
<https://github.com/strausste/cryptography-notebooks>

Canale privato vs canale pubblico:

Setting base:

A vuole inviare un messaggio segreto a B attraverso un **canale privato**.

Assumiamo che, generalmente, l'attaccante M , in un canale privato, non possa fare nulla.

$$A \Rightarrow_x B$$

Su un **canale pubblico**, invece, M può:

- leggere i messaggi
- inviare dei messaggi
- cancellare i messaggi

I **canali privati** sono molto costosi da realizzare e da utilizzare per scambiare grandi quantità di informazioni e per questo ci preoccupiamo della sicurezza nei **canali pubblici**.

$$A \Rightarrow_x^M B$$

Nel canale pubblico, A non invia il messaggio x in chiaro, ma l'**encryption** y di questo messaggio:

$$A \Rightarrow_{E_k(x)}^M B$$

Principio di Kerckhoffs:

L'attaccante M conosce lo schema utilizzato: la sicurezza di un cifrario deve dipendere soltanto dalla **segretezza della chiave**: non è realistico e conveniente pensare di nascondere le implementazioni della funzione di encryption e decryption.

Schema di cifratura:

P è l'insieme dei **plaintexts** $x \in P$

C è l'insieme dei **ciphertexts** $y \in C$

K è l'insieme delle **chiavi** $k \in K$

Su **bitstrings**: $|K| = 2^n$ scala **esponenzialmente**

$Gen()$: algoritmo **probabilistico** per la generazione delle chiavi: $k \leftarrow Gen()$

E : algoritmo **probabilistico** per l'encryption: $E : K \rightarrow P \rightarrow C$

D : algoritmo **deterministico** per la decryption: $D : K \rightarrow C \rightarrow P$

Condizione di correttezza:

$$\forall k . \ k \leftarrow Gen() : \forall x \in P : \quad D_k(E_k(x)) = x$$

Poteri dell'attaccante M :

- EAV: Ciphertext-only attack:
 M conosce soltanto il ciphertext
- KPA: Known Plaintext Attack:
 M conosce delle coppie x e y che sono state cifrate con chiave k (la chiave ovviamente non la conosce)
- CPA: Chosen Plaintext Attack:
 M crea delle coppie (x, y) . Ha accesso alla funzione di encryption $E_k(x)$ (*oracle access*)
- CCA: Chosen Ciphertext Attack:
 M ora ha accesso anche alla funzione di decryption $D_k(y)$ (*oracle access*)

Requisito minimo di sicurezza di un cifrario:

Richiediamo che un cifrario sia almeno **CPA-sicuro** ovvero resistente ad attacchi di tipo CPA (Chosen Plaintext Attacks).

Se è CPA-sicuro, allora è sicuramente anche EAV e KPA sicuro, ma non è detto che sia CCA-sicuro.

Se l'encryption è un algoritmo deterministico e non probabilistico, il cifrario non può essere CPA-sicuro.

Ripasso tipologia funzioni:

- suriettiva: ogni elemento del codominio è immagine di almeno un elemento del dominio (tutto il codominio è immagine)
- iniettiva: il codominio ha immagini distinte (non esistono due valori diversi che producono stessa immagine nel codominio)
- biettiva: sia iniettiva che suriettiva; detta anche biunivoca o invertibile

Se una funzione è iniettiva, può essere resa invertibile riducendo il codominio (suriettività)

Shift cipher su singolo carattere:

$$P = C = K = \mathbb{Z}_{26}$$

$$E_k(x) = (x + k) \bmod 26$$

$$D_k(y) = (y - k) \bmod 26$$

Questo cifrario (shift cipher su **SINGOLO** carattere) è perfetto se e solo se la distribuzione delle chiavi è uniforme

Plaintext x: c
Key k: 4
Ciphertext y: g

Shift cipher ECB-mode:

```
k = gen_k()
x = 'helloworld'

y = ''

for i in x:
    y += encryption(i, k)

print("Plaintext x: %s" % x)
print("Key k:      %s" % k)
print("Chipertext y: %s" % y)
```

```
Plaintext x: helloworld
Key k:      14
Chipertext y: vszzckcfzr
```

Substitution cipher in ECB-mode:

$$P = C = Z_{26}$$

K = insieme di tutte le funzioni biiettive: $Z_{26} \rightarrow Z_{26}$

$\sigma \in K$: le chiavi sono funzioni di permutazione

$E_\sigma(x) = \sigma(x)$ applico la funzione di permutazione

$D_\sigma(y) = \sigma^{-1}$ applico la funzione di permutazione inversa

Esempio:

$\sigma(x)$:

$A \rightarrow M$

$B \rightarrow X$

$C \rightarrow B$

$D \rightarrow Q$

$E \rightarrow P$

$F \rightarrow U$

...

$Z \rightarrow T$

σ è iniettiva: nel codominio (ciphertext) ci sono immagini distinte non potrò mai avere che lettera del cipher corrisponde a più di una lettera del plaintext

σ è suriettiva: ogni elemento del codominio (ciphertext) è immagine di almeno un elemento del dominio (plaintext)

σ è biiettiva: σ è sia suriettiva che iniettiva

È perfetto?

Se usassi il substitution cipher per cifrare un plaintext di un singolo carattere e "buttassi" la chiave, sì, **sarebbe** perfetto.

Ricorda molto lo shift cipher utilizzato su un singolo carattere con chiave casuale.

Infatti lo shift cipher è un caso particolare del substitution cipher con chiave σ che ha una struttura "incrementale".

Utilizzato in ECB-mode:

Questo cifrario utilizzato in ECB-mode, quindi per cifrare un carattere (o blocco di caratteri) con la **stessa chiave** degli altri caratteri (o blocco di caratteri), **non è assolutamente sicuro né tanto meno perfetto**. Infatti questo cifrario viene utilizzato nei giochi della settimana enigmistica.

Dimostriamo che non è perfetto:

$$x = AA$$

$$y = AB$$

$$E_\sigma(AA) = \sigma(A)\sigma(A) \neq AB$$

È un problema che affligge tutti i cifrari **monoalfabetici**: *stesso carattere* di plaintext viene cripratto nello *stesso carattere* di ciphertext: infatti, **i cifrari monoalfabetici non sono perfetti**.

Il cifrario di Vigenere è **polialfabetico**: lo stesso carattere può essere cifrato in modi diversi *infatti storicamente si è dimostrato davvero valido prima dell'arrivo dei calcolatori*.

```
Plaintext x: ciaone
Key k:
{'a': 'u', 'b': 'b', 'c': 'e', 'd': 'j', 'e': 'h', 'f': 'o', 'g': 'a', 'h': 'k', 'i': 'p', 'j': 's', 'k': 'g', 'l': 'w',
'm': 'x', 'n': 'c', 'o': 'f', 'p': 'm', 'q': 'l', 'r': 'd', 's': 'r', 't': 'z', 'u': 'y', 'v': 'n', 'w': 'v', 'x': 'q', 'y':
'i', 'z': 't'}
Ciphertext y: epufch
Decrypted message: ciaone
```

Cifrario di Vigenere:

$$P = C = K = Z_{26}^m$$

$$E_{k_1\dots k_m}(x_1\dots x_m) = (x_1 + k_1 \dots x_n + k_m)$$

$$D_{k_1\dots k_m}(y_1\dots y_m) = (y_1 - k_1 \dots y_n - k_m)$$

Il plaintext x viene spezzato in blocchi di lunghezza m e viene applicata la stessa chiave in ogni blocco.

Se **m=1**, è lo **shift cipher in ECB-mode** non sicuro quindi per plaintext più lunghi di un carattere, perché sto riutilizzando la stessa chiave per caratteri diversi

Sicurezza:

È un cifrario **polialfabetico**: lo stesso carattere del plaintext può corrispondere a caratteri diversi nel ciphertext. Questo cifrario aveva senso prima dei calcolatori (con m molto grande come le parole di un libro). Si può rompere rompendo m shift cipher alla volta e controllando la distribuzione dei caratteri

One-time pad

$$P = C = K = Z_2^n$$

$$E_k(x) = x \oplus k$$

$$D_k(y) = y \oplus k$$

Variabili aleatorie discrete

Una **variabile aleatoria discreta** è una coppia (X, Pr) dove:

X è un insieme finito o numerabile

$$Pr : X \rightarrow [0, 1] \quad \text{soddisfa} \quad \sum_{x \in X} Pr(x) = 1$$

Esempio (lancio di una moneta bilanciata)

$$X = 0, 1$$

$$Pr(0) = Pr(1) = 1/2$$

Evento

Data una variabile aleatoria discreta (X, Pr) , un **evento** è un sottoinsieme di $X : E \subset X$

La probabilità di un evento è $Pr(E) = \sum_{x \in E} Pr(x)$

Esempio (lancio di 3 monete bilanciate)

$$X = \{0, 1\}^3$$

$$Pr(x) = (1/2)^3 = 1/8$$

Evento = "Sono usciti più 1 che 0"

$$E = \{011, 101, 110, 111\}$$

$$Pr(E) = \sum_{x \in E} Pr(x) = 4 * 1/8 = 1/2$$

Probabilità congiunta

Siano (X, Pr) e (Y, Pr) due variabili aleatorie discrete.

La **probabilità congiunta** $Pr(X = x, Y = y)$ è la probabilità che $X = x$ e $Y = y$.

Indipendenza statistica

X e Y sono **indipendenti** se: $\forall x \forall y \quad Pr(X = x, Y = y) = Pr(X = x)Pr(Y = y)$

In altre parole, quando la probabilità congiunta è uguale al prodotto delle probabilità a priori.

Esempio (3 lanci di una moneta bilanciata)

Y = esito del primo lancio

$$Y = \{0, 1\}$$

X = somma dei 3 lanci

$$X = \{0, 1, 2, 3\}$$

Tutti i possibili risultati: $\{000, 001, 010, 011, 100, 101, 110, 111\}$

$Y = 0 : \{000, 001, 010, 011\}$ il primo lancio è 0

$Y = 1 : \{100, 101, 110, 111\}$ il primo lancio è 1

$$Pr(Y = 0) = Pr(Y = 1) = 1/2$$

$$Pr(X = 0) = 1/8$$

$$Pr(X = 1) = 3/8$$

$$Pr(X = 2) = 3/8$$

$$Pr(X = 3) = 1/8$$

X e Y sono indipendenti?

$$\forall x \forall y : Pr(X = x, Y = y) = Pr(X = x)Pr(Y = y)$$

Per dimostrare che non lo sono (il contrario): $\exists x \exists y : Pr(X = x, Y = y) \neq Pr(X = x)Pr(Y = y)$

$Y = 1$ il primo lancio sia 1

$X = 0$ la somma di tutti i lanci sia 0

$$Pr(X = 0, Y = 1) = 0 \text{ è impossibile}$$

$$Pr(X = 0)Pr(Y = 1) = 1/8 * 1/2 = 1/16$$

$$Pr(X = 0, Y = 1) \neq Pr(X = 0)Pr(Y = 1)$$

Quindi X e Y **non** sono indipendenti.

Probabilità condizionata

Siano (X, Pr) e (Y, Pr) due variabili aleatorie discrete.

La **probabilità condizionata** $Pr(X = x|Y = y)$ è la probabilità di avere $X = x$ sapendo che $Y = y$.

$$\text{Probabilità congiunta: } Pr(y, x) = Pr(y)Pr(x|y) = Pr(x, y) = Pr(x)Pr(y|x)$$

Teorema di Bayes

$$Pr(y, x) = Pr(x, y)$$

$$Pr(y)Pr(x|y) = Pr(x)Pr(y|x)$$

Risolvendo per $Pr(x|y)$ si ottiene il **teorema di Bayes**:

$$Pr(x|y) = \frac{Pr(x)Pr(y|x)}{Pr(y)} \quad \text{con } Pr(y) \neq 0$$

Se X e Y sono indipendenti

$$Pr(x, y) = Pr(x)Pr(y|x) \quad [1] \text{ probabilità congiunta}$$

$$\text{Se } x \text{ e } y \text{ sono indipendenti: } Pr(x, y) = Pr(x)Pr(y) \quad [2]$$

Eguagliando la formula [1] e la formula [2], otteniamo:

$$Pr(x)Pr(y) = Pr(x)Pr(y|x)$$

$$Pr(y) = Pr(y|x)$$

Rifacendo i conti con $Pr(y, x)$ otteniamo:

$$Pr(x) = Pr(x|y) \text{ che ricorda la definizione di cifrario perfetto}$$

Esempio 1: Questo cifrario è perfetto?

$$P = \{a, b\}$$

$$C = \{1, 2, 3\}$$

$$K = k_1, k_2$$

$$P(a) = 3/4$$

$$P(b) = 1/4$$

$$P(k_1) = P(k_2) = 1/2 \quad \text{chiavi equiprobabili}$$

$$E_{k_1}(a) = 1$$

$$E_{k_1}(b) = 2$$

$$E_{k_2}(a) = 2$$

$$E_{k_2}(b) = 3$$

Vogliamo dimostrare che non è perfetto:

$$\exists x \in P \quad \exists y \in C : \quad Pr(x|y) \neq Pr(x)$$

Osserviamo che con queste encryption, il carattere del plaintext '**a**' non verrà mai cifrato in '**3**'

$$x = a$$

$$y = 3$$

$$Pr(a|3) = 0$$

$$Pr(a) = 3/4$$

$Pr(a|3) \neq Pr(a) \rightarrow$ il cifrario **non** è perfetto.

Esempio 2: Questo cifrario è perfetto?

$$P = \{a, b\}$$

$$C = \{1, 2\}$$

$$K = \{k_1, k_2\}$$

$$P(k_1) = P(k_2) = 1/2 \quad \text{chiavi equiprobabili}$$

$$P(a) = 3/4$$

$$P(b) = 1/4$$

$$E_{k_1}(a) = 1$$

$$E_{k_1}(b) = 2$$

$$E_{k_2}(a) = 2$$

$$E_{k_2}(b) = 1$$

ricorda lo **XOR** dell' OTP che è perfetto

Vogliamo dimostrare che è perfetto:

$$\text{Dobbiamo dimostrare che } \forall x \in P, \quad \forall y \in C : \quad Pr(x|y) = Pr(x)$$

$$Pr(a|1) = Pr(a) = Pr(a|2)$$

$$Pr(b|1) = Pr(b) = Pr(b|2)$$

Calcoliamo le probabilità condizionate:

- $Pr(a|1)$

$$Pr(a|1) = \frac{Pr(a)Pr(1|a)}{Pr(1)}$$

$Pr(1|a) = \sum_{E_k(a)=1} Pr(k) = Pr(k_1) = 1/2$ la probabilità di avere $y=1$ dato $x=a$ è uguale alla somma delle probabilità delle chiavi quando $E_k(a) = 1$

$Pr(1) = \sum_{E_k(x)=1} Pr(x, k)$ la probabilità di avere $y=1$ è uguale alla somma delle probabilità congiunte della chiave e del plaintext quando $E_k(x) = 1$

Dal momento che x e k sono indipendenti: $Pr(x, k) = Pr(x)Pr(k)$

$$Pr(1) = \sum_{E_k(x)=1} Pr(x)Pr(k)$$

$$Pr(1) = Pr(a)Pr(k_1) + Pr(b)Pr(k_2) = 3/4 * 1/2 + 1/4 * 1/2 = 1/2$$

Sostituendo in $Pr(a|1)$ otteniamo:

$$Pr(a|1) = \frac{1/2Pr(a)}{1/2} = Pr(a)$$

- $Pr(a|2)$

$$Pr(a|2) = \frac{Pr(a)Pr(2|a)}{Pr(2)}$$

$$Pr(2|a) = Pr(k_2)$$

$$Pr(2) = Pr(a)Pr(k_2) + Pr(b)Pr(k_1)$$

$$Pr(a|2) = \frac{Pr(k_2)Pr(a)}{Pr(a)Pr(k_2) + Pr(b)Pr(k_1)} = \frac{1/2Pr(a)}{1/2Pr(a) + Pr(b)} = Pr(a)$$

- $Pr(b|1)$

$$Pr(b|1) = \frac{Pr(1|b)Pr(b)}{Pr(1)} = \frac{Pr(k_2)Pr(b)}{1/2} = Pr(b)$$

- $Pr(b|2)$

$$Pr(b|1) = \frac{Pr(2|b)Pr(b)}{Pr(2)} = \frac{Pr(k_1)Pr(b)}{1/2} = Pr(b)$$

Il cifrario è **perfetto** per ogni scelta del plaintext a prescindere dalla loro probabilità.

Esempio 3: Questo cifrario è perfetto?

$$P = \{a, b\}$$

$$C = \{1, 2\}$$

$$K = \{k_1, k_2\}$$

adesso le chiavi **non** sono più equiprobabili:

$$P(k_1) = 1/3$$

$$P(k_2) = 2/3$$

$$P(a) = P(b) = 1/2 \quad i \text{ plaintext sono equiprobabili}$$

$$E_{k_1}(a) = 1$$

$$E_{k_1}(b) = 2$$

$$E_{k_2}(a) = 2$$

$$E_{k_2}(b) = 1$$

Vogliamo dimostrare che NON è perfetto:

$$\exists x \in P, \quad \exists y \in C : \quad Pr(x|y) \neq Pr(x)$$

$$Pr(a|1) = \frac{Pr(1|a)Pr(a)}{Pr(1)}$$

$$Pr(1) = Pr(a)Pr(k_1) + Pr(b)Pr(k_2) = 1/2(Pr(k_1) + Pr(k_2)) = 1/2$$

$$Pr(1|a) = Pr(k_1) = 1/3$$

$$Pr(a|1) = \frac{1/3Pr(a)}{1/2} = \frac{2}{3}Pr(a)$$

$Pr(a|1) \neq Pr(a) \rightarrow$ il cifrario **non** è perfetto perché le chiavi **non sono uniformemente distribuite**

Segretezza dello shift cipher su singolo carattere

$$P = C = K = Z_{26}$$

$$E_k(x) = x + k$$

$$D_k(y) = y - k$$

$$Gen() \{k \leftarrow Z_{26}\} \quad \text{le chiavi sono equiprobabili}$$

Dimostriamo che è perfetto:

$$\forall x \in P, \quad \forall y \in C : \quad Pr(x|y) = Pr(x)$$

$$Pr(x|y) = \frac{Pr(x)Pr(y|x)}{Pr(y)} \quad (\text{per il teorema di Bayes})$$

$$Pr(y|x) = \sum_{E_k(x)=y} Pr(k) = \frac{1}{26} \quad \text{soltanto una chiave mi da } y \text{ come ciphertext con } x \text{ in input}$$

$$Pr(y) = \sum_{E_k(x)=y} Pr(x, k) = \sum_{E_k(x)=y} Pr(x)Pr(k) \quad x \text{ e } k \text{ sono indipendenti}$$

$$\sum_{E_k(x)=y} Pr(x)Pr(k) = \frac{1}{26} \sum_{E_k(x)=y} Pr(x) = \frac{1}{26} \sum_{x \in Z_{26}} Pr(x) = \frac{1}{26} * 1 = \frac{1}{26} \quad \text{somma di } 1/26 \text{ per 26 volte} = 1$$

$$Pr(x|y) = \frac{1/26Pr(x)}{1/26} = Pr(x)$$

Quindi lo shift cipher su Z_{26} (su singolo carattere) con chiavi uniformemente random è perfetto.

Shift cipher con chiavi non uniformi

$$Pr(k) = \begin{cases} 1/2 & \text{se } k = 25 \\ 1/50 & \text{se } k \neq 25 \end{cases}$$

Dimostriamo che non è perfetto:

$$\exists x \in P, \quad \exists y \in C : \quad Pr(x|y) \neq Pr(x)$$

Proviamo con $x = 0 \quad y = 25$

$$Pr(x = 0|y = 25) = \frac{Pr(y=25|x=0)Pr(x=0)}{Pr(y=25)} \quad \text{teorema di Bayes}$$

$Pr(y = 25|x = 0) = \sum_{E_k(0)=25} Pr(k) = Pr(k = 25) = 1/2$ la probabilità di avere ciphertext = 25 quando il plaintext è 0 è uguale alla somma delle probabilità delle chiavi quando l'encryption di $x = 0$ con quelle chiavi è uguale a 25

$Pr(y = 25) = \sum_{E_k(x)=25} Pr(x)Pr(k)$ la probabilità di avere $y = 25$ è uguale alla somma delle probabilità congiunte della chiave e del plaintext (che sono **indipendenti**) quando $E_k(x) = 25$

$$\begin{aligned} \sum_{E_k(x)=25} Pr(x)Pr(k) &= Pr(x = 0)Pr(k = 25) + \sum_{E_k(x)=25 \wedge x \neq 0} Pr(x)Pr(k) = 1/2Pr(x = 0) + 1/50 \sum_{E_k(x)=25 \wedge x \neq 0} Pr(x) \\ &= 1/2Pr(x = 0) + 1/50(1 - Pr(x = 0)) \end{aligned}$$

$1/2Pr(x = 0) + 1/50(1 - Pr(x = 0)) < 1/2Pr(x = 0) + 1/2(1 - Pr(x = 0))$ sicuramente il primo membro è minore del secondo perché $1/50$ è minore di $1/2$

$$= 1/2Pr(x = 0) + 1/2 - 1/2Pr(x = 0) = 1/2 \quad \text{sviluppando il prodotto precedente}$$

Quindi, lo shift cipher per un singolo bit con chiavi non uniformi **non è perfetto**.

Perfect secrecy:

Assumiamo che $\forall k \in K : Pr(k) > 0$ ogni chiave abbia una probabilità non nulla

Assumiamo che K , l'insieme delle chiavi, sia finito

Assumiamo che P , l'insieme dei plaintext, sia finito oppure numerabile. Se P è finito, allora $|P| > 1$

Assumiamo che $\forall x \in P : Pr(x) > 0$

La distribuzione di probabilità delle chiavi dipende dall'algoritmo $Gen()$.

La distribuzione di probabilità dei plaintext dipende dai linguaggi naturali.

La distribuzione di probabilità dei ciphertext dipende dal contesto, dalla probabilità di $Gen()$ e dall'encryption.

Esiste qualche $y \in C$ tale che $Pr(C = y) = 0$?

Questo è possibile quando $\forall x, \forall k : E_k(x) \neq y$

Per avere $Pr(C = y) > 0$ per ogni $y \in C$ restringiamo l'insieme dei ciphertext:

$$C = \{E_k(x) \mid x \in P, k \in K\}$$

Definizione:

Uno schema (Gen, E, D) di encryption con distribuzioni di probabilità K, P, C

è perfetto (**perfectly secret**) $\iff \forall x \in P, \forall y \in C : Pr(x|y) = Pr(x)$

Esempio cifrario non perfetto: Shift cipher in ECB-mode con lunghezza 2:

$$P = C = Z_{26}^2$$

Z_{26} la chiave è un singolo carattere e questo viene usato per ogni carattere del plaintext

$Gen() = \{k \leftarrow K\}$ chiavi uniformemente distribuite

$$E_k(x_1x_2) = (x_1 + k \quad x_2 + k)$$

$$D_k(y_1y_2) = (y_1 - k \quad y_2 - k)$$

È perfetto (perfectly secret)?

La congettura è che non sia perfetto. Dimostriamo allora che **non** è perfetto:

La perfect secrecy dice: $\forall x \in P, \forall y \in C : Pr(x|y) = Pr(x)$

La negazione della perfect secrecy è: $\exists x \in P, \exists y \in C : Pr(x|y) \neq Pr(x)$

Abbiamo assunto $Pr(x) > 0$, quindi per trovare una $Pr(x|y)$ che sicuramente sia diversa da $Pr(x) > 0$, cerchiamo una $Pr(x|y) = 0$

Prendiamo:

$$x = (0, 0)$$

È **impossibile** avere, per esempio, $y = (0, 1)$ con $x = (0, 0)$ visto che **la chiave è la stessa** per ogni carattere del plaintext.

$$E_k(x) = (0 + k, 0 + k) = (k, k) \neq (0, 1)$$

$$\nexists k \in K : E_k(x) = y$$

Abbiamo trovato x, y tali che $Pr(x|y) \neq Pr(x)$ possiamo allora dire che il cifrario **non** è perfetto

Esempio cifrario non perfetto: Shift cipher su singolo carattere, ma con lo spazio delle chiavi inferiore all'insieme dei plaintext

$$P = C = \mathbb{Z}_{26}$$

$$K = \mathbb{Z}_4$$

$Gen() = \{k \leftarrow K\}$ chiavi uniformemente distribuite

$$E_k(x) = x + k$$

$$D_k(y) = y - k$$

È perfetto (perfectly secret)?

