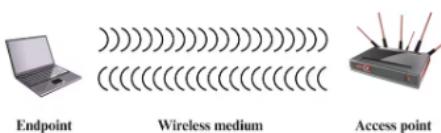
Ogni processo ha numerosi *id* associati:

- *real-user-id* e *real-group-id*: identificano l'utente ed il gruppo che ha lanciato il processo, sono letti dal file delle password e non cambiano durante l'esecuzione del processo.
- *effective-user-id* e *effective-group-id*: impostati dinamicamente durante l'esecuzione del processo tramite il meccanismo di `setuid`, sono usati per determinare i diritti di accesso durante l'interazione con il filesystem.
- *saved-user-id* e *saved-group-id*: contengono copie dei *effective-user-id* e *effective-group-id* esistenti al momento dell'utilizzo di `setuid`, permettendo al processo di tornare ad essi una volta che l'esecuzione di `setuid` termina.

Sicurezza di reti wireless

L'ambiente wireless è costituito da tre componenti che forniscono il punto di attacco:

- **client wireless**: può essere un telefono cellulare, un laptop, un sensore wireless...
- **punto di accesso wireless** (Access Point): fornisce una connessione alla rete o al servizio, ad esempio un ripetitore, un hotspot Wi-Fi...
- **mezzo di trasmissione**: trasporta le onde radio per la trasmissione dei dati



Ci sono alcuni fattori chiave che contribuiscono ad un rischio di sicurezza più elevato delle reti wireless rispetto alle reti cablate/fisse:

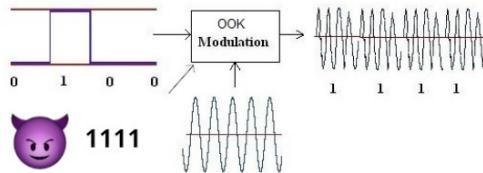
- **il canale wireless**: le reti wireless in genere prevedono comunicazioni broadcast, molto più soggette a intercettazioni e interferenze rispetto alle reti cablate. Le reti wireless sono anche più vulnerabili agli attacchi attivi, che sfruttano le vulnerabilità dei protocolli di comunicazione.
- **la mobilità**: i dispositivi wireless sono solitamente più portatili e mobili dei dispositivi cablati. Questa mobilità comporta una serie di rischi di sicurezza.
- **le risorse**: alcuni dispositivi wireless, come smartphone e tablet, dispongono di risorse di memoria ed elaborazione limitate per contrastare le minacce, tra cui attacchi DoS e malware.
- **l'accessibilità**: alcuni dispositivi wireless, come sensori e robot, possono essere lasciati incustoditi in luoghi remoti, aumentando notevolmente la loro vulnerabilità agli attacchi fisici.

Minacce alla rete wireless

- **Associazione dannosa**: un dispositivo wireless viene configurato in modo da apparire come un punto di accesso (AP) legittimo, consentendo all'operatore di rubare le password degli utenti e quindi di penetrare in una rete cablata attraverso un wireless AP legittimo.
- **Reti ad hoc**: si tratta di reti peer-to-peer tra computer wireless, senza alcun Access Point tra di essi. Tali reti possono rappresentare una minaccia per la sicurezza a causa della mancanza di un punto di controllo centrale.
- **Reti non tradizionali**: le reti e le connessioni non tradizionali, come i dispositivi Bluetooth, gli scanner di codici a barre, ecc, rappresentano un rischio per la sicurezza sia in termini di intercettazione che di spoofing.
- **Furto di identità (MAC spoofing)**: un attaccante è in grado di intercettare il traffico di rete e identificare l'indirizzo MAC di un computer con privilegi di rete.
- **Man-in-the-Middle**: ingannare un utente e un punto di accesso, facendogli credere di comunicare tra loro, quando in realtà la comunicazione avviene tramite un dispositivo di attacco intermedio. Le reti wireless sono particolarmente vulnerabili a questo tipo di attacco.
- **Denial of Service (DoS)**: nel contesto di una rete wireless, un DoS si verifica quando un attaccante bombarda continuamente un punto di accesso wireless o un'altra porta wireless accessibile con vari messaggi di protocolli che consumano risorse di sistema. L'ambiente wireless è vulnerabile a questi DoS perché per l'aggressore è molto facile indirizzare più messaggi verso il target.
- **Iniezione di rete (Network Injection)**: prende di mira i punti di accesso wireless esposti a traffico di rete non filtrato (messaggi di protocolli di routing o di gestione della rete). Ad esempio utilizzare comandi di riconfigurazione fasulli per influenzare router e switch, degradando le prestazioni della rete.

Jamming (DoS)

Il **jamming** è l'atto di disturbare le comunicazioni wireless, rendendo impossibile trasmettere o ricevere i dati, tipicamente trasmettendo sulla stessa frequenza e con la stessa modulazione del segnale che si vuole disturbare.



Questo attacco richiede parecchia energia da parte del Jammer. Per renderlo meno efficace, l'idea è di usare più frequenze portanti (banda larga), in modo che sia necessaria per il Jammer ancora più energia.

Un meccanismo per utilizzare la banda larga è il cosiddetto **Frequency Hopping Spread Spectrum (FHSS)**: si selezionano più frequenze portanti e si salta da una all'altra con una sequenza decisa a priori. Nel caso ci sia una collisione o un Jamming su una singola portante, ci sarà la perdita solo di quella parte di informazione.

Eavesdropping

Eavesdropping e **sniffing** sono molto semplici da perpetrare in ambito radio (sapendo il seed del PRNG dell'eventuale FHSS). È necessario:

- un ricevitore (più antenna) alla corretta distanza per ricevere il segnale
- conoscere la modulazione utilizzata
- conoscere il protocollo utilizzato

Replay attack

Gli **attacchi di replay** sono effettuabili senza informazioni come la modulazione utilizzata o il protocollo utilizzato. È sufficiente catturare una porzione di frequenze (spettro) e ritrasmetterle così come ricevute.

Il canale radio non è protetto da questo tipo di attacchi con i meccanismi di confidenzialità o integrità. Se inoltriamo un messaggio cifrato e integro che sappiamo servire per effettuare un'operazione, il messaggio verrà considerato valido.

Utilizzato con i cancelli automatici: catturando gli impulsi OOK (On-Off Keying) sulla frequenza corretta, è possibile re-inviare il messaggio al cancello per farlo aprire. Questa operazione non richiede la comprensione del contenuto degli impulsi.

Per una maggiore sicurezza si utilizza un meccanismo chiamato **Rolling Code**:

1. il trasmettitore seleziona un codice basandosi su un PRNG e lo invia
2. il trasmettitore seleziona il codice successivo, basandosi sul PRNG
3. il ricevitore controlla che il codice ricevuto sia consistente con il suo PRNG, nel caso effettua l'operazione e fa avanzare il PRNG

Se un codice viene trasmesso e non ricevuto, si ha un disallineamento del PRNG, quindi non viene più effettuata l'operazione. Per questo motivo, il ricevitore controlla una finestra di codici successivi a quello abilitato (supponiamo 100), poi sposta la finestra di conseguenza.

Questo meccanismo non impedisce gli attacchi di replay, però li rende meno efficienti, in quanto l'attaccante dovrà catturare vari codici, terminati i quali dovrà ricatturarne altri dal trasmettitore. Il problema di questi metodi è la mancanza di un canale di ritorno/feedback: i ricevitori non dialogano con il trasmettitore e si limitano a ricevere.

Tramite dei *transceiver* è possibile effettuare quello che viene chiamato **Challenge and Response**.

Presente anche in vari protocolli applicativi, funziona nel seguente modo:

1. l'autenticando (trasmettitore, nei casi precedenti) richiede di accedere al sistema tramite un messaggio
2. l'autenticatore (ricevitore, precedentemente) genera un nonce e lo invia all'autenticando
3. l'autenticando cifra il nonce e re-invia il valore cifrato all'autenticatore
4. l'autenticatore decifra il valore cifrato e valida o meno l'autenticazione

Ovviamente la cifratura può essere applicata, oltre che ai meccanismi di autenticazione, per mantenere la confidenzialità e l'integrità dei dati in transito. Nel mondo radio questa viene implementata tramite cifrari a blocco o a flusso.

Radio Frequency Identification (RFID)

Un altro sistema radio prende il nome di **Radio Frequency IDentification (RFID)**. L'identificazione a radiofrequenza (RFID) utilizza i campi elettromagnetici per identificare e tracciare automaticamente i tag attaccati agli oggetti.

Un sistema RFID è costituito da un minuscolo *tag*, un *ricevitore radio* e un *trasmettitore*. Quando viene attivato da un impulso di interrogazione proveniente da un lettore RFID nelle vicinanze, il tag trasmette dati digitali (il suo codice identificativo e/o altre eventuali informazioni contenute) al lettore. Questo sistema è normalmente utilizzato per autenticare tramite badge, telefoni o portachiavi.

I badge Unibo utilizzano una tecnologia chiamata *EM410X*, che prevede un ID interno univoco per ogni badge, che viene assegnato dal database Unibo al vostro account.

In questo modo, i portali in cui si richiede l'autenticazione effettueranno una ricerca sul database per controllare se il vostro account è autorizzato per passare un determinato varco.

I badge *EM410X* vengono venduti senza la possibilità di modificare il loro ID univoco. Per aggirare questa protezione, si comprano dei badge che hanno questa possibilità e si effettua la **clonazione**.

Se non fossero presenti badge "liberi" per una determinata tecnologia, in assenza di meccanismi crittografici, è possibile spacciarsi per un badge con un transceiver RFID (**spoofing**). L'attaccante intercetta i segnali legittimi emessi dai tag RFID nella comunicazione con i lettori, poi utilizza un dispositivo specializzato per imitare questi segnali.

