ALGORITMI DI CRITTOGRAFIA

La **crittografia** studia metodi per memorizzare, elaborare e trasmettere informazioni in maniera sicura in presenza di agenti ostili.

- Memorizzare, elaborare e trasmettere informazioni → sono operazioni classiche che operano con i dati
- Maniera sicura → il concetto di sicurezza dipende dal *requisito di sicurezza che si sta ricercando*
- Agenti ostili → il canale di comunicazione è sempre supposto essere *insicuro*

Un esempio applicativo molto interessante di uso congiunto di crittografia asimetrica e simmetrica: **TLS** (Transport Layer Security), diviso in due fasi principali:

1. Handshaking → Si stabilisce una chiave pubblica condivisa
2. Reconrd Layer → I dati vengono trasmessi secondo un algoritmo di crittografia simmetrica

TERMINOLOGIA COMUNE

Crittografia simmetrica (il cifrario *garantisce* ***confidenzialità***):

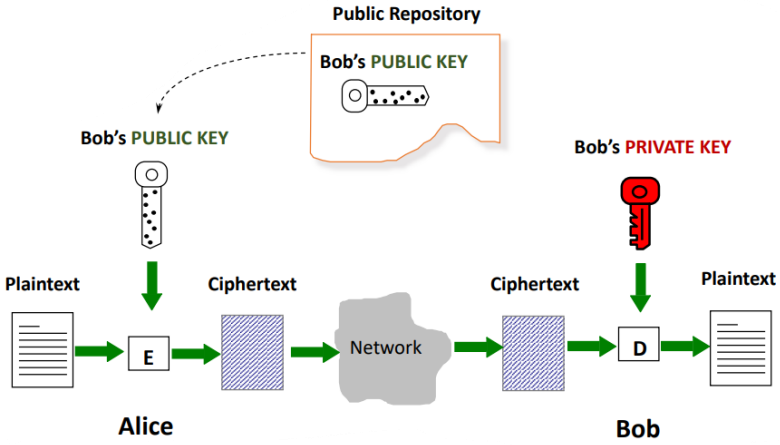
- k→ chiave segreta
- m→ testo in chiaro (messaggio)
- c→ testo cifrato (ciphertext)
- E→ algoritmo di cifratura (CIPHER)
- D→algoritmo di decifratura(CIPHER)

NOTA: gli algoritmi sono tutti **pubblici**, dunque è consigliabile non usare mai algoritmi di cifratura proprietari. La sicurezza dell'algoritmo risiede nella sua implementazione.

I cifrari simmetrici sono soggetti ad una distinzione dipendentemente dall'uso che si fa della chiave.

1. **Single-use key (one-time key)** →la chiave può essere usata solo una volta per la cifratura;
2. **Multi-use key (many-time key)** → la stessa chiave può cifrare più messaggi. Ad ogni modo, dopo un certo numero di utilizzi anche la chiave many-time deve essere cambiata. Inoltre, sono algoritmi che richiedono *maggior impegno nella loro formulazione* rispetto a quelli one-time key.

Crittografia asimmetrica (il cifrario *garantisce* ***confidenzialità***):



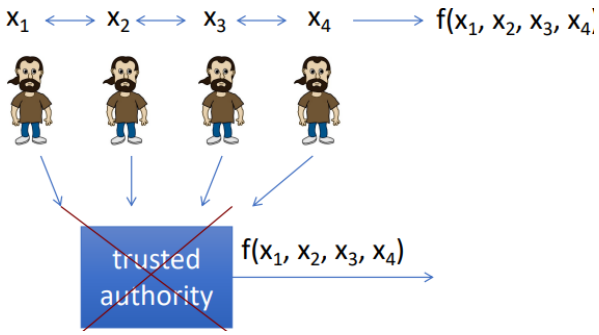
Nella realtà, la chiave pubblica non si trova in una repository pubblica.
La crittografia asimmetrica è spesso usata per effettuare la **firma digitale**.

ALCUNE APPLICAZIONI DELLA CRITTOGRAFIA

- Rendere le comunicazioni sicure e anonime (es. TOR);
- Protezione dei file su disco → protegge da **eavesdropping** (per la riservatezza) e da **tampering** (per l'integrità);
- Firma digitale (diversa per ogni documento digitale firmato);
- Valute digitali;
- Aste o elezioni online (sono un caso specifico del problema di **Secure Multi-Party Computation**): il server deve essere in grado di determinare il vincitore a partire dai voti cifrati e senza conoscere l'identità degli elettori;

SECURE MULTI-PARTY COMPUTATION

Si supponga di avere un certo numero di entità che devono inviare dei dati (x1, …, x4) ad un'autorità centrale, la quale ha come obiettivo calcolare f(x1),…, f(x4) senza ovviamente conoscere i singoli dati ma lavorando con la loro cifratura c(x1),…, c(x4).
Vi è un teorema che afferma che ciò può essere fatto in maniera **distribuita** senza fare uso di un server centrale.



STEP DI FORMULAZIONE DI TECNICHE DI CRITTOGRAFIA

1. Definire il modello di minaccia (*threat model*): obiettivo da proteggere e potere dell'avversario;
2. Proporre una costruzione dell'algoritmo;
3. Dimostrare che l'algoritmo sia in grado di perseguire l'obiettivo proposto;

BREVE STORIA DELLA CRITTOGRAFIA

Si parte dalla vasta gamma di **cifrari simmetrici** proposti nei secoli, tutti altamente inefficienti nei confronti di qualche proprietà.

- **CIFRARIO A SOSTITUZIONE (Substitution cipher)** → ogni lettera del testo in chiaro *viene sostituita* con un'altra lettera dello stesso alfabeto secondo una certa *chiave* (insieme di regole di sostituzione). Bisogna assicurarsi che la chiave non assegni mai la stessa lettera a più di un elemento nel testo in chiaro (la funzione di associazione deve essere **iniettiva**).
  - **CIFRARIO DI CESARE (Caesar cipher, caso particolare di cifrario a sostituzione)** → ogni lettera del testo in chiaro *viene sostituita* con la lettera dello stesso alfabeto che sta a *k passi* da quella in esame, dove k è la chiave e vale solitamente 3.

- **VIGENÈRE CIPHER (Cifrario polialfabetico)** → Viene usata una chiave di lunghezza *k*, inferiore alla lunghezza del messaggio. La chiave viene ripetuta finché non copre il messaggio per intero. Ogni carattere del testo, visto come un numero, viene sommato al carattere corrispondente della chiave +mod26.
  Per decifrare il messaggio, basta sottrarre il carattere del testo cifrato alla chiave +mod26.

  NOTA: a differenza dei cifrari a sostituzione, una lettera non viene cifrata sempre con la stessa lettera →occorrenze diverse della stessa lettera potrebbero essere cifrate in maniera diversa.

- **MACCHINE A ROTORI**→ dette "a rotori" perché c'era un disco che ruotando faceva le associazioni per ogni lettera, cambiando la propria posizione ad ogni inserimento. Per la decifratura, il destinatario doveva avere il disco nella stessa posizione. Tra queste rientrano la *macchina a singolo rotore (**Macchina di Hebern**)* ed ***Enigma***, a 3-5 rotori. Per queste macchine, **aumentare i rotori equivale ad aumentare il numero di chiavi**.

- **DATA ENCRYPTION STANDARD (DES)** → Cifrario a blocchi non più utilizzato perché insicuro (**chiave troppo corta**), riceve in input blocchi da 64 bit e restituisce in output il testo cifrato usando una chiave a 56 bit. Ad oggi si usa l'**Advanced Encryption Standard (AES)** e usa chiavi a 128, 256, 512 bit.

DEBOLEZZE DEI CIFRARI SIMMETRICI

1) **ATTACCO A FORZA BRUTA**
   Lo *spazio delle chiavi* possibili, supponendo di partire da un alfabeto composto da 26 lettere, è pari a 26! cioè la permutazione delle lettere, rendendo semplice un *attacco a forza bruta* per decifrarlo. Questo attacco fa sì che un *attaccante passivo* entri in possesso di un testo cifrato e ha come obiettivo decifrare il testo senza avere la chiave.

È un algoritmo che riceve in input il testo cifrato e restituisce in output il testo in chiaro. La complessità temporale nel caso peggiore è **O(n)**, dove *n* è il numero di chiavi (**lento se *n* è molto grande**). Un grande vantaggio dell'algoritmo è la sua **elevata generalità**: può essere usato per qualunque cifrario.

2) **ATTACCO PER MEZZO DELL'ANALISI DELLA FREQUENZA**
L'*analisi della lingua e della frequenza* con cui compaiono le lettere in un dato alfabeto rende ancora più semplice decifrare il testo senza la chiave. L'algoritmo consiste nell'ottenere una lista di lettere più frequenti in un dato alfabeto e sostituirle con le lettere che più frequentemente si presentano nel testo cifrato. Successivamente, si procede a calcolare la frequenza di *coppie di lettere* e associarle alle coppie più frequenti nel testo cifrato. Si procede poi con le triple e via dicendo. Questo meccanismo è **più veloce** dell'attacco a forza bruta, è dunque *a causa di questo algoritmo che i cifrari a sostituzione sono considerati insicuri.*

3) **ATTACCO PER IL CIFRARIO POLIALFABETICO**
Per decifrare un cifrario polialfabetico, l'attaccante prova tutte le lunghezze della chiave: parte da una chiave k=2, suddividendo il testo cifrato in blocchi da k=2. Prende il primo carattere di ogni blocco mantenendo i doppioni e li mette insieme. Ora conta il numero di occorrenze di ogni carattere. Se la lunghezza della chiave supposta è corretta, per queste lettere il cifrario è un cifrario a sostituzione. Allora, la lettera più frequente supponiamo sia la lettera più frequente di quella lingua. A questo punto per ottenere la lettera della chiave basta applicare:

$$c[i] = m[i] + k[i] \rightarrow \quad k[i] = c[i] - m[i]$$

Per le altre lettere della chiave, si applica lo stesso iter ma sulla seconda lettera di ogni blocco. Poi sulla terza, e così via…. Se k=2 non è la lunghezza corretta, si passa a k=3, ecc….

NOTA: una condizione **necessaria ma non sufficiente** per rendere un cifrario simmetrico sicuro è che **abbia un numero elevato di chiavi**. In generale, sia *n* la lunghezza in bit di una chiave, le possibili chiavi sono 2^n.

CENNI DI PROBABILITA' DISCRETA

Col termine di **spazio campionario U** (o universo) è l'insieme dei risultati di un esperimento aleatorio.
Una **distribuzione di probabilità P** è una funzione che associa ad ogni elemento di U un valore compreso tra 0 e 1. In particolare, è $P: U \rightarrow [0,1] \quad t.c. \quad \sum_{x \in U} P(x) = 1$

NOTA: $\boldsymbol{U = \{0,1\}^n}$, indica *l'insieme di tutte le sequenze di n bit*. Ad esempio: U = $\{0,1\}^2$= {00,01,10,11}

Esistono diverse tipologie di distribuzioni, tra le più importanti vi è la **distribuzione uniforme**, tale che per ogni elemento $x \in U, P(x) = \frac{1}{|U|}$.
Dato un universo U e una distribuzione di probabilità P su U, un **evento** è un sottoinsieme A di U: $A \subseteq U$.
La probabilità di A è $P(A) = \sum_{x \in A} P(x)$. Ovviamente vale che P(U)=1.

Dati due eventi A_1 e A_2, $\boldsymbol{A_1 \cup A_2}$ è ancora un evento. Inoltre, vale che:

- $P(A_1 \cup A_2) = P(A_1) + P(A_2) - P(A_1 \cap A_2)$
- $Se\ A_1 \cap A_2 = \emptyset \rightarrow \ P(A_1 \cup A_2) = P(A_1) + P(A_2)$
- $P(A_1 \cup A_2) \leq P(A_1) + P(A_2)$

Example (Rolling a dice):
U = { 1, 2, 3, 4, 5, 6 }
Uniform distribution P over U: P(1) = P(2) = P(3) = P(4) = P(5) = P(6) = 1/6

Random variable **X : U ⟶ { "even", "odd" }**
X(2) = X(4) = X(6) = "even"
X(1) = X(3) = X(5) = "odd"

Pr[X="even"] = 1/2 , Pr[X="odd"] = 1/2

Una **variabile aleatoria X** è una funzione nella forma $X: U \rightarrow V$.
X induce una distribuzione di probabilità su V.
Si scrive $\boldsymbol{r \leftarrow S}$ per denotare una **variabile aleatoria uniforme su S**.

ALGORITMO DETERMINISTICO	ALGORITMO RANDOMIZZATO
È un algoritmo che, se richiamato più volte sullo stesso input, restituisce sempre lo stesso output.	È un algoritmo che, se richiamato più volte sullo stesso input, potrebbe restituire output diversi. Molti algoritmi di cifratura sono randomizzati.

Due eventi A e B sono detti **indipendenti** se $P(A \cap B) = P(A) \cdot P(B)$
Due variabili aleatorie X, Y sono dette **indipendenti** se $\forall\, A, B \in V, P(A\ and\ B) = P(A) \cdot P(B)$

TEOREMA SULLO XOR
Siano:

- X una variabile aleatoria su $\{0,1\}^n$ con una *distribuzione uniforme;*
- Y una variabile aleatoria su $\{0,1\}^n$ con una *distribuzione arbitraria;*
- X e Y indipendenti;
- $Z = X \oplus Y$ è una variabile aleatoria con una *distribuzione uniforme* su $\{0,1\}^n$

Y	Pr
0	p_0
1	p_1

X	Pr
0	1/2
1	1/2

X	Y	Pr
0	0	$p_0/2$
0	1	$p_1/2$
1	0	$p_0/2$
1	1	$p_1/2$

Dim. Per n=1, voglio dimostrare che P(Z) valga ½ sia nel caso Z=0, che Z=1:

$$P(Z=0) = P(00\ or\ 11) = P(00) + P(11) = \frac{p_0}{2} + \frac{p_1}{2} = \frac{1}{2}$$
$$P(Z=1) = P(01\ or\ 10) = P(01) + P(10) = \frac{p_0}{2} + \frac{p_1}{2} = \frac{1}{2}$$

THE BIRTHDAY PARADOX
Siano $r_{1,\dots,}r_n \in U$ delle variabili aleatorie indipendenti *identicamente distribuite* (hanno la stessa distribuzione), allora

$$se\ n = 1{,}2 \times |U|^{\frac{1}{2}} \qquad allora \qquad P\big(\exists\, i \neq j :\ r_i = r_j\big) \geq \frac{1}{2}$$

Esempio: considerando $U = \{1, 2, \dots, 366\}$, quando $n = 1{,}2\ \times \sqrt{366} \approx 23$, la probabilità che due persone abbiano lo stesso compleanno è superiore o uguale a ½.

All'aumentare del numero di variabili aleatorie coinvolte, la probabilità aumenta fino ad arrivare a 1.

CIFRARI A FLUSSO (Stream Ciphers)

La particolarità è che per cifrare processano un testo in chiaro **bit per bit**.

Definizione:
Un **cifrario simmetrico** definito sulla tripla (K, M, C) è una coppia di **algoritmi efficienti** (E, D) dove:

- $E: K \times M \to C$, ed è spesso **randomizzato**
- $D: K \times C \to M$, ed è sempre **deterministico**

Tali che $\forall m \in M, \forall k \in K$:

$$D\big(k, E(k,m)\big) = m$$

NOTA: La chiave usata per la cifratura e la decifratura è la stessa.

Esempi di cifrari simmetrici considerati *sicuri*:

- **ONE-TIME PAD**→ utilizza una chiave lunga esattamente quanto il testo da cifrare, il che fa sì che possa essere utilizzato *solo una volta*. La cifratura avviene facendo lo *XOR* tra i bit del testo in chiaro e i bit della chiave. Formalmente, per un testo in binario: $K = M = C = \{0,1\}^n$, dove:
  - $E(k,m) = k \oplus m$
  - $D(k,c) = k \oplus c$
  - k è scelta *randomicamente da una distribuzione uniforme su K*

  Per dimostrare sia un cifrario, serve dimostrare che $D\big(k, E(k,m)\big) = m$. In questo caso:

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

PRO: è un algoritmo estremamente veloce;
CONTRO: necessita di una chiave molto lunga, quanto il testo da cifrare, che deve essere scambiata in maniera sicura → se il canale fosse supposto sufficientemente sicuro per trasmettere la chiave, sarebbe più semplice inviare direttamente il messaggio in chiaro.

SEGRETEZZA PERFETTA

Partendo dall'assunzione che vi siano solo attaccanti passivi, detti **CT only attack** (CT, cipher text), l'idea è che *il testo cifrato non dovrebbe rivelare alcuna informazione riguardo al testo in chiaro.* Formalmente:

Un cifrario (E, D) sulla tripla (K,M,C) ha **segretezza perfetta** se:

$$\forall c \in C \; e \; \forall \; coppia \; di \; messaggi \; m_0, m_1 \in M \; con \; len(m_0) = len(m_1)$$
$$P[E(k, m_0) = c] = P[E(k, m_1) = c]$$

Dove *k* è uniforme su K.

Ovvero il cifrario ha segretezza perfetta se, scelta in modo uniforme una chiave *k* sullo spazio delle chiavi K, la probabilità che la cifratura di m_0 con *k* sia *c* è uguale alla probabilità che la cifratura di m_1 con *k* sia *c.* Ovvero tale per cui per ogni due messaggi (m_0, m_1) essi hanno la stessa probabilità di essere il testo in chiaro da cui proviene il testo cifrato *c*.

- L'attaccante non ha quindi idea di quale possa essere il testo in chiaro corrispondente, dal momento che hanno tutti la stessa probabilità di esserlo. Non sa dire se il testo in chiaro sia m_0 o m_1.

Questa definizione vale fintanto che si considerano attacchi CT only attack, in caso contrario molti altri attacchi sono possibili.

ONE TIME PAD HA SEGRETEZZA PERFETTA

Ovvero dimostro che per ogni messaggio in chiaro e per ogni messaggio cifrato la probabilità che la cifratura di *m* con una chiave casuale *k* dia *c* come testo cifrato.

$$\forall m, c \quad P_{k \leftarrow K}[E(k, m) = c] = \frac{\#keys \; k \in K \; t.c. \; E(k, m) = c}{|K|} = \frac{1}{|K|}$$

Il numeratore vale 1 perché in un OTP, dato *m* e dato *c*, al più una chiave può cifrare *m* in *c*. Ciò accade perché OTP effettua uno xor bit a bit; quindi, è logico dedurre che, dato c[i] e dato m[i], esiste un unico valore di k[i] in grado di cifrare *m* in *c*. E così per tutti gli altri bit del messaggio.

Dunque, numeratore e denominatore sono costanti →

$$P[E(k, m_0) = c] = P[E(k, m_1) = c] \quad per \; OTP \text{ è } \quad \frac{1}{|K|} = \frac{1}{|K|}$$

Il cifrario ha segretezza perfetta.

CIFRARI CON SEGRETEZZA PERFETTA CHE USANO CHIAVI PIU' CORTE?

Ovvero cifrari con segretezza perfetta ma con k<m, evitando quindi lo svantaggio di OTP → NON ESISTONO

La segretezza perfetta può essere garantita solo se $|K| \geq |M|$

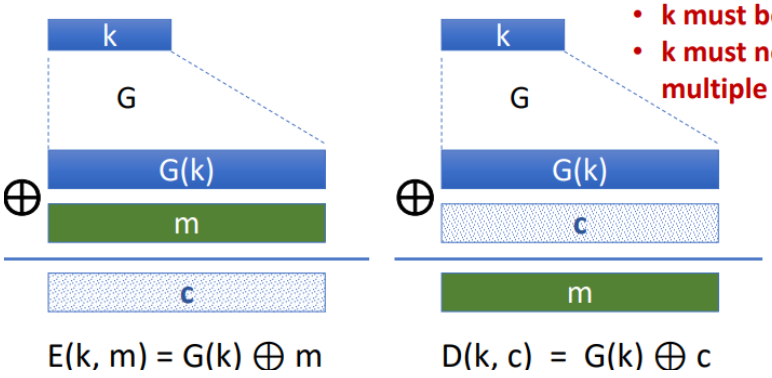
OTP IN PRATICA

Di fatto, OTP non è usato così come è formulato, ma viene leggermente modificato in modo tale da essere più pratico: nella cifratura non si fa lo XOR tra *m* e *k*, si fa lo XOR tra *m* e una sequenza lunga quanto *m* ma generata da una chiave molto più corta del testo in chiaro.
Si usano, cioè, dei **generatori pseudocasuali** (PRG) per generare la chiave.
Un PRG è un algoritmo efficiente deterministico tale da ricevere in input una sequenza *s* di bit (detta **seed**) e produrre in output una sequenza pseudocasuale di gran lunga più grande di *s.*

$$PRG: \; \{0,1\}^s \rightarrow \{0,1\}^n \quad n \gg s$$

Si dicono pseudocasuali dal momento che non è possibile generare reale casualità poiché l'algoritmo è deterministico.

G(k) è la chiave pseudocasuale. Questa viene messa in XOR col messaggio in chiaro per cifrarlo o con il testo cifrato per decifrarlo. Il seed del generatore è la chiave *k*.



- **k must be random**
- **k must not be used multiple times**

NOTA: Ad ogni modo, un **cifrario a flusso non può mai avere segretezza perfetta** dal momento che la chiave è più corta del messaggio in chiaro. Dunque, la sicurezza di un cifrario a flusso dipende dal tipo di PRG scelto e dalla definizione di sicurezza a cui si fa riferimento. Di fatto, esistono *PRG sicuri e PRG insicuri.*

ATTACCHI SU OTP E STREAM CIPHERS

- **One-time pad**:
  - E(k,m) = **k** ⊕ m
  - D(k,c) = **k** ⊕ c

- **Stream ciphers**
  making OTP practical using a **PRG** $G: K \longrightarrow \{0,1\}^n$
  - E(k,m) = **G(k)** ⊕ m
  - D(k,c) = **G(k)** ⊕ c

ATTACCO 1) TWO TIME PAD

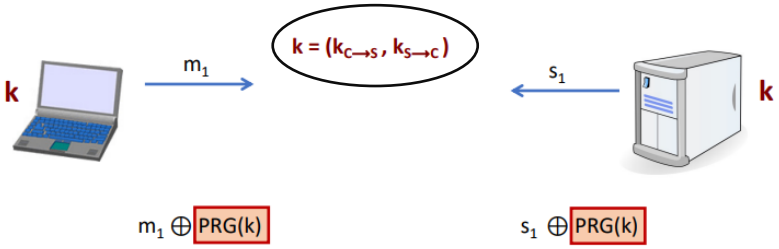
Ovvero un attacco compiuto verso qualcuno che ha violato il principio di one-time key. Infatti, se viene usata la stessa chiave PRG(k) per cifrare, allora facendo lo XOR tra i due testi cifrati si ottiene direttamente lo XOR tra i messaggi in chiaro. Recuperabili poi molto facilmente attraverso tecniche di analisi della lingua e della frequenza.

$$c_1 = m_1 \oplus PRG(k)$$
$$c_2 = m_2 \oplus PRG(k)$$

- $c_1 \oplus c_2 = m_1 \oplus PRG(k) \oplus m_2 \oplus PRG(k) = m_1 \oplus m_2$

ESEMPI REALI:

a) **Protocollo MS-PPTP di Windows NT**, in cui client e server cifravano i propri messaggi con la stessa chiave.



Avrebbero invece dovuto usare:

- una chiave per le comunicazioni da S→C, usata dal server per cifrare e dal client per decifrare;
- una chiave per C→S, usata dal client per cifrare e dal server per decifrare;

Sia client che server hanno entrambi le chiavi (è sempre un cifrario simmetrico), le quali sono parti di una chiave più lunga *k*.

b) **Protocollo 802.11b WEP**, usato per la comunicazione tra client e access point. Vi era una chiave *k* LONG-TERM KEY, usata cioè per cifrare più messaggi.

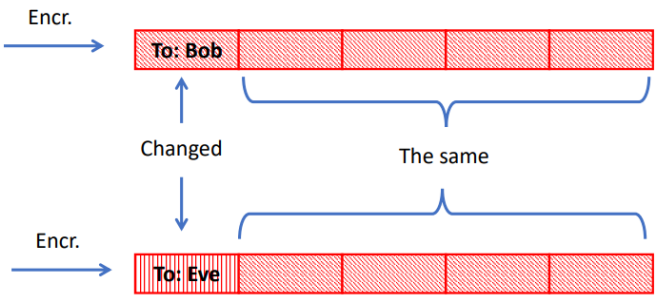
Il messaggio *m* veniva concatenato ad una sequenza CRC(m) per la verifica dell'integrità e il tutto era messo in XOR con PRG(IV || k). Dove:

- IV era un contatore, usato per variare leggermente l'input del generatore pseudocasuale;
- || è l'operatore per la concatenazione;

Il problema sta nel fatto che *k* veniva messa sempre alla fine dell'input di PRG. Dato un input di 128bit, venivano variati sempre e solo i primi 24, i restanti 104 erano della chiave *k*, invariata. Dunque, l'entropia dell'input era estremamente bassa.

