Appunti del corso di Elementi di crittografia

Carmine D'Angelo

A.A. 2022/2023

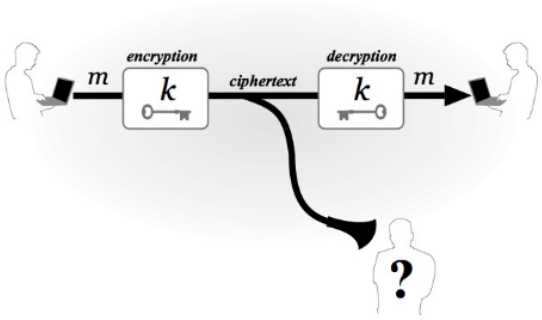
Indice

Capitolo 1

Introduzione

Crittografia moderna: Lo studio delle tecniche matematiche utili a proteggere l'informazione digitale, i sistemi di elaborazione e le computazioni distribuite da attacchi avversari.

1.1 Schemi di cifratura



A **chiave simmetrica** (symmetrickey setting) = stessa chiave per cifrare e decifrare. Abbiamo due applicazioni canoniche per lo scenario a chiave simmetrica, le parti sono separate "nello spazio" oppure una parte comunica con se stessa "nel tempo"
A **chiave asimmetrica** (asymmetrickey setting/ public keysetting) = chiavi diverse.

La sintassi che useremo per riferirci agli oggetti di uno schema è la seguente:

M: spazio dei messaggi leciti

C: spazio dei cifrati possibili

Gen: algoritmo per generare le chiavi

Enc: algoritmo di cifratura

Dec: algoritmo di decifratura

1.1.1 Principio di Kerckhoffs

Un qualunque attaccante per decifrare deve avere la chiave K e Dec. Pertanto la chiave K deve essere segreta mentre per l'algoritmo Dec ci sono diverse opinioni. Per Kerckhoffs la sicurezza non dovrebbe basarsi sulla segretezza degli algoritmi di cifratura, ma soltanto sulla segretezza della chiave. Questo perché è molto più semplice proteggere la chiave, in caso di rottura dello schema basta cambiare soltando la chiave, è molto più facile lo sviluppo in larga scala se si possono usare gli stessi algoritmi.

1.1.2 Cifrario di Cesare

Cifratura: Ogni lettera del messaggio in chiaro viene sostituita dalla lettera che si trova 3 posti in avanti nell'alfabeto: A -> D, B -> E, ..., Z-> C.
Decifratura: Si effettua l'operazione inversa (lettera che precede di 3 posti)
Lo schema è fisso non c'è chiave segreta.

1.1.3 Shift Cipher

L'associazione carattere in chiaro/carattere cifrato è uno spostamento fissato con un altro carattere dell'alfabero.
Abbiamo un alfabeto inglese mappato su interi: {a,b,c, ..., z} = {0,1,2, ..., 25}, e abbiamo un messaggio m=$m_1, m_2, ..., m_i$ con m_i in {0,1,2, ..., 25}.

- La funzione di encrypt è definita nel seguente modo: $\boldsymbol{Enc_K}(m_1, m_2, ..., m_i) = c_1, c_2, ..., c_i$ con $c_i = [(m_i + k) \text{mod } 26]$ per i=1,..., i, dove [a mod N] (riduzione modulo N)denota il resto della divisione per N e risulta 0 <= [a mod N] < N.
- La funzione di decrypt è definita nel seguente modo:$\boldsymbol{Dec_K}(c_1, c_2, ..., c_i) = m_1, m_2, ..., m_i$ con $m_i = [(c_i - k) \text{mod } 26]$.

Il cifrario non è sicuro perchè ci sono solo 26 chiavi e quindi basta una semplice ricerca esaustiva per potere ottenere il testo cifrato. Quindi una condizione necessaria (ma non sufficiente) affinché un cifrario sia sicuro è che lo spazio delle chiavi sia sufficientemente grande (sufficient key-space principle) cosicchè una attacco di tipo ricerca esaustiva sia impraticabile. Ad oggi lo spazio delle chiavi deve contenerne almeno 2^{80}.

1.1.4 Cifrari per sostituzione monoalfabetica

L'associazione tra un carattere in chiaro e uno cifrato è completamente arbitraria.

alfabeto in chiaro																							
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
X	E	U	A	D	N	B	K	V	M	R	O	C	Q	F	S	Y	H	W	G	L	Z	I	J	P	T
alfabeto cifrante																							

Lo spazio delle chiavi in questo caso sono tutte le possibili permutazioni dell'alfabeto: $26! \approx 2^{88}$. Anche se un attacco di tipo forza bruta è impraticabile ci sono dei problemi. La chiave specifica una sostituzione FISSA per ogni carattere del messaggio in chiaro, infatti lo stesso carattere viene cifrato sempre allo stesso modo. Si può utilizzare come attacco l'analisi delle frequenze.

Analisi delle frequenze

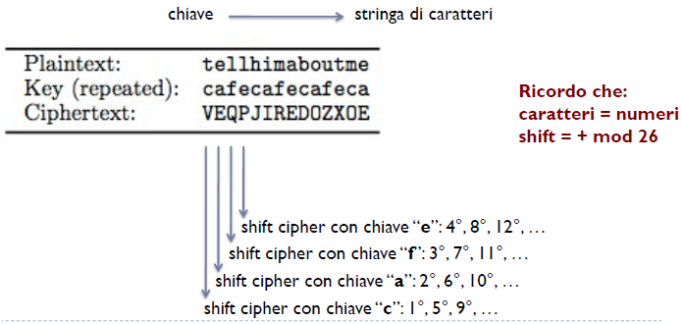
Si basa sul creare una tabella con tutte le frequenze di un determinato carattere in un'alfabeto. Una volta che si hanno tutte le frequenze di un alfabeto si controlla quante volte un carattere compaia nel nostro testo cifrato e calcoliamo: $\frac{\#occorrenze\_carattere\_cifrato}{\#caratteri\_cifrato}$ una volta che abbiamo le frequenze si passa a controllare quelle dell'alfabeto e si prova a sostituire con il carattere con frequenza simile.

Shift cipher - attacco automatizzato

Usiamo la distribuzione delle frequenze per calcolare la chiave segreta. Sia $0 \leq p_i \leq 1$ la frequenza della i-esima lettera dell'alfabeto inglese. Risulta: $\sum_{i=0}^{25} p_i^2 \approx 0.065$.
Sia $0 \leq q_i \leq 1$ la frequenza della i-esima lettera dell'alfabeto inglese nel cifrato. Abbiamo quindi p_i (i-esima lettera) sostituita con q_{i+k}(i+k-esima lettera nel cifrato) $\forall$ indice i=0...25, per ogni valore di j = 0, 1, ..., 25, calcoliamo $I_j \stackrel{\text{def}}{=} \sum_{i=0}^{25} p_i * q_{i+j}$, ci aspettiamo che $I_k \approx 0.065$.
L'attacco restituisce come ipotesi per k il valore di j per cui I_j è più vicino a 0.065,l'attacco è efficiente, è facile da automatizzare e non è richiesta alcuna "analisi del significato" (attacco precedente manuale).

1.1.5 Il cifrario di Vigenère

Realizza uno "shift poli-alfabetico", nei cifrari poli-alfabetici la chiave definisce una sostituzione da applicare a blocchi di caratteri del messaggio in chiaro, es: ab -> DZ o ac -> TY, la cifratura non è più associata ad un singolo carattere nel cifrato, in questo modo essenzialmente la distribuzione delle frequenze viene "appianata".
Il cifrario di Vigenere usa semplicemente "più istanze indipendenti dello shift cipher".



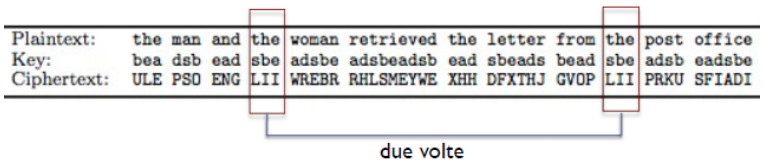
Rompere il cifrario di Vigenère

Prima di tutto assumiamo che sia nota la lunghezza della chiave k. Sia $k = k_1, k_2, ..., k_t$ dove k_j è un carattere della chiave, si divida il testo cifrato $k = c_1, c_2, ..., c_t$ parti.
Si calcoli la chiave k_j, per j=1,...,t, usando l'attacco efficiente contro lo shift cipher descritto in precedenza.

Test di Kasiski

In caso che conosciamo la lunghezza della chiave abbiamo due modi per procedere:

1. Se conosciamo un massimo per la lunghezza possiamo tentare tutte le ipotesi per t=1, t=2, ...
2. Usare il metodo di Kasiski cercando di individuare pattern ripetuti.



La distanza tra i pattern ripetuti, assumendo che non sia un evento accidentale, deve essere un multiplo del periodo della chiave, il massimo comune divisore delle distanze tra i pattern ripetuti restituirà il periodo (o un multiplo di esso) limite superiore al periodo.

Indice di coincidenza: automatizzazione dell'attacco

Se il periodo del cifrato $c = c_1, c_2, ...$ è t, allora $c_1, c_{1+t}, c_{1+2t}, c_{1+3t}, ...$ è la prima sequenza che risulta dalla cifratura ottenuta usando il "primo shift cipher". Ciò significa che $q_0, q_1, ..., q_{25}$ è semplicemente $p_0, p_1, ..., p_25$ "shiftata" di j posizioni pertanto, risulta $\sum_{i=0}^{25} q_i^2 \approx \sum_{i=0}^{25} p_i^2 \approx 0.065$.
Disponiamo, quindi, di un modo elegante per determinare la lunghezza t della chiave. Per w=1,2,3, ... consideriamo la sequenza $c_1, c_{1+w}, c_{1+2w}, c_{1+3w}, ...$ e calcoliamo i valori $q_0, q_1, ..., q_25$, e il valore $S_w = \sum_{i=0}^{25} p_i^2$. Ci aspettiamo che, quando w = t, risulta $S_w \approx 0.065$.
D'altra parte, se w $\neq$ t, i caratteri $c_1, c_{1+w}, c_{1+2w}, c_{1+3w}, ...$ sono presenti approssimativamente con la stessa frequenza, prossima a quella uniforme. Vale a dire, $q_i \approx \frac{1}{26}$, per i = 0, ..., 25. Da cui risulta: $S_w = \sum_{i=0}^{25} q_i^2 \approx \sum_{i=0}^{25} \frac{1}{26}^2 \approx 0.038$.
Pertanto, il valore più piccolo di w per cui $S_w \approx 0.065$ è verosimilmente la lunghezza della chiave. Una seconda sequenza $c_1, c_{1+w}, c_{1+2w}, c_{1+3w}, ...$ può essere usata per confermare l'ipotesi.
In conclusione possiamo dire che l'attacco è elegante ed efficiente, richiede un cifrato lungo per la stima delle frequenze e una chiave più lunga implica la necessità di un cifrato più lungo.

Capitolo 2

Richiami di probabilità

2.1 Prime definizioni

- Ω: insieme di tutti i possibili risultati di un esperimento (eventi elementari);
- Un evento E è un sottoinsieme di Ω;
- Con la probabilità un evento assume un valore compreso tra 0 e 1 con Pr(E) $\geq$0 e Pr(Ω) = 1;
- Se E_1 e E_2 sono mutualmente esclusivi(non hanno risultati in comune) allora:Pr(E_1 v E_2) = Pr(E_1) + Pr(E_2);
- Il complemento della probabilità Pr($\overline{E}$) = - Pr(E);
- Pr(E_1 Λ E_2) $\leq$ Pr(E_1);
- Pr(E_1 v E_2) $\geq$ Pr(E_1);
- Pr(E_1 v E_2) $\leq$ Pr(E_1) + Pr(E_2);
- **Union bound**: dati k eventi Pr($U_{i=1}^{k} E_i$) $\leq \sum_{i=1}^{k}$ Pr(E_i)

2.2 Teorema di Bayers

La probabilità condizionata di E_1 dato E_2, denotata con Pr($E_1|E_2$), è definita come:
Pr($E_1|E_2$) $\stackrel{\text{def}}{=} \frac{Pr(E_1 \wedge E_2)}{Pr(E_2)}$ quindi ne consegue che $Pr(E_1 \wedge E_2)$ = Pr($E_1|E_2$) * Pr(E_2).

Per il teorema di Bayers quindi se Pr(E_2) $\neq$ 0 allora Pr($E_1|E_2$) = $\frac{Pr(E_2|E_1)*Pr(E_1)}{Pr(E_2)}$

Dimostrazione: Pr($E_1|E_2$) = $\frac{Pr(E_1 \wedge E_2)}{Pr(E_2)}$ = $\frac{Pr(E_2 \wedge E_1)}{Pr(E_2)}$ = $\frac{Pr(E_2|E_1)*Pr(E_1)}{Pr(E_2)}$

2.2.1 Eventi indipendenti

Gli eventi E_1 ed E_2 sono indipendenti se Pr(E_1 | E_2) = Pr(E_1), vuol dire che l'evento E_2 non condiziona la probabilità che si verifichi E_1. Il fatto che i due eventi siano indipendenti implica che: Pr(E_1) = Pr($E_1|E_2$) = $\frac{Pr(E_1 \wedge E_2)}{Pr(E_2)}$ e quindi $Pr(E_1 \wedge E_2) = Pr(E_1) * Pr(E_2)$

2.3 Teorema delle partizioni

Gli eventi $E_1, E_2, ..., E_n$ costituiscono una partizione di Ω se $\Pr(E_1 \vee E_2 \vee ... \vee E_n) = 1$ e $\forall\, i \neq j$, $\Pr(E_i \wedge E_j) = 0$.
Per qualsiasi $F \subseteq \Omega$, risulta $\Pr(F) = \sum_{i=1}^{n} Pr(F \wedge E_i)$

Con n=2 abbiamo che $E_2 = \overline{E_1}$ infatti $\Pr(F) = Pr(F \vee E_1) + Pr(F \vee E_2) = Pr(F \vee E_1) + Pr(F \vee \overline{E_2}) = Pr(F|E_1) * Pr(E_1) + Pr(F|\overline{E_1}) * Pr(\overline{E_1})$

Se prendiamo $F = E_1 \vee E_2$ otteniamo una limitazione migliore dell'union bound:
$Pr(E_1 \vee E_2) = Pr(E_1 \vee E_2|E_1) * Pr(E_1) + Pr(E_1 \vee E_2|\overline{E_1}) * Pr(\overline{E_1} \leq Pr(E_1) + Pr(E_2|\overline{E_1})$. Se estendiamo il risultato per n eventi otteniamo che:
$Pr(\cup_{i=1}^{n} E_i) \leq Pr(E_1) + \sum_{i=2}^{n} Pr(E_i|\overline{E_1} \vee ...\overline{E}_{i-1})$

2.4 Problema del compleanno

Quanto deve essere numeroso un gruppo di persone affinchè, con probabilità almeno 1/2; due di esse siano nate lo stesso giorno? Assumiamo che i compleanni sono uniformemente distribuiti e che N = 365, rappresentiamo con y_i il compleanno della i-esima persona del gruppo $y_i, ..., y_q$ la soluzione del problema consisnte nel trovare il minimo q per cui risulta coll(q,365) $\geq 1/2$, è provato che q=23 è sufficiente.

Formalizziamo: Se scegliamo q elementi $y_i, ..., y_q$ uniformemente a caso da un insieme di taglia N, qual è la probabilità che esistano i e j distinti tali che $y_i = y_j$(collisione), la probabilità dell'evento sara indicata con coll(q,N).

2.4.1 Upper bound

Sia N un intero positivo fissato, e siano $y_1, ..., y_q$ q elementi scelti indipendentemente ed uniformemente da un insieme di taglia N. La probabilità che esistano i e j distinti per cui $y_i = y_j$ è coll(q,N) $\leq \frac{q^2}{2N}$

Dim: Applichiamo l'union bound. Sia Coll l'evento che denota una collisione e $Coll_{i,j}$ l'evento $y_i = y_j$.
Per le assunzioni fatte, $\Pr[Coll_{i,j}] = 1/N$ per ogni i e j distinti e Coll $= \cup_{i \neq j} Coll_{i,j}$. Pertanto, $Pr[Coll] = Pr[\cup_{i \neq j} Coll_{i,j}] \leq \sum_{i \neq j} Pr[Coll_{i,j} = \binom{q}{2} * \frac{1}{N} \leq \frac{q^2}{2N}$

2.4.2 Lower bound

Sia N un intero positivo fissato, e siano $y_1, ..., y_q \leq \sqrt{2N}$ elementi scelti indipendentemente ed uniformemente da un insieme di taglia N, allora la probabilità che esistano i e j distinti tali che $y_i = y_j$ è coll(q,N) ≥ 1 - $e^{\frac{q*(q-1)}{2N}} \geq \frac{q*(q-1)}{4N}$

Conclusione: se q = $\theta(\sqrt{N})$, la probabilità di avere una collisione è costante.

2.5 Variabili casuali

Sono variabili che possono assumere un insieme di differenti valori, ciascuno con una probabilità associata. I valori vengono assunti in accordo al risultato dell'esperimento a cui si fa riferimento. Più formalmente: $X : \Omega -> S$.
Dove Ω è lo spazio degli eventi elementari con relative probabilità ed S un insieme di valori. Solitamente S è un insieme finito di numeri reali. Se X non assume valori negativi, è detta non negativa.
Se S = $\{0, 1\}$, X viene detta variabile casuale 0/1 (o binaria).

2.5.1 Valore medio

Diremo che le variabili casuali 0/1 $X_1, ..., X_k$ sono indipendenti se per tutti i $b_1, ..., b_k$, vale che $Pr[X_1 = b_1 \wedge ... \wedge X_k = b_k = \prod_{i=1}^{k} Pr[X_i = b_i]$.
Il valore medio Exp(X) è definito come: Exp(X) = $\sum_{s \in S} Pr[X0s] * s$.
Il valore medio soddisfa la proprietà di linearità. Date le variabili casuali $X_1, ..., X_k$ risulta:
Exp$[\sum_{i=1}^{k} X_i] = \sum_{i=1}^{k} Exp[X_i]$

2.5.2 Disuguaglianza di Markov

Se X_1 e X_2 sono indipendenti Exp(X_1X_2) = Exp(X_1)*Exp(X_2).

La disuguaglianza di Markov dice che se X è una variabile casuale non negativa, e sia v > 0 allora: $Pr[X \geq v] \leq \frac{Exp[X]}{v}$

Dimostrazione : Supponiamo X assuma valori in S. Risulta:
$Exp[X] = \sum_{s \in S} Pr[X = s] * s \geq \sum_{s \in S, s<v} Pr[X = s] * 0 + \sum_{s \in S, s \geq v} Pr[X = s] * v \geq v * Pr[X \geq v]$.

2.5.3 Varianza

La varianza di una variabile casuale X, indicata con $Var[X]$, misura quanto X devia dal valore medio.
$Var[X] \stackrel{\text{def}}{=} Exp[(X - Exp[X])^2] = Exp[X^2] - Exp[X]^2$.
$Var[aX + b] = a^2 Var[X]$.
Per variabili casuali 0/1 X_i risulta $Var[X_i] \leq 1/4$ perchè in questo caso $Exp[X_i] = Exp[X_i^2]$ e quindi: $Exp[X_i^2] - Exp[X_i]^2 = Exp[X_i](1 - Exp[X_i])$ che ha valore massimo per $Exp[X_i] = 1/2$ da cui $Var[X_i] \leq 1/4$.

2.5.4 Disuguaglianza di Chebychev

Sia X una variabile casuale e sia $\delta > 0$ allora $Pr[|X - Exp[X] \geq \delta] \leq \frac{Var[X]}{\delta^2}$.
Si dimostra applicando la disuguaglianza di Markov.

2.6 Algoritmi randomizzati e variabili casuali

Sia A un algoritmo probabilistico (o randomizzato, cioè che usa random bit), le variabili casuali sono utili per rappresentare l'output di A; A rappresenta i possibili valori con relative probabilità che l'algoritmo può dare in output, a seconda delle scelte casuali che compie.
In uno schema di cifratura randomizzato abbiamo:

- Gen() –> $k \in K$ (spazio delle chiavi), la variabile casuale K può essere usata per rappresentare i possibili $k \in K$ che l'algoritmo di generazione delle chiavi può dare in output a seconda delle scelte casuali che effettua;
- $Enc_k()$ –> $c \in C$ (spazio dei cifrati), la variabile casuale C può essere usata per rappresentare i possibili $c \in C$ che l'algoritmo di cifratura può dare in output a seconda delle scelte casuali che effettua;

Per valutare la probabilità dei cifrati in generale utilizziamo la variabile casuale C definita da:
$Pr[C = c] = Pr[Enc_k(m) = c]$.

La distribuzione di C dipende dalle distribuzioni di K ed M e dalle scelte casuali che l'algoritmo di cifratura effettua:
$Pr[C = c] = Pr[Enc_k(m) = c | K = k, M = m] * Pr[K = k \wedge M = m] \Rightarrow C_{k,m}$.

2.6.1 Famiglie di distribuzioni

Una famiglia di distribuzioni (distribution ensemble) è una famiglia di distribuzioni di probabilità o variabili casuali $X = \{X_i\}_{i \in I}$,dove:

- X_i denota una distribuzione di probabilità o variabile casuale;
- i è l'indice che denota la i-esima distribuzione;
- I è l'insieme degli indici e può essere un sottoinsieme degli interi, un insieme di stringhe, un generico insieme contabile.

Capitolo 3

Segretezza perfetta

3.1 Segretezza perfetta

Un cifrario perfettamente sicuro deve garantire confidenzialità e riservatezza. Per provare che un cifrario è perfettamente sicuro assumeremo che l'avversario abbia potere computazionale illimitato e useremo definizioni matematiche precise e prove. Useremo infine esperimenti e algoritmi randomizzati.

In crittografia definiamo con entropia la funzione che misura l'incertezza dei dati che vogliamo avere in output.

Esempio estrazione di bit casuali: Supponiamo di disporre di una sequenza dei risultati del lancio di una moneta difettosa, rappresentata in binario, la moneta dà:

- T = 1 con probabilità p
- C = 0 con probabilità (1-p)

I lanci sono indipendenti l'uno dall'altro. Considerando le coppie:
Se vediamo: 00 01 10 11
Scriviamo: / 0 1 /
La sequenza di lanci usando questa moneta avrà una distribuzione uniforme, perchè:

- La probabilità che una coppia produca 0 è (1-p)p
- La probabilità che una coppia produca 1 è p(1-p)

3.1.1 Schema di cifratura

Uno schema di cifratura è rappresentato da una tripla di algoritmi (Gen, Enc, Dec) con M tale che $|M|>1$:

- Gen: l'algoritmo di generazione di chiavi viene realizzata in maniera probabilistica, quindi utilizza bit casuali e il cui output è un elemento $k \in K$, generato in accordo a una distribuzione di probabilità (distribuzione uniforme);
- Enc: l'algoritmo di cifratura viene realizzata in maniera probabilistica che prende in input un messaggio $m \in M$ e una chiave $k \in K$ avendo come risultato un cifrato. (Da notare che la generazione del cifrato avviene in maniera probabilistica);
- Dec: l'algoritmo di decifratura viene realizzata in maniera deterministica dove prende in input un cifrato $c \in C$ e una chiave $k \in K$ avendo come output un messaggio in chiaro $m \in M$.

Notazioni

Indicheremo con k <– Gen() e c <– Enc_k(m) l'output di algoritmi probabilistici.
Indicheremo c:= Enc_k(m) e m:= Dec_k(c) l'output di algoritmi deterministici.
Useremo la notazione x <– S per rappresentare l'estrazione di un elemento da un insieme S in accordo alla distribuzione uniforme.

3.2 Condizione di perfetta sicurezza

La condizione di perfetta sicurezza che ci aspettiamo da uno schema di cifratura è:

$\forall\, k \in K, \forall\, m \in M$ e qualsiasi $c \leftarrow Enc_k(m)$, risulta $Dec_k(c) = m$ con probabilità 1.

3.2.1 Variabili casuali

La variabile casuale M denoterà il messaggio in chiaro. La Pr[M = m] è la probabilità che un messaggio sia scelto per la cifratura. Fissato uno schema di cifratura ed una distribuzione su M si determina una distribuzione sullo spazio dei cifrati ottenuta:

- da k generata tramite Gen()
- scegliendo m in accordo alla distribuzione su M
- calcolando c $\leftarrow Enc_k(M)$

Con la variabile casuale C denoteremo l'output del processo di cifratura, C = $Enc_k(M)$

Esempio: Consideriamo lo Shift Cipher K = $\{0, ..., 25\}$ $Pr[K = k] = \frac{1}{26}, \forall k \in K$, sia data su M la distribuzione $Pr[M = a] = 0,7$ e $Pr[M = z] = 0,2$.
1)Qual è la probabilità che il cifrato sia B?
Abbiamo solo due possibilità che m = a e k =1 oppure m = z e k = 2, visto che le variabili casuali M e K sono indipendenti risulta:

- $Pr[M = a \wedge K = 1] = Pr[M = a] * Pr[K = 1] = 0,7 * \frac{1}{26}$
- $Pr[M = z \wedge K = 2] = Pr[M = z] * Pr[K = 2] = 0,3 * \frac{1}{26}$

Quindi $Pr[C = B] = Pr[M = a \wedge K = 1] + Pr[M = z \wedge K = 2] == 0,7 * \frac{1}{26} * 0,3 * \frac{1}{26} = \frac{1}{26}$

2) Qual è invece la probabilità che il messaggio in chiaro sia "a", dato il cifrato B?
Usando il teorema di Bayes risulta:
$Pr[M = a|C = B] = \frac{Pr[C=B|M=a]*Pr[M=a]}{Pr[C=B]} = \frac{Pr[C=B|M=a}{\frac{1}{26}}$
Ma $Pr[C = B|M = a] = \frac{1}{26}$, se M = a, infatti l'unico modo per ottenere C = B è che K = 1 che accade con probabilità $\frac{1}{26}$, quindi -: $Pr[M = a|C = B] = \frac{\frac{1}{26}*0,7}{\frac{1}{26}} = 0,7$

3.2.2 Definizione 2.3

Uno schema di cifratura (Gen,Enc,Dec) con spazio dei messaggi M è perfettamente segreto se:

- per ogni distribuzione di probabilità su M. Non avendo a conoscenza su dove verrà usata lo schema di cifratura, si richiede che potrà essere usata per qualunque contesto: messaggi che possono essere equiprobabili oppure alcuni messaggi più probabili rispetto le altri;
- per ogni messaggio $m \in M$;
- per ogni cifrato $c \in C$ per cui risulta $Pr[C = c] > 0$;

si ha: $Pr[M = m|C = c] = Pr[M = m]$

3.2.3 Lemma 2.4

Possiamo fornire una definizione equivalente richiedendo che la distribuzione di probabilità dei cifrati non dipenda dal messaggio in chiaro.
Dal lemma: Uno schema di cifratura(Gen,Enc,Dec) con spazio dei messaggi M è perfettamente segreto se e solo se $\forall m, m' \in M$ ed $\forall c \in C$:

$$Pr[Enc_k(m) = c] = Pr[Enc_k(m') = c]$$

dimostrazione

1)Proviamo che se la condizione vale, allora lo schema è perfettamente segreto. Fissiamo una distribuzione su M e sia e tale che $Pr[C = c] > 0$.

- Se $Pr[M = m] = 0$, allora banalmente $Pr[M = m|C = c]$

$= \frac{Pr[C=c|M=m]*Pr[M=m}{Pr[C=c]} = 0 \ \rightarrow Pr[M = m|C = c] = Pr[M = m]$

- Se Se $Pr[M = m] > 0$ invece notiamo che:
$Pr[C = c|M = m] = Pr[Enc_k(M) = c|M = m] = Pr[Enc_k(m) = c]$.
Sia $\delta_c = Or[Enc_k(m) = c]$.

Se la condizione del lemma vale per ogni m' risulta:
$Pr[Enc_k(m') = c] = Pr[C = c|M = m']\delta_c$.
Applicando il teorema di Bayes:
$Pr[M = m|C = c] = \frac{Pr[C=c|M=m]*Pr[M=m]}{Pr[C=c]}$
$= \frac{Pr[C=c|M=m]*Pr[M=m]}{\sum_{m\in M} Pr[C=c|M=m']*Pr[M=m']}$
$= \frac{\delta_c*Pr[M=m]}{\delta_c*\sum_{m\in M} Pr[M=m']}$ **Nota:**$\sum_{m\in M} Pr[M = m'] = 1$ perchè è la somma di tutte le probabilità
$= Pr[M = m]$
Pertanto lo schema è perfettamente segreto.

2) Viceversa sia lo schema perfettamente segreto. Sia M uniformemente distribuita, allora per il teorema di Bayes $\forall c : Pr[C = c] > 0$ e per ogni m risulta:

$$Pr[M = m|C = c] * Pr[C = c] = Pr[C = c|M = m] * Pr[M = m]$$

La segretezza perfetta $Pr[M = m|C = c] = Pr[M = m]$ implica:
$Pr[M = m|C = c] = Pr[C = c|M = m] = Pr[M = m]$ ovvero $Pr[C = c] = Pr[C = c|M = m]$.
Pertanto $\forall m, m'$ e $\forall c : Pr[C = c] > 0$ abbiamo: $Pr[Enc_k(m) = c] = Pr[C = c] = Pr[Enc_k(m') = c]$

3.2.4 Perfetta indistinguibilità

Consideriamo uno schema di cifratura Π=(Gen, Enc, Dec) con spazio di messaggi M. Sia A un avversario, l'esperimento: $PrivK^{eav}_{A,\Pi}$

- L'avversario sceglie i due messaggi in chiaro m e m' mandandoli in output;
- Il Challenger C genera k tramite Gen() e sceglie uniformemente a caso il bit in $0, 1$;
- C calcola $c = Enc_k(m_b)$ detto cifrato di sfida e lo passa ad A;
- A dà in output un bit b';
- L'output dell'esperimento è 1 se b'=b, 0 altrimenti. Se l'output è 1, A ha successo.

Uno schema di cifratura è perfettamente indistinguibile se nessun avversario può avere successo con probabilità $> \frac{1}{2}$. La strategia migliore che un avversario può adottare è prevedere in maniera casuale il messaggio con probabilità di un $\frac{1}{2}$.

3.2.5 Definizione 2.5

Uno schema di cifratura (Gen,Enc,Dec) con spazio dei messaggi M è perfettamente indistinguibile se, per ogni A (non efficiente o efficiente), risulta: $Pr[Priv^{eav}_{A,\Pi} = 1] = \frac{1}{2}$

Nota: É possible dimostrare che vale il seguente **Lemma 2.6:**uno schema di cifratura Π è perfettamente segreto se e solo se è perfettamente indistinguibile. (Dimostrazione non importante al fine dell'esame).

3.3 One-time Pad

Sia $l > 0$ un intero. Siano M = K = C = $\{0,1\}^l$.

- Gen: sceglie uniformemente a caso la chiave $k \in \{0,1\}^l$
- Enc: dati $k \in \{0,1\}^l$ ed $m \in \{0,1\}^l$ manda in output il cifrato tramite l'operazione di XOR: $c := k \oplus m$
- Dec: dati $k \in \{0,1\}^l$ ed $c \in \{0,1\}^l$ manda in output il messaggio tramite l'operazione di XOR: $m := k \oplus c$

É facile verificare che: $\forall k, \forall m$ risulta $Dec_k(Enc_k(m)) = (k \oplus (k \oplus m)) = m$

3.3.1 Teorema 2.9

Lo schema di cifratura one-time pad è perfettamente segreto.

dimostrazione

Prima di tutto calcoliamo $Pr[C = c|M = m']$ per un arbitrario $c \in C$ ed $m' \in M$. Risulta:

$$Pr[C = c|M = m'] = Pr[Enc_l(m') = c] = Pr[m' \oplus K = c] = Pr[K = m' \oplus c] = 2^{-l}$$

poichè k è una chiave scelta uniformemente a caso in $\{0,1\}^l$. Per ogni $c \in C$ abbiamo:

$Pr[C = c] = \sum_{m' \in M} Pr[C = c|M = m'] * Pr[M = m'] =$
$\sum_{m' \in M} 2^{-l} * Pr[M = m'] = 2^{-l} * \sum_{m' \in M} Pr[M = m'] = 2^{-l} * 1 = 2^{-l}$

dove la somma è calcolata su tutti gli $m' \in M : Pr[M = m'] > 0$. Applicando il teorema di bayes otteniamo:
$Pr[M = m|C = c] = \frac{Pr[C=c|M=m]*Pr[M=m]}{Pr[C=c]} \frac{2^{-l}*Pr[M=m]}{2^{-l}} = Pr[M = m]$
Pertanto lo schema di cifratura one-time pad è perfettamente segreto.

3.4 Limitazioni della segretezza perfetta

Nello schema di cifratura one-time pad, la chiave è tanto lunga quanto il messaggio che si intende cifrare ed è sicura soltanto per un determinato messaggio. Quindi tale chiave non può essere utilizzato per cifrare un altro messaggio perché fare lo XOR con due messaggi diversi e chiave uguale permetterebbe all'avversario di capire esattamente in quale posizione i due messaggi risultano diversi. Dato che: $c \oplus c' = (m \oplus k) \oplus (m' \oplus k') = m \oplus m'$
Quindi l'one-time pad non è perfettamente segreto per qualsiasi nozione di segretezza perfetta per messaggi multiplo.

Faremo vedere che ogni schema perfettamente segreto deve avere uno spazio delle chiavi almeno tanto grande quanto lo spazio dei messaggi.

3.4.1 Teorema 2.10 (Shannon)

Se (Gen,Enc,Dec) è uno schema di cifratura perfettamente segreto con spazio dei messaggi M e spazio delle chiavi K, allora $|K| \geq |M|$.

Dimostrazione

Dimostriamo per assurdo che se fosse $|K| < |M|$ lo schema non potrebbe essere perfettamente segreto.
Sia $|K| < |M|$, siam M distribuita uniformemente e sia $c \in C : Pr[C = c] > 0$.
Definiamo: $M(c) \stackrel{\text{def}}{=} \{m : m = Dec_k(c), \text{ per qualche } k \in K\}$.
Chiaramente $|M(c)| \leq |K|$. Se $|K| < |M|$, allora $\exists m' \in M : m' \notin M(c)$. Ma allora risulta:
$Pr[M = m'|C = c] = 0 \neq Pr[M = m']$.
Per lo schema non è perfettamente segreto. Quindi deve essere $|K| \geq |M|$.

3.4.2 Teorema 2.11 (teorema di Shannon o shannon bound)

È uno strumento utile per provare la segretezza perfetta di uno schema di cifratura.
Sia (Gen, ENc,Dec) uno schema di cifratura con spazio dei messaggi M per cui $|M| = |K| = |C|$. Lo schema è perfettamente segreto se e solo se:

- Gen sceglie ogni chiave $K \in K$ con probabilità uguale a $\frac{1}{|K|}$;
- $\forall m \in M, c \in C \ \exists k \in K : \ Enc_k(m) = c$

Queste due condizioni sono soddisfate dall'one-time pad perché la prima condizione è validata avendo assunto che la cardinalità dello spazio dei messaggi, dei cifrati e delle chiavi devono essere uguali, mentre per la seconda ogni messaggio viene cifrato con una data chiave.

Capitolo 4

Segretezza computazionale

La segretezza perfetta è una nozione molto forte che ci permette di gestire avversari di potere illimitato e che garantisce da parte degli avversari una probabilità nulla di successo. Questa condizione porta a degli svantaggi di efficienza, ovvero le risorse che richiede sono difficili da gestire, in quanto dover cifrare un messaggio molto lungo richiede che le parti comunicanti condividano a priori una chiave che sia totalmente casuale e che sia tanto lunga quanto il messaggio. Viene definita, così, la segretezza computazionale in cui una chiave di una specifica lunghezza (128 bit) può essere usata per cifrare messaggi lunghi e che sia generata tramite algoritmi pseudocasuali. Abbiamo due tipi di approcci per sviluppare quest'ultima teoria.

4.1 Approccio concreto

Limita esplicitamente la probabilità di successo massima di qualsiasi Adv probabilistico che esegue per una specificata quantità di tempo o più in generale:
Uno schema è $(t;\epsilon)$-sicuro se qualsiasi Adv che può eseguire per tempo al più t ha successo nella rottura dello schema con probabilità al più ϵ.
Questo approccio ha diversi svantaggi come per esempio una difficoltà nell'interpretare un'affermazione, per esempio: Nessun Adv che esegue per 5 anni ha successo con prob. maggiore di ϵ; non specifica con quale potere computazionale, oppure non specifica cosa succede alla probabilità di rompere il sistema a 5 anni e 1 giorno, infatti quest'ultima potrebbe essere 1 e quindi rende inutilizzabile il sistema.

4.2 Approccio asintotico

Viene introdotto un parametro di sicurezza n intero che coinvolge sia chi invia il messaggio che ad Adv, inoltre Adv conosce il valore di n.
In generale per ottenere una nozione di segretezza più debole ma utile nella pratica, sono stati definiti due aspetti:

- avversario computazionalmente quantificabile (algoritmi efficienti), limitando un avversario ad avere una capacità computazionale al più di tempo polinomiale, non potrà effettuare una ricerca esaustiva in maniera polinomiale in uno spazio delle chiavi che ha un numero esponenziale di elementi;
- probabilità trascurabile, che si traduce da un punto di vista pratico all'impossibilità da parte dell'avversario di rompere lo schema.

Queste restrizioni sono essenziali in quanto servono per escludere gli attacchi da parte degli avversari andando a restringere il potere computazionale dell'avversario e ammettendo delle piccole probabilità di successo.

4.2.1 Algoritmi efficienti

A esegue in tempo polinomiale se esiste un polinomio $p(\cdot) : \forall x \in \{0,1\}^*$,A(x) termina al più $p(|x|)$ passo, dove $|x|$ indica la lunghezza dell'input.
Il parametro di sicurezza n sarà indicato in unario 1^n. Gli algoritmi possono avere altri input e in più assumeremo che di default tutti gli algoritmi dell'Adv siano probabilistici.

4.2.2 Probabilità di successo trascurabili

Definizione 3.4 : Una funzione $f : N \to R^+$ è trascurabile(negligible) se, per ogni polinomio positivo p, esiste un $n_0 : \forall n > n_0$ risulta f(n) $< \frac{1}{p(n)}$.

Questa definizione è simile a: $\forall$ costante c, $\exists\, n_0 : \forall n > n_0$ risulta $\frac{1}{n^c}$.
Denoteremo una generica funzione trascurabile con *negl*.

Proposizione 3.6 : Siano $negl_1(n)$ e $negl_2(n)$ funzioni trascurabili:

1. la funzione $negl_3(n) = negl_1(n) + negl_2(n)$ è trascurabile, questo significa che un evento con probabilità trascurabile anche se viene ripetuto p(n) volte risulta lo stesso trascurabile;

2. per qualsiasi polinomio p, la funzione $negl_4(n) = p(n) * negl_1(n)$, questo significa che se una funzione g non è trascurabile allora la funzione $f(n) \stackrel{\text{def}}{=} \frac{g(n)}{p(n)}$ non è trascurabile per ogni polinomio.

4.2.3 Framework per le definizioni

Il framework generale per le definizioni è il seguente:
Uno schema è sicuro se, per ogni Adv PPT A che sferra un attacco di qualche tipo formalmente specificato, e $\forall$ polinomio positivo p, $\exists$ un intero n_0 :, quando n $>$ n_0, la probabilità che A abbia successo è $< \frac{1}{p(n)}$.
Nota:nulla è garantito per valori $n \leq n_0$.

4.3 Schemi di cifratura computazionalmente sicuri

4.3.1 Definizione 3.7

Uno schema di cifratura a chiave privata è una tripla Π=(Gen, Enc, Dec) di algoritmi PPT tale che:

- $k \leftarrow Gen(1^n)$, algoritmo probabilistico di generazione della chiave k dove la chiave $k \in K : |k| \geq n$;

- $c \leftarrow Enc_k(m)$, algoritmo probabilistico di cifratura dove il messaggio $m \in \{0,1\}^*$ la chiave $k \in K$ e il cifrato $c \in \{0,1\}^*$;

- $m := Dec_k(c)$, algoritmo deterministico di decifratura dove il cifrato $c \in \{0,1\}^*$. la chiave $k \in K$ e il messaggio $m \in \{0,1\}^*$, Dec(c) restituisce $\perp$ in caso di errore.

Correttezza: $\forall\, n, \forall k$ restituito da $Gen(1^n)$ e $\forall m \in \{0,1\}^*$, risulta: $Dec_k(Enc_k(m)) = m$. Usiamo n per indicare il parametro di sicurezza.

Note: La nostra definizione è di uno schema senza stato, in più se lo spazio dei messaggi è $\{0,1\}^{l(n)}$, allora Π uno schema di cifratura a chiave privata a lunghezza fissa, per messaggi di lunghezza l(n).

4.3.2 Perfetta indistinguibilità nel caso computazionale

Abbiamo A(Adv) è di tipo PPT; A può vincere con probabilità trascurabilmente migliroe di $\frac{1}{2}$, l'esperimento dipende dal parametro di sicurezza n.

$PrivK^{eav}_{A,\Pi}(n)$

1. A ottiene 1^ne dà in output $m_0, m_1 : |m_0| = |m_1|$;
2. il challenger calcola $c \leftarrow Enc_k(m_b)$, dove $b \leftarrow \{0,1\}$ e $k \leftarrow Gen(1^n)$;
3. $A(1^n)$riceve c e dà in output $b' \in \{0,1\}$;
4. Se b = b', l'output dell'esperimento è 1 (caso in cui indovina) (A(1^n) vince), altrimenti 0 (caso in cui sbaglia).

4.3.3 Definizione 3.8

Uno schema di cifratura a chiave privata Π = (Gen, Enc, Dec) ha cifrature indistinguibili in presenza di un avversario che ascolta (eavesdropper) o è EAV-sicuro se, per ogni Adv A PPT, esiste una funzione trascurabile negl tale che:

$$\Pr[PrivK^{eav}_{A,\Pi}(n) = 1] \leq \frac{1}{2} + negl(n)$$

Dove la probabilità è calcolata sull'algoritmo randomizzato di A, la scelta casuale della chiave, la scelta casuale uniforme del bit b, e i random bit usati da $Enc_k(\cdot)$

4.3.4 Definizione 3.9

Definiamo $PrivK^{eav}_{A,\Pi}(n, b)$con $b \in \{0,1\}$ e l'output di A con $out_A(PrivK^{eav}_{A,\Pi}(n, b))$diamo la seguente definizione:
Π = (Gen, Enc, Dec) è EAV-sicuro se, per ogni Adv A PPT, esiste una funzione trascurabile negl tale che, per tutti gli n si ha:

$$|Pr[out_A(PrivK^{eav}_{A,\Pi}(n, 0) = 1] - Pr[out_A(PrivK^{eav}_{A,\Pi}(n, 1) = 1]| \leq negl(n)$$

Le due probabilità trattano di esperimenti diversi, l'avversario non sa in quale dei due casi si trova.

4.4 Sicurezza semantica

Introduciamo due nozioni:

1. il cifrato non rivela alcuna informazione sui singoli bit del messaggio in chiaro
2. il cifrato non aiuta un Adv PPT nel calcolo di qualsiasi funzione del messaggio in chiaro, l'avversario non riesce a calcolare nulla partendo dal cifrato.

Proveremo che la nozione di indistinguibilità implica 1. e 2 cioè che in pratica se è indistinguibile non può rivelare informazioni sui singoli bit oppure calcolare funzioni su m.

4.4.1 Teorema 3.10 (punto 1)

L'algoritmo Gen sarà implicito e genera chiavi distribuite uniformememnte a caso.

Sia Π = (Enc,Dec) uno schema di cifratura a chiave privata per messaggi di lunghezza l EAV-sicuro. Allora, per ogni Adv A PPT ed ogni i ∈ 1, ... , l, esiste una funzione trascurabile negl tale che:

$$Pr[A(1^n, Enc_k(m)) = m^i] \leq \frac{1}{2} + negl(n)$$

dove la probabilità è calcolata su scelta uniforme di m $\in \{0,1\}^l$ scelta uniforme di $k \in \{0,1\}^n$ random bit usati da A random bit usati da $Enc_k(\cdot)$

Dimostrazione Se per assurdo fosse possibile, con probabilità non trascurabile, calcolare l'i-esimo bit m^i, sarebbe anche possibile, con probabilità non trascurabile, distinguere m_0 da m_1 che differiscono nell'i-esimo bit.
Fissiamo un Adv arbitrario A PPT ed $i \in \{1, ..., l\}$, usiamo A per costruire un Adv A' che utilizza A per distinguere con prob. non trascurabile m_0 da m_1 che differiscono nell'i-esimo bit. (A è utilizzato come subroutine).
Iniziamo suddividendo la stringa di l bit in due parti uguali:

- $l_0 \subset \{0,1\}^l$ insieme di stringhe con i-esimo bit uguale a 0;
- $l_1 \subset \{0,1\}^l$ insieme di stringhe con i-esimo bit uguale a 1;

Essendo $|l_0| = |l_1| = 2^{l-1}$ ed m scelto in modo uniforme, risulta:

$$\begin{aligned} Pr[A(1^n, Enc_k(m)) = m^i] &= Pr[m \in l_0] \cdot Pr[A(1^n, Enc_k(m))] = 0 | m \in l_0] \\ &\quad + Pr[m \in l_1] \cdot Pr[A(1^n, Enc_k(m)) = 1 | m \in l_1] \\ = \frac{1}{2} \cdot Pr[A(1^n, Enc_k(m)) = 0 | m \in l_0] &* \frac{1}{2} \cdot Pr[A(1^n, Enc_k(m)) = 1 | m \in l_1] \end{aligned}$$

Costruiamo A' come segue:

> Adv A' sceglie uniformemente $m_0 \in l_0$ e $m_1 \in l_1$ e li passa al challenger che invierà successivamente il cifrato c ad A'; A' invocherà A($1^n, c$). Se A dà in output 0, A' da in output b'=0 altrimenti se da 1 da come output b'=1.