IEEE 802.11

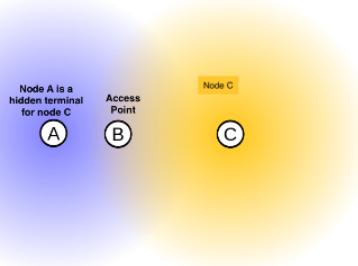
Una delle comunicazioni radio più influenti nel mondo informatico è lo standard **IEEE 802.11**, comunemente chiamato **Wi-Fi**. Esso è uno standard pensato per creare reti locali (Wireless Local Area Networks, WLAN) interoperabili con reti Ethernet. Lo standard è diviso in vari sotto-standard (802.11g per le reti 2.4GHz, 802.11ac per le reti 5GHz, ...). A seconda dello standard di riferimento, si utilizzano diversi livelli fisici e modulazioni (per accesso al canale principalmente).

Accesso alla rete

A livello fisico non vengono poste protezioni normalmente. Essendo le onde radio propagate nello spazio, esistono soluzioni per isolare spazialmente gli ambienti.

Le reti 802.11, essendo radio, sono sensibili alle collisioni. Per questo motivo, c'è bisogno di un protocollo di accesso al canale tra i dispositivi, in modo da risolvere eventuali collisioni: **Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA)**.

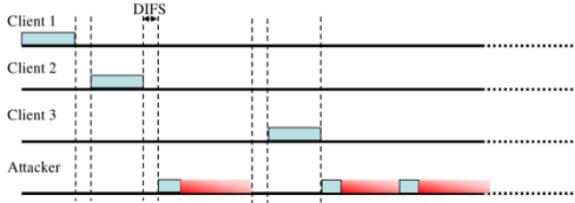
Inoltre, non è sempre possibile vedere tutti i nodi: l'access point ha normalmente visibilità dell'intera rete (se singolo), ma i singoli nodi no. Questo problema prende il nome di **Terminale Nascosto** (hidden terminal problem).



Inter-Frame Space (IFS)

Il problema della collisione viene "risolto" accorgendosi che la collisione è avvenuta ed aspettando un tempo random prima di ritrasmettere il messaggio. Il tempo è *DIFS* + *backoff* (un numero casuale estratto tra 1 e *CW*, ovvero Finestra di Congestione).

Il protocollo prevede quindi degli spazi di silenzio per evitare la collisione, che devono essere rispettati. La gestione di questi spazi è particolarmente complessa con diverse classificazioni (spazi che può aspettare solo l'AP, spazi dedicati ai client, ...) e prendono il nome di **IFS (inter frame space)**.



Se un singolo terminale non rispetta lo spazio di silenzio successivo a una collisione e non aspetta nessun IFS, parlerà sempre e solo lui (**appropriazione della rete**). Se invece sono due i terminali a fare ciò, nessuno può più parlare (**Denial of Service**).

Proteggere IEEE 802.11

SSID hiding

Un primo meccanismo di sicurezza è nascondere la rete agli occhi di un attaccante. Per accedervi bisognerà conoscerne il nome, che viene inviato in chiaro dagli host che richiedono l'accesso. Essendo un meccanismo di security by obscurity, non è efficace.

MAC whitelisting

A livello MAC/2, i terminali IEEE 802.11 utilizzano degli indirizzi MAC, esattamente come le schede di rete Ethernet.

Un meccanismo di sicurezza potrebbe essere quello di abilitare solo alcuni indirizzi MAC (MAC white list), in modo da vincolare la rete solo a quelli.

Il problema di questo approccio è che l'indirizzo MAC è inviato in chiaro nella comunicazione IEEE 802.11. È quindi possibile effettuare spoofing, a seguito di una fase di sniffing, spacciandosi per un host in whitelist.

Cifratura

L'autenticazione alla rete e la confidenzialità sono cruciali nelle reti wireless, in quanto per fare eavesdropping non è richiesto il Man-in-the-Middle. In una rete *open*, le informazioni vengono inviate nell'etere in chiaro, quindi tutti le possono leggere anche senza essere associati alla rete. Per questo motivo esistono diversi meccanismi di protezione basati sulla crittografia.

Nonostante i meccanismi di cifratura e accesso alla rete, IEEE 802.11 prevede messaggi che vengono inviati in chiaro, senza nessun meccanismo di autenticazione, confidenzialità o anti-replay.

Esiste un messaggio di "disassociazione a una rete", tramite cui un terminale elimina l'associazione con un determinato access point.

Deauthentication/Dissociation attack: l'attaccante invia un messaggio di de-autenticazione/disassociazione al posto di un utente, causando disservizio e perdita dei dati.

Wired Equivalent Privacy (WEP)

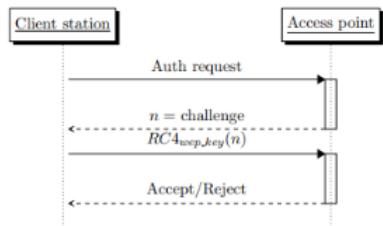
La prima forma di autenticazione e confidenzialità dello standard 802.11 è un meccanismo che prende il nome di **Wired Equivalent Privacy** (non sicuro!). Esso presenta due modalità di funzionamento:

- Shared Key
- Open System

Shared Key

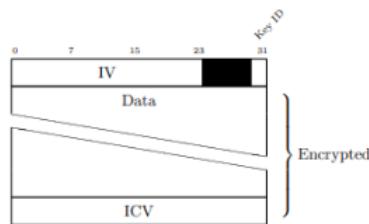
In questo caso, tutti i dispositivi nella rete condividono una chiave segreta. Per effettuare l'autenticazione, il client deve dimostrare all'access point di possedere tale chiave, utilizzando un meccanismo di challenge/response.

Questo metodo non è sicuro, in quanto si può attaccare con attacchi KPA (known plaintext), ricavando la chiave da tutte le autenticazioni.



Open System

In questo caso, non c'è una verifica del possesso della chiave segreta tramite l'utilizzo di un challenge. Il client che richiede la connessione è già a conoscenza del segreto comune, ovvero la chiave condivisa, altrimenti non sarebbe in grado di decifrare i pacchetti cifrati provenienti dall'access point (anche i suoi pacchetti verrebbero scartati dall'AP che li riceve).



Questo metodo utilizza **RC4** come algoritmo di cifratura. RC4 è un cifrario a flusso, che basandosi sulla chiave k e sul vettore di inizializzazione (IV) v , genera un keystream (flusso di chiavi) $RC4(v, k)$. Per inviare un messaggio M da Alice a Bob:

1. calcolare il checksum di integrità $c(M)$
2. plaintext $P = \{ M, c(M) \}$
3. cifrare P usando RC4: ciphertext $C = P \oplus RC4(v, k)$
4. trasmettere $C' = v, C$

Per decifrare un messaggio C' , il processo di crittografia viene invertito.

Dopo 30000 pacchetti trasmessi dalla rete, le probabilità di collisione sono praticamente impossibili da evitare, perciò catturando pacchetti con lo stesso IV è possibile fare attacchi statistici. Inoltre, catturando pacchetti con un IV noto, è possibile far circolare in rete pacchetti vecchi aumentando il traffico (replay attack).

Wi-Fi Protected Access (WPA/WPA3)

Per ovviare a questi (e altri) problemi di WEP, è stato sviluppato **Wi-Fi Protected Access (WPA)**.

Questo standard, arrivato alla versione 3, pone diversi modi di utilizzo per il canale:

- PSK (Pre-Shared Key) → per reti domestiche, con chiave segreta su ogni dispositivo
- Enterprise → per reti con molti utenti (come ALMAWIFI)
- WPS (Wi-Fi Protected Setup) → sistema di autenticazione facilitato

Lo standard **IEEE 802.11i** definisce due schemi per la protezione dei dati trasmessi nelle MPDU (MAC Protocol Data Unit):

- Temporal Key Integrity Protocol (TKIP), compatibile con WEP, deprecato
- Counter Mode-CBC MAC Protocol (CCMP), basato su AES



Figure: General IEEE 802 MPDU Format

Temporal Key Integrity Protocol (TKIP)

TKIP è progettato per richiedere solo modifiche software ai dispositivi implementati con WEP e fornisce due servizi:

- Integrità del messaggio: TKIP aggiunge un codice di integrità del messaggio (MIC) al frame MAC, dopo il campo *Dati*. Il MIC è un valore a 64 bit calcolato a partire dai valori dell'indirizzo MAC di origine e destinazione e il campo *Dati*, oltre al materiale della chiave.
- Confidenzialità dei dati: viene garantita cifrando il valore MPDU + MIC utilizzando RC4.

Il **Temporal Key (TK)** di 256 bit viene spezzato in due per l'utilizzo. Due chiavi, di 64 bit l'una, vengono usate per produrre il codice di integrità MIC: una chiave per proteggere i messaggi da client ad AP e l'altra per proteggere i messaggi da AP a client. I restanti 128 bit vengono troncati per generare la chiave RC4, utilizzata per crittografare i dati trasmessi.

Un **contatore di sequenza** (sequence counter) di TKIP, detto **TSC**, viene assegnato a ciascun frame per una protezione aggiuntiva. Il TSC ha praticamente due scopi:

- è incluso in ogni MPDU per proteggere il MIC dagli attacchi di replay
- è combinato con la TK della sessione per produrre una chiave dinamica che cambia ad ogni MPDU trasmesso, rendendo più difficile la crittoanalisi

Counter Mode-CBC MAC Protocol (CCMP)

CCMP è destinato ai dispositivi più recenti, dotati dell'hardware per supportare tale schema. CCMP fornisce due servizi:

- Integrità del messaggio: CCMP utilizza il codice di autenticazione del messaggio basato su Cipher Block Chaining (CBC), CBC-MAC.
- Confidenzialità dei dati: CCMP utilizza la modalità operativa CTR con AES per la crittografia.