Dunque, ogni 2^{24} le sequenze di input erano ripetute. Inoltre, ad ogni riavvio dell'access point IV ripartiva da 0.

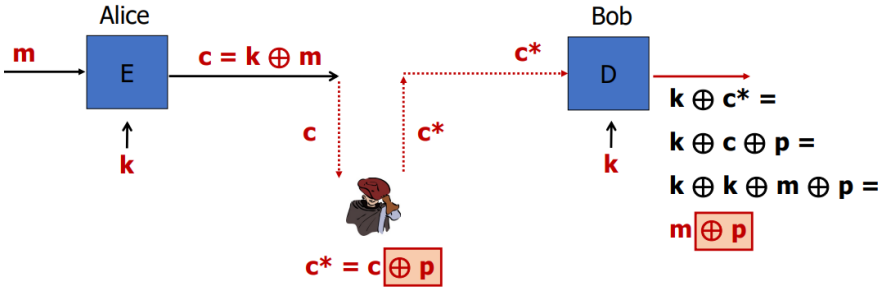
c) **Crittografia del disco**, se l'avversario riesce a capire che parte di due file cifrati coincidono, allora saprà che anche le corrispondenti parti in chiaro coincidono, il che porta ad una perdita di riservatezza.



RIASSUMENDO: mai usare più volte la stessa chiave. In particolare, per le *comunicazioni client/server* vanno usate due chiavi e per la *cifratura del disco* solitamente non si usano cifrari di flusso.

ATTACCO 2) Non c'è integrità

OTP non garantisce integrità! OTP soffre quindi di **malleabilità**.

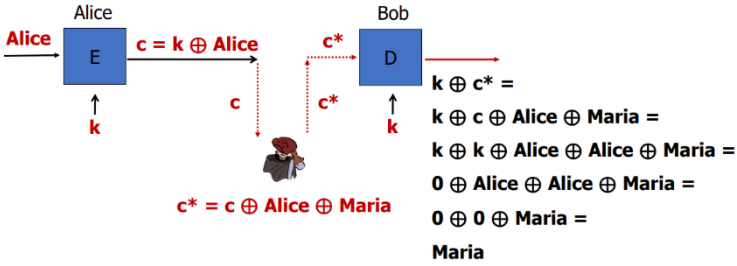


L'avversario può intercettare il testo cifrato *c* e modificarlo, ad esempio mettendolo in XOR con una sequenza *p* arbitraria.

Una volta che Bob riceve il testo cifrato modificato, tenterà di decifrarlo usando la propria chiave, di fatto ottenendo $m \oplus p$. L'avversario non può risalire ad *m* (OTP ha segretezza perfetta) ma può applicare delle modifiche affinché si ottenga una versione modificata di *m* predicibile.

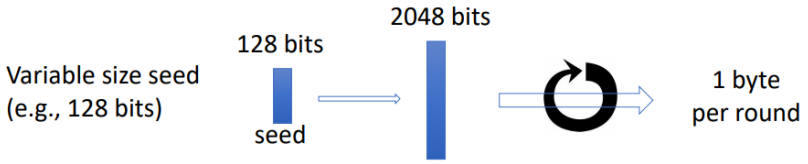
Esempi applicativi di questo attacco:

1) Se l'avversario sa che la risposta di Alice può essere solo 0/1, gli basterà mettere *c* in XOR con 1 per invertire il risultato.
2) L'avversario può anche modificare una parte del messaggio, ad esempio il mittente. Supponendo *m* contenga come prima riga il nome Alice, l'avversario può modificare *c* facendo sì che alla decifratura al posto di Alice ci sia Maria. Per fare ciò, gli basterà fare $c \oplus Alice \oplus Maria$.



ESEMPI REALI DI PRG:

a) **RC4 (software)**, un generatore pseudocasuale in cui il seed=128bit viene poi esteso a 2048bit ai quali viene poi aggiunto 1byte ad ogni ciclo. Questo generatore è insicuro perché i primi byte generati sono prevedibili. Inoltre, si presta bene ad attacchi basati su chiavi simili, ovvero mantenendo costanti i primi 128bit e modificando solo i rimanenti. Era usato nel protocollo 802.11b WEP e in HTTPS.



b) **CSS (hardware)**, utilizzava un particolare tipo di registro: *Linear feedback shift register* (LFSR). Il contenuto iniziale del registro rappresenta il seed del PRG. Ad ogni ciclo di clock, alcuni dei valori delle celle del registro venivano messi in XOR, dando come risultato un bit *b*, e poi il registro veniva sottoposto ad uno shift destro. Liberatasi la posizione più a sinistra, essa veniva occupata da *b*, mentre il bit più a destra veniva dato in input come primo valore del PRG. CSS usava 2 o più LFSR.

CIFRARI A FLUSSO MODERNI: eStream

È un generatore pseudocasuale $PRG: \{0,1\}^s \times R \rightarrow \{0,1\}^n$

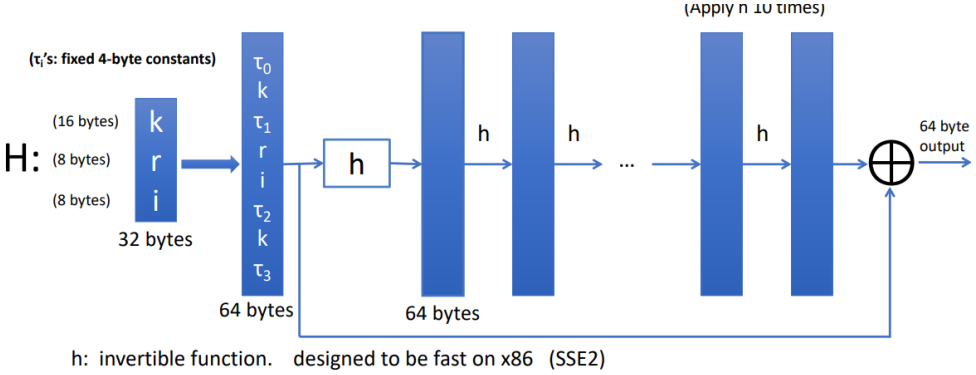
dove R è il **nonce**, ovvero un valore unico per una data chiave tale che ogni coppia (k,r) non è mai usata più di una volta → è possibile usare la stessa chiave *k* fintanto che viene cambiato il nonce.

$$\mathrm{E}(\mathrm{k}, \mathrm{m}, \mathrm{r}) = m \oplus PRG(k, r)$$

ESEMPIO: **Salsa20 (sw+hw)**, è un algoritmo che prende in input un seed di 128/256 bit, un nonce di 64 bit e restituisce una sequenza di *n* bit.

Salsa20: $\{0,1\}^{128 \text{ or } 256} \times \{0,1\}^{64} \longrightarrow \{0,1\}^n$

Salsa20(k, r) := **H(k , (r, 0))** || **H(k , (r, 1))** || …



Internamente, salsa20 costruisce la sequenza in modo ricorsivo, chiamando una certa funzione *H* di volta in volta. Il numero di invocazioni di H dipende dal numero *n.*

τ_1, τ_2, τ_3 sono valori fissati.

QUANDO UN PRG È SICURO/IMPREDICIBILE?

Un PRG è considerato **sicuro** o **impredicibile** (sono nozioni equivalenti). In particolare, un PRG deve essere NON PREDICIBILE. Quindi, **un PRG è sicuro se e solo se è impredicibile**.

DEF 1) IMPREDICIBILITA'

Se un PRG è predicibile, allora vuol dire che $\exists\ i : G(k)|_{1,\dots,i} \xrightarrow{Alg} G(k)|_{i+1,\dots,n}$

Formalmente, si dice che un PRG $G: K \to \{0,1\}^n$ è **predicibile** se:

$$\exists\ un\ algoritmo \text{ efficiente } A\ ed\ \exists\ \ 1 \le i \le n-1 \text{ t.c.}$$

$$P_{k \leftarrow K}\left[A\left(G(k)|_{1,\dots,i}\right) = G(k)|_{i+1}\right] > \frac{1}{2} + \varepsilon$$

$per\ \varepsilon\ non\ trascurabile\ (ad\ es. \varepsilon = \frac{1}{2^{30}})$

Cioè, è predicibile se esiste un algoritmo A che, vedendo solo i primi *i* bit generati dal PRG, è in grado di predire il bit *i+1* con buona probabilità ($> \frac{1}{2} + \varepsilon$).

Allora, un PRG è **impredicibile** se non è predicibile.

($\forall i\ non\ esiste\ un\ algoritmo\ efficiente\ in\ grado\ di\ predire\ il\ bit\ i+1-esimo\ per\ \varepsilon\ non\ trascurabile$).

Esempio) Dato $G: K \to \{0,1\}^n : XOR\left(G(k)\right) = 1$, G è predicibile? Sì, perché dati i primi *n-1* bit sono in grado di predire il valore del bit *n*-esimo.

SICUREZZA

Sia $G: K \to \{0,1\}^n$ un PRG, allora l'obiettivo è rendere il suo output pseudocasuale G(k) **indistinguibile** da un output realmente casuale. Ovvero tale per cui:

$$[k \leftarrow K, output\ G(k)] \quad \text{è } indistinguibile\ da \quad [r \leftarrow \{0,1\}^n, output\ r]$$

NOTA: L'input di un PRG è **realmente casuale**, mentre l'output è pseudocasuale.

Inoltre, un *requisito di sicurezza minimo* (**condizione necessaria**) per PRG è che la lunghezza del seed (s) dovrebbe essere *sufficientemente lunga* in modo tale che sia troppo costoso per un avversario iterare un attacco a forza bruta su tutte le possibili 2^s sequenze di seed.

Test Statistico

È un algoritmo che prende in input una sequenza di *n* bit e restituisce in output 0 oppure 1. Ovvero:

$$A: \{0,1\}^n \to \{0,1\}$$

Un test statistico può essere visto come un attaccante che cerca di mettere in difficoltà il PRG: gli viene data in input una sequenza (non sa se casuale o pseudocasuale) e la sua risposta (0,1) è la sua valutazione a riguardo.

Esempio) A(x) = 1 iff $|\#0(x) - \#1(x)| \le 10\sqrt{n}$, ovvero restituisce 1 se e solo se la differenza tra il numero di 0 e il numero di 1 è limitata superiormente.

Vantaggio

Sia $G: K \to \{0,1\}^n$ un PRG e $A: \{0,1\}^n \to \{0,1\}$ un test statistico su $\{0,1\}^n$. Allora il vantaggio di A nei confronti di G è definito come:

$$Adv_{PRG}[A,G] = \left|P_{k\leftarrow K}\left[A\left(G(k)\right)=1\right] - P_{r\leftarrow\{0,1\}^n}[A(r)=1]\right| \in [0,1]$$

Il vantaggio vuole quantificare la capacità di A di identificare le sequenze pseudocasuali o realmente casuali generate da G. È un numero reale compreso tra 0 e 1. Allora, se:

- $Adv \to 0$, A non riesce a distinguere G dal casuale (G inganna bene A);
- $Adv\ non\ trascurabile$, A può distinguere G dal casuale;
- $Adv \to 1$, A può distinguere G dal casuale molto bene;

Esempio)

> Sia $G: K \to \{0,1\}^n$ che soddisfa msb(G(k))=1 per 2/3 delle chiavi in K (ovvero tali per cui il bit più significativo della sequenza è pari ad 1);
> Sia A(x) il test statistico che restituisce 1 se msb(x)=1 e 0 altrimenti;

Allora, $Adv_{PRG}[A,G] = \left|P_{k\leftarrow K}\left[A\left(G(k)\right)=1\right] - P_{r\leftarrow\{0,1\}^n}[A(r)=1]\right| = \left|\frac{2}{3}-\frac{1}{2}\right| = \frac{1}{6}$, non trascurabile.

Perché:

- $P_{r\leftarrow\{0,1\}^n}[A(r)=1] = \frac{1}{2}$ dal momento che metà delle sequenze hanno il bit più significativo a 0 e l'altra metà a 1.
- $P_{k\leftarrow K}\left[A\left(G(k)\right)=1\right] = \frac{2}{3}$ dal momento che è la probabilità che msb(G(k)) = 1 è 2/3 per ipotesi.

DEF 2) PRG SICURO

Un PRG $G: K \to \{0,1\}^n$ è **sicuro** se $\forall\ test\ statistico\ efficiente\ A,\ Adv_{PRG}[A,G]$ è $trascurabile$.

Analogamente, un PRG è **insicuro** se $\exists\ un\ test\ statistico\ efficiente\ A:\ Adv_{PRG}[A,G]\ non$ è $trascurabile$.

(→)SE UN PRG È SICURO → È IMPREDICIBILE

Dim. Lo dimostro negando la tesi, arrivando a negare l'ipotesi (se è predicibile → è insicuro).

Se il PRG è predicibile, allora esiste un algoritmo efficiente A tale che:

$$P_{k\leftarrow K}\left[A\left(G(k)|_{1,\dots,i}\right) = G(k)|_{i+1}\right] > \frac{1}{2} + \varepsilon$$

$$B(X) = \begin{cases} \text{if } A(X|_{1,\dots,i}) = X_{i+1} \text{ output } 1 \\ \text{else output } 0 \end{cases}$$

Allora, definisco il test statistico B che prende una sequenza X in input come segue.

$k \leftarrow K:\ Pr[B(G(k))=1] > \frac{1}{2} + \epsilon$

$r \leftarrow \{0,1\}^n:\ Pr[B(r)=1] = \frac{1}{2}$

$\Rightarrow Adv_{PRG}[B,G] = |Pr[B(G(k))=1] - Pr[B(r)=1]| > \epsilon$

Il vantaggio non è trascurabile → il PRG è insicuro.

(←)SE UN PRG È IMPREDICIBILE → È SICURO

DEF 3) COMPUTAZIONALMENTE INDISTINGUIBILI

È una definizione che può essere usata per definire la sicurezza per PRG.

Siano P_1 e P_2 due distribuzioni su $\{0,1\}^n$. Allora diciamo che P_1 e P_2 sono **computazionalmente indistibguibili** (indicato con $P_1 \approx_p P_2$) se:

$$\forall\ test\ statistico\ efficiente\ A$$
$$\left|P_{X\leftarrow P_1}[A(X)=1] - P_{X\leftarrow P_2}[A(X)=1]\right| \quad \text{è } trascurabile$$

Cioè, ogni test statistico ha difficoltà a distinguere l'una dall'altra.

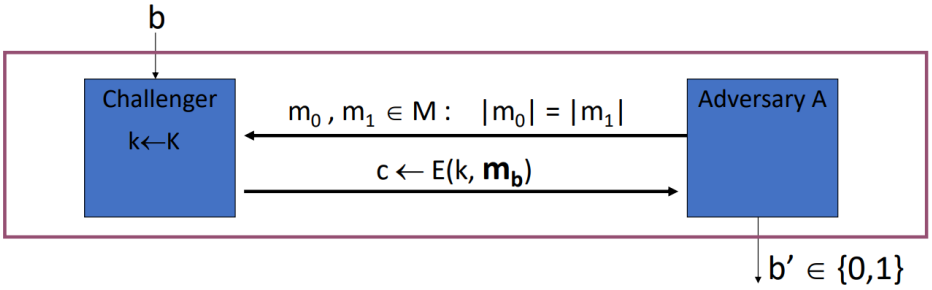
In particolare, un PRG è sicuro se: $\{\boldsymbol{k \leftarrow K}: \boldsymbol{G(k)}\} \approx_{\boldsymbol{p}} \boldsymbol{uniform}(\{\boldsymbol{0,1}\}^{\boldsymbol{n}})$

SICUREZZA SEMANTICA

In questa versione è solo per cifrari simmetrici one-time key, più avanti sarà presentata una definizione analoga per cifrari many-time key. È necessario introdurre questa nuova definizione dal momento che quelle presentate fino a questo momento sono fin troppo restrittive. Anche per questa definizione ci si basa sull'ipotesi che l'avversario attui un CT only attack.

Per un cifrario Q=(E,D) e un avversario A viene definito un *gioco* come segue:

1) A manda a Q due testi in chiaro della stessa lunghezza;
2) Q pesca una chiave in modo uniforme e cifra uno dei due testi inviati da A (senza dire quale), generando *c* e lo invia ad A. Se Q sceglie m_0 si dice che siamo nel caso dell'*esperimento 0* EXP(0), altrimenti *esperimento 1* EXP(1). La scelta di 0,1 è rappresentata da un bit b={0,1};
3) A riceve il testo cifrato e deve decidere se *c* sia la cifratura di m_0 oppure m_1. Dà la propria risposta con un bit b'={0,1};



Allora, il vantaggio di A è definito come:

$$Adv_{ss}[A,Q] = |P[EXP(0)=1] - P[EXP(1)=1]|$$

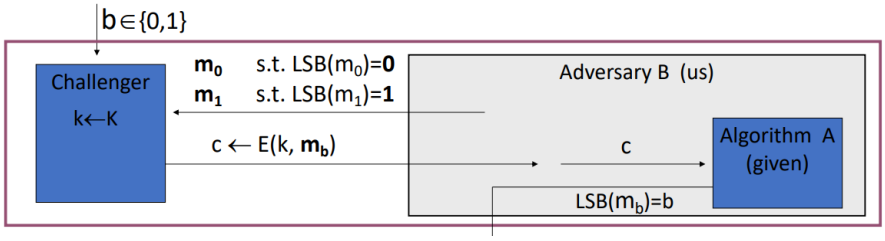
Dire che il vantaggio è basso equivale quindi a dire che A sceglie sia per m_0 che per m_1 b'=1, cioè non li distingue.

Dato un cifrario Q=(E,D) esso è detto **semanticamente sicuro** se per ogni avversario A, il vantaggio dell'avversario su Q è trascurabile. Ovvero se:

$$\forall\ avversario\ A \text{ efficiente}, \qquad Adv_{SS}(A,Q) \text{ è } trascurabile$$

Esempio)

Supponiamo che esista un avversario efficiente B in grado di dedurre sempre il bit meno significativo del testo in chiaro (LSB) a partire dal testo cifrato (CT) → Q non è semanticamente sicuro. Verifichiamo sia così con la definizione formale:



$$Adv_{ss}[B,Q] = |P[EXP(0)=1] - P[EXP(1)=1]| = |0-1| = 1$$

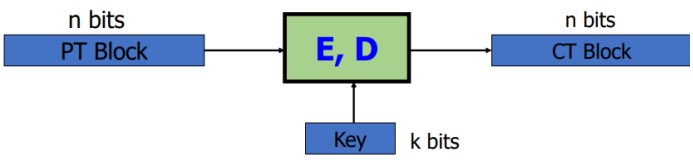
TEOREMA

Se G è un PRG sicuro → un cifrario a flusso Q derivato da G è semanticamente sicuro.

In particolare,

$$\forall\ avversario\ di\ sicurezza\ semantica\ A\ del\ cifrario\ Q,$$
$$\exists\ un\ avversario\ B\ del\ PRG\ G\ (come\ un\ test\ statistico)\ tale\ che$$
$$Adv_{SS}(A,Q) \leq 2 \cdot Adv_{PRG}(B,G)$$

Quindi se Adv_{PRG} è trascurabile, anche Adv_{ss} è trascurabile.



Canonical examples:

- **DES**: n= 64 bits, k = 56 bits
- **3DES**: n= 64 bits, k = 168 bits
- **AES**: n=128 bits, k = 128, 192, 256 bits

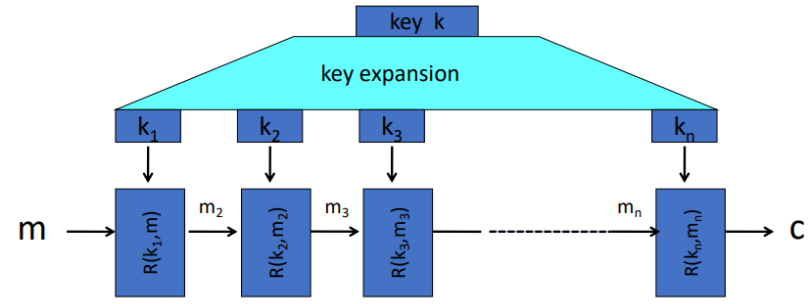
È la seconda famiglia di cifrari simmetrici: cifrano il messaggio in chiaro **a blocchi** (sequenza di bit di lunghezza prefissata).

Il cifrario riceve in input un PT block a *n* bit e restituisce un output un CT block della stessa lunghezza. Per farlo usa una chiave k di *k* bit.
L'algoritmo di decifratura riceve in input un blocco cifrato e restituisce un blocco decifrato della stessa lunghezza.

COSTRUZIONE PER ITERAZIONE

Ad alto livello, si può schematizzare il funzionamento di un generico cifrario a blocco come segue:

1) Si parte da una fase di *espansione della chiave*, in cui la chiave k genera una serie di sottochiavi;
2) Ogni sottochiave sarà usata all'interno di una *round function R(k,m)* che prende in input il testo in chiaro e genera una cifratura parziale. R è chiamata per un certo numero di iterazioni, pari al numero delle sottochiavi generate;
3) Alla fine di questa procedura si ottiene il testo cifrato C;



R(k,m) is called a **round function**

for 3DES (n=48), for AES-128 (n=10)

PRP E PRF

Detto X l'insieme dei blocchi da cifrare e Y l'insieme dei blocchi cifrati,

- Una **Pseudo Random Function** (PRF) definita sulla tripla (K, X, Y) è una funzione

$$F: K \times X \rightarrow Y$$

tale che esiste un algoritmo efficiente in grado di calcolare F(k,x)

- Una **Pseudo Random Permutation** (PRP) definita sulla coppia (K, X) è una funzione

$$E: K \times X \rightarrow X$$

tale che:

1. Esiste un *algoritmo deterministico efficiente* in grado di calcolare E(k,x) → l'algoritmo di cifratura deve essere efficiente e deterministico;
2. La funzione $E(k, \cdot)$ deve essere biettiva per ogni possibile k → non ci devono essere due blocchi in chiaro $m_1, m_2 \ con\, m_1 \neq m_2$ a cui corrisponde lo stesso blocco cifrato;
3. Deve essere invertibile, ovvero esiste un algoritmo inverso efficiente D(k, y) → deve esistere un algoritmo per decifrare;

NOTA

$E(k, \cdot)$ è una funzione derivante da E fissando il primo input di E ad un valore costante k. Ad esempio,

$$f(x,y) = x + y$$
$$f_0(y) = f(0,y) = 0 + y \quad ottenuta\ a\ partire\ da\ f$$

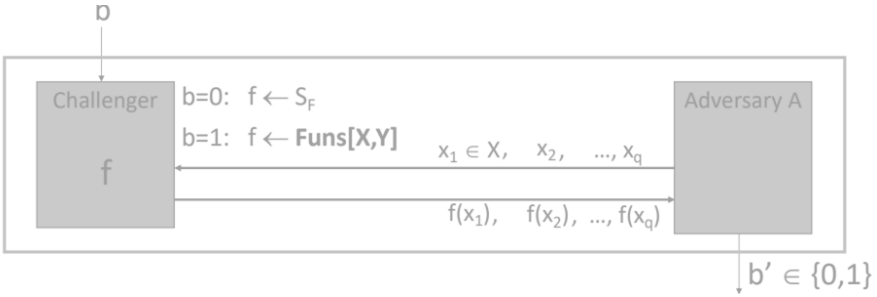
Inoltre, **qualunque PRP è anche una PRF** dove: X=Y, è efficiente ed invertibile.
Non è detto che una PRF sia anche una PRP!