L'Adv A' userà l'esperimento $PrivK^{eav}_{A',\Pi}(n)$ ed usa A come subroutine per calcolare m^i. Visto che A è di tipo PPT lo è anche A'.
Dalla definizione $PrivK^{eav}_{A',\Pi}(n)$, sappiamo che A' ha successo $\iff$ A restituisce b dopo aver ricevuto $Enc_k(m_b)$. Pertanto risulta:

$$\begin{aligned} &Pr[PrivK^{eav}_{A',\Pi}(n) = 1] = Pr[A(1^n, Enc_k(m_b)) = b] (\text{b viene scelto uniform.}) \\ &= \frac{1}{2} \cdot Pr[A(1^n, Enc_k(m_0)) = 0] + \frac{1}{2} \cdot Pr[A(1^n, Enc_k(m_1)) = 1] \\ &= Pr[A(1^n, Enc_k(m)) = m^i] \end{aligned}$$

Poichè abbiamo assunto che Π = (Enc, Dec) è EAV-sicuro allora esiste un funzione tracurabile negl tale che:

$$Pr[PrivK^{eav}_{A',\Pi}(n) = 1] \leq \frac{1}{2} + negl(n) \text{ che implica:}$$

$$Pr[A(1^n, Enc_k(m)) = m^i] \leq \frac{1}{2} + negl(n)$$

4.4.2 Teorema 3.11 (punto 2)

Sia $\Pi = (Enc, Dec)$ uno schema di cifratura a chiave privata per messagi di lunghezza l EAV-sicuro, allora $\forall$ Adv A PPT, $\exists$ un Adv A' PPT tale che $\forall S \subseteq \{0,1\}^l$ ed $\forall f : \{0,1\}^l \to \{0,1\}$ esiste una funzione trascurabile negl tale che:

$$|Pr[A(1^n, Enc_k(m)) = f(m)] - Pr[A'(1^n) = f(m)]| \leq negl(n)$$

Dove avremo che la prima probabilità è calcolata sulla scelta uniforme di $m \in S$ e $k \in \{0,1\}^n$, sui random bit usati da A e usati da $Enc_k(\cdot)$.
La seconda probabilità è calcolata sulla scelta uniforme di $m \in S$ e dei random bit usati da A'.

Informalmente: Vogliamo mostrare che un Adv A che calcola f(m) con una certa probabilità quando riceve $Enc_k(m)$ implica un Adv A' che calcola f(m) con la stessa probabilità senza conoscere $Enc_k(m)$.
NOTA:f(m) indica una generica funzione.

Dimostrazione (Sketch) Poichè Π è EAV-sicuro, $\forall S \subseteq \{0,1\}^l$, nessun Adv PPT può distinguere tra $Enc_k(m)$ ed $Enc_k(1^l)$.
Se consideriamo la probabilità con cui A calcola f(m) data $Enc_k(m)$, sappiamo che A dovrebbe calcolare f(m) data $Enc_k(1^l)$ con probabilità simile, altrimenti A potrebbe essere usato per distinguere tra $Enc_k(m)$ ed $Enc_k(1^l)$

Distinguisher:

1. Sceglie uniformemente $m \in S$ e passa al challanger $m_0 = m$(messaggio che ha scelto) e $m_1 = 1^l$(messaggio di tutti 1 non serve a niente)
2. dopo aver ricevuto c dal challenger, invoca $A(1^n, c)$
3. Se A dà in output f(m), allora dà in output b'=0; altrimenti b'=1.

Se A dà in output f(m) con una probabilità significativamente migliore nel caso in cui riceve $Enc_k(m)$ rispetto a quando riceve $Enc_k(1^l)$, allora l'algoritmo Distinguisher viola la definizione 3.9.
Quindi possiamo costruire A' come segue:

Adv A'(1^n)

1. sceglie uniformemente $k \in \{0,1\}^n$
2. invoca A$(1^n, Enc_k(1^l))$
3. dà in output qualsiasi cosa A dà in output.

A dà in output f (m) quando viene seguito come subroutine di A' con probabilità uguale o simile di quando riceve $Enc_k(m)$. Pertanto A' ha i requisiti richiesti dal teorema.

4.4.3 Definizione 3.12 (sicurezza semantica completa)

Uno schema di cifratura a chiave privata Π = (Gen; Enc; Dec) è semanticamente sicuro in presenza di un eav se per ogni Adv A PPT esiste un Adv A' PPT tale che per qualsiasi Samp(1^n)[1] PPT e per ogni coppia di funzioni f ed h, calcolabili in tempo polinomiale esiste una funzione trascurabile negl per cui si ha:

$$|Pr[A(1^n, Enc_k(m), h(m)) = f(m)] - Pr[A'(1^n, |m|, h(m)) = f(m)]| \leq negl(n)$$

dove la prima probabilità è calcolata sulla scelta uniforme di $k \in \{0,1\}^n$, sui random bit usati da Samp(1^n), da A e da $Enc_k(\cdot)$. La seconda probabilità è calcolata sui random bit usciti da Samp(1^n) e da A'[2].

Informalmente: La probabilità che un Adv conoscendo il messaggio cifrato e informazioni sulla storia passata dei messaggi(h) riesca a calcolare f(m) e che un altro Adv conoscendo la lunghezza del messaggio e la storia passata riesca a calcolare f(m) è trascurabile.

[1]Samp è una funzione efficentemente calcolabile cioè se vogliamo prendere 10 numeri casualmente dobbiamo estrarre 10 volte i 10 numeri in tempo polinomiale.Un'altro esempio è la validazione del blocco nella blockchain (nonce).

[2]Serve ad indicare che all'avversario non importa nulla di avere il testo cifrato

Capitolo 5

Pseudocasualità

5.1 Generatore pseudocasuale

Un generatore pseudocasuale G è un algoritmo deterministico efficiente per trasformare una stringa uniformemente corta, chiamata seme (indicheremo con s), in una più lunga che sembra uniforme.
Un generatore pseudocasuale buono dovrebbe superare tutti i test statistici efficienti (test che cercano di capire se una stringa sia ottenuta in modo casuale o deterministicamente), ad ogni osservatore efficiente l'output dovrebbe sembrare una stringa uniforme.

Che cosa significa per una distribuzione di probabilità essere pseudocasuale? Sia Dist una distribuzione su stringhe di l bit. È pseudocasuale se risulta impossibile per ogni algoritmo PPT dire, con chance significativamente migliori di quelle offerte dal lancio di una moneta, se essa derivi da Dist o dalla distribuzione uniforme.

5.1.1 Definizione 3.14

Sia l(n) un polinomio e G un algoritmo deterministico di tempo polinomiale tale che, per ogni n ed $s \in \{0,1\}^n$, G(s) è una stringa di l(n) bit. Diremo che G è un PRG (generatore pseudo casuale) se valgono le seguenti condizioni:

1. **Espansione**: per ogni n risulta l(n) > n;
2. **Pseudocasualità**: per ogni algoritmo D PPT, esiste una funzione trascurabile negl(n) tale che:

$$|Pr[D(G(s)) = 1] - Pr[D(r) = 1]| \leq negl(n)$$

Dove la prima probabilità è calcolata sulla scelta uniforme di $s \in \{0,1\}^n$ e random bit di D. La seconda sulla scelta uniforme di $r \in \{0,1\}^{l(n)}$ e random bit di.

Informalmente: La probabilità che il Distinguisher capisca che la nostra stringa sia generata da un generatore pseudocasuale meno la probabilità che capisca che sia generata casualmente è minore di un valore trascurabile.

Esempio costruzione non sicura

Sia $G : \{0,\}^n \to \{0,1\}^{n+1}$ tale che G(s) = $s||\oplus_{i=1}^{n} s_i$(XOR di tutte le precedenti stringhe). Possiamo costruire un Distinguisher come segue:

> Su un input ω dà in output 1 se e solo se il bit finale di ω è l'xor di tutti.

Risulta:

- se l'input di D è G(s) $Pr[D(G(s)) = 1] = 1$
- se l'input di D è r allora $Pr[D(r) = 1] = \frac{1}{2}$, cioè che la probabilità che l'ultimo bit sia uguale all'xor dei precedenti è $\frac{1}{2}$ perchè r viene scelta in modo uniforme in $\{0,1\}^{n+1}$

Poichè la differenza è:

$$|Pr[D(G(s)) = 1] - Pr[D(r) = 1] = |1 - \tfrac{1}{2}| = \tfrac{1}{2}$$

Ed è un valore costante non trascurabile, e questo permette a D di distinguere la stringa e quindi G non è pseudocasuale.

Osservazione: distinguisher esponenziale

Sappiamo che data una quantità illimitata di tempo è pertanto banale distinguere. Infatti preso un distinguisher D di tempo esponenziale che tenta tutti i semi $s \in \{0,1\}^n$ e dà in output 1 se e solo se G(s) = ω per qualche s.
Si noti che presa una stringa ω uniforme in $\{0,1\}^{2n}$ risulta $Pr[G(s) = \omega] = \frac{1}{2^n}$. Pertanto:

$$|Pr[D(G(s)) = 1] - Pr[D(r) = 1] = 1 - \tfrac{1}{2^n} \text{ non è trascurabile}$$

Quindi D distingue attraverso una ricerca esaustiva del seme $s \in \{0,1\}^n$. Ciò non contraddistingue la pseudocasualità di G perchè D non è efficiente computazionalmente.

5.1.2 Stream cipher

Uno stream cipher a differenza di un PRG non ha un limite al fattore di espansione e produce i suoi bit di output gradualmente.
Uno stream cipher è una coppia di algoritmi : (Init, GetBits) deterministici dove:

- Init(s, IV): prende in input un seme e un vettore di inizializzazione (IV) che però può essere omesso. In output da uno stato iniziale st_0
- GetBits(st_{i-1}): prende in input uno stato e da in output la coppia (st_i, y)(prossimo stato, stringa di bit).

Algorritmo 3.16

Dato uno stream cipher ed un fattore di espansione l, possiamo definire $G_l : \{0,1\}^n \to \{0,1\}^l$ come segue:

Costruiamo G_l da (Init, GetBits)

Input: Seed s e un vettore di inzializzazione ottimale IV
Output:$y_1, ..., y_l$

```
st_0 := Init(s,IV)
for i = 1 to l:
    (y_i, st_i):= GetBits(st_{i-1})
return y_1, ..., y_l
```

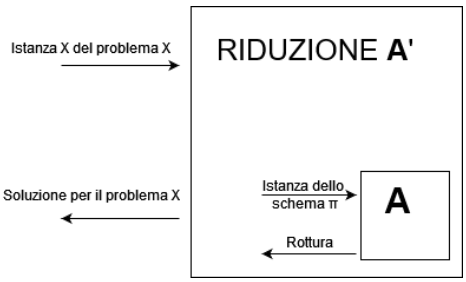
Uno stream cipher è sicuro (in un senso di base) se non prende in input IV e per ogni polinomio l(n) $>$ n, la funzione G_l è un generatore pseudocasuale con fattore di espansione l.

Lo stream cipher è sicuro se G_l è un PRG

5.2 Dimostrazioni per riduzione

Presenteremo una riduzione esplicita che mostra come trasformare un Adv efficiente A che ha successo nel rompere la costruzione data in un Adv efficiente A' che:

- risolve il problema matematico supposto difficile
- rompe la primitiva crittografica assunta sicura



A' vuole rompere il generatore decidendo se una stringa è prodotta da PRG o no. Per risolvere il suo problema vuole sfruttare A, A è un algoritmo che attacca l'algoritmo di cifratura. In questo esperimento A' prende il posto del challenger C e sfida A. A quindi darà due messaggi ad A'. A' sceglierà un cifrato per i due messaggi e lo sottoporrà ad A ed attenderà la sua risposta per capire se è riuscito a sottoporre una stringa di un PRG ad A e a rompere il PRG.

5.2.1 Riduzione

Assunzione: il problema X è difficile implica che non può essere risolto con algoritmi PPT. Vogliamo provare che la costruzione Π è sicura.

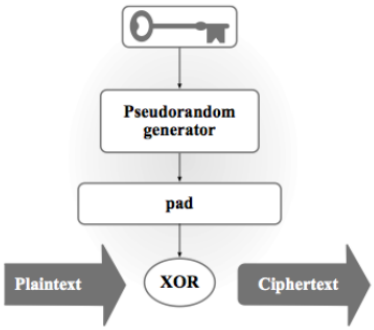
1. Fissiamo un Adv A PPT che attacca Π con probabilità di successo $\epsilon(n)$
2. Costruiamo A' PPT (detto la riduzione) che risolve X usando A come subroutine. A' non conosce il funzionamento interno di A; sa solo che serve ad attaccare Π. Quindi, data x, un'istanza di X, A' simula per A un'istanza della costruzione Π in modo tale che A pensa di star interagendo con Π, ha la stessa vista (o molto simile) che ha quando interagisce realmente con Π. Se A ha successo nel rompere Π sull'istanza che ha ricevuto da A', allora A' risolve l'istanza x del problema X che ha ricevuto in input con probabilità almeno $\frac{1}{p(n)}$
3. Le condizioni 1 e 2 implicano che A' risolve con probabilità almeno $\frac{\epsilon}{p(n)}$. Pertanto se $\epsilon(n)$ è non trascurabile allora anche $\frac{\epsilon}{p(n)}$ è non trascurabile.

4. Data l'assunzione di difficoltà sul problema X, concludiamo che nessun Adv A PPT può avere successo nel rompere Π con probabilità non trascurabile. Quindi, Π è computazionalmente sicuro.

Proviamo ora a costruire uno schema di cifratura PRG provandone la sicurezza.

Idea

Il one-time pad cifra tramite m ⊕ r, con stringa scelta uniformemente a caso. Nel nostro schema invece cifriamo con m ⊕ G(s), con G(s) stinga pseudocasuale, le due parti devono condividere il seme (chiave segreta).



Costruzione 3.17

Sia G un generatore pseudocasuale con un fattore di espansione l. Definiamo uno schema di cifratura a chiave privata per un messaggio di lunghezza l come segue:

- **Gen:** Prende in input 1^n, sceglie uniformemente $k \in \{0,1\}^n$ e da in output la chiave;
- **Enc:** Prende in input la chiave $k \in \{0,1\}^n$ e il messaggio $m \in \{0,1\}^{l(n)}$, da in output il testo cifrato;
- **Dec:** Prende in input la chiave $k \in \{0,1\}^n$ e il testo cifrato $c \in \{0,1\}^{l(n)}$, da in output il messaggio $m := G(k) \oplus c$

Per ogni $k \in \{0,1\}^n$ e per ogni $m \in \{0,1\}^l$, risulta:

$$G(k) \oplus c = G(k) \oplus (G(k) \oplus m) = m$$

Teorema 3.18

Se G è un PRG, la Costruzione 3.17 realizza uno schema di cifratura a chiave privata per messaggi di lunghezza fissa che ha cifrati indistinguibili in presenza di un eavesdropper.

Dim: Facciamo vedere che, per ogni Adv A PPT, esiste una funzione trascurabile negl(n) tale che:

$$Pr[PrivK^{eav}_{A,\Pi}(n) = 1] \leq \frac{1}{2} + negl(n)$$

Intuizione della prova:

- se Π usasse un pad uniforme invece di G(k), Π sarebbe identico allo schema one-time pad ed A vincerebbe nell'esperimento con prob. 1/2. Pertanto, se A fosse in grado di vincere nel nostro caso con probabilità significativamente maggiore di 1/2, allora A potrebbe essere usato per distinguere G(k) da una stringa uniforme.

Procediamo ora esibendo la riduzione. Costruiamo D che distingue G(k) da r, utilizzando A e la sua abilità nel capire quale messaggio tra m_0 ed m_1 è stato cifrato.
Quindi la probabilità di successo di D $\Rightarrow$ è collegata alla probabilità di successo di A.

Distinguisher D:
Riceve in input una stringa $\omega \in \{0,1\}^l(n)$

- Esegue A(1^n) per ottnere $m_0, m_1 \in \{0,1\}^l(n)$
- sceglie uniformemente $b \in \{0,1\}$ e pone $c := m_b \oplus \omega$ (Sceglie un bit a caso, prende la stringa ω fornita in input e la sua per costruire la cifratura)
- da c ad A ed ottiene da A il bit b'
- se b'=b dà in output 1, altrimenti dà in output 0.

Nel caso che otteniamo 1 possiamo dire che la cifratura provata dal distinguisher era fatta bene e ω era pseudocasuale e quindi abbiamo rotto il generatore.

Analizziamo il Distinguisher

Definiamo lo schema $\tilde{\Pi} = (\tilde{Gen}, \tilde{Enc}, \tilde{Dec})$. Ma questo schema è esattamente il one-time pad, infatti $\tilde{Gen}(1^n)$ riceve in input il parametro di dicurezza n e dà in output un chiave uniforme $k \in \{0,1\}^{l(n)}$. La segrettezza perfetta di $\tilde{\Pi} \Rightarrow Pr[PrivK^{eav}_{A,\tilde{\Pi}}(n) = 1] = \frac{1}{2}$. Ne consegue che:

1. Se ω è una stringa uniforme in $\{0,1\}^{l(n)}$ allora la visdta di A quando eseguito come subroutine da D = la vista di A in $Pr[PrivK^{eav}_{A,\tilde{\Pi}}(n)$. Poiché D da in output 1 quando A ha successo risulta:

$$Pr_{\omega \leftarrow \{0,1\}^{l(n)}}[D(\omega) = 1] = Pr[PrivK^{eav}_{A,\tilde{\Pi}}(n) = 1] = \frac{1}{2}$$

2. Se invece ω= G(k) , dove k è una stringa uniforme in $\{0,1\}^n$ allora la vista di A quando eseguito come subroutine da D = la vista di A in $Pr[PrivK^{eav}_{A,\Pi}(n)$. Poichè D dà in output 1 quando A ha successo, risulta:

$$Pr_{k \leftarrow \{0,1\}^n}[D(G(k)) = 1] = Pr[PrivK^{eav}_{A,\Pi}(n) = 1]$$

Procederemo dimostrando per assurdo, infatti sappiamo che la nostra differenza non può non essere trascurabile perchè se lo fosse il nostro avversario avrebbe probabilità $\geq \frac{1}{2}$ G per ipotesi è un PRG e D è PPT. Pertanto esiste una funzione negl(n) tale che:

$$|Pr_{\omega \leftarrow \{0,1\}^{l(n)}}[D(\omega) = 1] - Pr_{k \leftarrow \{0,1\}^n}[D(G(k)) = 1] = Pr[PrivK^{eav}_{A,\Pi}(n) = 1]| \leq negl(n)$$

ma allora:

$$|Pr[PrivK^{eav}_{A,\tilde{\Pi}}(n) = 1] - Pr[PrivK^{eav}_{A,\Pi}(n) = 1] \leq negl(n)|$$

ovvero

$$|\frac{1}{2} - Pr[PrivK^{eav}_{A,\Pi}(n) = 1] \leq negl(n)|$$

che implica

$$Pr[PrivK^{eav}_{A,\Pi}(n) = 1] \leq \frac{1}{2} + negl(n)$$

Poichè A è un Adv PPT arbitrario, possiamo concludere che Π ha cifrati indistinguibili in presenza di un eavesdropper.

Il vantaggio che otteniamo da questo nuovo schema è che con una chiave di pochi bit possiamo cifrare messaggi molto lunghi cosa non possibile con otp.

Limitazione concreta

Di seguito viene mostrata una limitazione concreta alla sicurezza di Π usando una limitazione concreta alla sicurezza di G:
Fissiamo n e assumiamo che G sia (t, ϵ)-pseudocasuale per n $\Rightarrow \forall D$che esegue in tempo al più t:

$$|Pr[D(r) = 1] - Pr[D(G(s)) = 1]| \leq \epsilon$$

5.3 Altre nozioni di sicurezza

5.3.1 Multi-messaggio

Verrà presentata ora una nozione di sicurezza per l'invio di più messaggi.

Nello schema che verrà mostrato vengono inviate liste di messaggi la cui unica limitazione è che le liste di messaggi inviati abbiano la stessa lunghezza.

$PrivK^{eav-mult}_{A,\Pi}(n)$

1. A(1^n) dà in output due liste della stessa lunghezza $M_0 = (m_{0,1}, ..., m_{0,t})$ ed $M_1 = (m_{1,1}, ..., m_{1,t}) : |M_{0,i}| = |M_{1,i}| \forall i$.
2. il Challenger sceglie $b \leftarrow \{0, 1\}$ e $k \leftarrow Gen(1^n)$ e calcola $c_i \leftarrow Enc_k(m_{b,i}) \forall i$
3. A(1^n) riceve $c = (c_1, ..., c_t)$ e dà in output $b' \in \{0, 1\}$
4. Se b = b' l'output dell'esperimento è 1 (A(1^n) vince); altrimenti 0.

Definizione 3.19 Uno schema di cifratura a chiave privata Π = (Gen, Enc, Dec) ha cifrature multiple indistinguibili in presenza di un eavesdropper se, per ogni Adv A PPT, esiste una funzione trascurabile negl tale che:

$$\Pr[PrivK^{eav-mult}_{A,\Pi}(n) = 1] \leq \tfrac{1}{2} + negl(n)$$

Dove la probabilità è calcolata sulla randomness usata da A, scelta della chiave, del bit b e andom bit usati da $Enc_k(\cdot)$

Indistinguibilitá rispetto a messaggi multipli

Π sicuro rispetto a $PrivK^{eav-mult}_{A,\Pi}(n) \Rightarrow$ Π sicuro rispetto a $PrivK^{eav}_{A,\Pi}(n)$ (caso speciale di un messaggio).
L'inverso non vale e lo mostreremo con un controesempio.

Proposizione 3.20: Esiste uno schema di cifratura a chiave privata che ha cifrature indistinguibili in presenza di un eavesdropper ma non ha cifrature multiple indistinguibili.

Dim: Consideriamo lo schema one-time pad (in breve otp). Segretezza perfetta ⇒ cifrature indistinguibili.
Sia A l'Adv che esegue e che attacca lo schema opt $PrivK^{eav-mult}_{A,otp}(n)$

Adv A

1. Sceglie $M_0 = (0^l, 0^l)$ ed $M_1 = (0^l, 1^l)$ e li passa al challenger (si noti e xor uguali danno 0)
2. Riceve dal Challenger la lista di cifratti c = (c_1, c_2)
3. Se $c_1 = c_2$ dà in output b'=0 altrimenti b'=1.

Analizziamo la probabilitá che b' = b. Lo schema otp é deterministico visto che la chiave di cifratura è calcolata in modo determinisctico:

se b=0 allora $c_1 = c_2 \Rightarrow Ada0$

se b=1 allora $c_1 \neq c_2 \Rightarrow Ada1$

Possiamo notare che A vince sempre con probabilità 1 e qindi non è sicuro secondo la definizione 3.19.

Teorema 3.11: Se Π uno schema di cifratura con Enc() deterministica, allora Π non puó avere cifrature multiple indistinguibili in presenza di un eavesdropper.

5.4 Attacchi Chosen-Plaintext

Per l'attacco chosen-plaintext, l'avversario dispone di cifrati corrispondenti a messaggi di propria scelta. Useremo un oracolo O, una scatola nera che cifra messaggi usando una chiave k con la quale l'avversario può interagire. L'avversario non conosce la chiave k e per comunicare con l'oracolo, invia richieste di cifratura, dette query, ad O specificando m ed ottenendo in risposta $Enc_k(m)$,Se $Enc_k(m)$ è randomizzato, O usa random bit nuovi ogni volta che riceve una query. L'avversario può inviare un numero polinomiale di query se si tratta di un PPT, dove a seconda dell'interazione adatta le query in relazione alle risposte che riceve dall'oracolo.

Siano $\Pi = (Gen, Enc, Dec)$ un avversario A e n parametro di sicurezza. Indichiamo con $A^{O(\cdot)}$ un avversario (algoritmo) che ha accesso all'oracolo $O(\cdot)$. L'esperimento viene modellato tramite questa terminologia:

$PrivK^{cpa}_{A,\Pi}(n)$

1. Il Challenger genera $k \leftarrow Gen(1^n)$ e setta l'oracolo O(·)
2. $A^{O(\cdot)(1^n)}$ stampa in output m_0 e m_1 : $|m_0| = |m_1|$
3. Sceglie $b \leftarrow \{0,1\}$ e calcola $c \leftarrow Enc_k(m_b)$
4. $A^{O(\cdot)(1^n)}$ riceve c e stampa in output b' $\in \{0,1\}$
5. Se b = b' l'output dell'esperimento è 1 quindi A vince, altrimenti 0.

Quindi abbiamo un Challenger che prepara l'oracolo con la chiave k nella sua memoria in modo tale che l'oracolo possa poi rispondere alle query dell'avversario. L'avversario viene poi mandato in esecuzione dove nel durante può interagire in un numero polinomiale di volte con l'oracolo inviando messaggio e

ricevendo cifrature di questi. Studiato lo schema l'avversario stampa in output la sua scommessa. A questo punto, il Challenger controlla se l'avversario ha indovinato.

Definizione 3.22 : Uno schema di cifratura a chiave privata Π = (Gen; Enc; Dec) ha cifrature indistinguibili rispetto ad attacchi di tipo chosen plaintext (CPA-sicuro) se, per ogni Adv A PPT, esiste una funzione trascurabile negl tale che:

$$Pr[PrivK^{cpa}_{A,\Pi}(n) = 1] \leq \frac{1}{2} + negl(n)$$

5.4.1 CPA cifrature multiple

Un Adv ha accesso ad un oracolo left-or-right, denotato con $LR_{k,b}$(k chiave di cifratura, b bit di scelta), che, su input m_0 ed m_1 restituisce $c \leftarrow Enc_k(m_b)$. Questo oracolo riceve sempre due messaggi e in base al valore del bit b decidee quale cifrare (0 il primo , 1 il secondo).

$PrivK^{LR-cpa}_{A,\Pi}(n)$

1. Genera $k \leftarrow Gen(1^n)$
2. Sceglie $b \leftarrow \{0,1\}$
3. $A^{LR_{k,b(\cdot)}}(1^n)$ dà in output b'$\in \{0,1\}$
4. Se b = b' l'output dell'esperimento è 1 (A vince) altrimenti 0

Questo esperimento p più potente di $PrivK^{cpa-mul}_{A,\Pi}(n)$ perchè l'avversario sceglie le due liste in maniera dinamica tenendo sempre conto dei messaggi cifrati precedentemente.

Definizione 3.23: Uno schema di cifratura a chiave privata Π = (Gen, Enc, Dec) ha cifrature multiple indistinguibili rispetto ad attacchi di tipo chosen plaintext se, per ogni Adv A PPT, esiste una funzione trascurabile negl tale che:

$$Pr[PrivK^{LR-cpa}_{A,\Pi}(n) = 1] \leq \frac{1}{2} + negl(n)$$

Π CPA-sicuro per cifrature multiple ⇒ Π CPA-sicuro. In questo caso vale anche l'inverso

Teorema 3.4: Ogni schema di cifratura a chiave privata CPA-sicuro risulta anche CPA-sicuro per cifrature multiple.
Discende che:

- é sufficiente provare che uno schema é CPA-sicuro (per una sola cifratura) per ottenere gratuitamente che é CPA-sicuro per cifrature multiple
- permette di concentrarci su schemi di cifratura per messaggi di lunghezza fissata

Π =(Gen, Enc, Dec) CPA-sicuro per messaggi di 1 bit ⇒ Π′ =(Gen', Enc', Dec') CPA-sicuro per messaggi di lunghezza arbitraria.
Struttura dello schema per messaggi di lunghezza arbitraria:

- Gen' = Gen
- $Enc'_k(m) = Enc_k(m_1)...Enc_k(m_l)$ dove $m = m_1...m_l$ ed $m_i \in \{0,1\}$
- $Dec'_k(c) = Dec_k(c_1)...Dec_k(c_l)$ dove $c = c_1...c_l$ e $c_i \in C \forall i$

La cifratura puó essere vista come la cifratura di messaggi multipli. Pertanto, la sicurezza CPA di Π' discende dalla sicurezza CPA di Π per messaggi multipli.

5.5 Funzioni Pseudocasuali

Per costruire schemi di cifratura che risultino CPA-sicuri si ha bisogno delle funzioni pseudocasuali (PRF) che si ispira alla nozione di generatore pseudocasuale. La differenza è che i PRG producono stringhe che sembrano casuali mentre le PRF sono funzioni che sembrano casuali. Non vengono utilizzate funzioni fissate in quanto si considera una distribuzione di funzioni. Pertanto, vengono utilizzate funzioni parametrizzate da una chiave, ovvero una funzione con due input.
F è efficiente se esiste un algoritmo polinomiale per calcolare F(k,x) dati k e x.
Negli usi tipici k viene scelto fissato, per cui si ha:

$$F_k(x) = F(k,x) \text{ ovvero } F_k : \{0,1\}* \rightarrow \{0,1\}*$$

Nel nostro caso il parametro di sicurezza n parametrizza 3 funzioni:

- $l_key(n)$: lunghezza della chiave
- $l_in(n)$: lunghezza dell'input
- $l_out(n)$: lunghezza dell'output

Per ogni $k \in \{0,1\}^{l_{key}(n)}$, F_k + definita solo per $x \in \{0,1\}^{l_{in}(n)}$ ed ha putput $y \in \{0,1\}^{l_{out}(n)}$. Quindi abbiamo $l_{key}(n) = l_{in}(n) = l_{out}(n) = n$(F preserva la lunghezza).

Possiamo anche dire poi che:

> F é pseudocasuale se F_k , per k scelta uniformemente a caso, é indistinguibile da una funzione scelta uniformemente a caso dall'insieme di tutte le funzioni aventi lo stesso dominio e lo stesso codominio.

Formalizziamo un distinguisher: Sia $Func_n = \{$ tutte le funzioni $f : \{0,1\}^n \rightarrow \{0,1\}^n\}$. Questa funzione può essere rappresentata in una tabella con 2^n righe ciascuna di n bit. Se concatenassimo tutte le righe della tabella, potremmo vedere f come una stringa di $2^n * n$ bit, quindi $|Func_n| = 2^{2^n * n}$. Possiamo provare a formalizzare un distinguer seguendo due idee:

- **Prima idea:** Diamo a D di tempo PPT le descrizioni di F_k ed f. D dovrebbe dare in output 1 all'incirca con la stessa probabilitá nei due casi. Problema la lunghezza di f è esponenziale $(2^n * nbit)$ e D è PPT non riuscirà nemmeno a leggere la sua descrizione.
- **Seconda idea:** dare a D accesso ad un oracolo O(·) che o implementa F_k , per k uniforme, o implementa f , per f uniforme. D puó chiedere il valore della funzione su un numero polinomiale di input x, non chiede mai due volte il valore per lo stesso x e al termine deve decidere se ha interagito con F_k o f.

Definizione 3.25: Sia $F : \{0,1\} * x\{0,1\}* \rightarrow \{0,1\}*$ una funzione con chiave efficiente che preserva la lunghezza. F é una funzione pseudocasuale se, per ogni distinguisher D PPT, esiste una funzione trascurabile negl tale che:

$$|Pr[D^{F_k(\cdot)}(1^n) = 1] - Pr[D^{f(\cdot)}(1^n) = 1]| \leq negl(n)$$

Informalmente: La differenza di probabilità tra la probabilità in cui un Distinguisher vinca l'esperimento interagendo con l'oracolo e dalla probabilità in cui vinca l'esperimento con la funzione scelta uniformemente a caso deve essere minore di una funzione trascurabile.

La prima parte di probabilità è calcolata sulla scelta uniforme di k e i random bit di D. La seconda sulla scelta uniforme di f e i random bit di D.
Nota:L'algoritmo D non riceve la chiave K, in quanto richiedendo all'oracolo $O(\cdot)$ una valutazione su x e ricevendo O(x) potrebbe calcolare $F_k(x)$ e controllare che $F_k(x) = O(x)$. Se l'uguaglianza sussiste, D con altissima probabilità sta interagendo con F_k.

5.5.1 Esempio di funzione non pseudocasuale

Sia F una funzione con chiave che preserva la lunghezza definita da $F(k, x) = k \oplus x$ (ogni funzione è definita con lo xor tra chiave e input).
Per ogni input x il valore $F_k(x)$ è uniformemente distribuito quando viene scelto in modo uniforme.

F non è pseudocasuale poiché i suoi valori su ogni coppia di punti sono correlati

Infatti D:

- Chiede all'oracolo valutazioni su $x_1 e x_2$
- ottiene $y_1 = O(x_1)$ e $y_2 = O(x_2)$
- Se $y_1 \oplus y_2 = x_1 \oplus x_2$ dà in output 1 (l'oracolo implementa una funzione pseudocasuale); altrimenti 0 (l'oracolo non implementa una funzione pseudocasuale).

É facile vedere che:

- Se $= O \equiv F_k$(equivalente), per ogni k, D dà in output 1 con probabilità 1 poiché:

$$y_1 \oplus y_2 = (x_1 \oplus k) \oplus (x_2 \oplus k) = x_1 \oplus x_2$$

- Se $O \equiv f$, D dà in output 1 con probabilità:

$$Pr[y_1 \oplus y_2 = x_1 \oplus x_2] = Pr[y_2 = x_1 \oplus x_2 \oplus y_1] = \frac{1}{2^n}$$

La differenza $|1 - \frac{1}{2^n}|$ non è trascurabile quindi F non è pseudocasuale.

5.6 Permutazioni pseudocasuali

Sia $Perm_n$ l'insieme di tutte le permutazioni $\{0,1\}^n$ (iniettiva). Anche qui possiamo vedere $f \in Perm_n$ può essere vista ancora come una tabelle dove ogni due righe della tabella sono differenti $|Perm_n| = 2^n$.
F è una permutazione con chiave $l_in(n) = l_out(n)$ e per ogni $k \in \{0,1\}^{l_{key}(n)}$ la funzione

$$F_k : \{0,1\}^{l_{in}(n)} \to \{0,1\}^{l_{out}(n)}$$

è uno a uno.

F é efficiente se esiste un algoritmo di tempo polinomiale per calcolare F(k, x), dati k ed x, cosí come un algoritmo di tempo polinomiale per calcolare F_k^{-1} , dati k e y:

F efficientemente calcolabile ed invertibile, data k

Proposizione 3.27 : Se F è una permutazione pseudocasuale e $l_{in}(n) \geq n$ allora F è anche una funzione pseudocasuale.

Visto che alcune costruzioni crittografiche richiedono alle parti oneste di usare anche F_k^{-1} dobbiamo tenere conto che anche Adv può conoscere questi valori e quindi ci serve una nozione più forte.

Definizione 3.28 Sia $F : \{0,1\} * x\{0,1\}* \rightarrow \{0,1\}*$ una permutazione con chiave efficiente che preserva la lunghezza. F è una permutazione pseudocasuale forte se per ogni distinguisher D PPT esiste una funzione trascurabile negl tale che:

$$|Pr[D^{F_k(\cdot),F_k^{-1}(\cdot)}(1^n) = 1] - Pr[D^{f(\cdot),f^{-1}(\cdot)}(1^n) = 1]| \leq negl(n)$$

La prima probabilità è calcolata sulle scelte uniforme di k e i random bit di D, la seconda sulla scelta uniforme di f e i random bit di D.

Informalmente: La probabilità che un distinguisher capisca che l'oracolo stia usando un inversa di una permutazione pseudocasuale meno la probabilità che l'oracolo stia usando l'inversa di una permutazione casuale deve essere minore di un valore trascurabile.

Nota: una permutazione pseudocasuale forte é anche una permutazione pseudocasuale.

5.6.1 Funzioni pseudocasuali e generatori pseudocasuali

Da una funzione pseudocasuale F possiamo costruire facilmente un generatore pseudocasuale G:

$$\text{G(s)} = F_s(1)||F_s(2)||...||F_s(l)$$

per ogni valore di l desiderato.
Informalmente: Costruiamo il generatore concatenando l'output della funzione pseudocasuale utilizzando come chiave della funzione il seed. Possiamo quindi costruire uno stram cipher a partire da F come segue:

Costruzione 3.29
Sia F una funzione pseudocasuale. Definiamo uno stream cipher (Init, GetBits), dove ogni chiamata di GetBits da in output n bit:

- **Init:** prende in input $s \in \{0,1\}^n$ e $IV \in \{0,1\}^n$, imposta $st_0 := (s, IV)$
- **GetBits:** prende in input st_i = (s, IV), calcola IV' := IV +1 e imposta y := $F_k(IV')$ e $st_{i+1} := (s, IV')$. Da in output (y, st_{i+1})

Un generatore pseudocasuale G dá immediatamente una funzione pseudocasuale F con lunghezza di blocco piccola. Sia:

$$G : \{0,1\}^n \rightarrow \{0,1\}^{2^{t(n)*n}}$$

Abbiamo quindi un generatore con fattore di espansione $2^{t(n)} * n$. Questo comporta che col tempo calcolare una stringa ha tempo esponenziale.
Possiamo dire quindi che questa costruzione è efficiente solo se t(n) = O(log n). Infatti:

$$2^{t(n)} * n = 2^{clogn} * n = n^c * n = poly(n)$$

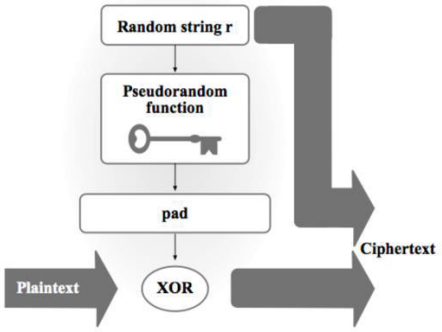
Notiamo che:

$$F : \{0,1\}^n x\{0,1\}^{clogn} \rightarrow \{0,1\}^n$$

associa stringhe di n bit a stringhe di input di c log n bit.

5.7 CPA sicuro da funzioni pseudocasuali

L'idea della costruzione consiste nello scegliere una stringa casuale r di n bit, dopodiché applicare una funzione pseudocasuale F_k su r, per produrre un "pad" pseudocasuale. Successivamente si calcola l'XOR tra il pad ed il messaggio m, producendo un cifrato.



Costruzione 3.30: Sia F una funzione pseudocasuale. Definiamo uno schema di crittografia a chiave privata per i messaggi di lunghezza n come segue:

- Gen: prende in input 1^n, sceglie in modo uniforme $k \in \{0,1\}^n$ e lo manda in output;
- Enc: prende in input la chiave $k \in \{0,1\}^n$ e il messaggio $m \in \{0,1\}^n$, sceglie uniformemente $r \in \{0,1\}^n$ e da in output il cifrato:

$$c := < r, F_k(r) \oplus m >$$

- Dec: prende in input la chiave $k \in \{0,1\}^n$ e il cifrato c=<r,s> e manda in output il messaggio:

$$m := F_k(r) \oplus s$$

5.7.1 Teorema 3.31

Se F è una funzione pseudocasuale, allora la costruzione 3.30 realizza uno schema di cifratura CPA-sicuro per messaggi di lunghezza n.

Dim: La dimostrazione per le prove di sicurezza basati su PRF procedono solitamente in due fasi:

- Fase 1: si considera una versione "ipotetica" della costruzione in cui la funzione pseudocasuale viene sostituita da una funzione casuale. La versione ipotetica dello schema non è implementabile nella realtà, ma risulta più facile da analizzare;
- Fase 2: si analizza lo schema ipotetico che utilizza la funzione casuale, dove si calcola qual è la probabilità di successo.

Nella prima fase si costruisce uno schema di cifratura ipotetico in cui al posto della funzione pseudocasuale F_k che viene usata da Π, viene usata una funzione f, scelta uniformemente a caso:
Quindi sia $\tilde{\Pi} = (\tilde{Gen}, \tilde{Enc}, \tilde{Dec})$ costruito a partire da $\Pi = (Gen, Enc, Dec)$ tale che:

- $\tilde{\Pi}$ usa $finFunc_n$ scelta uniformemente a caso (non efficiente f richiede spazio esponenziale in n per la memorizzazione);
- Π usa F_k dove k è scelta uniformemente a caso.

Per ogni avversario A PPT impostiamo un limite superiore che indichiamo con q(n) con n il numero di query che A rivolge al suo oracolo per la cifratura. Mostriamo che esiste una funzione trascurabile negl tale che:

$$|Pr[PrivK^{cpa}_{A,\Pi} = 1] - Pr[PrivK^{cpa}_{A,\tilde{\Pi}}]| \leq negl(n)$$

Usiamo A per costruire un distinguisher D per la funzione pseudocasuale tale che se A ha successo allora D, trovandosi di fronte a una funzione pseudocasuale o una funzione scelta uniformemente a caso, distingue quale dei due è stato utilizzato per la cifratura dei messaggi. Quindi:

- D ha accesso all'oracolo O($\cdot$) e deve stabilire se la funzione è "F_k, per k uniforme in $\{0,1\}^n$" oppure "$f \in Func_n$, uniforme".
- D emula l'esperimento $PrivK^{cpa}_{A,?}(n)$ per A ed osserva se A ha successo.

Distinguisher D:
Prende in input 1^n e ha accesso ad un oracolo O: $\{0,1\}^n \to \{0,1\}^n$:

1. Viene eseguito A(1^n), ogni volta che A interroga l'oracolo di cifratura su un messaggio $m \in \{0,1\}^n$, risponde come segue:
   a) Sceglie in modo uniforme $r \in \{0,1\}^n$;
   b) Interroga O(r) e ottiene la risposta y;
   c) Ritorna il cifrato <r, $y \oplus m$ > ad A.
2. Quando A manda in output i messaggi $m_0, m_1 \in \{0,1\}^n$, sceglie in modo uniforme $b \in \{0,1\}$ e dopo:
   a) Sceglie in modo uniforme $r \in \{0,1\}^n$;
   b) Interroga O(r) e ottiene la risposta y;
   c) Ritorna al challenger il cifrato <r, $y \oplus m_b$ > ad A.
3. Continua a rispondere alle query dell'oracolo di cifratura di A come prima finché A non restituisce un bit b'. L'output è 1 se b'=b altrimenti 0.

Spiegazione del distinguisher: D riceve come input il parametro di sicurezza 1^n e l'accesso ad un oracolo O : $\{0,1\}^n \to \{0,1\}^n$ al quale può inviare stringhe di n bit e ricevere stringhe di n bit che corrispondono al calcolo di una funzione, e deve capire che tipo di funzione è stata usata.

1. Il Distinguisher esegue l'algoritmo A come subroutine che gioca nell'esperimento $PrivK^{cpa}_{A,?}(n)$, quindi A può interrogare l'oracolo di cifratura su un messaggio. Quando A invia una query all'oracolo, verrà intercettato da D che risponde scegliendo uniformemente a caso una stringa r di n bit per poi mandare una query O(r) all'oracolo ricevendo come risposta il messaggio cifrato y. Dopodiché ritorna ad A il cifrato contenente una coppia avente la stringa r e il messaggio cifrato tramite lo XOR tra y e m. Questo passo va avanti per tutte le volte in cui l'avversario chiede cifrature di messaggi;

2. L'avversario A darà in output i due messaggi, m_0 e m_1, sui quali vuole essere sfidato. A questo punto D fa esattamente quello che avrebbe fatto il Challenger, ovvero sceglie un bit uniformemente a caso e sceglie una stringa r uniformemente a caso. Manda una query r all'oracolo ricevendo come risposta il cifrato y per poi ritornare all'avversario un cifrato contenete una coppia avente la stringa r e il messaggio cifrato tramite l'XOR del messaggio m_b e y. Successivamente, l'avversario potrebbe continuare a interagire col suo oracolo di cifrature e D continuerebbe a simulare le risposte, cosi come fatto al passo 1;

3. Quando A si sente pronto stampa in output b'. Se b = b' l'avversario ha vinto altrimenti no.

Analisi

L'algoritmo D computa in tempo polinomiale poiché A computa in tempo polinomiale. Inoltre, si noti che:

- se l'oracolo contiene una funzione pseudocasuale avremo che la vista di A come subroutine di D è uguale alla vista di A in $PrivK^{cpa}_{A,\Pi}(n)$ allora:

$$Pr_{k\leftarrow\{0,1\}^n}[D^{F_k(\cdot)}(1^n)=1] = Pr[PrivK^{cpa}_{A,\Pi}(n)=1]$$

- se l'oracolo contiene la funzione casuale la vista di A come subroutine di D è uguale alla vista di A in $PrivK^{cpa}_{A,\tilde{\Pi}}(n)$ allora:

$$Pr_{f\leftarrow Func_n}[D^{f(\cdot)}(1^n)=1] = Pr[PrivK^{cpa}_{A,\tilde{\Pi}}(n)=1]$$

Ma l'assunzione che F è pseudocasuale implica che $\exists$ negl(n) tale che:

$$|Pr_{k\leftarrow\{0,1\}^n}[D^{F_k(\cdot)}(1^n)=1] - Pr_{f\leftarrow Func_n}[D^{f(\cdot)}(1^n)=1]| \leq negl(n)$$

Ma questo implica che:

$$|Pr[PrivK^{cpa}_{A,\Pi}(n)=1] - Pr[PrivK^{cpa}_{A,\tilde{\Pi}}(n)=1]| \leq negl(n)$$

Pertanto possiamo analizzare lo schema ipotetico.