La stessa chiave AES a 128 bit viene utilizzata sia per l'integrità che per la confidenzialità dei dati. CCMP utilizza un numero di pacchetto (packet counter) a 48 bit per costruire un nonce, al fine di prevenire attacchi di tipo replay.

Attacchi a WPA

L'impostazione del nome della rete (SSID) non è crittografato. Questo rende possibili attacchi di tipo **Rogue Access Point**, in grado di effettuare spoofing del nome della rete, invitando ad accedere i client, che si troveranno impossibilitati utilizzando una password diversa da quella configurata.

Le reti **PSK** soffrono degli stessi problemi legati al meccanismo di autenticazione. È possibile perpetrare attacchi brute-force o a dizionario (come per le password) per indovinare la chiave.

WPS è una semplificazione del metodo d'accesso alla rete Wi-Fi, senza necessità di password, che attua una sorta di accesso fisico al dispositivo. Abilita diverse metodologie d'accesso, come un tasto fisico oppure un codice PIN. Il tasto fisico può essere facilmente aggirato tramite un rogue access point posto in una posizione tattica.

Il meccanismo di login tramite PIN invece comporta un grosso problema: il PIN è normalmente un numero di 7 cifre, quindi l'aggressore riesce a recuperare il codice in poche ore con un attacco brute-force, e tramite il PIN WPS, successivamente la chiave WPA della rete. Per qualche scelta implementativa, il PIN viene controllato dal sistema prima per le prime 4 cifre del codice, poi per le altre 3. Visto che le successive 3 cifre vengono controllate solo se le prime 4 sono corrette, i tentativi di brute-force necessari diventano $10000 + 1000, 11000$ tentativi, circa 20 ore.

Inoltre, l'implementazione di WPS su molti dispositivi embedded utilizzava un generatore di nonce predicable, portando questo brute-force a un paio di minuti, catturando e analizzando la comunicazione.

Krack è un famoso attacco di replay che sfrutta una vulnerabilità in WPA2. Nella fase di autenticazione della rete viene utilizzato un nonce che può essere riusato identico per velocizzare le autenticazioni successive. Questo comporta il poter reinstallare chiavi vecchie, in modo da poter analizzare facilmente la comunicazione e risalire alla chiave di cifratura. Grazie a questo attacco lo standard è stato ulteriormente modificato, arrivando a WPA3.

Buffer Overflow

Il software è noto per avere **bug**, ad esempio funzionalità che non funzionano come previsto (o non funzionano affatto) oppure crash che causano inaffidabilità e perdita di dati. Per un attaccante, i bug nel software sono delle opportunità per:

- Aggirare i controlli di autenticazione e autorizzazione
- Elevare i privilegi (ad admin o root)
- Dirottare i programmi per eseguire codice non intenzionale/arbitrario
- Consentire l'accesso persistente e non autorizzato ai sistemi

Con il termine **exploit** intendiamo utilizzare gli errori di programmazione a vantaggio di un utente malintenzionato.

Buffer overflow, noto anche come *buffer overrun* o *buffer overwrite*, è definito come "una condizione su un'interfaccia in cui è possibile inserire più input in un buffer o un'area di conservazione dei dati rispetto alla capacità assegnata, sovrascrivendo altre informazioni. Gli avversari sfruttano tale condizione per mandare in crash un sistema o per inserire codice appositamente predisposto che consente loro di ottenere il controllo del sistema."

Esecuzione del programma

I computer non eseguono codice sorgente (Java, C++, ...), eseguono codice macchina (machine code). I compilatori traducono il codice sorgente in codice macchina. L'assembly è un codice macchina leggibile dall'uomo.

Esempio:

The screenshot shows a debugger interface with three panes. The left pane displays the C source code for a "Hello World" program. The middle pane shows the corresponding x86-64 machine code in hexadecimal. The right pane shows the assembly code generated by the compiler. The assembly code includes instructions like push rbp, mov rbp, rsp, mov rbp, rbp, mov DWORD PTR [rbp-0x14], edi, and so on, which correspond to the machine code bytes.

```
#include <stdio.h>
int main(int argc, char** argv) {
    int i;
    if (argc > 1) {
        for (i = 1; i < argc; ++i) {
            puts(argv[i]);
        }
    } else {
        puts("Hello world");
    }
    return 1;
}
```

000000000040052d <main>:

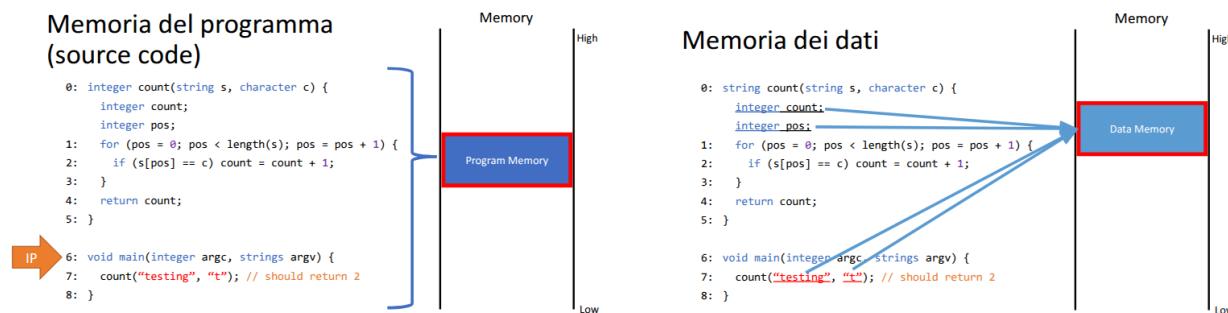
x86-64 machine code in hexadecimal	x86-64 assembly
40053c: 55	push rbp
40053d: 48 89 e5	mov rbp, rbp
40053e: 48 89 74 20	mov rbp, rbp+0x20
40053f: 48 7d ac	mov QWORD PTR [rbp-0x14], edi
400540: 48 89 75 00	mov QWORD PTR [rbp-0x20], rsi
400541: eb 23	cmp rbp, rbp+0x20
400542: 31 45 fc	mov eax, rbp
400543: 48 98	mov QWORD PTR [rbp-0x4], eax
400544: 48 99	cdqe
400545: 48 9d 14 c5 00 00 00	lea rdx, [rax+8+0x0]
400546: 00	mov rax, QWORD PTR [rbp-0x20]
400547: 48 8b 45 e0	add rax, rdx
400548: 40 00 00 00	mov rax, QWORD PTR [rax]
400549: 48 89 c7	call rax
40054a: 48 a6 fe ff ff	add DWORD PTR [rbp-0x4], 0x1
40054b: 48 83 45 fc 01	mov eax, DWORD PTR [rbp-0x4]
40054c: 48 83 e4 fc	cmp eax, DWORD PTR [rbp-0x14]
40054d: 48 99	jl 40054b <main+0x10>
40054e: 7c 45 ec	jmp 40054b <main+0x10>
40054f: c9	leave
400550: c3	ret

Uso della memoria

I programmi in esecuzione sono presenti nella memoria RAM, che è suddivisa in diverse parti:

- memoria del programma (program memory) → il codice per il programma
- memoria dei dati (data memory) → variabili, costanti e poche altre cose necessarie per il programma
- memoria del sistema operativo (OS memory) → sempre disponibile per le system call (aprire un file, eseguire un altro programma, stampare sullo schermo, ...)

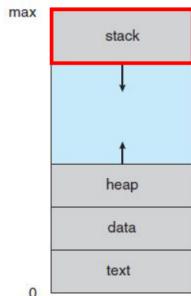
La CPU tiene traccia dell'attuale Instruction Pointer (IP).



Stack

La memoria dati è disposta utilizzando una struttura dati specifica: lo **stack**.

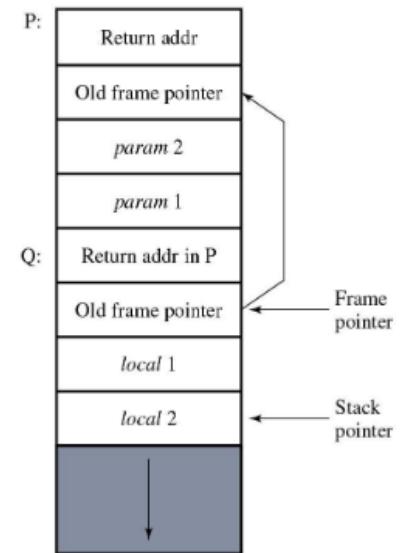
Quando viene chiamata una funzione (quando viene eseguita), si crea un frame sullo stack, che contiene variabili locali ed informazioni sul flusso di controllo. Il frame viene distrutto quando la funzione termina (exit o return). Lo stack cresce verso il basso.



Chiamata a funzione

Sia P la funzione chiamante, sia Q la funzione chiamata da P.

