Fondamenti di Cybersecurity
Appunti

Giovanni "Qua' Qua' dancer" Palma
e
Alex "Morbidelli WhatsApp" Basta

Contents

Ci sara' una domanda sul lab

Example 0.0.1

Quali opzioni ho per crackare una password?

Chapter 1

Key Exchange

1.1 Introduction to Cryptography

Definition 1.1.1: Cryptography

Art and science of using mathematics to obscure the meaning of data by applying transformations to the data thta are impractical or impossible to reverse without the knowledge of some key

Definition 1.1.2: Cryptoanalysis

Art/science of breaking encryption without knowing the key

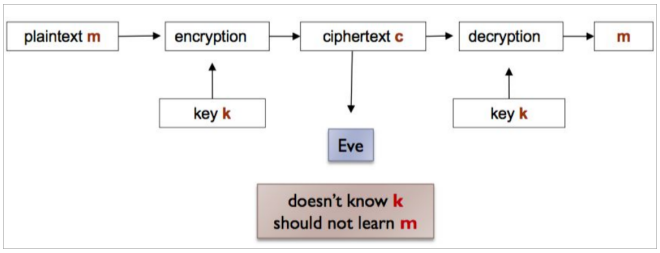
Used for:

- **Communication**: web traffic, wireless, vpn
- **Files on disk**
- **User authentication**

For secure communication, we also want to ensure no eavesdropping or tampering. Possible approaches are:

- **Steganography**: we 'hide' the existance of the message
- **Cryptography**: we instead hide the meaning of the message

1.1.1 Encryption Terminology



1.1.2 Goals and Protocols

The basic goals are:

- **Privacy**
- **Authenticity**

- **Integrity**
- **Non-repudiation**: no disclaming of authorship (guarantees Authenticity and Integrity)

The *protocols* need to guarantee these goals by understanding:

- The parties and the context
- The goals
- The **trusted computing base**
- The capabilities of the ... (**Threat Model**)

1.1.3 Kerchoff's Principle and the Threat Model

Important rule regarding the safety of cyber systems

> **Theorem 1.1.1**
> The security of a protocol shouldn't assume that the underlying methods/algorithms of the encryption are secret, as only the secrecy of the keys can be guaranteed.
> **Security by obscurity does not work.**

So the encryption functions need to remain secure even with the attacker knowing how the function works.
The attacker threat model consists of:

- Knowledge about the cipher (Kerchoff)
- Interaction with the messages and the protocol
- Interaction with the encryption algorithm
  - **Ciphertext-only**
  - **Chosen-plaintext attack (CPA)**
  - **Chosen-ciphertext attack (CCA)**
  - CPA and CCA may be *adaptive* (previous requests may change choices)
- Available resources (storage/computation)

1.1.4 Symmetric Encryption

Lowkey l'abbiamo fatta gia' due volte, aggiungo solo roba che dice in piu' (sta letteralmente leggendo le slide come il guerriero)

- **Single-use key**: the key is regenerated for each transmitted file
- **Multi-use key**: the key is used with a "nonce" in order to be used for multiple transmissions

1.1.5 Asymmetric Encryption

Private and public key. La pubblica e' salvata in una **public repo** gestita da un'autorita' trusted

1.1.6 Secure Communication

Una volta SSL ora TLS, poi RCS per end-to-end security (sono la stessa cosa??)

1.1.7 Boh

Criptography is a rigorus science that follows three main steps:

- Specify Threat Model
- Propose a construction
- ...

un po di storia, va veloce (manca la post-quantum criptology)
vabbe' sto scrivendo tutot in italiano facciamolo in italiano
la brodez sta davvero ripetendo il cifrario di Cesare.... e' ez da battere provando tutte le chiavi (non ce ne stanno molti) -¿ bruteforce/exhaustive-search criptanalysis

Monoalphabetic cipher

Come chiave prende una permutazione dell'alfabeto sono 2^{88} non possiamo fare brute force possiamo fare cript-analysis con metodi statistici, ovvero usando la frequenza di ogni lettera

Affine cipher

Caso specifico del sub-cipher1

1.1.8 Stream cipher

in Un simmetric cipher definito su (K, P, C), con K chiave, plaintext P e ciphertext C, abbiamo che la funzione di encryption è spesso randomizzata ed è definita come E : K x P -¿ C e invece la funzione di decryption è deterministica e tale per cui D: K x C -¿ P.