Nel secondo passo ovvero cercare di capire qual è la probabilità che l'avversario A possa avere successo. Mostreremo che:

$$Pr[PrivK^{cpa}_{A,\tilde{\Pi}}(n)=1] \leq \tfrac{1}{2} + \tfrac{q(n)}{2^n}$$

Nell'esperimento che stiamo considerando un mesasggio m cifrato $PrivK^{cpa}_{A,\tilde{\Pi}}(n)$ viene scelto da una stringa uniforme $r \in \{0,1\}^n$ il cifrato si ottiene come coppia <r, f(r) $\oplus$ m>.
Ora supponiamo che r* sia la stringa usata per produrre il cifrato di sfida, cioè $c* :=< r*, f(r*) \oplus m_b >$,possono verificarsi due casi:

- Il valore di r* non è mai usato prima da O(·) per rispondere alle query di A. A non sa nulla circa f(r*), che risulta uniforme ed indipendentemente distribuito dal resto dell'esperimento:

$$Pr[PrivK^{cpa}_{A,\tilde{\Pi}}(n)] = Pr[b'=b] = \tfrac{1}{2}$$

- Il valore di r* è stato usato in precedenza. A può capire facilmente se è stato cifrato m_0 o m_1. Infatti, disponendo di f(r*), poiché c* := < r*, f(r*) $\oplus$ m > risulta:

$$f(r*) \oplus (f(r*) \oplus m_b) = m_b$$

  Il valore f(r*) può essere recuperato dalla query in cui r* è usato: se A ha ricevuto dall'oracolo, per qualche m, il cifrato c:= <r*,s> = <r*, f(r*) $\oplus m$>, allora $s \oplus m = f(r*)$.

Adesso si calcola con che probabilità viene effettuata il secondo caso. Dato che A effettua al più q(n) query all'oracolo, al più q(n) valori distinti di r vengono usati, scelti uniformemente a caso in $\{0,1\}^n$ perché l'oracolo ogni volta che riceve una query sceglie una stringa casuale, scollegati dai precedenti. Pertanto, la probabilità che r*, scelto uniformemente, sia uguale ad un r precedente è al più $\frac{q(n)}{2^n}$(2^n considera tutti i valori possibili che possono essere generati).

Indichiamo con Repeat = {evento che r* sia uguale a qualche r scelto prima}.
Risulta $Pr[PrivK^{cpa}_{A,\tilde{\Pi}}(n) = 1]$ è uguale a:

$$Pr[PrivK^{cpa}_{A,\tilde{\Pi}}(n) = 1 \wedge Repeat] + Pr[PrivK^{cpa}_{A,\tilde{\Pi}}(n) = 1] \wedge \overline{Repeat} \leq \frac{q(n)}{2^n} + \frac{1}{2}$$

Poichè abbiamo mostrato che:

$$|Pr[PrivK^{cpa}_{A,\Pi}(n) = 1] - Pr[PrivK^{cpa}_{A,\tilde{\Pi}}(n) = 1]| \leq negl(n)$$

si ha:

$$\begin{aligned} |Pr[PrivK^{cpa}_{A,\Pi}(n) = 1] &\leq Pr[PrivK^{cpa}_{A,\tilde{\Pi}}(n) = 1]| + negl(n) \\ &\leq \frac{q(n)}{2^n} + \frac{1}{2} + negl(n) \text{ [somma di funzioni trascurabili]} \\ &\leq \frac{1}{2} + negl(n) \end{aligned}$$

Pertanto Π è CPA-sicuro.

Capitolo 6

Modalitá operative di cifratura

Le modalità operative sono utilizzate per la cifratura di messaggi molto lunghi.

6.1 Stream cipher

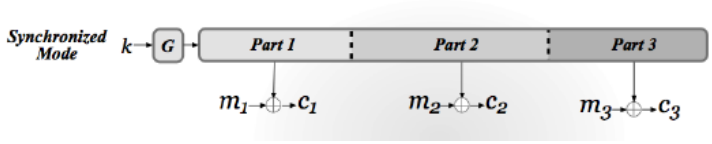
Precedentemente è stato mostrato come costruire un G_l con l fisso partendo da uno stream cipher, possiamo estendere quella costruzione per ottenere un generatore a lunghezza variabile: $G_\infty(s, 1^l)$ che invoca Init(s) e poi GetBits esattamente l volte. In questo caso cifratura e decifratura diventano:

$$c := G_\infty(k, 1^{|m|}) \oplus m \qquad\qquad m := c \oplus G_\infty(k, 1^{|c|})$$

Se le due parti mantengono uno stato i messaggi possono essere visti come un unico messaggio lungo. Lo stream cipher ha due modalità operative:

6.1.1 Modalità sincrona

Viene generata una lunga stringa pseudocasuale (pad) e una parte diversa viene usata per ogni messaggio:



In questa modalità la cifratura avviene nel seguente modo:

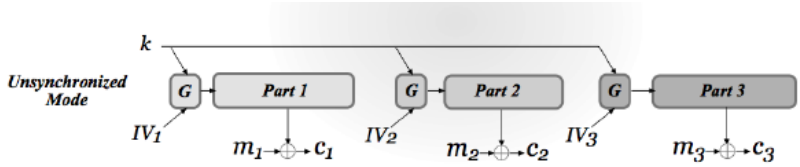
- st_0 := Init(k)
- Per cifrare m_1 il trasmittente invoca GetBits l_1 volte e calcola $c_1 := pad_1 \oplus m_1$
- Per decifrare c_1 il ricevente invoca GetBits l_1 volte e calcola $m_1 := pad_1 \oplus c_1$

Nelle cifrature successive si procede analogamente partendo sempre dallo stato raggiunto nell'ultima cifratura.

Nota: Se per ogni l, $G_\infty(k, 1^l)$ è un PRG allora lo schema di cifratura è EAV-sicuro.

6.1.2 Modalità asincrona

Ogni volta che dobbiamo cifrare un messaggio si utilizza il generatore dando non solo la chiave ma anche il vettore di inizializzazione(IV) in modo da produrre un pad necessario per cifrare il messaggio. Quindi ogni cifratura si comporta in maniera indipendente dalle altre cifrature ma al contempo ha una forte dipendenza con il vettore di inizializzazione.



La cifratura avviene nel seguente modo:

- Init($\cdot$): st_0 = Init(s, IV)
- Il generatore ha 3 input: $G_\infty(s, IV, 1^l)$
- Per cifrare un m di lughezza l usando la chiave k il trasmittente sceglie IV $\leftarrow \{0,1\}^n$ e calcola $c =< c_1, c_2 >:=< IV, G_\infty(k, IV, 1^l) \oplus m >$
- Per decifrare c usando la chiave k il ricevente calcola $m := G_\infty(k, c_1, 1^l) \oplus c_2$

Nota 1: Se per ogni l la funzione $F_k(IV) \stackrel{\text{def}}{=} G_\infty(k, IV, 1^l)$ è una PRF allora lo schema di cifratura è sicuro.
Nota 2: IV è scelto uniformemente a caso.

6.1.3 Stream cipher da una PRF

Si può costruire uno stream cipher a partire da una PRF:

Costruzione 3.29
Sia F una funzione pseudocasuale. Definiamo uno stream cipher (Init, GetBits), dove ogni chiamata di GetBits da in output n bit come segue:

- Init: input $s \in \{0,1\}^n$ e $IV \in \{0,1\}^n$ e imposta $st_0 := (s, IV)$
- GetBits: input $st_1 := (s, IV)$, calcola IV' := IV + 1 e setta $y := F_s(IV')$ e $st_{i+1} := (s, IV')$. Output(y, st_{i+1}).

Alternativa al calcolo di y:$y = F_s(IV|| < i >)$ con $IV \in \{0,1\}^{3/4n}$ e $i \in \{0,1\}^{n/n}$

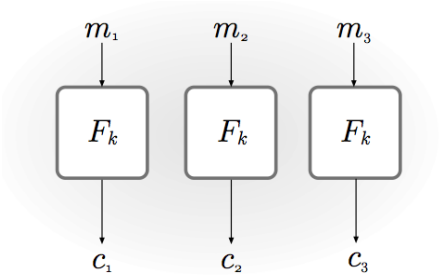
6.2 Cifrari a blocchi

Le modalità operative per i cifrari a blocchi sono metodi per cifrare messaggi di lunghezza arbitraria con cifrati "corti".
Sia F un cifrario a blocchi con lunghezza di blocco n. Sia $m = m_1 m_1 ... m_l \; m_i \in \{0,1\}^n \; \forall i = 1, ..., l.$ Se necessario all'ultimo blocco viene aggiunto un pad in modo tale che $|m_l| = n$.

Abbiamo diverse modalità operative per i cifrari a blocchi:

6.2.1 ECB (Electronic Code Book)

La cifratura avviene semplicemente passando il messaggio alla funzione pseudocasuale. Le cifrature dell'ECB nono sono indistinguibili ad un EAV, infatti se un blocco si ripete nel messaggio allora si ripetere anche nel cifrato, non si ha sicurezza CPA. Quindi ECB non dovrebbe mai essere usato.



6.2.2 CBC (Cipher Block Chaining)

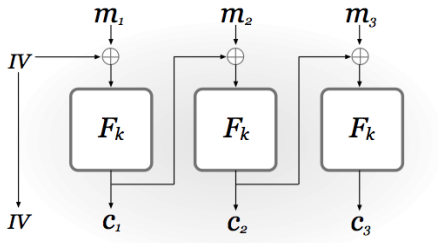
Supponiamo che il messaggio sia costituita da un certo numero di blocchi. L'idea è scegliere un vettore di inizializzazione, uniformemente a caso, per poi effettuare l'operazione XOR di questo con il messaggio stesso, dove il risultato verrà usato come input dalla funzione pseudocasuale per la generazione del cifrato. Questo cifrato viene usato per lo XOR del prossimo blocco del messaggio:

$$c_0 = IV \qquad c_i = F_k(c_{i-1} \oplus m_i) \qquad \forall i = 1, ..., l$$

Questa modalità è probabilistica perché il vettore di inizializzazione viene scelto ogniqualvolta si cifra un blocco del messaggio, in maniera uniformemente a caso. Quindi con 10 blocchi uguali avrei 10 cifrature diverse. Unico problema è che la cifratura deve essere effettuata sequenzialmente, per calcolare c_2 bisogna prima calcolare c_1:

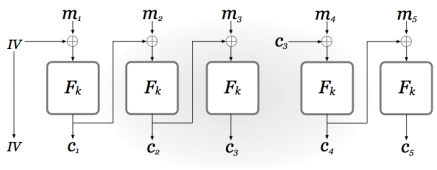
$$m_i = F_k^{-1}(c_i) \oplus c_{i-1} \qquad \forall i = 1, ..., l$$

Nota: Se IV non è scelto uniformemente a caso la cifratura non è più sicura.



6.2.3 Chained CBC mode

Una variante di CBC utilizzata è la modalità con concatenazione. Invece di usare un nuovo IV per ciascuna cifratura, l'ultimo blocco c_l del cifrato precedente viene utilizzato come IV per cifrare il prossimo blocco del cifrato. Questa modalità è efficiente perché nel momento in cui Alice vuole mandare un secondo messaggio a Bob non c'è bisogno che venga inviato anche il nuovo vettore di inizializzazione perché c_l sarà già a disposizione da parte di Bob. Quindi si tratta di una variante con stato. Tuttavia, non è così sicura quanto CBC in quanto è vulnerabile ad attacchi di tipo chosen-plaintext in cui attacco si basa sul fatto che il vettore IV è noto prima che la seconda cifratura abbia luogo.



Analisi dell'attacco:

Adv sa che $m_1 \in \{m_1^0, m_1^1\}$

- Osserva il primo cifrato $c :=< IV, c_1, c_2, c_3 >$ di $m_1 m_2 m_3$
- Chiede una cifratura all'oracolo di $m_4 m_5$ dove $m_4 = IV \oplus m_1^0 \oplus c_3$
- Ottiene $< c_4, c_5 >$
- Se $c_4 = c_1$ allora dà in output b'=0 ($m_1 = m_1^0$). Altrimenti b' = 1.

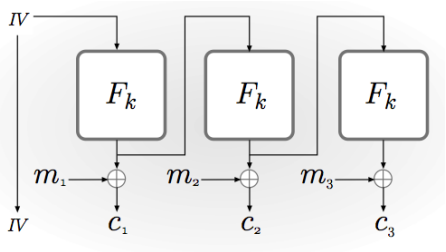
Si può verificare facilmente che $c_4 = c_1$ se e solo se $m_1 = m_1^0$:

$$c_4 = F_k(m_4 \oplus x_3) = F_k((IV \oplus m_1^0 \oplus c_3) \oplus c_3) = F_k(IV \oplus m_1^0)$$

mentre $c_1 = F_k(IV \oplus m_1)$. Una modifica ad uno schema crittografico può essere molto pericolosa.

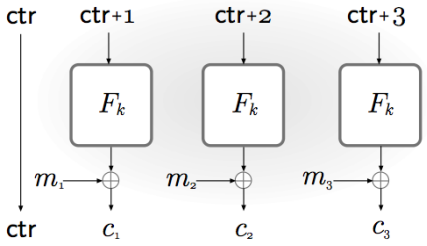
OFB (Output Feedback Mode)

Realizza uno stream cipher asincrono. L'idea è ispirata alla modalità precedente con la differenza che il blocco i-esimo del messaggio non viene coinvolto nell'operazione di XOR con il vettore di inizializzazione prima della funzione pseudocasuale, ma viene fatto dopo che il vettore di inizializzazione viene passato come input alla funzione per produrre il pad che verrà usato nell'operazione di XOR con il blocco i-esimo del messaggio. In questa modalità non è necessario che F sia invertibile, perché sia per cifrare che per decifrare si utilizza sempre un nuovo pad che viene utilizzato per cifrare il blocco i-esimo per poi essere utilizzato dalla funzione pseudocasuale successiva per generare un nuovo pad. Inoltre, non è necessario che il messaggio abbia una lunghezza multipla di n, in quanto la stringa pseudocasuale può essere troncata dove serve. Essendo i pad totalmente sganciati dai messaggi possono essere precomputati. OFB è CPA-sicuro se F è una funzione pseudocasuale.



6.2.4 CTR (Counter Mode)

Può essere vista come uno stream cipher asincrono costruito da un cifrario a blocchi. Si sceglie il valore di un contatore, uniformemente a caso ctr $\in \{0,1\}^n$, dopodiché il contatore viene incrementato e sottoposto alla funzione pseudocasuale così da produrre il pad che verrà coinvolto in un'operazione di XOR con il blocco i-esimo del messaggio in maniera da produrre il cifrato c_l. Per questa modalità, la decifratura non richiede che F sia invertibile. Inoltre, lo stream generato può essere troncato alla lunghezza del messaggio in chiaro e può essere generato in anticipo. CTR può essere parallelizzata perché per ogni blocco del messaggio si ha un valore specifico del contatore quindi il calcolo di un blocco i-esimo non dipende dalle informazioni del blocco precedente. In più, data questa indipendenza, la decifratura può essere selettiva quindi avere la possibilità di decifrare direttamente l'i-esimo blocco del cifrato. La lunghezza del blocco deve essere scelta con cura, per esempio se il ctr è di l bit allora dopo $2^{l/2}$ ctr uniformi lo stesso valore ricompare e quindi non è più sicuro. Anche qui utilizzando un IV non scelto uniformemente non viene garantita la sicurezza del nostro schema. Abbiamo due modi per



calcolare il nostro y:

- $y_0 = ctr \qquad y_i = F_k((ctr + i) mod 2^n) \qquad peri = 1, .., l$
- $y = F_k(ctr|| < i >)$ con $ctr \in \{0,1\}^{3/4n}$ e $i \in \{0,1\}^{n/4}$

CTR mode - Analisi

Teorema 3.32 : Se F é una funzione pseudocasuale, allora la modalitá CTR é CPA-sicura.
Dim: Utilizziamo la stessa strategia dimostrativa usata in precedenza. Sia $\tilde{\Pi} = (\tilde{Gen}, \tilde{Enc}, \mathbf{Dec})$ costrutio a partire da $\Pi = (Gen, Enc, Dec)$ tale che:

- $\tilde{\Pi}$ usa $f \in Func_n$ scelta uniformemente a caso
- Π usa F_k dove k è scelta uniformemente a caso

Per ogni Adv A PPT sia q(n) un limite superiore al numero di query che $A(1^n)$ rivolge al suo oracolo per la cifratura.
É facile far vedere (come prima) che esiste una funzione trascurabile negl tale che:

$$|Pr[PrivK^{cpa}_{A,\Pi}(n) = 1] - Pr[PrivK^{cpa}_{A,\tilde{\Pi}}(n) = 1]| \leq negl(n)$$

mostriamo quindi che:

$$Pr[PrivK^{cpa}_{A,\tilde{\Pi}}(n) = 1] \leq \frac{1}{2} + \frac{2*q(n)^2}{2^n}$$

da cui risulta che:

$$Pr[PrivK^{cpa}_{A,\Pi}(n) = 1] \leq \frac{1}{2} + \frac{2*q(n)^2}{2^n} + negl(n)$$

6.2.5 Schemi di cifratura nonce-based

Definizione 3.34 : Uno schema di cifratura a chiave privata nonce-based è una tripla Π = (Gen, Enc, Dec) di algoritmi PPT tale che:

- k ← $Gen(1^n)$, algoritmo probabilistico di generazione della chiave k dove la chiave $k \in K : |k| \geq n$
- c := Enc_k(nonce,m), algoritmo deterministico di cifratura dove $m \in \{0,1\}^*$, nonce $\in \{0,1\}^*$, $k \in K$ e $c \in \{0,1\}^*$
- m := Dec_k(nonce, c), algoritmo deterministico di decifratura dove $c \in \{0,1\}^*, k \in K$, nonce $\in \{0,1\}^*$ ed $m \in \{0,1\}^*$. Dec(nonce, c) restituisce $\perp$ in caso di errore

Correttezza: Per ogni n, per ogni k restituito da Gen(1^n), per ogni nonce $\in \{0,1\}^*$ e per ogni $m \in \{0,1\}^*$, risulta Dec_k(nonce,Enc_k(nonce,m)) = m.
Iil nonce deve essere diverso per ogni messaggio. La sicurezza si definisce come per i normali schemi di cifratura, modificando opportunamente gli esperimenti: l'avversario A sceglie il nonce, che deve essere sempre diverso.
La modalitá CTR, settando crt = nonce, produce uno schema di cifratura nonce-based CPA-sicuro. Il nonce è utile nel caso in cui generare randomness è difficile o troppo costoso, oppure per cifrare pochi messaggi.

Capitolo 7

Autenticazione

L'autenticazione dei messaggi deve garantire due proprietà:

1. **Confidenzialità:** la cifratura deve essere usata per prevenire che ascoltatori non autorizzati visionino il contenuto dei messaggi.
2. **Autenticità, Integrità:** ogni parte dovrebbe essere in grado anche di identificare il mittente del messaggio, essere certo di chi ha inviato i messaggi e capire se il messaggio è integro o è stato modificato.

La cifratura non risolve i problemi sopra riportati, se prendiamo ad esempio $Enc_k(m) = G(k) \oplus m$ dove G è un PRG costruito a partire da uno stream cipher. Dall'algoritmo di cifratura sopra indicato otteniamo uno che il cifrato è uno XOR e se modifichiamo un bit di quest'ultimo il messaggio rimarrà cifrato ma avremo come effetto secondario di aver modificato il contenuto originario del messaggio. Questo tipo di attacco si applica anche a OFB e CTR. Anche le cifrature ECB e CBC non sono esenti da questo tipo di attacco:

- ECB: modifiche anche di un solo bit del cifrato influenzano ancora il messaggio in chiaro, anche se la posizione non é controllabile
- CBC: cambiando il j-esimo bit di IV si cambia il j-esimo bit del primo blocco del messaggio: $m_1 := F_k^{-1}(c_1) \oplus IV$. Tutti gli altri blocchi restano inalterati, ma il primo contiene informazioni importanti.

7.1 MAC (message authentication code)

I codici MAC permettono di identificare ed autenticare il mittente e permettono di controllare l'integritá del messaggio. Entrambe le parti devono condividere un segreto che Adv non conosce.

Def. 4.1: Un codice MAC consiste in tre algoritmi PPT (Gen, Mac, Vrfy) dove:

- $k \leftarrow Gen(1^n)$, $|k| \geq n$, genera le chiavi;
- $t \leftarrow Mac_k(m) m \in \{0,1\}^*$, dove t è il tag associato ad m;
- $b := Vrfy_k(m,t), b \in \{0,1\}$

tali che, per ogni n, per ogni k dato in output da Gen(1^n) e per ogni $m \in \{0,1\}^*$ abbiamo $Vrfy_k(m, Mac_k(m)) = 1$.
Se esiste una funzione $l(n)$ e lo schema è definito solo per $m \in \{0,1\}^{l(n)}$, allora parleremo di MAC a lunghezza fissa per messaggi di lunghezza l(n).
Una classe importante è quella degli schemi deterministici in cui:

- $Mac_k(\cdot)$ è deterministico;
- La verifica si dice canonica e consiste nel ricalcolo del tag da parte di chi riceve e verifica che sia uguale a quello ricevuto. Quindi dato la coppia (m,t), ricalcola $Mac_k(m) = t'$ e verifica che t' = t.

7.1.1 Sicurezza del MAC

Un MAC per essere sicuro, nessun avversario PPT dovrebbe essere in grado di generare un tag su qualsiasi nuovo messaggio, che non sia stato autenticato ed inviato in precedenza da una delle parti, significa che l'avversario non deve poter creata una coppia (m',t') valida, siccome non conosce k. Definiamo quindi con **codici non falsificabili** gli esperimenti con la seguente struttura :
Sia $\Pi = (Gen, Mac, Vrfy)$ e A un Adv generico ed n il parametro di sicurezza

Mac-forge$_{A,\Pi}(n)$

1. Il challenger genera $k \leftarrow Gen(1^n)$ e setta l'orocolo $Mac_k(\cdot)$;
2. $A^{Mac_k(\cdot)}(1^n)$ invia all'oracolo un certo numero di query. Sia Q l'insieme delle query richieste. Alla fine $A^{Mac_k(\cdot)}(1^n)$ dà in output (m,t);
3. $A^{Mac_k(\cdot)}(1^n)$ ha successo se e solo se:
   - $Vrfy_k(m,t) = 1$ (ovvero si tratta di un tag lecito)
   - $m \notin Q$ (non è un messaggio che Adv ha precedentemente richiesto all'oracolo e quindi il tag è stato prodotto manualmente dall'avversario)

   In questo caso, l'output dell'esperimento é 1; altrimenti é 0.

La rottura di un MAC si tratta di una condizione di vincita da parte dell'avversario in cui dando in output la coppia (m,t) tale che il tag t è un tag valido, ovvero Vrfy$_k$(m, t)=1, e l'avversario non ha ottenuto t per m con una precedente richiesta all'oracolo che vale a dire l'avversario ha prodotto t da solo, quindi produce una **contraffazione**.

Def. 4.2: Un codice per l'autenticazione di messaggi Π = (Gen, Mac, Vrfy) é non falsificabile esistenzialmente rispetto ad un attacco adattivo di tipo chosen-message (existentially unforgeable under an adaptive chosen-message attack) se, per ogni A PPT, esiste una funzione trascurabile negl, tale che:

$$Pr[\text{Mac-forge}_{A,\Pi}(n)] \leq negl(n)$$

Abbiamo dato una definizione molto forte perchè così non ci dobbiamo preoccupare della specifica implementazione.

Attacchi di tipo replay

La nostra definizione è vulnerabile ad attacchi di tipo replay dove il nostro Adv può produrre un nuovo Mac su un messaggio per cui il Mac è stato già calcolato:
dato(m,t) produce(m, t') dove t $\neq t'$. In alcuni applicazioni è un problema.

7.1.2 MAC forte

Def. 4.2: Un codice per l'autenticazione di messaggi Π = (Gen, Mac, Vrfy) é fortemente sicuro (o è un Mac forte) se, per ogni A PPT, esiste una funzione trascurabile negl, tale che:

$$Pr[\text{Mac-sforge}_{A,\Pi}(n)] \leq negl(n)$$

dove Mac-sforge$_{A,\Pi}(n)$ è come $Mac - forge_{A,\Pi}(n)$ ma l'insieme delle query Q contiene coppie (m,t). Notiamo che:

- In Mac-forge$_{A,\Pi}(n)$: A produce (m,t') con $m \in Q \rightarrow$ non è un successo per A;
- In Mac-sforge$_{A,\Pi}(n)$: A produce (m,t') con $(m,t) \in Q \rightarrow$ non è un successo per A.

Mac-sforge$_{A,\Pi}(n)$

1. Il challenger genera $k \leftarrow Gen(1^n)$ e setta l'orocolo $Mac_k(\cdot)$;
2. $A^{Mac_k(\cdot)}(1^n)$ invia all'oracolo un certo numero di query. Sia Q l'insieme delle query richieste. Alla fine $A^{Mac_k(\cdot)}(1^n)$ dà in output (m,t);
3. $A^{Mac_k(\cdot)}(1^n)$ ha successo se e solo se:
   - $Vrfy_k(m,t) = 1$ (ovvero si tratta di un tag lecito)
   - $(m,t) \notin Q$ (non è una coppia che Adv ha precedentemente richiesto all'oracolo e quindi il tag è stato prodotto manualmente dall'avversario)

   In questo caso, l'output dell'esperimento é 1; altrimenti é 0.

Proposizione 4.4: Se Π = (Gen; Mac; Vrfy) é un Mac sicuro che usa la verifica canonica, allora Π é anche fortemente sicuro.

Side-channel attack

Un attacco di tipo side-channel usa l'osservazione fisica dell'hardware o l'analisi dei messaggi di errore per carpire informazioni circa bit segreti. L'attacco puó:

- sfruttare variazioni temporali, il consumo di energia, o la radiazione elettromagnetica che un dispositivo genera durante l'esecuzione di un algoritmo
- misurare le risposte agli errori di formattazione o ai guasti indotti
- sfruttare informazione rilasciata dalla computazione circa i pattern di accesso della CPU alla memoria cache
- sfruttare bug o peculiaritá del linguaggio di programmazione usato
- una combinazione dei differenti tipi di informazione rilasciata

Un esempio di questo attacco è analizzare i tempi di risposta della funzione *strcmp()* del linguaggio C. La funzione è stata creata in modo da ottimizzare i tempi di confronto di due stringhe, infatti confronta byte per byte e in caso che siano diversi da errore altrimenti continua finché non arriva all'ultimo byte dove dirà se sono uguali o meno. Un Adv analizzando il tempo di risposta potrebbe iniziare ad inviare una stringa con 1 al primo byte e tutti 0 e se riceve subito errore cambiare il byte in 0, per le altri posizioni della stringa procede nello stesso modo analizzando sempre il tempo di risposta finché non falsifica il MAC. Un esempio reale di questo attacco si è avuto nella pirateria dei giochi XBoX 360.

7.1.3 Costruzioni sicure di codici

Le funzioni pseudocasuali sono uno strumento naturale per la costruzione di codici per l'autenticazione dei messaggi,intuitivamente t = $F_k(m)$ è molto prossimo a un valore uniformemente distribuito su stringhe di n bit cioè 2^n

COSTRUZIONE 4.5
Sia F una funzione pseudocasuale. Si definisce un MAC di lunghezza fissa per i messaggi di lunghezza n:

- **Mac**: prende in input una chiave $k \in \{0,1\}^n$ e un messaggio $m \in \{0,1\}^n$, restituisce in output il tag t := $F_k(m)$. (Se $|m| \neq |k|$ allora non produrrà nulla)
- **Vrfy**: prende in input una chiave $k \in \{0,1\}^n$, un messaggio $m \in \{0,1\}^n$ e un tag $t \in \{0,1\}^n$, restituisce in output 1 se e solo se t := $F_k(m)$. (Se $|m| \neq |k|$ allora produrrà 0)

7.1.4 Teorema 4.6:

Se F è una funzione pseudocasuale, allora la costruzione 4.5 realizza un Mac sicuro di lunghezza fissata per messaggi di lunghezza n.

Dim : Sia A PPT e sia $\tilde{\Pi} = (\tilde{Gen}, \tilde{Mac}, \tilde{Vrfy}$ (ipotetico f $\in Func_n$ invece di F_k).Risulta:

$$Pr[\text{Mac-forge}_{A,\tilde{\Pi}}(n) = 1] \leq 1/2^n$$

Infatti, per ogni $m \notin Q$ il valore t = f(m) è uniformemente distribuito in $\{0,1\}^n$ dal punto di vista di A.
Vogliamo mostrare che esiste una funzione negl() tale che:

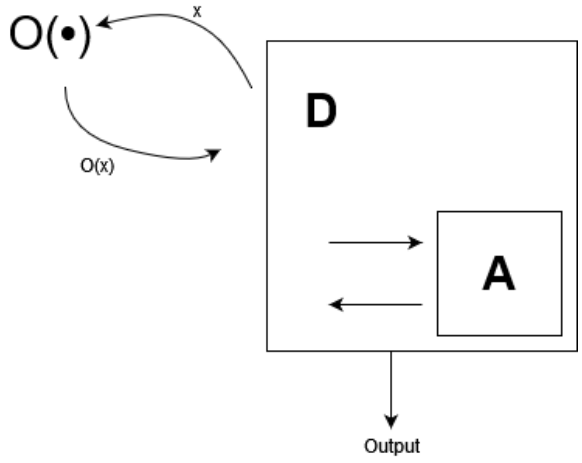
$$|Pr[\text{Mac-forge}_{A,\tilde{\Pi}}(n) = 1] - Pr[\text{Mac-forge}_{A,\Pi}(n) = 1] \leq negl(n)$$

che combinato con il risultato precedente implica:

$$Pr[\text{Mac-forge}_{A,\Pi}(n) = 1] \leq 1/2^n + negl(n)$$

Informalmente: Vogliamo ottenere che la probabilità che l'avversario vinca nell'esperimento ipotetico - la probabilità in cui l'avversario vinca nel nostro esperimento abbiamo un valore trascurabile.

7.1.5 Costruiamo il distinguisher



Per provare la nostra affermazione costruiremo un distinguisher D che ha accesso ad un oracolo con una funzione, e il suo obiettivo è di stabilire se è casuale o pseudocasuale. Per ottenere ciò D emulerà l'esperimento Mac-forge$_{A,?}(n)$, infatti cercherà di far credere al nostro esperimento che stia arrivando un input reale e se l'esperimento vince sull'input fornito da D allora la funzione usata dall'oracolo è pseudocasuale.

Distinguisher D: D prende in input il parametro di sicurezza 1^n e ha l'accesso ad un oracolo $O : \{0,1\}^n \to \{0,1\}^n$, funziona come segue:

1. Esegue A(1^n). Ogni volta che A interroga (invia delle query) il suo oracolo MAC su un messaggio (ad es. , ogni volta che A richiede un tag su un messaggio m), risponde a questa query nel seguente modo:
   Interroga O con m e ottiene la risposta t; restituisce t ad A.
2. Quando A da in output (m,t) al termine della sua esecuzione, esegue:
   a) Interroga O con m e ottiene la risposta $\hat{t}$.
   b) Se $\hat{t}$ = t e A non ha mai interrogato il suo oracolo MAC con$m$ emette 1 (l'avversario è riuscito a contraffare il messaggio); altrimenti, emette 0.

D è PPT se A è PPT, inoltre:

- Se O($\cdot$) è pseudocasuale A eseguito come subroutine di D ha la stessa visione che avrebbe nell'esperimento Mac-forge$_{A,\Pi}(n)$
- Se O($\cdot$) è casuale A eseguito come subroutine di D ha la stessa visione che avrebbe nell'esperimento Mac-forge$_{A,\tilde{\Pi}}(n)$

Quindi abbiamo che:

$$Pr[D^{F_k(\cdot)}(n) = 1] = Pr[\text{Mac-forge}_{A,\Pi}(n) = 1]$$

e

$$Pr[D^{f(\cdot)}(n) = 1] = Pr[\text{Mac-forge}_{A,\tilde{\Pi}}(n) = 1]$$

Ma poichè F è pseudocasuale, esiste allora una funzione negl(n) tale che:

$$|Pr[D^{F_k(\cdot)}(n) = 1] - Pr[D^{f(\cdot)}(n) = 1]| \leq negl(n)$$
$$\Downarrow$$
$$|Pr[Mac\text{-}forge_{A,\tilde{\Pi}}(n) = 1] - Pr[Mac\text{-}forge_{A,\Pi}(n) = 1]| \leq negl(n)$$

7.2 MAC a lunghezza arbitraria

Preso uno schema di cifratura $\Pi' = (Mac', Vrfy')$ come un MAC sicuro a lunghezza fissa per messaggi di lunghezza n. Se prendiamo un messaggio m = $m_1 m_2 ... m_d$ di d blocchi di lunghezza n, abbiamo diverse idee per costruire Π come schema di cifratura MAC a lunghezza arbitraria:

1. Possiamo provare a creare un'autenticità per ognuno dei blocchi:
   $m_1 m_2 ... m_d \to t_1 t_2 ... t_d$ dove $t_i = Mac'_k(m_i)$
   Questo previene l'invio di blocchi non autenticati ma ci espone ad attacchi di riordino: $m_2 m_1 \leftarrow t_2 t_1$
2. per evitare il riordino si può usare un indice di blocco $t_i = Mac'_k(i||m_i)$, ma questo ci espone ad attacchi di troncamento. L'attacco di troncamento sfrutta il fatto che preso un messaggio autenticato abbiamo già tutti i prefissi e quindi si possono usare per autenticare un nuovo messaggio

3. Aggiungiamo ad ogni blocco la lunghezza del messaggio: $t_i = Mac'_k(l||i||m_i)$, questo non previene attacchi di tipo mix-and-match; presi due messaggi della stessa lunghezza si può costruire un nuovo messaggio alternando i blocchi dei due messaggi:
$m_1...m_d \to t_1...t_d$ e $m'_1...m'_d \to t'_1...t'_d \quad \to \quad m_1 m'_2...m_d \to t_1 t'_2...t_d$

4. Aggiungiamo infine un identificatore casuale del messaggio ad ogni blocco: $t_i = Mac'_k(r||l||i||m_i)$

COSTRUZIONE 4.7
Sia $\Pi' = (Mac', Vrfy')$ un MAC a lunghezza fissa per messaggi di lunghezza n. Definiamo un MAC come segue:

- **Mac**: prende in input una chiave $k \in \{0,1\}^n$ e un messaggio $m \in \{0,1\}^*$ di lunghezza $l < 2^{n/4}$ (diverso da 0), trasforma m in d blocchi: $m_1, ..., m_d$ ognuno di lunghezza n/4(aggiungendo un padding all'ultimo se necessario). Sceglie infine un identificativo $r \in \{0,1\}^{n/4}$.
  For i =1,...,d calcola $t_i = Mac'_k(r||l||i||m_i)$ finché i,l sono codificati come stringhe di lunghezza n/4. Da in output il tag **t** := <**r,** $t_1, ..., t_2$>.
- **Vrfy**: rende in input una chiave $k \in \{0,1\}^n$ e un messaggio $m \in \{0,1\}^*$ di lunghezza $l < 2^{n/4}$ e un tag t := <r, $t_1, ..., t_2$>, trasforma m in d' blocchi: $m_1, ..., m_{d'}$ ognuno di lunghezza n/4(aggiungendo un padding all'ultimo se necessario). Da in output 1 se e solo se d' = d e Vrfy'$_k(r||l||i||m_i) = 1$ per $1 \leq ileqd$

Teorema 4.8 : Se Π' è un Mac a lunghezza fissa per messaggi di lunghezza n, la Costruzione 4.7 produce un Mac sicuro per messaggi di lunghezza arbitraria.

Dim: Nell'ipotesi in cui Π' è sicuro:

- A non può introdurre un nuovo blocco con un tag valido;
- le informazioni aggiunte ad ogni blocco prevengono il riuso

Quindi sia Π il Mac della costruzione 4.7, mostreremo che:

$$Pr[Mac\text{-}forge_{A,\Pi}(n) = 1] \leq negl(n)$$

Sia (m, t = < r, t_1, ... >) l'output finale di A definiamo due eventi:

- **Repeat**:lo stesso identificatore è presente in due dei tag restituiti dall'oracolo nell'esperimento.
- **Newblock**:almeno uno dei blocchi $r||l||i||m_i$ non è stato autenticato in precedenza tramite query all'oracolo.

Risulta $Pr[Mac\text{-}forge_{A,\Pi}(n) = 1]$(probabilità che l'avversario produca una contraffazione e che quindi vinca l'esperimento) uguale a:
$Pr[Mac\text{-}forge_{A,\Pi}(n) = 1 \wedge Repeat] + Pr[Mac\text{-}forge_{A,\Pi}(n) = 1 \wedge \overline{Repeat}]$ Inoltre $Pr[Mac\text{-}forge_{A,\Pi}(n) = 1 \wedge \overline{Repeat}]$ risulta uguale a:

$$Pr[Mac\text{-}forge_{A,\Pi}(n) = 1 \wedge \overline{Repeat} \wedge Newblock]$$
$$+Pr[Mac\text{-}forge_{A,\Pi}(n) = 1 \wedge \overline{Repeat} \wedge \overline{Newblock}]$$

Dalle precedenti risulta

$$\begin{gathered} Pr[Mac\text{-}forge_{A,\Pi}(n) = 1] \\ \leq Pr[Repeat] + Pr[Mac\text{-}forge_{A,\Pi}(n) = 1 \ \wedge Newblock] \\ + Pr[Mac\text{-}forge_{A,\Pi}(n) = 1 \ \overline{Repeat} \wedge \overline{Newblock}] \end{gathered}$$

Consideriamo i tre termini separatamente.

Claim 4.9 $Pr[Repeat]$ è trascurabile.
Dim: Sia q(n) il numero polinomiale di query all'oracolo effettuate da A. L'oracolo usa r_i tale che $|r_i = n/4$ scelti uniformemente a caso in $\{0,1\}^{n/4}$, sfruttando il paradosso del compleanno:

$$Pr[Repeat] = Pr[r_i = r_j peri \neq j] \leq \frac{q(n)^2}{2^{n/4}}$$

Claim 4.10 $Pr[\text{Mac-forge}_{A,\Pi}(n) = 1 \ \wedge Newblock]$ è trascurabile.
Il claim si poggia sulla sicurezza di Mac'.
Costruiamo infatti un avversario A' PPT che attacca lo schema Π'(utilizzato da Π. A' ha successo nel produrre un tag valido su un messaggio non autenticato in precedenza con:

$$Pr[\text{Mac-forge}_{A',\Pi'}(n) = 1] \geq Pr[\text{Mac-forge}_{A,\Pi}(n) = 1 \ \wedge Newblock]$$

Poichè Π' è per ipotesi sicuro, allora la prima probabilità deve essere trascurabile: $Pr[\text{Mac-forge}_{A,\Pi}(n) = 1 \ \wedge Newblock]$ deve essere altrettando.
A' esegue A come subroutine e risponde alle query di A per tag di specifici messaggi come segue:

- sceglie $r \leftarrow \{0,1\}^{n/4}$
- divide il messaggio in modo opportuno
- effettua le query necessarie al proprio oracolo Mac'$(\cdot)$.

A' tiene traccia di tutte le query che ha rivolto a Mac'$(\cdot)$. Quando A dà in output (m, t= <r, $t_1, ...$>), A' controlla se l'evento Newblock si verifica.
Se ciò accade, A' trova il primo blocco $r||l||1||m_i$ che non è stato usato in una query per $Mac'(\cdot)$ e dà in output $(r||l||1||m_i, t_i)$. Altrimenti non dà nulla in output.
La "vista" che A ha quando eseguito come subroutine da A' é distribuita esattamente come in Mac-forge$_{A,\Pi}(n) \Rightarrow$ le probabilità di Mac-forge$_{A,\Pi}(n)$ e Newblock non cambiano.

Se Mac-forge$_{A,\Pi}(n) = 1$, il tag restituito da A é corretto su ciascun blocco del messaggio m, ed in particolare sul blocco che non é stato mai usato in una query da A' verso l'oracolo Mac' e che A' dá in output. Quindi ogni volta che Mac-forge$_{A,\Pi}(n) = 1$ e Newblock occorre, Mac-forge$_{A',\Pi'}(n) = 1 \Rightarrow$ il Claim 4.10 vale.

Terzo termine: $Pr[Mac\text{-}forge_{A,\Pi}(n) = 1 \ \overline{Repeat} \wedge \overline{Newblock}]$. Faremo vedere che se $Mac\text{-}forge_{A,\Pi}(n) = 1$ ma Repeat non si verifica allora deve per forza verificarsi Newblock. Pertanto risulta:

$$Pr[Mac\text{-}forge_{A,\Pi}(n) = 1 \ \overline{Repeat} \wedge \overline{Newblock}] = 0$$

Sia q(n) il num. di query che fa A all'oracolo ed r_i l'identificatore della i-esima query.
Se Repeat non si verifica, allora $r_1, ..., r_{q(n)}$ sono tutti distinti.
Se $r \in \{r_1, ..., r_{q(n)}\}$ allora Newblock si verifica! Altrimenti r = r_j per qualche j, e i blocchi $r||l||1||m_1$, $r||l||1||m_1$, ..., di (m, t= < r, $t_1, ...$ >) possono essere stati autenticati soltanto durante la j-esima query all'oracolo.
Sia m_j il messaggio usato da A per la sua j-esima query all'oracolo ed l_j la sua lunghezza. Consideriamo due casi:

- $l \neq l_j \Rightarrow$ blocchi autenticati durante la j-esima query hanno tutti $l_j \neq l$ in seconda posizione. Quindi, nessun blocco di m puó essere stato autenticato durante la j-esima query. Per esempio: $r||l||1||m_1$ non è stato autenticato durante la j-esima query $\Rightarrow$ Newblock si verifica
- $l = l_j \Rightarrow$ poiché $m \neq m_j$ (per ipotesi Mac-forge$_{A,\Pi}(n) = 1$) esiste un indice i per cui $m \neq m_j^{(i)}$, dove $m_j^{(i)}$ denota l'i-esimo blocco di m_j . Il blocco $r||l||1||m_1$ non é stato autenticato durante la j-esima query $\Rightarrow$ Newblock si verifica

Pertanto mettendo assieme i risultati provati abbiamo:

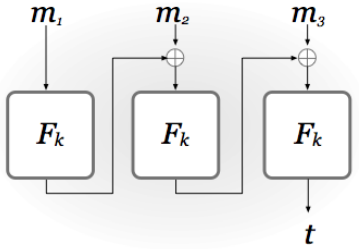
$$Pr[\text{Mac-forge}_{A,\Pi}(n) = 1] \leq negl(n) + negl(n) + 0 \leq negl(n)$$

7.3 CBC-MAC

La costruzione di codici di autenticazione per messaggi di lunghezza arbitraria non è efficiente la dimensione del messaggio aumenta di 4 volte:

$$\text{m} : |\text{m}| = \text{dn} \Rightarrow \text{tag t} : |\text{t}| = 4\text{dn}$$

Per fare un esempio un messaggio di 1 giga diventa di 4 giga.



Divido il messaggio in t parti, usiamo il primo messaggio per generare un valore intermedio che verrà usato in XOR con la seconda parte del messaggio che verra poi computata da F_k, il processo continuerà fino ad ottenere il tag t.

COSTRUZIONE 4.11
Sia F una funzione pseudocasuale e fissiamo una funzione di lunghezza $l > 0$. La costruzione di base del CBC-MAC è come segue:

- **Mac**: prende in input una chiave $k \in \{0,1\}^n$ e un messaggio m di lunghezza *l(n) * n*, imposta l = l(n) e fa le seguenti operazioni:
  1. Divide m in m $= m_1, ...m_l$ dove ogni m_i ha lunghezza n.
  2. Imposta $t_0 := 0^n$. Poi for i = 1 to l imposta $t_i := F_k(t_{i-1} \oplus m_i)$

  Da in output il tag t_l
- **Vrfy**: prende in input una chiave $k \in \{0,1\}^n$, un messaggio m e il tag t. Se m non è della lunghezza l(n) * n da in output 0. Altrimenti da in output 1 se e solo se t = Mac$_k$(m)

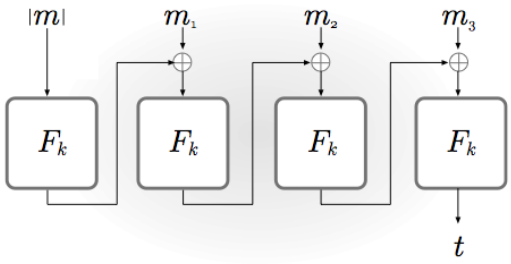
7.3.1 Teorema 4.12

: Sia l(·) un polinomio. Se F è una PRF allora la costruzione 4.11 è un Mac sicuro per messaggi di lunghezza l(n) * n.
Nota:si intende sicura soltanto quando la lunghezza dei messaggi è fissata e decisa in precedenza. I vantaggi sono che fornisce un Mac a lunghezza fissa molto più efficiente rispetto alla costruzione precedente.
Confronto con la modalità di cifratura CBC:

1. CBC-MAC non usa un vettore di inizializzazione IV. Questo aspetto é cruciale. CBC-MAC che usa un IV casuale non é sicuro;
2. La modalitá di cifratura CBC dá in output tutti i valori t_i intermedi. CBC-MAC solo t_l. Se viene modificata per dare in output tutti gli t_i, CBC-MAC non è più sicuro.