La congettura è che non sia perfetto:

$$\exists x \in P, \exists y \in C : Pr(x|y) \neq Pr(x)$$

Prendiamo:

$$x = 0$$

$$y = A \text{ oppure } 10 \text{ in } \mathbb{Z}_{16}$$

$E_K(x) = \{E_k(x) : k \in K\} = \{E_0(x), E_1(x), E_2(x), E_3(x)\}$ tutte le possibili encryption di x

$$E_K(0) = \{0, 1, 2, 3\}$$

$$y = A \notin E_K(0) = \{0, 1, 2, 3\}$$

$Pr(x|y) = 0 \neq Pr(x) > 0$ quindi il cifrario **non** è perfetto perchè lo spazio delle chiavi è più piccolo dell'insieme dei plaintext. Per essere perfetto dovrebbe avere lo spazio delle chiavi grande almeno quanto lo spazio dei plaintext (*condizione necessaria ma non sufficiente*).

Teorema di Shannon

Uno schema di encryption con $|P| = |C| = |K|$ è perfetto sse:

1. $\forall x \in P \quad \forall y \in C \quad \exists! k \in K : E_k(x) = y$ **un'unica chiave** che dia quel ciphertext dato quel plaintext
2. $\forall k \in K : Pr(k) = \frac{1}{|K|}$ chiavi uniformemente distribuite

Lemma:

π schema di cifratura.

P insieme plaintexts.

K insieme delle chiavi.

Se π è perfetto, allora $|P| \leq |K|$

Dimostrazione:

Per assurdo, supponiamo che $|K| < |P|$.

Dal momento che E_k è iniettiva, allora $|P| \leq |C|$

Scegliamo un $x \in P$, per ipotesi: $Pr(x) > 0$.

$|E_K(x)| = \{E_k(x) \mid k \in K\}$ encryption per ogni chiave

$$|E_K(x)| \leq |K| < |P| \leq |C|$$

$|K| < |P|$ ipotesi per assurdo

$|P| \leq |C|$ ipotesi iniettività

Quindi: $\exists y \in C$ and $y \notin E_K(x)$

$$Pr(x|y) = 0 < Pr(x) \rightarrow \pi \text{ non è perfetto.}$$

Esercizio #1:

$$P = \bigcup_{l \in 1..n} \{0, 1\}^l \quad n \text{ è il limite superiore}$$

$$Gen() = \{k \leftarrow \{0, 1\}^n\} \quad \text{chiavi equiprobabili}$$

$$E_k(x) = k_{1..|x|} \oplus x \quad \text{taglia } k \text{ alla lunghezza di } x$$

L'encryption rivela la lunghezza del messaggio: l'attaccante sceglie $x_0 = 0$ $x_1 = 00$ e vince sempre: $Pr(PrivK_{M,\pi}^{eav} = 1) = 1$

Esercizio #2.1

$$P = 0..1$$

$$C = 0..2$$

$$Gen() = \{k \leftarrow 0..1\}$$

$$E_k(x) = x + k \bmod 3$$

$Pr(0|2) = 0 \neq Pr(0) \rightarrow \text{non è perfetto}$ e quindi non è neanche perfectly indistinguishable: è impossibile avere 2 come ciphertext se il plaintext era 0 (la chiave è 0 oppure 1)

Esperimento di indistinguishability:

$$M: x_0 = 0 \quad x_1 = 1$$

$$M: b' = y == 2 ? 1 : 0$$

$$Pr(PrivK_{M,\pi}^{eav} = 1) = Pr(b = 0)Pr(b' = 0|b = 0) + Pr(b = 1)Pr(b' = 1|b = 1)$$

$$Pr(b' = 0|b = 0) = 1 \quad \text{in questo caso indovina sempre}$$

$$Pr(b' = 1|b = 1) = 1/2 \quad \text{quando esce 1, non può fare altro che tirare a indovinare}$$

$$Pr(PrivK_{M,\pi}^{eav} = 1) = 1/2(1 + 1/2) > 1/2 \rightarrow \text{non è perfectly indistinguishable}$$

Esercizio #2.2

$$P = 0..2$$

$$C = 0..2$$

$$Gen() = \{k \leftarrow 0..2\}$$

$$E_k(x) = x + k \bmod 3$$

È lo shift cipher su $Z_3 \rightarrow$ è perfetto

Esercizio #2.3

$$P = 0..1$$

$$C = 0..2$$

$$Gen() = \{k \leftarrow 0..2\}$$

$$E_k(x) = x + k \bmod 3$$

Shannon:

È perfetto sse

1. $\forall x \in P \quad \forall y \in C \quad \exists! k \in K : \quad E_k(x) = y$
2. $\forall k \in K : \quad Pr(k) = \frac{1}{|K|}$

$$x = 0 \quad E_0(0) = 0 \quad E_1(0) = 1 \quad E_2(0) = 2$$

$$x = 1 \quad E_0(1) = 1 \quad E_1(1) = 2 \quad E_2(1) = 0$$

Il cifrario è perfetto.

Esercizio #2.4

$$P = 0..1$$

$$C = 0..1$$

$$Gen() = \{k \leftarrow 0..2\}$$

$$E_k(x) = x + k \bmod_2$$

$$x = 0 \quad E_0(0) = 0 \quad E_1(0) = 1 \quad E_2(0) = 0$$

$$x = 1 \quad E_0(1) = 1 \quad E_1(1) = 0 \quad E_2(1) = 1$$

Se $y = 0$ è più probabile che $x = 0$

Se $y = 1$ è più probabile che $x = 1$

Eperimento di indistinguishability:

$$M : \quad x_0 = 0 \quad x_1 = 1$$

$$\text{strategia di } M: \quad b' = (y == 0) ? 0 : 1$$

$$Pr(b' = 0 | b = 0) = 2/3$$

$$Pr(b' = 1 | b = 1) = 2/3$$

$$Pr(PrivK = 1) = 1/2(2/3 + 2/3) > 1/2 \rightarrow \text{non è perfetto!}$$

Esercizio #3: Shift cipher with non-uniform keys

$$P = C = K = Z_{26}$$

$$E_k(x) = x + k \bmod_{26}$$

$$Gen() = \{\text{let } h \leftarrow \{0, 1\} \quad h == 0 ? 25 : 0..24\} \quad 1/2 \text{ probabilità che sia } 25; 1/50 \text{ che sia un numero da } 0 \text{ a } 24$$

$$Pr(k = 25) = 1/2$$

$$Pr(k) = 1/50 \quad k \neq 25$$

$$M \rightarrow A : \quad x_0 = 0 \quad x_1 = 1$$

$$\text{strategia di } M \quad b' = y == 25 ? 0 : 1$$

$$Pr(b' = 0 | b = 0) = \sum Pr(K = k) \mid E_k(0) = 25 \quad \text{la somma della probabilità delle chiavi che mi danno quel ciphertext con } x = 0 \\ = Pr(K = 25) = 1/2 \quad \text{c'è soltanto } K = 25$$

$$Pr(b' = 1 | b = 1) = 1 - Pr(b' = 0 | b = 1) = 1 - Pr(K = 24) = 1 - 1/50 = 49/50$$

$$Pr(PrivK = 1) = 1/2(1/2 + 49/50) = 74/100 \rightarrow \text{non è perfectly indistinguishable}$$

Esercizio #4: Vigenere cipher with non-uniform keys

$$P = C = K = Z_{26}^2$$

$$E_{k_1 k_2}(x_1 x_2) = (x_1 + k_1 \quad x_2 + k_2)$$

$$Gen() = \{m \leftarrow \{0, 1\}; \quad k_1 \leftarrow 0..25; \quad \text{if } m = 0 \text{ then } k_2 = k_1 \text{ else } k_2 \leftarrow 0..25\}$$

$$M \rightarrow A : \quad x_0 = aa \quad x_1 = ab$$

$$A \rightarrow M : \quad y = y_1 \quad y_2 = E_k(x_b) \quad b \leftarrow \{0, 1\}$$

strategia di M : $b' = y_1 == y_2 ? 0 : 1$ se i due caratteri del ciphertext sono uguali, dice 0; altrimenti dice 1

$$Pr(b' = 0 | b = 0) \quad \text{probabilità che } M \text{ dica 0 quando } b = 0 \rightarrow Pr(y_1 == y_2 | x_b = aa)$$

Ci sono due casi:

- $m = 0 \rightarrow k_1 == k_2$
- $m = 1 \rightarrow k_1 \text{ è random e anche } k_2 \text{ è random}$

$$Pr(y_1 == y_2 | b = 0, m = 0 \vee m = 1) = Pr(m = 0)Pr(b' = 0 | b = 0, m = 0) + Pr(m = 1)Pr(b' = 0 | b = 0, m = 1)$$

$$= 1/2(Pr(b' = 0 | b = 0, m = 0) + Pr(b' = 0 | b = 0, m = 1))$$

$Pr(m = 0) = Pr(m = 1) = 1/2$ praticamente è il lancio di una moneta

$Pr(b' = 0 | b = 0, m = 0) = 1$ M indovina sempre: $b_m = aa$ e $k_1 = k_2 \rightarrow y_1 = y_2$

$Pr(b' = 0 | b = 0, m = 1) = \frac{26}{26*26} = 1/26$ è la probabilità di avere $y_1 = y_2$ quando $b = 0$ ($x_b = aa$) e k_1 e k_2 , generate entrambe in maniera random, risultino uguali

$$= 1/2(1 + 1/26) \approx 0.52; \text{ mettendo in evidenza}$$

$Pr(b' = 1 | b = 1)$ probabilità che M dica 1 ($y_1 \neq y_2$) quando $b = 1$ (ovvero quando $x_b = ab$)

$$= 1 - Pr(b' = 0 | b = 1) = 1 - Pr(m = 0)Pr(b' = 0 | b = 1, m = 0) + Pr(m = 1)Pr(b' = 0 | b = 1, m = 1)$$

$$= 1 - 1/2(Pr(b' = 0 | b = 1, m = 0) + Pr(b' = 0 | b = 1, m = 1))$$

$$= 1 - 1/2(0 + 1/26) = 1 - 1/52 \approx 0.98$$

$Pr(PrivK = 1) = 1/2(Pr(b' = 0 | b = 0) + Pr(b' = 1 | b = 1)) = 1/2(0.52 + 0.98) \approx 0.75 > 1/2 \rightarrow$ **non è perfectly indistinguishable**

Esercizio #5

Exercise #5

$$\Pr(y) = \sum_{E_k(x)=y} \Pr(k) Pr(x) \quad \Pr(y|x) = \sum_{E_k(x)=y} \Pr(k)$$

$E_k(x)$	a	b	c
k_1	1	2	3
k_2	3	1	2
k_3	2	3	1

$$\begin{array}{lll}
 \Pr(a) = \Pr(k_1) = 1/2 & \Pr(b) = \Pr(c) = \Pr(k_2) = \Pr(k_3) = 1/4 \\
 \Pr(1|a) = ?? & \Pr(2|a) = ?? & \Pr(3|a) = ?? \\
 \Pr(1|b) = ?? & \Pr(2|b) = ?? & \Pr(3|b) = ?? \\
 \Pr(1|c) = ?? & \Pr(2|c) = ?? & \Pr(3|c) = ?? \\
 \Pr(1) = ?? & \Pr(2) = ?? & \Pr(3) = ?? \\
 \end{array}$$

$$\Pr(a|1) = (\Pr(a) \Pr(1|a)) / \Pr(1) = ??$$

$$\Pr(1|a) = \sum_{E_k(a)=1} \Pr(k) = \Pr(k_1) = 1/2$$

$$\Pr(2|a) = \sum_{E_k(a)=2} \Pr(k) = \Pr(k_3) = 1/4$$

$$\Pr(3|a) = \sum_{E_k(a)=3} \Pr(k) = \Pr(k_2) = 1/4$$

$$\Pr(1|b) = \sum_{E_k(b)=1} \Pr(k) = \Pr(k_2) = 1/4$$

$$\Pr(2|b) = \sum_{E_k(b)=2} \Pr(k) = \Pr(k_1) = 1/2$$

$$\Pr(3|b) = \sum_{E_k(b)=3} \Pr(k) = \Pr(k_3) = 1/4$$

$$\Pr(1|c) = \sum_{E_k(c)=1} \Pr(k) = \Pr(k_3) = 1/4$$

$$\Pr(2|c) = \sum_{E_k(c)=2} \Pr(k) = \Pr(k_2) = 1/4$$

$$\Pr(3|c) = \sum_{E_k(c)=3} \Pr(k) = \Pr(k_1) = 1/2$$

$$\Pr(1) = \sum_{E_k(x)=1} \Pr(k) Pr(x) = \Pr(k_1) Pr(a) + \Pr(k_2) Pr(b) + \Pr(k_3) Pr(c) = (1/2 * 1/2) + (1/4 * 1/4) + (1/4 * 1/4) = 3/8$$

$$\Pr(2) = \sum_{E_k(x)=2} \Pr(k) Pr(x) = \Pr(k_1) Pr(b) + \Pr(k_2) Pr(c) + \Pr(k_3) Pr(a) = (1/2 * 1/4) + (1/4 * 1/4) + (1/4 * 1/2) = 5/16$$

$$\Pr(3) = \sum_{E_k(x)=3} \Pr(k) Pr(x) = \Pr(k_1) Pr(c) + \Pr(k_2) Pr(a) + \Pr(k_3) Pr(b) = (1/2 * 1/4) + (1/4 * 1/2) + (1/4 * 1/4) = 5/16$$

$$\Pr(a|1) = \frac{\Pr(a) \Pr(1|a)}{\Pr(1)} = \frac{1/2 * 1/2}{1/2} = 1/2 \quad \text{teorema di Bayes}$$

Exercise #6

Let $\mathbf{P} = \mathbf{C} = \mathbf{K} = \mathbb{Z}_{26}^2$

$\text{Gen}() \{ k \leftarrow \mathbb{Z}_{26}^2 \}$

$$E_{k_1 k_2}(x_1 x_2) = \begin{cases} (x_1 + k_1, x_2 + k_2) & \text{if } k_1, k_2 \neq 0 \\ (x_1 + x_2 + 1, x_1 - x_2) & \text{otherwise} \end{cases}$$

Il cifrario è perfetto?

Analisi sulla funzione di encryption:

Quando k_1 e k_2 sono diversi da 0, il cifrario è utilizzato in maniera OTP, quindi è **perfetto**.

Bisogna però analizzare il caso quando k_1 oppure k_2 (o entrambi) sia uguale a 0.

Se scegliamo un plaintext dove $x_1 = x_2$ (come $x = aa$), nel caso che uno dei due caratteri della chiave sia 0, possiamo notare che il secondo carattere del ciphertext sarà sempre 'a' ($x_1 - x_2$): nel caso dell'esperimento di perfect indistinguishability, sarà questa la strategia dell'attaccante: scegliere $x_0 = aa$ e $x_1 = ab$ e scegliere x_0 nel caso che il secondo carattere del ciphertext (y_1 sia uguale a a).

Definizione di perfect secrecy:

La congettura è che non sia perfetto:

$\exists x \in P, \exists y \in C : Pr(x|y) \neq Pr(x)$

Scegliamo come $x = da$

Scegliamo come $y = aa$

È impossibile che se il ciphertext sia aa in quanto per avere lo stesso carattere del plaintext uguale nel ciphertext, il carattere corrispondente della chiave deve essere 0: ma se un carattere è 0, l'encryption diventa $(x_1 + x_2 + 1, x_1 - x_2)$: nel caso in cui il plaintext sia da , il secondo carattere del ciphertext, $(d - a)$, non è a , quindi è **impossibile** ottenere questo ciphertext dato quel plaintext:

$Pr(da|aa) = 0$

$Pr(da)$ abbiamo assunto essere > 0

$Pr(da|aa) \neq Pr(da) \rightarrow$ il cifrario **non è perfetto**

```
x = 'aa'
k = NoZeroInKey().gen()

y = NoZeroInKey().enc(x,k)
dec_x = NoZeroInKey().dec(y, k)

print("Plaintext x: %s" % x)
print("Key k: %s" % k)
print("Chipertext y: %s" % y)
print("Decrypted x: %s" % dec_x)
```

```
Plaintext x: aa
Key k: ['7', '15']
Chipertext y: hp
Decrypted x: aa
```

$Pr(da|aa)$:

```
x = 'da'
found = False # if 'aa' is found in the ciphertexts

# All the possible keys (0,0) up to (25,25)
for k1 in range(26):
    for k2 in range(26):
        k = [str(k1),str(k2)]
        y = NoZeroInKey().enc(x,k)

        # print(k[0], " ", k[1], " ", y)

        if(y == 'aa'):
            found = True

if (found):
    print("y='aa' has been found!")
else:
    print("y='aa' has NOT been found!")

y='aa' has NOT been found!
```

Esercizio #7

$$P = C = \mathbb{Z}_{26}$$

$$K = \mathbb{Z}_4$$

$$Gen() = \{k \leftarrow K\}$$

$$E_k(x) = x + k$$

$$D_k(y) = y - k$$

Il cifrario è perfetto?

La congettura è che **non sia perfetto**:

$$\exists x \in P, \exists y \in C : Pr(x|y) \neq Pr(y)$$

$$x = 0$$

$$y = F$$

Tutte le possibili encryption di $x = 0$:

$$E_0(0) = 0$$

$$E_1(0) = 1$$

$$E_2(0) = 2$$

$$E_3(0) = 3$$

$$\nexists k \in K : E_k(0) = F \rightarrow Pr(0|F) = 0 \rightarrow \text{il cifrario non è perfetto}$$

Esercizio #8

$$P = C = Z_{26}^2$$

$$K = \{\sigma \in Z_{26} \rightarrow Z_{26} \mid \sigma \text{ sia biiettiva}\}$$

$$E_\sigma(x_1 x_2) = \sigma(x_1) \ \sigma(\sigma(x_1) + x_2)$$

Trova la **decryption** e dimostra che è un cifrario:

$$x_1 = \sigma^{-1}(y_1)$$

$$x_2 = \sigma^{-1}(y_2) - \sigma(x_1)$$

$$D_{\sigma^{-1}}(y_1 y_2) = \sigma^{-1}(y_1) \ \sigma^{-1}(y_2) - \sigma(x_1) \quad \text{con } x_1 = \sigma^{-1}(y_1)$$

$$\begin{aligned} D_{\sigma^{-1}}(E_\sigma(x_1 x_2)) &= D_{\sigma^{-1}}(\sigma(x_1) \ \sigma(\sigma(x_1) + x_2)) = \sigma^{-1}(\sigma(x_1)) \ \sigma^{-1}(\sigma(\sigma(x_1) + x_2)) - \sigma(x_1) = \\ &= \cancel{\sigma^{-1}(\sigma(x_1))} \ \cancel{\sigma^{-1}(\sigma(\sigma(x_1) + x_2))} - \sigma(x_1) = x_1 \ \sigma(x_1) + x_2 - \sigma(x_1) = x_1 - x_2 \end{aligned}$$

```
x = 'ty'
k = SigmaCipher().gen()
y = SigmaCipher().enc(x,k)

print("Plaintext x: %s" % x)
print("Key k: \n %s" % k)
print("Chipertext y: %s" % y)

decrypted_x = SigmaCipher().dec(y,k)

print("Decrypted message: %s" % decrypted_x)

Plaintext x: ty
Key k:
{'a': 'e', 'b': 'x', 'c': 'b', 'd': 'a', 'e': 'g', 'f': 'p', 'g': 'n', 'h': 'o', 'i': 'u', 'j': 'l', 'k': 'w', 'l': 's',
'm': 'y', 'n': 'c', 'o': 'q', 'p': 'r', 'q': 'm', 'r': 'd', 's': 'h', 't': 'k', 'u': 'i', 'v': 'v', 'w': 'f', 'x': 'z', 'y':
't', 'z': 'j'}
Chipertext y: ku
Decrypted message: ty
```

Verifica correttezza del cifrario:

```
ciph = SigmaCipher()
ciph.dec(ciph.enc(x,k),k) == x
```

True

Dimostriamo che non è perfetto:

Per dimostrare che non è perfetto: $\exists x \in P, \ \exists y \in C : \ Pr(x|y) \neq Pr(x)$

$$x = aa$$

$$y = ab$$

Per avere a nel primo carattere del ciphertext, la chiave σ deve mappare $a \rightarrow a$.

Con la chiave così definita, il secondo carattere sarà mappato in $\sigma(\sigma(x_1) + x_2) = \sigma(\sigma(a) + a) = \sigma(a + a) = \sigma(a) = a$

Il ciphertext ottenuto sarà sempre aa , rendendo quindi **impossibile** avere come ciphertext ab (ricordiamo che la funzione σ è biiettiva (iniettiva e suriettiva)).

$$Pr(aa) > 0$$

$$Pr(aa|ab) = 0 \rightarrow \text{il cifrario non è perfetto.}$$

```

: # Changing the gen() function in order to have the 'a' always being mapped to 'a'

alphabet_without_a = ['b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u',
alphabet_without_a_random = alphabet_without_a[:] # copying a List by value (because the default is by reference) [source:
random.shuffle(alphabet_without_a_random)

altered_key = {alphabet_without_a[i] : alphabet_without_a_random[i] for i in range(len(alphabet_without_a))}

altered_key['a'] = 'a' # 'a' forced to being mapped to 'a'

# altered_key

x = 'aa'
y = SigmaCipher().enc(x,altered_key)

print("Plaintext x: %s" % x)
print("Key k:      \n%s" % altered_key)
print("Ciphertext y: %s" % y) # The ciphertext is always 'aa'

decrypted_x = SigmaCipher().dec(y,altered_key)

print("Decrypted message: %s" % decrypted_x)

```

```

Plaintext x: aa
Key k:
{'b': 'g', 'c': 'o', 'd': 'w', 'e': 'n', 'f': 'p', 'g': 'y', 'h': 'x', 'i': 'z', 'j': 'b', 'k': 'i', 'l': 'd', 'm': 'r',
'n': 'j', 'o': 'f', 'p': 'm', 'q': 'c', 'r': 's', 's': 'v', 't': 'k', 'u': 'h', 'v': 't', 'w': 'l', 'x': 'u', 'y': 'q', 'z':
'e', 'a': 'a'}
Ciphertext y: aa
Decrypted message: aa

```

Esercizio #9 (Quasi-OTP)

È come l'OTP, ma non ammette la chiave dove tutti i caratteri sono 0.

Con il **teorema di Shannon** si dimostra facilmente che **non è perfetto**.

Esperimento di perfect indistinguishability:

$$x_0 = 0^n$$

$$x_1 = 1^n$$

$M : b' = (y == 0^n) ? 1 : 0$ se i caratteri del ciphertext sono tutti 0, M dirà 1 (ovvero 1^n) altrimenti dirà 0

$Pr(b' = 0 | b = 0) = 1$ probabilità che M dica 0 quando $b = 0$ (ovvero quando $x_b = 0^n$). M indovina sempre in questo caso perché non esiste la chiave 0ⁿ

$Pr(b' = 1 | b = 1) = \frac{1}{2^{n-1}}$ in questo caso, la scelta è completamente casuale

$$Pr(PrivK = 1) = \frac{1}{2} \left(1 + \frac{1}{2^{n-1}}\right) > 1/2$$

M vince di una quantità **negligible**.

Definizione:

$$\exists x \in P, \exists y \in C : Pr(x|y) \neq Pr(x)$$

$$x = 0^n$$

$$y = 0^n$$

$$Pr(x|y) = 0$$

Perfect indistinguishability

Caratterizzazione alternativa (ma equivalente) della perfect secrecy.