1.1.9 One-Time Pad

Definition 1.1.3: One-Time Pad

Si definisce *One-Time Pad* il seguente cifrario:

- Spazio delle chiavi: $0,1^n$ (stringhe di bit di lunghezza n)
- Funzione di encryption: $E_k(m) = m \oplus k$
- Funzione di decryption: $D_k(c) = c \oplus k$

Dove k è la chiave e m è il plaintext e `len(k) = len(m)`

Praticamente si fa lo XOR tra i bit del plaintext e quelli della chiave. In questo caso la lunghezza della key deve essere uguale a quella del plaintext.

Proposition 1.1.1 Sulla corrtettezza dell'Otp

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

pro: è mega veloce sia in encyption che in decryption brodez contro: deve usare chiavi lunghissime perché la key deve essere lunga come il messaggio. ma dato che questa key deve essere mandata da alice a bob in modo sicuro, allora potrebbe usare sto meccanismo sicuro per mandare direttamente il messagio, quindi boh non serve a un cazzo.

Sicurezza del cifrario OTP

PEr supporre che un messaggio sia sicuro, si potrebbero usare due approcci naif:

- **L'attaccante non riesce a trovare la chiave**: Non basta. Magari non trova la chiave ma riesce a decifrare il messaggio lo stesso
- **L'attaccante non riesce a recuperare l'intero plaintext**: Non basta neanche. Se recupera anche solo un bit del messaggio, per molte applicazioni è già un disastro

SI RINGRAZIA PROFONDAMENTE IL BUON CLAUDE SHANNON PER LA DEFINIZIONE DI SICUREZZA, OVVERO CHE *ciphertext non deve rivelare assolutamente nessuna informazione sul plaintext*. Zero. Nulla. Neanche quante vocali ci sono, neanche la lunghezza approssimativa. Si noti la la seguente definizione:

Definition 1.1.4: Sicurezza perfetta secondo Shannon

Un cifrario (E, D) su (K, P, C) ha *perfect secrecy* se $\forall m_0, m_1 \in P : len(m_0) = len(m_1)$ e $\forall c \in C$ allora:

$$Pr[E(k, m_0) = c] = Pr[E(k, m_1) = c]$$

Dove k è la chiave

Note:

In altre parole la probabilità che il chipertext c è idendica qualunque sia il messaggio cifrato. PErtanto non c'è informazione in c su quale sia m_0 o m_1

Si noti il seguente teorema:

Theorem 1.1.2

Il One-Time Pad ha perfect secrecy

Dimostrazione: Sia m e c, vogliamo calcolare $Pr[E(k, m) = c]$ dove k è uniforme su K.
Tale probabilità è uguale al numero di chiavi $k \in K$ tali che $E(k, m) = c$ diviso al numero di chiavi totali $|K|$, pertanto basta dimostrare che questa probabilità è costante $\forall(m, c)$
Nell'OTP $E(k, m) = m \oplus k$ implica $k = m \oplus c$. C'è esattamente una chiave per ogni coppia (m, c) che produce quel ciphertext. Il numeratore quindi è sempre 1, indipendemente da m e c.
Pertanto si che

$$Pr[E(k, m) = c] = \frac{1}{|K|} \quad \forall m, c$$

La probabilità è la stessa per qualunque messaggio

Si ha tuttavia una conseguenza importante

Theorem 1.1.3 Teorema di Shannon

La sicurezza perfetta implica che $|K| \geqslant |P|$

Note:

Ovvero un cifrario ha sicurezza perfetta se e solo se la lunghezza della chiave è almeno quella del messaggio

Se volete sicurezza perfetta, pagate il prezzo di chiavi lunghissime. E questo è impraticabile su larga scala.
La conseguenza pratica è drastica: dobbiamo rinunciare alla sicurezza perfetta e accontentarci di qualcosa di "meno perfetto", la sicurezza computazionale, che è comunque più che sufficiente.

1.1.10 Pseudo - Random Generators e Stream Ciphers

Abbiamo due fatti:

- Il OTP è perfettamente sicuro ma richiede chiavi lunghe quanto il messaggio

- La sicurezza perfetta è impossibile con chiavi corte.

La soluzione è è quella di rinunciare alla sicurezza perfetta e sostituiamo la chiave veramente veramente casuale con una sequenza pseudo-casuale generata da una chiave corta. Questo è l'idea degli stream cipher. Formalmente

Definition 1.1.5: Stram Cipher

Si definisce *Stream Cipher* un sistema che mira a emulare la sicurezza del One-Time Pad in modo efficiente. A differenza dei cifrari a blocchi che operano su segmenti fissi di dati, lo stream cipher elabora il testo in chiaro bit per bit o byte per byte

Adesso si noti la seguente definizione

Definition 1.1.6: Pseudo-Random Generator (PRG)

Si definisce *PRG* la seguente funzione:

$$G : \{0,1\}^s \rightarrow \{0,1\}^n \text{ con } n >> s$$

In prarica riceve in input un seed (la chiave breve, s bit) e produce una sequenza molto più lunga (n bit) che sembra casuale. Lo stream cipher usa G così

- $E(k, m) = G(k) \oplus m$
- $D(k, c) = G(k) \oplus c$

La chiave k è breve (il seed), la sequenza $G(k)$ è lunga quanto il messaggio.