Esistono due modi per estendere la costruzione:

1. Anteporre ad m la sua lunghezza |m|, codificata come una stringa di n bit e calcolare il CBC-MAC di base;
2. L'algoritmo di generazione della chiave sceglie due chiavi e poi calcola il CBC-MAC di base usando k_1 e sarà t. Da poi in output t* := $F_{k_2}(t)$.



Il problema del CBC-MAC è la funzione pseudo-casuale, infatti quest'ultima è invocata molte volte per messaggi lunghi, quindi abbiamo un efficienza bassa.

7.4 GMAC (GayMAC) e Poly1305

Per risolvere i problemi del CBC-MAC sono nati diversi standard. Lo schema generale di questi standard è il seguente:

- Utilizzano una funzione ϵ(n)-difference universal $h_{k_h}(m)$(valutazione di una funzione sul messaggio m) e PRF $F_{k_F}(\cdot)$.
- il tag del MAC è t = <r, s> (r randomizzato e scelto uniformemente) dove s = $h_{k_h}(m) + F_{k_F}(r)$

Il GMAC e Poly1025 usano un block cipher e $h_{k_h}(m)$ basata su polinomi, utilizzano infine campi finiti diversi. Sono migliori in termini di prestazioni perchè la PRF viene valuta soltanto una volta e l'altra parte del calcolo si riduce al calcolo di un polinomio secondo le regole di Horner. Abbiamo riduzioni di sicurezza più strette rispetto a CBC-MAC

Regola di Horner Indichiamo con N il grado del polinomio, quindi sia:

$$P_N(x) = a_0x^N + a_1x^{N-1} + ... + a_{N-1}x + a_N$$

La regola di Horner ci permette di risolvere il polinomio svolgendo N addizioni e N moltiplicazioni, scrivendolo nella seguente forma:

$$P_N(x) = a_N + x(a_{N-1} + x(a_{N-2} + ... + x(a_1 + a_0x)...))$$

Il valore di tale polinomio si può calcolare sfruttando una definizione ricorsiva:
$p_0 = a_0$
$p_{k+1} = p_kx + a_{k+1}$
dove $0 \leq k \leq N-1$

7.4.1 Definizione 4.14

Una funzione con chiave h : $K_n x M_n \to T_n$ è $\epsilon(n)$-difference universal se per tutti gli n, per ogni m, m' $\in M_n$ ed ogni $\Delta \in T_n$, risulta:

$$Pr[h_k(m) - h_k(m') = \Delta] \leq \epsilon(n)$$

per k scelta uniformemente in K_n.
Informalmente: La probabilità che la differenza fra i due valore che la funzione h restituisce quando valuta m e m' sia uguale ad un certo Δ deve essere al più $\epsilon(n)$.
Maggiore sarà la differenza fra questi due valori e maggiore saranno il numero di chiavi disponibili.
Nota: $\epsilon(n) \geq \frac{1}{|T_n|}$, e h deve essere efficientemente calcolabile.

Schema generico:

1. **Gen:** su input 1^n, sceglie uniformemente $k_h \in k_n$ e $k_F \in \{0,1\}^n$
2. Mac: su input (k_h, k_F) ed m $\in M_n$, sceglie r uniformemente in $\{0,1\}^n$ e calcolta il tag

$$\text{t} = <\text{r}, h_{k_h}(m) + F_{k_F}(r)>$$

3. Vrfy: su input (k_h, k_F), $m \in M_n$ e t = <r,s> dà in output 1 se

$$\text{s} = h_{k_h}(m) + F_{k_F}(r)$$

Un campo finito (insieme con 2 operazioni con stesse proprietà in cui gli elementi sono un numero finito) F_q contenente q elementi esiste per ogni potenza di un numero primo.
Un polinomio non nullo di grado l su un campo finito ha al più l zeri.
Sia F un campo finito. Sia $\text{F}^{<l}$ l'insieme dei vettori con meno di l elementi. Sia M = $\text{F}^{<l}$. Sia $m = m_1, ...mm_{t-1} \in M$, con t $\leq l$ e sia m_t una codifica della lunghezza di m. Sia m(X) il polinomio:

$$M(x) = m_1 X^t + m_2 X^{t-1} + ... + m_t X$$

La funzione h : F x $\text{F}^{<l} \to F$ con chiave definita come:

$$h_k(\text{m}) = \text{m(k) (valutazione del polinomio nel punto k)}$$

è una funzione $l/|\text{F}|$-difference universal.

Prova : Siano m,m' $\in F^{<l}$ e $\Delta \in F$. Consideriamo il polinomio

$$\text{P(X) = m(X) - m'(X) - } \Delta$$

P è un polinomio non nullo di grado al più l. Risulta:

$$Pr_{k \in F}[h_k(m) - h_k(m') = \Delta] = Pr_{k \in F}[P(k) = 0] \leq \frac{l}{|F|}$$

poichè P ha al più l radici (zeri)
Nota: l = casi favorevoli, |F| = casi possibili (numero elementi del campo).

conclusione

La costruzione GMAC usa un cifrario a blocchi per istanziare la PRF $\text{F}(\cdot,\cdot)$ e la funzione $l/|\text{F}|$-difference universal basata sui polinomi definita sul campo F_{2^128}, contiene 2^128 elementi.
La costruzione Poly1305 è realizzata in modo simile, ma usa il campo F_p definito per il primo p = $2^{130} - 5$, per comodità però si riduce il campo a 2^128. Infatti il tag finale è calcolato come:

$$t =< r, [h_{k_h}(m) + F_{k_F}(r) mod 2^{128}]$$

7.5 Attacchi chosen ciphertext (CCA)

Attacchi di tipo chosen ciphertext sono attacchi in cui l'avversario ha anche l'abilità di ottenere i messaggi in chiaro corrispondenti a cifrati di propria scelta, cioè produce un cifrato e chiede all'altra parte di decifrarlo. Nell'esperimento questa capacità viene modellata dando all'avversario l'accesso a due oracoli per la cifratura e decifratura:

$\text{PrivK}^{cca}_{A,\Pi}(n)$

1. Il Challenger sceglie k $\leftarrow$Gen(1^n)
2. A riceve in input 1^n e ha accesso agli oracoli $\text{Enc}_k(\cdot)$ e $\text{Dec}_k(\cdot)$. Dá in output m_0 e m_1 della stessa lunghezza
3. Il Challenger sceglie b $\leftarrow \{0,1\}$ e calcola c* $\leftarrow Enc_k(m_b)$
4. A(1^n) riceve c*, continua ad accedere agli oracoli $\text{Enc}_k(\cdot)$ e $\text{Dec}_k(\cdot)$. Non può chiedere la decifratura di c*. Alla fine, dá in output b'$\in \{0,1\}$ (se chiedesse la cifratura la vittoria sarebbe banale)
5. Se b=b', l'output dell'esperimento è 1 (A vince), altrimenti 0.

Definizione 3.33 Uno schema di cifratura a chiave privata Π = (Gen; Enc; Dec) ha cifrature indistinguibili rispetto ad un attacco di tipo chosen ciphertext (CCA-sicuro) se, per ogni Adv A PPT, esiste una funzione trascurabile negl tale che:

$$Pr[PrivK^{cca}_{A,\Pi}(n) = 1] \leq \frac{1}{2} + negl$$

Dove la probabilità è calcolata sulla randomicità usata da A, e la randomicità usata nell'esperimento sulla scelta della chiave, del bit b e i random bit usati da $Enc_k(\cdot)$.

Tutti gli schemi che abbiamo studiato sono insicuri rispetto ad attacchi di tipo CCA.

Esempio di attacco CCA

L'Adv A che segue, di tipo CCA, é in grado di distinguere le cifrature con prob. 1.:

- L'avversario inizia scegliendo due messaggi $m_0 = 0^l$ (stringa tutti 0) e $m_1 = 1^l$ (stringa tutti 1).
- L'Adv riceve dal challenger il cifrato c = <r, s>
- Adv calcola s' invertendo il primo bit di s e chiede all'oracolo $Dec_k(\cdot)$ di decifrare c' =< r , s' > (la modifica di un bit nel cifrato si ripercuote anche nel messaggio decifrato)
- Se $Dec_k(c') = 10^{l-1}$ (primo bit uno e poi tutti 0) allora b' = 0; altrimenti $Dec_k(c') = 01^{l-1}$ (primo bit 0 e poi tutti 1) b' = 1.
- Adv dà in output b' (l'avversario guardando il primo bit sa quale messaggio è stato devifrato).

Osservazione: Ogni schema di cifratura in cui i cifrati possono essere "manipolati" in modo controllato non possono essere CCA-sicuri. Quindi Sicurezza CCA $\Rightarrow$ "non malleabilità".

Con non malleabilità chiediamo che un cifrato modificato nel momento della decifratura sia restituito un errore ($\perp$) oppure il messaggio m' sia completamente scorrelato dal messaggio originale m.

7.5.1 Padding-Oracle attack

Si tratta di un attacco CCA potente. L'attacco richiede solo l'abilità di determinare se un cifrato modificato viene decifrato correttamente oppure no, non c'è necessita dell'oracolo di decifratura $Dec_k(\cdot)$. In questa modalità di cifratura un messaggio deve avere lunghezza multipla della lunghezza del blocco di L byte, se risulta più piccolo si aggiunge b byte di padding(Se è necessario un byte si aggiunge 0x01, due byte 0x0202 e così via). Nella fase di decifratura il pad viene rimosso, prima di restituire il testo in chiaro. Nel caso in cui il pad non risulta essere corretto viene restituito un errore $\perp$. Lo schema che viene considerato è CBC.

Esempio di attacco: Supponiamo di avere un cifrato di 3 blocchi, cioè un messaggio costituito da m_1 e m_2 che viene esteso con del padding.
Nella cifratura si ha c=<c_0,c_1,c_2> = <IV,c_1,c_2>. Quando si decifra abbiamo che m_2 termina con del padding. Se l'avversario modifica un byte di una qualsiasi posizione del cifrato c_1 avremo che il risultato dell'operazione di decifratura comporta nell'avere un messaggio m_1' uguale m_1 eccetto per un byte modificato.
L'avversario utilizza questa relazione per calcolare b. L'avversario modifica alcuni byte di c_2 (cifrato del messaggio col padding) e chiede la decifratura. Se fallisce, la decifratura significa che viene trovato un valore diverso dal valore del pad. Se non fallisce, il byte modificato è del messaggio e non del pad, così facendo l'avversario ripete l'attacco modificando un altro byte, fintantoché non ottiene la lunghezza di b.
Ottenuto b, l'avversario può calcolare i byte del messaggio tramite operazioni di XOR, l'avversario sa che m_2 termina con =0xB0xb...b

- B è il byte sconosciuto del messaggio in chiaro che Adv vuole conoscere
- 0xb ... b è il pad con il byte b ripetuto b volte.

Adv cercherà quindi di trasformare B in un valore di padding. Per esempio abbiamo un padding = 3 quindi avremo 0xB0x03x03x03, quello che vuole tentare di fare l'avversario è di ottenere 0xB-1(0xB $\oplus$ i)0x04 0x0 4 0x04, dove lo XOR di 0xB con i deve dare 0x04, quando ha ottenuto il valore desiderato, quindi effettuando lo XOR tra il valore b+1 e i può ottenere il valore originale del byte B. A questo punto procede a rieseguire l'attacco per tutti i byte del messaggio.

Nota: Il CAPTCHA è CCA-sicuro se non lo fosse sarebbe possibile prendere il cifrato che ci viene inviato da un web server e automatizzare un bot in modo che applichi un Padding-Oracle conto il Server che si occupa di controllare la decifratura.

7.6 Cifratura autenticata

La cifratura autenticata mette assieme gli schemi di cifratura, che offrono confidenzialità/riservatezza, e MAC, che offrono integrità/autenticità, in modo da avere un unico strumento da utilizzare per le comunicazioni sicure. Così da avere schemi a chiave privata che soddisfano la nozione di segretezza rispetto ad attacchi CCA e la nozione di integrità, dove è non falsificabile da attacchi chosen message.

Definizione 4.16 Uno schema di cifratura a chiave privata Π é non falsificabile (unforgeable) se, per ogni avversario A ppt, esiste una funzione trascurabile negl tale che:

$$Pr[Enc - forge_{A,\Pi}(n) = 1] \leq negl(n)$$

Sia Π = (Gen, Enc, Dec), A un avversario, n parametro di sicurezza.
Definiamo $Enc - forge_{A,\Pi}(n)$:

1. Esegue Gen(1^n) per ottenere la chiave k.
2. Adv a prende in input 1^n e ha l'accesso all'oracolo di cifratura $Enc_k(\cdot)$. Adv da in output il testo cifrato c.
3. Sia m := $Dec_k(c)$, e denotiamo con Q l'insieme di tutte le query richieste da Adv all'oracolo di cifratura. L'esperimento da in output 1 se e solo se m $\neq \perp$ e $m \notin Q$ (la cifratura non è stata ottenuta servendosi dell'oracolo ma direttamente dall'avversario).

L'avversario ha poco potere perché accede solo all'oracolo di cifratura.

Definizione 4.17: Uno schema di cifratura a chiave privata Π é uno schema di cifratura autenticata se é CCA-sicuro e non falsificabile.
Siano:

- Π_E = (Gen, Enc, Dec) è uno schema CPA-sicuro
- Π_M = (Mac, Vrfy) è un codice per l'autenticazione di messaggi sicuro

Abbiamo tre combinazioni naturali:

1. **Cifra ed autentica:** Il mittente trasmette il cifrato <c,t> dove: c $\leftarrow Enc_{k_E}(m)$ e t $\leftarrow Mac_{k_M}(m)$
   Il ricevente decifra c per recuperare m. Poi verifica il tag t. Restituisce m se non si verificano errori, $\perp$ altrimenti.
2. **Autentica e poi cifra**: Il mittente trasmette il cifrato <c> calcolato come: t = $Mac_{k_M}(m)$ e c $\leftarrow Enc_{k_E}(m||t)$
   Il ricevente decifra c per recuperare m||t. Poi verifica il tag t. Restituisce m se non si verificano errori, $\perp$ altrimenti.
3. **Cifra e poi autentica**: Il mittente trasmette il cifrato < c; t > calcolato come: c $\leftarrow Enc_{k_E}(m)$ e t $\leftarrow Mac_{k_M}(c)$
   Il ricevente verifica il tag t. Se risulta corretto, decifra c e dá in output m. Altrimenti, restituisce $\perp$.

Analizziamo i tre approcci quando istanziati con "componenti generiche": Π_E CPA-sicuro e Π_M fortemente sicuro.

Cifra ed autentica

Purtroppo non garantisce neanche i requisiti minimali di segretezza. Infatti il Mac non garantisce alcuna segretezza di per sé, puó rilasciare informazioni importanti su m. In più CBC-mac é deterministico, ogni volta che lo stesso messaggio viene cifrato, il tag rimane inalterato: <c,t> <c',t> non è più CPA-sicuro.
Quindi poiché la maggior parte dei Mac é deterministica, l'approccio non funziona.

Autentica e poi cifra

Ipotizziamo di avere uno schema di cifratura CPA-sicuro che decifra il modalità CBC. Possiamo sfruttare il Padding-oracle attack. Visto che nella cifratura abbiamo m || t , possiamo interrogare l'oracolo di decifratura finché non riscontriamo un errore nel padding e ci viene restituito il messaggio "padding scorretto". Ottenuto il presunto messaggio si può provare a ad autenticarlo per ottenere t, e se otteniamo t' = t restituisce 1 e abbiamo autenticato il messaggio corretto altrimenti da un errore di "autenticazione fallita".

Si può provare a far restituire un singolo errore ma questo comporta diversi problemi (difficoltà di debugging, aumento della complessità delle applicazioni, ecc). In più non rende immune da timing attack.

7.6.1 Cifra e poi autentica

Funziona per ogni generica istanziazione delle primitive, assumendo che il Mac sia fortemente sicuro.

Nota: Indichiamo con k_E la chiave dell'algoritmo di cifratura e con k_M la chiave dell'algoritmo di autenticazione.

Costruzione 4.18:
Sia Π_E = (Enc, Dec) uno schema di crittografia a chiave privata e sia Π_M = (Mac, Vrfy) un MAC, dove in ogni caso la generazione della chiave viene eseguita semplicemente scegliendo una chiave di n bit. Si definisce uno schema di crittografia a chiave privata (Gen', Enc', Dec') come segue:

- Gen': prende in input 1^n, sceglie indipendentemente k_E $k_M \in \{0,1\}^n$ uniformemente e manda in output la chiave (k_E, k_M);
- Enc': prende in input la chiave (k_E, k_M) e un messaggio m, calcola c $\leftarrow Enc_{K_E}(m)$ e $t \leftarrow Mac_{K_M}(c)$. Manda in output <c,t>:
- Dec': prende in input la chiave (k_E, k_M) e il cifrato <c,t>, prima controlla se $\text{Vrfy}_{K_M}(c,t) = 1$, allora manda in output Dec_{K_E}, altrimenti restituisce $\perp$.

Lo schema utilizza un Mac sicuro per autenticare le cifrature, vuol dire che l'avversario non è in grado di falsificare le cifrature. Ma se un avversario nell'esperimento in cui gioca non ha possibilità di falsificare, l'oracolo di decifratura per l'avversario è inutile perché, non appena riceverà qualcosa prodotta dall'avversario, si renderà conto che non è conforme a ciò che si aspetta, restituendo un errore. In qualche modo, il Mac depotenzia l'avversario e lo rende equivalente ad un avversario che ha esattamente lo stesso potere di un avversario che gioca nell'esperimento $\text{Priv}^{cpa}_{A,\Pi}$.

Teorema 4.19:
Sia Π_E uno schema di cifratura CPA-sicuro a chiave privata e sia Π_M un codice di autenticazione di messaggi fortemente sicuro. Allora la costruzione 4.18 produce uno schema di cifratura autenticata.

7.6.2 Dimostrazione Teorema 4.19

Sia Π' = (Gen', Enc', Dec') lo schema risultante dalla costruzione 4.18. Dobbiamo provare che sia non falsificabile e CCA-sicuro.
Diremo che <c,t> è valido rispetto a (K_E,K_M) se $\text{Vrfy}_{k_M}(c,t) = 1$.
Sia A un Adv PPT che sferra un attacco di tipo CCA contro Π'. Diremo che <c,t> è nuovo se A non ha ricevuto <c,t> dall'oracolo di cifratura.
Sia ValidQuery = "A sottomette un cifrato nuovo <c,t> all'oracolo di decifratura che risulta valido".

7.6.3 Claim 4.20

La Pr[ValidQuery] è trascurabile.
Intuizione: una query valida ⇒ A ha falsificato una nuova coppia <c,t> valida in Mac-sforge(n).
Sia q(n) un limite superiore polinomiale al numero di query fatte da A all'oracolo.
Consideriamo un Adv A_M che attacca Π_M, sfruttando la capacitá di A di produrre <c,t> valido.
A_M "gioca" all'interno di Mac-sforge(n) e simula l'esperimento $\text{Priv}^{cca}_{A,\Pi'}(n)$.
Precisamente A_M riceve in input 1^n e ha accesso all'oracolo $Mac_{K_M})\cdot$ e agisce come segue:

A_M prende in input 1^n e ha accesso all'oracolo $\text{Mac}_{k_M}(\cdot)$.

1. Sceglie uniformemente $K_E \in \{0,1\}^n$ e $i \in \{1,...mq(n)\}$.
2. Esegue A sull'input 1^n. Quando A fa una query di cifratura all'oracolo per il messaggio m, risponde come segue:
   i. Calcola $c \leftarrow Enc_{k_E}(m)$
   ii. Invia c come query all'oracolo MAC e riceve come risposta t. Da in output <c,t> ad A.

   Il testo cifrato del challenger è preparato nello stesso modo (con $b \in \{0,1\}$ scelto uniformemente ed usato per selezionare m_b).
   Quando A fa una query di decifratura all'oracolo per il testo cifrato <c,t>, risponde come segue: Se è la query i-esima dell'oracolo di decriptazione da in output (c,t). Altrimenti:
   i. Se <c,t> è una risposta già registrata precedentemente dall'oracolo di cifratura per m, ritorna m
   ii. Altrimenti ritorna $\perp$ (errore).

Informalmente: A_M "scommette" che la i-esima richiesta di decifratura all'oracolo che A effettua nell'esperimento simulato $\text{Priv}^{cca}_{A,\Pi'}(n)$ é la prima nuova e valida query che A invia. Se indovina A_M dá in output un tag valido su un messaggio c che non é stato mai sottoposto all'oracolo $Mac_{k_M}(\cdot)$. Pertanto A_M vince!. Ricordiamo che A_M è PPT quindi anche A è PPT.

A quando viene eseguito come subroutine di A_M si comporta nello stesso modi di come se fosse eseguito da $Priv^{cca}_{A,\Pi'}(n)$ fino a quando occorre ValidQuery. Quindi possiamo dire che le query di cifratura sono perfettamente simulate da A_M, le query di decifrature fino a ValidQuery sono perfettamente simulate.
Se <c,t> è nuovo e ValidQuery non si è verificata, la decifratura è $\perp$.

Pertanto Pr[ValidQuery] in Mac-sforge(n) è identica a Pr[ValidQuery] in $Priv^{cca}_{A,\Pi'}$. Poichè:

$$Pr[A_M \text{ indovina i}] = \frac{1}{q(n)}$$

risulta:

$$Pr[\text{Mac-sforge}_{A_M,\Pi_M}(n) = 1] \geq \frac{Pr[ValidQuery]}{q(n)}$$

Informalmente: La probabilità che A_M produca una contraffazione è almeno la probabilità che ValidQuery è la i-esima query scelta da lui (Maggiore perché l'Adv potrebbe avere successo).

Ma per ipotesi Π_M è fortemente sicuro, quindi:

$$Pr[\text{Mac-sforge}_{A_M,\Pi_M}(n) = 1] \leq \text{negl(n)}.$$

Essendo un limite superiore per ValidQuery, discende che:

$$Pr[ValidQuery] \leq Pr[\text{Mac-sforge}_{A_M,\Pi_M}(n) = 1] \cdot q(n) \leq negl'(n).$$

Ricordiamo: Il prodotto di un polinomio per un valore trascurabile è sempre trascurabile.

Possiamo usare ora il Caim 4-20 per provare che Π' è sicuro.
Dal claim segue che Π' è non falsificabile. A' in Enc-forge$_{A',\Pi'}(n)$ è una versione ristretta di A in $PrivK^{cca}_{A,\Pi'}(n)$, ed usa soltanto l'oracolo di cifratura. Pertanto quando A' dà in output <c,t> ha successo solo se è valido e nuovo, pertanto il Claim 4.20 ha mostrato che accade con probabilità trascurabile.

Resta da mostrare che Π' è CCA-sicuro. A è un Adv PPT che attacca Π' con un attacco CCA. Risulta $Pr[Priv^{cca}_{A,\Pi'(n)=1}]$:

$$\begin{aligned} &= Pr[Priv^{cca}_{A,\Pi'(n)=1} \wedge ValidQuery] + Pr[Priv^{cca}_{A,\Pi'(n)=1} \wedge \overline{ValidQuery}] \\ &\leq Pr[ValidQuery] + Pr[Priv^{cca}_{A,\Pi'(n)=1} \wedge \overline{ValidQuery}] \end{aligned}$$

7.6.4 Claim 4.21

Esiste una funzione trascurabile negl(n) tale che:

$$Pr[PrivK^{cca}_{A,\Pi'}(n) = 1 \wedge \overline{ValidQuery}] \leq \frac{1}{2} + negl(n)$$

Proviamo il claim usando la sicurezza CPA di Π_E. Definiamo A_E che usa A contro Π' per vincere $PrivK^{cpa}_{A_e,\Pi_E}(n)$.
A_E cerca quindi di rompere l'esperimento CPA, può effettuare query di cifratura all'oracolo. A invece simula l'esperimento CCA e quindi dispone sia dell'oracolo di cifratura che decifratura.

Avversario A_E:
A_E prende in input 1^n e ha accesso a $\text{Enc}_{k_E}(\cdot)$.

1. Sceglie uniformemente $K_M \in \{0,1\}^n$
2. Esegue A su input 1^n. Quando A fa una query di cifratura all'oracolo per il messaggio m, risponde come segue:
   i. Invia una query m a $\text{Enc}_{k_E}(\cdot)$ e riceve c come risposta
   ii. Calcola $t \leftarrow Mac_{k_M}(c)$ e ritorna <c,t> ad A

   Quando A invia una query di decifratura all'oracolo per il cifrato <c,t> risponde come segue:
   - Se <c,t> è una risposta per una precedente query verso l'oracolo di cifratura per il messaggio m, ritorna m. Altrimenti ritorna $\perp$.
3. Quando A da in output i messaggi (m_0,m_1), Adv da in output gli stessi messaggi e riceve dal challenger il cifrato c come risposta. Calcola $t \leftarrow Mac_{k_M}(c)$, e ritorna <c,t> come cifrato del challenger ad A. Continua a rispondere alle query dell'oracolo di A come sopra.
4. Da in output lo stesso bit b' dato da A quando vince.

A_E esegue in tempo polinomiale, non ha bisogno di un oracolo di decifratura: assume che ogni decifratura che non corrisponde ad una query di cifratura precedente sia invalida.

A quando viene eseguito come subroutine di A_E si comporta nello stesso modi di come se fosse eseguito da $Priv^{cca}_{A,\Pi'}(n)$ fino a quando ValidQuery non si verifica.

Pertanto:

$$Pr[PrivK^{cpa}_{A_E,\Pi_E} = 1 \wedge \overline{ValidQuery}] = Pr[PrivK^{cca}_{A_E,\Pi_E} = 1 \wedge \overline{ValidQuery}]$$

Discende che:

$$Pr[PrivK^{cpa}_{A_E,\Pi_E} = 1] \geq$$
$$Pr[PrivK^{cpa}_{A_E,\Pi_E} = 1 \wedge \overline{ValidQuery}]$$
$$Pr[PrivK^{cca}_{A_E,\Pi_E} = 1 \wedge \overline{ValidQuery}]$$

Poiché per ipotesi Π_E è CPA-sicuro, esiste negl(n) tale che:

$$Pr[PrivK^{cpa}_{A_E,\Pi_E} = 1 \wedge \overline{ValidQuery}] \leq \tfrac{1}{2} + negl(n) \Rightarrow$$
$$Pr[PrivK^{cca}_{A_E,\Pi_E} = 1 \wedge \overline{ValidQuery}] \leq \tfrac{1}{2} + negl(n)$$

Quindi possiamo dire che:

$$Pr[ValidQuery] + Pr[Priv^{cca}_{A,\Pi'(n)=1} \wedge \overline{ValidQuery}]$$
$$\leq negl(n) + \tfrac{1}{2} + negl(n) \leq negl(n)$$

7.6.5 Necessità di chiavi indipendenti

"Istanze differenti di primitive crittografiche dovrebbero sempre usare chiavi indipendenti"
Consideriamo un esempio in cui applichiamo l'approccio "cifra e poi autentica" in cui la stessa chiave viene usata:

Sia F una PRP forte $\Rightarrow F^{-1}$ è una PRP forte

Consideriamo lo schema definito come:

- $F_k : \{0,1\}^n \to \{0,1\}^n$
- $Enc_k(m) = F_k(m||r)$, dove $m \leftarrow \{0,1\}^{n/2}$, $r \leftarrow \{0,1\}^{n/2}$
- $Mac_k(c) = F_k^{-1}(c)$

Si può mostrare che lo schema di cifratura é CPA-sicuro, il Mac é fortemente sicuro ma la combinazione usando la stessa chiave k non lo é!
Infatti:
$[Enc_k(m) = F_k(m||r), Mac_k(Enc_k(m)) = F_k^{-1}(F_k(m||r))] \quad [F_k(m||r), m||r]$
Ma nella costruzione 4.18 si richiede che le chiavi siano scelte uniformemente a caso.

Nota: Lo schema di cifratura sopra riportato è CCA-sicuro ma falsificabile, può essere utile nelle applicazioni dove l'integrità non è una preoccupazione. Non è l'unico schema con questa proprietà

7.6.6 Sessioni di comunicazione sicure

"Un periodo di tempo durante il quale le parti che comunicano mantengono uno stato e desiderano comunicare in modo sicuro"
Uno schema di cifratura autenticata da solo non é sufficiente. Abbiamo diversi tipi di attacchi a cui è vulnerabile:
- Attacco di riordino: Alice trasmette c_1, c_2, Adv intercetta e trasmette c_1, c_2
- Attacco di replay: Adv invia piú volte c
- Attacco di riflessione: Adv invia ad Alice un cifrato c che in precedenza Alice ha inviato a Bob

Si possono risolvere usando un **bit di direzionalitá** nelle comunicazioni, il bit $b_{X,Y}$ indica il verso, da X verso Y e **Contatori** dove il valore $ctr_{X,Y}$ conta i messaggi che X ha inviato a Y.
Quindi:
- $A \to B \quad c \leftarrow Enc_k(b_{A,B}||ctr_{A,B}||m)$
- $B \to A \quad c \leftarrow Enc_k(b_{B,A}||ctr_{B,A}||m)$
- $\text{Stato}_A = \text{Stato}_B = (\text{ctr}_{A,B}, \text{ctr}_{B,A})$

7.6.7 Schemi che si usano nella realtà

- **GCM (Galois/counter mode)**: Si basa sul paradigma cifra e poi autentica, usa CTR mode per la cifratura e GCM per l'autenticazione. Le chiavi non sono indipendenti e lo stesso IV é usato per cifrare e autenticare, nonostante ció la composizione é sicura. Tipicamente la PRF é istanziata con AES.
- **CCM (Counter with CBC-MAC)**:Si basa sul paradigma autentica e poi cifra, usa CTR mode per la cifratura e CBC-MAC per l'autenticazione. Usa la stessa chiave per cifrare e autenticare, nonostante ció la composizione é sicura. Facile da implementare (solo block cipher) ma lenta.
- **ChaCha20-Poly1305**: Si basa sul paradigma cifra e poi autentica. Usa lo stream cipher ChaCha20 in modo asincrono per cifrare. Usa Poly1305 con PRF istanziata da ChaCha20 per autenticare. Le implementazioni software molto veloci.

Capitolo 8

Funzioni Hash

Una funzione hash offre un modo per mappare una stringa di input lunga in una stringa di output piú corta, chiamata digest (impronta digitale). Un requisito fondamentale di queste funzioni è quello di evitare collisioni. Cioè presi x e x' non deve essere possibile che h(x) = h(x'). Nel caso che l'inseme universali degli elementi U ha cardinalità maggiore dell'insieme dei possibili valori hash allora sappiamo certamente che le collisioni esistono.
Formalmente:
H è resistente a collisioni (collision-resistant) se è impraticabile per un qualsiasi Adv PPT trovare una collisione in H.
Definiamo funzioni hash con chiavi una funzione hash con la seguente proprietà:
H è una funzione a due input tale che

$$\text{s,x} \rightarrow \text{H(s,x)} \stackrel{\text{def}}{=} H^s(x)$$

Deve essere difficile trovare una collisione in $H^s(\cdot)$ per un s generato a caso da Gen(1^n). Essendo che alcuni valori di s possono essere generati da Gen abbiamo che non tutti sono validi.
Inoltre s generalmente non è segreto e per questo viene usata la notazione h^s.

8.1 Definizioni

8.1.1 Definizione 5.1

Una funzione hash (con output di lunghezza l) è una coppia di algoritmi PPT (Gen, H) tali che:

- Gen è un algoritmo PPT che prende in input 1^n e dà in output s
- H prende in input s ed una stringa $x \in \{0,1\}^*$ e dà in output $H^s(x) \in \{0,1\}^{l(n)}$

Se H^s è definita solo per $x \in \{0,1\}^{l'(n)}$, con l'(n) > l(n), allora (Gen,H) è una funzione hash a lunghezza fissa per input di lunghezza l' allora H è detta funzione di compressione.

Sicurezza

Siano Π = (Gen, H), Adv A e n parametro di sicurezza

Definiamo l'esperimento trovare una collisione hash **Hash-coll**$_{A,\Pi}(n)$

1. Il Challenger esegue la funzione Gen(1^n) per generare la chiave;
2. L'avversario A riceve il parametro s e stampa in output due valori del dominio, x e x'.
3. Il Challenger controlla $x \neq x'$ e $H^s(x) = H^s(x')$. In caso affermativo stampa 1 (l'avversario A ha trovato una collisione), altrimenti 0.

8.1.2 Definizione 5.2

Una funzione Π = (Gen, H) è resistente a collisioni se, per ogni avversario PPT A, esiste una funzione trascurabile negl tale che:

$$\Pr[\text{ Hash-coll}_{A,\Pi}(n) = 1] \leq negl(n)$$

Nella pratica vengono usate funzioni hash senza chiavi e con lunghezza dell'output fissata: $H : \{0,1\}^* \to \{0,1\}^l$

8.1.3 Nozioni di sicurezza più deboli

- Resistenza alla seconda pre-immagine: Data s ed un y scelto uniformemente a caso $\{0,1\}^l$ è impraticabile per ogni Adv PPT trovare $x' \neq x$ tale che $H^s(x') = H^s(x)$. Quindi deve essere impossibile per un avversario trovare un x' che produca lo stesso valore hash di un x fissato.
- Resistenza alla pre-immagine (non invertibilità): Data s ed un x scelto uniformemente a caso, se fosse possibile trovare un x' tale che $H^s(x') = H^s(x) \Rightarrow$ la coppia (x,x') sarebbe una collisione (Scelto un y nel codominio della funzione deve essere impossibile per ogni Adv PPT trovare un x che produca il valore hash y).

É facile vedere che:

- Resistenza alla seconda pre-immagine ⇒ Resistenza alla pre-immagine: Infatti se, data s ed un y uniforme fosse possibile trovare un x tale che $H^s(x) = y$, Adv sceglierebbe uniformemente a caso e calcolerebbe y' = $H^s(x')$ e calcolerebbe una pre-immagine di y', chiamiamo questta pre-immagine x e avremo con alta probabilità risulterebbe x ≠ x', e quindi x sarebbe una seconda pre-immagine di y'.

8.2 Funzioni Hash

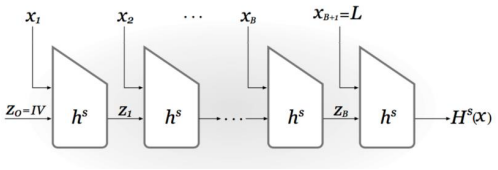
La costruzione delle funzioni hash sulla proprietà che siano resistenti a collisioni è divisa in due fasi:

- Progettazione di una funzione di compressione h^s;
- Estensione del dominio per input di lunghezza arbitraria.

A partire da una funzione hash che ha un dominio finito, si può costruire una funzione hash per un dominio infinito per una stringa di lunghezza arbitraria, ad ognuna delle quali viene associata una stringa di n bit usando la trasformata di Merke-Damgard.

8.2.1 Trasformazione Merke-Damgard

Il suo funzionamento è il seguente: una stringa di lunghezza arbitraria viene divisa in blocchi, ognuno di n bit per poi aggiungere alla fine della stringa un extra-blocco in cui viene codificato, sempre con n bit, la lunghezza della stringa. Ad ogni passo della trasformata, viene utilizzata la funzione di compressione h^s che prende in input il blocco di n bit e il risultato z_l prodotto dalla funzione di compressione precedente, ad eccezione del primo blocco che prende come secondo input un IV (il vettore corrisponde ad una stringa di n 0). Alla fine della trasformata, avremo come risultato il valore hash per la stringa x .



Sia (Gen, h) una funzione hash con lunghezza fissa per un input di lunghezza 2n e un output di lunghezza n. Costruiamo una funzione hash (Gen, H) come segue:

- Gen: Il Challenger esegue la funzione Gen(1^n) per generare la chiave;
- H: prende in input la chiave s e una stringa x $\in \{0,1\}^*$ di lunghezza $L < 2^n$, come segue:
  1. Settiamo B := $\lceil \frac{L}{n} \rceil$ (numero di blocchi in x). Il pad x contiene un multiplo di n 0. Analizza il padding risultante in una sequenza di n-bit blocchi $x_1, ..., x_B$. Imposta x_{B+1} := L, dove L è la codifica di n-bit stringhe.
  2. Imposta $z_0 := 0^n$. (IV)
  3. For i = 1, ... B+1, calcola $z_i := h^s(z_{i-1}||x_i)$
  4. Da in output z_{B+1}

8.2.2 Teorema 5.4

Se (Gen, h) é resistente rispetto a collisioni, allora anche (Gen,H) lo é.

Dimostrazione: Dimostreremo che per qualsiasi s, una collisione per la funzione di trasformata H^s dà anche una collisione per la funzione di compressione h^s.
Siano x e x' due stringhe differenti di lunghezza L e L':

$$x = x_1...x_B x_{B+1} \qquad x' = x'_1...x'_B x'_{B+1}$$

tali che $H^s(x) = H^s(x')$.
Abbiamo due casi da considerare:

Caso 1: La lunghezza delle due stringhe è diversa L $\neq$ L'. Gli ultimi passi del calcolo di $H^s(x)$ e di $H^s(x')$ sono:

$$z_{B+1} = h^s(z_b||x_{B+1}) \quad \text{e } z'_{B+1} = h^s(<'_b \, ||x'_{B+1})$$

Essendo che la funzione hash applicata per le due stringhe, x e x', generano una collisione, significa che z_{B+1} e z'_{B+1} sono gli stessi.

$$H^s(x) = H^s(x') \rightarrow z_{B+1} = z'_{B+1}$$

Quindi significa che:

$$w = z_{B+1}||x_{B+1} \text{ e } w' = z'_{B+1}||x'_{B+1}$$

sono una collisione per h^s, essendo $w \neq w$ dato che $x_{B+1} \neq x'_{B+1}$.

Caso 2: La lunghezza delle due stringhe è uguale L = L'. Quindi il numero di blocchi è uguale per entrambi B = B' e la codifica della lunghezza nell'ultimo blocco è uguali per entrambi $x_{B+1} = x'_{B+1}$
Per dimostrare tale caso introduciamo delle notazioni con lo scopo di renderlo facile. Siano:

- $z_1, ...z_{B+1}$ i valori prodotti dal calcolo di $H^s(x)$
- $I_1, ...I_{B+1}$ gli input per h^s, cioè $I_i = z_{i-1}||x_i$ per i=1, ..., B+1
- $z'_1, ...z'_{B+1}$ i valori prodotti dal calcolo di $H^s(x')$
- $I'_1, ...I'_{B+1}$ gli input per h^s, cioè $I'_i = z'_{i-1}||x'_i$ per i=1, ..., B+1

Poniamo inoltre $I_{B+2} = z_{B+1}$ e $I'_{B+2} = z'_{B+1}$.
A questo punto indichiamo con N il più grande indice per cui risulta $I_N \neq I'_N$
Dato che $x \neq x'$, deve per forza esistere un indice i tale che $x \not= x'_i$ e quindi come conseguenza a ciò esiste N.
D'altra parte dato che

$$I_{B+2} = z_{B+1} = H^s(x) = H^s(x') = z'_{B+1} = I'_{B+2}$$

dobbiamo anche avere $N \leq B + 1$ dato che c'è l'uguaglianza per I_{B+2} e I'_{B+2}.
Per definizione sappiamo però che N è l'indice più grande per cui abbiamo $I_N = I'N$. Quindi:

$$I_{N+1} = I'_{N+1} \rightarrow z_N = z'_N \rightarrow h^s(I_N) = z_N = z'_N = h^s(I'_N)$$

Sapendo che $I_{N+1} = z_N||x_{N+1}$ e $I'_{N+1} =' z_N||x'_{N+1}$. Di conseguenza le stringhe I_N e I'_N sono una collisione per la funzione di compressione h^s

8.3 Paradigma Hash-and-Mac

Questo paradigma fa uso delle funzioni hash per autenticare un messaggio:

$$m \in \{0,1\}^*, \quad \text{y=}H^s(m), \text{ Mac}_k\text{(y)}$$

H^s è collision-resistant. Il Mac viene calcolato su H^s invece che su m.

COSTRUZIONE 5.5
Sia Π = (Mac, Vrfy) un MAC per messaggi di lunghezza l(n), e sia Π_H = (Gen_H, H) una funzione hash con output di lunghezza l(n). Costruiamo un MAC Π′ = (Gen', Mac', Vrfy') per messaggi a lunghezza arbitrari come segue:

- Gen': prende in input 1^n, sceglie uniformemente $k \in \{0,1\}^n$ ed esegue $\text{Gen}_H(1^n)$ per ottenere s; la chiave k' := <k, s>.
- Mac': prende in input la chiave <k,s> e il messaggio $m \in \{0,1\}^*$, da in output $t \leftarrow Mac_k(H^s(m))$
- Vrfy': prende in input la chiave <k,s> e un messaggio $m \in \{0,1\}^*$ e il tag del MAC t, da in output 1 se e solo se $Vrfy_k(H^s(m), t) \stackrel{?}{=} 1$

La costruzione é sicura se Π é un Mac sicuro per messaggi di lunghezza fissa e Π_H é collision-resistant.
Quindi Π_H collision-resistant $\Rightarrow$ autenticare $H^s(m)$ é "essenzialmente uguale" ad autenticare m.
Più precisamente supponiamo che un mittente usi la costruzione per autenticare un insieme di messaggi Q, ed A riesca a produrre una falsificazione per $m^* \in Q$. Abbiamo due possibili casi:

1. $\exists$ un messaggio $m \in Q : H^s(m) = H^s(m^*) \Rightarrow$ A ha trovato una collisione per h^s;
2. $\forall m \in Q$, $H^s(m) \neq H^s(m^*) \Rightarrow$ A ha trovato un tag valido rispetto a Π per $H^s(m^*)$.

8.3.1 Teorema 5.6

Se Π un Mac sicuro per messaggi di lunghezza l e Π_H é resistente a collisioni, allora la Costruzione 5.5 é un Mac sicuro per messaggi di lunghezza arbitraria.

Dimostrazione: Sia Π' la costruzione 5.5 e A' un Adv che attacca Π'. In una esecuzione di Mac-forge$_{A',\Pi'}(n)$ sia k' = <k,s> e sia Q l'insieme dei messaggi di cui A' chiede i tag.
Sia $m^* \notin Q$. Indichiamo con *Coll* l'evento in Mac-forge$_{A',\Pi'}(n)$: "c'è un messaggio $m \in Q$ per cui $H^s(m) = H^s(m?*)$.
Risulta Pr[Mac-forge$_{A',\Pi'}(n) = 1$]

$$\begin{aligned} &Pr[\text{Mac-forge}_{A',\Pi'}(n) = 1 \wedge Coll] + Pr[\text{Mac-forge}_{A',\Pi'}(n) = 1 \wedge \overline{Coll}] \\ &\leq Pr[Coll] + Pr[\text{Mac-forge}_{A',\Pi'}(n) = 1 \wedge \overline{Coll}] \end{aligned}$$

Mostreremo che entrambi i termini sono trascurabili.

- Il primo termine è trascurabile per via di Π_H.
L'algoritmo C che segue usa A' che attacca Π' per trovare collisioni per Π_H.

Algoritmo C:
L'algoritmo prende in input s (n è implicito)

- Sceglie uniformemente $k \in \{0,1\}^n$
- Esegue A'(1^n). Quando A' richiede il tag dell'i-esimo messaggio $m_i \in \{0,1\}^*$, calcola $t_i \leftarrow Mac_k(H^s(m_i))$ e da t_i a A'.
- Quando A' da in output (m^*, t) se esiste un i per ogni $H^s(m^*) = H^s(m_i)$, da in output (m^*, m_i)

C è PPT, quando s viene generato da $Gen_H(1^n)$ la vista di Aì quando eseguito come subroutine di C è distribuita identicamente alla vista che A' ha quando esegue in Mac-forge$_{A',\Pi'}(n)$. Poichè C dà in output una collisione esattamente quando essa occorre risulta:

$$Pr[\text{Hash-Coll}_{C,\Pi_H}(n) = 1] = Pr[Coll].$$

Dall'assunzione che Π_H è collision-resistant, segue che $\exists$ negl tale che:

$$Pr[\text{Hash-Coll}_{C,\Pi_H}(n) = 1] \leq negl(n) \Rightarrow Pr[Coll] \leq negl(n)$$

- Il secondo termine è trascurabile per via della sicurezza dello schema Mac Π. L'algoritmo A in Π in Mac-forge$_{A,\Pi}(n)$.

Avversario A:
L'avversario può accedere all'oracolo MAC Mac$_k(\cdot)$.

- Calcola Gen$_H(1^n)$ per ottenere s.
- Esegue $A'(1^n)$. Quando A' richiede il tag per l'i-esimo messaggio $m_i \in \{0,1\}^*$; Poi:
  1. calcola $\hat{m_i} := H^s(m_i)$;
  2. ottiene il tag t_i per m_i dall'oracolo MAC;
  3. Da t_i a A'
- Quando A' da in output (m^*, t), poi da in output $(H^s(m^*), t$.

A è PPT. La vista di A' quando eseguito come subroutine di A è distribuita identicamente alla vista che A' ha quando esegue in Mac-forge$_{A',\Pi'}$.