1. P fa il push sullo stack dei parametri per la funzione Q (*param 2, param 1*), tipicamente in ordine inverso di dichiarazione
2. P esegue l'istruzione di chiamata a funzione, che fa push dell'indirizzo di ritorno sullo stack (*Return addr in P*)
3. Q fa il push del *frame pointer* attuale sullo stack, che punta al frame della funzione chiamante (*Old frame pointer*)
4. Q imposta il valore del *frame pointer* al suo stack frame
5. Q alloca spazio per le sue variabili locali (*local 1, local 2*), spostando conseguentemente lo *stack pointer*
6. Q esegue il corpo della funzione
7. alla terminazione, Q reimposta il valore dello *stack pointer* al valore del *frame pointer* scartando le variabili locali
8. Q reimposta il valore del *frame pointer*, facendo pop del *Old frame pointer* dallo stack
9. Q esegue l'istruzione di return, facendo il pop dallo stack dell'indirizzo di ritorno di P e ritornando ad essa il controllo
10. P fa il pop dallo stack dei parametri per Q
11. P prosegue con l'esecuzione del suo corpo della funzione

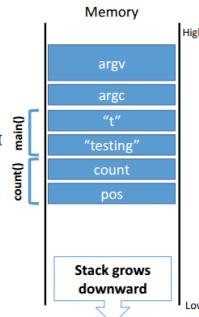


Un esempio di uno Stack Frame

```

0: string count(string s, character c) {
    integer count;
    integer pos;
1:   for (pos = 0; pos < length(s); pos = pos + 1) {
2:     if (s[pos] == c) count = count + 1;
3:   }
4:   return count;
5: }

6: void main(integer argc, strings argv) {
7:   count("testing", "t"); // should return 2
8: }
```



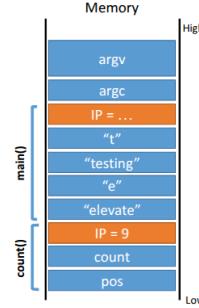
In questo esempio manca qualcosa di molto importante: l'IP deve tornare alla riga 8, ma la CPU non può saperlo poiché manca l'indirizzo di ritorno nello stack frame.

Esempio di Two Call

```

0: string count(string s, character c) {
    integer count;
    integer pos;
1-4: ...
5: }

6: void main(integer argc, strings argv) {
7:   count("testing", "t"); // should return 2
8:   count("elevate", "e"); // should return 3
9: }
```

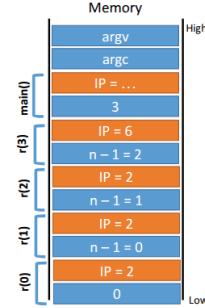


Esempio di una Ricorsione

```

0: integer r(integer n) {
1:   if (n > 0) r(n - 1);
2:   return n;
3: }

4: void main(integer argc, strings argv) {
5:   r(3); // should return 3
6: }
```



Stack Buffer Overflows

Uno **stack buffer overflow** si verifica quando il buffer si trova nello stack, solitamente come una variabile locale nello stack frame di una funzione. Questa forma di attacco viene anche chiamata *stack smashing*.

Gli attacchi di stack buffer overflow sono stati visti per la prima volta con il Morris Internet Worm nel 1988.

Un buffer overflow può verificarsi a causa di un errore di programmazione, quando un processo tenta di memorizzare dati oltre i limiti di un buffer (che si trova nello stack) di dimensione fissa e di conseguenza sovrascrive le posizioni di memoria adiacenti.

Un semplice programma vulnerabile

```
0: void print(string s) {
    // only holds 32 characters, max
    string buffer[32];
1: strcpy(buffer, s);
2: puts(buffer);
3: }

4: void main(integer argc, strings argv) {
5:     for (; argc > 0; argc = argc - 1) {
6:         print(argv[argc]);
7:     }
8: }
```

\$./print Hello World
World
Hello

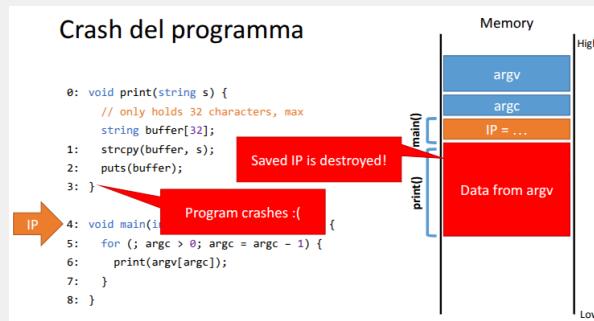
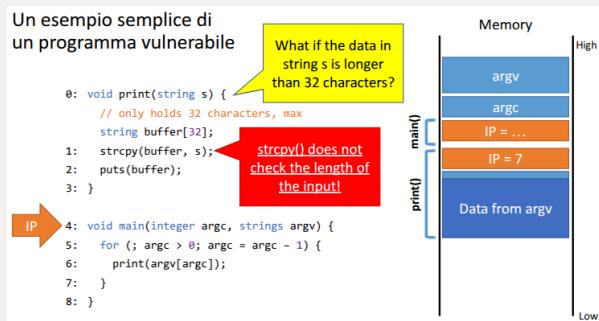
Copy the given string s into the new buffer
Print the buffer to the console/stdout

```
0: void print(string s) {
    // only holds 32 characters, max
    string buffer[32];
1: strcpy(buffer, s);
2: puts(buffer);
3: }

4: void main(integer argc, strings argv) {
5:     for (; argc > 0; argc = argc - 1) {
6:         print(argv[argc]);
7:     }
8: }
```

\$./print arg1 arg2 arg3
arg3
arg2
arg1

Dov'è il problema?



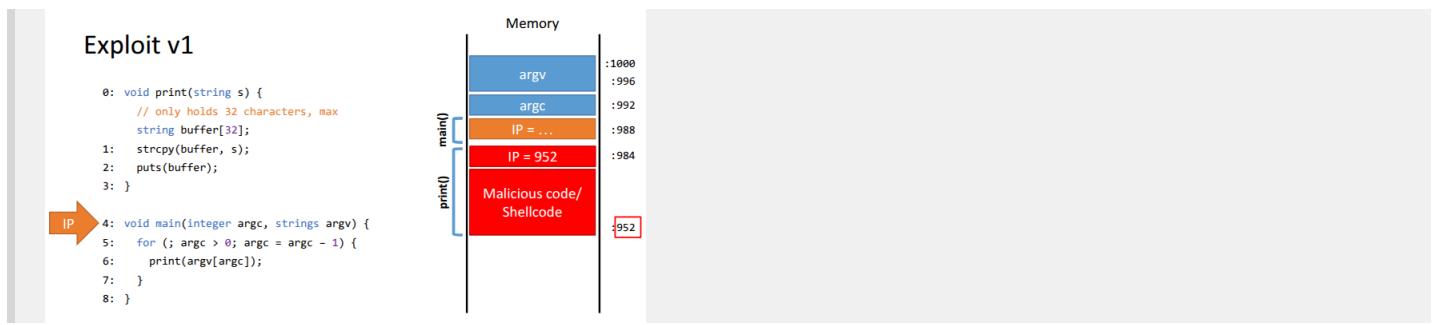
Quando la funzione tenta di trasferire il controllo all'indirizzo di ritorno (IP = 7), in genere salta a una posizione di memoria illegale, con conseguente *Segmentation Fault* e la terminazione anomala del programma. Fornire semplicemente un input casuale, che in genere porta al crash del programma, dimostra un attacco base dello stack overflow. E poiché il programma si è bloccato, non può più fornire la funzione o il servizio per cui era in esecuzione.

Nella sua forma più semplice, uno stack overflow può causare una qualche forma di attacco Denial-of-Service su un sistema.

Smashing the Stack

I bug di buffer overflow possono sovrascrivere i puntatori IP salvati, causando il crash del programma. Ma l'IP salvato è una semplice stringa di byte, modificabile per puntare ad una posizione arbitraria.

Idea chiave: sostituire l'IP salvato con un puntatore a del codice dannoso (shellcode) sullo stack.



Shellcode

Una componente essenziale di molti attacchi di buffer overflow è il trasferimento dell'esecuzione al codice fornito dall'aggressore e spesso salvato nel buffer in overflow. Questo codice dannoso è noto come **shellcode**. La sua funzione era quella di trasferire il controllo a un interprete della riga di comando dell'utente (shell) che dava accesso a qualsiasi programma disponibile sul sistema con i privilegi del programma attaccato.

Shellcode è semplicemente codice macchina: una serie di valori binari corrispondenti alle istruzioni macchina che implementano la funzionalità desiderata dall'aggressore. Lo shellcode è dunque specifico per una particolare architettura del processore e per un specifico sistema operativo.

Un esempio semplice di shellcode:

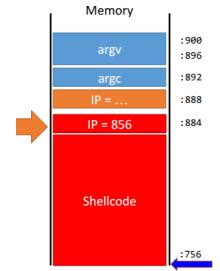
```
{
    // execute a shell with the privileges of the vulnerable program
    exec("/bin/sh");
}
```

Restrizioni generiche sul contenuto dello shellcode

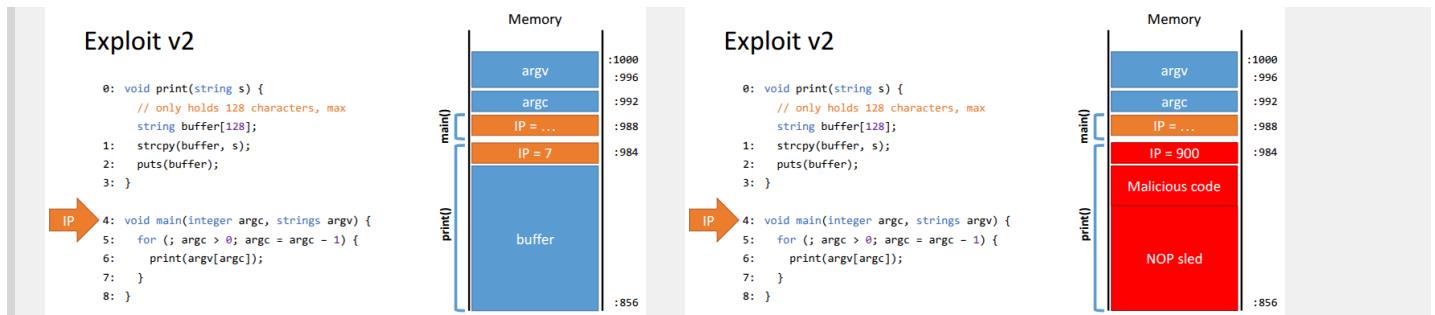
1. Lo shellcode deve essere indipendente dalla posizione. Possono essere utilizzati solo riferimenti di indirizzo relativi, offset rispetto all'indirizzo di istruzione corrente. L'attaccante non è in grado di specificare con precisione l'indirizzo di partenza delle istruzioni nello shellcode.
2. Lo shellcode non può contenere valori NULL. L'unico punto in cui può avere un NULL è alla fine, dopo tutto il codice, il vecchio puntatore di frame sovrascritto e i valori dell'indirizzo di ritorno.
3. Lo shellcode deve sopravvivere a qualsiasi modifica apportata dal programma. Deve essere creato per evitare o tollerare queste modifiche.