Esperimento di indistinguishability

$PrivK_{M,\pi}^{eav}$

$PrivK \rightarrow$ crittografia chiave privata

$\pi \rightarrow$ nome del cifrario $\rightarrow \pi = (Gen, E, D)$

$eav \rightarrow$ eavesdropper \rightarrow attaccante più debole (può soltanto guardare il canale pubblico)

$M \rightarrow$ Mallory, attaccante

1. $M \rightarrow A : x_0, x_1$ attaccante M invia due plaintext ad A
2. $A : k \leftarrow Gen()$ A genera una chiave con l'algoritmo $Gen() \in \pi$
 $b \leftarrow \{0, 1\}$ A genera un bit \mathbf{b} uniformemente random
3. $A \rightarrow M : y \leftarrow E_k(x_b)$ A invia ad M un ciphertext ottenuto da uno dei 2 plaintext (scelto in base a \mathbf{b})
4. $M : b' \leftarrow \dots$ M vuole calcolare \mathbf{b} in modo che $\mathbf{b}' = \mathbf{b}$ (vuole indovinare \mathbf{b})

L'esperimento ha successo ($PrivK_{M,\pi}^{eav} = 1$) se $\mathbf{b}' = \mathbf{b}$

$$Pr(PrivK_{M,\pi}^{eav} = 1) = Pr(b = 0)Pr(b' = 0|b = 0) + Pr(b = 1)Pr(b' = 1|b = 1)$$

Cifrario perfectly indistinguishable

π è **perfectly indistinguishable** (e anche perfetto) se:

$\forall M : Pr(PrivK_{M,\pi}^{eav} = 1) = 1/2$ **per ogni avversario M , la probabilità che l'esperimento abbia successo è 1/2** (anche il migliore avversario è costretto a tirare a indovinare come sua migliore strategia)

Qualunque M vuol dire anche l'avversario con HW illimitato e tempo illimitato (irrealistico)

Esercizio

$P = C = Z_{26}$ due caratteri in Z_{26}

$K = Z_{26}$ chiave con un singolo carattere

$Gen() : k \leftarrow K$ chiave uniformemente random

$E_k(xx') = (x+k)(x'+k)$ la stessa chiave viene utilizzata per cifrare caratteri diversi \rightarrow sicuramente non è un cifrario perfetto

1. $M \rightarrow A : x_0, x_1$ $x_0 = 00$ $x_1 = 01$ M invia questi due messaggi ad A
2. $A : k \leftarrow Gen()$ $b \leftarrow \{0, 1\}$ A genera il bit random \mathbf{b}
3. $A \rightarrow M : y \leftarrow E_k(x_b)$ $y = zz'$ A fa l'encryption di un messaggio (scelto in base a \mathbf{b}) che è composto da due caratteri z e z'
4. $M : b' \leftarrow \dots$ M deve indovinare \mathbf{b}

$b' = \{0 \text{ se } z=z'; 1 \text{ altrimenti}\}$ la sua strategia è: dire che è il messaggio 0 (caratteri uguali nel plaintext) se nel ciphertext ci sono i caratteri uguali, 1 altrimenti

$Pr(PrivK_{M,\pi}^{eav} = 1) = 1$ con questa strategia, la probabilità di indovinare è 1 \rightarrow indovina sempre (passando quei messaggi)

$$Pr(PrivK_{M,\pi}^{eav} = 1) = Pr(b' = 0, b = 0) + Pr(b' = 1, b = 1) = Pr(b' = 0|b = 0)Pr(b = 0) + Pr(b' = 1|b = 1)Pr(b = 1)$$

$$= 1/2Pr(b' = 0|b = 0) + 1/2Pr(b' = 1|b = 1) = 1/21 + 1/21 = 1 \neq 1/2 \rightarrow$$
 il cifrario **non è** perfectly indistinguishable!

Uncipher:

```
# Adversary for the Uncipher:

class MalloryUncipher(Mallory):

    def plaintexts(self):
        return ("a","b")

    def guess(self,y):
        if y=='a':
            bm = 0
        else:
            bm = 1

    return bm
```

Percentage of success: 100.0

Shift cipher in ECB-mode (with uniform keys and plaintexts of arbitrary length):

```
# Adversary for the ShiftECB:

class MalloryShiftECB(Mallory):

    def plaintexts(self):
        return ("aa","ab")

    def guess(self,y):
        if y[0]==y[1]:
            bm = 0
        else:
            bm = 1
    return bm
```

Percentage of success: 100.0

Shift cipher with non-uniform keys and plaintexts of length 1:

```
# Adversary for the Shift1Unbal:

class MalloryShift1Unbal(Mallory):

    def plaintexts(self):
        return ('a','b')

    def guess(self,y):
        if y=='z': # if y=='z', M says the original message *probably* was 'a' (the message 0) because 25 is the most pr
                    # probable message
            bm = 0
        else:
            bm = 1

    return bm
```

Percentage of success: 72.0

L'avversario non indovina sempre (100%) ma la probabilità che l'esperimento abbia successo è intorno al 75% che è comunque maggiore del 50% come richiede la definizione.

Quindi il cifrario Shift1Unbal (Shift cipher su singolo carattere, ma con **chiavi non uniformi**) **non è perfectly indistinguishable** (e quindi neanche perfetto).

Shift cipher in OTP-mode for the first n chars, then uncipher

```
# Adversary for the ShiftLazyOTP:

class MalloryShiftLazyOTP(Mallory):

    def plaintexts(self):
        return ("aaaaaa", "aaaaab")

    def guess(self,y):
        if y[5]=="a": # M knows that after n characters, the encryption performs a Uncipher (Kerckhoffs' assumption)
            bm = 0
        else:
            bm = 1

        return bm
```

Percentage of success: 100.0

M con questi due plaintext indovina sempre, perchè sa (**per il principio di Kerckhoffs**) che dopo n caratteri l'encryption non lavora più in OTP-mode, ma dall'(n+1)-esimo carattere, lascia i successivi così come sono anche nel ciphertext (Uncipher): questo è il motivo per il quale nella sua guess(), l'avversario controlla il 6 carattere (in posizione 5): se è una a, come nel primo messaggio, allora il messaggio cifrato è proprio il primo; altrimenti è il messaggio con indice 1.

Vigenere cipher with non-uniform keys and plaintexts of length 2

```
# Adversary for the Vigenere2Unbal:

class MalloryVigenere2Unbal(Mallory):

    def plaintexts(self):
        return ("aa", "ab")

    def guess(self,y):
        if y[0]==y[1]:
            bm = 0
        else:
            bm = 1

        return bm
```

Percentage of success: 72.0

Distribuzione delle chiavi non uniforme → **non indistinguibile e quindi non perfetto**

OTP

```
import random

class MalloryOTP(Mallory):
    # An adversary that randomly says 0 or 1

    def plaintexts(self):
        return ("0000000", "0101010") # they don't matter against OTP: M can only randomly guess

    def guess(self,y):
        return random.randint(0,1)
```

Percentage of success: 50.139

As we increase the number of experiments, the probability tends to 50%

OTPlastXOR: OTP where the last bit of the key if the XOR of the previous bits

```
# Adversary for OTPlastXor

class MalloryOTPlastXor(Mallory):

    def plaintexts(self):
        return ("000","001")

    def guess(self,y):
        n = 0
        for bi in y[:-1]: # (Excluding the Last bit)
            n = n ^ int(bi) # The XOR of the previous bits in y

        if n==int(y[-1]): # If the XOR of the previous bits is equal to the last bit of the ciphertext
            bm = 0
        else:
            bm = 1
        return bm
```

Percentage of success: 100.0

L'avversario vince sempre: l'algoritmo gen() rivela troppe informazioni sul keystream: l'avversario infatti si limita a replicare la strategia sul ciphertext (controlla lo XOR dei bit precedenti all'ultimo: questo è in comune con la chiave: quello che cambia è l'ultimo bit che non è stato generato in maniera casuale, infatti è lo XOR del keystream, e mettendo nei plaintext 0 e 1 come ultimo bit può portare avanti la strategia vincente!).

Ovviamente, questo cifrario non è **perfectly indistinguishable** anzi è proprio inutilizzabile.

TwoTP: OTP where the last half of the key is the same as the first one (this one is obtained as OTP)

```
# Adversary for TwoTP

class MalloryTwoTP(Mallory):

    def plaintexts(self):
        return ("0000","0010") # 0000 and 0011 works as well

    def guess(self,y):
        if y[0]==y[int(len(y)/2)]: # Compares the first bit of the first part of the key and the first bit of the last part
            bm = 0
        else:
            bm = 1
        return bm
```

Percentage of success: 100.0

Anche qui l'avversario vince sempre, rendendo questo cifrario assolutamente **non perfectly indistinguishable**.

La strategia dell'avversario è mettere a 1 il bit a metà nei plaintext: dal momento che la chiave è ripetuta una seconda volta, la seconda metà di chiave che va a cripare la seconda parte del ciphertext (dove appare 1 come terza posizione) renderà il corrispondente bit del ciphertext diverso dal primo bit.

Altrimenti, se sono uguali, il messaggio è il primo, quello con tutti 0.

Quasi-OTP: OTP ma non ammette la chiave dove tutti i bit sono 0

```
# Adversary for QuasiOTP

class MalloryQuasiOTP(Mallory):

    def plaintexts(self):
        return ("0000", "1111")

    def guess(self,y):
        allZero = True           # allZero=True iff all bits in ciphertext are 0
        for yi in y:
            if int(yi)==1:
                allZero = False

        if allZero:
            bm = 1
        else:
            bm = 0

        return bm
```

Percentage of success: 53.3244

A prima vista sembrerebbe essere perfectly indistinguishable, **TUTTAVIA**, aumentando sensibilmente il numero di esperimenti, il cifrario sembra superare la soglia del 50%.

Infatti, lo spazio delle chiavi è minore dello spazio dei plaintext (0000 non appare nelle chiavi, ma può apparire nel plaintext: l'avversario sta sfruttando questa possibilità per avere un qualche margine di probabilità di vittoria in più. Quindi il Quasi-OTP **non è perfectly indistinguishable**.

Complessità computazionale

Computabilità:

Studia se un problema è **computabile** (o calcolabile) in maniera **algoritmica**

Problemi decidibili vs problemi non decidibili (es: problema della fermata)

Complessità computazionale:

Studia se un problema è **trattabile** (ovvero, esiste un algoritmo **efficiente**) oppure no (non trattabile).

Metriche computazionali:

- Modello di esecuzione: **Macchina di Turing**
- Risorsa da misurare: **Tempo**
- Costo di utilizzo della risorsa: **Numero di passi della MdT**
- Dimensione dell'input: **Numero di simboli sulla MdT**
- Tipo di analisi: **Caso pessimo**

Funzione costo di una MdT (deterministica)

Sia M una MdT deterministica. La funzione costo $f: \mathbb{N} \rightarrow \mathbb{N}$ è una funzione tale che:

per ogni input x , $M(x)$ termina in $f(|x|)$ passi

$|x|$: numero di celle MdT per memorizzare l'input.

Diciamo che M ha **complessità in tempo** f .

Domanda: esistono altre funzioni costo della Macchina di Turing?

Risposta: Sì, la funzione f è un **upper bound**: vanno bene tutte quelle più grandi (es: complessità n^2 e dico n^3).

Quindi, $f(n)$ **sovra-approssima** il numero di step di M su input di lunghezza n .

Problemi di decisione:

Problemi dove la risposta è binaria: vero oppure falso (0 oppure 1).

Possiamo ricondurre ogni problema a un problema decisionale basato sul **linguaggio**: una parola appartiene o no al linguaggio?

La classe di complessità è un insieme (infinito) di problemi di decisione.

Big-O notation:

La notazione Big-O fornisce una stima superiore della crescita dell'algoritmo.

In altre parole, descrive il limite superiore del tempo (o dello spazio necessario) per eseguire l'algoritmo al crescere dell'input.

$$O(g) = f \in N \rightarrow N, \quad \exists c, n_0 : \quad \forall n > n_0 \quad f(n) \leq c * g(n)$$

Per input sufficientemente grandi ($n \geq n_0$), il tempo di esecuzione dell'algoritmo è limitato superiormente da una costante moltiplicata per la funzione $f(n)$.

Questo indica che l'algoritmo non richiede più tempo di quanto sia proporzionale a $f(n)$, tenendo conto dei fattori costanti: g è il **limite asintotico superiore** della funzione f .

Classe di complessità: TIME

Una classe di complessità è un insieme di problemi di decisione.

Sia $g \in N \rightarrow N$. La classe di complessità $TIME(g)$ è l'insieme dei problemi di decisione che sono risolti da una Macchina di Turing con complessità in tempo $f \in O(g)$.

La classe **non è robusta rispetto al modello di computazione** scelto (es. MdT mono-nastro e multi-nastro oppure MdT-deterministica e MdT **non**-deterministica).

Tesi di Church-Turing:

Se un problema è umanamente calcolabile, esiste una MdT che lo risolve. → Tutti i modelli di computazione (macchina RAM, funzioni ricorsivamente enumerabili, lambda-calcolo, MdT, etc.) sono equivalenti

Tesi di Church-Turing (estesa):

Tutti i modelli computazionalmente equivalenti a una MdT (tesi Church-Turing) sono anche **polynomialmente equivalenti** alla MdT.

Classe di complessità: PTIME:

PTIME è una classe **robusta**.

Modello di encoding:

Il modello unario non è efficiente. Il modello binario sì.

Tutti i modelli efficienti, sono polynomialmente equivalenti.

Avversari efficienti:

Algoritmi con tempo **probabilistico polinomiale**: $Adv \in PP\text{TIME}$

Parametro di sicurezza:

Lunghezza della chiave: 1^n .

Computational secrecy:

La perfect secrecy (e indistinguishability) è irrealistica: Assume che M abbia **potenza di calcolo illimitata**.

Richiede che l'avversario non ottenga **nessuna** informazione sul messaggio: $\Pr(\text{PrivK} = 1) = 1/2$

Encryption scheme:

Assumiamo che $P = C = K = \{0, 1\}^*$ operiamo su stringhe di bit

Schema di encryption a chiave privata: (Gen, E, D)

Gen -> algoritmo **probabilistico** di generazione delle chiavi: $k \leftarrow \text{Gen}(1^n)$ $|k| \geq n$

E -> algoritmo **probabilistico** di encryption: $y \rightarrow E_k(x)$

D -> algoritmo **deterministico** di decryption: $x = D_k(y)$

Condizione di correttezza:

$$\forall n \in \mathbb{N}, \quad \forall k \leftarrow \text{Gen}(1^n), \quad \forall x \in \{0, 1\}^*: \quad D_k(E_k(x)) = x$$

Funzione negligible:

Una funzione $f: \mathbb{N} \rightarrow \mathbb{R}^+$ è **negligible** se, per ogni polinomio P : $\exists n_0 : \forall n > n_0 \quad f(n) \leq \frac{1}{p(n)}$

$\frac{1}{n^k}$ **non** è negligible per nessun k

$f(n) = \frac{1}{2^n}$ invece è **negligible**

Computationally secret:

Uno schema di cifratura è **computazionalmente sicuro** per ogni avversario PPTIME se la probabilità di riuscita dell'attacco è **negligible**.

Indistinguishability experiment under EAV:

$$\text{PrivK}_{M,\pi}^{\text{eav}}$$

1. $M \rightarrow A : x_0, x_1 \quad |x_0| = |x_1| \quad$ attaccante M invia due plaintext ad A **di dimensione uguale**
2. $A : k \leftarrow \text{Gen}(1^n) \quad A$ genera una chiave con l'algoritmo $\text{Gen}() \in \pi$, M conosce la lunghezza 1^n
 $b \leftarrow \{0, 1\} \quad A$ genera un bit b **uniformemente random**
3. $A \rightarrow M : y \leftarrow E_k(x_b) \quad A$ invia ad M un ciphertext ottenuto da uno dei 2 plaintext (scelto in base a b)
4. $M : b_m \leftarrow \dots \quad M$ vuole calcolare b_m in modo che $b_m = b$ (vuole indovinare b)

L'esperimento ha successo $\text{PrivK}_{M,\pi}^{\text{eav}}(n) = 1$ se $b_m = b$

$$\Pr(\text{PrivK}_{M,\pi}^{\text{eav}}(n) = 1) = \Pr(b = 0)\Pr(b_m = 0|b = 0) + \Pr(b = 1)\Pr(b_m = 1|b = 1)$$

π è **computationally secret** rispetto a n se: $\forall M \in \text{PPTIME} \quad \Pr(\text{PrivK}_{M,\pi}^{\text{eav}} = 1) = 1/2 + f(n)$ dove $f(n)$ è **negligible**

Esempio (Quasi-OTP)

Come l'OTP, ma esclude la chiave 0^n durante la generazione della chiave in $\text{Gen}(1^n)$

Perfect secrecy (definizione):

$$x = 0^n \quad y = 0^n$$

$$\Pr(x|y) = 0 \neq \Pr(x) > 0 \rightarrow \text{non è perfetto}$$

Indistinguishability experiment:

$$M : x_0 = 0^n \quad x_1 = 1^n$$

$$M : b_m = (y == 0) ? 1 : 0$$

$$\Pr(b_m = 0 | b = 0) = 1 \quad \Pr(b_m = 1 | b = 1) = \frac{1}{2^n - 1}$$

[...] non è perfectly indistinguishable \rightarrow non è perfetto

Indistinguishability experiment under EAV (computational secrecy):

$$\Pr(\text{Priv}K_{M,\pi}^{\text{eav}} = 1) = 1/2 + \frac{1}{2^n - 1} \frac{1}{2}$$

$\frac{1}{2^n - 1} \frac{1}{2}$ è **negligible**

Osservazioni:

- Bisognerebbe dimostrare che non esistano attaccanti più efficienti di questo
- Le chiavi sono ancora lunghe come i plaintext
- Stiamo supponendo soltanto attaccante EAV (ciphertext only attack)

Teorema:

Se π è **perfectly secret** allora π è anche **computationally secret** (under EAV). Ma il viceversa non è necessariamente vero.

Esercizio:

Dimostra che il seguente cifrario **non** è EAV-sicuro:

$$|P| = |C| = |K| = \{0,1\}^n$$

```
Gen(1n) = let bp ← {0,1} in
                if bp = 0 then k ← {0,1}n
                else let k1 ← {0,1} in k := k1n
```

$$E_k(x_1 \dots x_n) = (x_1 \oplus k_1) \dots (x_n \oplus k_n)$$

Se b_p è 0 : k è completamente random

Se b_p è 1 : k è 0 oppure 1 ripetuto n volte in base a k_1 (generato random, ma poi ripetuto n volte)

$$M : x_0 = 0^n \quad x_1 = 0^{n-1}1$$

$$M : b_m = (y_1 == y_2 == \dots == y_n) ? 0 : 1 \quad \text{nelle slides è il contrario, perché?}$$

$$Pr(PrivK(n) = 1) = 1/2(Pr(b_m = 0|b = 0) + Pr(b_m = 1|b = 1))$$

$$Pr(b_m = 0|b = 0) = Pr(\text{bit siano uguali } | x = 0^n) = Pr(k = 0^n \text{ or } k = 1^n) = Pr(b_p = 1) = 1/2$$

$$Pr(b_m = 1|b = 1) = Pr(\text{bit non siano uguali } | x = 0^{n-1}1) = Pr(k \neq 1^{n-1}0 \text{ and } k \neq 0^{n-1}1) = 1 - \frac{1}{2^{n-2}}$$

$$Pr(PrivK(n) = 1) = (\frac{1}{2} * \frac{1}{2}) + (\frac{1}{2}(1 - \frac{1}{2^{n-2}})) = (\frac{1}{4}) + (\frac{1}{2}(1 - \frac{1}{2^{n-2}})) > 1/2 + negl(n) \rightarrow \text{non è eav-sicuro}$$

Esercizio (two-time pad):

$$Gen(1^n) = \{h \leftarrow \{0,1\}^m; k := hh\} \quad \text{con } (n = 2m) \quad \text{metà chiave random ripetuta}$$

$$E_k(x) = x \oplus k$$

$$M : x_0 = 0^m 0^m \quad x_1 = 0^m 1^m$$

$$M : b_m = (y[1\dots m] == y[m+1\dots m]) ? 0 : 1$$

$$Pr(PrivK(n) = 1) = 1/2(Pr(b_m = 0|b = 0) + Pr(b_m = 1|b = 1))$$

$$Pr(b_m = 0|b = 0) = Pr(\text{due metà uguali } | x = 0^m 0^m) = 1$$

$$Pr(b_m = 1|b = 1) = Pr(\text{due metà diverse } | x = 0^m 1^m) = Pr(k \neq 0^m 1^m \text{ and } k \neq 1^m 0^m) = 1 - \frac{1}{2^{n-2}}$$

Non è eav-sicuro.

Sicurezza (EAV) rispetto a encryption multiple:

1. $A : b \leftarrow \{0, 1\}$
2. $A : k \leftarrow Gen(1^n)$
3. $M \rightarrow A : x_0 = (x_0^0, x_0^1, \dots, x_0^t) \quad x_1 = (x_1^0, x_1^1, \dots, x_1^t) \text{ con } |x_0^i| = |x_1^i| \forall i \leq t$
4. $A \rightarrow M : y = E_k(x_b) = E_k(x_b^0), \dots, E_k(x_b^t) \text{ nota: } k \text{ è sempre la stessa}$
5. $M : b_m = \dots$

$$PrivK_{M,\pi}^{mult}(n) = 1 \text{ se } b_m = b$$

Teorema:

π è multi-eav sicuro $\rightarrow \pi$ è eav-sicuro

Sicurezza OTP rispetto a multiple encryptions:

$$Gen(1^n) = \{\{0, 1\}^n\}$$

$$E_k(x) = x \oplus k$$

$$D_k(y) = y \oplus k$$

$$M : x_0 = (0^n, 1^n) \quad x_1 = (0^n, 0^n)$$

$$M : b_m = (y^0 == y^1) ? 1 : 0$$

$$Pr(b_m = 0 | b = 0) = 1$$

$$Pr(b_m = 1 | b = 1) = 1$$

$$Pr(PrivK_{M,\pi}^{mult}(n) = 1) = 1 \rightarrow \text{vince sempre}$$

Pur essendo perfectly secret (e quindi EAV-sicuro), l'OTP **non è sicuro** alle multiple encryptions:

Questo perchè l'encryption E_k è **deterministico** e non probabilistico.

CPA-security:

Ricordiamo i poteri dell'attaccante:

- Ciphertext-only attack:
 M conosce soltanto il ciphertext
- KPA: Known Plaintext Attack:
 M conosce delle coppie x e y che sono state cifrate con chiave k (la chiave ovviamente non la conosce)
- **CPA: Chosen Plaintext Attack:** \leftarrow
 M crea delle coppie (x, y) . Ha accesso alla funzione di encryption $E_k(x)$ (*oracle access*)
- CCA: Chosen Ciphertext Attack:
 M ora ha accesso anche alla funzione di decryption $D_k(y)$ (*oracle access*)

CPA : M adesso ha **oracle access** alla funzione di encryption E_k (k è fissato, ma M non lo conosce!)

1. $A : b \leftarrow \{0, 1\}$
2. $A : k \leftarrow Gen(1^n)$
3. $M \rightarrow A : x_0, x_1 \in \{0, 1\}^* \text{ s.t. } |x_0| = |x_1| \quad M \text{ ha oracle access a } E_k$
4. $A \rightarrow M : y = E_k(x_b) \quad M \text{ ha oracle access a } E_k$
5. $M : b_m = \dots$

$$PrivK_{M,\pi}^{cpa}(n) = 1 \text{ if } b_m = b, \text{ else } 0$$

π è **computationally secret rispetto a CPA** se:

$$\forall M \in PPTIME \quad Pr(PrivK_{M,\pi}^{cpa} = 1) = 1/2 + negl(n)$$

Teorema:

π è CPA-sicuro \rightarrow π multi-CPA sicuro \rightarrow π è eav-sicuro

Se E_k è deterministica:

1. $A : b \leftarrow \{0, 1\}$
2. $A : k \leftarrow Gen(1^n)$
3. $M \rightarrow A : x_0, x_1 \in \{0, 1\}^* \text{ s.t. } |x_0| = |x_1|$
4. $A \rightarrow M : y = E_k(x_b)$

M usa oracle access a E_k :

$M \rightarrow A : x_0$

$A \rightarrow M : y' = E_k(x_0)$

5. $M : b_m = \{0 \text{ if } y = y' \text{ else } 1\}$

M sfrutta il **determinismo** di E_k e **indovina sempre** grazie all'**oracle**

$$Pr(PrivK_{M,\pi}^{cpa}(n) = 1) = 1$$

Teorema:

Se E_k è deterministica $\rightarrow \pi$ **non è** CPA-sicuro

Pseudorandom generators:

L'idea è quella di ottenere uno schema di encryption **EAV-sicuro**, basato su OTP, ma con chiavi più corte del plaintext.