Quando entrambi gli eventi "Mac-forge$_{A',\Pi'}(n) = 1$" e "$\overline{Coll}$ si verificano," A dà in output una falsificazione (t è un tag valido per $H^s(m^*)$ rispetto a Π). Infatti poiché Coll non si verifica $h^s(m^*)$ non è stata una query di A all'oracolo Mac$_k(\cdot)$. Quindi:

$$Pr[\text{Mac-forge}_{A,\Pi}(n) = 1] = Pr[\text{Mac-forge}_{A,\Pi}(n) = 1 \wedge \overline{Coll}]$$

Dall'assunzione che Π è un Mac sicuro segue che ∃ negl tale che

$$Pr[\text{Mac-forge}_{A,\Pi}(n) = 1] \leq negl(n) \Rightarrow Pr[\text{Mac-forge}_{A,\Pi}(n) = 1 \wedge \overline{Coll}] \leq negl(n)$$

8.4 HMAC

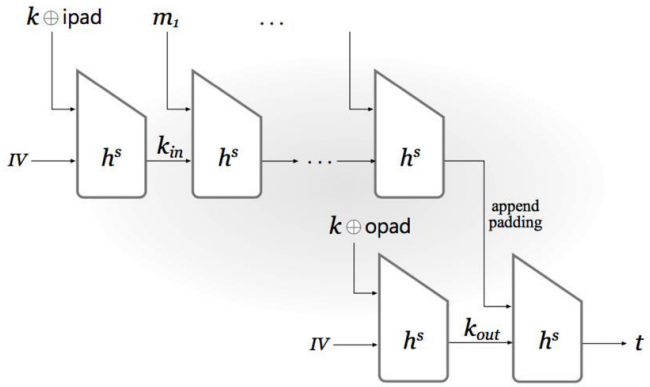
È possibile costruire uno schema Mac sicuro per messaggi di lunghezza arbitraria, basandosi direttamente su una funzione hash.
NOTA: costruire uno schema in cui si prende una chiave per poi fare la concatenazione del messaggio e da qui applicare la funzione hash non è molto sicuro, cioè Mac$_k(m) = H(k||m)$.

L'idea che funziona è quella di utilizzare due livelli di hash:

1. Un primo livello per creare il digest;
2. Un secondo livello per autenticare.

La funzione avrà la seguente struttura $H^s(k_1, H^s(k_2, m))$.

Il funzionamento è il seguente: il messaggio m viene diviso in blocchi. Il primo livello consiste nell'effettuare la trasformata di Merkle-Damgard; quindi, ogni blocco viene concatenato con la chiave prodotta dalla funzione hash precedente, ad eccezione del primo che invece viene utilizzato un IV. La differenza sta nel fatto che nel primo blocco la chiave k viene sottomessa all'operazione XOR con ipad (denota il livello interno). Il risultato di questo livello è chiamato digest e viene utilizzato, così, nel secondo livello mediante la concatenazione della chiave k_{out}, che viene prodotta mediante la concatenazione del vettore IV e lo XOR tra la chiave k e *opad* (denota il livello esterno), per poi applicare la funzione hash su questo avendo così come risultato il tag per il messaggio corrispondente.



COSTRUZIONE 5.7
Sia (Gen_H, H) una funzione hash costruita applicando la trasformata di Merkle-damgard su una funzione di compressione (Gen_H, h) che prende input di lunghezza n + n'. Definiamo il MAC come segue:

- Gen: prende in input 1^n , esegue $\text{Gen}_H(1^n)$ per ottenere la chiave s, sceglie uniformemente $k \in \{0,1\}^{n'}$. Da in output la chiave <s,k>;
- Mac: prende in input la chiave <s,k> e il messaggio $m \in \{0,1\}^*$, da in output
  t := $H^s(\ (k \oplus opad)||H^s(\ (k \oplus ipad)||m\ )\ )$
- Vrfy: prende in input la chiave <s,k>, un messaggio $m \in \{0,1\}^*$ e un tag t, da in output 1 se e solo se $t \stackrel{?}{=} H^s(\ (k \oplus opad)||H^s(\ (k \oplus ipad)||m\ )\ )$

Osservazione:

- La lunghezza del messaggio nella trasformazione viene codificata con un blocco extra $x_B + 1$. In realtá, in pratica viene codificata in una porzione di blocco, usando l bit.
- Il messaggio x viene completato con zeri fino ad ottenere una lunghezza multiplo di n' a meno di l bit. Poi viene aggiunta L = |x|, codificata con l bit (assumiamo che n + l < n' nella costruzione).

8.4.1 Sicurezza dell'HMAC

HMAC è sicuro perché essa può essere vista come una specifica istanza del paradigma Hash-and-Mac, opera come segue:

- Associa una stringa corta ad un messaggio di lunghezza arbitraria:
  y := $H^s((k \oplus ipad)||m)$
- Dopodiché ad y si aggiunge il blocco che codifica la lunghezza del messaggio, lo chiamiamo $\tilde{y}$, e successivamente viene calcolato il tag:
  t := $H^s((k \oplus opad)||\tilde{y})$

Sia un Mac sicuro a lunghezza fissa, allora HMAC è un'istanza di Hash-and-Mac perché col primo passo sto producendo l'hash e col secondo sto calcolando il Mac.

Primo passo: nel primo passo uno produciamo il tag t come segue:
$\mathrm{t} := H^s((k \oplus opad)||\tilde{y})$
che consiste proprio nel calcolare il valore della chiave $k_{out} \stackrel{\text{def}}{=} h^s(IV||k \oplus opad)$ e utilizzarla poi per calcolare $\mathrm{t} := h^s(k_{out}||\tilde{y}$
Se vediamo $\tilde{y}$ come y con l'aggiunta del pad e trattiamo k_{out} come una stringa uniforme possiamo assumere che $\tilde{Mac}_k(y) \stackrel{\text{def}}{=} h^s(k||y)$ è un Mac sicuro a lunghezza fissa, e quindi HMAC è un'istanza di Hash-and-Mac: $\mathrm{HMAC}_{s,k} = \tilde{Mac}_{k_{out}}(\tilde{H}^s(m))$.

Le costanti ipad e opad servono per derivare efficientemente due chiavi da una sola.
L'utilizzo della chiave k è giustificato per il livello esterno in quanto si utilizza un Mac che appunto richiede una chiave segreta per autenticare il messaggio, ma non nella computazione interna. Viene introdotta perché rende possibile provare la sicurezza della costruzione su una assunzione "più debole", la weak collision-resistance (resistenza a collisioni debole).

Secondo passo Nell'esperimento $\text{Hash-coll}_{A,\Pi}$ l'avversario una volta che ha ricevuto s, che specifica la funzione di compressione, si calcola da solo tutti gli hash che vuole e prova a trovare una possibile collisione. In tale contesto l'avversario ha la conoscenza totale della funzione che si sta utilizzando nell'esperimento.
Invece, per modellare la resistenza debole a collisioni, l'avversario interagirebbe con un oracolo all'interno del quale c'è una chiave segreta che restituisce l'hash su messaggi m come richiesta e quindi stiamo dando meno potere all'avversario in quanto non dispone totalmente della funzione hash ma soltanto accesso all'oracolo . Questo è importante perché se H è collision-resistance → allora H è weakly collision-resistance.

Nell'HMAC dovrebbero sempre essere usate due chiavi indipendenti, ma nelle implementazioni se ne usa soltanto uno che viene poi messa in XOR sia con opad e ipad.
Definiamo:

$$G^s(k) = h^s(IV||(k \oplus opad))||h^s(IV||(k \oplus ipad)) = k_{out}||k_{in}$$

Se assumiamo che G^s sia un PRG per qualsiasi valore di s, allora k_{out} e k_{in} possono essere considerate chiavi indipendenti ed uniformemente distribuite.

Teorema 5.8 Sia G^s un PRG per un qualsiasi s, sia $\tilde{Mac}_{k_{out}}(\cdot)$ un Mac sicuro per messaggi di lunghezza fissa n, e sia (Gen_H, H) una funzione hash wakly collision-resistant. Allora HMAC è un MAC sicuro per messaggi di lunghezza arbitraria.

8.5 Attacchi generici alle funzioni hash

Presa una funzione hash definita come segue: $\mathrm{H} : \{0,1\}^* \to \{0,1\}^l$, abbiamo sempre due possibili attacchi:

1. Valuta H su $2^l + 1$ input distinti ⇒ due output devono essere uguali;
2. Scelti q input distinti $x_1, .., x_q$ calcoliamo $y_i = H(x_i)$ per i = 1, ..., q e si controlla che due y_i siano uguali. Con $\mathrm{q} > 2^l$ la probabilità è 1, con q più piccolo è più complicato.

8.5.1 Primo attacco

Sia H una funzione casuale e con output uniformemente distribuito in $\{0,1\}^l$. Il nostro problema diventa:

Qual'è la probabilità che $\exists\, i, jtalechey_i = y_j$, se scegliamo $y_1, ..., y_q \in \{0,1\}^l$ indipendentemente ed uniformemente a caso? É uguale al problema del compleanno.

Prendendo q = $\Theta(\mathrm{N}^{1/2})$ la probabilità è circa $\frac{1}{2}$. Assumendo che H per resistere ad attacchi nel tempo richiede al più T tempo, abbiamo che la lunghezza dell'output deve essere almeno 2 log t bit, infatti:

$$2^{\frac{2logT}{2}} = 2^{logT} = T$$

Per trovare collisioni significative possiamo procedere nel seguente modo:
Vogliamo trovare due messaggi x e x' tale che H(x) = H(x'). Se per autenticare i messaggi è stato usato Hash-and-Mac, possiamo usare il tag del primo messaggio per autenticare il secondo, ma questo richiede che i messaggi $x_1, ..., x_q$ siano soltanto distinti (non casuali).
Pertanto un Adv può:

- produrre q = $\Theta(2^{l/2})$ messaggi del primo tipo (x)
- produrre q = $\Theta(2^{l/2})$ messaggi del secondo tipo (x')
- cercare collisioni tra i messaggi del primo gruppo e quelli del secondo

Come detto è un attacco simile a quello del compleanno e si ha collisione con probabilità circa 1/2. Il problema di questo attacco è l'uso eccessivo della memoria, abbiamo infatti bisogno di $\Theta(2^{l/2})$ valori da memorizzare.

8.5.2 Secondo attacco

L'idea si basa sull'algoritmo di Floyd per trovare cicli. Si consideri la sequenza x_i definita da $x_i = H(x_{i-1}$ con x_0 come valore iniziale e scelto a caso.
Se tale sequenza è ciclica(cioè ad un certo punto nella sequenza si ripetono i valori), esistono interi j > 0 e k > 0 tali che $x_j = x_{j+k}$ e quindi per tutti gli interi $i \geq j$ ed $n \geq 1$ risulta: $x_i = x_{i+nk}$. Quindi esiste un intero i tale che $x_i = x_{2i}$ (questo è un ciclo).

Per individuare i cicli in $\{x_i\}$ è sufficiente controllare per ogni intero positivo i se $x_i = x_{2i}$. Si inizia scegliendo un x_0 a caso e si calcola:

$$\begin{array}{rr} x_1 = H(x_0) & x_2 = H(H(x_0)) \\ x_2 = H(x_1) & x_4 = H(H(x_4)) \\ ... & ... \\ x_i = H(x_{i-1}) & x_{2i} = H(H(x_{2(i-1)})) \end{array}$$

Quando $x_i = x_{2i}$ sappiamo che $\exists$ un minimo j tra 0 ed i tale che $x_j = x_{j+1}$ e quindi (x_{j-1},x_{j+i-1}) è una collisione per H, questo perché x_{j-1} e x_{j+i-1} sono due elementi distinti che nel momento in cui gli si calcola il valore Hash, quest'ultimo è uguale.
In generale:

> Poniamo all'inizio y = x_0; z = x_i.
> Calcoliamo H(y) e H(z).
> Se H(y) = H(z) allora (y,z) è una collisione; altrimenti aggiorniamo y = H(y) e z = H(z) e ripetiamo il calcolo.

Se modelliamo H come una funzione casale, risulta per ogni y, H(x) = y con probabilità $1/2^l$.
Pertanto ci aspettiamo che una ripetizione occorra con probabilitá maggiore di 1=2 durante la generazione dei primi q = $\Theta(2^{\frac{l}{2}})$ termini della sequenza. Se esiste una ripetizione nei primi q termini, l'algoritmo impiega al piú q passi per trovarla.

Nota: non ci sono attacchi generici per le proprietá second-preimage resistance e preimage resistance.
Come troviamo collisioni per messaggi sugnificativi? Con questo attacco Adv non ha più controllo sugli x_i. Procediamo come segue:

- Adv scrive ogni messaggio in modo tale che ci siano l-1 parole intercambiabili

- definisce poi una funzione uno a uno *g:* $\{0,1\}^l \to \{0,1\}^*$ dove l'l-esimo bit seleziona il tipo (0 o 1= del messaggio e l'i-esimo bit la parola intercambiabile. Es:
  - g(0000) = Bob é un buon e onesto lavoratore
  - g(0010) = Bob é un buon e onesto dipendente
  - g(1101) = Bob é un fastidioso e irritante lavoratore

Definendo f : $\{0,1\}^l \to \{0,1\}^l$ come f(x) = H(g(x)).
Alice puó trovare collisioni in f usando l'attacco del compleanno con poco spazio come precedentemente descritto.

8.6 Random Oracle Model (ROM)

Per alcune funzioni hash non si conoscono riduzioni di sicurezza basate su una qualche assunzione. Usare schemi che giustificano soltanto l'efficienza e resistono ad attacchi noti non sono accettati. Un approccio che ha avuto successo è quello basato sull'utilizzo di un oracolo casuale : il ROM.

8.6.1 Modello

Il modello si basa sull'avere una funzione H totalmente casuale:

$$\mathrm{H} : \{0,1\}^* \to \{0,1\}^l$$

questa funzione è pubblica e può essere valutata soltanto attraverso delle query ad un oracolo.
In questo schema chiunque può interagire con l'oracolo. **Nota** un ROM non può esistere nella realtà, una funzione casuale richiede spazio esponenziale.

Per provare la correttezza di uno schema con ROM si effettuano i seguenti passi:

1. Si progetta uno schema e si prova la sua sicurezza all'interno del modello dove è presente il ROM.

2. Per implementare lo schema nel mondo reale si istanza il random oracle con una funzione hash crittografica $\hat{H}$

Quando lo schema impone ad una delle parti di inviare una query all'oracolo, nell'implementazione la parte calcola l'output della funzione hash $\hat{H}$.
Se $\hat{H}$ è sufficientemente buona nell'emulare il random oracle, la prova di sicurezza data nel modello dovrebbe estendersi allo schema reale.

8.6.2 Analzziamo più in dettaglio la strategia

Le parti oneste e Adv possono inviare query x all'oracolo ricevendo H(x), i meccanismi interni dell'oracolo non sono noti (scatola nera), le query sono private nessun altro conosce la query x fatta da una parte non sa neanche che ha inviato una query, Le query vengono risposte consistentemente, stessa query stessa risposta.
La scelta di una funzione casuale puó essere effettuata in due modi:

1. one-shot: scegliere in un solo colpo nell'insieme di tutte le funzioni su dominio e codominio specificati
2. on-the-fly: scegliere uniformemente al volo le risposte alle query e memorizzarle in tabella

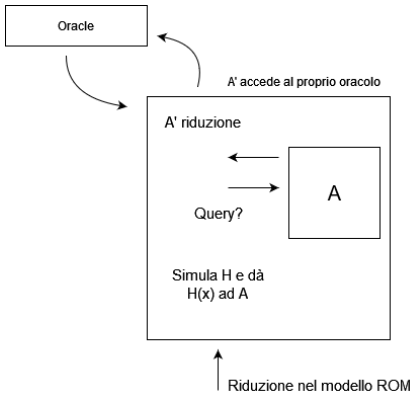
Nel ROM, per brevitá, Π poggia su un oracolo: Π é sicura se, ∀ A ppt, la probabilitá di un certo evento é sotto una determinata soglia, dove la probabilitá é calcolata sulle scelte casuali delle parti oneste, dell'Adv e dell'oracolo H.

8.6.3 Prove nel ROM

Le dimostrazioni nel modello ROM possono sfruttare il fatto che H viene scelta uniformemente a caso e puó essere valutata soltanto tramite query. Tre proprietá risultano utili:

- **Uniformità**: se x non é stato ancora usato per una query ad H, allora il valore H(x) é uniformemente distribuito. Si noti la differenza tra H ed un PRG:
  - G(x) é pseudocasuale per un osservatore ppt, assumendo che x sia scelto uniformemente e sia totalmente sconosciuto all'osservatore.
  - H(x) é totalmente casuale fino a quando un osservatore non ha inviato la query x all'oracolo. Anche se x é noto o se x non é scelto uniformemente.

  Le altre due proprietá sono relative alle prove di sicurezza. La riduzione A' nel ROM deve simulare l'oracolo H con cui interagisce A, rispondendo opportunamente alle sue query
- **estraibilitá**:se A invia la query x ad H, la riduzione A' vede questa query ed acquisisce il valore di x
- **programmabilitá**:la riduzione A' puó fissare il valore H(x) ad un valore di propria scelta, posto che il valore sia uniformemente distribuito



Osservazione: quando H, in una implementazione, viene istanziata con una funzione concreta $\hat{H}$, non c'é nessuna controparte per le proprietá dell'estraibilitá e della programmabilitá.

8.6.4 Creare primitive crittografiche tramite l'uso del ROM

Creazione di un generatore pseudocasuale

Supponiamo di disporre:

$$\text{H} : \{0,1\}^{l_{in}(n)} \Rightarrow \{0,1\}^{l_{out}(n)}$$

con $l_{in}(n)$, $l_{out}(n) >$ n (parametro di sicurezza).
Un oracolo casuale dove $l_{out}(n) > l_{in}(n)$ può essere usato come un PRG.
Infatti $\forall$ A PPT, $\exists$ una funzione tascurabile negl tale che:

$$|Pr[A^{H(\cdot)}(H(x)) = 1] - |Pr[A^{H(\cdot)}(y) = 1]| \leq negl(n)$$

La prima probabilità è calcolata sulla scelta uniforme di H(·), di x $\in \{0,1\}^{l_{in}(n)}$, e le scelte casuali di A.
La seconda probabilità è calcolata sulla scelta uniforme di H(·), di y $\in \{0,1\}^{l_{out}(n)}$, e le scelte casuali di A.

Informalmente La probabilità che un Adv capisca che il valore è generato da un generatore pseudocasuale - la probabilità che l'Adv capisca che è stato scelto uniformamente a caso deve essere trascurabile.

Sia S = { punti usati da A come query per H}
|S| è PPT e la probabilità che x $\leftarrow \{0,1\}^{l_{in}(n)}$ appartenga ad S è $\frac{PPT(query\ possibili)}{2^{l_{in}(n)}(cardinalit\ dell'insieme)}$
Inoltre, condizionato all'evento x$\in$2 S, l'input di A é in entrambi i casi uniforme ed indipendente dalle risposte alle query di A.

Pertanto $|Pr[A^{H(\cdot)}(H(x)) = 1] - |Pr[A^{H(\cdot)}(y) = 1]| =$
$|(Pr[A^{H(\cdot)}(H(x)) = 1 \wedge x \in S] + Pr[A^{H(\cdot)}(H(x)) = 1 \wedge x \notin S]) - Pr[A^{H(\cdot)}(y) = 1]$
(essendo la seconda e la terza probabilitá uguali)
$P[x \in S]$ (corrisponde alla probabilità $A \cap B$)
$= poly(n)/2^{l_{in}(n)}$ (che è negl(n)).

$x \in S$:x appartiene all'insieme delle query fatte

Creazione di una funzione resistente a collisione

Se $l_{in}(n) > l_{in}(out)$ un oracolo casuale puó essere usato come una funzione resistente a collisioni. Per fare ciò si consideri l'esperimento che segue:

Rom-Coll

1. Viene scelta una funzione H
2. A dà in output x ed x' distinti ed ha successo se H(x) = H(x')

La probabilitá di successo di qualsiasi A é trascurabile. Infatti, assumiamo che: A dá in output soltanto valori x ed x' precedentemente inviati come query ad H A non ripete mai la stessa query.

Siano $x_1, ..., x_q$ le query (numero polinomiale)
Pr[A ha successo] $\leq Pr[H(x_i) = H(x_j)$ per qualche $i \neq j]$ implica che la probabilità di prendere q stringhe $y_i \in \{0,1\}^{l_{out}(n)}$, indipendentemente ed uniformemente a casto tali che $y_i = y_j,\ i \neq j$, allora A ha successo con probabilità O($q^2/2^{l_{out}(n)}$ (probabilità del paradosso del compleanno).

Creazione di una funzione pseudocasuale

Sia $l_{in} = 2n$ ed $l_{out}(n) = n$, e sia $F_k(x) \stackrel{\text{def}}{=} H(k||x)$, dove |k| = |x| = n.
Si consideri l'esperimento

Rom-PRF

1. Vengono scelti uniformemente a caso una funzione H, un k $\in \{0,1\}^n$ ed un b $\in \{0,1\}$
2. Se b = 0, A riceve accesso ad un oracolo per $F_k(\cdot) = H(k||\cdot)$;altrimenti, A riceve accesso ad una funzione casuale H (indipendente da H)
3. A dá in output un bit b' ed ha successo se b' = b

Al passo 2: l'avversario A puó anche accedere all'oracolo casuale H oltre che all'oracolo $F_k(\cdot)$ o $\overline{H}(\cdot)$, a seconda se b=0 o b=1.

É possibile dimostrare che: $\forall$ A PPT, $\exists$ una funzione trascurabile negl tale che la probabilità di successo di A nell'esperimento Rom-PRF è al più 1/2 +negl(n).

Si noti:
tutti i claim valgono anche con avversari con potere computazionale illimitata finchè possono effettuare un numero polinomiale di query.

Non ci sono prove che che lo schema sia sicuro anche nel modo reale. Nessuno $\hat{H}$ può comportarsi come oracolo casuale. Ad oggi non esistono attacchi che hanno avuto successo su schemi che sono stati provati sicuri nel ROM, quando H è istanziato correttamente.

8.7 Utilizzo delle funzioni hash

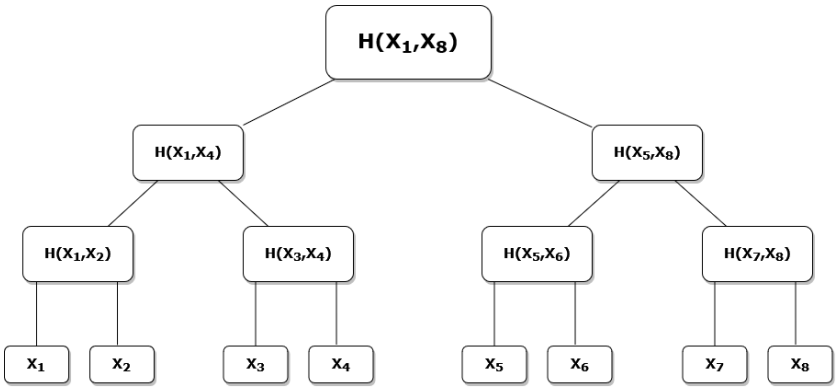
Le funzioni hash sono usate in molteplici applicazioni:

- **Virus fingerprinting:** si memorizzano in un database i digest calcolati sul codice dei virus. L'hash di un file viene confrontato con i digest;
- **Deduplication:** se in un servizio di tipo cloud un file é giá presente, un nuovo upload viene evitato. Per controllare la presenza del file il digest viene confrontato con i digest giá presenti nel database;
- **Sistemi P2P:** contengono tabelle con i digest dei file disponibili, per velocizzare le ricerche (o download legalissimi).

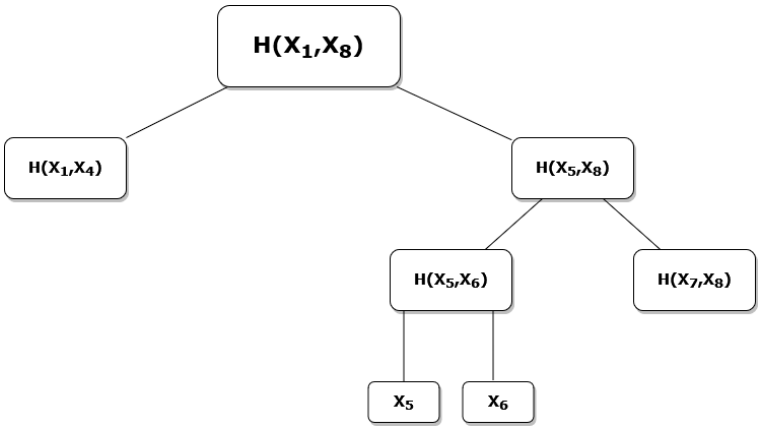
8.7.1 Merkle tree

Consideriamo lo scenario in cui un utente vuole memorizzare un file di grande dimensione sul cloud. Dopo un po' di tempo, l'utente ha la necessità di scaricarlo. Tuttavia, il file potrebbe essere manomesso in quell'arco di tempo quindi l'utente potrebbe non essere sicuro che il file è stato mantenuto integro. L'idea è quella di mantenere sulla macchina il digest del file, risparmiando memoria. Nel momento in cui si scarica un determinato file, viene calcolato anche il suo hash per verificare l'uguaglianza e quindi l'integrità con il file presente sul computer. Questo metodo però non è molto efficiente se si vogliono memorizzare più file, infatti se abbiamo $x_1, ...x_n$ messaggi, dovremmo memorizzare sul computer $H(x_1), ..., H(x_n)$ valori hash, e quindi occupare grandi spazi di memoria.

Una tecnica efficiente che permette di autenticare in modo efficiente più file è il Merkle-tree. L'idea è quella di memorizzare i digest ispirandosi alla struttura di un'albero. Abbiamo inizialmente n foglie quanti sono i file che verranno caricati sul cloud, per poi costruire i nodi mediante la concatenazione dei digest dei file. Come risultato finale avremo un valore hash come radice. In questa maniera basterebbe soltanto che l'utente memorizzi il valore hash presente all'interno della radice dell'albero $H_{1,8}$ per poi così uploadare i file $x_1, ..., x_8$ nel cloud.



Successivamente quando l'utente vuole recuperare un file e controllare la sua integrità riceve dal cloud il file x_i da lui desiderato e tutti i valori hash associati alla path da x_i alla radice. **Per esempio** se l'utente richiede il file x_5 oltre al file riceverà anche il file x_6 e gli hash $H(x_7, x_8)$ e $H(x_1, x_4)$. Dopodiché questo effettua i dovuti controlli per assicurare l'integrità del file scaricato.



Teorema 5.11

Sia (Gen_H, H) collision-resistant. Allora (Gen_H, MT_t) è collision-resistant per t fissato.
Intuizione per la prova: se il cloud potesse provare che $x_j \notin \{x_1, ..., x_t\}$ è parte dell'albero, allora il cloud sarebbe in grado di calcolare collisioni di H efficientemente.

Lo schema richiede:

- Spazio costante per la memorizzazione;
- Complessità di comunicazione 0(log t);
- Complessità di calcolo (verifiche hash) O(log t).

Un uso dei Merkle tree è l'autenticazione dei blocchi usati in Bitcoin.

8.7.2 Controllo delle password

Un altro possibile utilizzo è per la correttezza delle password. Infatti invece di memorizzare la password (pwd) possiamo direttamente memorizzare H(pwd), in caso che venisse rubata grazie alla "preimage resistance" di H è difficile recuperare pwd.
NOTA: Usare la password Paolo D'Arco, garantisce la sicurezza perfetta.
Esempio Immaginiamo che la pwd $\in$ D (dizionario della lingua inglese) dove |D| è piccola, una volta ottenuto il digest hpwd di pwd può essere applicato un attacco a dizionario. Quello che vogliamo ottenere è che l'attacco a dizionario sia il miglior attacco possibile da Adv. **PROBLEMA:** la proprietà di pre-image resistance non è sufficiente , perchè richiede che x sia scelto uniformemente in $\{0,1\}^n$ affinchè sia difficile da invertire, ma non dice nulla su spazi con distribuzioni arbitrarie (come D).

Se modelliamo H come un ROM, possiamo provare il risultato che desideriamo. Assumendo che pwd sia scelta in modo uniforme in D, tale attacco richiede $\frac{|D|}{2}$ valutazioni di H in media.
Se consideriamo l'evento: X_{val} = "valutazioni richieste".
Essendo H totalmente casuale:

$$Pr[X_{val} = i] = Pr[\text{ hpwd corrisponda alla i-esima pwd di D }] = \frac{1}{|D|}$$

Pertanto risulta:

$$\begin{aligned}\text{Exp}(\text{X}_{val}) &= \sum_{i=1}^{|D|} i \cdot Pr(X_{val} = i) = \sum_{i=1}^{|D|} i \cdot \frac{1}{|D|} \\ &= \frac{1}{|D|} \cdot \sum_{i=1}^{|D|} i = \frac{1}{|D|} \cdot \frac{|D| \cdot (|D|+1)}{2} = \frac{|D|}{2}\end{aligned}$$

Esistono diversi modi per migliorare lo schema della pwd:

- Rallentare il calcolo dell'hash: Si calcola H^l con l = 1000, quindi sono mille iterazione dell'hash. Per l'utente non è un problema ma per l'Adv si.
- Uso di un "salt": Il salt è un valore sconosciuto ad Adv ed è unico per utente:

  (s, hpwd = H(s||pwd))

  Abbiamo quindi che non può essere effettuato un pre-processing per calcolare precedentemente il dizionario. Usando un salt diverso per ogni file si può tentare solo una richiesta esaustiva.

8.7.3 Funzioni hash H come funzioni di derivazione

Per derivare una chiave segreta in modo che sia una stringa di bit uniformemente distribuita si possono provare diversi metodi come troncare alla lunghezza giusta oppure mappando l'informazione segreta su stringhe ad-hoc. Ma questo non è un approccio sicuro.
Esempio: Abbiamo una pwd di 28 lettere e abbiamo bisogno di una chiave a 128 bit. Se la pwd è in codice ASCII allora disponiamo di 224 bit, possiamo quindi di pensare di troncare e prendere solo i primi 16 byte. Questo però crea un problema i primi tre bit di ogni carattere sono "010" e quindi abbiamo che un Adv dovrebbe calcolare solo 2^{75} chiavi invece di 2^{128}.

Abbiamo quindi bisogno di una soluzione generica per derivare una chiave da un segreto condiviso ad alta entropia e non necessariamente uniforme.
Entropia → "il log della cardinalitá dello spazio delle possibili chiavi".

Definizione Una distribuzione di probabilità X ha m bit di **min-entropy** se per qualsiasi fissato valore di x risulta:

$$\Pr_{x \leftarrow X}[X = x] \leq 2^{-m}$$

Se modelliamo H come un ROM allora possiamo utilizzata come funzione di derivazione. Possiamo vedere ogni query fatta dall'attaccante come un tentativo di indovinare il valore. Se l'attaccante non invia x come query all'oracolo, allora H(x) é una stringa uniformemente distribuita ai suoi occhi. Quindi un Adv che effettua q query ha al più probabilità $q \cdot 2^{-m}$ di indovinare H(x).

8.7.4 Commitment Scheme

In molte occasioni è utile avere uno strumento attraverso il quale una parte può vincolarsi ad un messaggio m che consiste nel fare il commit di tale messaggio, inviando un valore vincolante *com* per il quale valgono due proprietà:

- Hiding (nascondere): il commitment non rivela nulla su m;
- Binding (legare): non esistono algoritmi PPT per la parte che fa il commitment dare in output un valore *com*, quindi vincolarsi, che successivamente potrà aprire in due modi diversi, ovvero dando in output due messaggi diversi m e m'.

In pratica, una volta che l'utente ha posto il messaggio all'interno della busta digitale questa può essere aperta soltanto per mostrare il messaggio contenuto, con la condizione che non deve rivelare nulla agli altri e inoltre colui che inserisce un messaggio all'interno può solo aprire la busta per mostrare il messaggio ma non aprire la busta e cambiare il valore che ha posto precedentemente dentro.

Uno schema **commitment** non interattivo è una tripla di algoritmi Π = (Gen, Com, Decom) tale che:

- Gen: probabilistico che genera i *params* pubblici;
- Com (params, r, m) → *com*: genera il commitment value com, su input params, una stringa random r ed il messaggio m;
- Decom(com) → (r, m), dà in output la stringa casuale r ed il messaggio m usati per calcolare com.

Nelle applicazioni, solitamente, uno schema di commitment viene utilizzato in due fasi:

- Fase di commitment: in cui Alice invia a Bob il valore vincolante com;
- Fase di decommitment: in cui Alice apre il commitment inviando a Bob (r,m). Bob usa (r, m) per verificare com = Com(params, r, m).

Le proprietá di Hiding e di Binding vengono formalizzate attraverso due esperimenti:

Esperimento commitment hiding: $\text{Hiding}_{A,Com(n)}$

1. Parametri: vengono generati come segue params $\leftarrow Gen(1^n)$;
2. L'Adv A prende in input params, e da in output i messaggi m_0 e $m_1 \in \{0,1\}^n$;
3. A sceglie uniformemente $b \in \{0,1\}$ e calcola com $\leftarrow Com(params, m_b, r)$;
4. A prende in input *com* e da in output il bit b';
5. L'output dell'esperimento è 1 se e solo se b' = b.

Esperimento commitment binding: $\text{Binding}_{A,Com(n)}$

1. Parametri: vengono generati come segue params $\leftarrow Gen(1^n)$;
2. L'Adv A prende in input params, e da in output (com, m, r, m', r');
3. L'output dell'esperimento è 1 se s solo se $m \neq m'$ e Com(params,m,r)) com = Com(params, m', r').

Definizione 5.13 Uno schema commitment **Com** è sicuro se per ogni Adv A PPT esiste una funzione trascurabile tale che:

$$Pr[Hiding_{A,Com}(n) = 1] \leq \frac{1}{2} + negl(n)$$
and
$$Pr[Binding_{A,Com}(n) = 1] \leq negl(n)$$

Usando il ROM è facile costruire uno schema commitment:

- per fare il commit di m, si sceglie $r \in \{0,1\}^n$ in modo uniforme e si dà in output *com* := H(r || m) (Gen e params non sono necessari poiché la funzione hash H é pubblica).
- per fare il decommit si invia (r,m) e il ricevente verifica che *com* = H(r || m).
- **Hiding:** se A non invia query del tipo H(r || ·), allora H(r || m) non rivela nulla di m. E poiché r é uniforme, A lo indovina con probabilità trascurabile.
- **Binding:** H é resistente a collisioni ⇒ se A ha successo allora A ha trovato una collisione.

Nota: schemi di commitment possono essere costruiti senza far ricorso al ROM, ma le costruzioni sono piú complicate e meno efficienti.
Un'applicazione reali di questi schemi può essere un'asta elettroniche o il voto elettronico.

Capitolo 9

Costruzioni di primitive simmetriche

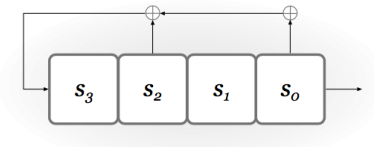
Nei precedenti capitoli abbiamo visto diverse costruzioni teoriche in questo capitolo vedremo se possono esistere e come vengono costruite.
Vedremo costruzioni euristiche efficienti di queste primitive. Con **euristiche** si intende che non possono essere provate sicure a partire da nessuna assunzione piú debole ma sono basate su principi di progettazione che a volte possono essere giustificati dall'analisi teorica.

9.1 Stream cipher

Fa uso di due algoritmi deterministici:

- **Init:** prende in input una chiave ed un IV(opzionale) e dà in output uno stato iniziale st.
- **GetBits (Next):** può essere invocata ripetutamente per avere in output una sequenza infinita di bit $y_1, y_2, ...$ a partire da st.

9.1.1 Linear-feedback Shift Registers: LFSR



Sono array di n registri con un meccanismo di retroazione (feedback loop) specificato da n coefficienti di feedback $c_0, c_1, ..., c_{n-1}$. Il numero n di registri è il **grado** dell'LFSR, infine ogni registro può memorizzare dei bit.

Funzionamento: al tempo t viene dato S_0 in output e viene eseguito uno shift a destra degli altri registri, S_3 viene calcolato tramite una funzione che utilizza i bit del registro S_0 e S_1. I coefficienti di feedback utilizzati sono quelli posti a 1 infatti: $c_0 = c_2 = 1$ e $c_1 = c_3 = 0$

In generale: se lo stato al tempo t è $s^t_{n-1}, ..., s^t_0$, allora il prossimo stato dopo il tick del clock è:

$$s^{t+1}_i = s^t_{i+1}, \text{per i} = 0, ..., \text{n-2}$$
$$s^{t+1}_{n-1} = \oplus^{n-1}_{i=0} c_i \cdot s^t_i$$

Se denotiamo con y_i i bit di output, allora:

$$y_i = s_{i-1}^0, \text{per } 1 = 0, ..., n$$
$$y_i = \oplus_{j=0}^{n-1} c_j \cdot y_{i-n+j-1} \text{ per i} > n$$

I primi n bit sono pertanto lo stato iniziale. Un esempio di LFSR è il seguente:
(0,0,1,1) t = 0
(1,0,0,1) t = 1 output = 1
(1,1,0,0) t = 2 output = 1
(1,1,1,0) t = 3 output = 0
(1,1,1,1) t = 4 output = 0
(1,1,1,1) t = 5 output = 1

Un LFSR può ciclare attraverso al più 2^n stati,la lunghezza del ciclo massimo prima che si inizi a ripetere è $2^n - 1$ (escludiamo lo stato con tutti 0 andrebbe in loop). Esistono configurazioni che permettono di avere una configurazione che fa esattamente $2^n - 1$.

9.1.2 Attacchi di ricostruzione

Gli LFSR hanno buone proprietá statistiche, ogni stringa di n bit si presenta con uguale frequenza nella sequenza di output. Tuttavia i bit prodotti sono facilmente predicibili.
Un attaccante può ricostruire l'intero stato di un LFSR di grado n dopo aver visto 2n bit output: $y_1, ..., y_n, y_{n+1}, ..., y_{2n}$
Questi primi bit rappresentano lo stato iniziale. Dei secondi invece è nota la forma. Pertanto possiamo determinare i coefficienti di feedback $c_0, ..., c_{n-1}$ col seguente sistema lineare:

$$\begin{aligned} y_{n+1} &= c_{n-1}\, y_n \oplus \cdots \oplus c_0\, y_1 \\ &\vdots \\ y_{2n} &= c_{n-1}\, y_{2n-1} \oplus \cdots \oplus c_0\, y_n. \end{aligned}$$

Per un LFSR di grado n di lunghezza massima le n equazioni sono linearmente indipendenti modulo 2. Infatti si può usare un qualsiasi algoritmo per le equazioni lineari per calcolare efficientemente i coefficienti.

Le soluzioni prevedono tutte di togliere la linearità:

1. Rendere il feedback non lineare:

$$s_i^{t+1} = s_{i+1}^t, \text{per i} = 0, ..., \text{n-2}$$
$$s_{n-1}^{t+1} = g(s_0^t, ..., s_{n-1}^t)$$

   dove g é una funzione non lineare.

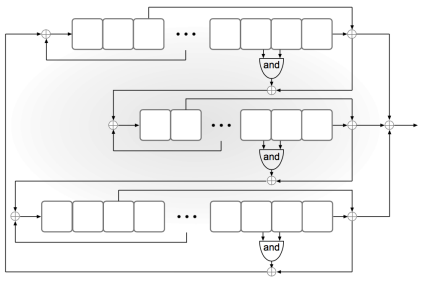
2. Combinazione: Introduce la non linearitá nella sequenza di output. Nella configurazione piú semplice abbiamo un LFSR modificato, in cui il bit di output é ottenuto calcolando una funzione non lineare g di tutti i registri, g deve essere bilanciata cioè stessa distribuzione di 1 e 0:

$$Pr[g(s_0^t, ..., s_{n-1}^t) = 1] \approx \tfrac{1}{2}$$

3. Una variente della precedente soluzione consiste nell'usare diversi LFSR, combinando assieme i bit di output dei singoli LFSR, attraverso una funzione non lineare g. Questi LSFR non devono avere per forza lo stesso grado. Anzi la lunghezza del ciclo viene massimizzata se i gradi sono diversi, bisogna anche avere cura che il bit dato in output non dipenda maggiormente da uno degli LFSR.

9.1.3 Trivium (2008)

Usa tre LFSR non lineari A (grado 93), B (grado 84) e C (grado 111) accoppiati:



Ha uno stato di 288 bit. Fu progettato per avere una descrizione semplice e un'implementazione hardware compatta.
L'output è lo XOR dei tre bit più a destra dei registri A, B, C.
Init(·) accetta:

- una chiave di 80 bit, caricata nei registri di A piú a sinistra
- un vettore IV di 80 bit, caricato nei registri di B piú a sinistra

Tutti i registri restanti sono posti a 0, eccetto i tre registri più a destra di C che sono posti a 1.
Lo stato iniziale st_0 si ottiene eseguendo GetBits 4288 volte e buttando sempre via i bit di output.
Inizialmente è stato molto utilizzato per la sua semplicità e efficienza, ma col passare del tempo la nascita di nuovi attacchi ne ha ridotto i margini di sicurezza. Può essere ancora usato.

9.1.4 RC4

É l'implementazione software di un LFSR.
ALGORITMO 6.1 Init

Input: prende una chiave k di 16 byte
Output: Lo stato iniziale (s,i,j)

```
for i = 0 to 255:
    S[i] := i
    k[i] := k[i mod 16]
j := 0

for i = 0 to 255:
    j := j + S[i] + k[i] // produce un indice
    Swap S[i] e S[j] // effettua lo scambio

i := 0, j := 0

return (S,i,j)
```

S dovrebbe rappresentare una permutazione uniforme di 256 byte. ciascun byte di S viene "swappato" almeno una volta in una locazione pseudocasuale.

ALGORITMO 6.1 GetBits

Input: Lo stato (S,i,j)
Output: byte y, stato (S,i,j) aggiornato

NOTA: tutte le addizioni avvengono in modulo 256.

```
i := i + 1
j := j + S[i] // j calcolato in modo pseudocasuale
Swap S[i] e S[j]
t := S[i] + S[j] // sommo i due registri appena scambiati
y := S[t]  // byte di output

return (S,i,j), y
```

Lo stato di S viene usato per generare la sequenza di output, i viene incrementato di 1 ad ogni invocazione di nuovo e ciascun byte di S viene "swappato" almeno una volta ogni 256 iterazioni, assicurando un buon mix della permutazione S.

NOTA: RC4 non fu progettato per usare un IV . Tuttavia, diverse implementazioni lo fanno, introducendo IV nell'array che contiene la chiave (prima o dopo la chiave). Sfortunatamente questa modalitá introduce debolezze in RC4. Il motivo è che IV è inviato in chiaro qunado il cifrario è usato in modo asincrono e questo rende parte di $k[]$ noto.

Attacco statistico contro RC4

In questo attacco sarà mostrato che in RC4 si ha molta più probabilità di ottenere 0 che un qualsiasi altro valore.

Siano S_t, lo stato S dopo t iterazioni di GetBits e S_0 lo stato iniziale. Se trattiamo S_0 come una permutazione uniforme di $\{0, .., 255\}$ risulta:

$$Pr[S_0[2] = 0 \wedge X = S_0[1] \neq 2] = \frac{1}{256} \cdot (1 - \frac{1}{255}) \approx \frac{1}{256}$$

Dove:

- $S_0[2]$ e $S_0[1]$ sono rispettivamente la prima e la seconda posizione dello stato iniziale;
- $S_0[1] \neq 2$: la prima posizione dello stato iniziale è diversa da 0 e da 2;
- $(1 - \frac{1}{255})$: probabilità che ci sia 2, questa probabilità tende a 1 quindi la differenza è quasi 0 e possiamo trascurarla.

Nella prima iterazione di GetBits abbiamo che i assume valore 1 mentre j := $S_0[i] = S_0[1] = X$, successivamente questi due stati vengono scambiati e quindi $S_0 1[X] = S_0[1] = X$.
Nella seconda iterazione di GetBits i := 2 mentre j diventa: j + $S_1[i] = X + S_1[2] = X + S_0[2] = X$.
É X perché j = X e che $S_1[i] = S_1[2] = S_0[2] = 0$, quindi abbiamo che X + 0 = X.
Successivamente $S_1[2]$ e $S_1[X]$ vengono scambiati e quindi:

$$S_2[X] = S_1[2] = S_0[2] = 0 \text{ e } S_2[2] = S_1[X] = X$$

Infine il valore di S_2 in posizione:
$S_2[i] + S_2[j] = S_2[2] + S_2[X] = 0 + X = X$, X verrà dato in output. Ma $S_2[X]$ ha proprio valore 0.
Dall'altra parte quando $S_0[2] \neq 0$, abbiamo che il secondo byte è uniformemente distribuito.

Quindi abbiamo che la Pr[il secondo byte di output sia 0]

$$= 1 \cdot Pr[S_0[2] = 0 \wedge X = S_0[1] \neq 2] + \frac{1}{256} \cdot Pr[S_0[2] \neq 0 \wedge X = S_0[1] \neq 2]$$
$$\approx 1 \cdot \frac{1}{256} + \frac{1}{256} \cdot (1 - Pr[S_0[2] \neq 0 \wedge X = S_0[1] \neq 2])$$
$$\approx \frac{1}{256} + \frac{1}{256} = \frac{2}{256}$$

- Calcoliamo la probabilità su due casi , uno in cui il primo byte è 0 e l'altro caso in cui non è ne 0 ne 2.
- $\frac{2}{256}$ indica che un avversario sa che ha probabilità doppia di ottenere 0.

Attacco utilizzando il vettore IV

Questo attacco è possibile su RC4 che utilizza IV. IV è usato nello standard di cifratura WEP.
L'attacco si basa sulla possibilità di estendere la conoscenza dei primi n byte della chiave k ai n+ 1 byte della chiave k.
Nota: se IV viene anteposto alla chiave k' allora k = IV || k' , e questo comporta che i primi k byte sono già noti.