Colpire il bersaglio

L'indirizzo dello shellcode deve essere indovinato esattamente, serve saltare all'inizio preciso del codice. Gli indirizzi dello stack cambiano spesso, ogni volta che un programma viene eseguito. Per indovinare in modo affidabile l'indirizzo dello shellcode possiamo "rendere il bersaglio più grande", così sarà più facile centrarlo.



La maggior parte delle CPU supporta istruzioni no-op, ovvero istruzioni semplici che non fanno nulla. L'idea è creare una **slitta NOP** davanti allo shellcode: se il puntatore all'istruzione atterra in un punto qualsiasi della slitta, eseguirà NOP finché non raggiunge lo shellcode.



Alcune routine comuni della libreria C standard non sicure:

<code>gets(char *str)</code>	read line from standard input into str
<code>sprintf(char *str, char *format, ...)</code>	create str according to supplied format and variables
<code>strcat(char *dest, char *src)</code>	append contents of string src to string dest
<code>strcpy(char *dest, char *src)</code>	copy contents of string src to string dest
<code>vsprintf(char *str, char *fmt, va_list ap)</code>	create str according to supplied format and variables

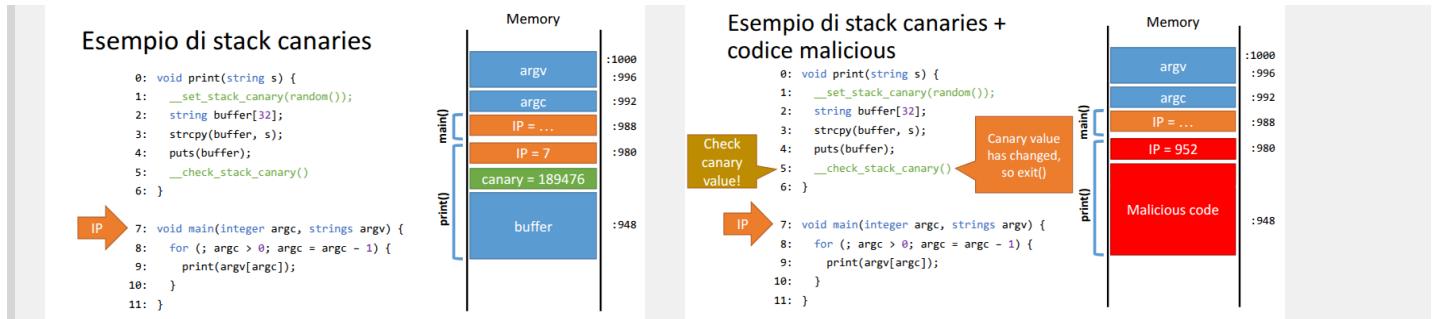
Esistono altre forme di attacchi di overflow:

- Heap Overflows
- Global Data Area Overflows
- Format string overflows
- Integer overflows
- ...

Difesa contro i Buffer Overflows

Stack Canaries - StackGuard

Il compilatore aggiunge valori speciali (*sentinel value*) sullo stack prima di ogni IP salvato. Il *canary* viene impostato su un valore casuale in ogni frame. All'uscita della funzione il *canary* viene controllato, se il numero previsto è stato alterato, il programma esce con un errore.



Stack non eseguibile

Le CPU moderne impostano la memoria dello stack come lettura/scrittura, ma non execute. Questo non evita i buffer overflow ma impedisce che lo shellcode venga inserito/eseguito nello stack.

Address-Space Layout Randomization (ASLR)

La randomizzazione del layout della memoria è una difesa a livello di sistema operativo. Ogni volta che un programma viene caricato in memoria, la posizione del codice (program memory) e dei dati (data memory) viene modificata.

Rende più difficile per l'attaccante indovinare la destinazione del buffer sullo stack, pur non impedendone lo sfruttamento. Questo meccanismo è supportato da tutti i moderni sistemi operativi, e funziona meglio quando la dimensione della memoria è molto grande.

Write Safe Code

Il messaggio importante è che se i programmi non sono codificati correttamente per proteggere le loro strutture dati fin dall'inizio, allora sono vulnerabili a potenziali attacchi. Sebbene le difese di cui abbiamo parlato possano bloccare molti di questi attacchi, alcuni semplicemente non possono essere bloccati se non tramite un codice sicuro.

Laboratorio

Lab 1 - Algoritmi

Implementazione

Per prima cosa serve importare la libreria `math`:

```
import math
```

Euclidean algorithm

```
# Returns (q,r) where r = gcd(x,y) and q is a list of rests
def euclidean(x, y):
    r = [x, y]
    q = []
    m = 1
    while(r[m] != 0):
        q.append(math.floor(r[m - 1] / r[m]))
        r.append(r[m - 1] - q[-1] * r[m])
        m += 1
    m -= 1
    return (q, r[m])
```

Extended Euclidean algorithm

```
# Returns (r,s,t) where r = gcd(x,y) and sx + ty = r
def extendedEuclidean(x, y):
    a, b = x, y
    t0 = 0
    t = 1
    s0 = 1
    s = 0
    q = math.floor(a / b)
    r = a - q * b
    while(r > 0):
        temp = t0 - q * t
        t0 = t
        t = temp
        temp = s0 - q * s
        s0 = s
        s = temp
        a = b
        b = r
        q = math.floor(a / b)
        r = a - q * b
    r = b
    return (r, s, t)
```

Multiplicative inverse algorithm

```
# Returns the inverse of y mod x
def multiplicativeInverse(x, y):
    a, b = x, y
    t0 = 0
    t = 1
    q = math.floor(a / b)
    r = a - q * b
    while(r > 0):
        temp = (t0 - q * t) % x
        t0 = t
        t = temp
        a = b
        b = r
        q = math.floor(a / b)
        r = a - q * b

    if b != 1:
        raise ValueError(f'No multiplicative inverse exists for {y} modulo {x}')

    return t
```

Square-and-multiply algorithm

```
# Returns x^c mod n
def squareAndMultiply(x, c, n):
    z = 1
    # binary representation of exponent c
    binc = bin(c)[2:]
    for bit in binc:
        z = (z*z) % n
        if bit == '1':
            z = (z * x) % n
    return z
```

Esercizi

Greatest Common Divisor (GCD) and the Euclidean Algorithm

```
print("GCD between 1759 and 550:", euclidean(1759, 550)[1])
r, s, t = extendedEuclidean(1759, 550)
print(f"s and t such that 1759s + 550t = {r}: s = {s}, t = {t}")
print("GCD between 57 and 93:", euclidean(57, 93)[1])
r, s, t = extendedEuclidean(57, 93)
print(f"s and t such that 57s + 93t = {r}: s = {s}, t = {t}")
```

Multiplicative inverse

```
try:
    print("Inverse of 17 mod 101:", multiplicativeInverse(101, 17))
except Exception as e:
    print(str(e))
try:
    print("Inverse of 357 mod 1234:", multiplicativeInverse(1234, 357))
except Exception as e:
    print(str(e))
try:
    print("Inverse of 3125 mod 9987:", multiplicativeInverse(9987, 3125))
except Exception as e:
    print(str(e))
```

RSA first example

```
print("p = 101, q = 113, n = 11413, fi(n) = 11200")
_, r = euclidean(11200, 3533)
print(f"GCD(11200, 3533) = {r}")
a = multiplicativeInverse(11200, 3533)
print(f"Bob secret exponent for decryption a is {a}")
c = squareAndMultiply(9726, 3533, 11413)
print(f"Ciphertext sent from Alice is {c}")
p = squareAndMultiply(c, a, 11413)
print(f"Plaintext decrypted by Bob is {p}")
```

Lab 2 - Password

Ethical hacking

Un **hacker** è persona esperta di informatica che utilizza le proprie conoscenze tecniche per raggiungere obiettivi o superare ostacoli in modi non standard, all'interno di un sistema computerizzato. Un hacker può essere di due tipi:

- White hat (hacker etici) → aiutano aziende e sviluppatori a controllare e migliorare la propria sicurezza. Seguono *Bug Bounty programs*, ovvero programmi che premiano chi scopre vulnerabilità non divulgata, oppure sono assunti per effettuare *Vulnerability Assessment and Penetration Testing (VAPT)*. Alcuni scelgono di vendere exploit a società come Zerodium.
- Black hat → agiscono per scopi malevoli

Un **VAPT** (Vulnerability Assessment and Penetration Testing) prevede diversi step:

1. Definizione dello scope e pianificazione
2. Raccolta informazioni
3. Analisi vulnerabilità
4. Sfruttamento (exploitation)
5. Reporting

Per fare esperienza si possono attaccare VM volutamente vulnerabili oppure si può partecipare a sfide *Capture The Flag*. Mai provare con bersagli reali se non si ha l'autorizzazione.

Servizi nascosti

L'**Onion Routing** (routing a cipolla) consente l'instradamento a strati cifrati. Dopo che i dati sono stati protetti da diversi livelli di crittografia, il traffico web viene trasmesso attraverso una serie di nodi di rete, chiamati *nodi/router a cipolla*. Ogni nodo "rimuove" uno strato di crittografia finché i dati non raggiungono la loro destinazione finale, completamente decifrati.