Per avere una chiave corta, è necessario espanderla:

$$k \leftarrow \{0, 1\}^m \quad m \ll n \quad G(k) = r \in \{0, 1\}^n$$

$$E_k(x) = r \oplus k = G(K) \oplus x$$

G è una funzione **deterministica** che espande k : k è il **seed** di un generatore pseudorandom

Vorrei che r fosse indistinguibile da una sequenza di n bit random (per un attaccante polinomiale)

Complessità di Kolmogorov:

Una sequenza di bit è **random** se non esiste un programma che la genera che sia sostanzialmente più corto della sequenza stessa:
in altre parole, se ho una sequenza e un programma che al genera, se il programam è più corto della sequenza, la sequenza non è random.

Kolmagorov considera anche i programmi **inefficienti**, noi restringiamo il campo soltanto a quelli **efficienti**.

Approccio crittografico:

Non consideriamo sequenza di bit random, ma una **distribuzione** random di sequenze.

Non consideriamo, inoltre, i programmi che generano le sequenze: definiamo come random una distribuzione quando supera tutti i possibili test statistici.

Distinguishers:

Un **distinguisher** è un **algoritmo PPTIME** che simula i test statistici: prende in input una sequenza di bit e restituisce in output 0 se la sequenza **non** sembra random oppure 1 se la sequenza in input sembra random.

Un generatore pseudorandom G è buono se il miglior distinguisher può soltanto **tirare a indovinare**.

Generatore pseudorandom:

G algoritmo **deterministico PTIME** da $\{0, 1\}^n$ a $\{0, 1\}^{l(n)}$

G è un **generatore pseudorandom** (PRG) se:

1. $\forall n \quad l(n) > n \quad$ espansività
2. $Pr(PRG_{M,G}(n) = 1) \leq 1/2 + negl(n) \quad \forall M \in PPTIME$

Esperimento $PRG_{M,G}(n)$

1. $A : b \leftarrow \{0, 1\}$

- *if* $b == 0$: $r \leftarrow \{0, 1\}^{l(n)}$ completamente random
- *if* $b == 1$: $s \leftarrow \{0, 1\}^n \quad r := G(s) \quad$ soltanto il **seed** è random, r è **deterministico**

2. $A \rightarrow M : r$

3. $M : b_m \quad M$ deve indovinare b_m

$$PRG_{M,G}(n) = 1 \text{ se } b_m = b$$

M deve distinguere **randomness** ($b = 0$) da **pseudorandomness** ($b = 1$)

Attenzione: bisogna controllare che M (ovvero l'attaccante o, in questo caso il **distinguisher**, sia **polinomiale** (efficiente) e anche la condizione di **espansività**

Esercizio #1:

$$G(s) = sb' \quad b' = \oplus_{i=1..n}^n s[i]$$

$$M : \quad 0 \text{ if } \oplus_{i=1..n}^n r[i] \neq r[n+1] \text{ else : } 1$$

G **non** è pseudorandom

Praticamente, M sta facendo gli stessi passi di G visto che può vedere il **seed** (restituito in chiaro) con lo XOR di tutti i bit del seed messo come ultimo.

Esercizio #2:

$$G(s) = ss$$

$$l(n) > n \quad \text{soddisfa espansività}$$

strategia del distinguisher: 1 if $w[1..n] == w[n+1..2n]$ 0 otherwise

$$\Pr(\text{PRG}_{D,G}(n) = 1) = \frac{1}{2}(\Pr(b_D = 0|b = 0)\Pr(b_D = 1|b = 1))$$

$$\Pr(b_D = 0|b = 0) = 1 - \frac{1}{2^{n-2}} \quad \text{i casi sfortunati dove } r = 0^n \text{ o } r = 1^n$$

$$\Pr(b_D = 1|b = 1) = 1$$

$$\Pr(\text{PRG}_{D,G}(n) = 1) = \frac{1}{2}(1 - \frac{1}{2^{n-2}} + 1) > \frac{1}{2} + \text{negl}(n)$$

Esercizio #3:

$$G(s) = s||f(s)$$

$$|f(s)| > 0 \quad f \in \text{PTIME}$$

D (o M) conosce la lunghezza di s : n e conosce anche la funzione f (principio di Kerckhoffs): ovviamente non può però conoscere l'esito delle generazioni random private di Alice.

$$1. \quad A : \quad b \leftarrow \{0, 1\}$$

- if $b == 0$: $r \leftarrow \{0, 1\}^{l(n)}$
- if $b == 1$: $s \leftarrow \{0, 1\}^n \quad r := G(s) = s||f(s)$

$$2. \quad A \rightarrow M : \quad r = w = w_1..w_n w_{n+1}..w_m \quad \text{dove } |w| = m = l(n)$$

$$w_1..w_n = s$$

$$w_{n+1}..w_m = f(s) \setminus$$

3. M : $b_m = \{(w_{n+1}..w_m == f(w_1..w_n)) ? 1 : 0\}$ M calcola $f(s)$ perché, conoscendo $n = |s|$, è in grado di dividere la stringa come vuole

$\Pr(b_m = 0|b = 0) = \Pr((f(w_1..w_n) \neq w_{n+1}..w_m) | \text{randomness}) = 1 - \frac{1}{2^{m-n}}$ indovina quasi sempre, tranne nel caso sfortunato dove, per caso, la seconda parte è il risultato della funzione f applicata alla prima parte

$\Pr(b_m = 1|b = 1) = 1$ indovina sempre

G **non** è pseudorandom

Pseudo-OTP ($_pOTP$):

Sia $G \in \{0,1\}^n \rightarrow \{0,1\}^{l(n)}$

$$_pOTP = (Gen, E_k, D_k)$$

$$Gen(1^n) = \{k \leftarrow \{0,1\}^n\}$$

$$E_k(x) = G(k) \oplus x$$

$$D_k(y) = G(k) \oplus y$$

$$x, y \in \{0,1\}^{l(n)}$$

Teorema:

G è PRG $\iff {}_pOTP$ è EAV-sicuro

Dimostrazione:

Controposizione logica:

$$A \implies B \quad \neg B \implies \neg A$$

Hanno la stessa tavola di verità.

$$(\Leftarrow): {}_pOTP \text{ è EAV-sicuro} \Rightarrow G \text{ è PRG}$$

Usiamo la **controposizione logica**: G non è PRG $\implies {}_pOTP$ non è EAV-sicuro

Cosa vuol dire G non è PRG?

$$\exists D \in PPTIME : Pr(PRG_{D,G}(n) = 1) > 1/2 + negl(n)$$

$$PrivK_{M, {}_pOTP}^{eav}(n) :$$

1. $M \rightarrow A : x_0 \leftarrow \{0,1\}^{l(n)} \quad x_1 = 0^{l(n)} \quad x_0$ è completamente **random**, x_1 no (stesso pattern di G)
2. $A \rightarrow M : y = E_k(x_b) = x_b \oplus G(k)$ con $b \leftarrow \{0,1\}$
3. $M : b_m = D(y) \quad M$ usa il distinguisher sul ciphertext y

M sta usando il distinguisher D che è in grado di "rompere" G , e $D \in PPTIME$

$$\text{se } b = 0 \quad y = r \oplus G(k) \quad r \text{ è random (è } x_0\text{)}$$

$$\text{se } b = 1 \quad y = 0^n \oplus G(k) = G(k)$$

$$Pr(PrivK_{M, {}_pOTP}^{eav}(n) = 1) = \frac{1}{2}(Pr(b_m = 0|b = 0) + Pr(b_m = 1|b = 1))$$

$$\begin{aligned} Pr(b_m = 0|b = 0) &= Pr(D(y) = 0) = Pr(D(r \oplus G(k)) = 0) = Pr(D(r) = 0) \quad \text{perchè: } (random \oplus non\ random) = random \\ Pr(b_m = 1|b = 1) &= Pr(D(y) = 1) = Pr(D(G(k)) = 1) \end{aligned}$$

$$Pr(PrivK_{M, {}_pOTP}^{eav}(n) = 1) = \frac{1}{2}(Pr(D(r) = 0) + Pr(D(G(k)) = 1)) = \frac{1}{2}(Pr(b_d = 0|b = 0) + Pr(b_d = 1|b = 1)) =$$

$$= Pr(PRG_{D,g}(n) = 1)$$

$$Pr(PrivK_{M, {}_pOTP}^{eav}(n) = 1) = Pr(PRG_{D,g}(n) = 1) > 1/2 + negl(n) \quad \text{perchè } G \text{ non è PRG} \rightarrow {}_pOTP \text{ non è EAV-sicuro}$$

(\Rightarrow): G è PRG $\Rightarrow {}_pOTP$ è EAV-sicuro

Usiamo la **controposizione logica**: ${}_pOTP$ non è EAV-sicuro $\implies G$ non è PRG

Cosa vuol dire ${}_pOTP$ non è EAV-sicuro?

$$\exists M \in PPTIME : \Pr(PrivK_{M, {}_pOTP}^{eav}(n) = 1) > 1/2 + negl(n)$$

Il distinguisher D **sfrutta** l'attaccante efficiente M dell'esperimento $PrivK_{M, {}_pOTP}^{eav}(n)$ ovvero ci gioca contro (impersona A e quindi scriviamo $D(A)$, ma non possiamo assolutamente controllare M)

$$PrivK_{M, {}_pOTP}^{eav}(n):$$

$M \rightarrow D(A) : x_0, x_1$ non possiamo sceglierli noi perché non siamo M , ma $D(A)$

$D(A) \rightarrow M : y = w_{b_{PRG}} \oplus x_{b_{PrivK}}$ il nostro input (da distinguisher) è $w_{b_{PRG}}$ e lo usiamo nell'encryption del messaggio di M : $x_{b_{PrivK}}$

$M : b_m$ M restituisce il risultato della sua strategia contro lo pseudo-OTP

se $b_{PRG} = 0$: $w_{b_{PRG}} = r \implies OTP \implies \Pr(b_m = b_{PrivK}) = 1/2$

se $b_{PRG} = 1$: $w_{b_{PRG}} = G(k) \implies {}_pOTP \implies \Pr(b_m = b_{PrivK}) > 1/2 + negl(n)$ per ipotesi

Nel caso in cui b_{PRG} sia 0, M sta giocando contro l' OTP e quindi ha 1/2 di probabilità di vincere.

Nel caso in cui b_{PRG} sia 1, M sta giocando contro lo pseudo-OTP che per ipotesi **non** è EAV-sicuro: probabilità di vincere è maggiore di $1/2 + negl(n)$

È più probabile che M indovini quando b_{PRG} è uguale a 1 (pseudo-OTP)

Cosa succede adesso nell'esperimento PRG ?

$$PRG_{D,G}(n) :$$

1. $A : w_0 = r \leftarrow \{0,1\}^{l(n)}$ $w_1 = G(s)$ $s \leftarrow \{0,1\}^n$

2. $A \rightarrow D : w_{b_{PRG}} b_{PRG} \leftarrow \{0,1\}$

3. $D : se b_m = b_{PrivK} \implies b_D = 1$

Nel passo 3 del PRG -experiment, D sfrutta l'attaccante M (che rende lo pseudo-OTP non sicuro) dell'esperimento $PrivK$.

Allora D si chiede: M ha vinto ($b_m = b_{PrivK}$)? E allora dice 1 (ovvero pseudorandomness).

$$\Pr(PRG_{D,G}(n) = 1) = \frac{1}{2}(\Pr(b_D = 0 | b_{PRG} = 0) + \Pr(b_D = 1 | b_{PRG} = 1))$$

$\Pr(b_D = 0 | b_{PRG} = 0) = \Pr(b_m \neq b_{PrivK} | OTP) = 1/2$ D dice 0 quando $b_m \neq b_{PrivK}$ nell'esperimento $PrivK$ e, essendo OTP , la probabilità è 1/2

$$\Pr(b_D = 1 | b_{PRG} = 1) = \Pr(b_m = b_{PrivK} | {}_pOTP) > 1/2 + negl(n) \text{ per ipotesi}$$

$$\Pr(PRG_{D,G}(n) = 1) > \frac{1}{2}(\frac{1}{2} + \frac{1}{2} + negl(n)) = \frac{1}{2} + negl(n) \implies G$$
 non è PRG

Nota: $M \in PPTIME \implies D \in PPTIME$

Exercise

- $\text{Gen}(1^n) = \{ k \leftarrow \{0,1\}^n \}$
- $E_k(x) = (r, G(r) \oplus x) \quad x \in \{0,1\}^{n+1}, r \leftarrow \{0,1\}^n,$
 $G \in \{0,1\}^n \rightarrow \{0,1\}^{n+1}$ is a PRG
- $D_k(r,z) = G(r) \oplus z$

Is the scheme EAV-secure? (hint: the key is never used)

$PrivK_{M,\pi}^{eav}(n)$:

1. $A : b \leftarrow \{0,1\} \quad k \leftarrow \text{Gen}(1^n)$
2. $M \rightarrow A : x_0 = 0^{n+1} \quad x_1 = 1^{n+1}$
3. $A \rightarrow M : y = E_k(x_b) \quad y = y_1 \dots y_n, y_{n+1_1} \dots y_{n+1_{n+1}}$
4. $M : b_m = \{r = y[1..y_n] \mid (y_{n+1_1} \dots y_{n+1_{n+1}} == G(r) \oplus x_0) \ ? \ 0 \ : \ 1\}$

r è visibile in chiaro in y , quindi M può calcolare $G(r)$ (principio di Kerckhoffs: G è noto a M , gli stiamo regalando il **seed** r)
Se xorando $G(r)$ con x_0 ottiene la seconda parte del ciphertext, allora dirà 0

M **vince sempre** → non è EAV-sicuro

PRF (Pseudorandom functions)

Abbiamo visto il teorema:

G è PRG $\iff {}_p\text{OTP}$ è EAV-sicuro

Noi vogliamo però almeno la CPA-security

$PrivK_{M,\pi}^{CPA}(n)$:

1. $A : b \leftarrow \{0,1\} \quad k \leftarrow \text{Gen}(1^n)$
2. $M \rightarrow A : x_0 \quad x_1 \quad \text{con } |x_0| = |x_1|$
3. $A \rightarrow M : y = E_k(x_b) \quad M \text{ ha oracle access a } E_k$
4. $M : b_m$

$PrivK_{M,\pi}^{CPA}(n) = 1 \text{ if } b_m = b$

Funzione pseudorandom:

Abbiamo visto in precedenza i generatori pseudorandom (G), algoritmi **deterministici** che **espandono** un seed k random e nessun distinguisher/attaccante **efficiente** è in grado di romperli (ovvero discriminare l'output del generatore da una sequenza effettivamente random) trascurando le probabilità negligibili.

$$F \in \{0,1\}^{l_{key}(n)} \rightarrow (\{0,1\}^{l_{in}(n)} \rightarrow \{0,1\}^{l_{out}(n)})$$

a meno di specificare il contrario assumiamo che: $l_{key}(n) = l_{in}(n) = l_{out}(n) = n$

$$F \in \{0,1\}^n \rightarrow (\{0,1\}^n \rightarrow \{0,1\}^n) \quad F \in PTIME \quad \textbf{ATTENZIONE: } F \text{ è una } \mathbf{funzione} \text{ e quindi è } \mathbf{deterministica}$$

A differenza dei PRG, le funzioni pseudorandom non hanno il vincolo di espansività.

La differenza sostanziale è:

PRG genera **sequenze** pseudorandom

PRF genera **funzioni** pseudorandom

Esperimento PRF:

$$PRF_{D,F}(n):$$

1. $A : f_0 \leftarrow (\{0,1\}^n \rightarrow \{0,1\}^n) \quad f_1 = F_k \quad \text{dove } k \leftarrow \{0,1\}^n \quad b \leftarrow \{0,1\}$
2. D ha **oracle access** a f_b questo per ovviare al problema della complessità esponenziale di f_0
3. $D : b_d$

$$PRF_{D,F}(n) = 1 \text{ se } b_d = b$$

Nel passo 1, in base al risultato di b , A sceglie una **funzione**:

se $b = 0$: sceglie *letteralmente* una funzione a caso tra tutte le funzioni da n bit a n bit

se $b = 1$: utilizza la funzione F applicata a k

F è una **pseudorandom function** se $\forall D \in PPTIME : Pr(PRF_{D,F}(n) = 1) \leq 1/2 + negl(n)$

Esercizio:

$F_k = x \& k$ $\&$ = bitwise AND

F_k è una PRF?

$PRF_{D,F}(n)$:

1. $A : f_0 \leftarrow (\{0,1\}^n \rightarrow \{0,1\}^n) \quad f_1 = F_k \quad$ dove $k \leftarrow \{0,1\}^n \quad b \leftarrow \{0,1\}$

2. D usa f_b in modalità **oracle**

$D : f_k(0^n)$

3. $D : b_d = (y == 0^n) ? 1 : 0$

D è chiamato a scegliere una x quando usa la funzione in modalità oracle; dal momento che D sa come funziona F : manda come input una sequenza di 0 (0 & qualcosa = 0): se il risultato restituito da f_b è 0^n , la sua strategia è quella di dire 1 (F_k), altrimenti dice 0

$$Pr(PRF_{D,F}(n) = 1) = \frac{1}{2} (Pr(b_d = 0|b = 0) \cdot Pr(b_d = 1|b = 1))$$

$$Pr(b_d = 1|b = 1) = 1 \quad vince \ sempre$$

$$Pr(b_d = 0|b = 0) = 1 - Pr(b_d = 1|b = 0)$$

$Pr(b_d = 1|b = 0)$ = probabilità che la funzione random restituiscia 0^n (quindi la probabilità che D dica 1 in maniera errata in quanto la funzione è effettivamente random e lui pensava fosse l'and di k e 0^n)

Quante sono le funzioni in $(\{0,1\}^n \rightarrow \{0,1\}^n)$?

Rappresentiamo le funzioni (**non è** richiesto che sia iniettiva) come sequenza (in un array) dei loro output:

$input \rightarrow output$

$n = 2$:

$$00 \rightarrow x_1x_2$$

$$01 \rightarrow x_3x_4$$

$$10 \rightarrow x_5x_6$$

$$11 \rightarrow x_7x_8$$

bitstring: $x_1x_2 \ x_3x_4 \ x_5x_6 \ x_7x_8 \rightarrow 2^8$ possibili bitstrings (8 bit necessari per la rappresentazione)

00000000 00000001 00000010 00000011 ... 11111110 11111111

$n = 3$:

$$24 \text{ bit} \rightarrow 2^3 \cdot 3 = 24$$

$n = 4$:

$$64 \text{ bit} \rightarrow 2^4 \cdot 4 = 64$$

n generico:

2^n blocchi da n bit

$2^n \cdot n$ bit

$$|Fun_n| = 2^{2^n \cdot n} = (2^n)^{2^n}$$

Insieme di tutte le funzioni n bit lo rappresentiamo come Fun_n :

$$|Fun_2| = 2^8$$

$$|Fun_3| = 2^{24}$$

Esercizio (continuo)

Nell'esercizio sopra dovevamo calcolare:

$$Pr(b_d = 1 | b = 0) = \text{probabilità che la funzione random restituisca } 0^n$$

Rappresentiamo un nuovo insieme X_2 come l'insieme di tutte le $f \in Fun_2$: $f(00) = 00$ (l'insieme di tutte le funzioni $\{0, 1\}^2 \rightarrow \{0, 1\}^2$ che prendono 00 in input e restituiscono 00 in output, generalizzando per n , otteniamo lo stesso setting dell'esercizio).

Quante sono queste funzioni?

Secondo la rappresentazione in array:

$f(00)$	$f(01)$	$f(10)$	$f(11)$	
00	00	00	00	
00	00	00	01	
00	00	00	10	
00	00	00	11	
...				
00	11	11	11	← fino a qui
da	00	00	00	$\rightarrow 2^6$

$$|X_2| = 2^{2 \cdot 2^2 - 2} = 2^6$$

In generale:

$$|X_n| = 2^{n \cdot 2^n - n}$$

$$\frac{|X_n|}{|Fun_n|} = \frac{2^{n \cdot 2^n - n}}{2^{n \cdot 2^n}} = 2^{n \cdot 2^n - n - n \cdot 2^n} = 2^{-n} = \frac{1}{2^n} = Pr(b_D = 1 | b = 0)$$

$$Pr(b_D = 0 | b = 0) = 1 - Pr(b_D = 1 | b = 0) = 1 - \frac{1}{2^n}$$

$$Pr(PRF_{D,F}(n) = 1) = \frac{1}{2} \left(1 - \frac{1}{2^n} + 1\right) = 1 - \frac{1}{2^{n+1}} > 1/2 + negl(n) \rightarrow F \text{ non è PRF.}$$

Esercizio:

$F \in \{0,1\}^n \rightarrow (\{0,1\}^n \rightarrow \{0,1\}^n)$ è PRF

$$Gen(1^n) = \{k \leftarrow \{0,1\}^n\}$$

$$E_k(x) = F_k(0) \oplus x$$

$$D_k(x) = F_k(0) \oplus y$$

$$|x| = n$$

Condizione correttezza: $D_k(E_k(x)) = D_k(F_k(0) \oplus x) = F_k(0) \oplus F_k(0) \oplus x = x$

È EAV-sicuro?

L'encryption è **deterministica** in quanto viene applicata F_k su un argomento fisso (sempre 0) e poi xorata con x . F però è PRF e M non conosce k : lo schema è EAV-sicuro

È CPA-sicuro?

L'encryption è **deterministica** quindi lo schema sicuramente non è CPA-sicuro.

PRF-OTP:

$$Gen(1^n) = \{k \leftarrow \{0,1\}^n\}$$

$$E_k(x) = (r, F_k(r) \oplus x)$$

$$D_k(r, z) = (F_k(r) \oplus z)$$

Condizione correttezza: $D_k(E_k(x)) = x$

$$D_k(r, F_k(r) \oplus x) = F_k(r) \oplus (F_k(r) \oplus x) = 0 \oplus x = x$$

Perchè z della decryption è $F_k(r) \oplus x$, se sostituisco in $F_k(r) \oplus z$ ottengo $F_k(r) \oplus F_k(r) \oplus x$ e mi rimane x

Così facendo, **non** è richiesto che F_k sia invertibile, perchè F_k viene ricalcolata nella decryption con lo stesso parametro r . La sicurezza risiede ancora in k !