Tre Tipi di Generatori

Vale la pena distinguere tre concetti

Definition 1.1.7: TRNG (True Random Number Generator)

usa fenomeni fisici genuinamente casuali ex rumore termico, eventi radioattivi, jitter dell'orologio. Produce vera casualità, ma è lento

Definition 1.1.8: PRNG (Pseudo-Random Number Generator)

algoritmo deterministico che a partire da un seed produce una sequenza lunga e statisticamente "casuale"

Note:

Veloce ma deterministico, da un seed produce sempre la stessa sequenza

Definition 1.1.9: PRF (Pseudo-Random Function)

una funzione che, dato un input, produce un output pseudo-casuale

Note:

è una generalizzazione del PRNG

Requisiti di un buon PRNG

Un PRNG per uso crittografico deve soddisfare due requisiti fondamentali

- **Randomness (apparente)**: la sequenza prodotta deve superare tutti i test statistici noti — distribuzione uniforme dei bit, assenza di pattern, correlazioni basse tra bit vicini
- **Unpredictability (imprevedibilità)**: dato un prefisso della sequenza, non deve esistere nessun algoritmo efficiente (in tempo polinomiale) in grado di predire il bit successivo con probabilità significativamente maggiore di 1/2 (**next-bit test**)

PRNG deboli, NON USARE @ALEXBASTA, NON USARLI PERFAVORE

Due esempi di PRNG statisticamente accettabili ma crittograficamente insicuri

- `glibc random()`: Utilizza un generatore a feedback lineare (non-linear additive feedback). conoscendo alcuni output si può calcolare facilmente il seed e predire tutti gli output futuri
- **Linear Congruential Generator**: Si ottiene attravero questo algoritmo `r[i] = (a · r[i-1] + b) mod p`. Semplice, veloce, ha buone proprietà statistiche, ma è matematicamente elementare da invertire: dati due output consecutivi, si trovano immediatamente a e b

Un Buon PRNG: Blum Blum Shub (BBS)

Il BBS è un esempio di PRNG crittograficamente sicuro la cui sicurezza è ridotta a un problema matematico difficile

Algorithm 1: Generatore Blum Blum Shub

Scegli due grandi numeri primi p e q tali che $p \equiv q \equiv 3 \pmod 4$;
Calcola $n \leftarrow p \cdot q$;
Scegli un seed s tale che $\gcd(s, n) = 1$;
Calcola $X_0 \leftarrow s^2 \bmod n$;
for $i = 1$ **to** k **do**
    $X_i \leftarrow X_{i-1}^2 \bmod n$;
    $B_i \leftarrow X_i \bmod 2$;
return $\{B_1, B_2, \ldots, B_k\}$;

Proposition 1.1.2 Sicurezza di BBS

Predire il bit successivo della sequenza BBS è computazionalmente equivalente a fattorizzare $n = p \cdot q$

Note:

non esiste nessun algoritmo efficiente noto per fattorizzare numeri grandi pertanto BBS è crittograficamente sicuro

Tuttavia è relativamente lento rispetto ad altri PRNG crittografici. Per applicazioni ad alta velocità si usano altri approcci.

Formalizzare queste note TODO

Gli Stream Cipher: Struttura Generica Uno stream cipher tipico funziona così: a partire dalla chiave K e da un valore di inizializzazione IV, si mantiene uno stato interno s_i che evolve a ogni passo. A ogni passo si produce un byte del keystream z_i tramite una funzione $g(s_i)$, e lo stato avanza con una funzione $f(s_i)$. Il keystream viene poi XOR-ato byte a byte con il plaintext: $c_i = p_i \oplus z_i$. L'IV serve a garantire che lo stesso key K produca keystream diversi in sessioni diverse — un meccanismo simile al nonce che vedremo nei cifrari moderni.
Gli Stream Cipher Non Hanno Perfect Secrecy Una domanda naturale: uno stream cipher può avere perfect secrecy? No, per definizione. La chiave k è corta (il seed), mentre il messaggio è lungo. Per il teorema di Shannon, $|K| \geqslant |P|$ è necessario per la perfect secrecy — condizione violata da qualsiasi stream cipher pratico. Dobbiamo quindi usare una definizione diversa di sicurezza, la sicurezza computazionale: il sistema è sicuro se non esiste nessun algoritmo efficiente in grado di distinguere il keystream G(k) da una sequenza veramente casuale. La sicurezza dello stream cipher dipende interamente dalla qualità del PRG: un PRG computazionalmente indistinguibile dal random implica uno stream cipher computazionalmente sicuro.