Tor è un browser che trasmette dati cifrati anonimamente lungo tre livelli (nodi *entry, middle, exit*) di proxy internazionali, che compongono il circuito di Tor. Nei **nodi** i dati dell'utente vengono completamente decifrati, essendo inviati attraverso una serie di nodi che decifrano i dati un livello alla volta. Per garantire l'anonimato, ogni nodo intermedio conosce solo l'identità del nodo precedente e di quello successivo, senza sapere chi sia il mittente o la destinazione.

Tor può nascondere l'indirizzo IP e l'attività di browsing utilizzando la crittografia multi-livello, ma non esiste l'anonimato perfetto online. Inoltre, si potrebbe essere identificati accedendo ad un account online o fornendo dettagli a siti web.

I servizi nascosti di Tor funzionano all'interno della rete Tor e consentono di registrare un servizio interno *Tor-only* con il suo personale *.onion* hostname. Tor risolve gli indirizzi *.onion* e indirizza al servizio nascosto dietro tale nome. I servizi nascosti forniscono un anonimato reciproco: il server non conosce l'indirizzo IP del client ed il client non conosce quello del server.

Password

Gli utenti spesso impostano la stessa password per numerosi servizi. Se la password utilizzata per un servizio viene leakata, può essere usata per accedere ad altri account dello stesso utente. Esiste un interno business dietro ciò: gli attaccanti non usano i dati degli utenti personalmente, li vendono sul dark web.

Hash

Un **hash** è come un'impronta digitale, è una funzione crittografica che crea un codice univoco, di lunghezza fissata, a partire da "oggetti" di dimensione variabile (testi, file, ...). Una **funzione hash** ha diverse caratteristiche:

- veloce da calcolare
- dimensione dell'output ridotta
- sensibilità alle modifiche (piccola modifica all'input porta un grande cambiamento nell'output)
- resistenza alle collisioni (impossibile trovare due input con lo stesso hash)
- irreversibilità (proprietà di *one-way*)

È computazionalmente infattibile trovare un oggetto che viene mappato ad uno specifico hash pre-definito oppure due oggetti che mappino allo stesso hash.



Lo scopo delle funzioni hash è garantire:

- Integrità, ovvero la certezza che un file non sia stato modificato durante il transito o la memorizzazione
- Confidenzialità, anziché salvare la password in chiaro, si salva l'hash, così in caso di data leak l'attaccante non conoscerà le password originali

Le funzioni hash più comuni sono MD5, SHA-1 e la famiglia SHA-2. Nella maggior parte delle distribuzioni Linux sono presenti strumenti per utilizzarle (GNU coreutils).

```
# generate hash
{md5, sha1, sha256}sum (filename)
# check hash (two spaces between hash and filename!)
echo "(hash) (filename)" | md5sum -c
```

Attaccare gli hash

L'obiettivo è trovare il plaintext a partire dal valore hash. Dovrebbe essere irreversibile, di fatto possiamo attaccare solo gli algoritmi di hash non sicuri. Esistono tecniche differenti per fare ciò:

- Pre-computed
- Brute-force
- Dizionario

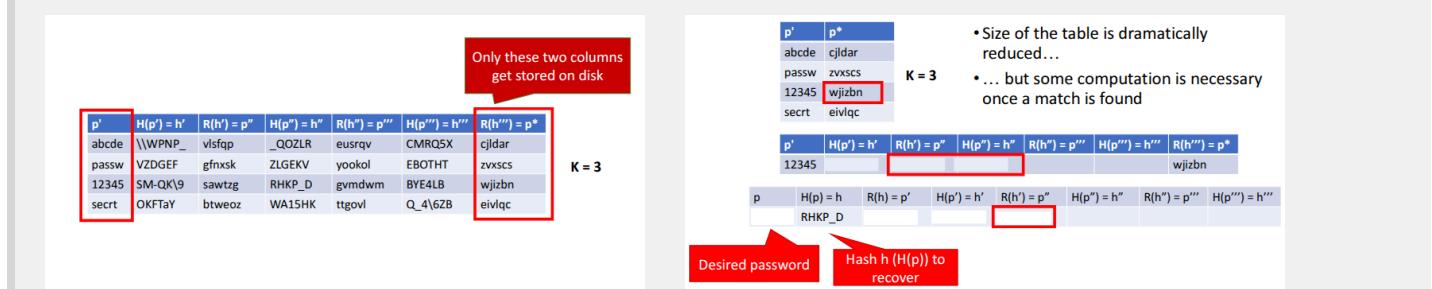
Pre-computed attack

Si pre-calcolano gli hash delle password più comuni e si memorizzano, poi si confrontano con quelli "rubati" per trovare la password corrispondente. Questo metodo risparmia tempo ma occupa molto spazio. Lo spazio può essere ottimizzato usando le rainbow tables.

Le **hash chain** consentono l'inversione efficiente (tempo e spazio) delle funzioni hash. L'idea è pre-calcolare catene di k password, salvando solo l'inizio e la fine della catena.

- k grande = meno catene da salvare, più CPU per ricostruirle
- k piccolo = più catene da salvare, meno CPU per ricostruirle

password originale → hash → altra password → altro hash → altra password → ...



Il grosso problema delle hash chain sono le collisioni, che causano sovrapposizioni tra le catene. Le collisioni portano inoltre a sprecare spazio nel file e ad avere falsi positivi, ovvero una catena potrebbe non contenere la password anche se la fine combacia. La scelta della riduzione è critica in questo caso: l'obiettivo è coprire le password più comuni, non tutte le password in generale.

Le **rainbow tables** migliorano le hash chain riducendo la probabilità di collisione. L'idea alla base è usare una famiglia di riduzioni invece di una riduzione singola, mentre la funzione di hash resta la stessa. In questo modo, una collisione tra due catene può avvenire solo nella stessa posizione (es. R_i in entrambe).

Le funzioni hash sono deterministiche, quindi per difendersi dagli attacchi pre-computed si usa il **salting**. Si aggiungono dei dati casuali (il *salt*) alla password e si salvano in chiaro insieme ad essa. Il pre-calcolo degli hash diventa così inutile. Se l'attaccante possiede l'hash della password, tipicamente ha anche il salt con essa, poiché è memorizzato in chiaro nello stesso database. Perciò altri attacchi sono ancora possibili.

ID	PASSWORD (HASH)	SALT
1	6e6bc4e49dd477ebc98ef4046c067b5f	7b6b1c1077c3fbe74b19
2	7a36be31fbe72d24c2d4bdb44c8055a6	41942cad1e6c17e7e2e3
3	0225205578734fc6ea670eae72e92160	32f38b5e593075f974d7
4	a00a34a06520ccf4d7e0e3d6442cb85f	e6dde301236a3891ca88
5	3ba94ed6ae8ac1891ef96c136853a5cc	2ff137a14978890fe1e9
6	e20d4b60cd5a8ebe1ca51b52eb0a1377	f9ee4a12e3ea69aa7be0

Brute-force attack

Semplicemente provare ogni possibile combinazione fino a quando non si trova quella corretta. Con password lunghe potrebbe essere un processo infinito.

Attacchi a dizionario

Provare tutte le parole di una lista predefinita. Le liste intelligenti sono l'opzione migliore: contengono le password più comuni e sono disponibili online. In uno scenario reale, questo attacco potrebbe richiedere settimane o mesi. Il crack degli hash può essere ottimizzato eseguendo numerose istanze in parallelo.

Cracking di password

Per prima cosa dobbiamo capire quale algoritmo di hash è stato usato. Il comando `hashid` può aiutare, ma non sempre.

Successivamente dobbiamo trovare la combinazione giusta. Useremo il comando `hashcat`.

```
hashcat (-m mode) (-a attack) (hash) [OPTIONS]
```

- `mode` codice della funzione hash (0 = MD5, 100 = SHA1)
- `attack` codice del metodo (0 = dictionary, 3 = brute-force)
- `hash` una stringa o un file contenente uno o più hash
- `OPTIONS` può introdurre la lista di parole

Usare `man` per vedere tutti i codici e configurazioni.

Gli hash cracciati vengono salvati nel *potfile* in `~/.local/.../hashcat/hashcat.potfile`. Usare `--show` per comparare la lista di hash in input con il *potfile*, mostrando quelli corrispondenti.

Creare l'hash MD5 di "hola"

```
echo -n "hola" | md5sum
```

Risultato: 4d186321c1a7f0f354b297e8914ab240

In questo caso la risposta di hashid è ambigua: MD2/4/5?

Crack con hashcat

MD5 vuol dire m = 0, dictionary è a = 0, usiamo la lista *rockyou.txt*

```
hashcat -a 0 -m 0 "4d186321c1a7f0f354b297e8914ab240"  
/usr/share/wordlists/rockyou.txt
```

Creare l'hash MD5 di "hola", con salt = 1234

Risultato: ccee5504c9d889922b101124e9e43b71

Crack con hashcat

La sintassi è hash:salt

```
hashcat -a 0 -m 10 "ccee5504c9d889922b101124e9e43b71:1234"  
/usr/share/wordlists/rockyou.txt
```

Si può essere anche più furbi con attacchi brute-force. Ad esempio, molte persone usano come password nome e anno di nascita. Possiamo usare le **maschere** (parametro `-m`) per guidare hashcat a provare solo parole che seguono un determinato pattern.

Le **regole** sono anche più potenti, permettendo di cambiare la lista di parole per seguire un certo pattern. Si possono creare regole personali, ma hashcat ne ha alcune built-in in

`/usr/share/hashcat/rules/`. Usare le regole con il parametro `-r`.

Creare l'hash MD5 di "Hola123!"

Risultato: 401518eee35b49f00bc0a3ab74c4915e

Questa parola non è inclusa in *rockyou.txt*, perciò hashcat non sarebbe in grado di craccarla senza regole.