Partiamo supponendo che IV è lungo 3 byte. Adv attenderà che IV abbia una determinata forma. Una buona forma studiata per questo attacco è IV = (3, 255, X) , X può essere un qualsiasi byte. Da questa forma di IV sappiamo che $k[0] = 3, k[1] = 255$ e $k[2] = X$.

É possibile verificare che dopo le prime 4 iterazioni del secondo ciclo $Init(\cdot)$ risultano:

$$S[0] = 3, S[1] = 0, \text{ e } S[3] = X + 6 + k[3] \tag{9.1}$$

$k[3]$ è il primo byte della chiave segreta.
Nelle successive 252 iterazioni, $S[0], S[1]$, e $S[3]$ non vengono modificati fino a quando $j \notin \{0, 1, 3\}$.
Se j assume valori in accordo alla distribuzione uniforme allora

$$Pr[j \notin \{0, 1, 3\}] = (\frac{253}{256})^{252} \approx 0,05 \tag{9.2}$$

Quindi il 5% delle volte $S[0], S[1]$, e $S[3]$ non vengono più modificati, pertanto sappiamo che il primo byte che GetBits dà in output sarà $S[3] = X + 6 + k[3]$, e quindi $k[3]$ è rivelato.
Anche se sempra che il 5% sia poco, normalmente la probabilità di indovinare andando a caso è dello 0.4%. In più cCollezionando un numero sufficientemente grande di campioni del primo byte di output per diversi IV di inizializzazione della forma giusta, Adv ottiene una stima molto accurata di $k[3]$.

9.1.5 ChaCha20

Introdotto nel 2008 per sostituire RC4. Utilizza il Mac Poly1305 per produrre una cifratura autenticata.
Richiede soltanto l'uso di tre operazioni che lavorano su parole di 32 bit
- $\boxplus$: addizione in modulo 32;
- Rot() : rotazione;
- XOR

Questo sistema è chiamato ARX-based.
Usa anche una permutazione fissa P su stringhe di 512 bit, tramite questa P definisce la funzione F(k, x) con chiave 256 bit, input di 128 e output di 512:

$$F_k(x) = P(\text{ const } ||k||x) \boxplus \text{ const } ||k||x \tag{9.3}$$

Quindi più precisamente:
Dati s $\in \{0,1\}^{256}$ ed IV $\in \{0,1\}^6 4$, l'output dello stream cipher è

$$F_s(IV||<0>), F_s(IV||<1>)... \tag{9.4}$$

dove <i> è la codifica binaria dell'intero i con 64 bit.
Può essere provato sicuro nel modello della permutazione casuale (simile al ROM), avendo accesso all'oracolo P e P^{-1} per valutare se P è casuale, se lo è allora F$(\cdot,\cdot)$ è una PRF.

9.2 Cifrari a blocchi

Un cifrario a blocchi é una permutazione con chiave efficientemente calcolabile, cioè $F : \{0,1\}^n$ x $\{0,1\}^l \rightarrow \{0,1\}^l : \forall k$ abbiamo

$$F_k(x) =^{\text{def}} F(k,x) \text{ è una permutazione} \tag{9.5}$$

ed F_k ed $F^{-1} - k$ sono coefficienti calcolabili.
n è la lunghezza della chiave, l invece è la lunghezza del blocco (sono quindi i nostri parametri di sicurezza).
Nella pratica n ed l sono costanti fissate.
Un cifrario a blocchi viene generalmente considerato buono se l'attacco migliore che si conosce ha complessitá di tempo equivalente ad una ricerca esaustiva di k.
I cifrari a blocchi vengono progettati per esibire un comportamento pari a PRP. Modellarli come una PRP permette di fornire delle prove di sicurezza per costruzioni basate sui cifrari a blocchi. **Nota:** cifrari a blocchi non sono schemi di cifratura. Tuttavia, la terminologia standard per attacchi contro un cifrario a blocchi F é la stessa.

La chiave k non nota in tutti i possibili attacchi (Known-plaintext attack,Chosen-plaintext attack,Chosen-ciphertext attack), infatti l'obbiettivo di un Adv è quello di distinguere F_k da una permutazione uniforme e di recuperare la chiave k.
Nota: Una permutazione pseudocasuale non puó essere distinta da una permutazione uniforme rispetto ad un attacco di tipo chosen-plaintext (si richiede l'uso di un oracolo). Mentre una permutazione pseudocasuale forte non puó essere distinta da una permutazione uniforme rispetto ad un attacco di tipo chosen-ciphertext (il cifrario a bocchi approssima a questo).

Pseudocasualità nella pratica : L'ideale è che cambiando un bit nell'input del cifrario a blocchi $F_k(\cdot)$ con k uniforme e non nota all'avversario, si dovrebbe ottenere un output quasi del tutto indipendente dall'output precedente, che significa dire che ogni bit dell'output dovrebbe poter cambiare con probabilità $\frac{1}{2}$. Nella pratica quello che vogliamo ottenere è che cambiando un singolo bit nell'input cambi quasi del tutto l'output.

9.2.1 Paradigma della confusione e della diffusione

Costruiamo, quindi, una F che soddisfa tale proprietà secondo la seguente idea: Costruire una permutazione F che sembra casuale con una lunghezza di blocco grande da molteplici permutazioni $\{F_i\}_i$ più piccole casuali o che sembrano casuali:

$$F_k(x) = f_1(x_1)||...||f_n(x_n) \tag{9.6}$$

Le funzioni f_1, dette funzioni di round, introducono confusione in F.
Però f_k non è casuale perché se due input x e x' differiscono soltanto nel primo byte e con il resto dei byte uguali avremo che i due output F(x) e F(x') saranno differenti soltanto nel prime byte. Il problema è dovuto dal fatto che le funzioni di round introducono **confusione locale**. Quindi, è necessario

avere un passo che introduca **diffusione**, ovvero la confusione deve essere estesa a tutti gli altri byte. Pertanto, i bit dell'output vengono permutati (mixing permutation) così da diffondere i cambiamenti locali e tale operazione può essere effettuata più volte. Questa è la ragione per cui f viene chiamato **funzioni di round**.
Questo metodo viene chiamato **paradigma della confusione e diffusione** potrebbe essere un modo attraverso il quale si cerca di riprodurre l'astrazione della pseudocasualità dell'output. Quindi, attraverso passi che realizzano confusione e diffusione globale, si potrebbe, con un certo numero di round, riuscire ad ottenere un output che sia difficile da distinguere dall'output prodotto da una funzione scelta a caso o una permutazione scelta uniformemente a caso. La realizzazione di questo può essere fatta attraverso due forme: reti a sostituzione e permutazione e reti di Feistel.

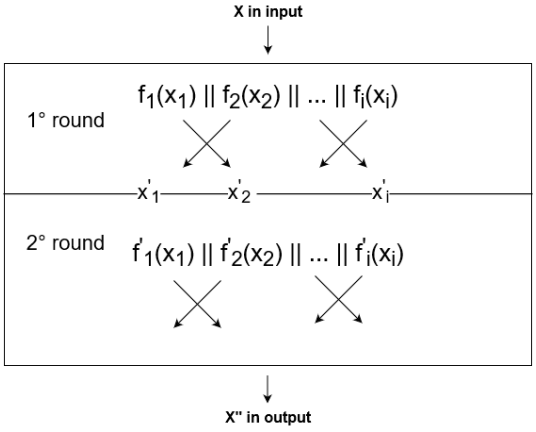


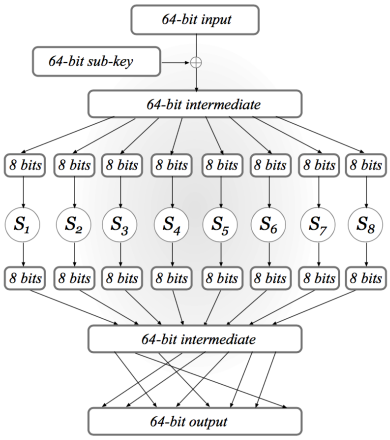
Figura 9.1: Idea funzionamento diffusione e permutazione

9.2.2 Rete a sostituzione e permutazione (SPN)

Una SPN è un'implementazione diretta del paradigma della confusione e della diffusione. L'idea di fondo è che invece di usare una porzione della chiave k per scegliere una f_1 fissiamo una funzione di sostituzione pubblica S. Quindi diremo che S è una S-box e useremo la chiave k o una porzione di essa per specificare la funzione f come: $f(x) = S(k \oplus x)$. Consideriamo una rete a sostituzione e permutazione con un blocco di 64 bit, basata su una collezione di S-box $S_1, ...S_8$ di 8 bit. Viene effettuato, quindi, un XOR tra l'input e la chiave del round ottenendo una stringa intermedia di 64 bit.
Questa stringa viene diviso in gruppi di 8 bit dove per ogni gruppo diventa l'input della -box che in qualche modo processa stampando l'output calcolato. Da qui si applica una permutazione sui bit intermedi per ottenere i 64 bit di output. In un round ci sono i seguenti passi:

- **Key mixing**: Poni $x := x \oplus k$, con k sottochiave del round corrente;
- **Substitution**: Poni $x := S_1(x_1)||...||S_8(x_8)$, con x_i i-esimo byte di x
- **Mixing Permutation**: Permuta i bit producendo l'output del round

Le S-box e la mixing permutation sono pubbliche. La chiave del cifrario è detta master key, le sottochiavi sono derivate da quest'ultima e ogni round ne usa una differente.



Una SPN ad r round ha r round pieni con key mixing, S-box substitution e mixing permutation piú un passo finale di key mixing.

Proposizione 6.3 Sia F una funzione con chiave definita da una SPN in cui le S-box sono tutte permutazioni. Allora, indipendentemente dal numero di round e dall'algoritmo di schedulazione delle chiavi, F_k é una permutazione per qualsiasi valore di k.

DIM: Basta mostrare che il singolo round è invertibile, questo implica che tutta F_k è invertibile.

- Mixing permutation è invertibile
- S-Box sono delle permutazioni e sono invertibili
- Key mixing (xor) è invertibile, utilizzando la sottochiave opportuna

Effetto valanga

Le S-box sono tali che modificando un singolo bit di input si modificano almeno due bit di output della S-box. le mixing permutation sono progettate in modo tale che i bit di output di una data S-box sono usati come input in molteplici S-box nel round successivo. Quindi abbiamo che un piccolo cambiamento nell'input ha effetto su tutti i bit dell'output, questo prende nome di effetto valanga.
Le S-Box per avere un buon effetto valanga non devono essere scelte a caso ma devono essere progettate con cura. Per questo una S-box deve essere progettata in modo che cambiando un bit nell'input se ne cambino due almeno nell'output.
Se un cifrario a blocchi deve realizzare una permutazione pseudocasuale forte, allora l'effetto valanga deve essere prodotto anche dalla permutazione inversa.

Attacchi contro SPN con un round ridotto

Il numero di round in una SPN è cruciale.
- Con un solo round e senza il passo finale di key-mixing, un Adv preso una coppia (x,y) può recuperare la chiave partendo da y. Infatti sa invertire la mixing permutation e le S-Box perchè pubbliche, e fatto questo sa calcolare lo xor tra l'input delle S-boc e x.

- Consideriamo ora una SPN con un round ed il passo di key mixing finale. Assumendo che la lunghezza del blocco sia 64, le S-Box abbiano come lunghezza dei bit di input/output, che le sottochiavi $k_1 e k_2$ usate nei due passi di key mixing sono indipendenti allora la master key sarà K = $K_1||K_2$ (128 bit).
Un Adv disponendo di una coppia (x,y) può agire in 2 modi:

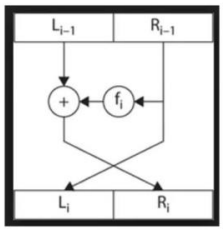
1. Enumera tutte le possibili chiavi K_2 ($2^6 4$), per ognuna di esse può invertire il passo finale di key mixing e usando l'attacco descritto precedentemente ottiene il valore di K_1. In 2^{64} produce una lista di 2^{64} chiavi K = $K_1||K_2$, da qui usando coppie (x_i, y_i) riduce la lista fino a trovare la chiave K giusta.

2. enumera tutti i possibili valori di un byte di K_2 (2^8 in totale), le posizioni dei bit scelti dipendono dalla mixing permutation usata nel round (pubblica) per ognuno di essi, puó invertire il passo finale di key mixing, ottenendo l'output di una S-Box usando l'attacco precedente ottiene, per il byte di K_2, i valori di un byte di K_1. Per ogni ipotesi sul byte di K_2 ottiene un'unica scelta possibile per il byte di K_1. Ripete il processo per ognuna delle S-Box. Ripetendo l'attacco per tutti gli 8 byte, Adv ottiene 8 liste che contengono tutte le possibili master key, il tempo totale per fare ciò p 2^{11}. L'attacco é possibile perché parti differenti della chiave possono essere isolate da altre è quindi necessaria un'ulteriore diffusione per essere sicuri che tutti i bit della chiave influenzino tutti i bit dell'output, quindi più round sono necessari.

- É facile vedere che una SPN a 2 round non é pseudocasuale. Infatti se Adv ottiene il risultato della valutazione della SPN su due input, x ed x_0, che differiscono in un solo bit, allora gli output corrispondenti differiranno in pochi bit.

9.3 Reti di Feistel

Le reti di Feistel rappresenta un altro modo per la costruzione di cifrari a blocchi. Il vantaggio rispetto alla SPN è legato alle funzioni sottostanti usate nelle reti di Feistel, infatti, contrariamente alla S-box usate nelle SPN, non devono essere invertibili. Quindi le reti di Feistel possono essere viste come un modo per costruire una funzione invertibile tramite componenti non invertibili. Rispetto alle SPN, c'è meno struttura nella rete ed opera attraverso una serie di round in cui per ognuno viene applicata una funzione del round con chiave, tipicamente costruita tramite S-box e mixing permutation.
Nelle reti di Feistel bilanciate, la f-esima funzione di round $\hat{f}$.



- Prende in input una sottochiave K_1 ed una stringa R di l/2 bit dove l è la lunghezza della stringa x;
- Stampa in output una stringa di l/2 bit.

Una volta scelta una master key K, che determina le sottochiavi K_1, definiamo le funzioni specifiche di ogni singolo round:

$$f_i : \{0,1\}^{l/2} \to \{0,1\}^{l/2} \text{ come } \quad f_i(R) \overset{\text{def}}{=} \hat{f}(K_i, R) \tag{9.7}$$

Le $\hat{f}$ sono fissate e pubblicamente note, in più, dipendono dalla master key (non nota all'avversario). Quindi con l stiamo indicando la lunghezza di blocco. L'input viene rappresentato tramite due sottostringhe di l/2, una parte destra e una sinistra. Il round i-esimo opera come segue: l'input di tale round sono L_{i-1} e R_{i-1} ove l'output viene calcolato in:

$$L_i = R_{i-1} \text{ concatenato con } R_i = L_{i-1} \oplus f_i(R_{i-1}) \tag{9.8}$$

Le reti di Feistel sono invertibili.

Proposizione 6.4 Sia F una funzione con chiave definita da una FN. Indipendentemente dalle funzioni di round $\{\overset{\wedge}{f_i}$ e dal numero di round, F_k é una permutazione efficiente invertibile per qualsiasi valore di K.

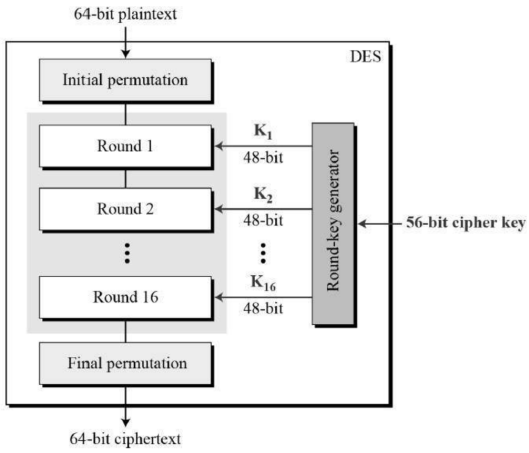
Prova: Per rendersene conto, basta considerare un singolo round e notare che:

$$(L_i, R_i) \to \begin{cases} R_{i-1} = L_i \\ L_{i-1} = R_i \oplus f_i(R_{i-1}) \end{cases} . \tag{9.9}$$

In particolare, si noti che le f_i vengono valutate in una sola direzione.

9.3.1 Data Encryption Standard (DES)

Il DES è una rete di Feistel a 16 round con lunghezza di blocco di 64 bit e lunghezza della chiave di 56 bit. La funzione di round prende in input una sottochiave di 48 bit ed una stringa di 32 bit. L'algoritmo di scheduling delle chiavi DES deriva le sottochiavi dalla master key. Da una master key K di (56 bit) si generano 16 chiavi di 48 bit ciascuna. Di questi 48 bit 24 dalla parte sinistra di K e 24 dalla parte destra.



La funzione di round del DES è costruita usando le SPN:

$$\hat{f}_i(K_i, R) \;\; K_i \in \{0,1\}^{48}, \;\; R \in \{0,1\}^{32} \tag{9.10}$$

Viene usata la stessa funzione $\hat{f}_i$ per tutti i round, i passi di quest'ultima sono:

- R viene esteso attraverso la funzione di espansione E : R $\rightarrow$ E(R) a 48 bit
- E(R) $\oplus$ Ki é una stringa di 48 bit, input per le S-box

Le S-box non sono invertibili (gli input sono piú lunghi degli output). La specifica dell'algoritmo é completamente pubblica. Soltanto la master key é segreta. Anche un minimo cambiamento puó introdurre debolezze, hanno le seguenti proprietà:

1. ciascuna S-box é una funzione 4-a-1
2. ciascuna riga della tabella contiene tutte le possibili stringhe di 4 bit esattamente una volta
3. cambiando un bit ad ogni stringa di input di una S-box, si cambiano sempre almeno due bit di output

I 4 bit di output di qualsiasi S-box influenzano l'input di 6 S-box nel round successivo.
Come per le SPN, abbiamo un effetto esponenziale: dopo 7 round ci aspettiamo che tutti i 32 bit della parte destra siano influenzati dalla differenza di un bit nei due input. DES ha 16 round. Assicurano che su input simili gli output sembrino indipendenti.

Attacchi contro DES ridotto

Glia attacchi contro il DES che consideriamo sono a 1 , 2 e 3 round. SI considerà un Adv che conosce $\{(x_i, y_i)\}$ dove $y_i = DES_K(x_i)$.

- 1 round: data la coppia (x, y) risulta che y = (L_1, R_1) e x = (L_0, R_0). Poiché $R_0 = L_1$ ed $R_1 = L_0 \oplus f_1(R_0)$ possiamo ricavare $f_1(R_0) = L_0 \oplus R_1$ implca che conosciamo una coppia input/output per f_1 ovvero $(R_0, L_0 \oplus R_1)$. Applicando l'inversa della mixing permutation ad $L_0 \oplus R_1$ otteniamo le stringhe di output delle S-box. Ci sono 4 possibili valori di input (di 6 bit) per ognuna delle S-box. L'input alle S-box è l'xor tra $E(R_0)eK_1$. Essendo R_0 noto ci risulta noto anche la sua funzione di espansione (E(R_0)) e quindi possiamo calcolare i 4 possibili valori per ciascuna porzione di 6 bit di K_1, quindi si fa lo xor tra i 4 possibili input e della loro porzione della funzione di espansione corrispondente. Le coppie ora da provare sono 2^16.

- con 2 round conosciamo anche L_2eR_2, disponiamo quindi di due coppie di input f_1ef_2, quindi si può usare lo stesso metodo dell'attacco ad un round per derivare K_1eK_2 in tempo circa $2 \cdot 2^16$

- con 3 round data la coppia (x, y) conosciamo soltanto (L_0, R_0) ed (L_3, R_3). Sappiamo però che l'xor degli output di f_1 ed f_3 é uguale ad $L_0 \oplus R_3$ (che é noto). Inoltre, R_0 ed R_2, input di f_1 ed f_3, rispettivamente, sono noti. Quindi possiamo determinare l'input di f_1 ed f_3 tramite lo xor dei loro output. Si procede provando ad indovinare e quindi fissare un valore per K_L e utilizziamo questo valore per calcolare i primi 4 bit di input per le S-box di f_1 ed f_3. Fatto questo possiamo calcolare l'xor tra i bit di output ottenuti valutando f_1 ed $_3$, li compariamo con i bit corrispondenti di $L_0 \oplus R_3$, se non abbiamo corrispondenza proviamo per un'altro valore K_L. La complessitá dell'attacco é data da relativamente all'uso di (x, y), circa $2 \cdot 2^{28}$ passi. Relativamente all'uso di (x', y'), circa 2^{24} passi per la ricerca esaustiva oppure circa $2 \cdot 2^{12}$ passi per ripetere l'attacco. In entrambi i casi, non piú di 2^{30} passi, che é molto meno di 2^{56}.

Sicurezza del DES oggi

Ad oggi la lunghezza della chiave é troppo corta per la potenza computazionale attuale. Anche la lunghezza del blocco in alcune applicazioni é fonte di preoccupazione.
Esempio: in CRT-mode la sicurezza dipende dalla lunghezza del blocco, e A vince con probabilità $\frac{2 \cdot q^2}{2^l}$ se ottiene q coppie (m, c), quindi usando DES abbiamo l = 64 e q = 2^{30} e risulta:

$$\frac{2 \cdot q^2}{2^l} = \frac{2 \cdot (2^{30})^2}{2^l} = \frac{2 \cdot 2^{60}}{2^{64}} = \frac{1}{2^3} = \frac{1}{8} \tag{9.11}$$

è una probabilità troppo alta.
Col tempo sono nate altre tecniche di attacco oltre alla ricerca esaustive, ma quest'ultime per ridurre il tempo computazionale hanno bisogno di spazio di memoria e sono:

- crittoanalisi differenziale (Biham e Shamir 1991): attacco chosen plaintext e richiede 2^{47} coppie e rompe in tempo 2^{37}

- crittoanalisi lineare (Matsui, 1993): attacco known plaintext e richiede 2^{43} coppie e rompe in tempo 2^{39}

Sono quindi più attacchi teorici che pratici.

Cifratura doppia

Per migliorare la sicurezza del DES si è pensato di applicarlo due volte. Abbiamo così un nuovo cifrario con chiave 2n definito come segue:

$$F'_{K_1,K_2}(x) \stackrel{\text{def}}{=} F_{K_2}(F_{K_1}(x)) \tag{9.12}$$

La chiave a questo punto è di 112 bit, e anche se in un primo momento si può pensare che l'attacco migliore richieda tempo 2^{112}. Si può sfruttare infatti l'attacco denominato meet-in-the-middle. Supponiamo di conoscere una particolare coppia (X,Y), dove X è il testo in chiaro e Y il testo cifrato, e di voler determinare la chiave (K_1, K_2) utilizzata nella cifratura di x. Per prima cosa memorizziamo in una lista tutte le cifrature di X usando tutte le 2^{56} possibili chiavi di K_1. Successivamente facciamo lo stesso con y ma decifrandolo con tutte le possibili 2^{56} possibili chiavi di K_2. A questo punto conoscendo la coppia (K_1, K_2), basta confrontare i valori cifratura e decifratura per x e y contenute nelle liste: $F_{K_1}(x) = F^{-1}_{K_2}(y)$, se risulta uguale per due valori allora potremmo aver trovato la coppia desiderata. Questo atacco richiede tempo $O(n \cdot 2^n)$ e spazio $O((n + l) \cdot 2^n)$

Cifratura doppia

In questo caso è un DES iterato tre volte, e lo definiamo come segue:
Usando tre chiavi indipendenti

$$F''_{K_1,K_2,K_3}(x) \stackrel{\text{def}}{=} F_{K_3}(F^{-1}_{K_2}(F_{K_1}(x))) \tag{9.13}$$

Oppure usando due chiavi indipendenti

$$F''_{K_1,K_2}(x) \stackrel{\text{def}}{=} F_{K_1}(F^{-1}_{K_2}(F_{K_1}(x))) \tag{9.14}$$

L'invocazione della decifratura del DES (F^{-1}_K) è per compatibili con il DES a singolo round.

D'altra parte, se F é una SPRP, allora F^{-1} é una SPRP e quindi non ci sono problemi in generale in questo uso.

- **Sicurezza della prima variante:** la chiave é lunga 3n bit ma, per l'attacco precedente contro la cifratura doppia - che si applica anche qui otteniamo sicurezza rispetto ad attacchi di tempo $\leq 2^{2n}$
- **Sicurezza della seconda variante:** la chiave é lunga 2n bit. Al momento non si conoscono attacchi di complessitá migliore di 2^{2n}, dato soltanto un piccolo numero di coppie (x, y), è una buona scelta nella pratica

Il triplo DES ha però uno svantaggio lunghezza di blocco relativamente piccola e cifrario complessivamente lento.

9.3.2 Advanced Encryption Standard (AES)

Sostiutisce il DES, gli attacchi migliori conosciuti ad oggi sono di ricerca esaustiva. Ha lunghezza di blocco di 128 bit. Puó usare chiavi di 128; 192 e 256 bit. La lunghezza della chiave influisce sulla schedulazione delle chiavi e sul numero di round non influisce sulla struttura di alto livello di ciascun round. AES é essenzialmente una rete SPN Un array di 16 byte di taglia 4 x 4 (lo stato di AES) viene modificato durante la computazione.

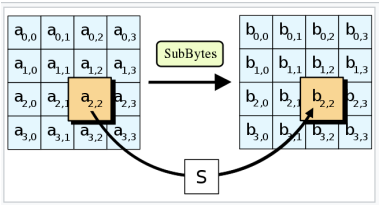
Struttura dell'AES

Lo stato iniziale corrisponde all'input. La struttura di ogni round é costituita da 4 passi:

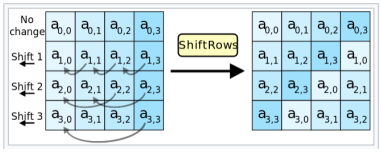
1. **AddRoundKey:** Una sottochiave di round di 128 bit viene "aggiunta" tramite xor allo stato:

$$b_{2,2} = a_{2,2} \oplus k_{2,2} \tag{9.15}$$

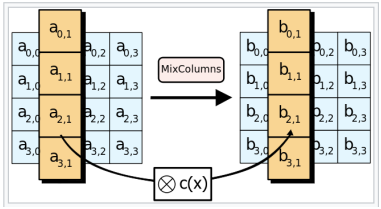
2. **SubBytes:** Ciascun byte dello stato viene sostituito con un altro byte, in accordo ad una tabella di lookup S.



3. **ShiftRows:** I byte di ciascuna riga vengono traslati (shift) a sinistra. Riga i → shift di i posizioni.



4. **MixColumns:** Una trasformazione invertibile viene applicata ai 4 byte di ogni colonna (moltiplicazione matriciale).



se due input differiscono in b > 0 byte, allora applicando la trasformazione, i due output differiscono in almeno 5-b byte.

Nel round iniziale viene effettuato soltanto il passo AddRoundKey. Nei round intermedi i passi sono SubBytes, ShiftRows, MixColumns e AddRoundKey. Nel round finale, MixColumns non viene eseguito. I round cambiano in base al numerod i bit della chiave:

- 128 bit ha 10 round
- 192 bit ha 12 round
- 256 bit ha 14 round

Quando in uno schema viene richiesto di utilizzare una permutazione pseudocasuale forte (SPRP) AES è una scelta eccellente da fare.

9.4 Costruzioni reali di funzioni hash

9.4.1 Costruzione Davies-Meyer

Un modo per costruire una funzione di compressione collision resistant usando un cifrario a blocchi è il seguente:

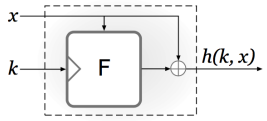
$$F : \{0,1\}^n x \{0,1\}^l \rightarrow \{0,1\}^l \tag{9.16}$$

un cifrario a blocchi con lungh. chiave n e lungh. blocco l. dove:

- $\{0,1\}^n$: è la chiave del cifrario a blocchi;
- $\{0,1\}^l$: è la lunghezza del cifrario a blocchi;
- $\{0,1\}^n x \{0,1\}^l$: è l'input della funzione.

Definiamo quindi:

$$h : \{0,1\}^{n+l} \to \{0,1\}^l \text{ come } h(k,x) \stackrel{\text{def}}{=} F_k(x) \oplus x. \tag{9.17}$$



Si prende un valore x e una chiave k e si produce un pad tramite un cifrario a blocchi che usa una funzione F. Successivamente si fa lo XOR tra l'output di F e x così da ottenere il valore della funzione di compressione h(k,x).

9.4.2 Ideal cipher model (ICM)

Per provare che h sia collision resistant assumendo che F sia SPRP si usa il Modello del cifrario ideale.

Nell'ICM tutte le parti hanno accesso ad un oracolo per F e F^{-1}, si ha una permutazione totalmente casuale $\forall$ k, x, e risulta F^{-1}(k, F(k,x)) = (k,x).
Le query sono l'unico modo per calcolare F(k,x) ed F^{-1}(k,y).

Possiamo osservare che questo modello è molto più forte del ROM.Non è possibile tener conto di attacchi **key-related attack** (cioè che sfruttano le dipendenze delle chiavi). Non ci son chiavi deboli. F_k si dovrebbe comportare casualmente anche se K è nota. **Problema:** non è detto che un cifrario reale F soddisfi queste proprietà.

Teorema 6.5 : Se F viene modellata come un cifrario ideale, allora la costruzione di Davies-Meyer dà una funzione di compressione collision resistant. Ogni Adv che sfrutta $q < 2^{l/2}$ query all'oracolo, trova una collisione con probabilità al più $q^2/2^l$
NOTA: é stato mostrato che ROM e ICM sono equivalenti, in accordo ad una nozione che estende la nozione di indistinguibilitá e si chiama *indifferenziabilitá*.

9.4.3 MD5

Progettata nel 1991. Nel 1993 (Der Boer e Bosselaers) pseudo-collisione e nel 1996 (Dobbertin) prima collisione. Nel 2004 un team cinese (Wang e altri) di crittoanalisti ha presentato un metodo efficiente per trovare collisioni L'attacco é stato migliorato diverse volte successivamente e oggi bastano pochi minuti su PC. Non dovrebbe essere utilizzata (la postale la usa).

9.4.4 Secure hash algorithm (SHA)

Esistono diversi tipi di SHA: SHA-0, SHA-1, SHA-2, SHA-3.

- SHA-0: solo presentato mai implementato;
- SHA-1: Introdotto nel 1995 ha 160 bit di output. Nel Febbraio del 2017 è stata trovata una collisione esplicita in circa 2^{63}. Non deve più essere utilizzato;

- SHA-2: ha due version SHA-256 e SHA-512 con rispettivamente 256 e 512 bit di output.

Tutte le funzioni SHA tranne SHA-3 sono realizzate utilizzando lo stesso schema di progettazione:

- viene progettata una funzione di compressione utilizzando la costruzione di Davies-Meyer ad un cifrario a blocchi
- il dominio viene esteso usando la trasformazione di Merkle-Damgard.
- il cifrario a blocchi, identificato retroattivamente nell'analisi della costruzione e denotato con i nomi SHACAL-1 (per SHA-1) e SHACAL-2 (per SHA-2), non é usato per la cifratura ma progettato esplicitamente per la costruzione

Il cifrario a blocchi utilizzato funziona solo per creare funzioni hash e non per crittografare.

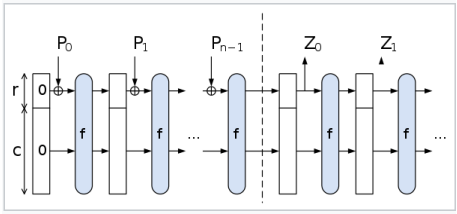
9.4.5 SHA-3 (Keccak)

Supporta output di 256 e 512 bit. SHA-3. Utilizza una permutazione f senza chiave con una lunghezza di blocco molto grande di 1600 bit, non utilizza la trasformazione di Merkle-Damgard sfrutta un approccio nuovo, denominato sponge construction, per gestire input di lunghezza arbitraria. Puó essere analizzata nel modello della permutazione casuale.
Nota: il modello della permutazione causale é un modello piú debole dell'ideal cipher model (fissando k nel secondo si ottiene il primo).

Sponge construction

Opera in due fase:

1. **Sponge** (assorbimento): lo stato iniziale è composto da tutti 0. Successivamente la parte r dello stato iniziale viene messo in XOR con il blocco P_0 e viene inviato ad una funzione f che darà poi in output così un nuovo stato, si continua così con tutti i P_{n-1} stati. Questo corrisponde ad un assorbimento;
2. **Squeeze** (spremitura): Nella seconda parte si inizia von l'ultimo stato ottenuto dalla fase di sponge, qui vengono estratti i singoli digit effettuando permutazioni.



Capitolo 10

One-way Function

La pseudocasualità è rappresentata in tre forme: Generatori PRG, Funzioni PRF e Permutazioni PRP. Queste astrazioni possono essere realizzati tramite le funzioni one-way, definite come funzioni **facili da calcolare ma difficili da invertire**.

Sia f : $\{0,1\}^* \to \{0,1\}^*$ una funzione. Sia l'avversario A un algoritmo efficiente ed indiciamo con n il parametro di sicurezza. Si può definire l'esperimento:

> $\text{Invert}_{A,f}(n)$
>
> 1. Il challenger sceglie uniformemente $x \in \{0,1\}^n$ e calcola y = f(x);
> 2. A riceve in input 1^n e y dà in output x';
> 3. L'output dell'esperimento è 1 se f(x') = y (funziona invertita); altrimenti da 0.

A non deve trovare necessariamente x. Basta una pre-immagine x' tale che f (x') = y = f(x).

Definizione 8.1 . Una funzione f : $\{0,1\}^* \to \{0,1\}^*$ è one way se:

1. (Facile da calcolare): esiste un algoritmo di tempo polinomiale M_f per calcolare f, per ogni x risulta M_f (x) = f (x);
2. (Difficile da invertire): per ogni A ppt, esiste una funzione trascurabile negl(n) tale che

$$Pr[Invert_{A,f}(n) = 1] \leq negl(n) \tag{10.1}$$

Il secondo requisito equivale a dire che $\forall A$ ppt, $\exists$ negl(n):

$$Pr_{x \leftarrow \{0,1\}^n}[A(1^n, f(x)) \in f^{-1}(f(x)))] \leq negl(n) \tag{10.2}$$

Informalmente: La probabilità che scelto un x uniformemente a caso e che A avendo il parametro di sicurezza e f(x) restituisca una preimmagine che appartiene all'insieme delle preimmagini di f(x) deve essere trascurabile.
NOTA: l'inverso di una funzione invertibile è: $\exists$ A PPT ed una funzione γ non trascurabile tale che A inverte f(x) con probabilità almento $\gamma(n)$. **Esempio:** se esiste A che inverte f con probabilità n^{-10} per tutti i valori pari di n (ma fallisce sui dispari), allora f non è one-way.

10.1 Famiglia di permutazioni

Spesso saremo interessati a funzioni one-way con proprietá strutturali aggiuntive f preserva la lunghezza se |f (x)| = |x| per ogni x.
Una funzione one-way che preserva la lunghezza ed é 1-a-1 é una permutazione one-way, cioè ogni valore ha soltanto un'unica pre-immagine. Nonaste ciò è ancora difficile trovare x in tempo polinomiale.

Molte papabili funzioni one-way sono definite per certi parametri I . La funzione f_I dovrebbe essere one-way sulle scelte di I. Conviene considerare allora famiglie di funzioni (permutazioni) one-way.

Definizione 8.2 Una tripla Π = (Gen, Samp, f) (Genova, Sampdoria, fiorentina) di algoritmi PPT è una famiglia di funzioni se:

1. L'algoritmo di generazione dei paramentri Gen(1^n) dò in output I, con |I| ≥ n. Ciascun I definisce D_I (dominio) ed R_I(codominio) tali che:

$$f_I : D_I \to R_I \tag{10.3}$$

2. L'algoritmo di campionamento Samp(I) dà in output un elemento uniformemente distribuito di D_I

3. L'algoritmo di valutazione f deterministico, su input I ed $x \in D_I$, dà in output $y \in R_I$. Scriveremo y := f_I(x).

Π è una famiglia di permutazioni se, per ogni I ← restituiscono da Gen(1^n), risulta $D_I = R_I$ e la funzione f_I: $D_I \to R_I$ è una biiezione.
L'esperimento $Invert_{A,f}(n)$ è definito come segue:

$Invert_{A,f}(n)$

1. Il challenger sceglie I ← $Gen(1^n)$, $x \leftarrow Samp(I)$ e calcola y := $f_I(x)$.
2. A riceve in input I e y e dà in output x'.
3. L'output dell'esperimento è 1 se $f_I(x') = y$, altrimenti 0.

Definizione 8.3 Una famiglia di funzioni (permutazioni) Π = (Gen, Samp, f) è one way se per ogni A PPT esiste una funzione trascurabile negl(n) tale che

$$Pr[Invert_{A,f}(n) = 1] \leq negl(n) \tag{10.4}$$

Allo stato attuale delle nostre conoscenze non sappiamo se esistono. Congetturiamo (assumiamo) la loro esistenza. Congettura basata sul fatto che diversi problemi computazionali naturali hanno ricevuto notevole attenzione ma non sono stati trovati algoritmi risolutivi di tempo polinomiale.

10.1.1 Fattorizzazione

Presi due numeri interi è facile farne il prodotto ma è difficile trovare i due fattori:

$$f_{mult}(x, y) = x \cdot y \tag{10.5}$$

Se non poniamo alcuna restrizione è faciel da invertire. Infatti se il numero è fari basta dividere per 2 e otteniamo così i 2 fattori e quindi l'inversa della funzione. Quindi si restringe il campo al dominio di x e y numeri primi con lunghezza uguale e molto grandi.

10.1.2 Subset Sum

Il problema Subset Sum ed é definito da:

$$f_{SS}(x_1, ..., x_n, J) = (x_1, ..., x_n, \sum_{j \in J x_j} mod 2^n) \tag{10.6}$$

dove

- $x_1, ..., x_n$ sono stringhe di n bit viste come interi
- J stringa di n bit che specifica un sottoinsieme di $\{1, ..., n\}$

Invertire f_{SS} su $(x_1, ..., x_n$, y) significa trovare un:

$$Z \subseteq \{1, ..., n\} \text{ tale che } y = \sum_{j \in Z x_j} \text{ mod } 2^n \tag{10.7}$$

Sappiamo che subset sum è NP-completo, ma anche se $P \neq NP$ non implica che f_{SS} è one-way.
Con $P \neq NP$ abbiamo che ogni algoritmo di tempo polinomiale fallisce nel risolvere il problema Subset Sum su almeno un input. D'altra parte f_{SS} , per essere one-way, richiede che ogni algoritmo di tempo polinomiale fallisca nel risolvere il problema (almeno per certi parametri) quasi sempre. Pertanto, la nostra convinzione che f_{SS} sia one-way é basata sul fatto che non conosciamo algoritmi polinomiali che risolvono il problema anche con probabilitá bassa su istanze casuali, piuttosto che sul fatto che il problema sia NP Completo.

Sia Π = (Gen; Samp; f) definita come segue:

- Gen(1^n) → (p,g), dove p è un primo di n bit e g ∈ $\{2, ...mp - 1\}$ è un elemento speciale detto generatore di Z_p^*.
- Samp(p.g) → x ∈ $\{1, ..., p - 1\}$ restituisce un x scelto uniformemente
- $f_{p,g} = [g^x \text{ mod } p$

$f_{p,g}$ risulta efficientemente calcolabile e 1-a-1, e dunque é una permutazione. La difficoltá di inversione é basata sulla difficoltá congetturata del problema del logaritmo discreto.

10.2 Predicati Hard-core

Dire che una funzione é one-way significa dire che é difficile da invertire: dato y = f (x), il valore x non puó essere calcolato nella sua interezza in tempo polinomiale.
Non significa che nulla possa essere calcolato su x in tempo polinomiale! Per esempio, se g é one-way, allora f (x_1, x_2) = (x_1, g(x_2)); dove $|x_1| = |x_2|$ risulta one-way anche se rivela metá del suo input.

Un predicato **hard-core** hc : $\{0,1\}^* \to \{0,1\}$ di f ha la proprietà che hc(x) è difficile da calcolare con probabilità significativamente migliore di 1/2 dato f(x).

Definizione 8.4: Una funzione hc : $\{0,1\}^* \to \{0,1\}$ é un predicato hard-core di una funzione f se hc puó essere calcolato in tempo polinomiale e, per ogni A ppt, esiste una funzione trascurabile negl(n) tale che

$$Pr_{x \leftarrow \{0,1\}^n [A(1^n, f(x)) = hc(x)] \leq 1/2 + negl(n)} \tag{10.8}$$

Non è facile costruire predicati hard-core.

NOTA: con hard-core la difficoltà è insità all'interno della funzione. Un hc(x) è quindi un bit che la nostra funzione nasconde bene.

10.2.1 hc non è sempre un predicato per f

Sia g una funzione one-way e sia

$$f(x) = (g(x), \oplus_{i=1}^{n} x_i) \tag{10.9}$$

f é one-way ma non nasconde $\oplus_{i=1}^{n} x_i$, che é parte del suo output! Pertanto, hc non é un predicato hard-core per f.

Predicati hc banali

Aalcune funzioni hanno predicati hard-core banali. f$(x_1, ..., x_n) = x_1, ..., x_{n-1}$(butta via l'ultimo bit di input). É difficile determinare x_n dato f$(x_1, ..., x_n)$ poiché x_n è indipendente da f. La funzione f non é neanche one-way.

Domanda: esiste un predicato hard-core per qualsiasi funzione one-way? Non ha ancora risposta.

10.2.2 Teorema 8.5 Goldreich-Levin

Supponiamo che le funzioni (permutazioni) one-way esistano. Allora esiste una funzione (permutazione) one-way g ed un predicato hardcore hc di g.
Sia f una funzione one-way. Le funzioni g ed hc sono costruite come segue:

$$g(x, r) = (f(x), r), \text{ dove } x = x_1, ..., x_n, \ r = r_1, ..., r_n \tag{10.10}$$

$$hc(x, r) = \oplus_{i=1}^{n} x_i \cdot r_i \tag{10.11}$$

dove x_i (risposta r_i) è i-esimo bit di x (risposta r).
Se r é uniforme, allora hc(x, r) é l'xor di un sottoinsieme casuale di bit di x. Essenzialmente il teorema di Goldreich-Levin afferma che, se f é one-way, allora f (x) nasconde l'xor di un sottoinsieme casuale dei bit di x.

Dimostrazione

Si procede con una prova per passi.

Se esiste A PPT che calcola **sempre correttamente**

$$gl(x, r) = hc(x, r) \text{ data } g(x, r) = (f(x), r) \tag{10.12}$$

NOTA: g(x,r) è il risultato della permutazione.

per ogni n e per ogni x,r $\in \{0,1\}^n$, allora esiste A' tale che:

$$A'(1^n, f(x)) = x \quad \forall n \, e \, \forall x \in \{0,1\}^n \quad \textbf{(inverte la permutazione)} \tag{10.13}$$

Questo è il caso semplice.
$A'(1^n, y)$ calcola $x_i = A(y, e^i)$ per i = 1,..., n e dà in output $x_1, ..., x_n$.

Essendo e^i la stringa di n bit con un solo 1 in posizione i

$$x_i = A(F(x^*), e^i) = gl(x^*, e^i) = \oplus_{j=1}^{n} x_j^* \cdot e_j^i = x_j^* \tag{10.14}$$

Pertanto $x_i = x_i^*$, per tutti gli i ed A' dà in output l'inversa corretta x.

Nota: Il risultato generale si prova mostranto:

- Prima il caso in cui A calcola con prrobabilità $> 3/4$ il predicato gl(x,r)
- poi si raffina la prova mostrando che se A calcola con probabilità non trascurabile gl(x,r) allora A' inverte con probabilità non trascurabile.

10.2.3 PRG da predicati hardcore

Teorema 8.6: Sia f una permutazione one-way, e sia hc un predicato hardcore di f . Allora

$$G(s) = f(s)||hc(s) \tag{10.15}$$

é un generatore pseudocasuale con fattore di espansione l(n) = n + 1.
Questo funziona perché

- s uniforme ⇒ f (s) uniformemente distribuito, essendo f una permutazione, ovvero i primi n bit di G(s) sono uniformi;
- hc(s) predicato hard-core di f ⇒ hc(s) sembra casuale dato il valore di f (s)

Pertanto, l'intero output di G é pseudocasuale.

Per applicazioni abbiamo bisogno di PRG con espansione maggiore.

Teorema 8.7: Se esiste un PRG con fattore di espansione l(n) = n + 1 allora, per ogni polinomio poly (n), esiste un PRG con fattore di espansione poly (n).
Pertanto, PRG con fattori di espansione arbitraria possono essere costruiti a partire da una qualsiasi permutazione one-way.

Teorema 8.8: Se esiste un PRG con fattore di espansione l(n) = 2n, allora esiste una PRF.

Teorema 8.9: . Se esiste una PRF, allora esiste una PRP forte.

Discendono immediatamente i seguenti corollari:

Corollario 8.10: Assumendo l'esistenza di permutazioni one-way, esistono PRG con fattore di espansione polinomiale, PRF e PRP forti.