PRF SICURA

Sia $F: K \times X \to Y$ una PRF e definiamo:

- $Funs[X,Y]$, l'insieme di tutte le funzioni $f: X \to Y$
- $S_F = \{F(k, \cdot)\ t.c.\ \ k \in K\} \subseteq Funs[X,Y]$, ovvero l'insieme di tutte le funzioni derivabili da F fissando il primo input di F con una sequenza costante k presa da K;

Allora, definendo un avversario A e un esperimento EXP(b), con b= {0,1}:



1) A sceglie *q* valori da X e li manda a F;
2) F pesca una funzione da Funs[X,Y] oppure da S_F e calcola $f(x_i)$ per ogni valore di x_i mandato da A. Siamo in EXP(0) se è stata scelta una funzione da S_F, EXP(1) altrimenti;
3) A deve dare la propria opinione con b';

$$F \text{ è } \boldsymbol{sicuro}\ se\ \forall\ avversario\ efficiente\ A$$

$$Adv_{PRF}[A,F] = \ |P[EXP(0)=1] - P[EXP(1)=1]| \ \text{ è } trascurabile$$

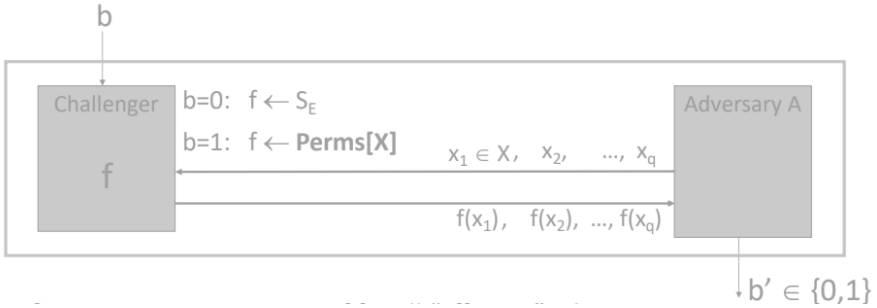
Cioè se una funzione casuale presa da Funs[X,Y] è indistinguibile da una presa da S_F.

PRP SICURA

Sia $E: K \times X \to X$ una PRP e siano:

- $Perms[X]$, l'insieme di tutte le funzioni biettive $f: X \to X$, cioè l'insieme delle permutazioni;
- $S_E = \{E(k, \cdot)\ \ t.c.\ \ k \in K\} \subseteq Perms[X]$

Allora, definendo un avversario A e un esperimento EXP(b), con b= {0,1}:



1) A sceglie *q* valori da X e li manda a F;
2) F pesca una funzione da Perms[X] oppure da S_E e calcola $f(x_i)$ per ogni valore di x_i mandato da A. Siamo in EXP(0) se è stata scelta una funzione da S_E, EXP(1) altrimenti;
3) A deve dare la propria opinione con b';

$$E \text{ è } \boldsymbol{sicuro}\ se\ \forall\ avversario\ efficiente\ A$$

$$Adv_{PRP}[A,E] = \ |P[EXP(0)=1] - P[EXP(1)=1]| \ \text{ è } trascurabile$$

NOTA: una **PRP sicura è un cifrario a blocchi sicuro**.
Cioè, se una funzione casuale presa da Perms[X] è indistinguibile da una presa da S_E.
Idealmente, per cifrare un blocco bisognerebbe prendere in maniera casuale una funzione da Perms[X] perché molto grande. Realisticamente, verrà presa una chiave casuale dall'insieme delle chiavi del cifrario, ovvero da S_E.
Quindi, una PRP è sicura quando pescare dall'insieme piccolo (S_E) equivale a pescare dall'insieme più grande (Perms[X]).

DATA ENCRYPTION STANDARD (DES)

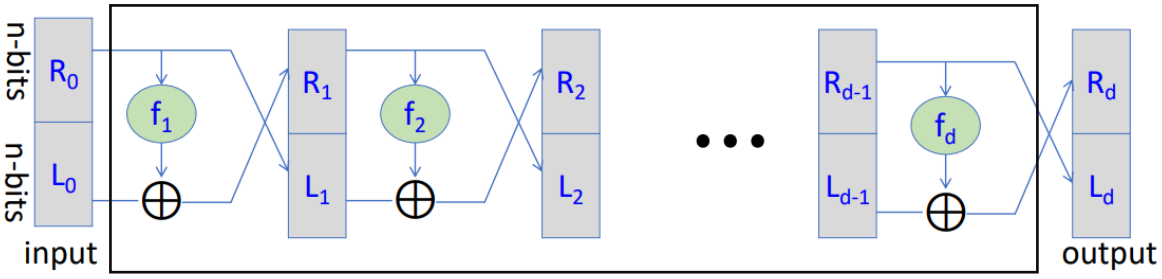
È un cifrario a blocchi che, ad oggi, è considerato *insicuro* (chiavi troppo piccole). Sebbene originariamente (1970) la lunghezza delle chiavi del DES era di 128bit, nel momento in cui venne reso uno standard (1976) essa venne dimezzata a 56bit. Infine, nel 1997 un attacco a forza bruta riuscì a decifrare DES. Nel 2000 DES venne sostituito da AES.

Idea di base: **FEISTEL NETWORK**

L'obiettivo dell'algoritmo era: date delle funzioni non necessariamente invertibili, le si vuole usare per costruire una funzione invertibile (ovvero l'algoritmo di cifratura di un cifrario a blocchi).

Cioè:

$$date\ f_1, \dots, f_d: \{0,1\}^n \to \{0,1\}^n \quad non\ necessariamente\ invertibili$$
$$costruire\ F: \{0,1\}^{2n} \to \{0,1\}^{2n}\ invertibile$$



In symbols: $R_i = f_i(R_{i-1}) \oplus L_{i-1}$, $L_i = R_{i-1}$

1. L'input da 2n bit viene suddiviso in due metà, da n bit l'uno (R_0 e L_0).
2. A partire da R_0 e L_0 vengono generati R_1 e L_1, tali che
$$L_1 = R_0$$
$$R_1 = f_1(R_0) \oplus L_0$$
3. E così via.

La Feistel Network è interessante per due motivi:

1) Consente di generare una funzione invertibile, e quindi un cifrario a blocchi, a partire da funzioni non invertibili;
2) È considerata sicura sotto specifiche condizioni;

TEOREMA DI INVERTIBILITA' DELLA FEISTEL NETWORK

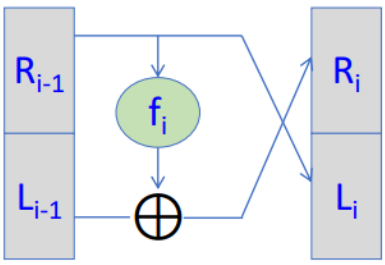
La Feistel Network **è invertibile**. Più formalmente,

$$per\ tutte\ le\ funzioni\ arbitrarie\ f_1, \dots, f_d: \{0,1\}^n \to \{0,1\}^n$$
$$la\ Feistel\ Network\ F: \{0,1\}^{2n} \to \{0,1\}^{2n}\ \text{è}\ invertibile$$

Dim.

Basta dimostrare che a partire da un output di uno dei sottoblocchi della rete riesco ad ottenere l'input dello stesso sottoblocco. Infatti:

$$R_{i-1} = L_i$$
e
$$L_{i-1} = R_i \oplus f_i(L_i)$$

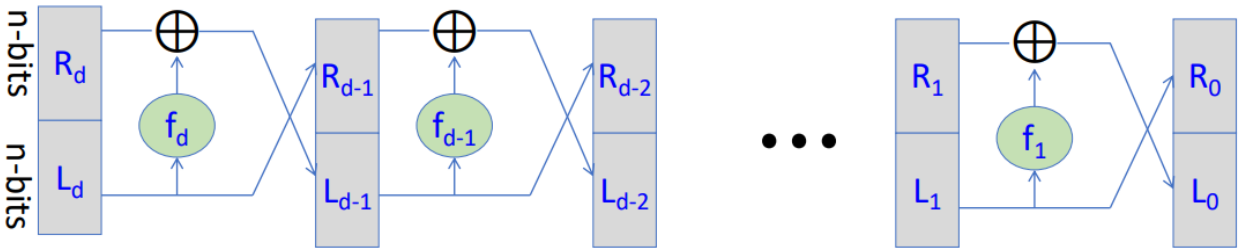


Questo è vero per la proprietà dello XOR secondo cui

$$se\ c = a \oplus b, \quad allora\ b = a \oplus c$$

CIRCUITO DI DECIFRATURA

Dunque, per invertire una Feistel Network, basta invertire ogni suo sottoblocco:



Nello specifico, l'inverso è lo stesso circuito di partenza ma in cui le funzioni $f_1, \dots, f_d$ sono applicate in ordine inverso.

La Feistel Network è utilizzata in molti cifrari a blocchi, ma non in AES.

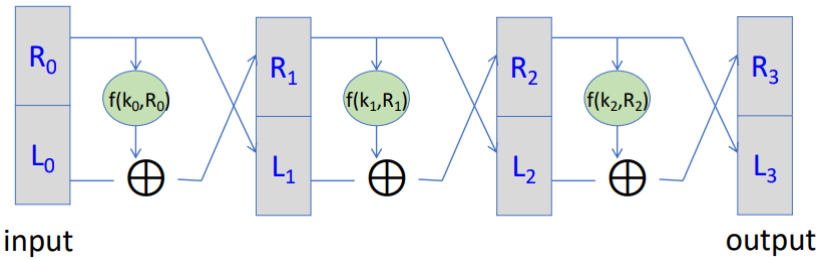
TEOREMA (Luby-Rackoff, 1985)

Sia $f: K \times \{0,1\}^n \to \{0,1\}^n$ una PRF sicura.

- $Feistel\ network\ di\ 3\ round$

$F: K^3 \times \{0,1\}^{2n} \to \{0,1\}^{2n}$,

$con\ k_0, k_1, k_2\ chiavi\ indipendenti$, è una **PRP sicura**



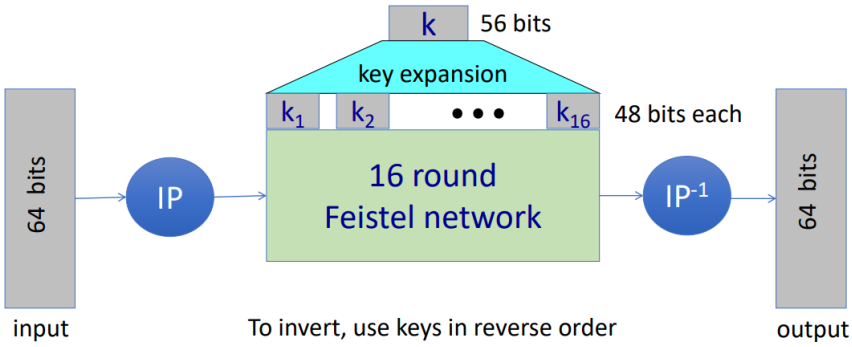
La particolarità sta nella funzione utilizzata: essa prende in input una chiave e una sequenza e restituisce una sequenza.

DES: 16 ROUND FEISTEL NETWORK

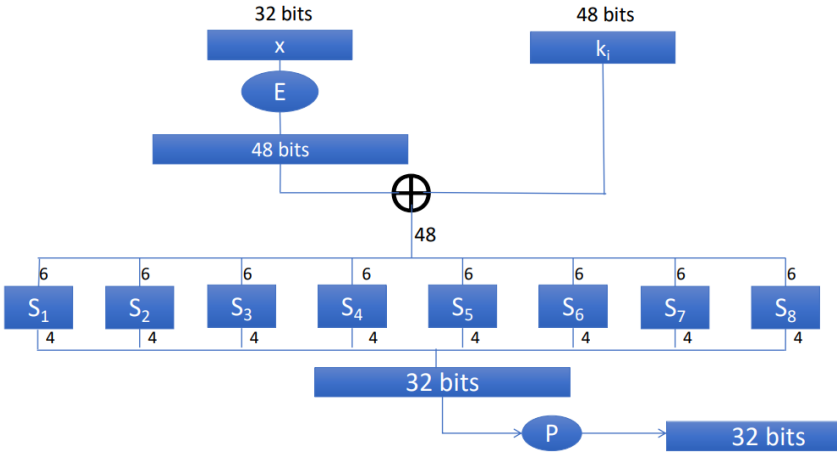
$f_1, \dots, f_{16}: \{0,1\}^{32} \to \{0,1\}^{32}$

$f_i(x) = F(k_i, x)$

1. Si parte da una chiave da 56 bit, la quale viene suddivisa in 16 chiavi da 48 bit;
2. Il blocco in chiaro da 64 bit viene sottoposto ad una prima permutazione (IP);
3. Il blocco passa poi attraverso la Feistel Network da 16 round, la quale utilizza le chiavi da 48 bit;
4. Infine, passa attraverso una permutazione inversa (IP^{-1});



La funzione $F(k_i, x)$, internamente:



S_i è detta **S-box** (*substitution box*) ed è una generica procedura che riceve in input 6bit e restituisce 4bit, cioè $s: \{0,1\}^6 \to \{0,1\}^4$

Vi è una tabella precompilata, in cui l'input rappresenta la cella a cui accedere: i 4bit centrali sono la colonna, mentre il bit più significativo e quello meno significativo la riga. Es. 111001 è la cella [11][1100].

Il contenuto della cella è l'output. È quindi una **look-up table**. Vi sono delle euristiche che compilano la matrice delle S-box: in generale non dovrebbe essere una funzione lineare.

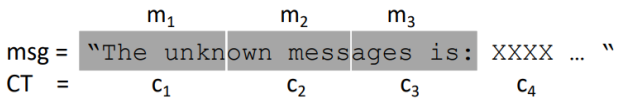
DES è insicuro perché una delle sue S-box è insicura: si comporta in modo quasi lineare, prevedibile.

NOTA: Perché nel th. di Luby-Rackoff sono sufficienti 3 round per rendere la PRP sicura ma con DES non ne bastano 16? Perché in DES manca l'ipotesi di *chiavi indipendenti*.

ATTACCHI DI RICERCA ESAUSTIVA (attacchi a forza bruta)

Usati per mettere in crisi DES. Dunque, vennero indette delle competizioni, dette **DES Challenges**, in cui l'obiettivo era dimostrare che DES fosse insicuro.

Queste gare erano strutturate nel seguente modo:



- Veniva fornito un testo in chiaro *msg*, suddiviso in blocchi da 64bit;
- Gli organizzatori sceglievano poi una chiave per cifrare questi blocchi;
- L'obiettivo era trovare la chiave k usata per cifrare il testo avendo a disposizione il testo cifrato C e i primi 3 blocchi in chiaro, ovvero trovare $k \in \{0,1\}^{56} : DES(k, m_i) = c_i \; \forall i$;

Furono quindi proposti alcuni meccanismi per rendere DES più sicuro:

1. **Triple-DES**

Dato un cifrario a blocchi

$$E: K \times M \to M$$
$$D: K \times M \to M$$

Triple-DES è il cifrario a blocchi $3E: K^3 \times M \to M$, ovvero:

$$3E\ (k_1, k_2, k_3, m) = E(k_1, D(k_2, E(k_3, m)))$$

Prende, cioè, 3 chiavi dal cifrario a blocchi E e cifra *m* con k_3, lo decifra con k_2 e infine lo cifra di nuovo con k_1.

È strutturato in questo modo per motivi di retrocompatibilità: se voglio usare 3DES ma cifrare come DES, basta applicare 3DES su $k_1 = k_2 = k_3$. Infatti, si avrebbe: $3E(k, k, k, m) = E\left(k, D\left(k, E(k, m)\right)\right) = E(k, m)$

Questo vale perché, per ipotesi, (E, D) è un cifrario.

CARATTERISTICHE DI 3DES:

- Usa **chiavi da 168bit** (3x56);
- È **tre volte più lento** di DES;
- Se $k_1 = k_2 = k_3$, 3DES=DES;

3DES È ATTACCABILE DA UN BRUTE FORCE ATTACK? No, perché ci mette $O(2^{168})$. Vi è però un attacco che in un tempo $O(2^{118})$ viola il 3DES, che comunque è un tempo troppo elevato.

PERCHE' NON 2DES?

Ha una struttura del tipo: $2DES(k_1, k_2, m) = E(k_1, E(k_2, m))$ e una complessità di $O(2^{112})$ per un attacco di ricerca esaustiva.

Tuttavia, è stato sviluppato l'attacco *meet in the middle* che ci impiega solo $O(2^{63})$→ insicuro.

2. **DESX**

Dato un cifrario a blocchi

$$E: K \times M \to M$$
$$D: K \times M \to M$$

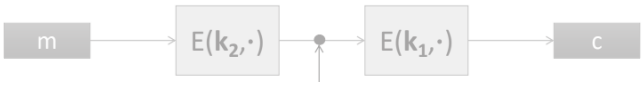
Definiamo EX come: $EX(k_1, k_2, k_3, m) = k_1 \oplus E(k_2, m \oplus k_3)$

CARATTERISTICHE DI DESX

- **chiavi di 184bit** (64+56+64), dove $len(k_1) = len(k_3) = 64bit, \quad len(k_2) = 56bit$
- Un attacco di mette $O(2^{120})$
- Modificarlo con $k_1 \oplus E(k_2, m)$ o con $E(k_2, m \oplus k_1)$ rende DESX insicuro

Queste due varianti possono essere applicate a qualunque cifrario a blocchi.

1. MEET IN THE MIDDLE ATTACK



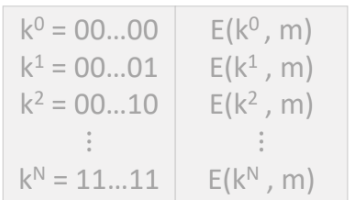
INPUT: blocco in chiaro *m* e blocco cifrato *c* tale che 2DES(m)=c.
OUTPUT/OBIETTIVO: trovare

$$(k_1, k_2)\ t.c.\ E\big(k_1, E(k_2, m)\big) = c$$

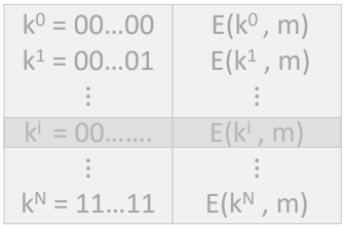
o, in modo equivalente, trovare

$$(k_1, k_2)\ t.c. E(k_2, m) = D(k_1, c)$$

L'attacco si sviluppa in due passi.



PASSO 1)
Viene costruita una tabella con 2 colonne e 2^{56} righe come segue: per ogni riga si ha nella prima colonna una possibile chiave e nella seconda colonna la corrispondente cifratura.
La tabella è ordinata *rispetto ai valori della seconda colonna*.



PASSO 2)
Per ogni $k \in \{0,1\}^{56}$, verifica che $D(k, c)$ si trovi nella seconda colonna della tabella (*binary search*). Se così fosse, allora avremmo che:

$$E(k^i, m) = D(k, c)\ allora\ \ (k^i, k) = (k_1, k_2)$$

COMPLESSITA')

$$\text{Time} = \underbrace{2^{56}\log(2^{56})}_{\text{build + sort table}} + \underbrace{2^{56}\log(2^{56})}_{\text{search in table}} < 2^{63} \ll 2^{112}, \qquad \text{Space} \approx 2^{56}$$

SU 3DES)

$$\text{Time} = 2^{118}, \qquad \text{space} \approx 2^{56}$$

$$\text{Time} = \underbrace{2^{56}\log(2^{56})}_{\text{build + sort table}} + \underbrace{2^{112}\log(2^{56})}_{\text{search in table}} < 2^{118}$$

2. ATTACKS ON IMPLEMENTATION

a. SIDE-CHANNEL ATTACK

Sfruttano canali secondari per compromettere la sicurezza. Ad esempio, monitorare il tempo di esecuzione o del consumo elettrico.

b. FAULT ATTACKS

Errori nell'esecuzione dell'algoritmo può portare all'esposizione della chiave → Non implementare primitive crittografiche da sé!

3. LINEAR ATTACKS ON DES

Una piccola linearità nella S-box S_5 ha abbassato la complessità dell'attacco a forza bruta sul DES da 2^{56} a 2^{43}.

4. QUANTUM ATTACKS

Considerando un generico problema di ricerca tale per cui, sia $f: X \to \{0,1\}$, una funzione
L'obiettivo è trovare $x^* \in X\ t.c.\ f(x^*) = 1$

- Un computer classico ci impiega $O(|X|)$
- Un computer quantistico ci impiega $O\left(\left|X^{\frac{1}{2}}\right|\right)$ secondo l'**algoritmo di Grover**

Allora, è possibile effettuare una **ricerca esaustiva quantistica**.

In particolare, per la ricerca di una chiave, dati m e $c = E(k, m)$ definiamo:

$$per\ k \in K,\ \ f(k) = \begin{cases} 1 & se\ E(k,m) = c \\ 0 & altrimenti \end{cases}$$

Ovvero tramite questo tipo di ricerca si definisce la funzione *f(k)* tale da restituire 1 se k è la chiave utilizzata per cifrare; 0 altrimenti.

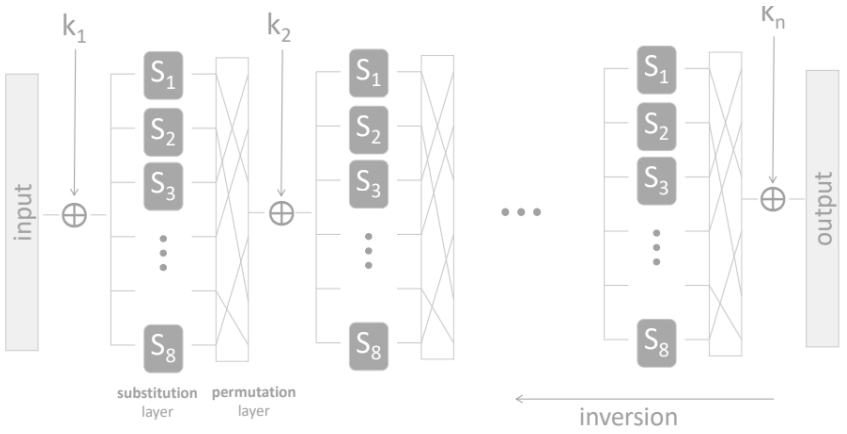
Quindi, sapendo che un computer quantistico ci impiega $O(\left|X^{\frac{1}{2}}\right|)$, il costo temporale per DES e AES diventa:

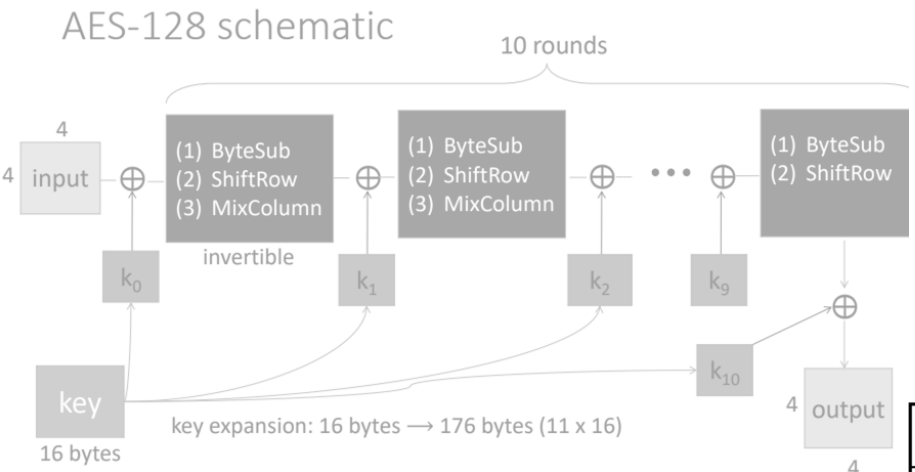
- DES→ $O(2^{28})$
- AES a 128-bit → $O(2^{64})$

Questo porta alla nascita di una nuova branca della crittografia: la *crittografia post-quantistica*, che si occupa di garantire un buon livello di sicurezza a fronte dell'aumento della capacità computazionale degli elaboratori.

ADVANCED ENCRYPTION STANDARD (AES)

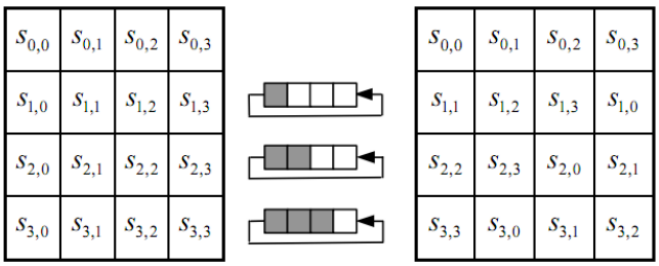
Utilizza **blocchi da 128 bit** e **chiavi da 128/192/256** bit.
AES non utilizza una rete di Feistel ma una **rete a permutazione e sostituzione**.





Prende in input un blocco da 128bit, visto come una matrice 4x4.
La round function è **invertibile** ed è definita da 3 funzioni in XOR con la chiave. Questa procedura è chiamata 10 volte, tenendo a mente che nell'ultimo round la funzione 3) non viene eseguita. Le sottoprocedure sono:

1) *ByteSub*, è una lookup table applicata ad ogni valore della matrice di input;



2) *ShiftRow*, fa una serie di shift. Nella figura seguente, a sx vi è l'input e a dx l'output: la prima riga resta invariata. La seconda riga dell'output è ottenuta a partire dalla prima facendo uno shift circolare sinistro; ecc….
3) *MixColumn*, una sequenza di 128bit viene vista come una matrice 4x4 e si compiono delle operazioni sulle colonne;

<u>AES IN HARDWARE</u>

Ci sono alcuni processori che agevola la scrittura di AES: ad esempio *Intel Westmere* consente di effettuare un round di AES in un ciclo di clock (**aesenc, aesenclast**) oppure effettuare la key expansion (**aeskeygenassist**). Alcuni processori scelgono di supportare nativamente AES dal momento che è ampiamente utilizzato.