Crack con hashcat usando le regole

Usiamo una regola predefinita di hashcat.

```
hashcat -a 0 -m 0 -r /usr/share/hashcat/rules/T0xLcv2.rule "401518eee35b49f00bc0a3ab74c4915e"  
/usr/share/wordlists/rockyou.txt
```

Altre regole sono `InsidePro-PasswordsPro.rule`

Esercizi

1. 6e6bc4e49dd477ebc98ef4046c067b5f

Plaintext: ciao

2. 427ade9c15ec643751860eba9899355b

Plaintext: gatto

3. 6c00f2d6e1610bfc9b415daf80d45855f2c56443c2dc2f71e7ef27168d1f2857d6168f4d374ed8ec
a349f2debd18d4ccac339218ca70446adf999060395742b4

Salt: hjt88q

Plaintext: markinho

4. 0e8ae09ae169926a26b031c18c01bafa

Plaintext: ILOVEME8320

5. c73fceab80035a75ba3fd415ecb2735

Plaintext: soccer23!

6. dc612dc12fb4540a88b88875c2bee3b4

Plaintext: NON RICAVATO

Lab 3 - Crittografia applicata

Introduzione

Advanced Encryption Standard (AES) è un block cipher a chiave simmetrica, con blocchi da 128

bit e chiavi da 128, 192 o 256 bit. È leggero, in quanto consuma poca RAM, ed è veloce.

Le modalità di operazione per la confidenzialità esistenti:

- ECB (Electronic Code Block)
- CBC (Cipher Block Chaining)
- CFB (Cipher Feedback)
- OFB (Output Feedback)
- CTR (Counter)

Useremo **OpenSSL** per lavorare con gli algoritmi di crittografia. OpenSSL è una libreria open source che implementa le primitive di crittografia di base: algoritmi di hash, protocolli SSL e TLS, generatori di numeri primi, PRNG, ecc. Si utilizza tramite una CLI.

Transport Layer Security (TLS) e il suo predecessore **Secure Sockets Layer (SSL)** sono dei protocolli crittografici a livello presentazione usati nel campo delle telecomunicazioni e dell'informatica. Permettono una comunicazione sicura dalla sorgente al destinatario (end-to-end) su reti TCP/IP fornendo autenticazione, integrità dei dati e confidenzialità.

Certificati web

Controllare il certificato di un sito web:

- Vedere le informazioni del certificato

```
openssl s_client -connect www.unibo.it:443 2>/dev/null | openssl x509 -noout -text
```

- Vedere i dettagli della connessione TLS

```
openssl s_client -connect www.unibo.it:443 2>/dev/null
```

- Controllare la versione di TLS supportata

```
openssl s_client -connect www.unibo.it:443 -servername www.unibo.it -tls1_2
```

Oppure con `-tls1` o `-tls1_1`.

SSLLABS è un sito per verificare i certificati, i parametri a cui fare più attenzione sono:

- versioni di TLS1.X supportate
- cifrari supportati
- HSTS abilitato (potrebbero essere visibili dei parametri, tipo max-age) oppure no, HSTS preloading abilitato oppure no
- dispositivi particolari che non possono più visualizzare il sito a causa delle impostazioni particolarmente restrittive
- alternative names interessanti (ftp, smtp, ...)
- punteggio

<https://www.ssllabs.com/ssltest/>

Let's Encrypt è una Certificate Authority automatizzata che fornisce certificati TLS gratuiti, semplificando l'abilitazione della crittografia HTTPS da parte dei siti web e creando un Internet più sicuro per tutti.

<https://letsencrypt.org/it/>

Crittografia

PGP e GPG

Per utilizzare i meccanismi di crittografia simmetrica e asimmetrica possiamo usare un software chiamato **Pretty Good Privacy (PGP)**. Esso permette di proteggere dati (salvati su disco o inviati in rete), firmare e usare certificati.

Esiste un'implementazione GNU (quindi rilasciata come Free software sotto licenza GPL) chiamata **GNU Privacy Guard (GPG)**.

Per verificare l'autenticità di una email utilizzando PGP è sufficiente cercare all'inizio del testo "---- BEGIN PGP SIGNED MESSAGE----", se c'è, l'email è probabilmente a posto.

Non esiste una CA (Central Authority) per la fiducia, quindi tutti sono responsabili della firma delle chiavi pubbliche altrui.

AES

Cifrare un semplice file di testo con AES-256 usando ECB:

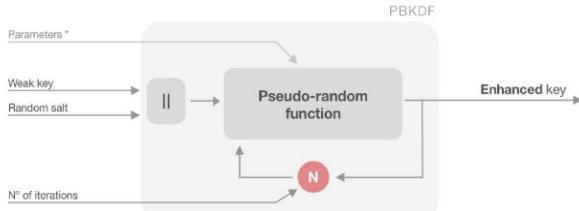
```
openssl aes-256-ecb -e -in (plaintext) -out (ciphertext)
```

Decifrare:

```
openssl aes-256-ecb -d -in (ciphertext) -out (plaintext)
```

Lunghezza della chiave

La chiave in AES deve essere 128, 192 o 256 bit in lunghezza. Quindi la nostra chiave human-readable (debole) viene utilizzata per generare una chiave robusta con maggiore entropia. Queste funzioni sono dette *Password Based Key Derivation Functions (PBKDFs)*.



In OpenSSL, usare `-p` per stampare la chiave "robusta", il salt e l'IV (se usato). Usare `-nosalt` per disabilitare l'uso del salting, incrementando la casualità della chiave.

Di default, OpenSSL applica un PBKDF banale:

- `key = sha256(passphrase)` se il salting non è abilitato
- `key = sha256(passphrase || salt)` se il salting è attivo

Un'opzione migliore è usare più iterazioni (`-iter (number of iterations)`) oppure PBKDF2 (`-pbkdf2`).

Dimensione del file

La dimensione del plaintext e del ciphertext potrebbe differire (cipher > plain). Questo avviene per due ragioni:

- Il salt è memorizzato nell'header del ciphertext (a meno che non si usi `-nosalt`)
- Il plaintext subisce del padding prima di essere cifrato (con ECB o CBC)

Visualizzare un file cifrato utilizzando un normale editor di testo o stamparlo sulla console non può funzionare, in quanto il plaintext contiene caratteri ASCII stampabili, mentre il ciphertext può contenere caratteri non stampabili.

Quando abbiamo a che fare con questo genere di dati, abbiamo bisogno di vedere i file usando il comando `hexdump`, ad esempio. In questo modo, possiamo visualizzare dati binari codificati in formato esadecimale. Usare `xxd` per visualizzare l'hexdump di un file preso in input.

Debolezza della modalità ECB

La modalità **ECB** pecca di diffusione: blocchi di plaintext identici generano blocchi di ciphertext identici. Possiamo verificare questo comportamento cifrando una semplice immagine bitmap (formato `.ppm`), il logo di Linux (Tux) ad esempio. PPM (Portable PixMap) è il formato di immagine più semplice.

Visualizzare:

```
xxd -g 3 -c 15 tux.ppm
```

Separare header e body in due file differenti:

```
head -n 3 Tux.ppm > Tux.header  
tail -n +4 Tux.ppm > Tux.body
```

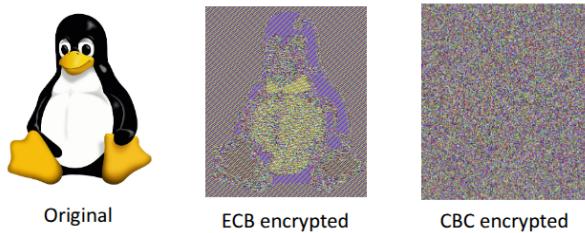
Cifrare il body:

```
openssl aes-256-ecb -e -in Tux.body -out Tux.body.ecb
```

Riassemblare header e body insieme:

```
cat Tux.header Tux.body.ecb > Tux.ecb.ppm
```

Ora facciamo la stessa cosa utilizzando la modalità **CBC**, fornendo un IV tramite l'opzione `-iv`. La modalità CBC nasconde i pattern del plaintext grazie allo XOR del primo blocco di plaintext con l'IV, prima di cifrare il tutto. Inoltre, sfrutta la concatenazione di blocchi, ponendo in XOR il blocco attuale di plaintext con il precedente blocco di ciphertext.



Esercizi

Dopo numerosi tentativi fallimentari, ho rinunciato a completare il laboratorio.

- HASH PASSWORD 1: 9e02fdbbc6df842939b75a98e92e98317d29046c
PASSWORD 1: spaghettiolognase1987
FLAG #1: !bella_vez09567!
- HASH PASSWORD 2a: 977cd0822fe6f0a1ca64787f933820f3 (per questa non servono nemmeno le rules)
HASH PASSWORD 2b:
3b2fd659c8f48db0688148fabe3cd9d37e928e9b04d7f3a1ab691263abf1c72949a632cf1d77695
e7c6cb887bfa3dbaff6e52900e75d2f0296006cf79ef5ca7e (con salt kjsahdjhasd)
PASSWORD 2a: ajeje
PASSWORD 2b: NON RICAVATA
- HASH PASSWORD 3:
e547671ff6e7e1d7b5fc2141f6b63648b45a2df2d1a7270ed9f0773a79109007
PASSWORD 3: NON RICAVATA

Lab 4 - Attacchi in rete

Wireshark

Wireshark è un analizzatore di pacchetti open source e gratuito. Ha un'interfaccia grafica per mostrare i pacchetti sniffati e numerosi filtri per trovare i pacchetti desiderati. Wireshark può effettuare analisi di rete in real-time oppure offline su traffico catturato precedentemente tramite file PCAP.

L'interfaccia grafica mostra una lista di pacchetti con un riassunto di ognuno. Cliccando su un pacchetto, verranno mostrati tutti i dettagli dei livelli TCP/IP con i rispettivi protocolli. Si possono selezionare proprietà di pacchetti specifici per impostarle come regole di filtraggio (dimensione, metodo HTTP...).