Corollario 8.11: Assumendo l'esistenza di permutazioni one-way, esistono schemi di cifratura a chiave privata CCA-sicuri e schemi di autenticazione di messaggi sicuri:

$$OWP \Rightarrow CCA - sicuro, MAC - sicuro \tag{10.16}$$

10.2.4 PRG con espansione polinomiale

Supponiamo di disporre di un PRG G:$\{0,1\}^n \to \{0,1\}^{n+1}$, usiamo G ripetutamente per costruire $\hat{G}$ come segue:

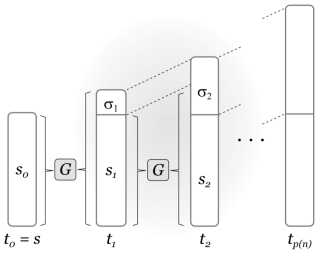
Costruiamo $\hat{G}\{0,1\}^n \to \{0,1\}^{n+2}$
Dato s $\in \{0,1\}^n$, calcola $t_1 = G(s)$.
Degli n + 1 bit di t_1, cioè $t_1^1 ... t_1^n t_1^{n+1}$, i primi n vengono usati come seme per G.
Vale a dire ponendo t = $t_1^n ... t_1^n$, calcola w_2 = G(t).
Agli n + 1 bit di w_2, cioè $w_2^1 ... w_2^{n+1}$, viene concatenato t_1^{n+1}
Pertanto:

$$\hat{G}(s) = w_2^1 ... w_2^{n+1} t_1^{n+1} \tag{10.17}$$

Indicando con p(n) il fattore di espansione voluto, il caso generale diventa:

1. Setta t_0 :=s.
2. Per i = 1, ..., p(n):
   - Siano s_{i-1} i primi n bit di t_{i-1} e denotiamo con σ_{i-1} i rimanenti i - 1 bits. (Quando i = 1, $s_0 = t_0$) e σ_0 è una stringa vuota)
   - Setta t_i :=G(s'_{i-1})|| σ_{i-1}.
3. Da in output $t_{p(n)}$

Ogni iterazione allunga "il suffisso" σ_i un bit. L'ultima iterazione aggiunge n + 1 bit "in testa".



Se usiamo come PRG di partenza il generatore

$$G(s) = f(s)||hc(s) \text{ f permutazione one-way e hc predicato hardcore} \tag{10.18}$$

allora la costruzione generale precedente dá luogo a

$$\hat{G} = f(f^{l-1}(s))||hc(f^{l-1}(s))||...||hc(f^i(s))||...||hc(s) \tag{10.19}$$

dove f^i, per i = 1, ... , l, e denota l'applicazione iterata i volte di f. Il fattore di espansione del generatore è l + n bits.

10.2.5 Costruzione di PRF

Mostriamo ora come costruire una PRF a partire da un PRG che raddoppia la lunghezza dell'input. Restringeremo l'attenzione a funzioni che preservano la lunghezza:

$$F_k : \{0,1\}^n \to \{0,1\}^n \tag{10.20}$$

Per familiarizzare con la costruzione, consideriamo un caso semplice. Sia n = 2. Disponiamo di G : $\{0,1\}^2 \to \{0,1\}^4$ e vogliamo realizzare:

$$F : \{0,1\}^2 x \{0,1\}^2 \to \{0,1\}^2 \tag{10.21}$$

Sia G(k) = $G_0(k)||G_1(k)$. Definiamo F come segue:

$$F_k(00) = G_0(G_0(k)), \qquad F_k(10) = G_1(G_0(k))$$
$$F_k(01) = G_0(G_1(k)), \qquad F_k(11) = G_1(G_1(k))$$

Intuitivamente F é pseudocasuale perché, se G é un PRG, allora $G_0(k)||G_1(k)$ è indistinguibile da $K_0||K_1$ uniforme e quindi abbiamo che

$$G_0(G_0(k))||G_0(G_1(k))||G_1(G_0(k))||G_1(G_1(k)) \tag{10.22}$$

è indistinguibile da

$$G_0(K_0)||G_0(K_1)||G_1(K_0)||G_1(K_1) \tag{10.23}$$

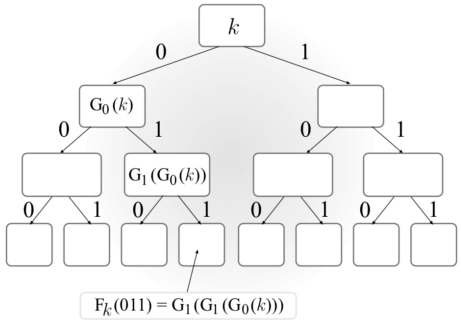
Se questo non fosse vero vuol dire che Adv sarebbe in grado di distinguere e quindi di rompere il PRG.

10.2.6 Costruzione di Goldreich-Goldwasser-Micali

COSTRUZIONE 7.21
Sia G un generatore pseudocasuale con fattore di espansione l(n) = 2n, e definiamo G_0, G_1 come fatto in precedenza. Per k $\in \{0,1\}^n$, definiamo una funzione $f_k : \{0,1\}^n \to \{0,1\}^n$ come segue:

$$F_k(x_1 x_2 ... x_n) = G_{x_n}(...(G_{x_2}(G_{x_1}(k)))...) \tag{10.24}$$

Graficamente abbiamo:



k è il seme / chiave pseudocasuale.
Si applica inizialmente G a k e otteniamo due parti una sinistra e una destra. In base al valore di x_i si procede in una delle due direzioni.
La taglia dell'albero é esponenziale in n.
Nonostante ció si noti che per calcolare F_k non occorre costruire tutto l'albero e memorizzarlo. Bastano n valutazioni di G.

Teorema 7.22 Se G é un PRG con fattore di espansione l(n) = 2n, allora la Costruzione 7.21 produce una funzione pseudocasuale.

É possibile costruire permutazioni pseudocasuali e permutazioni pseudocasuali forti usando una funzione pseudocasuale F e reti di Feistel a 3 round (PRP) e a 4 round (SPRP).

Capitolo 11

Teoria dei numeri

11.1 Introduzione

- Useremo il simbolo j j per denotare il valore assoluto di un intero, definito al modo seguente: |a| = a se a $>$ 0 e |a| = -a se a $<$ 0
- Dati due interi d e a, diremo che d divide a se esiste un terzo intero k tale che a = k $\cdot$ d: Per denotare tale evento, useremo la notazione d|a:

Teorema della divisione : Per qualsiasi intero a e qualunque intero positivo n, esistono due interi, q ed r, tali che a = q $\cdot$ n + r e 0 $\cdot$ r $<$ n. Gli interi q ed r sono unici.

Teorema Infinità dei numeri primi : Esistono infiniti numeri primi

11.1.1 Relazioni di equivalenza

Dato un insieme E, possiamo costruire l'insieme E x E, detto anche prodotto cartesiano, che contiene tutte le coppie possibili formate da elementi di E. Con il termine relazione indichiamo un qualsiasi sottoinsieme S dell'insieme E x E. Se gli elementi della relazione soddisfano le proprietà:

1. riflessiva: per ogni a $\in$ E, la coppia (a, a) appartiene a S
2. simmetrica: per ogni a, b $\in$ E, se la coppia (a, b) appartiene a S, allora anche la coppia (b, a) appartiene a S
3. transitiva: per ogni a, b, c $\in$ E, se le coppie (a, b) e (b, c) appartengono a S, allora anche la coppia (a, c) appartiene a S

allora si dice che la relazione S è una relazione di equivalenza. Indichiamo una relazione di ricorrenza con $\sim$.

Teorema: Per tutti gli a, b $\in$ E, se a $\in [b]$ allora $[a] = [b]$

11.1.2 Z partizionato in classi di congruenza

La nozione di congruenza di due interi modulo n definisce una relazione di equivalenza sull'insieme degli interi Z. Abbiamo infatti che l'insieme viene partizionato in classi di congruenza, a seconda del resto modulo n. Precisamente, la classe di congruenza contenente l'intero a è la classe di equivalenza: $[a]_n = \{a + k \cdot n : k \in Z\}$

L'insieme di tutte le classi di equivalenza si denota con:

$$Z_n = \{[a]_n : 0 \leq a \leq n-1\} \tag{11.1}$$

In conclusione, la relazione di congruenza modulo n, partiziona l'insieme:

$$Z = \{..., -2, -1, 0, 1, 2, ...\} \tag{11.2}$$

nelle classi che sono disgiunte e costituiscono un ricoprimento di Z. Ci sono inteessanti proprietà tra

$$\begin{aligned}
&\mathbb{Z}_0 = \{\mathbf{0}\} \\
&\mathbb{Z}_1 = \{\ldots, -n+1, \mathbf{1}, n+1, \ldots\} \\
&\mathbb{Z}_2 = \{\ldots, -n+2, \mathbf{2}, n+2, \ldots\} \\
&\ldots \\
&\mathbb{Z}_{n-1} = \{\ldots, -1, \mathbf{n-1}, 2 \cdot n - 1, \ldots\}
\end{aligned}$$

le classi:

- se sommo due qualsiasi elementi di due classi (che può anche essere la stessa), il risultato è sempre un elemento della stessa classe
- se moltiplico due qualsiasi elementi di due classi (che può anche essere la stessa), il risultato è sempre un elemento della stessa classe

11.1.3 Massimo comune divisore

Dati gli interi a, b e l'intero non negativo d, se d|a e d|b allora d è un divisore comune di a e di b.

- Se d|a e d|b; allora d|(a + b) e d|(a - b). Più in generale, d|(a · x + b · y), per ogni x, y ∈ Z.

- Se a|b, allora |a| · |b| oppure b = 0.

- Il massimo comune divisore di due interi a e b, non entrambi nulli, è il più grande divisore comune. Lo denoteremo con MCD oppurre gcd.

Teorema : Se a e b sono interi qualsiasi, non entrambi nulli, allora il MCD(a, b) è il più piccolo intero positivo dell'insieme: $\{a \cdot x + b \cdot y : x, y \in Z\}$

Alcune proprietà

Corollario 1 : Se a e b sono interi qualsiasi e l'intero d è tale che d|a e d|b, allora d|MCD(a; b)

Corollario 2 : Per tutti gli interi a e b e per qualsiasi n non negativo, risulta MCD (a *n, b * n) = n * MCD(a,b)

Definizione 1 :Due interi a e b sono detti relativamente primi se il loro unico divisore comune è 1, cioè MCD(a; b) = 1

Corollario 3 : Per tutti gli interi n, a e b, se n|a · b e MCD(a, n) = 1, allora n|b

Teorema : Per qualsiasi interi a, b e p, se MCD(a, p) = 1 e MCD(b, p) = 1, allora risulta MCD(a * b, p) = 1

Corollario 4 : Per tutti i primi p e tutti gli interi a e b, se p|a · b allora p|a o p|b.

Lemma 2 : Se a|n e b|n e MCD(a, b) = 1, allora a · b|n.

Teorema unicità fattorizzazione : Un intero composto a può essere scritto come prodotto:

$$a = p_1^{e_1} \cdot ... \cdot p_r^{e_r} \tag{11.3}$$

in modo unico, dove, per i = 1, ..., r, gli interi p_i sono numeri primi tali che $p_1 < ... < p_r$ e gli e_i sono interi positivi.

Teorema Per qualunque intero a non negativo e qualunque intero b positivo, risulta MCD(a,b) = MCD (b, a mod b).

Algoritmo Euclideo

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

Algoritmo Euclideo esteso

In una forma estesa l'algoritmo precedente, oltre a restituire il MCD(a, b), restituisce anche gli interi x e y tali che d = a * x + b * y. Precisamente:

```
Extended-Euclid(a, b)
    if b=0
        then return (a, 1, 0)
    (d', x', y')   Extended-Euclid(b, a mod b)
    (d, x, y) = (d', y', x' - ⌊ a=b ⌋ * y0)
    return (d, x, y)
```

Perché funziona?
Se b = 0, allora l'output (a, 1, 0) è sicuramente corretto, essendo a = 1 * a + 0 * b.
Se b ≠ 0, allora l'output (d', x', y'), con d' = MCD(b, a mod b), ci permette di scrivere

$$d' = b \cdot x' + (a \bmod b) \cdot y' \tag{11.4}$$

da cui discende che:

$$\begin{aligned} d = d' &= b \cdot x' + (a \bmod b) \cdot y' \\ &= b \cdot x' + (a - \lfloor a/b \rfloor \cdot b) \cdot y' \\ &= a \cdot y' + b \cdot (x' - \lfloor a/b \rfloor \cdot y') \\ &= a \cdot x + b \cdot y \end{aligned}$$

11.2 Gruppo

Un gruppo (G, ⊕), è un insieme G su cui è definita un'operazione binaria ⊕, per cui valgono le seguenti proprietà:

1. **Chiusura**: Per ogni a,b ∈ G, l'elemento a ⊕ b ∈ G;
2. **Identità o unità**: Esiste un e ∈ G tale che a ⊕ e = e ⊕ a = a, per ogni a ∈ G;
3. **Associatività**: Per ogni a,b,c ∈ G, risulta a ⊕ (b ⊕ c) = (a ⊕ b) ⊕ c;
4. **Reciproco**: Per ogni a ∈ G, esiste un unico b ∈ G tale che a ⊕ b = c;
5. **Commutatività**: Per ogni a,b ∈ G, risulta a ⊕ b = b ⊕ a

Quando sono soddisfatte tutte le proprietà il gruppo si dice abeliano.

Teorema 8 Legge della cancellazione Sia (G, $\oplus$) un gruppo e siano a, b e c suoi elementi. Allora:

1. se a $\oplus$ b = a $\oplus$ c, allora b = c;
2. se b $\oplus$ a = c $\oplus$ a, allora b = c

Se un gruppo (G, $\oplus$) soddisfa la condizione $|G| < \infty$, allora il gruppo si dice finito. Il numero di elementi di G costituisce l'ordine del gruppo.

11.2.1 Gruppi finiti additivi e moltiplicativi modulo n

Possiamo costruire gruppi finiti usando l'insieme Z_n. Infatti dati gli interi a, a', b e b' se:

$$a \equiv a' \ mod \ n$$
$$b \equiv b' \ mod \ n$$

allora

$$(a + b) \equiv (a' + b') \ mod \ n$$
$$(a \cdot b) \equiv (a' \cdot b') \ mod \ n$$

Pertanto, possiamo definire consistentemente la somma, il prodotto e la differenza su Z_n:

$$[a]_n + [b]_n = [a + b]_n$$
$$[a]_n \cdot [b]_n = [a \cdot b]_n$$

Teorema 9 $(Z_n, +_n)$ è un gruppo finito abeliano.

Lemma 3 Per qualsiasi a ed r interi ed n intero positivo, risulta

$$(a \cdot n + r) \cdot r \ mod \ n \tag{11.5}$$

Lemma 4 Siano a ed n interi tali che MCD(a, n) = 1. Risulta per ogni $k \in Z$

$$MCD(a + k \cdot n, n) = 1 \tag{11.6}$$

Sia $Z_n^* = \{[a]_n \in Z_n : MCD(a, n) = 1\}$ (insieme di interi relativamente primi con n)
Esempio: $Z_{15}^* = \{[a]_{15} \in Z_{15} : MCD(a, 15) = 1\} = \{1, 2, 4, 7, 8, 11, 13, 14\}$

Teorema 10 $(Z_n, \cdot_n)$ è un gruppo finito abeliano.

Per dimostrare l'esistenza dei reciproci si noti che $\forall a \ \ MCD(a, n) = 1 \Rightarrow$ Extended-Euclid (a,n) dà (d,x,y) = (1,x,y) $\Rightarrow$ = a · x + n · y $\Longleftrightarrow$ ax = 1 - ny questo implica che **ax mod n** = (1 - ny) mod n $\equiv$ mod n $\Rightarrow$ x è reciproco di a.

Quanti elementi ha Z_n^*

- Se n = p (primo) allora tutti gli a $\in \{1, ..., p-1\}$ tale che MCD(a, p = 1 $\Rightarrow |Z_n^*|$ = p- 1;
- Se n = p $\cdot q$ (primi) se $a \in \{1, ..., n-1\}$ allora non è relativamente primo, MCD(a, n) $\neq$ 1, poiché n non può dividere a; essendo n più grande di a, allora p|a o q|a:
  - Se gli elementi a divisibili da p sono: p, 2p, 3p, ..., (q-1) p. Allora p = q-1;

– Se gli elementi a divisibili da q sono: q, 2q, 3q, ..., (p-1) q. Allora q = p-1.

Gli elementi restanti, che non sono nè multipli di p nè multipli di q, sono in tutto:

$$\begin{aligned} n-1-(q-1)-(p-1) &= pq-1-(q-1)-(p-1) \\ &= pq-q-p+1 \\ &= q(p-1)-(p-1) \\ &= (p-1)-(q-1) \end{aligned}$$

Quindi abbiamo che $|Z_n^*|$ = (p - 1) (q - 1).

Quanti elementi ha Z_n^* con n = p^e Preso p primo ed e ≥ 1 intero e ragionando come prima se a $\in \{1, ..., n-1\}$ non è relativamente primo, allora p o qualche suo multiplo divide a.
I multipli di p tra 0 e $p^e - 1$ sono:

$$0 \cdot p, 1 \cdot p, ..., (p^{e-1} - 1) \cdot p \tag{11.7}$$

Ovvero esattamente p^{e-1}. Pertanto $|Z_n^*| = p^e - p^{e-1} = p^{e-1} \cdot (p-1)$.

Per un n generico abbiamo che siano $p_1, ..., pk$ primi distinti e siano $e_1, ..., e_k$ interi non negativi, sia n = $\prod_{i=1}^{k} p_i^{e_i}$ risulta

$$|Z_n^*| = \phi(n) = \prod_{i=1} p_i^{e_i - 1}(p_i - 1) \tag{11.8}$$

$\phi(n)$ è la funzione di Eulero.

Esempio: $|Z_{15}^*| = |Z_{3\cdot 5}^*|$ = (3-1) $\cdot$ (5-1) = $2 \cdot 4 = 8$.

11.2.2 Sottogruppi e proprietà

Sia (G, $\oplus$) un gruppo, e sia H un sottoinsieme di G. Se anche (H, $\oplus$) è un gruppo, allora (H, $\oplus$) è detto sottogruppo di (G, $\oplus$).

Per ogni a $\in G$ indichiamo con $a^{(m)}$ il risultato dell'applicazione iterata m - 1 volte dell'operazione $\oplus$ ad a. Si pone $a^{(0)}$ = e cioè l'elemento identità ed $a^{(1)}$ = a.

Teorema 11 : Se (G, $\oplus$) è un gruppo finito allora per ogni a $\in G$, esiste un m ≥ 1 tale che $a^{(m)}$ = e

Teorema 12 : Se (G, $\oplus$) è un gruppo finito ed H è un qualsiasi sottoinsieme di G tale che, per ogni a,b $\in$ H risulta a $\oplus b \in$ H, allora (H, $\oplus$) è un sottogruppo di (G, $\oplus$).

Teorema 13 Teorema di Lagrange : Se (G, $\oplus$) è un gruppo finito ed (H, $\oplus$) è un suo sottogruppo, allora |H| è un divisore di |G|.

Un sottogruppo (H, $\oplus$) di un gruppo (G, $\oplus$) è detto proprio se H $\neq$ G.

Corollario 5 Se (H, $\oplus$) è un sottogruppo proprio di un gruppo finito (G, $\oplus$), allora $|H| \leq |G|/2$.

11.2.3 Sottogruppi generati da un elemento

Il sottogruppo generato da a, denotato con <a> o con (<a>, $\oplus$) è definito da:

$$< a >= \{a^{(k)} : k \geq 1\} \tag{11.9}$$

e si dice che a genera il sottogruppo <a> o che a è generatore di <a>.

Poichè G è finito, <a> è un sottoinsieme finito di G, eventualmente comprendente tutto G.

L'associatività di $\oplus$ nel gruppo G implica che $a^{(i)} \oplus a^{(j)} = a^{(i+j)} \Rightarrow < a >$ è chiuso rispetto a $\oplus$.

Si definisce ordine di a (e useremo la notazione ord(a)) il minimo t > 0 tale che $a^{(t)} = e$

Teoerma 14 Per qualsiasi gruppo finito (G, $\oplus$) e per ogni a $\in$ G, l'ordine di a è uguale alla cardinalità del sottogruppo che genera, cioè ord(a) = |<a>|

Corollario 6 Sia a un elemento di ordine t. Risulta $a^{(i)} = a^{(j)}$ se e solo se i $\equiv$ j mod t.

Corollario 7 Se (G, $\oplus$) è un gruppo finito con unità e, per ogni a $\in$ G, risulta $a^{(|G|)} =$ e.

11.2.4 Gruppi ciclici

Se esiste un elemento a $\in$ G tale che <a> = G, il gruppo G si dice ciclico e l'elemento a si dice generatore o radice primitiva di G.
Informalmente: I gruppi ciclici sono gruppi per i quali esiste almeno un elemento a partire dal quale, applicando ripetutamente l'operazione del gruppo, è possibile generare tutti gli elementi del gruppo.

Esempio: Consideriamo il gruppo $(Z_{15}, +_{15})$. É un gruppo ciclico. L'intero 1 è un generatore. Infatti, relativamente ai multipli di 1, risulta:

$$15 \cdot 1 \equiv 0 \text{ mod } 15 \text{ e per } 0 < \text{i} < 15 \text{ risulta i} \cdot 1 = \text{i} \neq 0 \text{ mod } 15$$

Il gruppo $(Z_{15}, +_{15})$ ammette anche altri generatori comper per esempio 2:

$$<2> = \{0, 2, 4, 6, 8, 10, 12, 14, 1, 3, 5, 7, 9, 11, 13\}$$

D'altra parte non tutti gli elmenti di Z_{15} sono generatori per esempio 3:

$$<3> = \{0, 3, 6, 9, 12\}$$

NOTA: Quanto appena detto per il gruppo $(Z_p, +_p)$ in realtà vale per tutti i gruppi di ordine primo. Se riprendiamo il teorema di Lagrange quando (G, $\oplus$) è un gruppo finito di ordine primo p, questo non può avere sottogruppo propri perchè p non ammette divisori non banali e di conseguenza tutti gli elementi di G (eccetto l'identità) sono generatori di G.

11.3 Equazioni modulari

ax ≡ b mod n è detta equazione modulare. Bisogna trovare tutte le x che soddisfano l'equazione.

Sia <a> il sottogruppo di Z_n generato da a allora <a> = $\{a^{(x)} : x > 0\}$ = {a x mod n : x > 0 }. L'equazione ha soluzione se e solo se b ∈ <a> (multiplo di a).

Teorema 15 : Per ogni a, n interi positivi, se d = MCD(a, n), allora

$$< a >=< d >= \{0, d, 2d, ..., (n/d - 1)d\} \tag{11.10}$$

e quindi |<a>| = n/d.

Corollario 8 : Sia d = MCD(a, n). L'equazione lineare modulare ax ≡ b mod n è risolubile se e solo se d|b.

Corollario 9 : Sia d = MCD(a, n). L'equazione lineare modulare ax ≡ b mod n ha d soluzioni distinte mod n oppure non ha soluzioni.

Teorema 16 : Sia d = MCD(a, n) e si supponga che d = ax' + ny', per opportuni interi x' e y'. Se d|b allora una delle soluzioni dell'equazione ax ≡ b mod n è x_0 = x'(b/d) mod n.

Teorema 17 : Si supponga che ax ≡ b mod n abbia almeno una soluzione e sia essa x_0. Allora l'equazione ha esattamente d soluzioni distinte, date da

$$x_i = x_0 + i(n/d) mod n \quad per \ \ i = 1, ..., d - 1. \tag{11.11}$$

11.4 Teorema cinese del resto

Lemma 6 : Siano p ed n interi positivi tali che p|n. Allora, per ogni intero a, risulta

$$(a mod n) mod p = a mod p. \tag{11.12}$$

Lemma 7 : Siano $n_1, ..., n_r$ interi positivi relativamente primi e sia N = $n_1 n_2 ... n_r$ il loro prodotto. Per ogni x e per ogni a interi

$$x \equiv a (mod\, n_i), \quad per i = 1, ..., r \quad se\ e\ solo\ se \ \ x \equiv a (mod N) \tag{11.13}$$

Teorema 18 - Enunciato teorema cinese del resto: Siano $n_1, ..., n_r$ interi positivi relativamente primi e sia N = $n_1 n_2 ... n_r$il loro prodotto. Siano $a_1, ..., a_r$ interi. Il sistema di r equazioni

$$x \equiv a_i (mod\, n_i), \quad per i = 1, ..., r \tag{11.14}$$

ha un'unica soluzione modulo N: Precisamente, per i = 1, ..., r, indicando con $N_i = N/n_i$ e con $y_i = N_i^{-1}$ mod n_i, risulta

$$x = \sum_{i=1}^{r} a_i N_i y_i \ \ mod \ \ N \tag{11.15}$$

Teorema 19 - Enunciato alternativo teorema cinese del resto: Siano $n_1, ..., n_r$ interi positivi relativamente primi e sia N = $n_1 n_2 ... n_r$il loro prodotto. Si consideri la corrispondenza

$$a \in Z_N \longleftrightarrow (a_1, ..., a_r) \in Z_{n_1} x ... x Z_{n_r} \tag{11.16}$$

è biunivoca.
Questo implica che le operazioni sugli elementi di Z_N possono essere eseguite in modo equivalente sulle corrispondenti r-uple.

$$(a+b) \bmod N \leftrightarrow ((a_1+b_1) \bmod n_1, \ldots, (a_r+b_r) \bmod n_r)$$
$$(a-b) \bmod N \leftrightarrow ((a_1-b_1) \bmod n_1, \ldots, (a_r-b_r) \bmod n_r)$$
$$(a \cdot b) \bmod N \leftrightarrow ((a_1 \cdot b_1) \bmod n_1, \ldots, (a_r \cdot b_r) \bmod n_r)$$

Nel linguaggio dell'algebra astratta una corrispondenza biunivoca tra due gruppi (A, $\oplus_1$) e (B,$\oplus_2$) che preserva le operazioni si dice isomorfismo ed i gruppi si dicono isomorfi (i gruppi hanno la stessa forma).

Teorema 20 - Enunciato alternativo teorema cinese del resto: Siano $n_1, ..., n_r$ interi positivi relativamente primi e sia N = $n_1 n_2 ... n_r$il loro prodotto. I gruppo $(Z_N, +_N)$ e (Z_{n_1} x ... x Z_{n_r}, $+_{n_1}, ...,_{n_r}$) sono isomorfi con isomorfismo, dove a_i = a mod n_i, per i = 1, ... r, l'isomorfismo inverso è dato da

$$a = \sum_{i=1}^{r} a_i N_i y_i \ mod \ N \tag{11.17}$$

Identico discorso anche per (Z_N^*, $\cdot_N$) e ($Z_{n_1}^*$ x ... x $Z_{n_r}^*$, $\cdot_{n_1}, ...,_{n_r}$)

Esempio 1: Vogliamo calcolare 14 · 13 mod 15 in Z_{15}^*. Abbiamo n = 15 , p =5 e q = 3. Usando il teorema cinese del resto e quindi l'isomorfismo otteniamo $Z_{15}^* = Z_5^*$ x Z_3^*, possiamo quindi procedere come segue:

$$14 \Longleftrightarrow (4, 2)$$
$$13 \Longleftrightarrow (3, 1)$$

Calcoliamo

$$(4, 2) \cdot (3, 1) = ([4 \cdot 3 \ mod \ 5], [2 \cdot 1 \ mod \ 3]) = (2, 2) \tag{11.18}$$

Esempio 2: Vogliamo calcolae 18^{35} mod 35 in Z_{35}^*. Abbiamo n = 35 , p =5 e q = 7. Usando il teorema cinese del resto e quindi l'isomorfismo otteniamo $Z_{35}^* = Z_5^*$ x Z_7^*, possiamo quindi procedere come segue:

$$18 \longleftrightarrow (3, 4) \Rightarrow 18^{25} \ mod \ 35 \longrightarrow ([3^{25} \ mod \ 5], [4^{25} \ mod \ 6]) \tag{11.19}$$

Z_5^* è un gruppo di ordine $\phi(5) = 4$, risulta:

$$3^{25} mod 5 = 3^{25 \ mod \ 4} mod \ 5 = 3^1 \ mod \ 5 = 3 \tag{11.20}$$

Z_7^* è un gruppo di ordine $\phi(7) = 6$, risulta:

$$4^{25} mod 7 = 4^{25 \ mod \ 6} mod \ 7 = 4^1 \ mod \ 7 = 4 \tag{11.21}$$

Pertanto $(3, 4) \longleftrightarrow 18$, quindi $18^{25} \ mod \ 35 = 18 \ mod \ 35$

NOTA: Presi due gruppi $(G_1, \oplus_1)$ e $(G_2, \oplus_2)$, con isomorfismo f da G_1 a G_2 ed isomorfismo inverso f^{-1} da G_1 a G_2 entrambi calcolabili efficientemente. Allora per ogni coppia di elementi $g_1, g_2 \in G_1$. per calcolare g = $g_1 \oplus_1 g_2$ si può calcolare direttamente G_1 oppure:

1. Calcolare $h_1 = f(g_1)$ ed h_2 = f(g_2);
2. Calcolare h = $h_1 \oplus_2 h_2$;
3. Calcolare g = $f^{-1}(h)$

11.5 Proprietà generali dei gruppi

Teorema 21 : Sia G un gruppo finito di ordine n > 1. Sia e > 0 un intero e sia:

$$f_e : G \to G \text{ definita da } f_e(g) = g^e \tag{11.22}$$

Se il MCD(e,n) = 1 allora f_e è una permutazione su G. Inoltre indicando con d = e^{-1} mod n, allora la permutazione $f_d : G \to G$ definita da $f_d(g) = g^d$ è la permutazione inversa di f_e.

Teorema 22 :Sia (G, $\oplus$) un gruppo ciclico di ordine n, e sia g un generatore di G. La funzione:

$$f : Z_n \to G \text{ definda da } f(a) = g^a \tag{11.23}$$

è un isomorfismo tra $(Z_n, +_n)$ e (G, $\oplus$).
Pertanto segue che tutti i gruppi ciclici dello stesso ordine sono isomorfi.
NOTA: non è vero che sono la "stessa cosa" da un punto di vista computazionale, f infatti potrebbe essere calcolabile in modo efficiente mentre $f^{-1} : G \to Z_n$ NO!

11.5.1 Proprietà del gruppo Z_n^*

Teorema di Eulero : Per qualsiasi intero n > 1 e per ogni a $\in Z_n^*$, risulta:

$$a^{\phi(n)} \equiv 1 mod n \tag{11.24}$$

Teorema di Fermat : Per qualsiasi primo p e per ogni a $\in Z_p^*$, risulta:

$$a^{p-1} \equiv 1 mod p \tag{11.25}$$

Teorema di Niven e Zuckermann :I valori di n > 1 per cui Z_n^* è ciclico sono 2, 4, p^e e $2p^e$, per tutti i primi dispari p e per tutti gli interi positivi e.

Teorema del logaritmo discreto : Se g è un generatore di Z_n^*, allora l'equazione:

$$g^x \equiv g^y \text{ mod n vale se e solo se } x \equiv y \text{ mod } \phi(n) \tag{11.26}$$

Teorema radici quadrate dell'unità : Se p è primo dispari ed e è un intero positivo maggiore o uguale a 1, allora:

$$x^2 \equiv 1 mod p^e \tag{11.27}$$

ha solo due soluzioni, x=1 e x = -1.

Corollario 12 : Se esiste una radice quadrata non banale di 1 modulo n allora n è composto.

11.6 Generazione di numeri primi

Indichiamo con $\Pi(x)$ la funzione di distribuzione dei primi, che specifica il numero di primi minori o uguali a x, per ogni valore reale x. Per esempio: $\Pi(10) = 4$ (i primi sono 2,3,5, e 7).

Teorema dei numeri primi :

$$\lim_{x\to\infty} \frac{\Pi(x)}{x/lnx} = 1 \tag{11.28}$$

Al crescere di x, x/ln x è una stringa reagionevole per $\Pi(x) \Rightarrow 1/\ln x$ stima della probabilità che x scelto a caso sia primo.

Per capire se un numero è primo ci sono diversi modi, quello più semplice è effettuare di dividere n per ogni intero più piccolo fino a $\sqrt{n}$, questo ci costa $\sqrt{n}$ nel caso peggiore. Se consideriamo interi di β bit, con $\beta\lceil log(n+1)\rceil$, e indichiamo con poly($\beta$) il costo polinomiale di un ragionevole algoritmo per la divisione. Pertanto il tempo di esecuzione è dell'ordine poly(β) $\cdot 2^{\beta/2}$ nel numero di bit.
Per dimostrarlo si può usare il teorema di Fermat.

IDEA: Se n è primo allora $a^{n-1} \equiv 1$ mod n $\forall a \in Z_n^*$.
Se trovo una a tale che $a^{n-1} \not\equiv$ mod n allora n è composto (sicuramente), altrimenti si scommette che sia primo (probabile).
Questo test funziona abbastanza bene. Problema alcuni numeri composti sono scambiati per numeri primi, questi numeri sono i numeri di Carmichael.

Usiamo anche un secondo risultato, se n è primo $x^2 \equiv 1$ mod n ha solo radici banali $\Rightarrow$ una radice non banale, n è composto.

11.6.1 Test di Miller e Rabin (Primalità)

Si scelgono più valori di a uniformemente a caso e si verifica che

$$a^{n-1} \equiv 1 mod n \tag{11.29}$$

Mentre si calcola a^{n-1} controlla che non abbia generato radici non banali dell'unità.

Di seguito l'algoritmo per il calcolo dell'esponenziazione modulare.

```
Modular-exp(a, b, n)
    c <-- 0
    d <-- 1
    sia < b_k...b_0 > la rappresentazione binaria di b

    for i = k to 0
    c <-- 2 * c
    d <-- d * d mod n

    if b_i = 1
        then c <-- c + 1
             d <-- d * a mod n
    return d
```

NOTA:

- il termine c contiene il valore di b
- il termine d contiene il valore a^b mod n

- $< b_k ... b_0 >$ viene usato un numero di moltiplicazioni che dipende dalla lunghezza della rappresentazione binaria di n-1.

Di seguito l'algoritmo del test di Miller e Rabin. Si utilizzano due funzioni:

- Random(1, n-1): restituisce un numero a tale che 1 < a < n -1, scelto in modo casuale.
- Witness(a,n): un algoritmo che restituisce true se e solo se il valore di a è testimone della compostezza di n.

```
Miller-Rabin(n, s)
    for j = 1 to s
        a <-- Random(1, n-1)
        if Witness(a, n) = true
            then return composto
    return primo
```

Witness viene implementato attraverso semplice modifiche a Modular-exp:

```
Witness(a, n)
    sia < b_k...b_0 > la rappresentazione binaria di n-1
    d <-- 1
    for i = k to 0
        x <-- d
        d <-- d * d mod n
        if d = 1 e x ≠ 1 e x ≠ n-1
            then return true (x e' una radice quadrata non banale di 1 mod n )
        if b_i = 1
            then d <-- d * a mod n
    if d ≠ 1
        then return true (per il teorema di Fermat)
    return false
```

La procedura di Miller e Rabin è una ricerca probabilistica parametrizzata di una prova che n è composto. Sceglie s valori casuali e, se una di queste scelte risulta essere un testimone che n è composto, allora la procedura restituisce composto. D'altra parte, se nessun testimone viene trovato negli s tentativi, assume che ciò accada perché non vi sono testimoni e, di conseguenza, restituisce primo. Ma allora come sceglie s?

Teorema Testimoni : Se n è un intero dispari composto, allora il numero dei testimoni della sua compostezza è almeno (n-1)/2 (più della metà sono testimoni).

Teorema 30 : Per ogni intero dispari n > 2 ed ogni intero positivo s, la probabilità che il test di Miller e Rabin sbagli è al più 2^{-s}
DIM:
Usando il teorema dei testimoni ci permette di affermare che il test di Miller e Rabin, ad ogni iterazione del ciclo che sceglie un ttestimone con probabilità $\leq \frac{1}{2}$. Infatti in s iterazioni indipendenti la probabilità che non trovi testimoni è al più $(\frac{1}{2})^s = 2^{-s}$.

11.6.2 Gruppi ciclici di ordine primo

Sia p primo e sia Z_p^* un gruppo ciclico con ordine $\phi(p) = p - 1$. Sia β può essere scritto come $\beta = \alpha^i$ per qualche $0 \leq i \leq p - 2$.

Lemma 8 : L'ordine di $\beta = \alpha^i$ è dato da $\text{ord}_p(\beta)$ = (p-1)/ MCD(p-1,i).
OSSERVAZIONE: Se MCD(p-i,i) = 1, β è un generatore. Il numero di generatori di Z_P^* è $\phi(p-1)$. Se il gruppo G ha ordine q primo, tutti gli elementi eccetto l'unità sono generatori.

Come scegliamo un generatore per gruppi Z_p^*?

Teorema 31 : Sia p primo e sia $\alpha \in Z_p^*$. Allora α è un generatore se e solo se

$$\alpha^{(p-1)/q} \neq \ modp \tag{11.30}$$

per tutti i primi q tali che q|(p-1).

Come posso costruire un gruppo di ordine primo?

Teorema 32 : Sia p = rq + 1 con p e q primi. Allora

$$G \stackrel{\text{def}}{=} \{h^r \text{ mod p } | h \in Z_p^* \tag{11.31}$$

è un sottogruppo di Z_p^* di ordine q.
NOTA: Se r = 2 (es: p=2*q+1) G contiene i quadrati ddi tutti gli elementi di Z_p^*, che chiameremo residui quadratici.

- q sono i primi di Sophie German;
- p sono i primi sicuri (safe primes).

Il teorema precedente offre:

- un metodo per scegliere uniformemente a caso un elemento di G
- un metodo per verificare se a $\in Z_p^*$ appartiene anche a G.

Precisamente:

1. Generazione: si sceglie uniformemente a caso h $\in Z_p^*$ e si calcola h^r mod p. Poich* l'ordine di G è primo, ogni elemento di G è generatore di G;
2. Verifica: Si controlla che $h^q \equiv 1$ mod p. Perchè funziona?
   Sia g gen di Z_p^*, $h = g^i$.
   $h^q \equiv$1 mod p $\iff (g^i)^q \equiv 1$ mod p $\iff iq \equiv 0$ mod (p-1) $\iff iq \equiv 0$ mod rq $\iff iq \equiv k \cdot q$ $\iff eq|iq \iff r|i$. Ma se r|i allora i = cr per qualche costante $c \in Z$.Pertanto:

   $$h = g^i = g^{cr} = (g^c)^r \tag{11.32}$$

   Quindi h appartiene a G.

In conclusione, sapendo generare efficientemente numeri primi casuali, abbiamo trovato un modo semplice per costruire gruppi ciclici di ordine primo, sottogruppi di Z_p^*.

11.7 Problemi difficile e assunzioni crittografiche

11.7.1 Problema della fattorizzazione

Sia GenMod() un algoritmo PPT che genera du numeri primi p e q, di n bit e calcola N = pq. Sia A un Adv PPT. Definiamo l'esperimento $Factor_{A,GenMod(n)}$ come segue:

$Factor_{A,GenMod(n)}$:

1. C esegue GenMod(1^n) per ottenere (p, q, N)
2. A riceve da C il modulo N e dà a C i valori p', q' maggiori di 1
3. Se p'q' = N, allora C dà in output 1(A ha trovato i fattori di N); altrimenti 0.

Def: Relativamente a GenMod(1^n), il problema della fattorizzazione è difficile se per ogni algoritmo PPT A, esiste una funzione trascurabile tale che:

$$Pr[\text{Factor}_{A,GenMod}(n) = 1] \leq negl(n) \tag{11.33}$$

Algoritmi per la fattorizzazione

Esistono diversi algoritmi per la fattorizzazione:

- Polland's p-1: applicabile solo se p-1 ha fattori piccoli. Se tutti i suoi fattori sono minori di un certo B la complessità dell'algoritmo è O(B log B(log n)^2) + (log n)^3, ed ha probabilità di successo basse. Se B ha invece valore O((log N)^i) ha complessità di esecuzione esponenziale in n ma ha probabilità di successo alta. L'algoritmo cerca di calcolare un x tale che MCD(x, N) = p, i fattori piccoli di p-1 servono per calcolare x.
- Polland's Rho: cerca di trovare due interi x e x' ≡ mo p (non conosce p) in modo tale che: MCD(x-x', N) = p. La ricerca della coppia (x, x') è effettuata in modo efficiente O($2^{n/4}$).
- Quadratic Sieve: cerca di trovare due interi x e y tali che $x^2 \equiv y^2 mod N$ con $x \neq \pm y$ per calcolare p = MCD (x-y, N). La sua complessità è O($2^{\sqrt{2logn}}$).
- I due algoritmi più usati ed efficienti sono quelli del *Number field sieve* e *Elliptic curve factoring algorithm*, per valori molto grandi il primo è più efficiente.

11.7.2 Inversione della permutazione RSA

Sia GenRSA(1^n) un algoritmo PPT che genera due numeri primi p e q, di n bit e calcola N = pq e due interi e,d maggiori di 0 tale che MCD(e, ϕ (N)) = 1 e ed ≡ mod $\phi(N)$. Sia A un Adv PPT. Definiamo l'esperimento RSA-inv$_{A,GenRSA}$(n):

RSA-inv$_{A,GenRSA}$(n):

1. C esegue GenRSA(1^n) per ottenre (N, e, d)
2. C sceglie in modo uniforme $y \in Z_N^*$
3. A riceve da C i valori (N, e, y) e dà a C il valore $x \in Z_N^*$
4. Se $x^e \equiv y \bmod N$, allora C dà in output 1 (Adv inverte RSA); altrimenti 0.

Def : Relativamente a GenRSA(1^n), il problema dell'inversione della permutazione RSA è difficile se per ogni Adv A PPT esiste una funzione trascurabile tale che:

$$Pr[\text{RSA-inv}_{A,GenRSA}(n) = 1] \leq negl(n) \tag{11.34}$$

Teorema: Fattorizzazione di N dato d : Esiste un algoritmo PPT che ricevendo in input N = pq con p e q primi e gli interi e e d tale che ed $\equiv$ mod $\phi(n)$, dà in output la fattorizzazione di N, eccetto con probabilità trascurabile.

Idea : $x^2 \equiv 1$ mod N ha 4 radice quadrati dove 2 sono banali: ± 1. Quindi ogni radice q non banale può essere usata per calcolare efficientemente un fattore di N tramite: MCD(y $\pm$ N).

Pertanto indicando con GenMod il generatore in GenRSA, affiché il problema RSA sia difficile relativamente a GenRSA. il problema della fattorizzazione deve essere difficile relativamente a GenMod.

Factoring difficile $\Rightarrow$ RSA difficile?

Non è dato saperlo. Sappiamo soltanto che calcolare d a partire da (n, e) è tanto difficile quanto fattorizzare. Pertanto se l'unico modo per invertire RSA fosse tramite il calcolo preventivo di d, allora si potrebbe concludere che le due assunzioni siano equivalenti.

11.8 Problema del logaritmo discreto

Sia GenG(1^n) un algoritmo PPT per la generazione di gruppi ciclici (G, g, q) dove:

- G = insieme degli elementi;
- g = generatore;
- q = ordine del gruppo.

L'operatore $\oplus$ è efficientemente calcolabile, così come l'appartenenza di un elemento a G.
Se G è un gruppo ciclico con generatore g di ordine q allora $G = \{g^0, g^1, ..., g^{q-1}\}$ e per ogni $h \in G$ esiste un unico $x \in Z_q$ tale che $g^x = h$. Sia anche A un Adv PPT, definiamo l'esperimento $\text{Dlog}_{A,GenG}$(n) come segue:

$\text{Dlog}_{A,GenG}$(n)

1. C esegue GenG(1^n) per ottenere (G, g, q)
2. C sceglie in modo uniforme $h \in G$
3. A riceve da C i valori (G, g, q) ed h e dà a C il valore x $\in Z_q$
4. Se $g^x = h$ allora C da in output 1; altrimenti 0.

Il problema del logaritmo discreto consiste quindi nel calcolare x = $\log_g$h per un elemento h $\in$ G scelto uniformemente a caso.

Def: Relativamente a GenG(1^n), il prolema del logaritmo discreto + difficile se per ogni algoritmo PPT A esiste una funzione trascurabile tale che:

$$Pr[\text{Dlog}_{A,GenG}(n) = 1] \leq negl(n) \tag{11.35}$$

Algoritmo per il calcolo del logaritmo discreto

- Pohling-Hellman può essere applicato nel caso in cui l'ordine q del gruppo non sia primo, e la sua fattorizzazione sia nota o facilmente calcolabile. Il problema viene ridotto al calcolo del logaritmo discreto in sottogruppi del gruppo. La complessità non è maggiore della complessità di risolvere il problema in un sottogruppo di ordine q', con q' massimo tra gli ordini dei sottogruppi.

- Baby step-giant step di Shanks: calcola il logaritmo discreto in un gruppo di ordine q in tempo $O(\sqrt{q} \cdot polylog(q)) = O(2^{n/2} poly(n))$. L'algoritmo richiede la memorizzazione di $O(\sqrt{q})$. L'algoritmo divide la sequenza $\{g^0, g^1, ..., g^q\}$ in $\sqrt{q}$ sotto-intervalli. L'elemento h di cui si vuole calcolare x appartiene a uno dei sotto-intervalli. Calcolando $h \cdot g^1, ..., h \cdot g^t$, dove $g^1, ..., g^t$ rappresentando i piccoli passi, ed è come se shiftassimo h in uno degli intervalli fino a quando raggiunge l'estremo superiore. Per cui per qualche i $g \cdot g^i = g^{kt} \Rightarrow x = kt - i$ mod q.