ATTACCHI

- Il miglior attacco contro AES è 4 volte più veloce di un attacco a forza bruta. Cioè, da $O(2^{128})$ a $O(2^{126})$.
- Vi è un altro attacco che ci impiega solo $O(2^{99})$ ma è considerato innocuo dal momento che richiede uno scenario molto improbabile all'atto pratico: serve che ci siano 2^{99} blocchi in chiaro cifrati con 4 chiavi a 256bit correlate tra loro.

PRF→PRG E PRG →PRF

DA PRF→PRG

$Sia\ F: K \times \{0,1\}^n \to \{0,1\}^n\ \ una\ PRF,$
$definiamo\ \ PRG\ G: K \to \{0,1\}^{nt}\ (con\ t\ arbitrario)\ come\ segue:$

$$G(k) = F(k,0) \parallel F(k,1) \parallel \cdots \parallel F(k,t-1)$$

Dove $F(k,0)$ è l'output della PRF che riceve come parametro k e una sequenza di *n* bit che codifica il numero 0.
F(k,i) va richiamata tante volte quanti sono i bit di cui si ha bisogno nell'output di G. Ad esempio:

- $G: K \to \{0,1\}^{2n}$ = $F(k,0) \parallel F(k,1)$
- $G: K \to \{0,1\}^{3n}$ = $F(k,0) \parallel F(k,1) \parallel F(k,2)$
- Se non è multiplo di *n*, ne genero di più e scarto gli eccessi

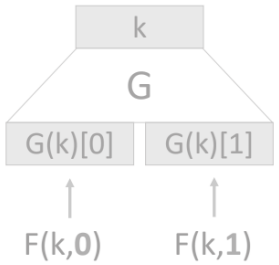
Dunque, per ottenere un generatore pseudocasuale a partire da un cifrario a blocchi non si fa altro che cifrare le sequenze 0,1,…,t-1 usando come chiave il seed del generatore e concatenarle insieme.
NOTA: Questa costruzione è **parallelizzabile**, cioè è possibile calcolare parallelamente tutte le F(k,i) e poi concatenarle.

TEOREMA: *Se la PRF è sicura*, allora il PRG così costruito è un **PRG sicuro**.

DA PRG→PRF a 1-bit

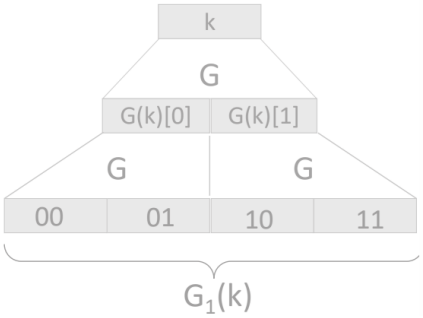
$Sia\ G: K \to K^2\ un\ PRG$ (ovvero tale per cui la lunghezza dell'output è il doppio dell'input), allora definiamo una PRF ad 1-bit:

$$F: K \times \{0,1\} \to K, \qquad cioè$$
$$F(k, x \in \{0,1\}) = G(k)[x]$$



Quindi F riceve in input *11 bit* (k ha 10 bit).
G restituisce una sequenza di 20 bit, la quale viene divisa in due sottosequenze da 10 bit.
Se x=0, viene restituita G(k)[0] altrimenti G(k)[1].
Il primo input di F viene usato come *seed* del generatore.

TEOREMA: Se G è un PRG sicuro, allora la PRF così costruita è una **PRF sicura**.



DA PRG→PRF a 2-bit

$Sia\ G: K \to K^2\ un\ PRG$, allora definiamo una PRF a 2-bit:

$$F: K \times \{0,1\}^2 \to K, \qquad cioè$$
$$F(k, x \in \{0,1\}^2) = G(k)[x]$$

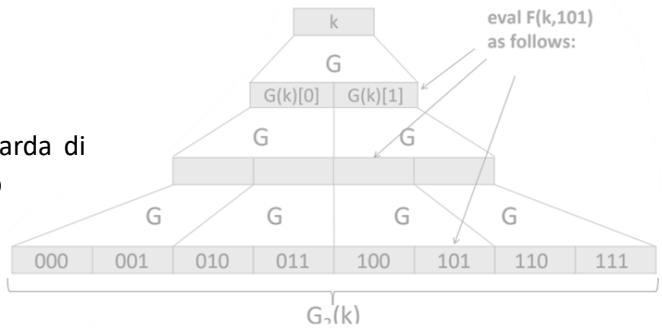
F in questo caso riceve *12 bit* in input.
G restituisce una sequenza da 40 bit, divisa in due sottosequenze da 20bit l'una.
Anche in questo caso però l'input è limitato: accetta solo blocchi da 2-bit.

DA PRG→PRF a 3-bit

$Sia\ G: K \to K^2\ un\ PRG$, allora definiamo una PRF a 3-bit:

$$F: K \times \{0,1\}^3 \to K, \qquad cioè$$
$$F(k, x \in \{0,1\}^3) = G(k)[x]$$

Per ottimizzare, piuttosto che generare interamente l'albero, si guarda di volta in volta l'input di F e si effettua un pruning sui rami dell'albero che non verranno visitati. Ad esempio, per F(k,101) posso eliminare tutto il ramo G(k)[0] solo guardando il primo bit (1), e così via.



Così facendo, costruisce un numero di istanze di G pari alla lunghezza della sequenza di input di x. Lo stesso ragionamento può essere applicato per input più grandi.
In ogni caso, è **troppo costoso** per input grandi.

DA PRG→PRP SICURO?

È possibile costruire una PRP sicura a partire da un PRG sicuro? Si, basta utilizzare una PRF sicura ottenuta a partire dal PRG sicuro all'interno di una Feistel Network a 3 round (Th. Luby-Rackoff). Cioè, **PRG sicuro → PRF sicura → PRP sicuro**.
Come già detto, a causa dei costi di generazione, non viene applicato.

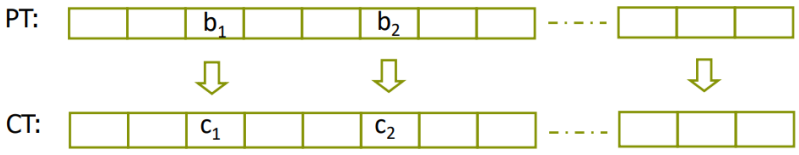
MODI OPERATIVI

Un **modo operativo** è un cifrario simmetrico (algoritmo) che *usa un cifrario a blocchi per cifrare messaggi di più blocchi*.
Esistono:

- *One-time key*
  - Electronic Code Book (ECB)
  - Deterministic Counter Mode (DETCTR)
- *Many-time key*
  - Cipher Block Chaining (CBC)
  - Counter Mode (CTR)

MODES OF OPERATION ONE-TIME KEY

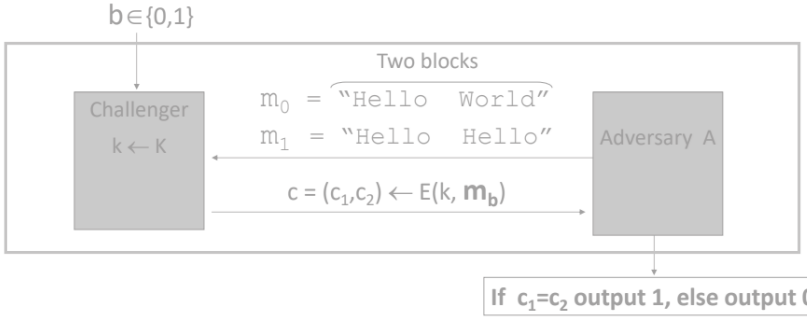
ELECTRONIC CODE BOOK (ECB) → INSICURO



Suddivide il testo in chiaro in blocchi (se si usa AES, viene suddiviso in blocchi da 128bit). I blocchi vengono poi cifrati col cifrario scelto.
La chiave non cambia per i diversi blocchi nello stesso messaggio.

PROBLEMA: se $b_1 = b_2\ allora\ c_1 = c_2$
Cioè, nonostante si stia utilizzando un PRP sicuro, questo viene applicato in maniera insicura.

Per dimostrare formalmente l'insicurezza di ECB, bisogna applicare la definizione di *sicurezza semantica*. In particolare, risulta che **ECB non è semanticamente sicuro per messaggi contenenti più di un blocco**. Infatti, è sufficiente che l'avversario mandi due messaggi $m_0 \neq m_1$ tali da poter essere suddivisi in uno stesso numero di blocchi e tali per cui m_1 presenta due blocchi identici.

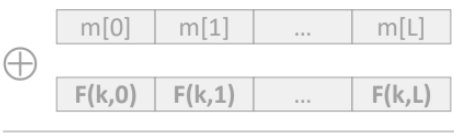


Se la cifratura risultante dei due blocchi è tale per cui $c_1 = c_2$, allora $b' = E(k, m_1) = 1$, 0 altrimenti.

Allora, $Adv_{SS}[A, ECB] = 1$ → ECB è semanticamente insicuro.

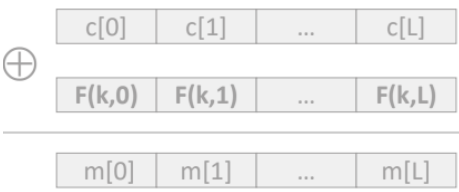
DETERMINISTIC COUNTER MODE (DETCTR)→ SICURO

Sia data una PRF $F: K \times \{0,1\}^n \rightarrow \{0,1\}^n$ (come AES, 3DES), allora $E_{DETCTR}(k, m) =$



Cioè, ottenuto il PRG a partire dalla PRF, lo si mette in XOR con i blocchi del messaggio in chiaro per ottenere i blocchi cifrati (è la stessa idea del **cifrario a flusso**).

Per decifrare, $D_{DETCTR}(k, c) =$



NOTA: *non c'è bisogno di invertire F* per le proprietà dello XOR.
Inoltre, *il cifrario a blocchi è applicato solo alla PRF*.

TEOREMA $\forall\, L > 0, se\ F$ è $una\ \boldsymbol{PRF\ sicura}\ su\ (K, X, X)$,
$allora\ DETCTR$ è $\boldsymbol{sematicamente\ sicuro}\ su\ (K, X^L, X^L)$

In particolare,

$\forall\ avversario\ efficiente\ A\ (avv.della\ sicurezza\ semantica) che\ attacca\ DETCTR,$
$\exists\ un\ avversario\ efficiente\ B\ (avv.della\ PRF) che\ attacca\ F\ t.c.$
$$\boldsymbol{Adv_{SS}[A, DETCTR] = 2 \cdot Adv_{PRF}[B, F]}$$

Sapendo che, per ipotesi, F è una PRF sicura:
$$Adv_{PRF}[B, F]\ \text{è}\ trascurabile \rightarrow\ Adv_{SS}[A, DETCTR]\ deve\ essere\ trascurabile$$

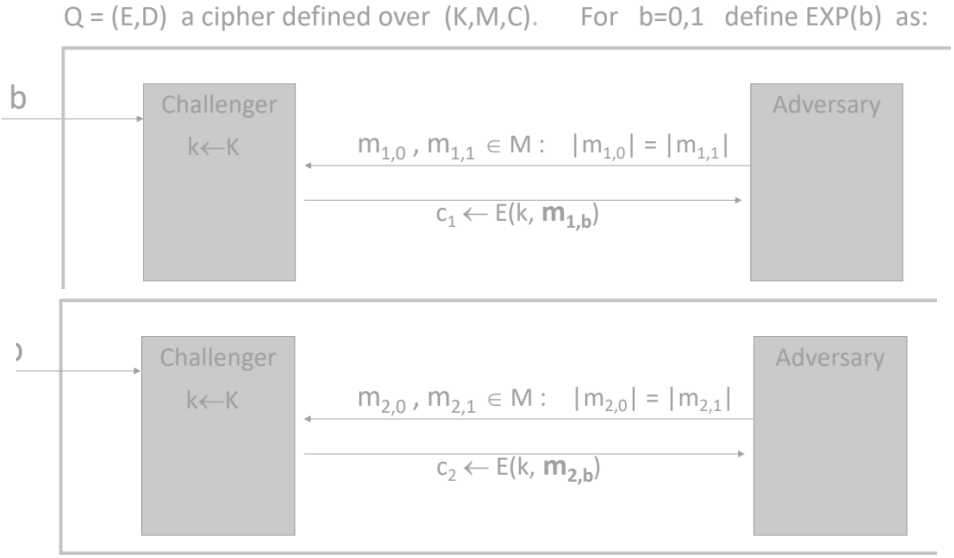
MODES OF OPERATION MANY-TIME KEY

Chiave usata più di una volta per cifrare testi diversi.

SICUREZZA SEMANTICA PER MANY-TIME KEY

POTERI DELL'AVVERSARIO: può intercettare testi cifrati *da lui scelti* diversi ottenuti per mezzo di una stessa chiave (**Chosen-PT Attack**, **CPA**).
OBIETTIVO DELL'AVVERSARIO: violare la riservatezza.

Anche in questo caso si fa riferimento ad un gioco che si sviluppa su **round**.



ROUND 1)
1. A manda una coppia di messaggi a Q;
2. Q cifra una dei due messaggi;

ROUND 2)
1. A sceglie altri due messaggi e li invia a Q;
2. Q cifra uno dei due messaggi con la stessa chiave del round precedente tale per cui: se aveva scelto il primo messaggio nel round 1, cifrerà il primo messaggio al round 2, e viceversa.

Definiamo:
- EXP(0)=Q decide di cifrare sempre il primo messaggio di ogni coppia;
- EXP(1)= Q sceglie di cifrare sempre il secondo messaggio di ogni coppia;

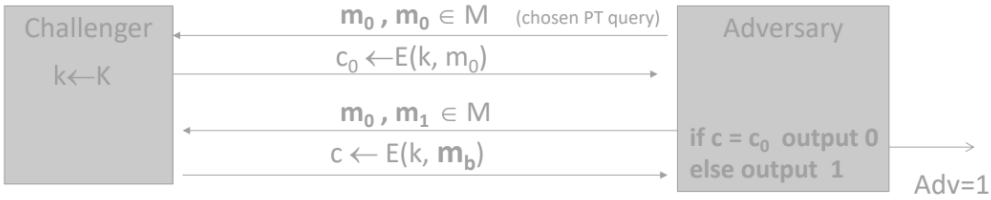
L'avversario, se vuole, può inviare in un round due messaggi **uguali**, ottenendo la cifratura del messaggio che aveva scelto.

Allora, Q è **semanticamente sicuro** sotto CPA se $\forall\ avversario\ efficiente\ A$,
$$Adv_{CPA}[A, Q] = |P[EXP(0) = 1] - P[EXP(1) = 1]|\quad \text{è}\ trascurabile$$

CIFRARI INSICURI SOTTO CPA

Dato un cifrario simmetrico many-time key, se il suo algoritmo di cifratura E(k,m) è *deterministico*, ovvero se richiamato con lo stesso input (stessa chiave e stesso PT) restituisce sempre lo stesso output, l'intero cifrario è insicuro.
Cioè, **se l'algoritmo di cifratura è deterministico, allora il cifrario many-time key che ne fa uso è insicuro.**

Dim. Si usa la definizione di sicurezza semantica.
A può scegliere di inviare:
ROUND 1) due messaggi identici $m_0 = m_1$
ROUND 2) due messaggi m_0, m_1



Allora, A confronta i testi cifrati ottenuti nei due round. Ponendo:

$$b' = \begin{cases} 0 \text{ se } c = c_0 \\ 1 \; altrimenti \end{cases}$$

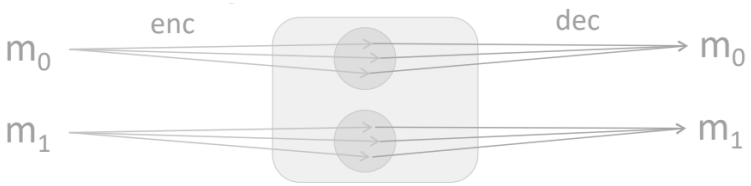
Il che porta a **Adv=1** → è semanticamente insicuro.

CONDIZIONE NECESSARIA ALLA SICUREZZA DEL CIFRARIO MANY-TIME KEY

È importante che **per un cifrario simmetrico many-time key restituisca output diversi sullo stesso input**. Questa proprietà è *necessaria ma non sufficiente* per la sicurezza del cifrario.
Ciò si può ottenere in due modi:

1. Usando un *algoritmo randomizzato;*
2. Usando un *Nonce*;

SOLUZIONE 1. RANDOMIZED ENCRYPTION



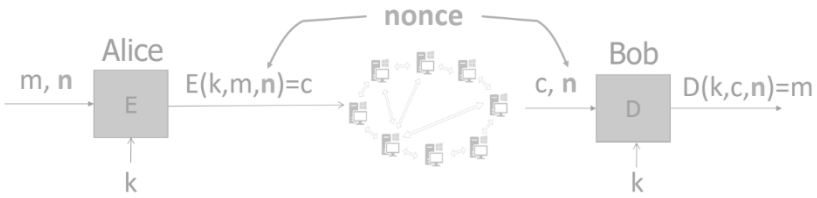
Sia E(k,m) un algoritmo randomizzato. Allora:

- Ad ogni cifratura dello stesso PT si ottengono CT diversi con *un'elevata probabilità* (potrebbe capitare che non sia così);
- È importante che gli insiemi dei CT associati a due PT diversi siano *disgiunti tra loro*, per evitare che i due PT siano cifrati con lo stesso CT;
- *CT deve essere più lungo del corrispondente PT*. Questa proprietà deve valere per gli algoritmi randomizzati dal momento che ad uno stesso PT sono associati diversi CT: deve essere possibile mappare più CT ad uno stesso PT, utilizzando di fatto bit aggiuntivi. Di conseguenza, ci devono essere *più CT di quanti siano i PT in numero*, poiché il $\#CT = 2^{len(CT)}$ *e* $\#PT = 2^{len(PT)}$.

In altre parole: $CT_{size} = PT_{size} + \#random\ bits$

SOLUZIONE 2. NONCE-BASED ENCRYPTION

L'algoritmo di cifratura deterministico viene modificato in modo da ricevere in input (k,m,n), dove *n* è il Nonce.



Il Nonce ha alcune proprietà:

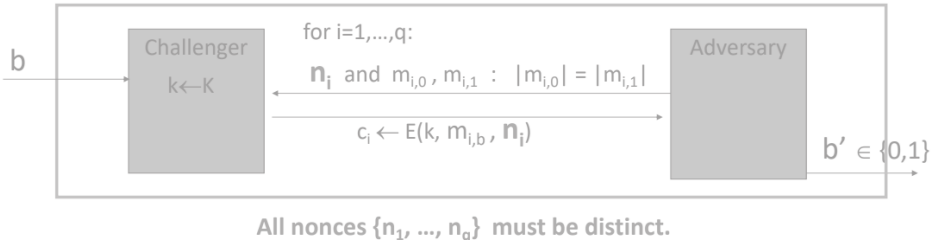
- È un valore variabile da msg a msg;
- La coppia (k,n) non può essere mai usata più di una volta;
- Non è necessario che *n* sia segreto né che sia casuale;

Per mettere in pratica questo tipo di cifratura, si possono seguire due metodi distinti per la gestione del Nonce:

1) Nonce come **contatore**: in questo caso non è necessario che *n* sia scambiato tra mittente e destinatario; infatti, basta che A e B siano *sincronizzati* sul valore iniziale di *n* da usare. Importante allora che i CT arrivino nell'ordine in cui sono stati inviati.
2) Nonce come **numero casuale**, $n \leftarrow N$: N sia sufficientemente grande da garantire con elevata probabilità che uno stesso nonce non sia estratto più di una volta.

SICUREZZA SEMANTICA PER NONCE-BASED ENCRYPTION

In questa versione, il **nonce è scelto dall'avversario**. Quindi, A invia al cifrario non solo la coppia di messaggi da cifrare ma anche i Nonce *n*. L'unica limitazione dell'avversario è che deve scegliere dei *nonce distinti tra loro*.



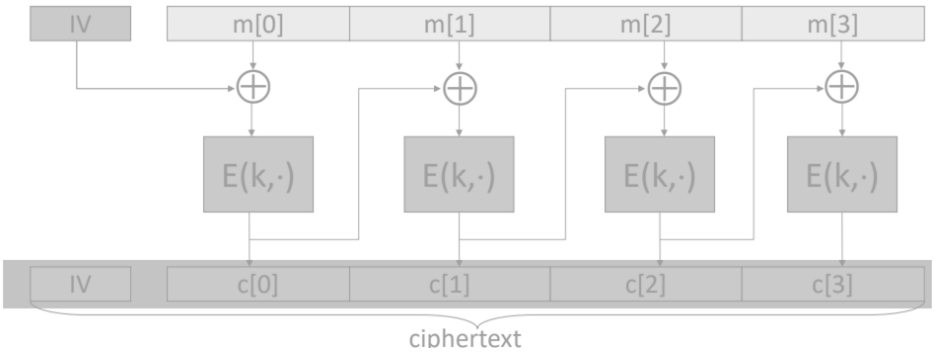
Allora, il cifrario è considerato **semanticamente sicuro sotto CPA** se

$\forall\ avversario\ efficiente\ A,$
$Adv_{nCPA} =$
$|P[EXP(O)] = 1 - P[EXP(1) = 1|$
è $trascurabile$

CIPHER BLOCK CHAINING (CBC)

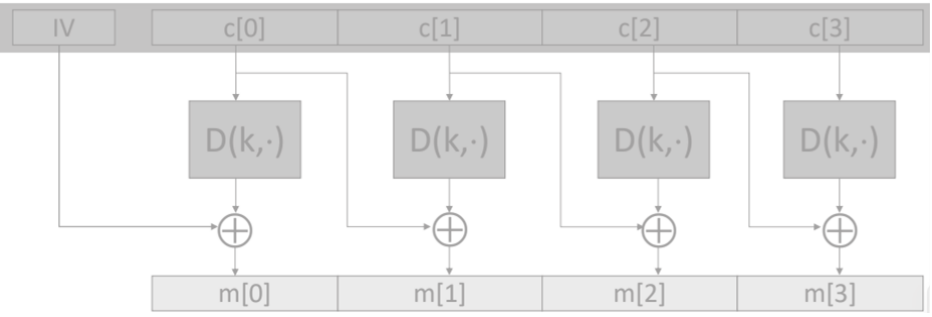
COSTRUZIONE 1) CBC con random IV *(Initialization Vector)*

Data una PRP $E: K \times \{0,1\}^n \rightarrow \{0,1\}^n$, $E_{CBC}(k, m)$ sceglie un valore casuale $IV \in \{0,1\}^n$ e procede come segue:



1. Divide *m* in una serie di blocchi della lunghezza di *n* bit (la lunghezza dipende dalla PRP che si sta usando);
2. Sceglie una sequenza IV di *n* bit (quanto il blocco).
3. IV viene messa in XOR con il blocco m[0], ottenendo il Nonce del singolo blocco.
4. Il Nonce viene dato in input a E(k, *), ottenendo c[0].
5. Il risultato è messo in XOR col blocco in chiaro successivo;
6. IV viene inviato insieme all'insieme dei blocchi cifrati, poiché è necessario al destinatario per decifrare.

Per decifrare, l'algoritmo inverso è $D: K \times \{0,1\}^n \rightarrow \{0,1\}^n$, $D_{CBC}(k, c)$ procede nel seguente modo:



1. A partire da c[0] applico D(k,c) per ottenere la decifratura;
2. La decifratura è messa in XOR con IV per ottenere m[0];
3. c[1] viene decifrato con D(k,c) e il risultato è messo in XOR con c[0] per ottenere m[1];
4. e così via;

SICUREZZA DI CBC (RANDOMIZZATO)

Teorema: per qualunque $L > 0$

$se\ E$ è $una\ PRP\ sicura\ su\ (K, X), allora\ CBC$ è $semanticamente\ sicuro\ sotto\ CPA\ su\ (K, X^L, X^{L+1})$

In particolare,

$$\forall\ avversario\ efficiente\ A\ che\ attacca\ CBC$$
$$\exists\ un\ avversario\ efficiente\ B\ che\ attacca\ E\ t.c.$$
$$\boldsymbol{Adv_{CPA}[A, CBC] \leq 2 \cdot Adv_{PRP}[B, E] + \frac{2q^2L^2}{|X|}}$$

Dove:

- $q = \#messaggi\ cifrati\ con\ la\ stessa\ chiave$
- $X = insieme\ dei\ blocchi\ della\ PRP\ utilizzati\ all'interno\ di\ CBC$
- $L = \#blocchi\ del\ messaggio\ più\ lungo\ tra\ quelli\ cifrati\ con\ la\ stessa\ chiave$

Dunque, $Adv_{CPA}[A, CBC]$ è trascurabile? Dipende dal secondo membro. So che $Adv_{PRP}[B, E]$ è trascurabile (per Hp), allora dipende tutto da $\frac{2q^2L^2}{|X|}$. → CBC è sicuro fintanto che $q^2L^2 \ll |X|$

Il numeratore non è costante! Aumenta all'aumentare del numero di blocchi e di messaggi cifrati con la stessa chiave.