No.	Time	Source	Destination	Protocol	Length	Info
1	9.000000	192.168.2.1	239.255.255.258	SSDP	216	M-SEARCH + HTTP/1.1
2	1.000000	192.168.2.1	239.255.255.258	SSDP	216	M-SEARCH + HTTP/1.1
3	2.001916	192.168.2.1	239.255.255.258	SSDP	216	M-SEARCH + HTTP/1.1
4	4.000000	192.168.2.1	239.255.255.258	SSDP	216	M-SEARCH + HTTP/1.1
5	14.000881	192.168.2.1	192.168.2.255	BROWSER	243	Local Master Announcement LAPTOP-HCSFMST7, Workstation, Ser
6	21.000272	192.168.1.100	ff02::1:13	LLNR	84	Standard query 0x7246 A who
7	21.000272	192.168.1.100	ff02::1:132	LLNR	84	Standard query 0x7246 A who
8	21.500682	fe80::1665:1469:9::4e	ff02::1:13	LLNR	84	Standard query 0x7246 A who
9	21.500697	192.168.2.1	224.0.2.25	LLNR	64	Standard query 0x7246 A who
10	21.500700	00:0c:29:91:02:a9	ff1f:ffff:ffff:ffff:ffff:ffff	ARP	64	Who has 10.0.0.5 Tell 10.0.0.6
11	37.000463	10.0.0.5	10.0.0.5	TCP	66	SYN - 8999 Len=24
12	37.000319	10.0.0.5	10.0.0.5	TCP	74	48528 - 8999 [SYN, ECE, CCR] Seq=0 Win=25200 Len=8 MSS=1460
13	37.000341	10.0.0.5	10.0.0.5	TCP	54	8999 - 48528 [RST, ACK] Seq=1 Ack=1 Win=8 Len=8
14	37.000341	10.0.0.5	10.0.0.5	TCP	54	8999 - 48528 [RST, ACK] Seq=1 Ack=1 Win=8 Len=8
15	39.002355	10.0.0.5	ff02::1:13	ARP	66	Who has 10.0.0.5 Tell 10.0.0.6
16	39.002359	10.0.0.5	ff02::1:13	ARP	66	Who has 10.0.0.5 Tell 10.0.0.6
17	39.002359	10.0.0.5	ff02::1:13	ARP	66	Who has 10.0.0.5 Tell 10.0.0.6
18	39.005954	00:0c:29:91:02:a9	ff:ffff:ffff:ffff:ffff:ffff	ARP	66	Who has 10.0.0.5 Tell 10.0.0.6
19	42.010389	10.0.0.9	10.0.0.5	TCP	66	SYN - 8999 Len=24
20	42.010389	10.0.0.9	10.0.0.5	TCP	54	8999 - 48524 [SYN, ACK] Seq=1 Ack=1 Win=25200 Len=8 MSS=1460
21	42.022658	10.0.0.4	10.0.0.5	TCP	54	8999 - 48524 [SYN, ACK] Seq=1 Ack=1 Win=8 Len=8
22	42.039326	00:0c:29:91:02:a9	ff:ffff:ffff:ffff:ffff:ffff	ARP	68	Who has 10.0.0.5 Tell 10.0.0.6
23	42.028199	00:0c:29:91:02:c7	00:0c:29:91:02:a9	ARP	42	Who has 10.0.0.6 Tell 10.0.0.6

Filtri utili

```
tls.handshake.extension.type == "server_name" : controlla se il dominio di un sito web
(SNI, Server Name Indication) esiste

ssl.handshake.extension.data contains "example.com"

http

http && http.request.method == GET

http && http.request.method == GET && http.request.full_uri contains unibo

Display filters e Capture filters non sono la stessa cosa!
```

Monitorare il traffico di rete (sniffing) può essere essenziale per la scoperta di attacchi e/o anomalie, ma anche per effettuare attacchi.

Un analizzatore di pacchetti può essere impiegato:

- sulla rete, per monitorare il traffico in transito tra i vari host
- su un host, per "ascoltare" il traffico in entrata ed uscita (si possono vedere i dati dei livelli applicazione)

Il computer potrebbe avere più di un'interfaccia di rete. In generale si può scegliere tra interfacce cablate o wireless. In caso di interfacce cablate, potremo vedere soltanto il traffico diretto/proveniente esattamente verso/da l'host in questione. In caso di wireless, possiamo sniffare qualunque cosa.

Wireless

Nelle reti wireless pubbliche non c'è nessuna cifratura, quindi chiunque può sniffare i pacchetti, o peggio, chiunque può effettuare un attacco Man-in-the-Middle.

Con il Wi-Fi possiamo decidere tra algoritmi differenti per proteggerci:

- WEP → algoritmo RC4 (stream cipher), CRC32 checksum, INSICURO
- WPA → TKIP
- WPA2 → niente più RC4
- WPA3

Tutti questi algoritmi richiedono l'uso di password, che come sappiamo devono essere robuste per non essere craccate (attacchi brute-force e dizionario).

Nelle reti wireless protette, la cifratura dovrebbe garantire la confidenzialità dei pacchetti.

Attaccare WEP

WEP è stato progettato parecchi anni fa, adesso è obsoleto. Un utente a conoscenza della chiave condivisa può decifrare qualunque pacchetto altrui.

WEP utilizzava un IV, mandato in chiaro nei pacchetti, con pochi bit (24), perciò catturando alcuni pacchetti era possibile derivare matematicamente la chiave condivisa.

Attaccare WPA

Sniffando l'handshake a 4 vie, durante la fase di distribuzione delle chiavi, si poteva effettuare in seguito un attacco offline. L'attaccante era in grado di raccogliere tutte le informazioni necessarie per eseguire un attacco brute-force o a dizionario. In questo caso, la probabilità di successo era principalmente legata alle password.

Monitor Mode

Con i cavi fisici, non considerando le vecchie topologie con hub, non si potevano vedere i pacchetti a meno che non si fosse fisicamente connessi al cavo. Con il wireless invece, tutto è "nell'aria", quindi è sufficiente ascoltare per ricevere tutti i pacchetti.

Questo metodo è chiamato **Monitor Mode**: permette di catturare e vedere tutti i pacchetti provenienti da uno specifico canale wireless. Il primo passo è trovare il canale dell'AP bersaglio, per poi monitorarne il traffico.

Abilitare la monitor mode sul pc dipende dalla scheda di rete wireless. Si può seguire la *aircrack-ng guide*. Con *airmon-ng*, dopo aver catturato l'handshake WPA, si può stoppare l'analisi e proseguire con il file PCAP ottenuto.

Attaccare il Wi-Fi

Con la monitor mode possiamo catturare i pacchetti scambiati durante l'handshake a quattro vie. Fatto ciò, salviamo il file *.pcap* contenente i pacchetti e possiamo provare un attacco a dizionario o brute-force offline. Possiamo anche usare Wireshark per analizzare i pacchetti.

Per attaccare WEP e WPA/WPA2 possiamo usare la suite *aircrack-ng*.

```
aircrack-ng (-w wordlist) (-b BSSID) (pcap file) [OPTIONS]
```

- **wordlist** : lista di parole da usare per il crack della password
- **BSSID** : indirizzo MAC dell'AP bersagliato
- **PCAP file** : file contenente l'handshake sniffato
- **OPTIONS** : per ottimizzare gli attacchi a WEP o WPA2

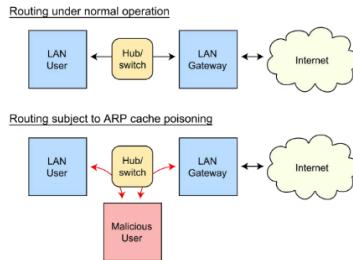
ARP

Address Resolution Protocol (ARP) è il protocollo che risolve gli indirizzi IP in indirizzi MAC. Con l'attacco detto "ARP poisoning" l'attaccante è in grado di falsificare le risposte ARP, dirottando il traffico legittimo a sè.

```
Source          Destination      Protocol Length Info
PCSSystemtec_bc:84: Broadcast   ARP      42 Who has 10.0.2.3? Tell 10.0.2.15
52:54:00:12:35:03  PCSSystemtec_bc:84:... ARP      60 10.0.2.3 is at 52:54:00:12:35:03

(kali㉿kali)-[~]
└─# arp -a
(kali㉿kali)-[~]
└─# ping 10.0.2.3
PING 10.0.2.3 (10.0.2.3) 56(84) bytes of data.
64 bytes from 10.0.2.3: icmp_seq=1 ttl=64 time=0.215 ms
64 bytes from 10.0.2.3: icmp_seq=2 ttl=64 time=0.145 ms
64 bytes from 10.0.2.3: icmp_seq=3 ttl=64 time=0.190 ms
"C"
10.0.2.3 ping statistics --
3 packets transmitted, 3 received, 0% packet loss, time 2182ms
rtt min/avg/max/mdev = 0.145/0.183/0.215/0.026 ms

(kali㉿kali)-[~]
└─# arp -a
? (10.0.2.3) at 52:54:00:12:35:03 [ether] on eth0
```



Esercizi

Crack WiFi passwd

1. Usando `aircrack-ng WIFI-analisi1.cap` troviamo l'handshake con ESSID *teddy*
2. Passiamo a wireshark per analizzare i pacchetti ed isolare quelli relativi all'handshake in un file `handshake.cap`
3. Utilizzando `aircrack-ng -w /usr/share/wordlists/rockyou.txt handshake.cap` troviamo la chiave, ovvero **44445555**

Detect the intruder

1. Filtrando le conversazioni HTTP contenenti ".html" individuiamo l'IP sorgente dell'intruder, ovvero **10.0.2.15**, e l'IP del server web, ovvero **10.0.2.3**
2. Controllando la risposta del server troviamo il seguente testo: "Il tuo codice è CCIT{asdkjhasd92837ekasjhdi8q98aksduh912}"