Teorema:

F è PRF \iff PRF-OTP è CPA-sicuro

Esercizio:

Vediamo cosa succede nell'esperimento di CPA-security con $F_k(x) = n \& k$

$PrivK_{M, PRFOTP}^{cpa}(n)$:

- 1 : A : $k \leftarrow Gen(1^n)$ $b \leftarrow \{0, 1\}$
- 2 : $M \rightarrow A$: $x_0 = 0^n$ $x_1 = 1^n$
- 3 : $A \rightarrow M$: $y = E_k(x_b) = (r, F_k(r) \oplus x)$
- 4* : M ha oracle access ad A

Esempio:

$$b = 0 : F_k(r) \oplus 0^n = F_k(r) = k \& r$$

$$b = 1 : F_k(r) \oplus 1^n = \overline{F_k(r)} = \overline{k \& r}$$

Per natura, gli 0 sono più probabili degli 1 nell'AND (nella negazione, ovviamente, è il contrario), quindi la sua strategia è dire $b = 0$ (ovvero $x_b = 0^n$, se il numero degli zeri nel cipher text è maggiore del numero degli 1):

$$b_m = (|y|_0 > |y|_1) ? 0 : 1$$

CCA-security:

M può usare A come encryption e **decryption** oracle

$PrivK_{M, \pi}^{CCA}(n)$:

- 1 : A : $k \leftarrow Gen(1^n)$ $b \leftarrow \{0, 1\}$
- 2 : $M \rightarrow A$: $x_0 \quad x_1 \quad con \quad |x_0| = |x_1|$
- 3 : $A \rightarrow M$: $y = E_k(x_b)$
- 4* : M ha oracle access ad A in encryption e decryption ma **non** può decifrare y
- 5 : b_m

$$PrivK_{M, \pi}^{CCA}(n) = 1 \quad se \quad b_m = b$$

π è CCA-sicuro se $\forall M \in PPTIME : Pr(PrivK_{M, \pi}^{CCA}(n) = 1) \leq 1/2 + negl(n)$

PRF-OTP non è CCA-sicuro:

$A : k \leftarrow Gen(1^n) \quad b \leftarrow \{0, 1\}$
 $M \rightarrow A : x_0 = 0^n \quad x_1 = 1^n$
 $A \rightarrow M : y \leftarrow E_k(x_b) = (r, F_k(r) \oplus x_b)$

M usa A come decryption oracle:

$y' = y[1..n-1] \overline{y[n]}$ M definisce una y' dove tutti i bit, tranne l'ultimo che è flippati, sono uguali a quelli di y (infatti non potrebbe mandare direttamente y)

$M \rightarrow A : y'$
 $A \rightarrow M : D_k(y') = D_k(r, F_k(r) \oplus x') = x' = x[1..n-1] \overline{x[n]}$

Dal momento che x può essere 0^n oppure 1^n , la strategia di M è guardare il primo bit della stringa e rispondere di conseguenza:
 $b_m = x[1]$

M vince sempre $\rightarrow Pr(PrivK_{M, PRFOTP}^{CCA} = 1) = 1 \rightarrow$ PRF-OTP **non** è CCA-sicuro

Questo perchè **PRF-OTP è MALLEABILE**: abbiamo cambiato un bit nel ciphertext e questo ha cambiato un bit anche nella sua decryption

Requisito di non malleabilità per la CCA-security:

La CCA-security richiede il requisito di **non malleabilità**:

Se M modifica $y = E_k(x)$ in y' , allora $x' = D_k(y')$ non deve avere nessuna somiglianza con x

PRF-OTP e multiple encryptions:

X insieme di blocchi x_i di plaintexts: $X = x_1 \ x_2 \ \dots \ x_t$ con $|x_i| = n$ posso estendere l'encryption a tutto l'insieme X

$E_k(X) = E_k(x_1) \ E_k(x_2) \ \dots \ E_k(x_t)$

Questo schema è **ancora CPA-sicuro**.

Attenzione. questo schema **non** è in ECB-mode (anche se ci assomiglia molto)!

Nonostante stia usando la stessa chiave k in tutte le encryption, in ogni singola encryption c'è comunque della randomness!

Come vedremo nel capitolo successivo (modes_of_operation.ipynb), l'ECB-mode utilizza F_k , l'estensione del PRF-OTP per lunghezza arbitraria, invece utilizza l'encryption E_k (basata comunque su una PRF) che introduce della randomness.

Siamo riusciti a **estendere** l'ecryption CPA-sicura (quindi soddisfacente) a lunghezze arbitrarie e k fissata.

Questo schema però è **inefficiente** in spazio:

$|X| = n \cdot t$ bit n bit di un blocco per t blocchi

$|E_k(X)| = 2 \cdot n \cdot t$ bit l'encryption di un singolo blocco richiede $2 \cdot n$ bit (composto da due sequenze di n bit), tutto questo per t blocchi

Utilizzando le **PRF** è possibile ottenere schemi CPA-sicuri tali che:

$|E_k(x)| = |X| + n$ n è costante

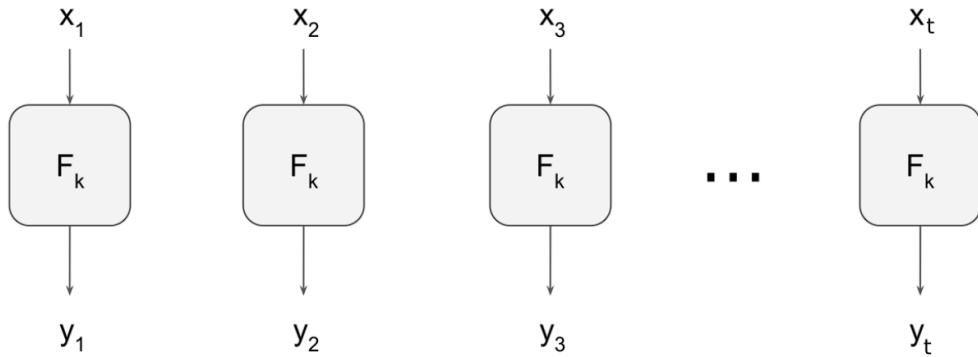
Noi vorremmo: $|E_k(X)| \leq n \cdot (t + 1)$ aumentare al massimo di un blocco rispetto al blocco di partenza

Modes of operation:

Modalità di funzionamento dei cifrari a blocchi per estendere uno schema di encryption **fixed length** a messaggi di lunghezza arbitraria.

ECB-mode:

Electronic CodeBook



$X = x_1 \dots x_t$ plaintext, con $|x_i| = n \quad \forall i \in 1..t$

$Y = y_1 \dots y_t$ ciphertext: $Y = E_k(X)$

È assolutamente **insicuro** e praticamente inutilizzabile per fini pratici.

Non introduce mai della **randomness**: la funzione pseudo-random è un algoritmo **deterministico** (si chiama proprio funzione).

M inoltre conosce 1^n (lunghezza della chiave) per il principio di Kerchoffs.

Caratteristiche:

- Richiede che F_k sia invertibile per eseguire la decryption: $x_1 = F_k^{-1}(y_1) \dots x_t = F_k^{-1}(y_t)$
- L'encryption è totalmente **parallelizzabile**
- La decryption è **random access**: posso decifrare i blocchi in qualunque ordine e poi rimetterli insieme

Non è neanche EAV-sicuro:

$$M \rightarrow A : \quad X_0 = x_0 x_0 \quad X_1 = x_0 x_1 \quad (x_0 \neq x_1)$$

$A \rightarrow M : \quad y = E_k(X_b) = E_k(X[1]) E_k(X[2]) \quad b \leftarrow \{0, 1\} \quad$ l'encryption è la concatenazione delle due encryption (funzione deterministica) con la stessa chiave

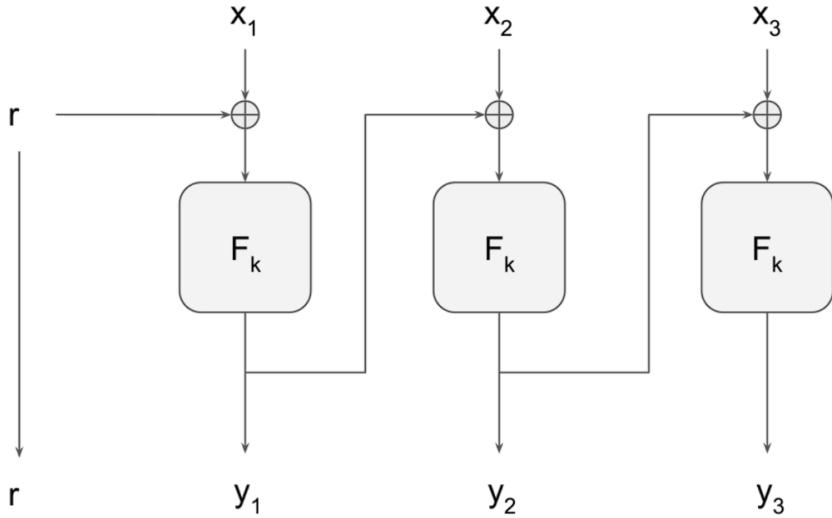
$M : \quad b_m = (Y[1] == Y[2]) ? 0 : 1 \quad M$ dice 0 se i due blocchi sono uguali (visto che due cose uguali verranno cifrate allo stesso modo)

$$\Pr(\text{PrivK}_{M,\pi}^{\text{eav}} = 1) = 1$$

CBC-mode:

Cipher Block Chaining

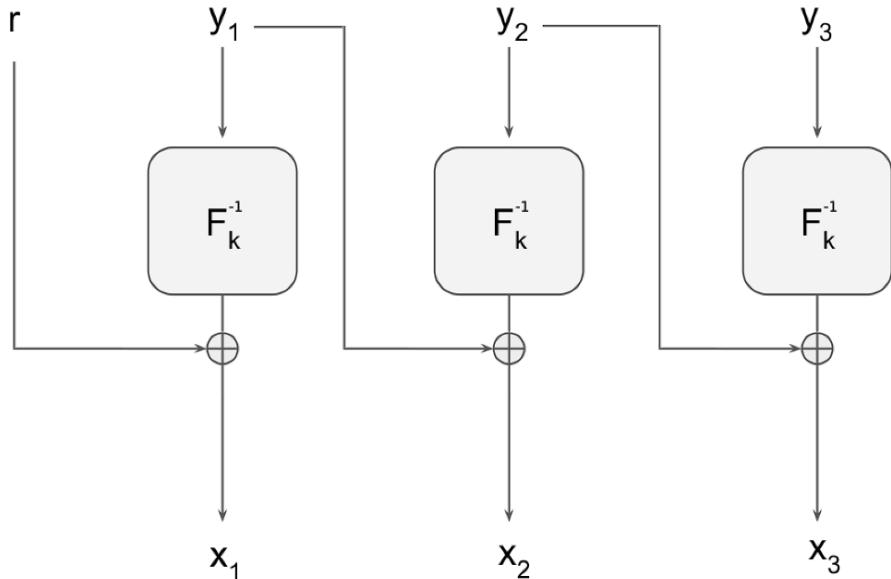
Encryption:



(estendibile fino a t)

Introduco un **initialization vector r random** (per avere il **non determinismo** e quindi la **CPA-security**):
 $r \leftarrow \{0, 1\}^n$

Decryption:



(estendibile fino a t)

Caratteristiche:

- Per poter fare la decryption, F_k deve essere invertibile
- Encryption **non è parallelizzabile** (per calcolare il blocco i -esimo, devo aver calcolato il blocco $(i - 1)$ -esimo)
- Decryption **non è random access** (perchè comunque dipende dal blocco precedente)
- La lunghezza del ciphertext è esattamente un blocco in più rispetto a quella del plaintext (perchè ci aggiungo $y_0 = r$), ma ci va bene: $|E_k(X)| = |x| + n$

Teorema:

Se F è PRF, allora CBC è **CPA-sicuro**

Esercizio: CBC-mode con F_k non pseudo-random:

Ci aspettiamo che non sia CPA-sicuro perchè F_k non è PRF

Come PRF prendiamo $F_k(x) = x \oplus k$ come l'OTP che sappiamo **non essere** pseudo-random

In realtà **non è neanche** EAV-sicuro

$$PrivK_{M,\pi}^{eav}(n)$$

$$\begin{aligned} A : \quad & k \leftarrow Gen(1^n) \quad b \leftarrow \{0,1\} \\ M \rightarrow A : \quad & X_0 = x_0 \ x_0 \quad X_1 = x_0 \ x_1 \quad \text{con } x_0 \neq x_1 \\ M : \quad & b_m = ? \end{aligned}$$

$b = 0$:

$$y_0 = r \quad y_1 = (r \oplus x_0) \oplus k = r \oplus x_0 \oplus k \quad y_2 = (y_1 \oplus x_0) \oplus k = (r \oplus x_0 \oplus k \oplus x_0) \oplus k = r$$

$b = 1$:

$$y_0 = r \quad y_1 = (r \oplus x_0) \oplus k = r \oplus x_0 \oplus k \quad y_2 = (y_1 \oplus x_1) \oplus k = (r \oplus x_0 \oplus k \oplus x_1) \oplus k = r \oplus x_0 \oplus x_1$$

$b_d = (y_2 == y_0) ? 0 : 1$ sfruttando il risultato ottenuto semplificando y_2 nel caso $b = 0$ (controlla se l'ultimo blocco del ciphertext è uguale al primo blocco del ciphertext)

$$\begin{aligned} Pr(b_m = 0 \mid b = 0) &= 1 \quad \text{vince sempre nel caso } b = 0 \\ Pr(b_m = 1 \mid b = 1) &= 1 - Pr(b_m = 0 \mid b = 1) = 1 - 0 = 1 \end{aligned}$$

Come abbiamo calcolato $Pr(b_m = 0 \mid b = 1) = 0$?

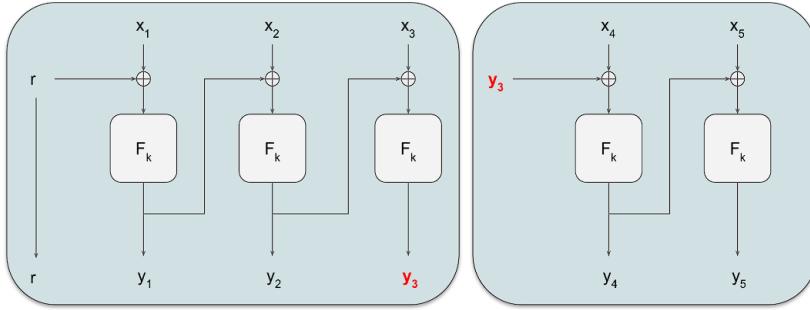
$Pr(b_m = 0 \mid b = 1)$ è la probabilità che M sbagli, ovvero dica 0 quando in realtà $b = 1$.

M dice 0 quando $y_2 = y_0$, ma quando $b = 1$, $y_2 = r \oplus x_0 \oplus x_1$. Per avere $y_2 = y_0$, y_2 dovrebbe essere uguale a r e questo è possibile solo quando $x_0 = x_1$, ma nel caso $b = 1$ questo è impossibile perchè A sta cifrando $x_b = X_1 = x_0 \ x_1 \neq x_0 \ x_0$.

In altre parole, $Pr((x_0 \oplus x_1) = 0) = 0$

$$Pr(PrivK_{M,\pi}^{eav}(n) = 1) = 1/2 \cdot (Pr(b_m = 0 \mid b = 0) + Pr(b_m = 1 \mid b = 1)) = 1/2 \cdot (1 + 1) = 1 \rightarrow \text{CBC-OTP non è EAV-sicuro}$$

Chained CBC (modifica insicura di CBC):



Show that chained CBC mode is **not** CPA-secure

È una variante **insicura** del CBC mode: ogni nuovo blocco non genera un y_0 random come dovrebbe fare il CBC, ma utilizza invece l'ultimo pezzo del blocco precedentemente cifrato come nuovo y_0 .

$PrivK_{M,\pi}^{cpa}(n)$:

1. $A : k \leftarrow Gen(1^n) \quad b \leftarrow \{0,1\}$
2. $M \rightarrow A : X_0 = x_1 \ x_2 \ x_3 \quad X_1 = x'_1 \ x_2 \ x_3 \quad x_1 \neq x'_1$
3. $A \rightarrow M : y = E_k(X_b) = y_1 \ y_2 \ y_3$

M usa E_k in oracle mode:

$M \rightarrow A : x_4 \ x_5 \ x_6 \quad con \ x_4 = r \oplus x_1 \oplus y_3$

$A \rightarrow M : y = E_k((r \oplus x_1 \oplus y_3) \ x_5 \ x_6)$

Per cifrare questo nuovo plaintext scelto da M :

$$E_k((r \oplus x_1 \oplus y_3) \ x_5 \ x_6) = F_k((r \oplus x_1 \oplus y_3) \oplus y_3) \ F_k(y_3 \oplus x_5) \ F_k(y_4 \oplus x_6)$$

Quindi il primo carattere del ciphertext diventa:

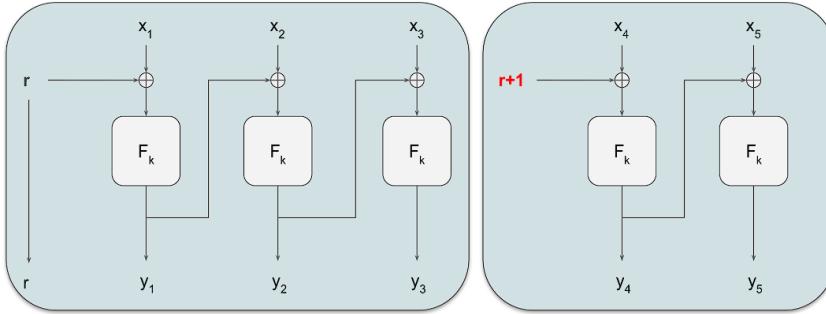
$$y_4 = F_k((r \oplus x_1 \oplus y_3) \oplus y_3) = F_k(r \oplus x_1) \quad \text{che è lo stesso carattere di } y_1 \text{ se } b = 0$$

Quindi la strategia di M è:

4. $M : b_m = (y_1 == y_4) ? 0 : 1$

$$\Pr(PrivK_{M,\pi}^{cpa}(n) = 1) = 1$$

Counter CBC (modifica insicura di CBC):



Show that counter CBC mode is **not** CPA-secure

È un'altra variante **insicura** del CBC mode: ogni nuovo blocco non genera un y_0 random come dovrebbe fare il CBC, ma utilizza invece il valore di r del primo blocco aumentato di 1 come nuovo y_0 .

$PrivK_{M,\pi}^{cpa}(n)$:

1. $A : k \leftarrow Gen(1^n) \quad b \leftarrow \{0, 1\}$
2. $M \rightarrow A : X_0 = x_1 \ x_2 \ x_3 \quad X_1 = x'_1 \ x_2 \ x_3 \quad x_1 \neq x'_1$
3. $A \rightarrow M : y = E_k(X_b) = y_1 \ y_2 \ y_3$

M usa E_k in oracle mode:

$$M \rightarrow A : x_4 \ x_5 \ x_6 \quad \text{con } x_4 = r \oplus x_1 \oplus (r + 1)$$

$$A \rightarrow M : y = E_k((r \oplus x_1 \oplus (r + 1)) \ x_5 \ x_6)$$

Per cifrare questo nuovo plaintext scelto da M :

$$E_k((r \oplus x_1 \oplus (r + 1)) \ x_5 \ x_6) = F_k((r \oplus x_1 \oplus (r + 1)) \oplus (r + 1)) \ F_k((r + 1) \oplus x_5) \ F_k((r + 1) \oplus x_6)$$

Quindi il primo carattere del ciphertext diventa:

$$y_4 = F_k((r \oplus x_1 \oplus (r + 1)) \oplus (r + 1)) = F_k(r \oplus x_1) \quad \text{che è lo stesso carattere di } y_1 \text{ se } b = 0$$

Quindi la strategia di M è:

$$4. M : b_m = (y_1 == y_4) ? 0 : 1$$

$$\Pr(PrivK_{M,\pi}^{cpa}(n) = 1) = 1$$

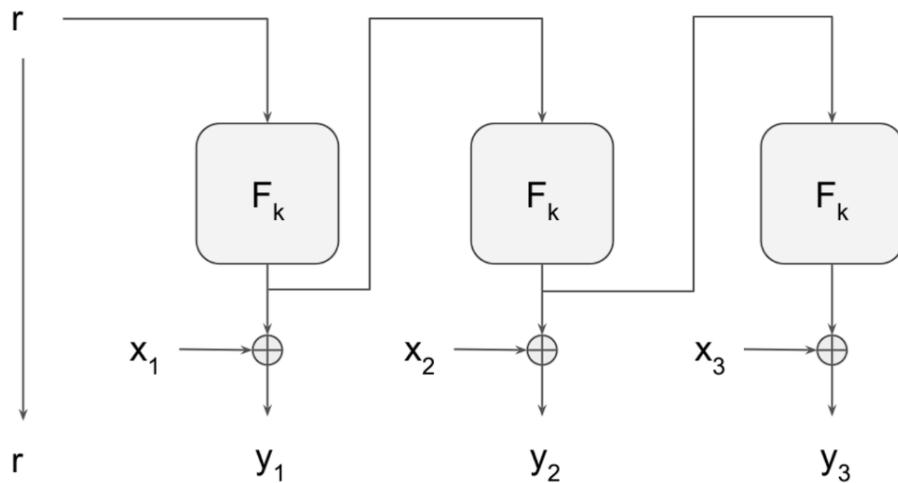
Considerazione:

La strategia nelle due varianti insicure è stata quella di annullare tramite lo xor i valori noti dell'initialization vector: infatti questo parametro, per avere senso dovrebbe essere random!

OFB-mode:

Output FeedBack

Encryption:



(estendibile fino a t)

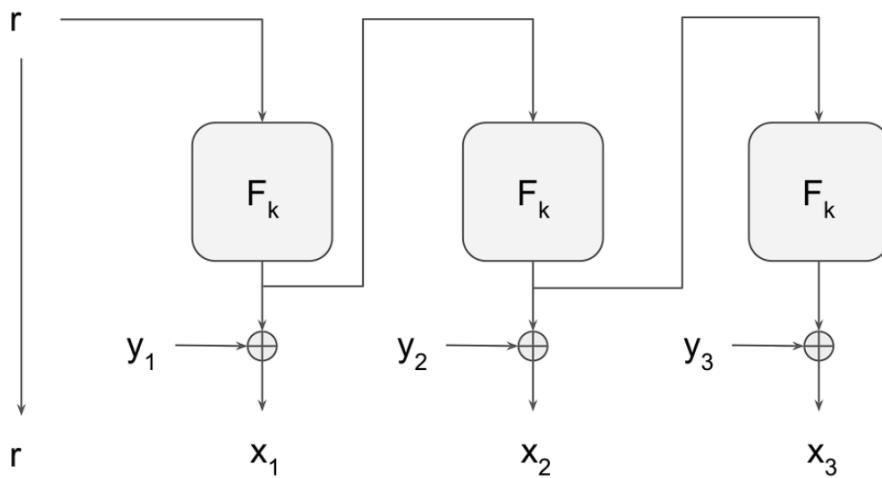
La differenza con CBC è che:

In CBC, mettevo in input di F_k il blocco di plaintext xorato: $F_k(x_i \oplus y_{i-1})$

In OFB, metto il blocco di plaintext xorato in uscita a F_k

In input a F_k ci va' l'output dell' F_k precedente.