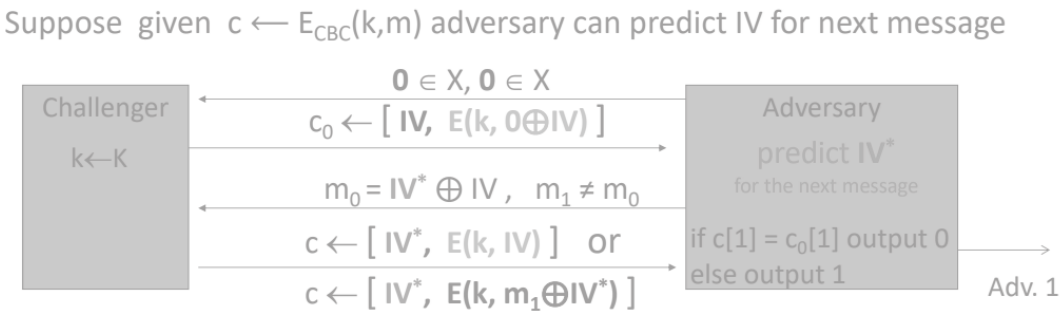
➔ Nonostante una stessa chiave possa essere usata più volte, è necessario che ad un certo punto venga cambiata (cioè, quando il numeratore non è più trascurabile)

Sarebbe ideale avere una chiave che *necessita di essere cambiata poco frequentemente*: ciò accade *quando il denominatore è molto grande,* facendo sì che il numeratore si avvicini ad esso molto lentamente; cioè, quando i blocchi sono molto lunghi.

Quindi, CBC per essere sicuro:

- Vantaggio trascurabile
- Cambiare chiave quando necessario
- È necessario che l'IV scelto sia **impredicibile**.

Infatti, se IV *fosse predicibile:*



1. A invia due messaggi uguali (CPA) composti da un solo blocco di tutti 0;
2. C ne cifra uno dei due usando CBC con una chiave scelta da lui. Genera così c_0, composto da due blocchi: IV e $E(k, 0 \oplus IV)$
   I blocchi restituiti sono due per il modello di funzionamento di CBC: invia anche IV.
3. IV* è noto ad A per ipotesi. A invia due messaggi m_0 e m_1 diversi tra loro, tali per cui $m_o = IV^* \oplus IV$
4. A questo punto, se C sceglie di cifrare m_0 siamo in EXP(0), in EXP(1) altrimenti. Allora:
   a. Restituisce c composto dai blocchi $[IV^*, E(k, IV)]$ se sceglie EXP(0)
   b. Restituisce c composto dai blocchi $[IV^*, E(k, m_1 \oplus IV^*)]$ se sceglie EXP(1)
5. A sceglie nel seguente modo:
   a. Se $c_0[1] = c[1] \rightarrow EXP(0)$ perché il secondo blocco di c_0 contiene $IV \oplus 0 = IV$, quindi se coincidono i due secondi blocchi vuol dire che è stato cifrato m_0
   b. EXP(1) altrimenti

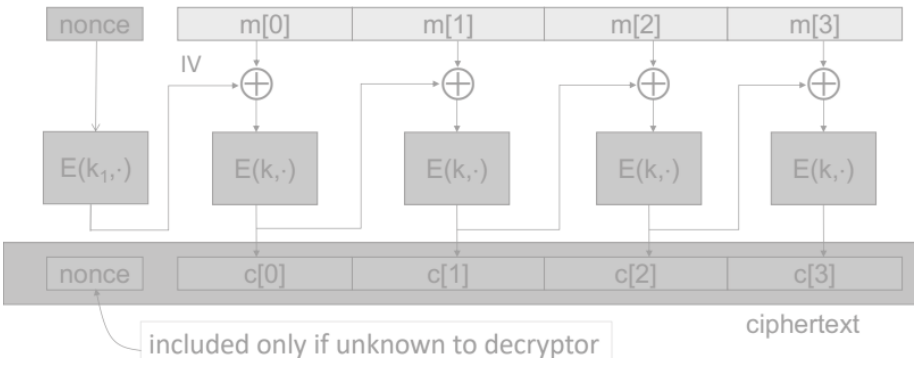
➔ $Adv_{CPA}[A, CBC] = 1$

➔ **Se IV è predicibile, CBC è insicuro**

COSTRUZIONE 2) NONCE-BASED CBC

Si usano **due chiavi (k, k_1)** e, inoltre, la **coppia (k,n) può essere usata solo una volta**.

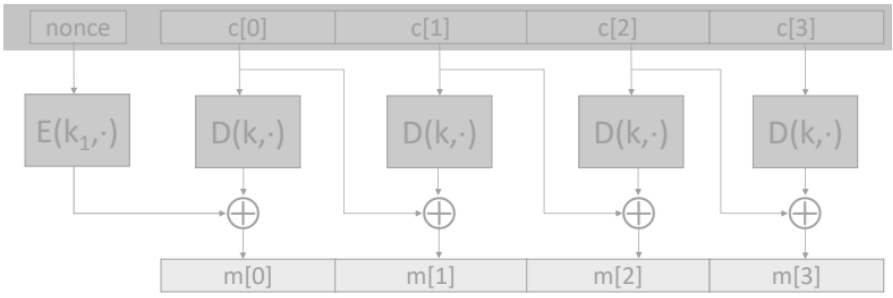
La cifratura avviene *dividendo m in blocchi* e poi:



1. Il nonce viene cifrato usando la chiave k_1, ed il suo output definisce IV;
2. IV è messo in XOR col primo blocco in chiaro e poi cifrato con la chiave k. Il risultato è messo in XOR col blocco in chiaro successivo;

La decifratura, invece:

1. E(k_1,n) per ottenere IV;
2. Decifra il primo blocco cifrato con k:D(k, c[0]) e il risultato va in XOR con IV;
3. C[0] è poi utilizzato in XOR con D(k,c[1]) per ottenere m[1];

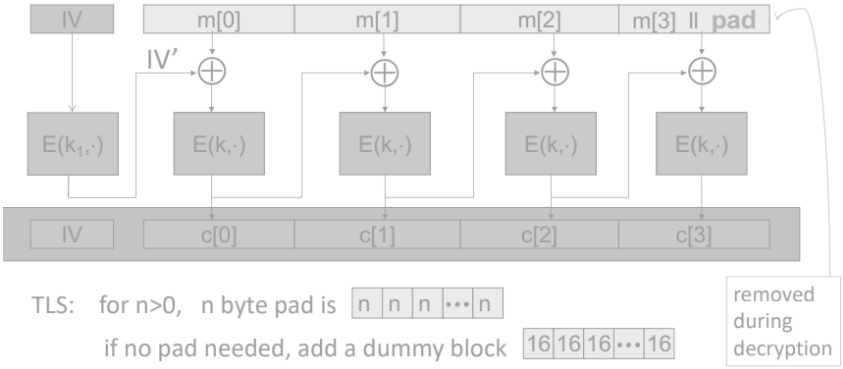


NOTA: La lunghezza del nonce influenza la frequenza con cui va cambiata la chiave dal momento che i nonce generabili sono un numero finito: terminati quelli per una stessa chiave, va cambiata. Inoltre, il *nonce va cifrato* poiché è prevedibile.

SE LA LUNGHEZZA DI *m* NON È MULTIPLO DELLA LUNGHEZZA DEL BLOCCO? **PADDING**

Ad ogni modo, anche quando la lunghezza di *m* è multiplo del blocco, va aggiunto un **dummy block**, un blocco contenente sottoblocchi da 16byte rappresentativi del numero 16.

Questo va fatto per evitare che il destinatario elimini parte del messaggio in fase di decifratura.



Esempio reale: funzione di **OpenSSL**

```
void AES_cbc_encrypt(
        const unsigned char *in, ⟵ puntatore a PT
        unsigned char *out, ⟵ puntatore a CT
        size_t length,
        const AES_KEY *key,
        unsigned char *ivec, ⟵ IV fornito dal chiamante
        AES_ENCRYPT or AES_DECRYPT
);
```

Questa funzione può essere usata sia per cifrare che per decifrare (la tipologia è definita dall'ultimo parametro della funzione, booleano).

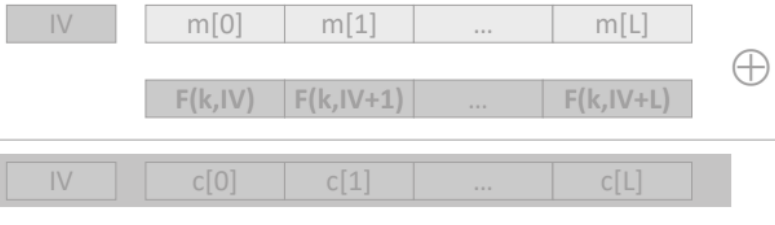
Il problema di questa funzione sta nel fatto che IV è usato direttamente senza essere crittografato prima!

COUNTER MODE (CTR)

COSTRUZIONE 1) CTR RANDOMIZZATO

Data una PRF $F: K \times \{0,1\}^n \rightarrow \{0,1\}^n$, l'algoritmo di cifratura $\boldsymbol{E_{CTR}(k,m)}$ sceglie un $IV \in \{0,1\}^n$ *casuale* e procede nel seguente modo:

1. Divide il PT in più blocchi;
2. Il blocco i-esimo è messo in XOR con F(k,IV+i), con $0 \leq i \leq L$ (L=#blocchi);



Così come accadeva in DETCTR, anche qui la concatenazione degli output della F(k, IV) definisce un PRG.

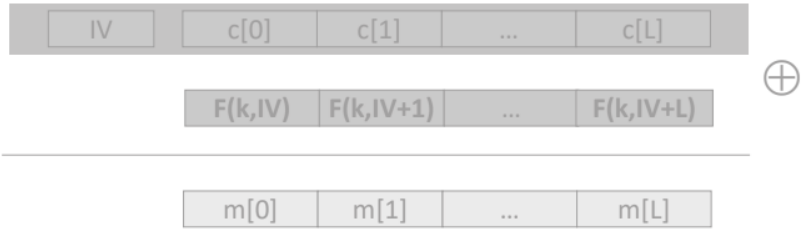
Inoltre, CTR è **parallelizzabile**, a differenza di CBC.

Per la decifratura $\boldsymbol{D_{CTR}(k,c)}$

➔ $c[i] \oplus F(k, IV + i) = m[i]$

➔ Non c'è bisogno di invertire F.

➔ Il destinatario ha bisogno di ricevere IV dal momento che è *generato casualmente*.



SICUREZZA DI CTR RANDOMIZZATO

Teorema: per qualunque $L > 0$

$se\ F$ è $una\ PRF\ sicura\ su\ (K, X, X), allora\ CTR$ è $semanticamente\ sicuro\ sotto\ CPA\ su\ (K, X^L, X^{L+1})$

In particolare,

$$\forall\ avversario\ efficiente\ A\ che\ attacca\ CTR$$
$$\exists\ un\ avversario\ efficiente\ B\ che\ attacca\ F\ t.c.$$
$$\boldsymbol{Adv_{CPA}[A, CTR] \leq 2 \cdot Adv_{PRF}[B, F] + \frac{2q^2L}{|X|}}$$

Dove:

- $q = \#messaggi\ cifrati\ con\ la\ stessa\ chiave$
- $X = insieme\ dei\ blocchi\ della\ PRF\ utilizzati\ all'interno\ di\ CTR$
- $L = \#blocchi\ del\ messaggio\ più\ lungo\ tra\ quelli\ cifrati\ con\ la\ stessa\ chiave$

➔ CTR è sicuro fintanto che $q^2L \ll |X|$, migliore di CBC.

Esempio:

supponendo di voler ottenere $Adv_{CTR}[A, CTR] \leq \frac{1}{2^{32}}$

voglio avere $2q^2L < \frac{1}{2^{32}}$

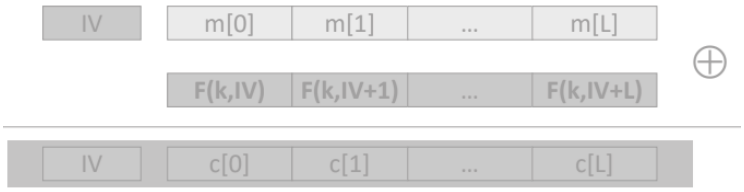
per AES: $|X| = 2^{128} \rightarrow qL^{\frac{1}{2}} < 2^{48}$

quindi, dopo 2^{32} CT di lunghezza 2^{32} (per un totale di 2^{64} blocchi) bisogna cambiare chiave.

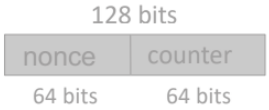
Per CBC in AES: $|X| = 2^{128} \rightarrow qL < 2^{48}$ → bisogna cambiare dopo 2^{48} blocchi.

COSTRUZIONE 2) NONCE-BASED CTR

Data una PRF $F: K \times \{0,1\}^n \rightarrow \{0,1\}^n$, l'algoritmo di cifratura $\boldsymbol{E_{CTR}(k, m, nonce)}$ e procede nel seguente modo: identico a CTR randomizzato, con l'unica differenza che IV non è scelto casualmente ma deriva dal nonce.



Si noti inoltre come il *nonce resti invariato per blocchi di uno stesso messaggio*. Varia invece per messaggi diversi.



IV è quindi suddiviso in due metà della stessa dimensione. Il counter parte da 0 e viene incrementato per ogni blocco.

Per la decifratura, $\boldsymbol{D_{CTR}(k, c, nonce)}$ e funziona esattamente come la decifratura in CTR randomizzato.

Riassumendo:

PRINCIPI CHIAVE	**CBC**	**CTR**
Uses	PRP	PRF
Parallel processing	NO	YES
Security of random encryption	$q^2L^2 \ll \|X\|$	$q^2L \ll \|X\|$
Dummy padding block	YES*	NO**

*esiste una variante, detta *ciphertext stealing*, che non richiede padding.

**nel caso in cui *m* non sia multiplo della lunghezza del blocco, i bit aggiuntivi rispetto a m[L] nella sequenza F(k, IV+L) vengono ignorati.

NOTA: CTR ha una *maggiore applicabilità* rispetto a CBC dal momento che necessita di una PRP che non sia necessariamente anche una PRF. Inoltre, se ho una PRP che è anche una PRF, posso usare sia CBC che CTR.

INTEGRITA'

Se A invia un messaggio a B, B deve essere in grado di determinare se il messaggio ricevuto sia stato modificato o meno. L'attaccante in questo scenario diventa **attivo**: oltre a leggere il traffico, lo può fermare, modificare o generare altro traffico. L'obiettivo dell'avversario è quindi *modificare il traffico e far sì che il destinatario non si accorga di tali cambiamenti*.
Nella gestione dell'*integrità dei messaggi* in questa prima analisi non ci si preoccuperà della *confidenzialità*. Ad esempio, verificare l'integrità di un messaggio inviato da un server (checksum).

MESSAGE AUTHENTICATION CODE (MAC)

Alice e Bob sono interessati esclusivamente all'integrità della loro conversazione, e fanno uso di una stessa chiave *k*.

1. Alice invia insieme al PT un **tag *t***, una sequenza di bit generata per mezzo del messaggio in chiaro *m* e la chiave *k*;
2. Bob riceve *(m,t)* e per verificare sia integro, calcola un tag **t'** utilizzando *m* e *k*. Confronta poi t con t'.

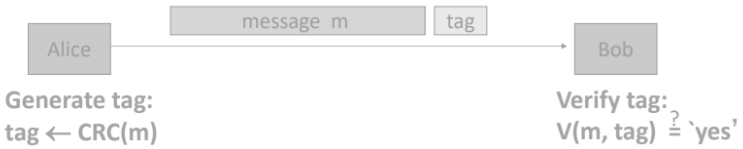
Formalmente, un **MAC** definito sulla tripla (K, M, T) è una coppia di algoritmi efficienti **I=(S, V)** dove:

- $S: K \times M \to T$, algoritmo di **firma** (*signing algorithm*)
- $V: K \times M \times T \to \{yes, no\}$, algoritmo di **verifica** (*verification algorithm*)

$$t.c. \quad \forall m \in M, \forall k \in K \qquad V\big(m, k, S(k,m)\big) = 'yes'$$

L'obiettivo dell'avversario è quindi quello di generare un tag valido che B possa accettare: è importante che la *chiave resti segreta*. Se dovesse ottenere il tag corretto, basterebbe che invii la coppia *(m', t)* a B.
Ma, in ogni caso, **il tag è più corto del messaggio** → potrebbe capitare che uno stesso tag sia associato a messaggi diversi: *deve essere difficile per l'avversario trovare due messaggi a cui è associato lo stesso tag*.
D'altra parte, non si può usare un tag molto lungo per motivi di **efficienza**. Per motivi di **sicurezza**, non può nemmeno essere troppo corto. Se il tag fosse molto corto, infatti, ci sarebbe un'alta probabilità per l'avversario di azzeccare il tag corretto tra tutti quelli generabili (che non saranno molti). L'avversario **ha un solo tentativo**: non è ricerca esaustiva perché, se il tag inviato dovesse essere sbagliato, la connessione verrebbe chiusa.

➔ Il MAC non è altro che il **CRC con chiave**. Il CRC è usato per garantire integrità nei confronti di **errori casuali** (dovuti ad esempio al mezzo di comunicazione), non per opera di un agente malevolo.

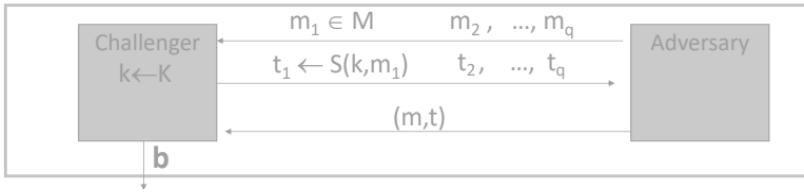


Si può notare come, in questo caso, non c'è alcuna chiave in gioco: all'avversario basta ricalcolare il tag per mezzo di CRC usando però il messaggio modificato *m'* invece che *m*. CRC(m')→tag.

SICUREZZA DEL MAC

Poteri dell'avversario: **Chosen Message Attack**→ $per\ m_1, \dots, m_q$ l'attaccante riceve $t_i \leftarrow S(k, m_i)$ e può scegliere i messaggi m_i. Quindi possiede le coppie valide $(m_1, t_1), \dots, (m_q, t_q)$. Può ottenere, cioè, il tag di messaggi arbitrari scelti da lui.
Obiettivo dell'avversario: **Existential Forgery** → produrre *nuove coppie valide messaggio/tag* $(m, t) \notin \{(m_1, t_1), \dots, (m_q, t_q)\}$
Formalmente, dato un MAC I=(S,V) su (K,M,T) e un avversario A:

1. A invia una serie di messaggi $m_1, \dots, m_q$. Può anche non inviare nulla;
2. C sceglie una chiave *k* e con essa genera i tag $t_1, \dots, t_q$;
3. A invia una coppia (m,t);
4. C deve decidere se la coppia inviata da A sia valida o meno;



$$b = \begin{cases} 1\ se\ V(k,m,t) = yes\ and\ (m,t) \notin \{(m_1,t_1), \dots, (m_q,t_q)\} \to attaccante\ vince \\ 0\ altrimenti \to attaccante\ perde \end{cases}$$

Allora, un MAC I=(S,V) è **sicuro** se $\forall\ avversario\ efficiente\ A$:

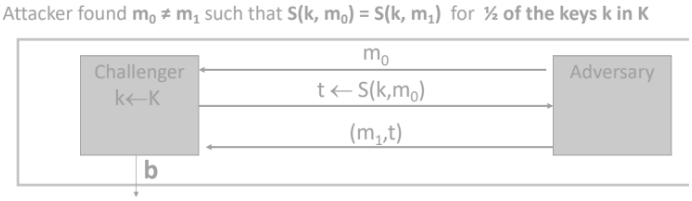
$$\boldsymbol{Adv_{MAC}[A,I] = P[b=1]}\ \text{è}\ \boldsymbol{trascurabile}$$

Esempio di MAC insicuro: supponendo A sia in grado di trovare

$$m_0 \neq m_1 \; t.c. \;\; S(k, m_0) = S(k, m_1) \;\; per \frac{1}{2} delle \; chiavi \; k \in K$$

Basta che A invii m_0 e riceva, quindi, t. A questo punto inviando una coppia (m_1,t) ha il 50% di probabilità che sia valida.

$Adv_{MAC}[A, I] = P[b = 1] =$
$P[(m_1, t) sia \; valida] =$
$P[S(k, m_1) = t] =$
$P[S(k, m_0) = S(k, m_1)] = \frac{1}{2}$

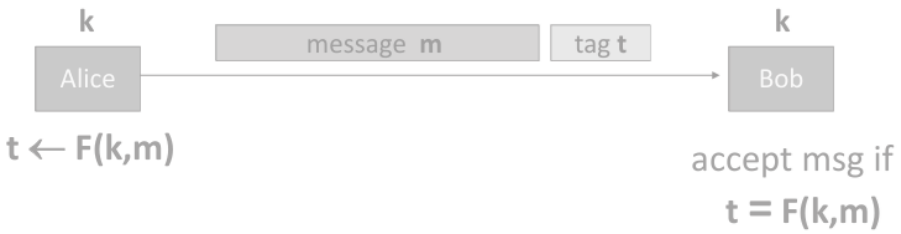


MAC BASATO SU PRF (PRF è un MAC).

Data una PRF $F: K \times X \rightarrow Y$ definisce un MAC $I_F = (S_F, V)$ come:

- $S_F(k, m) := F(k, m)$
- $V(k, m, t) = \begin{cases} yes & se \; t = F(k, m) \\ no & altrimenti \end{cases}$

Ovvero il tag *t* sarà la cifratura di *m* con *k* (ad esempio usando AES).



TEOREMA SULLA SICUREZZA

Se $F: K \times X \rightarrow Y$ è una **PRF sicura** e $\frac{1}{|Y|}$ è **trascurabile** (cioè |Y| è molto grande), allora I_F è un **MAC sicuro**.

In particolare,

$$\forall \; avversario \; efficiente \; A \; del \; MAC \; I_F$$
$$\exists \; avversario \; efficiente \; B \; della \; PRF \; F \; t.c.$$
$$\boldsymbol{Adv_{MAC}[A, I_F] \leq Adv_{PRF}[B, F] + \frac{1}{|Y|}}$$

➔ $\boldsymbol{I_F}$ **è sicuro fintanto che |Y| è molto grande** (ad esempio $|Y| > 2^{80}$).

➔ In altre parole, è importante che l'output della PRF, cioè l'insieme dei tag, sia molto grande. Ad esempio, una PRF $F: K \times X \rightarrow \{0,1\}^{10}$ genera un MAC insicuro dal momento che la probabilità (e quindi il vantaggio) per l'avversario di azzeccare il tag corretto è di $\frac{1}{2^{10}} = \frac{1}{1024}$, troppo alto.

Sebbene questa costruzione sia semplice da implementare e sicura, ha come limite principale il fatto che **può processare solo per blocchi** (detto **Small-MAC**), cioè, generare tag a blocchi (la PRF è un cifrario a blocchi).

Per generare tag per messaggi di lunghezza arbitraria (detto **Big-MAC**) si usano due tecniche principali:

1. ECBC-MAC
2. HMAC (protocolli Internet)

Entrambi consentono di convertire una small-PRF in una big-PRF (le due nozioni coincidono dal momento che, ottenuta una PRF sicura con output lunghi, posso facilmente ricavare il MAC corrispondente).

LEMMA

Data una PRF sicura $F: K \times X \rightarrow \{0,1\}^n$, allora anche la sua **troncatura** $F_W: K \times X \rightarrow \{0,1\}^W$ **è sicura**, ed è definita come $F_W(k, m) = F(k, m)[1 \ldots w]$ (cioè, contiene i primi *w-bit* dell'output di F).

Allora, se (S, V) è un MAC ottenuto a partire da F con *n-bit* di output, il **MAC troncato** con *w-bit* di output **è ancora sicuro**. Ovviamente, ciò vale fintanto che $\frac{1}{2^W}$ **è trascurabile**.