RC4

Il RC4 (Rivest Cipher 4) è stato lo stream cipher dominante per decenni, usato in SSL/TLS per HTTPS e in WEP/WPA per le reti Wi-Fi. Oggi, tuttavia, è ufficialmente deprecato ed è considerato insicuro per qualsiasi applicazione crittografica.

Struttura : usa un array di stato S di 256 byte (una permutazione dei valori 0-255) e una chiave di lunghezza variabile tra 8 e 2048 bit

Fase 1 :

Algorithm 2: Inizializzazione di S

```
for i = 0 to 255 do
    S[i] = i;
    T[i] = K[i mod keylen];
j = 0;
for i = 0 to 255 do
    j = (j + S[i] + T[i]) mod 256;
    swap(S[i], S[j]);
```

Note:

Inizializza l'array di 256 byte. Prende la tua chiave segreta (che può variare da 40 a 2048 bit) e la utilizza per "mescolare" l'array

Fase 2 :

Algorithm 3: Generazione del keystream

```
i = 0;
j = 0;
while true do
    i = (i + 1) mod 256;
    j = (j + S[i]) mod 256;
    swap(S[i], S[j]);
    t = (S[i] + S[j]) mod 256;
    output S[t];
```

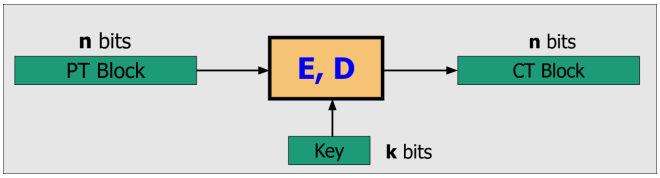
Note:

A ogni ciclo, continua a scambiare due byte all'interno dell'array ed estrae un valore pseudo-casuale in uscita

1.2 Block Ciphers

Lavora su blocchi di lunghezza fissa, puo' generare un po' di delay. I principali sono:

- **DES**: $n = 64$ bits, $k = 56$ bits (ma possiamo anche dire 64, vedremo perche')
- **3DES**: $n = 64$ bits, $k = 3 \cdot 56 = 168$ bits
- **AES**: $n = 128$ bits, $k = 56$ bits



Il block cipher e' costruito per iterazioni:

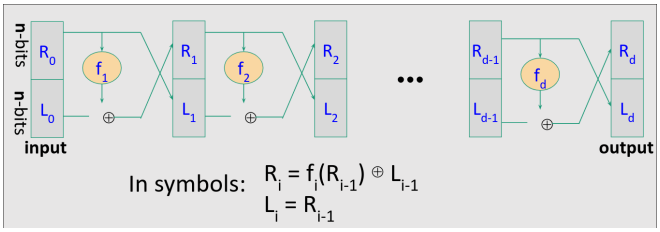
1. Key expansion: partendo da chiave lunga, genero l sottochiavi k_i (dove l e' il numero di iterazioni)

2. Applico la **round function** a m_i e la sottochiave k_i per generare m_{i+1} (dove m_1 e' il plain text)

Il problema e' che e' molto lento rispetto agli altri cifrari, cosa che AES migliora abbastanza rispetto a DES.

1.2.1 DES

L'idea core e' il *Feistel Network*:
Given functions $f_1, ..., f_d : \{0,1\}^n \to \{0,1\}^n$ (non per forza invertibili) dobbiamo costruire una funzione **invertibile** $F : \{0,1\}^{2n} \to \{0,1\}^{2n}$



Come possiamo calcolare $R_i = L_{i-1} + f_i(R_{i-1})$ e $L_i = R_{i-1}$, allora:

> **Proposition 1.2.1**
> Per ogni $f_1, ..., f_d$, il Feistel network is **invertible**.

Dimostrazione per Costruzione: Boh lo dimostra, guarda il libr onon ssi capisce niente.

Per fare la decription, basta applicare lo stesso circuito in ordine opposto, WOW!
La lunghezza della chiave di DES in verita' e' composta da 8 byte, ma l'ottavo bit di ognuno viene usato come parity-check

Chapter 2

Modular Arithmetic

Chapter 3

Asymmetric Criptography

Chapter 4

IPsec and TLS

Chapter 5

Access Control

Chapter 6

Exploits and Patches