Decryption:



Caratteristiche:

- Encryption è **probabilistica**
- Per poter fare la decryption, **non è richiesto** che F_k sia invertibile
- Posso precalcolare (in maniera **sequenziale** e non parallela) tutte le F_k (keystream: dipende soltanto da r e da k), addirittura prima di conoscere il plaintext x
- Se ho **precalcolato il keystream**, la decryption è **random access**
- La lunghezza del ciphertext è esattamente un blocco in più rispetto a quella del plaintext (perchè ci aggiungo $y_0 = r$) (è così per tutti i modes of operation tranne ECB)

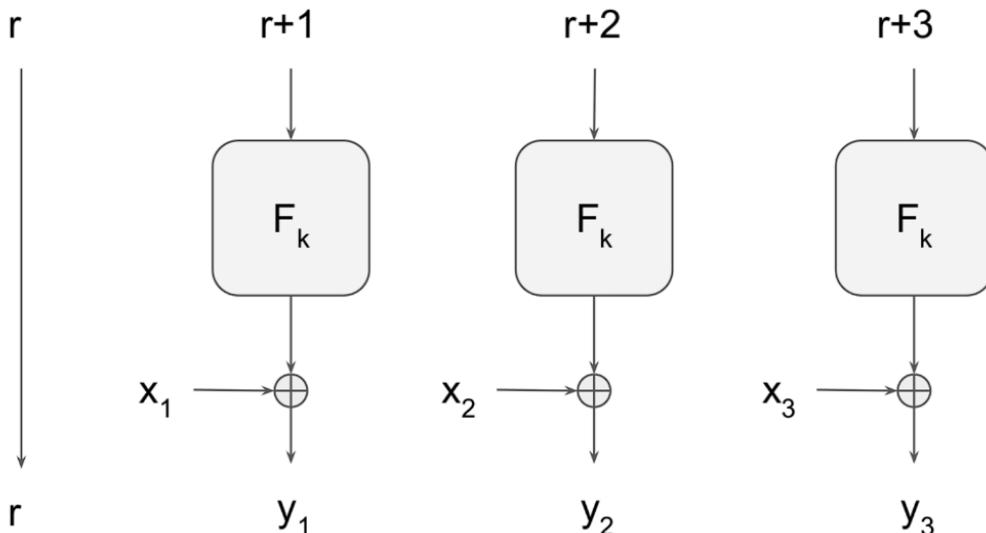
Teorema:

Se F è PRF, allora OFB è **CPA-sicuro**

CTR-mode:

CounTeR mode

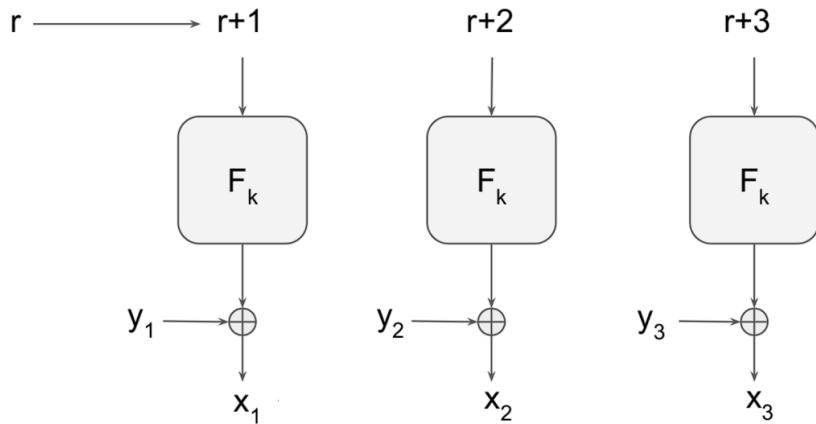
Encryption:



(estendibile fino a t)

Incremento r per ogni blocco

Decryption:



(estendibile fino a t)

Caratteristiche:

- Come per OFB, per fare la decryption, non è richiesto che F_k sia invertibile
- Questa volta posso precalcolare (prima di conoscere il plaintext) il **keystream** anche in **parallelo** (F_k non dipende più da quella precedente)
- In generale, l'encryption è **completamente parallelizzabile**
- Se ho precalcolato il keystream, la decryption è **random access**
- Come per gli altri modes of operation, la lunghezza del ciphertext è esattamente un blocco in più rispetto a quella del plaintext (ci aggiungo $y_0 = r$)

Teorema:

Se F è PRF, allora CTR è **CPA-sicuro**

Message Authentication Codes

Finora ci siamo occupati dell'aspetto della **segretezza** di un messaggio.

Sappiamo però che su un canale pubblico, M può leggere i messaggi, ma anche eliminarli o modificarli.

Ignoriamo il caso in cui M elimini i messaggi e concentriamoci sull'aspetto legato all'**autenticità** dei messaggi.

Garantire l'autenticità con le primitive di encryption per la sicurezza non va bene.

Insieme al messaggio, inviamo un **authentication tag**.

Schema MAC:

Tripla di algoritmi *PPTIME*:

- Gen* algoritmo **probabilistico** per generare le chiavi: prende in input dimensione chiave 1^n e restituisce chiave di dimensione $|k| \geq n$
Mac algoritmo (può essere probabilistico o deterministico): prende in input la chiave k e un messaggio $m \in \{0, 1\}^*$ e restituisce **tag**
Vrfy algoritmo **deterministico** che prende in input k, m, t e restituisce un bit $b \in \{0, 1\}$: 0 se t **non** è il tag di m ; 1 se t è il **tag** di m

```
 $k \leftarrow Gen(1^n)$ 
 $t \leftarrow Mac_k(m)$ 
 $b := Vrfy_k(m, t)$ 
```

Condizione di correttezza di uno schema MAC:

$$\forall k \leftarrow Gen(1^n) . \forall m \in \{0, 1\}^* \quad Vrfy_k(m, Mac_k(m)) = 1$$

Verifica canonica (solo se Mac è deterministico):

$$Vrfy_k(m, t) = \{t' = Mac_k(m); (t' == t) ? 1 : 0\}$$

Destinatario ricalcola lo stesso authentication tag (che è deterministico).

Attacco: forgery

L'obiettivo dell'attaccante M è quello di **falsificare** l'authentication tag.

Supponiamo che M conosca un insieme Q di messaggi (supponiamo che li lui per avere un avversario più potente che ricorda la CPA-security: chosen message attack) e i relativi y_i authentication tags.

L'attacco ha successo se M riesce a produrre una forgery **per nuovi messaggi** ($\notin Q$) con **probabilità non negligible**.

Una forgery è una coppia $(m, t) . m \notin Q$

M conosce 1^n , la lunghezza della chiave, ma non dovrebbe conoscere la chiave (principio di Kerchoffs)

Esperimento message authentication:

MAC-forge_{M,π}(n):

1. $A : k \leftarrow Gen(1^n)$
2. M ha oracle access a Mac_k :
 $M \rightarrow A : Q = \{m_1 \dots m_q\}$
 $A \rightarrow M : \{Mac_k(m) | m \in Q\}$
3. $M : (m, t) \quad M$ deve produrre una forgery

$$MAC\text{-}forge_{M, \pi}(n) = 1 \text{ se } Vrfy_k(m, t) = 1 \text{ and } m \notin Q$$

π is existentially unforgeable under adaptive chosen message attacks se:

$$\forall M \in PPTIME : \Pr(\text{MAC-forg}e_{M,\pi}(n) = 1) \leq \text{negl}(n)$$

Existentially unforgeable \rightarrow non esiste neanche uno

Chosen message attacks: è M che passa Q ad A come oracle

Non compare più $1/2$ perché non c'è più il bit random.

Esercizio #1

F_k è una PRF

$$Mac_k(m_1 \dots m_L) = F_k(m_1) \oplus \dots \oplus F_k(m_L) \quad |m_i| = n$$

Qual è la $Vrfy_k$?

Mac_k è deterministico quindi la $Vrfy_k$ è quella canonica

È unforgeable?

Strategia #1:

$MAC\text{-forg}e_{M,\pi}(n)$:

1. $A : k \leftarrow Gen(1^n)$
2. M ha oracle access a Mac_k :
 $M \rightarrow A : Q = \{\emptyset\}$ M sceglie di non inviare nessun messaggio
3. $M : (m_1 m_1, 0^n)$ M sceglie come messaggio $m = m_1 m_1$ e come tag $t = 0^n$

Controlliamo se $(m_1 m_1, 0^n)$ è una forgery, ovvero se $Vrfy_k(m_1 m_1, 0^n) = 1$

$$Vrfy_k(m_1 m_1, 0^n) = 1 \text{ se } Mac_k(m_1 m_1) = 0^n$$

$$Mac_k(m_1 m_1) = F_k(m_1) \oplus F_k(m_1) = 0^n$$

$$m_1 m_1 \notin Q = \{\emptyset\}$$

$$\Pr(MAC\text{-forg}e_{M,\pi}(n) = 1) = 1 \quad M \text{ vince sempre}$$

Strategia #2:

$MAC\text{-forg}e_{M,\pi}(n)$:

1. $A : k \leftarrow Gen(1^n)$
2. M ha oracle access a Mac_k :
 $M \rightarrow A : Q = \{m_1 m_2\}$
3. $A \rightarrow M : t = Mac_k(m_1 m_2)$
4. $M : (m_2 m_1, t)$

Controlliamo se $(m_2 m_1, t)$ è una forgery, ovvero se $Vrfy_k(m_2 m_1, t) = 1$

$$Vrfy_k(m_2 m_1, t) = 1 \text{ se } Mac_k(m_1 m_1) = t$$

$$Mac_k(m_2 m_1) = F_k(m_2) \oplus F_k(m_1) = F_k(m_1) \oplus F_k(m_2) = t$$

$$m_2 m_1 \notin Q = \{m_2 m_1\}$$

$$\Pr(MAC\text{-forg}e_{M,\pi}(n) = 1) = 1 \quad M \text{ vince sempre}$$

Esercizio #2

F_k è una PRF

$$Mac_k(m_0 \dots m_1) = F_k(0 \ m_0) \ F_k(1 \ m_1) \quad |m_0| = |m_1| = n - 1$$

Qual è la $Vrfy_k$?

Mac_k è deterministico quindi la $Vrfy_k$ è quella canonica

È **unforgeable**?

$MAC\text{-}forge_{M,\pi}(n)$:

$$1. A : \quad k \leftarrow Gen(1^n)$$

2. M ha oracle access a Mac_k :

$$M \rightarrow A : \quad Q = \{m_0 \ m_0 \quad m_1 \ m_1\}$$

$$A \rightarrow M : \quad t = Mac_k(m_0 \ m_0) = F_k(0 \ m_0) \quad t' = Mac_k(m_1 \ m_1) = F_k(1 \ m_1)$$

$$3. M : \quad (m_0 \ m_1, \ F_k(0 \ m_0) \ F_k(1 \ m_1))$$

$$Pr(MAC\text{-}forge_{M,\pi}(n) = 1) = 1 \quad M \text{ vince sempre}$$

$$m_0 \ m_1 \notin Q = \{m_0 \ m_0 \quad m_1 \ m_1\}$$

M non ha esplicitamente calcolato i valori di F_k : ha **riciclato intelligentemente** quelli che ha ricevuto dall'oracle.

Esercizio #3

F_k è una PRF

$$Mac_k(m_0 \ m_1) = F_k(m_0) \ F_k(F_k(m_1)) \quad |m_0| = |m_1| = n$$

È **unforgeable**?

$MAC\text{-}forge_{M,\pi}(n)$:

$$1. A : \quad k \leftarrow Gen(1^n)$$

2. M ha oracle access a Mac_k :

$$M \rightarrow A : \quad Q = \{m_0 \ m_0 \quad m_1 \ m_1\}$$

$$A \rightarrow M : \quad t = Mac_k(m_0 \ m_0) = F_k(m_0) \ F_k(F_k(m_0)) \quad t' = Mac_k(m_1 \ m_1) = F_k(m_1) \ F_k(F_k(m_1))$$

$$3. M : \quad (m_0 \ m_1, \ F_k(m_0) \ F_k(F_k(m_1)))$$

$$m_0 \ m_1 \notin Q = \{m_0 \ m_0 \quad m_1 \ m_1\}$$

Esercizio #4

F_k è una PRF

$$Mac_k(m_1 \dots m_L) = F_k(1 m_1) \oplus \dots \oplus F_k(L m_L)$$

È **unforgeable**?

$MAC\text{-}forge_{M,\pi}(n)$:

1. $A : k \leftarrow Gen(1^n)$

2. M ha oracle access a Mac_k :

$$M \rightarrow A : Q = \{m_1 m_2 \quad m_2 m_1 \quad m_2 m_2\}$$

$A \rightarrow M :$

$$t = Mac_k(m_1 m_2) = F_k(1 m_1) \oplus F_k(2 m_2)$$

$$t' = Mac_k(m_2 m_1) = F_k(1 m_2) \oplus F_k(2 m_1)$$

$$t'' = Mac_k(m_2 m_2) = F_k(1 m_2) \oplus F_k(2 m_2)$$

3. $M : (m_1 m_1, t \oplus t' \oplus t'')$

$$Mac_k(m_1 m_1) = F_k(1 m_1) \oplus F_k(2 m_1)$$

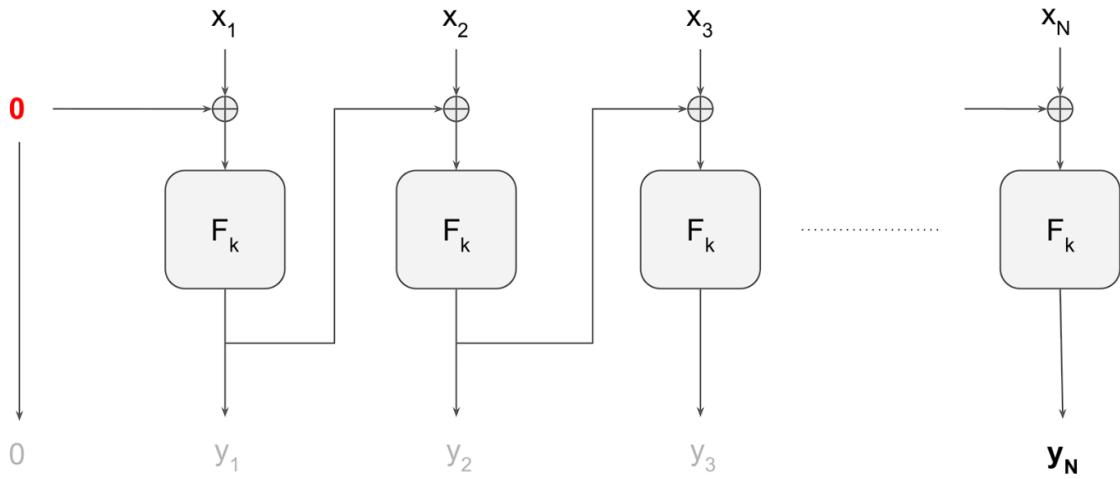
$$t \oplus t' \oplus t'' = (F_k(1 m_1) \oplus F_k(2 m_2)) \oplus (F_k(1 m_2) \oplus F_k(2 m_1)) \oplus (F_k(1 m_2) \oplus F_k(2 m_2)) =$$

$$= F_k(1 m_1) \oplus F_k(2 m_2) \oplus F_k(1 m_2) \oplus F_k(2 m_1) \oplus F_k(1 m_2) \oplus F_k(2 m_2) =$$

$$= F_k(1 m_1) \oplus \cancel{F_k(2 m_2)} \oplus \cancel{F_k(1 m_2)} \oplus F_k(2 m_1) \oplus \cancel{F_k(1 m_2)} \oplus \cancel{F_k(2 m_2)} = F_k(1 m_1) \oplus F_k(2 m_1)$$

$$m_1 m_1 \notin Q = \{m_1 m_2 \quad m_2 m_1 \quad m_2 m_2\}$$

CBC-MAC



$$m = x_1 \dots x_N$$

$$\text{Mac}_k(m) := y_N$$

Best known attack obtains a forgery with probability $\frac{1}{2}$ using $O(2^{t/2})$ MAC requests, where $F_k : \{0,1\}^t \rightarrow \{0,1\}^t$

L'initialization vector r non è più random, ma una costante (in genere $r = 0$).

L'algoritmo Mac_k è **deterministico** e quindi la $Vrfy_k$ è quella **canonica**.

Come tag uso soltanto gli ultimi n bit (y_n)

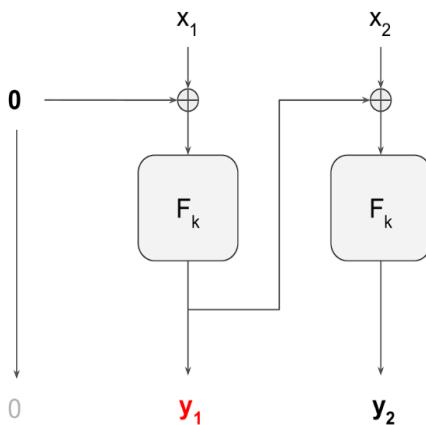
Esiste un attaccante noto che riesce a romperlo con $Pr = 1/2$ ma è un attaccante inefficiente (esponenziale).

Importante: la dimensione dei messaggi è **predefinita** e non arbitraria: $l = L(n)$

Teorema:

F è PRF \implies CBC-MAC è **sicuro** $\forall m \in \{0,1\}^{l \cdot n}$ (sui messaggi della lunghezza rischiesta)

Esercizio #1 (variante insicura CBC-MAC)



$$L(n) = 2 \quad m = x_1 x_2 \quad \text{Mac}_k(m) := y_1 y_2$$

In questa variante **insicura** dove $L(n) = 2$ il tag non è più composto dagli ultimi bit dell'ultimo blocco y_2 ma viene usato anche il primo blocco (y_1)

MAC-forge_{M,π}(n):

1. $A : k \leftarrow Gen(1^n)$
2. M ha oracle access a Mac_k :
 $M \rightarrow A : Q = \{0^n m_1\}$
 $A \rightarrow M : t = Mac_k(0^n m_1) = t_1 t_2 \quad t_1 = F_k(0^n \oplus 0^n) = F_k(0^n) \quad t_2 = F_k(t_1 \oplus m_1)$
3. $M : (0^n t_1, t_1 t_1)$

Vrfy_k(0ⁿ t₁, t₁ t₁):

$$Mac_k(0^n t_1) = y_1 y_2$$

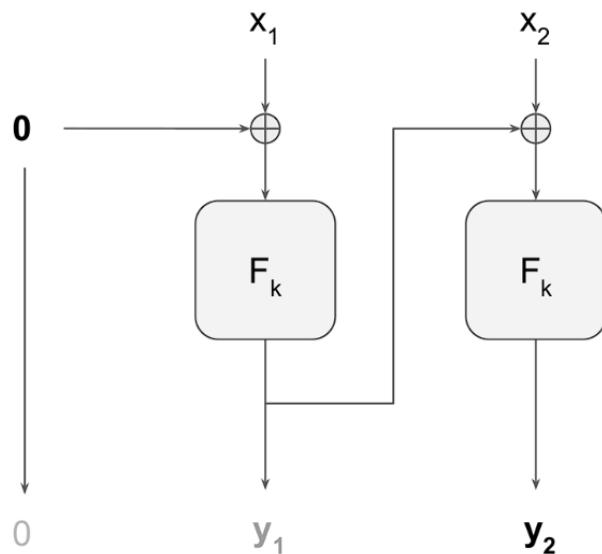
$$y_1 = F_k(0^n) = t_1$$

$$y_2 = F_k(y_1 \oplus t_1) = F_k(0^n) = t_1$$

$0^n t_1 \notin Q = \{0^n m_1\}$ con $m_1 \neq t_1$ m_1 non era importante, potevo passarli qualunque cosa

$$Pr(\text{MAC-forge}_{M,\pi}(n) = 1) = 1$$

Esercizio #2 (variante insicura CBC-MAC)



$$L(n) = 2 \quad m = x_1 x_2 \quad Mac_k(m) := y_2 \quad \text{Vrfy}_k(x_1 \dots x_N, t) = (y_N = t)$$

In quest'altra variante **insicura** M può produrre una forgery di dimensione arbitraria nel messaggio

$MAC\text{-}forge}_{M,\pi}(n)$:

1. $A : k \leftarrow Gen(1^n)$
2. M ha oracle access a Mac_k :

$M \rightarrow A : m_1 m_2$

$A \rightarrow M :$

$$t = Mac_k(m_1 m_2) = F_k(F_k(0^n \oplus m_1) \oplus m_2) = F_k(F_k(m_1) \oplus m_2)$$

$M \rightarrow A : t m_4$

$$A \rightarrow M : t' = Mac_k(t m_4) = F_k(F_k(0^n \oplus t) \oplus m_4) = F_k(F_k(t) \oplus m_4)$$

3. $M : (m_1 m_2 0^n m_4, t')$

$Vrfy_k(m_1 m_2 0^n m_4, t')$:

$$Mac_k(m_1 m_2 0^n m_4)$$

$$y_4 = F_k(y_3 \oplus m_4)$$

$$y_3 = F_k(y_2 \oplus 0^n) = F_k(y_2)$$

$$y_2 = F_k(y_1 \oplus m_2)$$

$$y_1 = F_k(0^n \oplus m_1) = F_k(m_1)$$

$$y_2 = F_k(F_k(m_1) \oplus m_2) = t$$

$$y_3 = F_k(y_2)$$

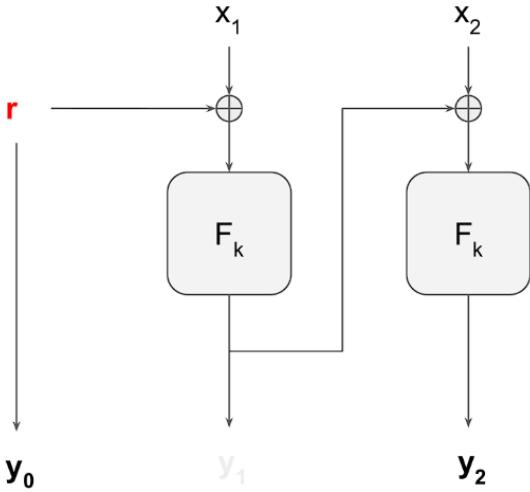
$$y_4 = F_k(F_k(y_2) \oplus m_4) = F_k(F_k(t) \oplus m_4) = t'$$

$Vrfy_k(m_1 m_2 0^n m_4, t') = 1 \quad y_N = y_4 = t'$

$$m_1 m_2 0^n m_4 \notin Q = \{m_1 m_2, t m_4\}$$

$$\Pr(\text{MAC-forge}_{M,\pi}(n) = 1) = 1$$

Esercizio #3 (variante insicura CBC-MAC)



$$L(n) = \ell \quad m = x_1 \dots x_\ell \quad Mac_k(m) := y_0 \ y_N$$

MAC-forge_{M,π}(n):

1. $A : k \leftarrow Gen(1^n)$
2. M ha oracle access a Mac_k :
 $M \rightarrow A : Q = \{0^n \ m_2\}$
 $A \rightarrow M : t = Mac_k(0^n \ m_2) = t_0 \ t_2$
 t_0 random
 $\cancel{t_1} = F_k(t_0 \oplus 0^n) = F_k(t_0)$
 $t_2 = F_k(t_1 \oplus m_2) = F_k(F_k(t_0) \oplus m_2)$
3. $M : (t_0 \ m_2, \ 0^n \ t_2)$

$Mac_k(t_0 \ m_2) :$

$$\begin{aligned}
 y_0 &= r = 0^n && \text{va bene qualunque cosa, in questo caso è } 0^n \\
 \cancel{y_1} &= F_k(t_0 \oplus y_0) = F_k(t_0 \oplus 0^n) = F_k(t_0) \\
 y_2 &= F_k(F_k(t_0) \oplus m_2) = t_2
 \end{aligned}$$

Autenticazione entità

Vogliamo **verificare l'identità del mittente** di un messaggio.

Chiamiamo **claimant** il partecipante che dichiara di essere qualcuno.

Chiamiamo **verifier** il partecipante che verifica l'identità del claimant.

Indichiamo con $I(A)$ un attaccante che cerca di **impersonare** un partecipante, in questo caso A .

Indichiamo con i la prima sessione di comunicazione.

Indichiamo con ii una nuova sessione di comunicazione diversa da quella iniziale.

Autenticazione con MAC (v1, insicura):

Il verifier B chiede ad A , il claimant, il MAC su un messaggio **fissato** M con una chiave simmetrica k_{AB} .

- (i) 1. $B \rightarrow A : m$
- (i) 2. $A \rightarrow B : MAC_{k_{AB}}(m)$

Questo protocollo funziona, ma non è **per niente sicuro**.

Freshness attack:

M impersona A e manda al verifier, in un'altra sessione, gli stessi risultati calcolati da A , che M ha osservato sul canale pubblico:

- (i) 1. $B \rightarrow A : m$
- (i) 2. $A \rightarrow B : MAC_{k_{AB}}(m)$

- (ii) 1. $B \rightarrow I(A) : m$
- (ii) 2. $I(A) \rightarrow B : MAC_{k_{AB}}(m)$

Autenticazione con MAC (v2, insicura):

Per risolvere il problema del **freshness attack** introduciamo una **random challenge**, una bitstring random, al posto del messaggio fisso m .

- (i) 1. $B \rightarrow A : r$
- (i) 2. $A \rightarrow B : MAC_{k_{AB}}(r)$

Non è più possibile effettuare un freshness attack perché r cambia ad ogni sessione, ma questo protocollo è vulnerabile ai **reflection attacks**.