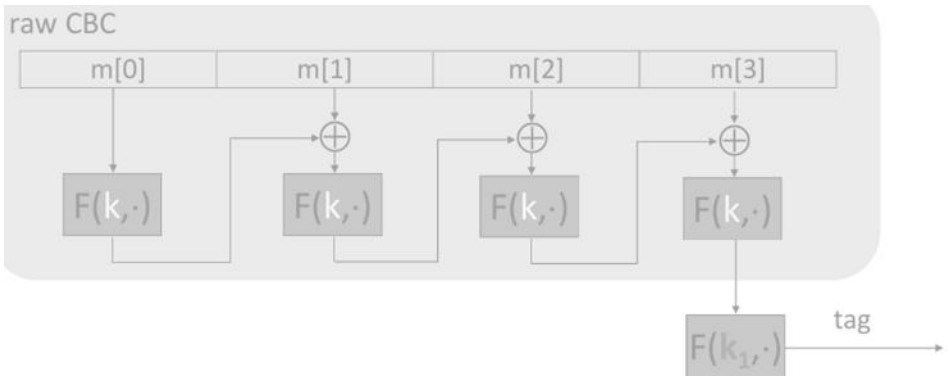
1. ENCRYPTED CIPHER BLOCK CHAINING MAC (ECBC-MAC)

Sia data una PRF $F: K \times X \to X$ (NOTA: dom(F)=cod(F)), definiamo una nuova PRF $F_{ECBC}: K^2 \times X^{\leq L} \to X$, e due chiavi (k, k_1) (è l'algoritmo di generazione dei tag)

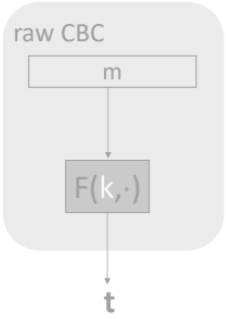
$$dove\ X^{\leq L} = \bigcup_{i=1}^{L} X^i$$

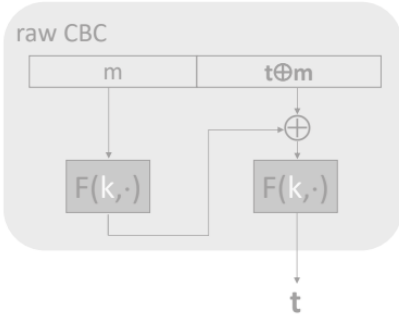
Allora, la generazione del tag avviene nel seguente modo:

1. Il messaggio è composto da un certo numero di blocchi la cui dimensione dipende dai blocchi della PRF che si ha a disposizione;
2. Ogni blocco viene dato in input a F(k, *) e il risultato va in XOR col blocco successivo;
3. Infine, il tag è ottenuto dando in input il risultato delle precedenti elaborazioni a F(k_1,*);



Se a questo ECBC-MAC viene tolto l'ultimo passo (F(k_1,*)) si ottiene un altro MAC: **raw CBC**, che è <u>insicuro</u>.





<u>PERCHE' RAW CBC È INSICURO?</u> Perché basta che A, inviato un messaggio m di un blocco e ottenuto il corrispondente tag t, manda una coppia $(m', t)\ t.c.\ \ m' = (m, m \oplus t)$ ovvero è composto da due blocchi e usa come tag quello precedente.
Ciò vale perché:

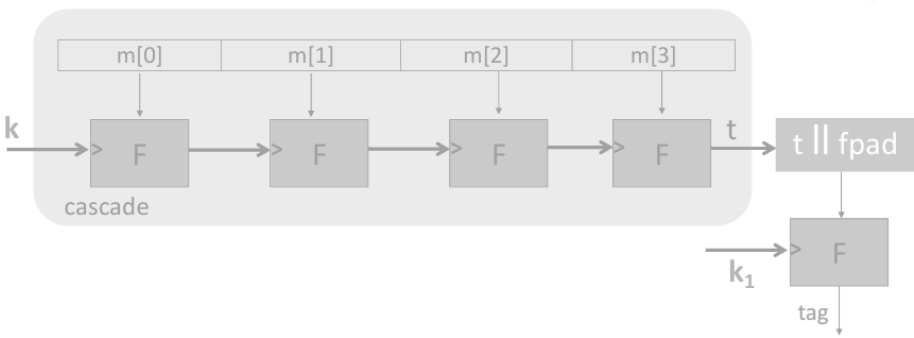
$$t' = F(k, m') = F\big(k, (m, m \oplus t)\big) = F\big(k, F(k,m) \oplus (m \oplus t)\big)$$
$$= F\big(k, t \oplus (t \oplus m)\big) = F(k, m) = t$$

Quindi, la coppia inviata è valida dal momento che il tag risultante t' so per certo che è un valore valido (=t). Adv=1 → è insicuro.

2. NMAC

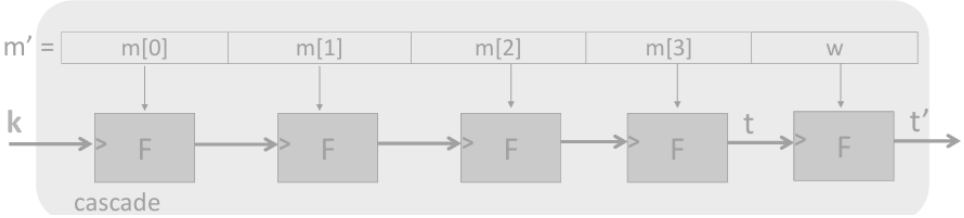
Sia data una PRF $F: K \times X \to k$ (NOTA: k= cod(F)), definiamo una nuova PRF $F_{NMAC}: K^2 \times X^{\leq L} \to K$, e due chiavi (k, k_1): il *tag sarà lungo quanto le chiavi*.
L'algoritmo procede nel seguente modo:

1. Ogni blocco del messaggio è dato in input a F(k,*);
2. Il risultato dell'invocazione viene dato in input alla funzione applicata al blocco successivo;
3. L'ultima elaborazione viene poi concatenata ad un *padding fissato* (**si suppone che la chiave sia più breve del blocco**, per questo è necessario concatenare *fpad*);
4. Infine, il tutto viene dato in input a (k_1,*) per ottenere il tag;



L'algoritmo senza fpad e l'ultima invocazione di F è un altro MAC: **cascade** ed <u>è insicuro</u>.

<u>PERCHE' CASCADE È INSICURO?</u> Perché basta che A, inviato un messaggio *m* e ottenuto il corrispondente tag *t*, manda una coppia $(m', t')\ t.c.\ \ m' = (m||w)$ con w qualunque e $t' = F(t, m')$ ovvero usa come chiave il risultato dell'elaborazione precedente su *m*. <u>nota:</u> F è pubblica! La chiave no.



Adv=1 → *Cascade è insicuro*.

TEOREMA SULLA SICUREZZA

ECBC-MAC: Per qualunque $L > 0$,

$$\forall\ avversario\ efficiente\ A\ della\ PRF\ F_{ECBC},$$
$$\exists\ avversario\ efficiente\ B\ che\ attacca\ la\ PRF\ sottostante\ F\ t.c.$$
$$\boldsymbol{Adv_{PRF}[A, F_{ECBC}] \leq Adv_{PRF}[B, F] + 2\frac{q^2}{|X|}}$$

Dove, q è il numero di messaggi per i quali è stato generato un tag usando la stessa chiave (cioè, quante volte è stata usata la stessa chiave per generare i tag).

Quindi, la PRF ECBC è sicura fintanto che $\boldsymbol{q \ll |X|^{\frac{1}{2}}}$

NMAC: Per qualunque $L > 0$,

$$\forall\ avversario\ efficiente\ A\ della\ PRF\ F_{NMAC},$$
$$\exists\ avversario\ efficiente\ B\ che\ attacca\ la\ PRF\ sottostante\ F\ t.c.$$
$$\boldsymbol{Adv_{PRF}[A, F_{NMAC}] \leq q \cdot L \cdot Adv_{PRF}[B, F] + \frac{q^2}{2|K|}}$$

Dove:

- L è il numero di blocchi del messaggio più lungo processato con la stessa chiave;
- K è l'insieme delle chiavi della PRF utilizzata all'interno di NMAC

Quindi, la PRF NMAC è sicura fintanto che $\boldsymbol{q \ll |K|^{\frac{1}{2}}}$

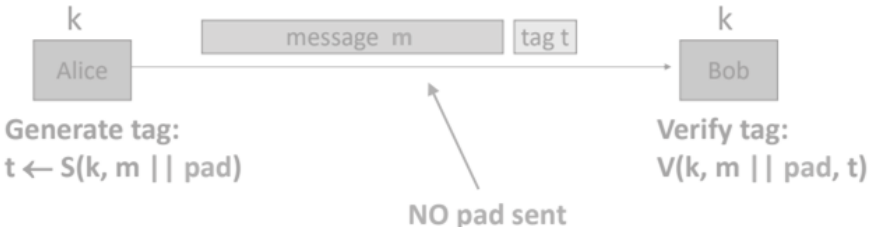
NOTA: il vantaggio calcolato su ECBC/NAMC è quello della PRF poiché ECBC/NMAC è una PRF e se essa è sicura, allora anche il MAC che ne deriva è sicuro (a patto che i tag siano sufficientemente lunghi). Cioè, a valle di questo teorema va applicato quello sulla sicurezza del MAC.

CONFRONTO

ECBC-MAC	NMAC
Usato solitamente come un MAC basato su AES	Non è usato solitamente né con AES né con DES perché richiede di cambiare chiave e key expansion su ogni blocco
È alla base di una variante standardizzata dal NIST: CMAC	Comunemente usato come base per un altro MAC: HMAC

MAC PADDING

Se il messaggio non è un multiplo della dimensione del blocco, si ricorre al **padding:** esso non viene inviato tra sorgente e destinazione ma viene concatenato al messaggio in fase di generazione/verifica del tag dai due end point.



IDEA 1) padding di **soli 0**: è insicuro. Questo perché l'avversario, inviato *m* e ottenuto *t,* può sempre generare un nuovo messaggio *m'* ottenuto a partire da m e concatenando quanti 0 vuole, il nuovo tag è $t' = t$. Ciò accade perché $(m||pad) = (m||pad||\ 0)$ → Adv=1, insicuro.

PER GARANTIRE SICUREZZA, il **padding deve essere invertibile**: $m_0 \neq m_1 \rightarrow (m_0||pad) \neq (m_1||pad)$. Infatti, se : $m_0 \neq m_1 \rightarrow (m_0||pad) = (m_1||pad)$, allora A può trovare un messaggio m_1 tale da generare lo stesso padding di m_0

STANDARD ISO:

1. Se *len(m)* non è multiplo della dimensione del blocco, aggiungo un pad composto da 10…0;
2. Se *len(m)* è multiplo della dimensione del blocco, aggiungo un **dummy block** 10….0;

l'1 identifica l'inizio del padding.

Di fatto, se non venisse aggiunto il dummy block, il MAC sarebbe insicuro.

Perché A può inviare un primo messaggio m di due blocchi $m = (m[0], m[1])\ incompleto$, ottenuto il tag t e conoscendo lo standard ISO, genera un nuovo messaggio m' organizzato in due blocchi come segue: $m'(m'[0], m'[1])\ completo\ t.c.$

$$m'[0] = m[0] \quad e \quad m'[1] = m[1]||pad$$

➔ m e m' avranno allora lo stesso tag. Adv=1➔ insicuro.

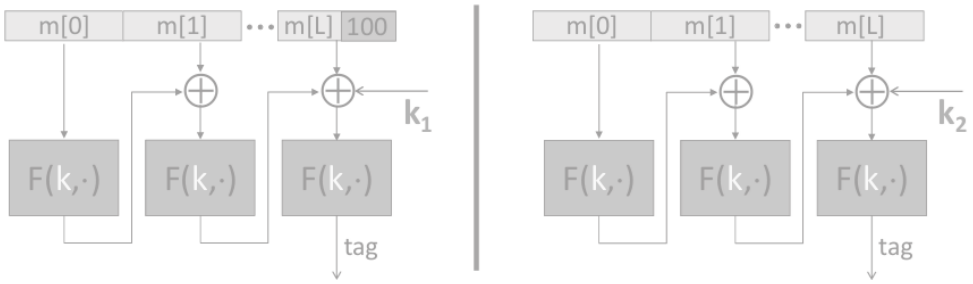
3. CMAC (Cipher-based MAC)

È una variante di ECBC-MAC ed introduce due miglioramenti:

1. Se l'ultimo blocco è completo, **non è necessario aggiungere il dummy block**;
2. **Non richiede l'ultimo passaggio crittografico di ECBC**;

Di fatto, fa uso di tre chiavi $(k, , k_1, k_2)$ con $k_1 e\ k_2$ derivate da k.

A sinistra, CMAC genera un tag per un blocco *incompleto*. In sintesi, se l'ultimo blocco richiede padding, si usa k_1, altrimenti k_2.

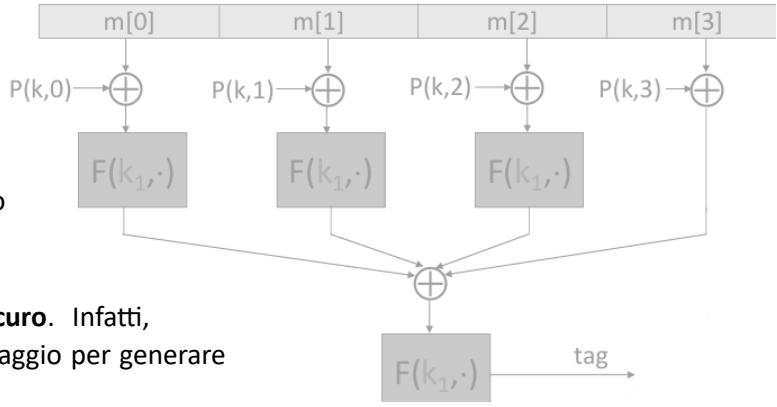


4. PMAC (Parallel MAC)

È un MAC **parallelizzabile** ed **incrementale**. La nascita di PMAC è legata al fatto che ECBC-MAC e NMAC sono sequenziali. Allora, data una PRF $F: K \times X \to X$, viene definita una nuova PRF $F_{PMAC}: K^2 \times X^{\leq L} \to X$ che prende in input una coppia di chiavi (k, k_1).

Inoltre, fa uso di una funzione $\boldsymbol{P(k, i)}$ definita nelle specifiche di PMAC. Allora, funziona nel seguente modo:

1. Ogni blocco m[i], <u>ad eccezione dell'ultimo blocco</u>, è messo in XOR con P(k,i) e l'output viene dato in input a $F(k_1,)$;
2. Il risultato di queste elaborazioni parallele viene poi dato in input a $F(k_1,)$ per ottenere il tag;



<u>NOTA:</u> se **P(k,i) non viene usato, allora PMAC è insicuro**. Infatti, basterebbe scambiare tra loro due blocchi interni del messaggio per generare due messaggi diversi a cui è associato lo stesso tag.

<u>SICUREZZA DEL PMAC</u>

Per qualunque $L \geq 0$, se F è una PRF **sicura** su (K, X, X), allora la PRF F_{PMAC} è **sicura** su $(K^2, X^{\leq L}, X)$. In particolare,

$$\forall\ avversario\ efficiente\ A\ della\ PRF\ F_{PMAC},$$
$$\exists\ avversario\ efficiente\ B\ della\ PRF\ F\ t.c.$$
$$\boldsymbol{Adv_{PRF}[A, F_{PMAC}] \leq Adv_{PRF}[B, F] + \frac{2q^2L^2}{|X|}}$$

➔ X è il numero di blocchi nella PRF, ed è costante.

Dunque, PMAC è sicuro fintanto che $\boldsymbol{qL \ll |X|^{\frac{1}{2}}}$.

Per rendere PMAC **incrementale serve che F sia una PRP**: non basta più che sia una PRF.

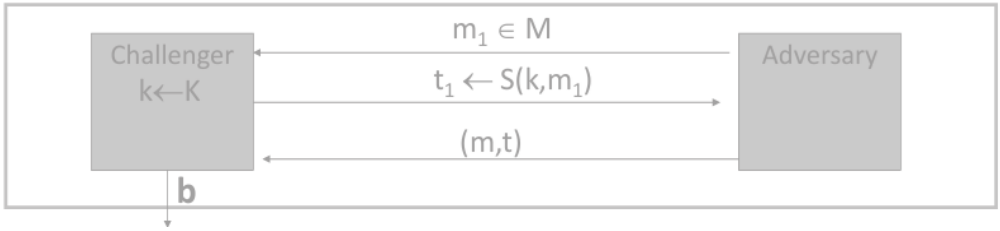
Può essere utile renderlo incrementale perché, se è stato calcolato il tag t per un messaggio m, all'arrivo di un messaggio *m' molto simile ad m* piuttosto che calcolare *t'* da zero, si può ottenere *t'* più velocemente sfruttando t.

Supponendo di avere $m = m[0]m[1]m[2]m[3]$ per il quale ottengo t. Allora, all'arrivo di $m' = m[0]m'[1]m[2]m[3]$ ottengo il tag corrispondente nel seguente modo:

$t' = F\big(k_1, (\ F^{-1}(k_1, t) \oplus \mathrm{F}\big(\mathrm{k}_1, \mathrm{m}[1] \oplus \mathrm{P}(\mathrm{k}, 1)\big) \oplus F\big(k_1, m'[1] \oplus P(k, 1)\big)\big)\big)$ mettendo in XOR la parte non in corsivo si elimina dalla computazione finale la cifratura del blocco m[1] (unico variato in *m')*, alla quale va quindi poi aggiunta la computazione di *m'[1]*.

5. ONE-TIME MAC (analogo di OTP)

In un One-Time MAC la chiave può essere usata solo *una volta per tag*. Allora, per un MAC one-time key $I = (S, V)$ si definisce un avversario A e un gioco del tipo:



1. A invia un messaggio m_1 a C;
2. C sceglie una chiave *k* e genera il tag t_1;
3. A deve generare una coppia *(m,t)* e la invia a C;
4. C risponde con $b = \begin{cases} 1 \; se \; V(k,m,t) =' yes' \; e \; (m,t) \neq (m_1,t_1) \\ 0 \; altrimenti \end{cases}$

Allora, $I = (S, V)$ è un **MAC sicuro** se per ogni avversario efficiente A, $\boldsymbol{Adv_{1MAC}[A,I] = P[b = 1]}$ è $\boldsymbol{trascurabile.}$

Esempio di 1MAC sicuro contro tutti gli avversari e molto più veloce dei MAC PRF-based.

- si sceglie un numero primo *q* molto grande.
- Come chiave $key = (a, b) = \{1, \dots, q\}^2$ ovvero due interi casuali compresi tra 1 e q.
- Come messaggio $msg = (m[1], \dots, m[L])$ con ogni blocco da 128bit.
- $S(key, msg) = P_{msg}(a) + b \; (mod \; q)$

Dove $P_{msg}(x) = x^{L+1} + m[L] * x^L + \cdots + m[1] * x$ → ovvero è un polinomio di grado L+1 i cui coefficienti sono i blocchi di msg. Ogni blocco è dunque visto come un numero intero.

6. CARTER-WEGMAN MAC → DA ONE-TIME MAC A MANY-TIME MAC

Sia (S, V) un 1MAC definito sulla tripla $\{K_I, M, \{0,1\}^n\}$, dove $\{0,1\}^n$ è l'insieme dei tag.
Sia $F: K_F \times \{0,1\}^n \rightarrow \{0,1\}^n$ una PRF, allora il MAC di **Carter-Wegman** è definito come:

$$\boldsymbol{CW\big((k_1,k_2),m\big) = \big(r, F(k_1,r) \oplus S(k_2,m)\big)}$$

Dove:

- $r \leftarrow \{0,1\}^n$ fa parte del tag ed è casuale. Fa parte del tag perché serve alla verifica;
- $F(k_1, r)$ è lento ma con input piccolo, produce un BLOCCO;
- $S(k_2, m)$ è veloce ma ha input grande, produce un TAG→ Si può usare anche 1MAC veloce (come quello sopra);

➔ Il mac di Carter-Wegman è quindi un **MAC randomizzato** costruito a partire da un **1MAC veloce**.

TEOREMA

Se (S, V) è un **1MAC sicuro** e F è una **PRF sicura**, allora CW è un **MAC sicuro** che produce in output tag $t \in \{0,1\}^{2n}$.

HASH- MAC (HMAC)

COLLISION RESISTANCE

Una **funzione hash** è una funzione $H: M \rightarrow T \;\; con \; |M| \gg |T|$
Una **collisione** per H è una coppia $m_0, m_1 \in M, m_0 \neq m_1 \;\; t.c. \; H(m_0) = H(m_1)$

Una funzione hash H è **resistente alle collisioni** se

$$\forall \; avversario \; (algoritmo) \; esplicito \; efficiente \; A,$$
$$\boldsymbol{Adv_{CR}[A,H] = P[A \; trova \; una \; collisione \; per \; H]} \text{ è } trascurabile$$

è impossibile trovare una H senza collisioni dal momento che il dominio è di gran lunga maggiore in cardinalità del codominio → principio della piccionaia.
NOTA: *esplicito* indica che l'algoritmo non solo è efficiente ma inoltre va costruito.

DA SMALL MAC A BIG MAC PER MEZZO DI H

Sia $I = (S, V)$ un MAC (*small MAC*) per piccoli messaggi sulla tripla $(K, \boldsymbol{M}, T)$, ad esempio AES.
Allora sia $H: M^{big} \rightarrow \boldsymbol{M}$ una funzione hash che mappa *messaggi grandi in messaggi più piccoli*. L'obiettivo è costruire un big-MAC, in grado, cioè, di processare messaggi di qualunque dimensione.

Definiamo allora $I^{big} = (S^{big}, V^{big})$ su (K, M^{big}, T) nel seguente modo:

- $S^{big}(k, m) = S(k, H(m))$, posso passare H(m) come parametro a S poiché il codominio di H coincide con l'insieme dei messaggi di I.
- $V^{big}(k, m, t) = S(k, H(m), t)$

TEOREMA

Se I è un **MAC sicuro** e H è **resistente alle collisioni**, allora I^{big} è a sua volta un **MAC sicuro**.

Il fatto che H sia resistente alle collisioni è **necessario per la sicurezza**. Infatti, se non dovesse essere rispettata questa condizione, si avrebbe che A è $in\ grado\ di\ trovare\ una\ collisione\ con\ probabilità\ non\ trascurabile$.
A questo punto, all'avversario A basterebbe trovare due messaggi diversi (m_0, m_1) tali da generare lo stesso tag (e lo fa con probabilità non trascurabile). Ovvero:

1. A invia a C un messaggio m_0;
2. C invia t;
3. A invia la coppia (m_1, t);

➔ La coppia è sicuramente accettata da C → Adv=P[A trovi la collisione] → Adv non trascurabile

ATTACCO GENERICO CONTRO QUALUNQUE FUNZIONE HASH

Sia $H: M \to T$ una funzione hash tale che $|M| \gg 2^n > 2^n$. Un generico algoritmo è in grado di trovare una collisione in tempo $\boldsymbol{O}(2^{\frac{n}{2}})$.
L'algoritmo funziona nel seguente modo:

1. Sceglie $2^{\frac{n}{2}}$ messaggi casuali in M: $m_1, \dots, m_{2^{\frac{n}{2}}}$ distinti con elevata probabilità;
2. Per $i = 1, \dots, 2^{\frac{n}{2}}$ calcola $t_i = H(m_i) \in \{0,1\}^n$;
3. Cerca una collisione $t_i = t_j$. Se non la trova, torna al punto 1;

La probabilità $P[\ algoritmo\ trova\ una\ collisione\ in\ una\ sola\ iterazione] \geq \frac{1}{2}$
Per dimostrare ciò, si fa riferimento al paradosso del compleanno. Ricordando che:
Siano $r_{1,\dots,}r_m \in U$ delle variabili aleatorie indipendenti *identicamente distribuite* (hanno la stessa distribuzione), allora

$$se\ m = 1{,}2 \times |U|^{\frac{1}{2}} \qquad allora \qquad P(\exists\ i \neq j :\ r_i = r_j) \geq \frac{1}{2}$$

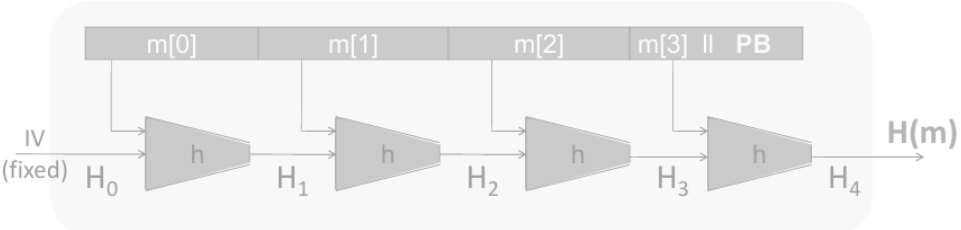
Definiamo allora $H: M \to \{0,1\}^n$
Ad ogni iterazione pesco $2^{\frac{n}{2}}$ messaggi e calcolo i corrispondenti valori hash $t_i = H(m_i)$, ognuno dei quali può assumere 2^n valori (il codominio di H è $\{0,1\}^n$).
Considerando i valori hash t_i come variabili aleatorie (nell'ottica del paradosso del compleanno), devo chiedermi se $m = 1.2 * U^{\frac{1}{2}}?\ \rightarrow\ m = \sqrt{U}$, con $U = 2^n$ e $m = 2^{\frac{n}{2}}$.

$$\blacktriangleright\ 2^{\frac{n}{2}} = \sqrt{2^n} \rightarrow\ 2^{\frac{n}{2}} = 2^{\frac{n}{2}}$$

$$mediamente\ in\ 2\ iterazioni\ si\ trova\ la\ collisione \rightarrow O(2^{\frac{n}{2}})$$

PARADIGMA DI MERKLE-DAMGARD

Consente di definire un metodo iterativo per la **costruzione di una funzione big-hash a partire da una small-hash**. Data una **funzione di compressione** $\boldsymbol{h}: \boldsymbol{T} \times \boldsymbol{X} \to \boldsymbol{T}$, cioè una funzione hash con due input di dimensione prefissata, è possibile ottenere $H: X^{\leq L} \to T$. Lo usa SHA-256.



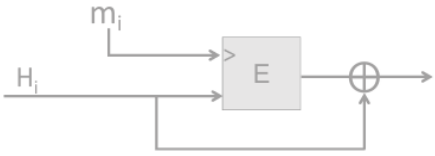
- IV è **fissato**. Viene dato in input insieme al primo blocco a *h*;
- L'output viene dato in input insieme al blocco successivo ad *h*;
- Così via;

NOTA: il **padding block è sempre aggiunto**.
PB = bit mancanti al completamento del blocco + codifica in 64bit del numero di blocchi del messaggio (per l'esempio sono 4 blocchi, quindi 4 rappresentato in 64bit).
Se non c'è spazio per inserire, oltre al padding, *msg len*, viene aggiunto un altro blocco che negli ultimi 64 bit ha *msg len* e in tutti gli altri solo 0….0.

TEOREMA: Se *h* è **resistente alle collisioni**, allora *H* è a sua volta **resistente alle collisioni**.
Quindi, per costruire una funzione big-hash resistente alle collisioni è sufficiente costruire una funzione di compressione resistente alle collisioni.

COME COSTRUIRE UNA FUNZIONE DI COMPRESSIONE C.R. (collision resistant)

1) Funzione di **Davies-Meyer**→ Sia $E: K \times \{0,1\}^n \to \{0,1\}^n$ un cifrario a blocchi, allora la funzione è definita come $\boldsymbol{h(H,m) = E(m,H) \oplus H}$



NOTA: m_i e H_i fanno riferimento a due generici input nel paradigma di Merkle-Damgard.

2) Funzione di **Miyaguchi-Preneel**→ Sia $E: \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ un cifrario a blocchi, allora la funzione è definita per mezzo di diverse varianti (12 in totale). Ad esempio:
   a. *Whirlpool*→ $\boldsymbol{h(H,m) = E(m,H) \oplus H \oplus m}$
   b. $\boldsymbol{h(H,m) = E(H \oplus m, m) \oplus m}$

3) Si sceglie un *numero primo casuale* ***p*** *di 2000bit* e due *numeri casuali* $\boldsymbol{u \geq 1, v \leq p}$.
   Allora, per $m, H \in \{0, \dots, p-1\}$ si definisce la funzione di compressione come: $\boldsymbol{h(H,m) = u^H \cdot v^m \cdot (mod\ p)}$.
   Questa funzione è sicura dal momento che trovare una collisione per essa ha una difficoltà pari al *problema del logaritmo discreto* (problema NP-hard). È però un **approccio lento**;

SHA-256

È una funzione hash utilizzata spesso e fa uso di:

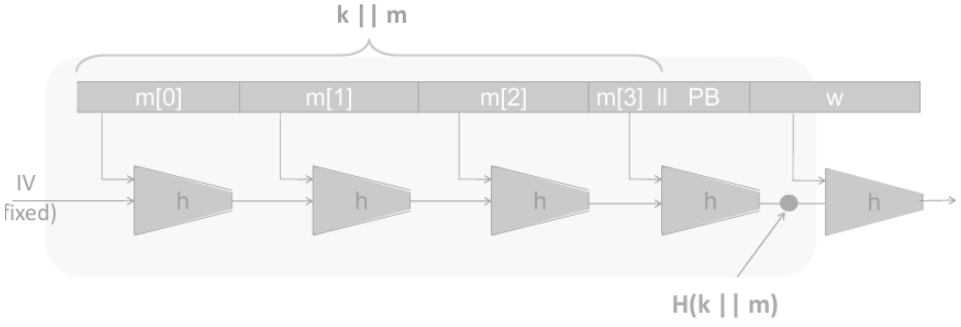
- *Paradigma di* ***Merkle-Damgard****;*
- *Funzione di compressione di* ***Davies-Mayer****;*
- *Cifrario a blocco* ***SHACAL-2****;*

HMAC

Avendo una funzione hash implementata per mezzo del paradigma di Merkle-Damgard, come usarla per generare un MAC? Avendo $H: X^{\leq L} \to T$, funzione hash di Merkle-Damgard C.R.

Proposta 1) $S(k,m) = H(k \parallel m)$

è **insicuro** perché dato $t = H(k \parallel m)$ e per il messaggio m, l'attaccante può generare $t' = H(k \parallel m \parallel PB \parallel w)$ e ottenere quindi il tag per il nuovo messaggio $m' = (m \parallel PB \parallel w)$.

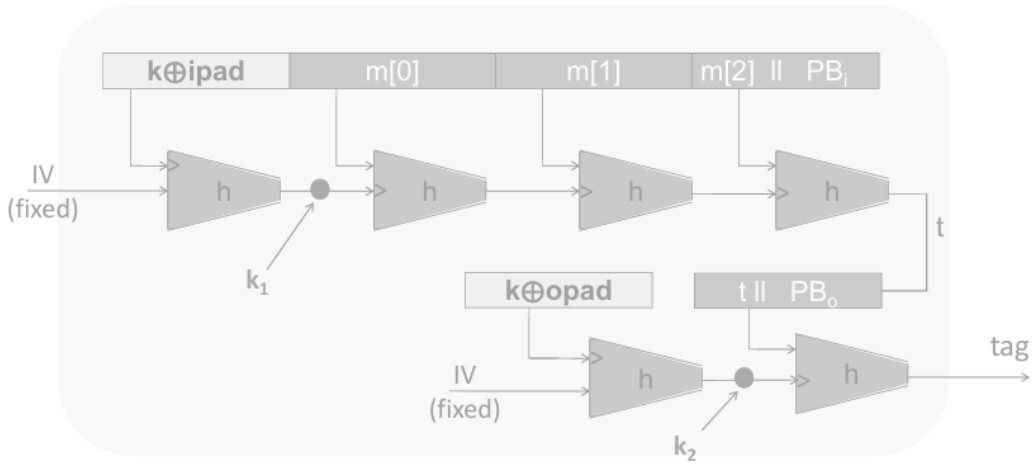


L'approccio realmente utilizzato, invece, è il seguente:

$$HMAC: \boldsymbol{S(k,m) = H(k \oplus opad \parallel H(k \oplus ipad \parallel m))}$$

È usato in SHA-256.

è *molto simile a NMAC* ma le differenze sostanziali stanno nell'utilizzo di **due chiavi dipendenti** $(\boldsymbol{k_1}, \boldsymbol{k_2})$ e nell'utilizzo di una **funzione di compressione** (al posto della PRF).
IV, ipad, opad sono *sequenze prefissate*.
Inoltre, affinchè HMAC sia **sicuro,** è necessario che $\frac{q^2}{|T|}$ sia trascurabile, cioè che $\boldsymbol{q^2 \ll |T|^{\frac{1}{2}}}$**.**



AUTHENTICATED ENCRYPTION

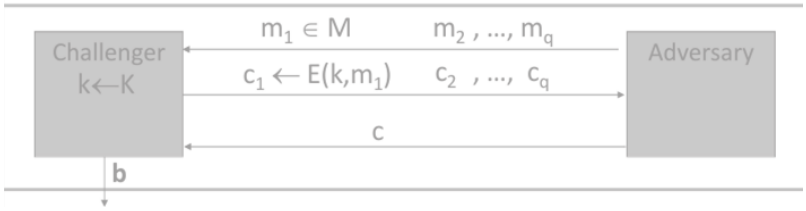
In questa fase si vuole garantire **sia riservatezza che integrità**. Qui l'avversario può *alterare, modificare, bloccare e generare traffico*.
Un **sistema di authenticated encryption (AE)** è un cifrario atto a garantire sia sicurezza che integrità ed è definito da una coppia di algoritmi efficienti **(E, D)** tali che:

- $E: K \times M \times N \rightarrow C$
- $D: K \times C \times N \rightarrow M \cup \{\perp\}, \quad con \ \perp \notin M$

Dove:
➔ ⊥ (*bottom*) definisce il **rifiuto del CT** poiché non è integro;
➔ N è il nonce, ma potrebbe anche essere assente;
Anche in questo caso E *cifra* e D *decifra* ma D potrebbe restituire **PT oppure** ⊥.

SICUREZZA
La sicurezza di questo cifrario è perseguita per mezzo di **sicurezza semantica sotto CPA attack** e **ciphertext integrity**.

CIPHERTEXT INTEGRITY
L'avversario non deve essere in grado di generare nuovi CT che il cifrario decifra correttamente. Formalmente,
sia Q=(E,D) un cifrario authenticated encryption.



1. A invia una serie di messaggi $m_1,\ldots, m_q$ a C;
2. C restituisce i CT $c_1,\ldots,c_q$ usando la stessa chiave k;
3. A invia un CT da lui generato, *c*;

$$b = \begin{cases} 1 & se\ D(k,c) \neq \perp \ \wedge \ c \neq \{c_1, \ldots, c_q\} \\ 0 & altrimenti \end{cases}$$

Allora, Q ha **ciphertext integrity** se

$$\forall\ avversario\ efficiente\ A,$$
$$\boldsymbol{Adv_{CI}[A,Q] = P[b = 1]} \quad \text{è } \boldsymbol{trascurabile}$$

Dunque, un cifrario (E, D) fornisce *Authenticated Encryption* (AE) se possiede:

1. Sicurezza semantica sotto CPA;
2. Ciphertext integrity;

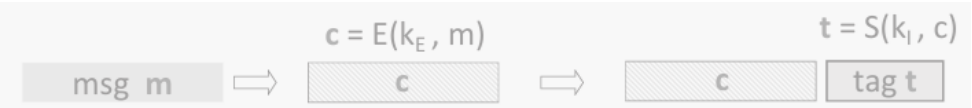
COSTRUZIONE PER MEZZO DI CIFRARI E MAC

Originariamente, si tentava di perseguire AE *combinando insieme cifrari e MAC* senza realmente sapere se fossero sempre sicuri o meno. Considerando un cifrario **(E,D)**, una chiave di cifratura $\mathbf{k_E}$, un MAC **(S,V)** e una chiave del MAC $\mathbf{k_I}$, I principali approcci sono stati:

1. **MAC-then-Encrypt (SSL)**

2. **Encrypt-then-MAC (IPsec)**

L'unico che è *sempre sicuro*.



3. **Encrypt-and-MAC (SSH)**

TEOREMI SU AE

Sia (E, D) un cifrario **sicuro sotto CPA** e (S, V) un MAC **sicuro**, allora:

- Mac-then-Encrypt: *potrebbe essere insicuro*;
- Encrypt-then-MAC: *è sempre sicuro*;
- Encrypt-and-MAC: *potrebbe essere insicuro*. Potrebbe accadere perché calcolare il tag sul messaggio in chiaro consente ad alcuni bit del testo in chiaro di essere presenti nel tag, causando un *leak* di informazioni;

Ad ogni modo, utilizzare rand-CTR o rand-CBC associato a MAC-then-encrypt garantisce AE.

Altri standard molto comuni sono:

1. **GCM→** CTR mode encryption e poi CW-MAC;
2. **CCM→** ECBC-MAC e poi CTR mode encryption;
3. **EAX→** CTR mode encryption e poi CMAC;

Esiste inoltre una variante di AE: **Authenticated Encryption with associated data**, in cui solo una parte del messaggio è crittografata ma per tutto il messaggio si garantisce integrità. Esempio: pacchetti internet (payload e header: per entrambi si vuole garantire integrità ma solo per il payload è richiesta riservatezza).

TRANSPORT LAYER SECURITY PROTOCOL (TLS)

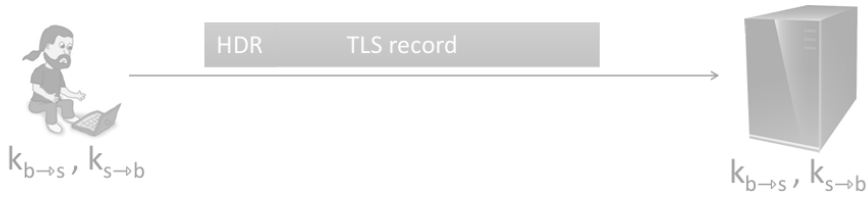
È un protocollo che regola la comunicazione tra un *browser* ed un *server*. È un MAC-then-encrypt: calcola il tag con **HMAC-SHA1** e poi cripta con **CBC AES-128.**

Vengono usate **due chiavi** che stanno sia sul browser che sul server:

1. $\boldsymbol{k_{b\to s}}$, per gestire il traffico dal browser al server: è usata dal browser per **preparare i dati** da inviare al server;
2. $\boldsymbol{k_{s\to b}}$**,** per gestire il traffico dal server al browser: è usata dal server per **preparare i dati** da inviare al browser (sia crittografare che autenticare);

Sia B che S mantengono **due contatori** a 64 bit (tot. 4 contatori):

1. $\boldsymbol{ctr_{b\to s}}$
2. $\boldsymbol{ctr_{s\to b}}$



Inizialmente sono a 0 (inizio sessione).

Quando B invia un pacchetto ad S, incrementa la propria copia locale di $ctr_{b\to s}$. Quando riceve da S, incrementa $ctr_{s\to b}$.

Quando S riceve un pacchetto da B, incrementa la propria copia locale di $ctr_{b\to s}$.Quando invia un pacchetto a B, incrementa $ctr_{s\to b}$

L'obiettivo è difendersi dal **replay attack**.

Esempio di comunicazione da B a S. Stabilito $\boldsymbol{k_{b\to s} = (k_{mac}, k_{enc})}$**,**

LATO BROWSER: B procede nel seguente modo: $\boldsymbol{enc(k_{b\to s}, data, ctr_{b\to s})}$**.** Si costruisce il pacchetto come segue:

1. $tag \leftarrow S(k_{mac}, [++ ctr_{b\to s} \parallel header \parallel data])$;
2. $pad\ [\ header \parallel data \parallel tag]$, detto **record**, per raggiungere la dimensione di AES-128;
3. Si cripta con $CBC, k_{enc}, random\ IV$;
4. Si concatena l'header;

LATO SERVER: S procede nel seguente modo: $\boldsymbol{dec(k_{b\to s}, record, ctr_{b\to s})}$**.** Poi:

1. CBC decifra il record con k_{enc};

2. Controlla il formato del pad: se non è corretto viene inviato il messaggio di errore **bad_record_mac** al mittente e si blocca la comunicazione;
3. Controlla la validità del tag su $[++ ctr_{b\to s} \parallel header \parallel data]$: se non è valido, invia **bad_record_mac** e interrompe la comunicazione;

In caso di comunicazione interrotta, sarà necessario stabilire nuove chiavi e generare nuovi record.
La procedura è pressocché equivalente nel caso di messaggio inviato da S a B (ovviamente non si userà $k_{b\to s}, ctr_{b\to s}$ ma $k_{s\to b}, ctr_{s\to b}$).
Alcuni bug nelle versioni precedenti erano:

- **Padding oracle**, in cui venivano inviati messaggi di errore diversi in caso di invalidità di pad o di mac: ciò forniva informazioni sul PT all'avversario. SOLUZIONE: inviare lo stesso messaggio di errore *bad_record_mac* in entrambi i casi.
- **IV predicibile**;

SCAMBIO DELLE CHIAVI

Le possibili implementazioni sono:

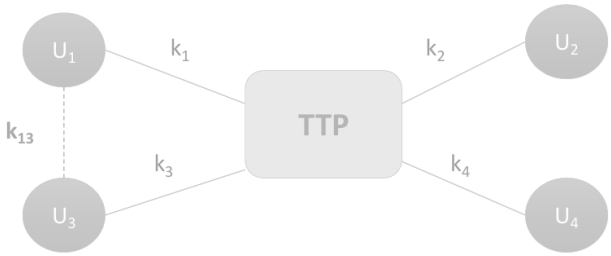
1. Affidarsi a **entità di terze parti**;
2. Usare **puzzle di Merkle**;
3. Usare il **Protocollo di Diffie-Hellman**;

Questi hanno come obiettivo *consentire a due interlocutori di scambiarsi delle chiavi in modo tale da potersi scambiare dei messaggi*.

Il problema principale nella crittografia simmetrica è lo **scambio delle chiavi**. Inoltre, per *n* utenti ogni utente dovrà memorizzare *n-1* chiavi e saranno necessarie $O(n^2)$ chiavi in totale, due per ogni coppia di utenti: **non è scalabile**.

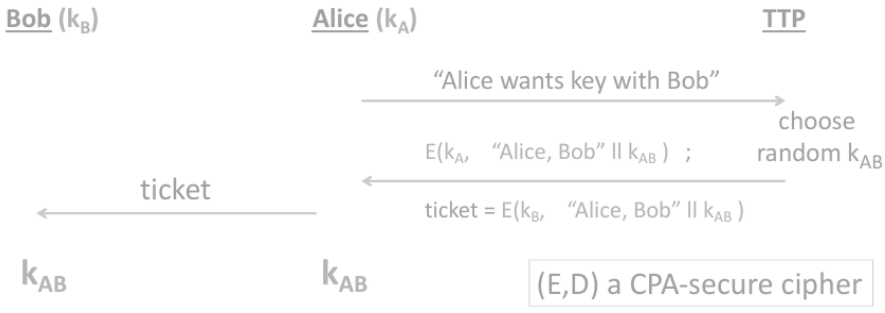
ENTITA' TERZE

La **Trusted 3rd Party (TTP)** rappresenta una prima soluzione per il problema della gestione delle chiavi: quest'entità centrale si occupa di *generare chiavi di sessione* per ogni utente e *tenere traccia delle chiavi* per ogni utente. **Ciascun utente ha una sola chiave**, usata per gestire il traffico con TTP.
Quando due utenti vogliono comunicare tra di loro, avvisano il TTP per la generazione della loro chiave di sessione.



SEMPLICE PROTOCOLLO PER LA GENERAZIONE DELLE CHIAVI

PREMESSA: serve avere un **cifrario sicuro**. NOTA: k_A (k_B) sono possedute sia da A (B) che da TTP.



A vuole iniziare una connessione con B:

1. chiede a TTP di generare una chiave di sessione;
2. TTP sceglie una chiave casuale e la invia ad A usando un messaggio cifrato con k_A. Inoltre, invia anche un **ticket** cifrato con k_B e contenente la chiave di sessione;
3. A invia il ticket a B, il quale lo decifra, ottenendo così la chiave di sessione;

Questo protocollo è **sicuro contro attaccanti passivi**. CONTRO: TTP è il **punto debole del sistema**. Infatti, TTP deve essere *affidabile e sempre disponibile*. Ad oggi, quindi, non è una soluzione attuabile.

PUZZLE DI MERKLE (1974)

Primo esempio di crittografia asimmetrica, **sicuro contro attaccanti passivi** e fa uso di un **cifrario simmetrico**.
L'aspetto chiave sono i cosiddetti **Puzzle**, ovvero testi cifrati ottenuti cifrando dei PT usando una chiave $k \in \{0,1\}^{128}$, i cui primi 96bit sono **sempre zero e i rimanenti casuali**.

Quando si parla di *risolvere un puzzle*, si intende *risalire al testo in chiaro*. Sapendo che i primi 96bit della chiave sono a 0, un attacco a forza bruta deve scandire solo 2^{32} possibili combinazioni.

A→ prepara 2^{32} puzzle nel seguente modo:

1. sceglie tre sequenze $P_i \in \{0,1\}^{32}$ e $x_i, k_i \in \{0,1\}^{128}$.
2. Genera il generico puzzle come $puzzle_i \leftarrow E(0^{96} \parallel P_i, \text{"Puzzle \#"} \parallel x_i \parallel k_i)$

A tiene traccia di una **tabella** in cui associa i valori di x_i, k_i al corrispondente puzzle i-esimo; in questa tabella, la prima colonna non ammette duplicati.
Infine, li invia a B.

x_i	k_i
…	…

B→ Sceglie uno dei puzzle in maniera casuale e lo **risolve** usando un *attacco a forza bruta*. Sa di aver ottenuto il PT corretto quando legge "Puzzle #".
Ottiene così (x_j, k_j) e usa k_j come **chiave condivisa.** Infine, invia x_j ad A.

A→ cerca nella propria tabella il valore k_j associato a x_j e usa il primo come chiave condivisa;

SICUREZZA

- A prepara 2^{32} puzzle, impiegando $O(2^{32})$. In generale, $\boldsymbol{O(n)}$**;**
- B impiega $O(2^{32})$ per risolvere un solo puzzle (sceglie casualmente quale, poi A si adatta). In generale, $\boldsymbol{O(n)}$**;**
- L'avversario dovrebbe tentare di risolvere tutti i puzzle nel caso peggiore, impiegandoci $O(2^{64})$. In generale, $\boldsymbol{O(n^2)}$**.**

Dunque, questo meccanismo è **insicuro** dal momento che $O(2^{64}) < O(2^{90})$: è basso.
Per renderlo sicuro si potrebbe **aumentare il numero di puzzle generati** e, di conseguenza, **diminuire il numero di bit fissati nella chiave**. Ad esempio, generando 2^{48} puzzle, i bit fissati sarebbero $0^{80} \parallel \cdots$.
Ovviamente, però, aumentando il numero di puzzle aumenta anche il lavoro di A e B: l'avversario ha un gap sul lavoro svolto che è solo **quadratico** rispetto a quello di A e B. L'ideale è renderlo quantomeno ***esponenziale***.

➔ È un algoritmo **inefficiente** perché **per renderlo sicuro deve essere aumentato non solo l'onere dell'avversario ma anche quello di A e B.**

PROTOCOLLO DI DIFFIE-HELLMAN

PREMESSA: **non ha bisogno di alcuna primitiva crittografica**, al contrario dei puzzle di Merkle, infatti, non fa uso di cifrari simmetrici.

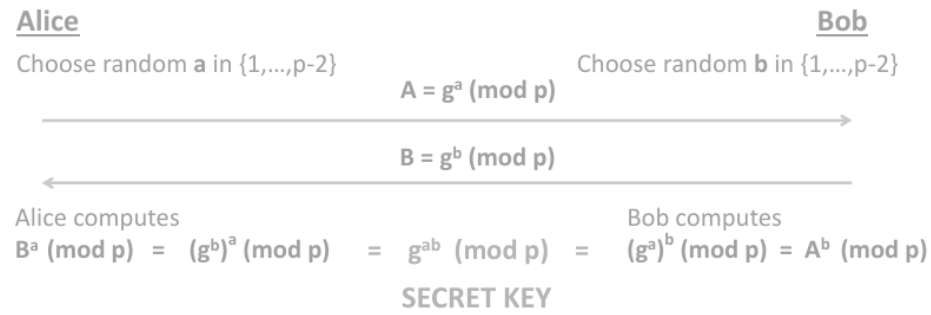
È ritenuto **sicuro** dal momento che è basato sul **Problema del Logaritmo Discreto**: dati $g, p, g^k mod\ p$ si vuole trovare *k*.