Reflection attack:

M impersona A , il verifier B propone a $I(A)$, ovvero M , una random challenge r che M non ha modo di risolvere non conoscendo la chiave k_{AB}

M però apre una nuova sessione con B dove M è questa volta il verifier e B è il claimant (dichiara di essere proprio B): M chiede a B di autentificarsi con la stessa random challenge r :

- (i) 1. $B \rightarrow I(A) : r$

- (ii) 1. $I(A) \rightarrow B : r$
- (ii) 2. $B \rightarrow I(A) : MAC_{K_{AB}}(r)$

- (i) 2. $I(A) \rightarrow B : MAC_{k_{AB}}(r)$

Autenticazione con MAC (v3, sicura):

Questo protocollo è **sicuro**: adesso chi calcola il MAC deve aggiungere anche una bitstring che identifica i partecipanti: in questo modo M non può fare nulla con $MAC_{K_{AB}}(B, r)$ calcolato da B :

- (i) 1. $B \rightarrow A : r$
- (i) 2. $A \rightarrow B : MAC_{k_{AB}}(A, r)$

Funzioni hash:

Una funzione hash H è un algoritmo **deterministico** (si chiama proprio *funzione*) che mappa un messaggio di **lunghezza arbitraria** * in un messaggio di **lunghezza fissa** l (questo messaggio si chiama **message digest**, sintesi del messaggio):

$$H \in \{0,1\}^* \rightarrow \{0,1\}^l$$

In algoritmi e strutture dati, le funzioni hash sono utilizzate per implementare le **tabelle hash**, strutture dati efficienti con operazione di ricerca efficiente in tempo costante $O(1)$: l'indice del dato è l'indice della tabella hash.

La funzione H deve ridurre le *collisioni*: coppie di messaggi **diversi** che producono però lo **stesso hash**

$$(x, x') \quad x \neq x' \quad H(x) = H(x')$$

In crittografia vogliamo che trovare le collisioni sia un'operazione **inefficiente**.

Funzioni hash crittografiche (definizione incompleta):

Una funzione hash crittografica H con **digest lenght** $l(n)$ dove n è il **parametro di sicurezza** è un algoritmo **deterministico**:

$$H \in \{0,1\}^* \rightarrow \{0,1\}^{l(n)}$$

Intuitivamente, aumentando il parametro n , aumentiamo anche la dimensione del message digest, riducendo la probabilità di trovare una collisione (aumentando così la sicurezza).

Funzione di compressione:

In generale, non siamo interessati esclusivamente a tutte le funzioni che prendono in input un messaggio di lunghezza arbitraria.

Certe volte ci interessano le funzioni hash dove la dimensione dell'input è **fissata**.

Funzione di compressione per messaggi di lunghezza fissata: $\forall n \exists l' : H \in \{0,1\}^{l'(n)} \rightarrow \{0,1\}^{l(n)} \quad l'(n) > l(n)$

$l'(n) > l(n) \rightarrow H$ non è iniettiva

Se $l'(n) \leq l(n)$ sarebbe banale evitare le collisioni, come per esempio nella funzione identità $H(x) = x$ che ovviamente non ci interessa.

In generale, **non** ci interessano le funzioni hash *iniettive*, le collisioni ci saranno ma è importante che calcolare queste collisioni, per l'attaccante, sia un'**operazione inefficiente**.

Funzione hash collision resistant (definizione errata):

Definiamo **erroneamente** l'esperimento di **collision resistance** come:

$Hash\text{-}coll_{M,H}(n) :$

$M : (x, x') \quad M$ deve trovare una collisione

L'esperimento ha successo se M ha trovato una collisione: $Hash\text{-}coll_{M,H}(n) = 1$ se $x \neq x'$, $H(x) = H(x')$

Definiamo (**erroneamente**) **H collision resistant** se $\forall M \in PPTIME : Pr(Hash\text{-}coll_{M,H}(n) = 1) \leq negl(n)$

Questa definizione **ha un problema**:

Abbiamo detto che se $l'(n) > l(n)$, la funzione H **non** può essere iniettiva, quindi ci saranno per forza delle collisioni...

Se M è un avversario che non calcola niente, ma **si limita a** restituire una **collisione** (x, x') **hardcoded**, allora

$\exists M \in O(1) : Hash\text{-}coll_{M,H}(n) = 1 \rightarrow H$ **non** è collision resistant

Funzioni hash crittografiche (definizione completa):

M si limitava a restituire una **collisione hardcoded**, per evitare questo problema, aggiungiamo una **chiave** (o seed) s come **parametro** della funzione hash; questa chiave però non è segreta, ma visibile, infatti serve solo per migliorare la definizione di collisione resistance escludendo l'avversario con la collisione hardcoded:

$$s \leftarrow Gen(1^n)$$

$H^s(x)$ adesso H prende come parametro anche s

Una funzione hash crittografica \mathbf{H} con digest lenght $l(n)$, allora, è una **coppia di algoritmi**:

$$\mathbf{H} = (H, Gen)$$

$$Gen \in 1^n \rightarrow \{0, 1\}^n$$
$$H \in (\{0, 1\}^* \rightarrow \{0, 1\}^n) \rightarrow \{0, 1\}^{l(n)}$$

Funzione hash collision resistant (definizione corretta):

$Hash\text{-}coll}_{M, \mathbf{H}}(n)$:

1. $A : s \leftarrow Gen(1^n)$
2. $M : (x, x')$

L'esperimento ha successo se M ha trovato una collisione: $Hash\text{-}coll}_{M, \mathbf{H}}(n) = 1$ se $x \neq x'$, $\mathbf{H}^s(x) = \mathbf{H}^s(x')$ (una collisione questa volta in \mathbf{H}^s e non più in H senza la chiave/seed random)

\mathbf{H} **collision resistant** se $\forall M \in PPTIME : Pr(Hash\text{-}coll}_{M, \mathbf{H}}(n) = 1) \leq negl(n)$

Esercizio:

$$Gen(1^n) = \{s \leftarrow \{0, 1\}^n\}$$

$$H^s(x) = x[1..n] \oplus s$$

$$|x| = n + 1 \quad l'(n) = n + 1 > n = l(n)$$

H è collision resistant?

L'ultimo bit di x non viene mai usato:

$$M : x = 0^{n+1} \quad x' = 0^n 1$$

$$H^s(x) = 0^n \oplus s = s$$

$$H^s(x') = 0^n \oplus s = s$$

$$H^s(0^{n+1}) = H^s(0^n 1) \rightarrow H^s(x) = H^s(x')$$

$$M \in PPTIME \quad Pr(Hash\text{-}coll}_{M, \mathbf{H}}(n) = 1) = 1$$

Unkeyed hash functions:

Nella pratica, le funzioni hash sono **unkeyed**, non c'è Gen

$$H \in \{0, 1\}^* \rightarrow \{0, 1\}^{l(n)}$$

Quindi le funzioni hash unkeyed non possono essere **collision resistant** secondo la definizione formale.

Indoboliamo allora la nozione sicurezza: definiamole **sicure se** è infattibile trovare una collisione utilizzando meno di $2^{l/2}$ invocazioni della funzione H (**birthday attack**)

Esempi noti di unkeyed hash functions:

Empiricamente, $l \geq 160$

MD5 (1991), $l = 128$

Non più sicuro: si è in grado di produrre **collisioni controllate**. Attacchi noti del 2004 e 2005.

SHA-1 (1995), $l = 160$

SHA: Secure Hash Algorithm (famiglia di cinque algoritmi prodotti dalla NSA americana).

Ipotizzato attacco teorico nel 2005 con meno tentativi del birthday attack.

Realizzato con successo un attacco da parte di Google nel 2017 rendendo così l'algoritmo insicuro.

SHA-2 (2001), $l \in \{256, 512\}$

Variante considerata sicura del SHA-1

SHA-3 (2012), $l \in \{224, 256, 384, 512\}$

Appartenente a una nuova famiglia di funzioni hash diverse da SHA-2 e SHA-1

Preimage resistance:

L'attaccante conosce l'hash di un x random e per vincere, deve trovare un x' (preimage) con lo stesso hash di x

$\text{Hash-preimg}_{M,H}(n)$:

1. $A \rightarrow M : (s, H^s(x)) \quad s \leftarrow \text{Gen}(1^n), \quad x \leftarrow \{0,1\}^{l'(n)}$
2. $M \rightarrow A : x'$

$$\text{Hash-preimg}_{M,H}(n) = 1 \text{ se } H^s(x') = H^s(x).$$

H è **preimage resistant** se:

$$\forall M \in \text{PPTIME} : \Pr(\text{Hash-preimg}_{M,H}(n) = 1) \leq \text{negl}(n)$$

Esercizio:

$$\text{Gen}(1^n) = \{s \leftarrow \{0,1\}^n\}$$

$$H^s \in \{0,1\}^{m \cdot n} \rightarrow \{0,1\}^n \quad m > 1 \quad \rightarrow H^s \text{ è una funzione di compressione}$$

$$H^s(x_1 \dots x_m) = s \oplus shL_1(x_1) \oplus \dots \oplus shL_m(x_m) \quad |x_i| \in \{0,1\}^n$$

H è preimage resistant?

$\text{Hash-preimg}_{M,H}(n)$:

1. $A \rightarrow M : (s, y) \quad s \leftarrow \text{Gen}(1^n), \quad y = H^s(x) \quad x \leftarrow \{0,1\}^{m \cdot n}$
2. $M \rightarrow A : x' = x_1 x_2$

Strategia: riprodurre hash y :

$$H^s(x_1 x_2) = y = s \oplus shL_1(x_1) \oplus shL_2(x_2)$$

Per avere $H^s(x_1 x_2) = y$, $shL_1(x_1) \oplus shL_2(x_2) = s \oplus y$ così ottengo:

$$H^s(x_1 x_2) = \cancel{s} \oplus (\cancel{s} \oplus y) = y$$

$$x_1 = shR_1(s)$$

$$x_2 = shR_2(y)$$

$$H^s(x_1 x_2) = H^s(shR_1(s) shR_2(y)) = s \oplus shL_1(shR_1(s)) \oplus shL_2(shR_2(y))$$

$$= s \oplus \cancel{shL_1}(\cancel{shR_1}(s)) \oplus \cancel{shL_2}(\cancel{shR_2}(y)) = \cancel{s} \oplus \cancel{s} \oplus y = y$$

Second-preimage resistance:

L'attaccante riceve un **messaggio** x e, per vincere deve trovare un $x' \in \{0, 1\}^* \neq x$ che abbia lo stesso hash di x . Ovviamente, M può calcolare l'hash di x

$\text{Hash-2preimg}_{M,H}(n)$:

1. $A \rightarrow M : (s, x) \quad s \leftarrow \text{Gen}(1^n), \quad x \leftarrow \{0, 1\}^{l(n)}$
2. $M \rightarrow A : x'$

$$\text{Hash-2preimg}_{M,H}(n) = 1 \text{ se } x \neq x' \text{ and } H^s(x) = H^s(x').$$

H è **second-preimage resistant** se:

$$\forall M \in \text{PPTIME} : \Pr(\text{Hash-2preimg}_{M,H}(n) = 1) \leq \text{negl}(n)$$

Collision resistance \implies **second-preimage resistance** \implies **preimage resistance**

Nella collision resistance, M può trovare coppie (x, x') arbitrarie; nella second-preimage resistance è limitato a trovare un x' che abbia lo stesso hash del messaggio ricevuto x ; mentre nella preimage resistance deve trovare l' x' che abbia lo stesso hash y che ha ricevuto da A .

Birthday attack:

In crittografia, gli attacchi a forza bruta sono sempre possibili (tranne che nella perfect secrecy).

Il **birthday attack** è un attacco alle funzioni hash che ha come obiettivo quello di trovare una collisione a forza bruta:

Prende in input la funzione H e un numero q che rappresenta i **tentativi** che l'attaccante è disposto a effettuare per cercare la collisione.

$\text{findCollision}(H, q)$:

```
for each i in 1..q :  
     $x_i \leftarrow \{0, 1\}^*$   
     $y_i = H(x_i)$   
if  $y_i = y_j$  for some  $i \neq j$  return  $(x_i, x_j)$   
return failure
```

Paradosso del compleanno:

In una classe di 23 studenti, la probabilità di trovare due studenti nati nello stesso giorno è $1/2$.

Probabilità di successo di un birthday attack:

Per avere una probabilità di successo $\varepsilon = 1/2$ sono necessari **circa** $q = 1.17 \cdot 2^{l/2}$ tentativi (ignoriamo la costante 1.17 e ci concentriamo su $2^{l/2}$).

Consideriamo sicuri gli $l \geq 160$.

Collisioni significative (controllate):

Nell'esperimento di **collision resistance**, M deve trovare una **qualunque coppia** $/(x, x')$, nella pratica è più interessante però produrre una **collisione significativa** ovvero che x e x' abbiano una **correlazione**.

Applicazioni delle funzioni hash:

Fingerprinting:

L'obiettivo è scoprire se ci sono state manomissioni su x (che può essere un documento, un hard-disk, etc.). In genere, $|x| = l'(n) > l(n) \rightarrow H(x)$ message digest

M vuole modificare x in x' in modo che x' abbia lo stesso hash di x : questo è un esempio di **second-preimage attack**.

Quindi H deve essere *almeno* second-preimage resistant.

Non possiamo escludere *con certezza assoluta* che il messaggio non sia stato alterato anche se l'hash è rimasto lo stesso, perchè magari M ha trovato una collisione, ma sappiamo che questo è (o dovrebbe essere) al limite dell'impossibile.

Virus fingerprinting:

Invece di confrontare interamente il contenuto di un file scaricato con il codice sorgente dei virus presenti nel repository dell'antivirus, questo può invece, confrontare l'hash del file con l'hash del sorgente dei virus.

Password hashing:

Un database, invece di memorizzare le password in chiaro, può memorizzare l'hash delle password.

M che vuole fare l'accesso al database, vede un hash e deve trovare un x' che abbia lo stesso hash di un x che non conosce: questo è un **preimage attack**.

Anche se H è collision resistant, non possiamo garantire la sicurezza perchè:

1. La definizione di preimage resistance assumeva che il messaggio x di cui si inviava poi ad M l'hash fosse scelto **in modo random**; le password degli utenti hanno invece delle distribuzioni di probabilità.
2. Conosciamo attacchi (birthday attack) che hanno successo in $O(2^{n/2})$ dove $n = |\text{password}|$, quindi se $|\text{password}| = 64 \text{ bit}$, un attaccante riuscirebbe a portare avanti un attacco a forza brutta in $O(2^{32})$.
3. Problema delle **rainbow tables**:

Strutture dati efficienti (con compromessi di tempo e spazio: utilizzano catene di hash e funzioni di riduzione per diminuire la memory footprint) costruibili in solo $O(n^{2/3})$ (efficiente) dove N è la dimensione del *password space*.

Esempio: password di 8 caratteri con 62 possibili caratteri alfanumerici: $N = 62^8 = (2^6)^8 = 2^{48} \rightarrow N^{2/3} \approx 2^{2(48/3)} \approx 2^{32}$

Soluzione #1: rendere il calcolo delle funzioni hash globalmente più lento

Non conviene perchè si limiterebbero anche gli utenti normali.

Soluzione #2: aggiungere del sale random:

Tecnica effettivamente utilizzata: nel database non si memorizza più l'hash della password, ma l'hash della password concatenata a del sale s random: $s \leftarrow \{0, 1\}^n$

$$H(s \parallel \text{password})$$

Rende inutile per l'attaccante precalcolare le rainbow tables.

Key distribution:

Nel contesto della crittografia a chiave simmetrica, A e B si scambiavano una **chiave precondivisa** su un costosissimo **canale privato**.

Introduciamo una nuova entità T nel setting della comunicazione tra i partecipanti (A, B, M e tutti gli altri possibili partecipanti). T è la **trusted authority** che ha il compito di **precondividere** una chiave con tutti i partecipanti:

Esempio: A vuole richiedere a T una chiave (k_{AB}) per comunicare in maniera sicura con B :

T invia a A la chiave k_{AB} cifrata con la chiave (k_{AT}) **precondivisa** tra A e T

$A \Rightarrow_{voglio k_{AB}} T$

$T \Rightarrow_{E_{k_{AT}}(k_{AB})} A$

Needham-Schroeder, 1978 (insicuro):

Protocollo rivelatosi poi **insicuro** (attacco di **Denning-Sacco** del 1981, freshness attack: M utilizza una vecchia chiave di una sessione precedente e B la accetta)

Bellare-Rogaway, 1995 (sicuro):

Versione sicura del Needham-Schroeder basata su MAC

Parametri inviati:

A è una sequenza *univoca* che identifica A

B è una sequenza *univoca* che identifica B

r_A è la *random challenge* che A propone

r_B è la *random challenge* che B propone

1. $A \rightarrow B : A, B, r_a \quad A$ invia a B la propria identità, quella di B e la random challenge r_a
2. $B \rightarrow T : A, B, r_a, r_b \quad B$ fa lo stesso ma con T e gli invia anche quello che A gli aveva inviato
3. $T \rightarrow A : E_{k_{AT}}(k_{AB}), MAC_{k_{AT}}(B, A, r_A, E_{k_{AT}}(k_{AB}))$
4. $T \rightarrow B : E_{k_{BT}}(k_{AB}), MAC_{k_{BT}}(A, B, r_A, E_{k_{BT}}(k_{AB}))$

Nei passi 3 e 4, T invia ai partecipanti la chiave k_{AB} (cifrata) e un MAC per preservare l'autenticità della chiave cifrata. Il MAC è calcolato sulle identità di A e B e le rispettive random challenge oltre che ovviamente la chiave cifrata: A e B hanno tutti gli elementi per verificare il MAC.

Senza random challenge, il protocollo sarebbe vulnerabile agli attacchi di tipo freshness.

Svantaggi del protocollo:

La presenza della trusted authority T :

T (o anche KDC, key distribution center) è un **single point of failure**: se va giù T , va giù tutto il protocollo.

In più, nella crittografia di massa non è fattibile questo approccio.

Crittografia asimmetrica (a chiave pubblica):

Key exchange:

In un **canale pubblico** A e B vogliono scambiarsi una chiave *senza aver mai prima precondiviso* segreti, in assenza di canale privato e in assenza di una trusted authority.

$A \Rightarrow^M B$

M può essere passivo o attivo

Fondamenti di algebra:

Teoria dei gruppi:

Un **gruppo** è una **coppia** (G, \circ) dove G è un insieme e \circ è un'operazione binaria e sono verificate le seguenti condizioni:

- **chiusura:** $\forall x, y \in G : x \circ y \in G$ l'operazione \circ rimane dentro l'insieme
- **associatività:** $\forall x, y, z \in G : x \circ (y \circ z) = (x \circ y) \circ z$
- **esiste identità:** $\exists e \in G : \forall x \in G : x \circ e = x$
- **esiste inverso:** $\forall x \in G : \exists y \in G : x \circ y = e$

Un gruppo è **abeliano** (o commutativo) se vale anche la **commutatività**:

- **commutatività:** $\forall x, y \in G : x \circ y = y \circ x$

Se il gruppo è **finito** indichiamo con $|G|$ il numero degli elementi in G e questo si chiama **ordine** del gruppo.

$(Z_n, +)$ è un **gruppo abeliano** di ordine n

$(Z_n, *)$ **non è sempre** un gruppo: n deve essere primo (tutti gli elementi, escludendo lo 0, coprimi con n), altrimenti non tutti gli elementi (escludendo lo 0) hanno un inverso moltiplicativo

Definizioni:

Divisibilità:

Si indica $d | n$ e si legge d divide n

$$d | n \iff \exists q : n = d \cdot q$$

Esempi: $2 | 6$ $1 | n (\forall n)$ $d | 0 (\forall d)$

Congruenza in modulo n :

Si indica $a =_n b$ e si legge a congruo a b in modulo n

$$a =_n b \iff n | (a - b)$$

Esempio: $25 =_6 1$ ($6 | (25 - 1) = 24$)

Massimo comune divisore:

Si indica con $gcd(a, b)$

$$gcd(a, b) = \max \{d : d | a \text{ and } d | b\}$$

Esempi: $gdc(12, 18) = 6$ $gdc(1, n) = 1 (\forall n > 0)$ $gdc(0, n) = n (\forall n > 0)$

Coprimi:

Si indica con $a \perp b$

$$a \perp b \iff gcd(a, b) = 1$$

Proprietà del massimo comun divisore:

Lemma #1:

$$\gcd(a, b) = \gcd(a, b - a)$$

Lemma #2:

$$\gcd(a, b) = \gcd(a, b \bmod a)$$

Lemma #3:

$$\exists x, y : (x \cdot a) + (y \cdot b) = \gcd(a, b)$$

Lemma #4:

$a \perp n \iff \exists x : a \cdot x =_n 1$ *a e n sono coprimi se e solo se a ammette un **inverso** moltiplicativo x in modulo n*

Esempio (Lemma #4):

$$n = 6$$

- $x = 0 : \gcd(0, 6) = 6 \quad \nexists \text{ inverso} \quad (6 | 6 \text{ e } 6 | 0)$
- $x = 1 : \gcd(1, 6) = 1 \quad \exists \text{ inverso} : 1$
- $x = 2 : \gcd(2, 6) = 2 \quad \nexists \text{ inverso}$
- $x = 3 : \gcd(3, 6) = 3 \quad \nexists \text{ inverso}$
- $x = 4 : \gcd(4, 6) = 2 \quad \nexists \text{ inverso}$
- $x = 5 : \gcd(5, 6) = 1 \quad \exists \text{ inverso} : 5$

$$5 \cdot 0 = 0 \bmod_6 = 0$$

$$5 \cdot 1 = 5 \bmod_6 = 5$$

$$5 \cdot 2 = 10 \bmod_6 = 4$$

$$5 \cdot 3 = 15 \bmod_6 = 3$$

$$5 \cdot 4 = 20 \bmod_6 = 2$$

$$5 \cdot 5 = 25 \bmod_6 = 1 \leftarrow$$

Il gruppo (Z_n^*, \cdot)

$Z_n^* = \{x \in 0..n-1 : x \perp n\}$ tutti i numeri da 0 a $n-1$ che siano **coprimi** con n (lo 0 non ci sarà mai)

Non è detto che aumentando n , aumenti anche l'ordine del gruppo:

$$Z_2^* = \{1\}$$

$$Z_3^* = \{1, 2\}$$

$$Z_4^* = \{1, 3\}$$

$$Z_5^* = \{1, 2, 3, 4\}$$

$$Z_6^* = \{1, 5\}$$

Se n è **primo** il gruppo contiene tutti gli elementi da 1 a $n-1$:

$$Z_n^* = \{1, 2, \dots, p-1\}$$

(Z_n^*, \cdot) è un gruppo **commutativo**, $\forall n$

Elevamento a potenza nei gruppi:

(G, \cdot) è un gruppo. $\forall x \in G \ \forall m \in \mathbb{N}$:

$$x^m = x \cdot \dots \cdot x \quad m \text{ volte}$$

Le proprietà delle potenze sono preservate anche nei gruppi:

$$x^1 = x$$

$$x^{a+b} = x^a \cdot x^b$$

$$(x^a)^b = x^{a \cdot b}$$

Se il gruppo è **commutativo**:

$$(x \cdot y)^m = (x \cdot y) \cdot (x \cdot y) \dots (x \cdot y) = x^m \cdot y^m$$

Gruppi ciclici e generatori:

Ordine di un elemento di un gruppo:

L'**ordine** $|x|$ di un elemento x di un gruppo (G, \cdot) è **il più piccolo $i > 0$ tale che $x^i = 1$**

Esempio:

$$(Z_{15}^*, \cdot)$$

$$Z_{15}^* = \{1, 2, 4, 7, 8, 11, 13, 14\}$$

$$|2| = 4$$

$$2^1 = 2$$

$$2^2 = 4$$

$$2^3 = 8$$

$$2^4 = 16 \bmod_{15} 1 \rightarrow |2| = 4$$

Generatore:

Un generatore g di un gruppo (G, \cdot) è un elemento del gruppo $g \in G : |g| = |G|$

L'ordine del gruppo dell'elemento è uguale all'ordine del gruppo (numero di elementi del gruppo)

Esempio:

$$(Z_7^*, \cdot)$$

$$Z_7^* = \{1, 2, 3, 4, 5, 6\}$$

$$|Z_7^*| = 6 \quad 6 \text{ elementi nel gruppo}$$

3 è un **generatore** di Z_7^*

$$3^1 = 3$$

$$3^2 = 2$$

$$3^3 = 6$$

$$3^4 = 4$$

$$3^5 = 5$$

$$3^6 = 1 \leftarrow$$

$$3^6 = 1 \rightarrow |3| = 6$$

$$|Z_7^*| = 6$$

$$|G| = |g|$$

Gruppo ciclico :

Un gruppo (G, \cdot) è **ciclico** se esiste un **generatore** nel gruppo.

Elevando a potenza il generatore del gruppo, visitiamo (otteniamo) tutti gli elementi del gruppo!

Teorema:

Se p è **primo**, allora (Z_p^*, \cdot) è **ciclico**

One-way trapdoor functions:

L'idea è quella di avere delle funzioni che siano **facili da calcolare** in un verso, ma **difficili** da invertire *senza conoscere una chiave segreta*.

Sfruttando un'analogia del mondo reale, possiamo immaginare un lucchetto: è facile chiuderlo anche senza conoscere la combinazione, ma è davvero difficile aprirlo senza conoscerla.

Allo stesso modo, immaginiamo una **cassetta delle lettere**: è facile spedire una lettera a B (basta conoscere il suo indirizzo, **chiave pubblica**), ma è difficile aprire la cassetta senza conoscere la **chiave privata** di B .

Esponenziale discreto:

Utilizzando l'algoritmo ingenuo per il calcolo dell'elevamento a potenza x^m , la dimensione dell'input è la rappresentazione in binaria di x ed m , ad esempio n bit, mentre la **complessità computazionale** è $O(2^n) \rightarrow$ inefficiente.

Esistono però algoritmi noti **polinomiali**:

$$x^m = \begin{cases} 1 & \text{if } m=0 \\ x^{2(m/2)} = (x^{m/2})^2 & \text{if } m \bmod 2 = 0 \\ x^{2((m-1)/2)+1} = x(x^{(m-1)/2})^2 & \text{if } m \bmod 2 = 1 \end{cases}$$

```
def expSquareAndMultiply(x, m):
    if m==0 : return 1
    else :
        z = expSquareAndMultiply(x, m/2)
        return z*z if m%2 == 0 else z*z*x
```

Logaritmo discreto:

Operazione **inversa** dell'elevamento a potenza.

Discreto perchè lavora su un gruppo ciclico finito:

(G, \cdot) è un **gruppo ciclico** con un generatore g , allora:

$$\forall x \in G : \exists! m \in 1..|G| : g^m = x$$

Per ogni elemento del gruppo, esiste un **unico** esponente tale che elevando il generatore g a m , ottengo l'elemento del gruppo.

Il **logaritmo discreto** (di base g) è quell'**unico** m tale che $g^m = x$

$$\log_g x = m$$

Esercizio:

$$\begin{aligned} Z_{11}^* &= \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\} \quad 11 \in \text{prime} \\ \log_2 9 & \end{aligned}$$

$$Z_{11}^* = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\} \quad 11 \in \text{prime}$$

$$2^1 = 2$$

$$2^2 = 4$$

$$2^3 = 8$$

$$2^4 = 16 \bmod_{11} = 5$$

$$2^5 = 32 \bmod_{11} = 10$$

$$2^6 = 64 \bmod_{11} = 9 \leftarrow$$

$$2^7 = 128 \bmod_{11} = 7$$

$$2^8 = 256 \bmod_{11} = 3$$

$$2^9 = 512 \bmod_{11} = 6$$

$$2^{10} = 1024 \bmod_{11} = 1$$

$$2^6 = 9 \rightarrow \log_2 9 = 6 \text{?}$$

Problema di calcolo del logaritmo discreto:

Informalmente, il **problema del logaritmo discreto** in un gruppo ciclico (G, \cdot) con generatore g è:

Dato un $x \in G$ random, calcola il $\log_g x$

Diciamo che questo problema relativo al gruppo (G, \cdot) e generatore g è difficile (**hard**) se **non esiste** un algoritmo $\in PPTIME$ che sia in grado di risolverlo tralasciando le probabilità negligibili.

Dal momento che tutti i gruppi ciclici finiti sono isomorfi a $(Z_n, +)$, la difficoltà del problema dipende dalla rappresentazione di G , ovvero dall'algoritmo che genera il gruppo.

Per usare questo problema, bisogna trasformarlo da problema di calcolo a un **problema di decisione**.

Questo è possibile grazie alla funzione di **Diffie-Hellman**.

Funzione di Diffie-Hellman:

Dato un gruppo ciclico (G, \cdot) e un generatore g . Definiamo la funzione di Diffie-Hellman DH per ogni coppia $(h_1, h_2) \in G$

$$DH_g(h_1, h_2) = g^{x \cdot y} \quad x = \log_g h_1 \quad y = \log_g h_2$$

La funzione prende in input tre argomenti: il generatore g , h_1 , h_2 e restituisce il generatore g elevato al prodotto dei logaritmi discreti di h_1 , h_2

Problema di decisione di Diffie-Hellman:

(G, \cdot) gruppo ciclico con generatore g

Dati $h_1, h_2 \in G$ random, discrimina $DH_g(h_1, h_2)$ da un $x \in G$ random

(Ricorda molto gli esperimenti di indistinguishability con i PRG)

Esempio:

In $Z_{9451485124}^*$ qual è $DH_2(454561563, 23587423329)$?

456158942 è la risposta oppure è soltanto un elemento a caso del gruppo?

Assunzione:

Ci sono rappresentazioni di G per le quali il problema è **hard**.

Diffie-Hellman key exchange (1976):

n parametro di sicurezza

$Gen(1^n)$ algoritmo $PPTIME$ che genera un gruppo ciclico (G, \cdot) di ordine q ($|q| = n$) e con generatore g

0. $A :$ $(G, q, g) \leftarrow Gen(1^n)$ $a \leftarrow \{1..q - 1\}$ A genera un gruppo ciclico e un numero random $a < q$
1. $A \rightarrow B :$ (G, q, g) g^a A invia a B il gruppo e il generatore elevato al numero random a
2. $B \rightarrow A :$ g^b B sceglie un $b < q$ e invia ad A : g^b

Il **segreto** (es: la chiave da scambiare) è $K_{AB} = g^{a \cdot b}$

B può conoscere il segreto facendo l'elevamento a potenza di quello che A gli ha inviato con il proprio b : $(g^a)^b = g^{a \cdot b}$
 A può conoscere il segreto facendo l'elevamento a potenza di quello che B gli ha inviato con il proprio a : $(g^b)^a = g^{b \cdot a}$

$$g^{a \cdot b} = g^{b \cdot a}$$

A non ha mai condiviso il suo a segreto

B non ha mai condiviso il suo b segreto

Entrambi hanno inviato g elevato il proprio segreto!

M leggendo sul canale g^a e g^b non può ricostruire il segreto $g^{a \cdot b}$, infatti conoscerebbe soltanto $g^{a+b} \neq g^{a \cdot b}$

Esperimento Key Exchange:

π è il protocollo di key exchange

M è l'attaccante

Il protocollo produce in **output** il log L di tutti i messaggi scambiati e la chiave K_{AB}

$KE_{\pi,M}(n)$:

1. A, B eseguono il protocollo $\pi(n)$
2. $A, B \rightarrow M : L, K' \quad K' = (b == 0) ? K_{AB} : \{0,1\}^n$
3. $M : b_m$

L'esperimento ha successo $KE_{\pi,M}(n) = 1$ se $b_m = b$

A e B eseguono il protocollo e inviano all'attaccante M il log dei messaggi scambiati e un K' influenzato dall'esito di un bit b random (come i precedenti esperimenti).

Se $b = 0$, A e B inviano a M la chiave, altrimenti una sequenza random.

M è chiamato a distinguere la chiave da una sequenza random: ovvero, come al solito deve indovinare il bit b .

Sicurezza di un protocollo di Key Exchange in presenza di eavesdroppers:

Uno schema di key exchange è sicuro in presenza di **eavesdroppers** se:

$$\forall M \in PPTIME : \Pr(KE_{\pi,M}(n) = 1) \leq 1/2 + negl(n)$$

L'attaccante è passivo e non attivo: non può interagire con A e B (irrealistico)!

Teorema:

Se il problema di decisione di Diffie-Hellman su (G, \cdot) è **hard**, allora il protocollo di key exchange di Diffie-Hellman è **sicuro in presenza di eavesdroppers**.

Attaccante attivo: attacco man in the middle a Diffie-Hellman:

Diffie-Hellman:

0. $A : (G, q, g) \leftarrow Gen(1^n) \quad a \leftarrow \{1..q-1\}$
1. $A \rightarrow B : (G, q, g) \quad g^a$
2. $B \rightarrow A : g^b$

Attacco MiM:

$I(A)$ l'attaccante sta impersonando A (A non riceve i messaggi intercettati)

$I(B)$ l'attaccante sta impersonando B (B non riceve i messaggi intercettati)

1. $A \rightarrow I(B) : (G, q, g) \quad g^a$
1. $I(A) \rightarrow B : g^{a'}$
2. $B \rightarrow I(A) : g^b$
2. $I(B) \rightarrow A : g^{b'}$

Quello che M sta facendo è **creare due chiavi fasulle** che condivide con A e B senza che questi se ne accorgano (a' e b' li sceglie proprio M).

Quando A e B devono comunicare, M si mette nel mezzo e utilizza le chiavi fasulle:

$$K_{AM} = g^{b' \cdot a} \text{ conosciuta da } A \text{ e } M$$

$$K_{BM} = g^{a' \cdot b} \text{ conosciuta da } B \text{ e } M$$

Encryption a chiave pubblica:

Piccolo teorema di Fermat:

Sia (G, \cdot) un gruppo finito.

$$\forall x \in G : x^{|G|} = 1$$

Esempio:

$$Z_6^* = \{1, 5\}$$

$$|Z_6^*| = 2$$

$$1^2 = 1$$

$$5^2 = 25 \bmod_6 = 1$$

Corollario:

Sia (G, \cdot) un gruppo finito.

$$\forall x \in G, \forall i \in \mathbb{N} : x^i = x^{i \bmod |G|}$$

Questo corollario ci permette di calcolare in maniera più veloce x^i

Teorema delle permutazioni:

Sia (G, \cdot) un gruppo di ordine $m > 1$.

Sia $f_k \in G \rightarrow G$ tale che $f_k(x) = x^k$

Se $e \perp m$, allora f_e è una permutazione (ovvero, f_e è **biettiva**)

Dimostrazione:

Per il **lemma 4**: $e \perp m \implies \exists d : e \cdot d =_m 1$

Proviamo che f_d è l'**inversa** di f_e :

$$f_d(f_e(x)) = f_d(x^e) = (x^e)^d = x^{e \cdot d} = x^{e \cdot d \bmod_m} = x^1 = x$$

$x^{e \cdot d} = x^{e \cdot d \bmod_m}$ per il corollario del piccolo teorema di Fermat

$x^{e \cdot d \bmod_m} = x^1$ per il lemma 4 visto che $e \perp d$ e si sta facendo il prodotto in modulo m

Nel contesto della **crittografia a chiave pubblica**, pensiamo a:

e come la chiave pubblica per effettuare l'encryption (tutti possono effettuare l'encryption con una chiave pubblica)

d come la chiave privata per effettuare la decryption (solo chi possiede la chiave privata può effettuare la decryption)

d e e devono essere l'uno l'inverso dell'altro

Problema:

Come possiamo garantire che condividendo pubblicamente e qualcuno non sia in grado di ottenere la chiave privata d ?

Funzione di Eulero:

La funzione di Eulero $\Phi \in \mathbb{N} \rightarrow \mathbb{N}$

$$\Phi(N) = |Z_N^*|$$

restituisce la cardinalità (l'ordine) del gruppo $Z_N^* = \{x \in \mathbb{Z} \mid x \perp N\}$

Teorema:

Se N è primo, $\Phi(N) = N - 1$

Esempio:

$$Z_7^* = \{1, 2, 3, 4, 5, 6\}$$

$$|Z_7^*| = \Phi(7) = 7 - 1 = 6$$

Teorema:

Se N è il prodotto di due numeri primi p e q ($N = p \cdot q$) e $p \neq q$, allora $\Phi(N) = (p - 1)(q - 1)$

Esempio:

$$N = 10 = 5 \cdot 2$$

$$Z_{10}^* = \{1, 3, 7, 9\}$$

$$|Z_{10}^*| = \Phi(10) = (5 - 1)(2 - 1) = 4$$

Questo teorema è importante perché ci permette di calcolare $\Phi(N)$ in maniera efficiente conoscendo p e q

Teorema di Eulero:

$$x \perp N \implies x^{\Phi(N)} \text{ mod}_N = 1$$

Dimostrazione:

Sappiamo che (Z_N^*, \cdot) è un gruppo commutativo di ordine $\Phi(N)$.

$$\text{Per il piccolo teorema di Fermat in } Z_N^* \quad x^{\Phi(N)} = 1$$

$$\text{Quindi, in } Z: \quad x^{\Phi(N)} \text{ mod}_N = 1$$

Permutazioni in (Z_N^*, \cdot) :

Sappiamo già che in ogni gruppo (G, \cdot) di ordine $m > 1$:

$$e \perp m \implies f_e(x) = x^e \text{ è una permutazione}$$

$$f_e^{-1} = f_d \quad d = e^{-1} \text{ mod}_m$$

Il gruppo (Z_N^*, \cdot) ha ordine $\Phi(N)$, quindi:

$$e \perp \Phi(N) \implies f_e(x) = x^e \text{ è una permutazione}$$

$$f_e^{-1} = f_d \quad d = e^{-1} \text{ mod}_{\Phi(N)}$$

Per invertire f_e è necessario conoscere $\Phi(N)$!

L'obiettivo è scegliere un N in modo che il partecipante onesto che vuole creare la sua coppia di chiavi sia in grado di calcolare efficientemente $\Phi(N)$, ma l'attaccante no!

L'attaccante può cercare di **fattorizzare** N per conoscere p e q e quindi $\Phi(N)$.

Non sono noti algoritmi efficienti per fattorizzare un numero primo

Notazione:

$\|n\|$ = dimensione in bit di n

Cifrario RSA:

$GenRSA(1^n)$:

Algoritmo per generare la chiave pubblica e la chiave privata (con parametro di sicurezza n)

1. genera due numeri primi p e q tali che $p \neq q$ e $\|p\| = \|q\| = n$
2. calcola $N = p \cdot q$
4. calcola $\Phi(N) = (p - 1) \cdot (q - 1)$
5. scegli un e random tale che $1 < e < \Phi(N)$ and $e \perp \Phi(N)$
6. calcola $d = e^{-1} \text{ mod}_{\Phi(N)}$
7. return (N, e, d)

Già dagli anni '70 (del Novecento) si conoscevano algoritmi **probabilistici** per determinare se un numero fosse primo (con una certa tolleranza).

Nel 2002 è stato scoperto un algoritmo **deterministico** polinomiale per decidere se il numero è primo o no (AKS algorithm).

Quindi per generare p e q si possono generare numeri pseudorandom con i generatori che abbiamo visto e controllare se la sequenza pseudorandom è prima oppure no con l'AKS.

Se non lo è incremento, e così via.

$RSA = (Gen, Enc, Dec)$

Gen : con input n genera i parametri: esegue $GenRSA(1^n)$ per ottenere (N, e, d)

La chiave pubblica è $p_k = (N, e)$

La chiave privata è $p_s = (N, d)$

Enc :

$$E_{pk}(x) = x^e \text{ mod}_N$$

Dec :

$$D_{sk}(x) = y^d \text{ mod}_N$$

Il problema della fattorizzazione:

Dati due interi x e y , calcolare il prodotto $x \cdot y$ è molto facile.

Il problema inverso, il problema della **fattorizzazione** è **considerato** difficile: non sono noti al momento algoritmi efficienti polinomiali.

Dato N , trova $x, y > 1$ tali che $x \cdot y = N$

N però deve essere sufficientemente grande per rendere il problema difficile.

Dato N tale che $e \perp \Phi(N)$ e un $y \in Z_n^*$

Trova $x \in Z_n^*$ tale che $x^e \text{ mod}_N = y$

Ovvero:

Trova un plaintext x tale che $E_{pk}(x) = y$

Se p e q sono numeri primi sufficientemente grandi, RSA è **considerato** sicuro.

Firma digitale:

Digital signature schemes:

A vuole mandare a B un messaggio **autenticato** x .

A firma il messaggio x con la sua **chiave privata**: $y = \text{sig}_{sk_A}(x)$ e ottiene la **firma** y

B verifica la firma y con la **chiave pubblica** di A $\text{ver}_{pk_A(x,y)}$

Un **signature scheme** è una tripla di algoritmi PPTIME

- gen : prende in input un parametro di sicurezza 1^n e restituisce la coppia di chiavi (pk, sk) dove pk e sk hanno almeno n bit
- sig : prende in input sk e un messaggio $x \in \{0,1\}^*$ e restituisce una **firma** $\text{sig}_{sk}(x)$
- ver : prende in input pk un messaggio x e una firma y e restituisce in output un **booleano** $\text{ver}_{pk}(x, y)$

$$\text{ver}_{pk}(x, y) = (y = \text{sig}_{sk}(x)) ? \text{true} : \text{false}$$

RSA signature scheme:

Soltanto il proprietario della chiave privata può firmare un messaggio:

$$\text{sig}_{sk}(x) = D_{sk}(x) \quad \text{decryption per firmare}$$

Chiunque conosca la chiave pubblica può verificare la firma:

$$\text{ver}_{pk}(x, y) = (E_{pk}(y) = x) ? \text{true} : \text{false}$$

Condizione di correttezza:

$$y = \text{sig}_{pk}(x) \implies E_{pk}(y) = E_{pk}(D_{pk}(x)) = x \implies \text{ver}_{pk}(x, y) = \text{true}$$

Forgeries:

A ha generato un key-pair (sk, pk) e ha firmato un insieme X di messaggi $X = \{x_1, x_2, \dots\}$

Una **forgery** è una **coppia** (x, y) tale che:

- $y = \text{sig}_{sk}(x)$
- $x \notin X \quad A \text{ non ha mai firmato } x$

Questa definizione di forgery è esattamente la controparte a chiave pubblica delle forgery nel contesto dei MAC.
In base all'avversario e al suo obiettivo, possono esistere diversi tipi di forgeries.

Forgeries in base alla conoscenza dell'avversario:

- key-only attacks: M conosce soltanto la chiave pubblica pk
- known message attacks: M conosce un insieme di messaggi con rispettive firme create da A
- chosen message attacks: M può scegliere un insieme arbitrario di messaggi e chiedere a A di firmarli

Forgeries in base all'obiettivo dell'avversario:

- total break: M vuole ottenere sk in modo da firmare poi ogni successivo messaggio
- selective forgery: M ottiene la forgery di un messaggio che vuole
- existential forgery: M riesce a creare una forgery ma di un messaggio che non ha desideratamente scelto

Attacchi al RSA signature scheme:

Key only attack:

M conosce solo la chiave pubblica

M sceglie il valore della firma (attenzione: non sceglie il messaggio)

M calcola $x = E_{pk}(y)$ è come se M stesse verificando un y virtualmente ottenuto (l'ha scelto lui)

$$E_{pk}(y) = x \implies ver_{pk}(x, y) = true$$

Questo **key-only attack** ha prodotto una **existential forgery**.

È un attacco **non distruttivo** visto che M non ha scelto il messaggio da firmare, ma ha soltanto **verificato** una firma y fasulla con la chiave pubblica, ottenendo un valore *spazzatura*.

Known message attack:

1. M knows two signed messages $(x_1, y_1), (x_2, y_2)$, both signed by A
2. M computes $x = x_1 x_2$ and $y = y_1 y_2$
3. (x, y) is a forgery:

$$\begin{aligned} x_1 &= E_{pk}(y_1) \text{ and } x_2 = E_{pk}(y_2) & \Rightarrow x_1 = y_1^e \pmod{N} \text{ and } x_2 = y_2^e \pmod{N} \\ \Rightarrow x_1 x_2 \pmod{N} &= y_1^e y_2^e \pmod{N} \Rightarrow x_1 x_2 \pmod{N} = (y_1 y_2)^e \pmod{N} \\ \Rightarrow x \pmod{N} &= y^e \pmod{N} \quad \Rightarrow x = E_{pk}(y) \quad \Rightarrow y = \text{sig}_{sk}(x) \end{aligned}$$

This is an **existential forgery** with a **known message attack**.

Chosen message attack:

1. M chooses a message x , and factors it as $x = x_1 x_2$ (x_1, x_2 are "harmless")
2. M asks A to sign x_1 and x_2 , obtaining the signatures y_1 and y_2
3. $(x, y_1 y_2)$ is a forgery:

$$\begin{aligned} x_1 &= E_{pk}(y_1) \text{ and } x_2 = E_{pk}(y_2) & \Rightarrow x_1 = y_1^e \pmod{N} \text{ and } x_2 = y_2^e \pmod{N} \\ \Rightarrow x_1 x_2 \pmod{N} &= y_1^e y_2^e \pmod{N} \Rightarrow x_1 x_2 \pmod{N} = (y_1 y_2)^e \pmod{N} \\ \Rightarrow x \pmod{N} &= (y_1 y_2)^e \pmod{N} \quad \Rightarrow x = E_{pk}(y_1 y_2) \quad \Rightarrow y_1 y_2 = \text{sig}_{sk}(x) \end{aligned}$$

This is an **selective forgery** with a **chosen message attack**.

Soluzione: utilizzare le funzioni hash

Non si firma più il messaggio, ma l'**hash del messaggio** avendo così vantaggi dal punto di vista della sicurezza e dell'efficienza.

Firmare un messaggio: $\text{sig}_A^H(x) = \text{sig}_A(H(x))$

Verificare una firma. $\text{ver}_A^H(x, y) = \text{ver}_A(H(x), y)$

Correttezza: $\text{sig}_A^H(x) = \text{sig}_A(H(x)) \implies \text{ver}_A^H(x, y) = \text{ver}_A(H(x), y) = \text{true}$

Se H è **sicura**, allora gli **attacchi agli schemi** di firma RSA **non sono più praticabili**.

Esempio di attacco con H non sicura:

H non è second-preimage resistant: M è in grado di trovare un x' con lo stesso hash di un dato x

1. M sceglie x
2. H non è second-preimage resistant $\implies M$ calcola $x' \neq x$. $H(x') = H(x)$
3. M chiede a A di firmare x' perché sa che $\text{sig}_{sk}^H(x') = \text{sig}_{sk}^H(x)$
4. (x, y) è una forgery

$\text{ver}_{pk}^H(x, y) = \text{ver}_{pk}(H(x), y) = \text{ver}_{pk}(H(x'), y) = \text{true}$

Questo è un *attacco pericoloso*: **selective forgery** con **chosen message attack**

Verificare l'identità con le firme digitali:

Il verifier B invia una random challenge r al claimant A

B accetta se la verifica di r e la firma è valida

1. $B \rightarrow A : r$
2. $A \rightarrow B : \text{sig}_A(r)$

Questo protocollo è **sicuro**.

Station-to-station key exchange:

Il protocollo di scambio di chiavi Diffie-Hellman è sicuro quando l'attaccante è passivo, però quando M diventa attivo, il protocollo è vulnerabile agli attacchi di tipo Man in the Middle.

Con la **firma digitale** si riesce a scongiurare l'attacco:

(omettiamo, dando per scontati, la generazione e l'invio del gruppo G a B)

1. $A \rightarrow B : g^a$
2. $B \rightarrow A : g^b, \text{sig}_B(A, g^b, g^a)$
3. $A \rightarrow B : g^a, \text{sig}_A(B, g^a, g^b)$

A e B questa volta accettano soltanto se la firma è valida

M può tentare di inviare i suoi $g^{a'}$ e $g^{b'}$ fasulli, ma non è in grado di firmare i messaggi