Allora, funziona nel seguente modo:

1. Viene scelto un numero primo **p** molto grande;
2. Viene scelto un numero interno **g** compreso tra (2, …, p-2);

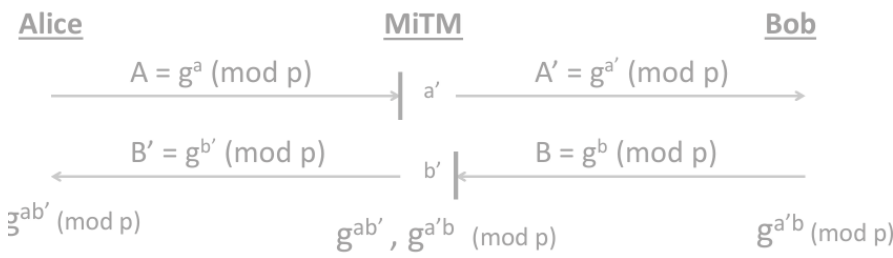
A questo punto:

NOTA: p e g sono **pubblici!**



SICUREZZA: Supponendo *p* sia di *n bit*, il miglior algoritmo noto per trovare la chiave segreta ci impiega **tempo esponenziale** su *n*. È quindi **sicuro contro attaccanti passivi**.

Ad ogni modo, è **insicuro per attaccanti attivi**, ad esempio contro **man-in-the-middle attack**.
In questo attacco, infatti, l'avversario può inviare un proprio valore di *a'* e *b'*, in modo tale da poter ottenere in modo totalmente autonomo la chiave di sessione.



CRITTOGRAFIA ASIMMETRICA

Come si è visto, i cifrari simmetrici avevano come obiettivo principale la **riservatezza** e richiedevano che mittente e destinatario **condividessero una stessa chiave**. La nascita della crittografia simmetrica si lega alla domanda se *sia possibile comunicare in modo sicuro senza far uso di chiavi condivise*.
Nella crittografia asimmetrica ogni utente possiede una **coppia di chiavi**: ***chiave pubblica*** *e* ***chiave privata***. La chiave pubblica è nota a tutti.

Ogni utente può, con la propria chiave:

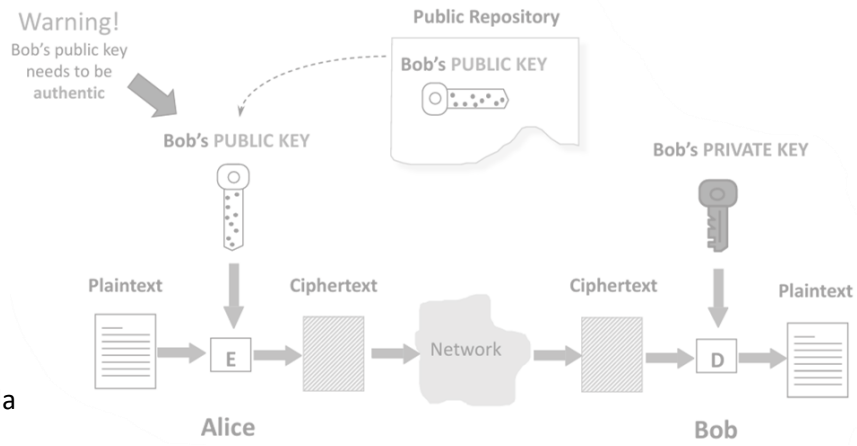
- *Privata*, cifrare/decifrare i propri messaggi;
- *Pubblica,* decifrare/cifrare i propri messaggi;

Nonostante le due chiavi siano, di fatto, legate tra di loro, **deve essere difficile risalire alla chiave privata data quella pubblica**. Altrimenti, un avversario potrebbe facilmente **impersonare** qualcun altro.

SCENARIO DI RISERVATEZZA

Anche in questo caso valgono le stesse considerazioni fatte per la crittografia simmetrica:

- Gli algoritmi sono pubblici;
- Le chiavi private devono rimanere segrete;



Per garantire riservatezza, si procede nel seguente modo:

1. A vuole inviare un messaggio a B: lo *cifra con la chiave pubblica di B*;
2. B riceve il messaggio cifrato e per decifrarlo usa la *propria chiave privata*;

È molto importante che sia **possibile verificare l'autenticità delle chiavi pubbliche**. Si useranno allora le **Certification Authorities** (CAs).

RSA

Questo algoritmo specifica come *generare le chiavi* e come procedere con *cifratura* e *decifratura*.
In modo molto blando, si può affermare che RSA lavora con **numeri interi non negativi**.

GENERAZIONE DELLE CHIAVI

B vuole generare la propria coppia di chiavi:

1. Genera $p, q \;\; numeri\ primi\ molto\ grandi\ t.c. p \neq q$
2. Calcola $n = p \times q$
3. Sia $\boldsymbol{\varphi(n)} = (p-1) \times (q-1)$
4. Sceglie $\boldsymbol{e\ numero\ casuale}\ t.c. 1 < e < \varphi(n) \;\; and \;\; \gcd(e, \varphi(n)) = 1$
5. Calcola $\boldsymbol{d} \;\; t.c. 1 < d < \varphi(n) \;\; and \;\; (e \times d) mod\ \varphi(n) = 1$. A differenza di *e*, ***d* è univoco**;
6. Pone come **chiave pubblica** $(\boldsymbol{e}, \boldsymbol{n})$
7. Pone come **chiave privata** $(\boldsymbol{d}, \boldsymbol{n})$ o anche solo **(d)** dal momento che solo *d* è realmente privato;

CIFRATURA (*confidentiality*)	**DECIFRATURA *(confidentiality)***
A vuole inviare un messaggio *m* a B: 1. Prende la chiave pubblica di Bob (e,n); 2. Calcola $\boldsymbol{c = m^e mod\ n}$**;** 3. Invia *c* a B;	B riceve *c* e lo decifra con $\boldsymbol{m = c^d mod\ n}$

NOTA: in RSA è necessario che ***m* sia un intero [0,n)**, cioè una sequenza di bit che codifichi un numero intero compreso tra 0 ed n (non compreso). Qualora il messaggio fosse **più lungo**, si usano dei meccanismi simili ai **modi operativi**.

SICUREZZA DI RSA

L'attaccante vuole *leggere m*: **attaccante passivo**.

L'avversario ha a disposizione (c, e, n) e vuole risalire ad *m*. Come può farlo?

1. Potrebbe tentare di *risalire alla chiave privata d*→ altamente improbabile;
2. Sapendo come funziona l'algoritmo di cifratura e sapendo che *m* è un intero in (0,n], tenta una *ricerca esaustiva*→ richiede troppo tempo se *n* è grande: dovrebbe provare a cifrare tutti i possibili *m* fino ad ottenere *c*;
3. Potrebbe provare a *calcolare la chiave privata usando (e,n)*→ sembra il più promettente, ma quanto è difficile ricavare la chiave privata? Il problema si riduce al **problema della fattorizzazione**, ovvero, dato $n = p \times q$, $trovare\ p, q$ numeri primi. Questo è un problema **NP-completo**. Non si sa se esista un algoritmo che fattorizzi in tempo polinomiale (in sintesi, P=NP).
4. Potrebbe provare a *risalire direttamente al testo in chiaro, senza passare dalla chiave privata,* detto **RSA problem**→ questo problema ha una complessità minore o uguale a quello della fattorizzazione. Ad oggi, il miglior algoritmo che lo risolve richiede sempre la fattorizzazione, ovvero ha complessità NP;
5. Potrebbe *inviare ad A una falsa chiave pubblica di B*→ è importante che A sia in grado di verificare l'autenticità della chiave pubblica utilizzata;

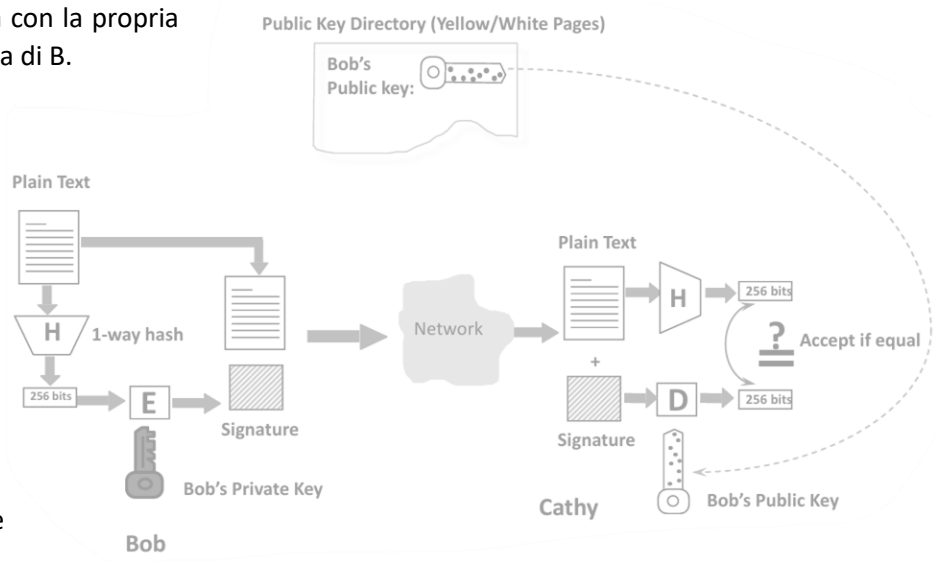
IN SINTESI: RSA ha come unico punto debole il fatto di essere **molto lento**. In pratica, molto spesso **crittografia simmetrica e asimmetrica sono usate insieme**: RSA per distribuire le chiavi e simmetrica per cifrare/decifrare.

FIRME DIGITALI

In questo scenario **non c'è interesse per la riservatezza**. In generale:

1. B invia *m* e *firma* a C;
2. C risale alla *firma* a partire da *m* e lo confronta con quella inviata da B. Se coincidono, c'è integrità;

NOTA: le chiavi di C non sono usate→ B firma con la propria chiave privata e C verifica con la chiave pubblica di B.



Realisticamente, la **firma** è ottenuta per mezzo di una **funzione hash**: il documento viene dato in input ad una funzione hash e la **firma viene calcolata sul documento hash**, non direttamente su *m.*

Il **confronto** è fatto quindi sul **valore hash**.

Inoltre, le funzioni hash in questo contesto sono molto utili poiché sono in grado di condensare documenti lunghi in documenti più brevi.

Spesso sono **funzioni one-way**, ovvero dotate nelle seguenti caratteristiche:

1. *semplici nella loro direzione naturale* (da x→ h(x));
2. *difficili* da ottenere nel verso opposto (h(x)→x);
3. *collision resistant*;

Un algoritmo one-way hash dovrebbe essere quindi:

1. veloce nel calcolo del valore hash;
2. difficile da invertire;
3. difficile trovare collisioni;
4. Tale per cui un piccolo cambiamento nell'input comporta notevoli cambiamenti nell'output;

Dunque, la firma digitale si articola in tre fasi:

1. Stabilire la coppia di chiavi pubbliche e private→ Bob sceglie la coppia e pubblica la chiave pubblica. La generazione avviene **una sola volta**;
2. Firma del documento → Bob firma il documento usando la sua chiave privata;
3. Verifica della firma→ A verifica la firma di B usando la sua chiave pubblica;

CERTIFICATE AUTHORITY & CERTIFICATES

Un **certificato** è un documento digitale che definisce il legame tra la **chiave pubblica** e **un'entità** ed è **firmato digitalmente da una CA.**
Una **Certificate Authority (CA)** è un'entità di terze parti che rilascia certificati.

Come fa A ad essere sicura di avere la reale chiave pubblica di B?

1. Recupera il certificato contenente la chiave pubblica di B;
2. A deve verificare la firma della CA→ Ha bisogno della chiave pubblica della CA, solitamente memorizzata nel browser o nel sistema operativo. Queste sono dette **root CA**;

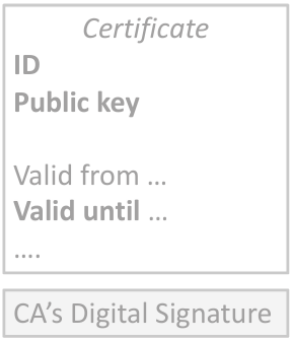
Se però la CA non è root, A può risalire ad una root CA che ha firmato quella di partenza. Ad esempio:

1. A ottiene la chiave di B firmata dalla CA XYZ;
2. XYZ non è root, ma ha un certificato firmato da una root CA;

L'idea è quella di risalire le CA finché non se ne incontra una root. La fiducia è quindi *transitiva*.

Il **modello di fiducia** prevede dunque che A *si fidi implicitamente di alcune chiavi*. Nel modello usato da SSL/TLS le **CA** sono organizzate in una **gerarchia**.

CICLO DI VITA DEL CERTIFICATO



1. **Rilascio del certificato**: si chiede ad una CA di rilasciare un certificato per la propria chiave pubblica (associazione ID/chiave pubblica);
2. **Rinnovo**: ogni certificato ha una **data di scadenza** per motivi di sicurezza; quando scade va rinnovato;
3. **Revoca**: consente di invalidare la chiave pubblica nel caso in cui venisse compromessa da esterni;
4. **Verifica**: controlla innanzitutto che il certificato non sia scaduto né revocato, per poi procedere con la verifica di validità vista fino a questo momento.

INFORMAZIONI SULLO STATO DEL CERTIFICATO

Due sono i meccanismi attraverso i quali verificare la revoca di un certificato:

1. **Certificate Revocation List (CRL)** → è una lista di certificati revocati che ogni utente è tenuto a scaricare. Ha come lato negativo il fatto che la lista potrebbe essere *estremamente lunga*, soprattutto se quello che mi interessa è solo uno specifico certificato; ad ogni modo, una possibile soluzione potrebbe essere quella di scaricare solo la porzione di lista non scaricata precedentemente: richiede dunque un aggiornamento periodico. Come lato positivo c'è il fatto che è memorizzata *in locale*, evitando dunque di inoltrare richieste a server;
2. **Online Certificate Status Protocol (OCSP)** → è un protocollo che *interroga un server* e chiede se *uno specifico certificato è stato revocato o meno*. Ha come lato negativo il fatto di fare affidamento ad un server: se il server è sovraccarico, non si ha modo di effettuare il controllo.
   NOTA: se il server è irraggiungibile, molti os danno per scontato che il certificato non sia stato revocato e vanno avanti.

Entrambi specificano *se un certificato è stato revocato* e, eventualmente, il *perché* e il *quando* della revoca.

ALTRI SISTEMI CRITTOGRAFICI (SIMMETRICI)

CIFRARIO DI RABIN

La **generazione** delle chiavi funziona nel seguente modo (per B):

1. Vengono generati due numeri *primi casuali* $p, q \ \ t.c. \ p\ (mod\ 4) = q\ (mod\ 4) = 3$
2. $n = p \times q$
3. $k^+ = (n)$, chiave pubblica
4. $k^- = (p, q)$, chiave privata

La **cifratura** avviene nel seguente modo: A vuole inviare *m* a B.

1. Prende la chiave pubblica di B: *n*;
2. Calcola $c = m^2 mod\ n$ → **è molto più veloce di RSA** perché *m* viene elevato al quadrato, non ad *e* (molto grande);
3. Invia *c* a B;

La **decifratura**→ B effettua dei calcoli su *c* usando (p,q) fino ad ottenere quattro numeri: $r, -r, s, -s$.
Viene garantito che *uno dei quattro numeri risultanti sia il testo in chiaro*→ **non è pratico** perché molto spesso il messaggio *m* è una sequenza casuale (come una chiave segreta), il che lo rende **indistinguibile dagli altri**, sbagliati.
Ha però alcuni vantaggi, come il fatto di essere **estremamente veloce** e **sicuro** dal momento che la complessità di ricavare *m* a partire da *c* è *uguale alla complessità della fattorizzazione* → *è più sicuro di RSA*.

CIFRARIO DI ElGamal

La **generazione** delle chiavi avviene nel seguente modo:

1. B sceglie un numero primo *p* e un numero *g* radice primitiva mod p;
2. B sceglie un numero casuale $a \in [0, p-2]$;
3. Calcola $A = g^a mod\ p$;
4. $k^+ = (p, g, A)$
5. $k^- = a$

Per la **cifratura**: A vuole inviare *m* e possiede la chiave pubblica di B (p,g,A)

1. Sceglie un numero casuale $b \in [0, p-2]$ → è una **cifratura randomizzata**;
2. Calcola $B = g^b mod\ p$;
3. Calcola $c = A^b m\ mod\ p$;
4. Il testo cifrato sarà la coppia (B, c), inviato a Bob;

Per la **decifratura**, B:

1. Calcola $x = p - 1 - a$
2. Calcola $m = B^x c\ mod\ p$

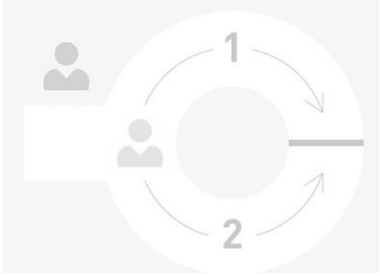
PRO: Questo cifrario è **sicuro** dal momento che è *tanto difficile quanto il problema del logaritmo discreto*.
CONTRO: il testo cifrato è **lungo il doppio** del testo in chiaro→ **non è efficiente**.
Inoltre, si noti come lo scambio delle chiavi si basa su **Diffie-Hellman**.

ZERO-KNOWLEDGE PROOF OF KNOWLEDGE

A (**Prover**) vuole convincere B (**Verifier**) di essere a conoscenza di un segreto, senza rivelarlo.
Questo tipo di scenario è molto utile nel contesto dell'**autenticazione**: con questo tipo di protocolli ci si potrebbe autenticare senza mai inviare la password. È importante che questi algoritmi rispettino tre proprietà:

- **Completezza (*Completeness*)**: dato un *prover onesto* e un *verifier onesto,* il protocollo ha successo;
- **Correttezza (*Soundness*)**: Se il prover non conosce il segreto, è altamente improbabile che riesca a convincere il Verifier;
- **Conoscenza Zero (*Zero Knowledge*)**: la prova fornita dal Prover non causa alcun leak di informazioni. In altre parole, né il Prover né il Verifier acquisiscono informazioni aggiuntive alla fine dell'algoritmo rispetto a quelle possedute all'inizio;

Esempio: LA CAVERNA DI ALI BABA



C'è una caverna a forma di cerchio con un'entrata su un lato e una porta magica che blocca l'altro lato. La porta può essere aperta solo con una parola magica.
Ci sono due attori: Ali Baba (prover) e noi (verifier). Ali Baba vuole convincerci di conoscere la parola segreta.
Procede allora nel seguente modo:

1. In maniera casuale e a noi sconosciuta, Ali Baba sceglie se entrare dal lato 1 o 2.
2. In maniera casuale, gli chiediamo di uscire da 1 o da 2;

Allora, se:

- Era entrato da 1 (2) e gli chiediamo di uscire da 1 (2), non ha problemi;
- Era entrato da 1 (2) e gli chiediamo di uscire da 2 (1), non ha problemi perché conosce la parola segreta;
- Non conosce la parola segreta e si trova sul lato sbagliato, rimane bloccato ed è costretto a tornare dal lato da cui era entrato;

Dunque, dal punto di vista del Verifier, se esce dal lato sbagliato→ non conosce la chiave.
Se però esce dal lato corretto→ non si può concludere nulla (potrebbe essere stato fortunato o conoscere realmente la chiave). In questo caso, scegliamo di *ripetere il gioco*.
Infatti, la probabilità che sia fortunato è ½ e ripetendo l'esperimento aleatorio *k* volte, la probabilità che sia davvero solo fortunato è pari a $\frac{1}{2^k}$, cioè molto bassa → probabilmente conosce davvero la chiave. Se invece ad un certo punto sbaglia, è chiaro che non possiede la chiave.

FUNZIONAMENTO DEI PROOF PROTOCOLS

Ogni protocollo si sviluppa in **round**: è caratterizzato da uno **scambio di messaggi** tra P e V. Alla fine del round il Verifier dà il proprio verdetto (accetta/rifiuta).
Se V **accetta tutti i round** non vuol dire che è certo di non essere stato ingannato, ma solo che **la probabilità che ciò sia accaduto è molto bassa**.
Solitamente la comunicazione inizia con l'invio di una *challenge* da parte di V.

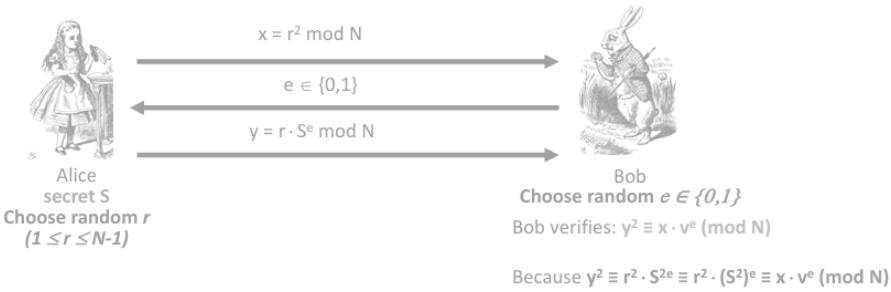
PROTOCOLLO DI FIAT-SHAMIR

- Siano dati due numeri primi *p,q* e si calcoli il loro prodotto $N = pq$;
- A ha un segreto S;
- N e $v = S^2 mod\ N$ sono **pubblici**, *S* è **segreto**;
- A vuole convincere B di conoscere *S* senza rivelare alcuna informazione su *S*;

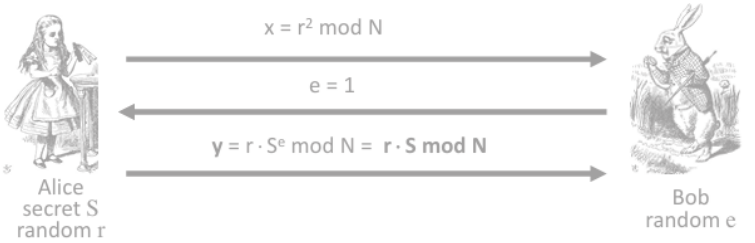
NOTA→ *mod N* è stato inserito per complicare il problema: è il **problema della radice quadrata modulare** (equivalente alla fattorizzazione).

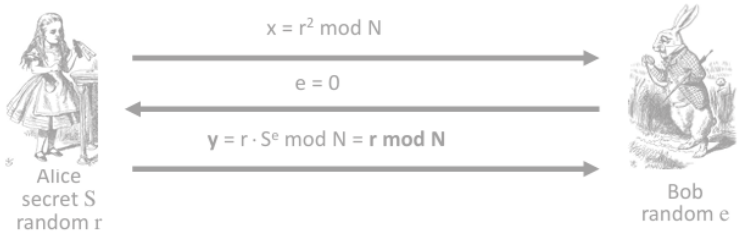
Per il generico round, il funzionamento è il seguente:

1. A sceglie $r\ casuale, 1 \leq r \leq N-1$
2. A calcola $x = r^2 mod\ N$
3. B sceglie un numero casuale $e \in \{0,1\}$
4. A calcola e invia $y = rS^e mod\ N$



B è in grado di verificare la veridicità delle affermazioni di A controllando che la seguente uguaglianza sia rispettata: $y^2 \equiv xv^e mod\ N$





Nelle figure precedenti sono analizzati i casi in cui *e=1* ed *e=0* rispettivamente. Si noti come nel secondo caso **A non ha bisogno di conoscere S**. Sebbene possa sembrare controproducente inviare *e=0*, di fatto B potrebbe essere ingannato anche inviando sempre *e=1*. Ciò avviene nel seguente modo:

- L'attaccante può scegliere di inviare non il valore di *x* come lo calcolava A ma invia $x = r^2 v^{-1} mod\ N$;
- Riceve e$= 1$;
- L'attaccante invia alla fine $y = r\ mod\ N$;

L'attaccante non segue il protocollo ma poco importa, perché non ha alcun vincolo reale nel seguirlo.

Alla fine, l'attaccante vincerà perché il Verifier effettuerà la seguente verifica $y^2 \equiv xv\ mod\ N$, dove *y=r* per l'ultimo messaggio dell'attaccante. E allora: $r^2 \equiv r^2 v^{-1} v \equiv r^2$ e la verifica va a buon fine.

Dunque, **è necessario che V scelga in modo casuale *e*** cosicché l'attaccante non possa prevedere la sua scelta (infatti deve inviare il proprio messaggio prima che V invii *e*)→ Il prover disonesto si gioca tutto sulla previsione che fa sul valore di *e*.

NOTA: A deve **variare sempre *r*** al fine di **proteggere il segreto S**. Infatti, se P usa lo stesso *r* in due round diversi e nel primo *e=0* e nel secondo *e=1*, l'attaccante può facilmente risalire ad *S*. In particolare,

- $e = 0,\ A$ invia $r\ mod\ N$;
- e$= 1,\ A$ invia $rS\ mod\ N$;

Chiunque può risalire a S avendo questi due valori a disposizione.

NOTA 2: Se B sta parlando con P disonesto, la probabilità che questo ha di convincere B dopo *t* round è di solo $\frac{1}{2^t}$.