- Rho di Pollard: simile all'algoritmo di shanks inserisce un trade off sul tempo a discapito dello spazio.

11.8.1 Problemi Diffie-Hellman

Sia GenG(1^n) un algoritmo ppt per la generazione di gruppi ciclici (G, g, q), dove G è la descrizione dell'insieme degli elementi del gruppo e di come sono rappresentati, g è un generatore del gruppo e q è il suo ordine. Supponiamo che valgano le stesse condizioni richieste per il problema del logaritmo discreto. Sia A un Adv PPT. Consideriamo l'esperimento $\text{CDH}_{A,GenG}$(n):

$\text{CDH}_{A,GenG}$(n)

1. C esegue GenG(1^n) per ottenere (G, g, q)
2. C sceglie in modo uniforme $x_1 \in Z_q$,$x_2 \in Z_q$ e calcola
$$h_1 = g^{x_1} \in G \text{ e } h_2 = g^{x_2} \in G \tag{11.36}$$
3. A riceve da C i valori (G, g, q) ed h_1, h_2 e dà a C il valore $h_3 \in G$
4. Se $h_3 = g^{x_1 x_2}$, allora C dà in output 1; altrimenti 0.

Def : Relativamente a GenG(1^n), il problema Diffie-Hellman computazionale è difficile se per ogni Adv A PPT, esiste una funzione trascurabile tale che

$$Pr[\text{CDH}_{A,GenG}(n) = 1] \leq negl(n) \tag{11.37}$$

Esiste anche una variante decisionale del problema, quest'ultima richiede di distinguerà tra una tripla generata da diffie-hellman: (h_1, h_2, h_3) e una tripla scelta uniformemente a caso: (h_1, h_2, h_3). Consideriamo l'esperimento $\text{DDH}_{A,GenG}$(n):

$\text{DDH}_{A,GenG}$(n)

1. C esegue GenG(1^n) per ottenere (G, g, q)
2. C sceglie in modo uniforme $x_1 \in Z_q$,$x_2 \in Z_q$ e calcola
$$h_1 = g^{x_1} \in G \text{ e } h_2 = g^{x_2} \in G \tag{11.38}$$
3. C sceglie un bit b in modo uniforme. Se b = 1, calcola $h_3 = g^{x_1 x_2}$. Altrimenti sceglie in modo uniforme $x_3 \in Z_q$ e calcola $h_3 = g^{x_3}$
4. A riceve da C i valori (G, g, q) ed (h_1, h_2, h_3) e dà a C un bit b'
5. Se b' = b, allora C dà in output 1; altrimenti 0.

Def : Relativamente a GenG(1^n), il problema Diffie-Hellman decisionale è difficile se per ogni Adv A PPT, esiste una funzione trascurabile tale che

$$Pr[\text{DDH}_{A,GenG}(n) = 1] \leq negl(n) \tag{11.39}$$

11.8.2 Relazioni tra i problemi

- DL facile $\Rightarrow$ CDH facile: Dati h_1 e h_2 calcolo $x_1 = logh_1$ e poi $h_2^{x_1} = h_3$. Non sappiamo se DL difficile $\Rightarrow$ CDH difficile.
- CDH facile $\Rightarrow$ DDH facile: Data la tripla (h_1, h_2, h_3), calcolo h'_3 da h_1, h_2 e controllo se $h'_3 \stackrel{?}{=} h_3$.
- CDH diffcile $\stackrel{?}{\Rightarrow}$ DDH facile: non sembra essere vero, ci sono dei gruppi in cui è stato provato che DL e CDH sono difficili ma in DDH sono facili (Z_p^* dove p è primo).

11.8.3 Gruppi ciclici di ordine primo

Ci sono varie classi di gruppi ciclici in cui i problemi Dlog e Diffe-Hellman sono ritenuti difficili. Una delle classi che i crittografia preferiscono è quella dei gruppi ciclici di ordine primo. Le motivazioni sono le seguenti:

- DL più difficile da risolvere con gli algoritmi noti;
- DDH sembra essere facile se l'ordine del gruppo q ha fattori primi piccoli;
- Trovare un generatore è immediato;
- Semplificano le prove di sicurezza dove è richiesto il calcolo di inversi moltiplicativi. Nei gruppi di ordine primi ogni esponente è invertibile;
- Usando DDH, se |G| = q, e q è primo, la distribuzione dei valori g^{x_1,x_2}, con $x_1, x_2 \in Z_q$ scelti uniformemente a caso, è quasi uguale.

11.8.4 Campi finiti

Un campo (S, $\oplus$, $\bigodot$) è una struttura algebrica costituita da un insieme di elementi S e due operazioni binarie, $\oplus$ e $\bigodot$, definite su di esso che godono delle seguenti proprietà:

1. (S, $\oplus$) è un gruppo abeliano
2. (S $\setminus e, \bigodot$), dove e è l'identità in S rispetto ad $\oplus$, è un gruppo abeliano
3. Vale la proprietà distributività. Per ogni a, b, c $\in$ S risulta:

$$(a \oplus b) \bigodot c = a \bigodot c \oplus a \bigodot b \tag{11.40}$$

Se risulta |S| $< \infty$, ovvero S ha un numero finito di elementi allora il campo si dice finito.

Definizione: Esistono campi finiti se e solo se n = p^e, con p primo, ed e ≤ 1.

Idea: Sia p primo e sia $Z_p[x]$ l'insieme di tutti i polinomi nell'incognita x con coefficienti in Z_p. I polinomi di $Z_p[x]$ possono essere sommati e moltiplicati.
$f(x), g(x) \in Z_p[x]$, diremo che f(x) divide g(x).
f(x) | g(x) se esiste q(x) $\in Z_p[x]$ tale che:

$$g(x) = q(x) \cdot f(x) \tag{11.41}$$

Il grado di è indicanto con deg(f).

Come per gli interi in generale abbiamo che:

$$g(x) = q(x) \cdot f(x) + r(x) \tag{11.42}$$

Con def(f) = n, deg(r) < n.
I polinomi q(x) ed r(x) sono unici. Indicheremo r(x) con g(x)(modf(x)): Se f(x), g(x) ed h(x) sono polinomi in $Z_p[x]$ e deg(f) = n ≥ 1; diremo che g(x) è congruente ad h(x) modulo f(x) e scriveremo:

$$g(x) \equiv h(x)(mod\ f(x)) \tag{11.43}$$

se g(x)(modf(x)) = h(x)(modf(x)).
Ogni polinomio in $Z_p[x]$, è congruente ad un unico polinomio di grado al più n-1. In altre parole, così come fatto per gli interi Z, partizionati in classi di Z_n, stiamo partizionando $Z_p[x]$ in classi di congruenza rispetto al polinomio f(x) prescelto. Indicheremo l'insieme delle classi di congruenza con $Z_p[x]$=f(x).

Campo irriducibile

Un polinomio f(x) si dice irriducibile se non esistono due polinomi $f_1(x)$ ed $f_2(x)$ tale che:

$$f(x) = f_1(x) \cdot f_2(x) \tag{11.44}$$

dove def(f_1) > 0, deg(f_2) > 0.
$(Z_p[x]/f(x), +_{poly}, *_{poly})$ è un campo se e solo se f(x) è irriducibile.

Si può dimostrare che esiste almeno un polinomio irriducibile per ogni grado n in $Z_p[x] \Rightarrow \exists$ un campo finito con p^n elementi per tutti i primi p, n ≥ 1. Si può dimostrare che i campi costruiti a partire da due polinomi irriducibili diversi risultano isomormi $\Rightarrow$ non esiste un campo con p^n elementi.

Sottogruppi di campi finiti

Il problema DL è ritenuto difficile anche nel gruppo moltiplicativo di un campo finito con caratteristica grande:

$$F_{pn}^{*} \text{ è un gruppo ciclico di ordine } p^n - 1 \tag{11.45}$$

Se q è un fattore primo grande di $P^n - 1$ allora F_{pn}^{*} ha un sottogruppo ciclico di ordine q.

11.9 Teoria dei numeri per generare funzioni one-way

11.9.1 assunzione della fattorizzazione ⇒ una funzione one-way

L'algoritmo di generazione Gen(1^n) usa un numero di bit polinomiale per dare in output la coppia (N, p, q) dove N = p q. Più precisamente Gen(1^n) → (N, p, q).
Idea: Possiamo definire una funzione f_{Gen} che usa il suo input x come "random bits" per l'esecuzione di Gen(1^n). Modifichiamo allora Gen(1^n) portando la randomness fuori, es: Gen(1^n, x).

Algoritmo per calcolare f_{Gen}

- **Input:** stringa x di n bit
- **Output:**intero N di n bit
- Calcola (N,p,q) ← Gen(1^n,x), x sono i random bit fissati forniti come input a Gen; return N.

$f_{Gen} : \{0,1\}^n \to \{0,1\}^n$ rispetta one-way, osserviamo che:

- i moduli N restituiti di $f_{Gen}(x)$ con x$\in \{0,1\}^n$ uniforme
- i modulo N restituiti da Gen(1^n)

sono distribuiti identicamente.
Se i moduli generati da Gen(1^n) sono difficili da fattorizzare, così risultano quelli generati da f_{Gen}(x).
Discende che trovare una qualsiasi preimmagine x' di N rispetto a f_{Gen} deve essere difficile:

$$f_{Gen}(x') = f_{Gen}(x) = N. \tag{11.46}$$

Deve essere difficile perchè Gen(1^n) → (N, p, q) eseguito in tempo polinomiale implica l'aver fattorizzato N in tempo polinomiale.

11.9.2 assunzione RSA ⇒ una funzione one-way

Definiamo l'esperimento come Π = (Gen, Samp, f) come segue:

1. Gen(1^n) -> I : |I| ≥ n, $f_1 : D_I \leftarrow D_I$
2. Samp : x ∈ D_I
3. l'algoritmo di valutazione di f su input I dà y = $f_i(x)$.

Definiamo quindi a partire da Π GenRSA:

1. Gen : su input 1^n, esegui GenRSA(1^n) ← (N,e, d) e da in output I) (N, e). $D_I = Z_N^*$
2. Samp : su input I =(N, e) scegli uniformemente x ∈ Z_N^* fino a quando gcd(x, N) = 1.
3. f: su input I = (N; e) ed x inZ_N^* da in output x^e mod N.

È immediato constatare che se il problema RSA è difficile relativamente a GenRSA, allora la famiglia di permutazioni proposta è one-way.
Procedendo in un modo simile si può costruire una famiglia di permutazioni one-way nella difficoltà del problema DL in Z_p^*

- Gen(1^n) ← $(p, q, p-1)$
- Samp ← $x \in Z_p^*$
- f: g^x mod p

11.9.3 La teoria dei numeri permette di realizzare funzioni hash collision-resistant.

Utilizzando la teoria dei numeri è possibile realizzare una funzione hash resistente a collisione basata sulla difficoltà del problema del logaritmo discreto (DL). Quindi, qualunque algoritmo efficiente che sia in grado di calcolare collisioni per le funzioni hash potrebbe essere utilizzato per il risolvere il problema DL.
Supponiamo che G sia un algoritmo PPT per la generazione di gruppi in cui restituisce triple (G,q,g) dove G è la rappresentazione di un gruppo ciclico, q il suo ordine e g un generatore che permette di generare tutti gli elementi del gruppo. Consideriamo una funzione hash (Gen,H) a lunghezza fissata in cui:

- **Gen**: su input 1^n genera (G, q, g). Seleziona $h \in G$ in modo uniforme e restituisce s=(G,g,q,h), una sequenza che rappresenta la descrizione del gruppo. s è la chiave che permette di individuare la specifica funzione hash all'interno della famiglia;
- **H**: data una chiave s = (G,g,q,h) e un input $(x_1, x_2) \in Z_q x Z_q$, restituisce l'hash della stringa x = $x_1||x_2 \rightarrow H^s(x_1, x_2) = g^{x_1} h^{h_2} \in G$.

Nota: Il gruppo G e la funzione H possono essere calcolate in tempo polinomiale perché Gen(1^n) e $H^s(x_1, x_2)$ è polinomiale.

H^s prende in input una stringa di 2(n-1) bit. La funzione hash G^s può essere definita come una funzione che comprime la stringa di input, quando gli elementi del gruppo G possono essere rappresentati con

meno di 2(n-1) bit:

$$H^s : \{0,1\}^{2(n-1)} \leftarrow \{0,1\}^l, \text{ con l} < 2\text{(n-1)} \tag{11.47}$$

La scelta del gruppo deve essere una scelta tale che gli elementi del gruppo possono essere rappresentati con l bit $<$ 2(n-1).

Possiamo dimostrare che tale funzione hash è una funzione resistente a collisione se nel gruppo generato il problema del logaritmo discreto è difficile, come evidenziato dal seguente teorema:
Teorema: Se il problema DL è difficile relativamente a G, allora H è una funzione hash resistente a collisione.

Dimostrazione: Supponiamo che A PPT cerca di trovare collisioni per la funzione, quindi gioca all'interno dell'esperimento Hash-coll$_{A,\Pi}$ (n). Questo esperimento coinvolge un avversario a cui viene data la descrizione della funzione e deve provare a trovare due elementi x e x' che generino una collisione. Supponiamo che A riesca a vincere con probabilità $\epsilon(n)$:

$$Pr[\text{Hash-coll}_{A,\Pi}(n) = 1] = \epsilon(n) \tag{11.48}$$

Possiamo usare A per costruire A' che risolve il problema del logaritmo discreto con la stessa probabilità di successo $\epsilon(n)$.
L'algoritmo A' riceve un'istanza del problema e deve restituire il logaritmo discreto. In maniera più specifica opera come segue:

1. A' riceve in input la descrizione del gruppo ciclico, data dalla terna (G,g,a) e h, elemento di cui calcolare il DL;

2. Utilizza l'input per definire una chiave s=<G,g,q,h> per individuare una specifica funzione della famiglia;

3. Esegue l'algoritmo A(s) fornendo come input la chiave s così da individuare delle collisioni. Quindi A produce x e x', che sono la sua scommessa nel tentativo di trovare una collisione, e li restituisce all'avversario A';

4. A' controlla se i due valori sono diversi $x \neq x'$ e producono la stessa immagine $H^s(x) = H^s(x')$. In caso affermativo, vuol dire che siamo di fronte ad una collisione quindi A' controlla se h=1 restituisce 0 perché g^0=1, altrimenti ($h \neq 1$) scompone:

   - x come (x_1, x_2) con $x_1, x_2 \in X_q$
   - x' come (x'_1, x'_2) con $x'_1, x'_2 \in Z_q$

   e ritorna $[(x_1 - x'_1)(x_2 - x'_2)^{-}1 \bmod q]$.

Se esiste un algoritmo A in grado di produrre collisioni, A' sarà in grado di risolvere il problema DL per una serie di motivi:

1. L'algoritmo A' esegue in tempo polinomiale perché la parte più importante della computazione viene da A. Dato che abbiamo assunto che questo sia un algoritmo PPT e che i passi che vengono effettuati da A' sono elementari, anche A' dovrebbe essere PPT;

2. L'algoritmo A, quando viene mandato in esecuzione da A', non si accorge esattamente di nulla, ovvero ha l'impressione di stare seguendo l'esperimento Hash-coll$_{A,\Pi}$ perché il valore s che riceve non è stato costruito da A' in accordo a qualche distribuzione di probabilità, ma viene costruito esattamente come accade nell'esperimento reale in quanto il gruppo viene generato dallo stesso algoritmo di generazione utilizzato in quello reale. Di conseguenza, il valore s dato ad A è distribuito esattamente come nell'esperimento Hash-coll$_{A,\Pi}$ per lo stesso valore del parametro

di sicurezza n. Questo implica che la probabilità in cui viene prodotta una collisione all'interno dell'esperimento reale è esattamente la stessa probabilità con cui A produce una collisione quando viene eseguito all'interno dell'esperimento simulato.

Supponendo che A sia stato in grado di produrre una collisione, avremo tale comportamento:

$$\begin{aligned} H^s(x_1, x_2) = H^s(x_1', x_2') &\longleftrightarrow g^{x_1} h^{x_2} = g^{x_1'} h^{x_2'} \\ &\longleftrightarrow (g^{x_2} h^{x_2} g^{-x_1'} h^{-x_2}) = (g^{x_2'} h^{x_2'} g^{-x_1'} h^{-x_2}) \\ &\longleftrightarrow g^{x_1 - x_1'} = h^{x_2' - x_2} \end{aligned}$$

Si nota che $x_2' - x_2 \neq 0$ mod q, altrimenti sarebbe che $g^{x_1 - x_1'} = h^0 = 1 = g^0 \leftarrow x_1 = x_1'$ e $x_2 = x_2'$. Questo significa che siamo di fronte al caso in cui x=x' e quindi non siamo più di fronte al caso della collisione che abbiamo supposto. Se siamo di fronte al caso di una collisione, i valori x e x' devono essere diversi. Da qui il fatto che $x_2' - x_2 \neq 0$ mod q. Poiché q è primo esiste l'inverso moltiplicativo di $x_2' - x_2 \neq 0$ mod q:

$$\begin{aligned} (x_2' - x_2)^{-1} \text{ mod q} \leftarrow g^{(x_1 - x_1')(x_2 - x_2')^{-1}} &= h^{(x_1 - x_1')(x_2 - x_2)^{-1}} = \\ &= h^1 = h \leftarrow log_2 h = (x_1 - x_1')(x_2' - x_2)^{-1} \end{aligned}$$

Di conseguenza, tale funzione hash definita per una opportuna scelta del gruppo, vale a dire un gruppo in cui il problema DL è difficile, e la rappresentazione degli elementi soddisfa la condizione che permette alla funzione di comprimere l'input realizza una funzione hash resistente a collisione.

11.10 Introduzione alle curve ellittiche (ECC) su campi finiti

Una curva ellittica è una curva definita da un'equazione in due incognite del tipo:

$$y^2 = x^3 + ax + b \tag{11.49}$$

11.10.1 ECC sui reali

Def: Siano a,b ∈ R tale che $4a^3 + 27b^2 \neq 0$. Una curva ellittica non singolare è l'insieme E di soluzioni (x,y) ∈ RxR dell'equazione $x^3 + ax + b$, più un punto speciale θ (detto punto all'infinito).

Su quest'insieme proviamo a definire l'operazione della somma. Abbiamo P = (x_1, y_1) e Q = (x_2, y_2). Consideriamo 3 casi:

1. $x_1 \neq x_2$
2. $x_1 = x_2$ e $y_1 = -y_2$
3. $x_1 = x_2$ e $y_1 = y_2$

Caso 1: Sia L una retta che passa attraverso P e Q. La retta L interseca la curva E in P e Q e in un ulteriore punto R'. Riflettendo R' rispetto all'asse delle ascisse si ottiene un punto R. Definiamo P+Q=R.
Per trovare le coordinate del punto R procediamo come segue:

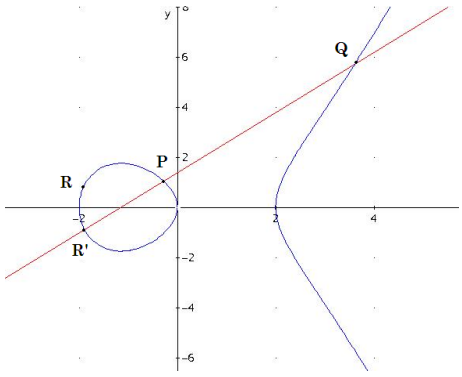
1. l'equazione della retta L è data da y = $\lambda x + v$, dove:
   - $\lambda = \frac{y_2 - y_1}{x_2 - x_1}$
   - v = $y_1 - \lambda x_1 = y_2 - \lambda x_2$

2. Per poter trovare i punti in cui L ed E si intersecano sostituiamo y = $\lambda x + v$ in E, ottenendo:

$$x^3 - \lambda^2 x^2 + x(a - 2\lambda v) + b - v^2 = 0 \tag{11.50}$$

L'equazione risultante è un'equazione cubica sui reali avente due radici reali, e quindi anche la terza sarà reale. La somma delle tre radici deve essere uguale a λ^2, cioè l'opposto del coefficiente del termine quadratico e quindi avremo:

$$x_3 = \lambda^2 - x_1 - x_2 \tag{11.51}$$



Quindi x_3 è l'ascissa del punto R'. Indichiamo con -y_3 l'ordinata. Un modo facile per calcolare y_3 è usare il coefficiente angolare δ della retta L. Infatti è determinato da ogni coppia di punti della retta L : $y_3 = \lambda(x_1 - x_3) - y_1$.
Una volta trovato y_3 abbiamo trovato il punto R.

Caso 2: Definiamo (x,y) + (x,-y) = θ per tutti i punti (x,y) $\in$ E. Il punto θ è l'elemento unitario (identità), cioè P + θ = θ + P = P, per ogni P $\in$ E. Inoltre, per ogni punto (x,y) $\in$ E, il punto (x,-y) funge da inverso.

Caso 3: In questo caso stiamo sommando P a se stesso. Supponiamo $y_1 \neq 0$ altrimenti vale il caso precedente. E' possibile dimostrare che valgono le stesse regole di calcolo del caso 1. Varia soltanto il calcolo di λ che risulta uguale a:

$$\lambda = \frac{3{x_1}^2 + a}{2y_1} \tag{11.52}$$

11.10.2 ECC su campi finiti

Si consideri il campo finito $(Z_p, +, \cdot)$ con p primo.

Def: Sia p > 3 primo. La ECC $y^2 = x^3 + ax + b$ su Z_p è l'insieme delle soluzioni (x,y) e Z_px Z_p alla congruenza:

$$y^2 = x^3 + ax + b \text{ mod p} \tag{11.53}$$

dove a,b e Z_p sono constanti tale che: $4a^3 + 27b^2 \neq 0$ (mod p), con un punto speciale θ, detto punto all'infinito.

L'operazione di somma $+_E$ tra punti di E si definisce come segue:
Siano P = (x_1, y_1) e Q = (x_2, y_2) punti di E. Se $x_2 = x_1$ e $y_2 = -y_1$ allora P $+_E Q = \theta$. Altrimenti abbiamo che:

$x_3 = \lambda^2 - x_1 - x_2$

$y_3 = \lambda(x_1 - x_3) - y_1$

dove:

$$\lambda = \begin{cases} (y_2 - y_1)(x_2 - x_1)^{-1} & \text{se P} \neq \text{Q} \\ (3{x_1}^2 + a)(2y_1)^{-1} & \text{se P} = \text{Q} \end{cases} \tag{11.54}$$

Risulta $\forall P \in E$, e quindi P $+_E$ Q = Q $+_E$P.

Per vedere se un valore a è un residuo quadratico si può usare il criterio di Eulero.

Criterio di Eulero : sia p un primo dispari allora a è un residuo quadratico se e solo se $a^{(p-1)/2 \equiv \text{ mod p}}$.

11.10.3 Calcolo delle potenze

Teorema: se l'ordine di un gruppo è primo, il gruppo è ciclico e ogni elemento (escluso θ) di G è un generatore.

Per calcolare le potenze basta fare a + a, dove a è una coordinata di un punto e + è la somma modulo p.
Osservazione: vogliamo che il DLP sia difficile.

Teorema di Hasse: sia E una curva ellittica definita su Z_p(p > 3 e primo). Il numero di punti di E, indicato con #E risulta:

$$p + 1 - 2\sqrt{p} \leq \#E \leq p + 1 + 2\sqrt{p} \tag{11.55}$$

Quindi possiamo dire che su E ci saranno circa p punti. Il numero preciso può essere calcolato tramite un algoritmo efficiente (algoritmo di Schoof).
Se #E è primo o è un prodotto di primi distinti, allora è ciclico.

11.10.4 Point compress - point decompress

Dato x ci sono due valori di y tali che $y^2 = x^3 + ax + b$ mod p. Questi due valori sono uno il negato dell'altro. Poiché p è primo: uno sarà pari e l'altro dispari, possibile definire un punto P = (x,y) specificando il valore di x e usare un bit per indicare la parità.

Formalemtne la deminiamo come segue:

- **Point compress**: Point-compress(p) = (x,y mod 2);
- **Point decompress**: restituisce il punto P(x,y):

```
Point_decompress(x,i):
   z <- x^3 + ax + b mod p
   se z non e' un residuo quadratico
       return "fallito"
   altrimenti
       y <- √z mod p
       se y ≡ i (mod 2) return (x,y)
       altrimenti
           return(x, p-y)
```

L'operazione principale nella ECC è kP calcolare multipli di un punto P. Si può fare questa operazione in modo efficiente:

```
SQUARE AND MULTIPLY (x,c,n):
    z <--1
    for i <- l-1 downto 0 do:
        z <- z^2 mod n
        if c_i = 1 then z <- (zx) mod n

    return z
```

Nota: su una curva ellittica l'inverso additivo di un punto è facile da calcolare e.g. P = (x,y) -> (x,-y)

11.10.5 Square and multiply

Sia c un intero. Una rappresentazione binaria con segno di c è un'equazione della forma:

$$\sum_{i=0}^{l-1} c_i \cdot 2^i \quad \text{dove } c_i \text{ e } \{0, -1\} \text{ per ogni i} \tag{11.56}$$

Esempio:
11 in binario è (0,1,0,1,1) ma può essere anche rappresentato come (1,0,-1,0,-1).

11.10.6 Double and (add or substract)

Sia P un punto di ordine n di una curva ellittica Data la rappresentazione binaria con segno $(c_{l-1}, .., c_0)$ di un intero $0 \leq c \leq n-1$ è possibile calcolare il multiplo cP attraverso una serie di raddoppi (2P), addizioni e sottrazioni, usando il seguente algoritmo:

```
DOUBLE AND (ADD OR SUBSTRACT)(P, (c_{l-1}, ..., c_0), n):
    Q <-- 0
    for i <-- l-1 downto 0 do:
        Q <-- 2Q
        if c_i = 1then Q <-- Q +P
        else if c_i = -1 then Q z-- Q-P

    return Q
```

Rappresentazione NAF (Non Adiacent Form)

Una rappresentazione binaria con segno $(c_{l-1}, ..., c_0)$ di un intero c è detta in forma non adiacente (NAF) se non ci sono due ci consecutivi diversi da zero.

L'idea alla base della trasformazione è di sostituire le sottostringhe di forma (0,1,1, ... , 1) nella rappresentazione binaria tramite la sottostringa (1,0, ...,0,-1).

L'uso di questa forma è molto utile nell'algoritmo di *double and (add or substract).* Infatti si passa da l raddoppi e l/2 add o sub , a l raddoppi e l/3 add o sub.

Assumendo che un'operazione di raddoppio richieda circa lo stesso tempo di una add o substract il rapporto tra i tempi medi di soluzione tra i due algoritmi è:

$$\frac{l + \frac{1}{2}}{l + \frac{1}{3}} = \frac{\frac{3}{2}l}{\frac{4}{3}l} = \frac{3}{2}l \cdot \frac{3}{4}l = \frac{9}{8} = 1.125 \tag{11.57}$$

Si ha un guadagno dell'11 % circa.

Capitolo 12

Gestione delle chiavi simmetriche e chiave chiave pubblica

La crittografia a chiave privata è molto utile per rendere sicuro un canale insicuro, ma c'è un problema come scambiarsi la chiave privata. Esistono vari modi per scambiarsi una chiave, un modo possibile sarebbe quello di affidare la chiave a qualcuno e aspettare che venga consegnata, ma questo approccio è possibile per le grandi aziende ma non per le persone comuni.
Un possibile modo per scambiare le chiavi in un azienda è dare ad un controllore tutte chiavi private dei dipendenti, e fare in modo che quest'ultimo cifri e decifri i messaggi. Problema il controllore sa sempre il contenuto dei messaggi. Se il controllore è fidato però in contesti piccoli (un'azienda) può essere applicato tramite l'uso di smart-card. In contesti grandi ad esempio internet no conviene usarlo.
Una soluzione per internet è avere un centro di distribuzioni (KDC) di chiave online:

- Ogni utente condivide una chiave con il KDC, generata e consegnata all'utente (es: il primo giorno di lavoro di un dipendente);
- Quando un utente A vuole comunicare con un utente B, viene inviata una richiesta al KDC autenticata con la chiave precedentemente condivisa per stabilire una connessione sicura;
- Il KDC sceglie una chiave segreta detta chiave di sessione e la invia ad A cifrata con la sua chiave condivisa e a B cifrata con la sua chiave condivisa;
- A e B ottenuta la chiave di sessione possono comunicare;
- Al termine della comunicazione viene cancellata la chiave

I vantaggi di questo approccio sono che ogni utente ha solo una chiave, il KDC ne memorizza molte può essere installato in un luogo sicuro. Un nuovo utente comporta l'aggiunta di soltanto una nuova chiave. La memorizzazione solitamente ha memoria lineare.
Come svantaggio ha che attaccato il KDC si compromette l'intero sistema. Chi ha accesso al KDC può decifrare tutte le comunicazioni. Il KDC può essere sovraccaricato di comunicazioni rallentandolo o portandolo a più fallimenti.

Alcune soluzioni possibili sono:

- Replicare il KDC all'interno del sistema: funziona ma introduce più punti di attacco, nascono probelmi di condivisione dei dati nel momento di aggiunta o rimozione di un utente;
- Rendere distribuito il KDC (DKDC): in cui nessun server da solo conosce le chiavi, si riduce la possibilità di disservizi, un utente per ottenere una chiave di sessione deve contattare un sottoinsieme dei server.

12.0.1 Protocollo di Needham-Schroeder

Il funzionamento è il seguente:

- Quando a vuole comunicare con B, il KDC non invia la chiave di sessione ad entrambi;
- Il centro manderà ad A due chiavi di sessione cifrate con la chiave condivisa di A e B.
- Quando A vuole inviare un messaggio invia sia il messaggio che la chiave di sessione cifrata con quella di B.

Si è pensato di realizzare questo approccio così se uno dei due utenti è offline la comunicazione può sempre avvenire. Un esempio di questo sistema è KERBEROS.

PROBLEMA: Due utenti su reti diverse che non si sono mai incontrati.

12.1 Scambio di chiavi Diffie-Hellman

Due utenti C e M non condividono nessuna chiave comune, ma decidono soltanto di generarla di un determinato numero di bit su un protocollo Π. Quello che vogliamo è che Π sia corretto cioè:

$$K_C = K_M = K \tag{12.1}$$

In più le due chiavi devono essere totalmente impredicibili ad un Adv (cenzino) che ascolta la comunicazione.

12.1.1 Esperimento Scambio di chiavi KE

Un protocollo di sicurezza di scambio chiavi è sicuro se la chiave che Alice e Bob danno in output è totalmente impredicibile da un avversario che ascolta la comunicazione. Quindi, l'avversario non deve essere in grado di calcolare la chiave che viene utilizzata dalle parti oneste. Questo può essere formalizzata richiedendo che l'avversario, dopo aver ascoltato una esecuzione, non è grado di distinguere la chiave K, generata dal protocollo Π, da una chiave scelta uniformemente a caso di lunghezza n.
Questa formalizzazione è più forte della richiesta di impredicibilità perché se non si è in grado di distinguere, non si è in grado di calcolare informazioni parziali sul valore. La formalizzazione viene fatta tramite un esperimento in cui denotiamo:

$KE^{eav}_{A,\Pi}$:

1. Le due parti eseguono il protocollo Π su input 1^n. Sia trans il transicript della comunicazione e sia K la chiave che danno in output;
2. Il challenger prende il trans e seglie un bit $b \leftarrow \{0,1\}$. Se b = 0, si pone $\hat{K} := K$(valore che hanno realmente calcolato), altrimenti se b = 1 pone $\hat{K} \leftarrow \{0,1\}^n$ uniformemente;
3. L'avversario A riceve trans dell'esecuzione reale tra Alice e Bob e $\hat{K}$. Quindi prova a capire se questo $\hat{K}$ è la chiave sottostante al transcript o è un valore scelto uniformemente a caso. Stampa in output b';
4. L'output dell'esperimento è 1 se b=b' cioè l'avversario è stato in grado di capire che $\hat{K}$ corrisponde alla chiave o ad un valore casuale, 0 altrimenti.

Definizione 8.1 : Un protocollo di scambio di chiavi Π è sicuro in presenza di un ascoltatore se $\forall A$ PPT esiste una funzione trascurabile negl tale che:

$$Pr[KE^{eav}_{A,\Pi}(n) = 1] \leq \frac{1}{2} + negl(n) \tag{12.2}$$

Costruzione protocollo scambio chiavi:
Sia G(1^n) un algoritmo PPT per la generazione di gruppi ciclici che restituisce una tripla (G, q, g) dove G è un gruppo ciclico, q l'ordine e g un generatore. Il protocollo richiede che:

1. Alice esegue G(1^n) ed ottiene (G, q, g). Sceglie $x \leftarrow Z_q$ e calcola h_A:= g^x. Quindi invia (G, q, g, h_A) a Bob;
2. Bob riceve (G, q, g, h_A). Quindi calcola $y \leftarrow Z_q$ e h_B:= g^y. Invia h_B ad Alice e stampa in output K_B:= h_A^y;
3. Alice riceve h_B e dà in output $K_A := h_B^x$.

facile vedere che il protocollo è corretto perché: $K_B = h_A^y = (g^x)^y = g^{xy}$ e $K_A = h_B^x = (g^y)^x) = g^{yx}$. Il protocollo permette ad Alice e Bob di calcolare un elemento comune del gruppo. Viene applicata una trasformazione all'elemento in comune del gruppo K = g^{xy} usando un'appropriata funzione di derivazione H $\rightarrow \overline{K} = H(K)$.

Teorema 10.3 : Se il problema DDH è difficile relativamente a G, allora lo scambio di chiavi Diffie-Hellman Π è EAV-sicuro.
Nota: Sia $\hat{KE}^{eav}_{A,\Pi}$ l'esperimento $KE^{eav}_{A,\Pi}(n)$ in cui A distingue tra K e un elemnto G scelto uniformemente a caso.
Dimostrazione:
Sia A un avversario PPT. Poiché Pr[b=0] = Pr[b = 1] = $\frac{1}{2}$ risulta:

$$Pr[\hat{KE}^{eav}_{A,\Pi}(n) = 1] = \tfrac{1}{2} \cdot Pr[\hat{KE}^{eav}_{A,\Pi}(n) = 1|b = 0] + \tfrac{1}{2} \cdot \hat{KE}^{eav}_{A,\Pi}(n) = 1|b = 1]$$

Nell'esperimento $\hat{KE}^{eav}_{A,\Pi}$, A riceve (G, p, q, h_A, h_B) e $\hat{K}$ che può essere la chiave K o un elemento u uniforme in G. Pertanto la Pr[$\hat{KE}^{eav}_{A,\Pi} = 1$] risulta uguale a:

$$\begin{aligned}
&\frac{1}{2} \cdot Pr[\hat{KE}^{eav}_{A,\Pi}(n) = 1|b = 0] + \frac{1}{2} \cdot Pr[\hat{KE}^{eav}_{A,\Pi}(n) = 1|b = 1] \\
&= \cdot\frac{1}{2} Pr[A(G, q, g, g^x, g^y, g^{xy}) = 0] + \frac{1}{2} \cdot Pr[A(G, q, g, g^x, g^y, g^z) = 1] \\
&= \frac{1}{2} \cdot (1 - Pr[A(G, q, g, g^x, g^y, g^{xy}) = 1]) + \frac{1}{2} \cdot Pr[A(G, q, g, g^x, g^y, g^z) = 1] \\
&= \frac{1}{2} + \frac{1}{2} \cdot (Pr[A(G, q, g, g^x, g^y, g^z) = 1] - Pr[A(G, q, g, g^x, g^y, g^{xy}) = 1]) \\
&\leq \frac{1}{2} + \frac{1}{2} \cdot |Pr[A(G, q, g, g^x, g^y, g^z) = 1] - Pr[A(G, q, g, g^x, g^y, g^{xy}) = 1]|
\end{aligned}$$

Se DDH è difficile relativamente a d allora esiste negl tale che:

$$|Pr[A(G, q, g, g^x, g^y, g^z) = 1] - Pr[A(G, q, g, g^x, g^y, g^{xy}) = 1]| \leq negl(n)$$

Discende quindi che:

$$Pr[\hat{KE}^{eav}_{A,\Pi} = 1] \leq \frac{1}{2} + \frac{1}{2} \cdot negl(n) \tag{12.3}$$

12.2 Crittografia a chiave pubblica

Definizione 11.1 : Uno schema di cifratura a chiave pubblica è una tripla di algoritmi PPT (Gen, Enc, Dec) tale che:

1. L'algoritmo di generazioni delle chiavi Gen prende come un input il parametro di sicurezza 1^n e restituisce una coppia di chiave (pk,sk) dove pk denota una chiave pubblica e sk una chiave privata. Assumiamo per convenienza che pk e sk ciascuno ha lunghezza almeno n, e che n può essere determinata da pk, sk;

2. L'algoritmo di cifratura Enc prende come input una chiave pubblica pk e un messaggio m da qualche spazio messaggi (che potrebbe dipendere da pk). Restituisce un testo cifrato c, e scriviamo questo come c $\leftarrow Enc_{pk}(m)$;

3. L'algoritmo deterministico di decifratura Dec prende come input una chiave privata sk e un testo cifrato c, e restituisce un messaggio m o un simbolo speciale $\perp$ di fallimento. Scriviamo questo come m := Dec$_s k$(c).

È richiesto che per ogni possibile scelta della chiave privata e pubblica, prodotta tramite l'utilizzo dell'algoritmo Gen(1^n), applicando l'algoritmo di decifratura avremo che Dec$_{sk}$(Enc$_{pk}$(m)) = m per ogni messaggio m.

L'uso della crittografia a chiave pubblica ha diversi vantaggi; per esempio abbiamo che la chiave privata non sarà mai esposta e quindi è segreta, si risolve il problema di condividere la chiave, ecc... Il maggior svantaggio di questo sistema è che è molto più lenta rispetto alla crittografia a chiave privata, anche di 1000 volte.

Indistinguibilità verso un ascoltatore

PubK$^{eav}_{A,\Pi}$(n).

1. Il challenger genera tramite Gen(1^n) una coppia di chiavi (pk, sk);
2. L'avversario A riceve come input pk, e da in output una coppia di messaggi di uguale lunghezza m_0 e m_1;
3. A sceglie un bit b $\in \{0, 1\}$ scelto uniformemente, e genera il testo cifrato c $\leftarrow Enc_{pk}(m_b)$ e lo da ad A;
4. A da in output un bit b'. L'output dell'esperimento è 1 se b' = b, 0 altrimenti. Se b' = b Adv ha avuto successo.

Definizione 11.2 : Uno schema di cifratura a chiave pubblica Π = (Gen, ENc, Dec) è indistinguibile a un ascoltatore (eav) se per ogni Adv A PPT esiste una funzione trascurabile tale che:

$$Pr[PubK^{eav}_{A,\Pi}(n) = 1] \leq \frac{1}{2} + negl(n) \tag{12.4}$$

Rispetto all'esperimento PrivK$^{eav}_{A,\Pi}$ Adv riceve la chiave pubblica pk, questo equivale ad avere accesso ad un oracolo per la cifratura. La definizione quindi coincide con quella di sicurezza rispetto ad attacchi CPA.

Proposizione : Se uno schema di cifratura a chiave pubblica ha cifrature indistinguibili in presenza di un avversario che ascolta allora è CPA-sicuro.

12.2.1 Impossibilità di sicurezza perfetta

Ogni avversario A con potenza di calcolo illimitata da pk ed un cifrato c può sempre detrminare con probabilità 1.
Sketch: Assumiamo che Enc_{pk} usi n random bit, A prova a cifrare m con tutte le 2^n possibili random string, finora ad ottenere c. La condizione di correttezza garantisce che non esistono m_0 e m_1 che possono essere cifrati con lo stesso c. Quindi A è riuscito a decifrare c.
Adv vincerebbe banalmente anche se l'algoritmo di cifratura fosse deterministico perchè ad ogni messaggio è associata solo una cifratura ed è sempre la stessa.

12.2.2 Cifrature multiple

Utilizziamo lo stesso approccio usato nel contesto simmetrico:

$PubK^{LR-cpa}_{A,\Pi}(n)$

1. Viene eseguito $Gen(1^n)$ per ottenere le chiavi (pk, sk)
2. A Sceglie $b \leftarrow \{0,1\}$ in modo uniforme
3. A prende in input pk ed ha accesso all'oracolo $LR_{pk,b}(\cdot,\cdot)$;
4. A dà in output b';
5. Se b = b' l'output dell'esperimento è 1 (A vince) altrimenti 0

Definizione 11.15: Uno schema di cifratura a chiave pubblica Π = (Gen, Enc, Dec) ha cifrature multiple indistinguibili rispetto ad attacchi di tipo chosen plaintext se, per ogni Adv A PPT, esiste una funzione trascurabile negl tale che:

$$Pr[PubK^{LR-cpa}_{A,\Pi}(n) = 1] \leq \frac{1}{2} + negl(n)$$

Vale anche il seguente teorema:

Teorema : Se Π ha cifrature indistinguibili (cpa-sicuro) allora ha anche cifrature multiple indistinguibili.

12.2.3 Cifratura di messaggi di lunghezza arbitraria

Indichiamo inizialmente:

- Π = (Gen, Enc, Dec): schema di cifratura per messaggi con lunghezza 1
- Π′ = (Gen', Enc', Dec'): schema di cifratura per messaggi con lunghezza di l bit

Da quin possiamo costruire Π′ come:

- Gen' ≡ Gen: m = $m_1 m_2 ... m_l$
- Enc'(m) ≡ Enc(m): $\text{Enc}_{pk}(\text{m}_c) = c_1 c_2 ... c_l$ = c
- Dec'(c) ≡ Dec(c): $\text{Dec}_{sk}(\text{c}_c) = m_1 m_2 ... m_l$ = m

12.2.4 Chiave pubblica CCA-sicura

$\text{PubK}^{cca}_{A,\Pi}(n)$

1. Il Challenger genera tramite Gen(1^n) una coppia di chiavi (pk, sk);
2. A riceve come pk e ha accesso ad un oracolo per la decifratura $\text{Dec}_{sk}(\cdot)$. (Da notare la differenza nel contesto simmetrico in cui l'avversario aveva l'accesso all'oracolo sia per la cifratura che per la decifratura). Dá in output una coppia di messagi m_0 e m_1 della stessa lunghezza;
3. Il Challenger sceglie uniformemente b $\leftarrow \{0,1\}$ e calcola c $\leftarrow Enc_{pk}(m_b)$, restituisce poi tale risultato ad A;
4. L'avversario A continua ad interagire con l'oracolo di decifratura ma non può chiedere la decifratura del cifrato c. Alla fine l'avversario scommette su uno dei due messaggi selezionati dal challenger e restituisce b';
5. Se b=b', l'output dell'esperimento è 1 (A vince), altrimenti 0.

Definizione 11.8 Uno schema di cifratura a chiave pubblica Π=(Gen , Enc, Dec) ha una cifratura indistinguibile in presenza di un attacco chosen-ciphertext (o CCA-sicuro) se per ogni avversario A PPT esiste una funzione trascurabile tale che:

$$Pr[PubK^{cca}_{A,\Pi}(n) = 1] \leq \frac{1}{2} + negl(n) \tag{12.5}$$

Anche qui vale che se uno schema di cifrature è indistinguibile rispetto ad attacchi CCA allora lo è anche con cifrature multiple.
Non vale però che se Π è CPA-sicuro allora Π′ è CPA-sicuro

12.3 KEM - meccanismi di incapsulamento della chiave

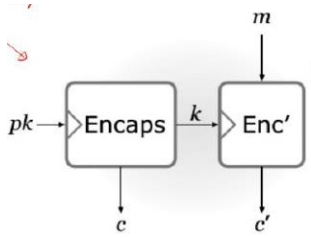
Vengono combinati schemi di cifratura a chiave pubblica e schemi di cifratura a chiave privata, in modo da realizzare i cifrari ibridi. È possibile farlo introducendo il concetto di KEM, ovvero meccanismo di incapsulamento della chiave.

Definizione 11.9: Un meccanismo di incapsulamento della chiave (KEM) è una tupla di algoritmi PPT (Gen, Encaps, Decaps) tale che:

1. Gen: prende come input un parametro di sicurezza 1^n e restituisce una coppia di chiavi (pk, sk). Assumiamo che pk e sk hanno almeno lunghezza n e che n può essere determinata da pk;
2. Encaps: prende in input una chiave pubblica pk e il parametro di sicurezza 1^n. Restituisce un testo cifrato c e una chiave $k \in \{0,1\}^{l(n)}$ dove l è la lunghezza della chiave. Scriviamo questo come (c, k) $\leftarrow$Encaps$_{pk}(1^n)$;
3. Decaps: prende in input una chiave privata sk e il testo cifrato c, e restituisce una chiave k o un simbolo speciale di fallimento $\perp$. Scriviamo questo come k := $\text{Decaps}_{sk}(c)$.

Affinché risulti corretto richiediamo che per ogni coppia di chiave pubblica e privata che può essere prodotta dall'algoritmo Gen(1^n),se Encaps$_{pk}(1^n)$ restituisce (c, k) allora $\text{Decaps}_{sk}(c)$ restituisce k.

In questo caso si utilizza Encaps per produrre il cifrato c e la chiave k che serve per l'algoritmo simmetrico per poi utilizzare l'algoritmo simmetrico Enc' per cifrare il messaggio m. Enc' viene detto meccanismo di incapsulamento dei dati (DEM).



Di seguito è riportata una costruzione di uno schema ibrido:

Costruzione 11.10:
Sia Π = (Gen, ENcaps, Decaps) un KEM con chiave di lunghezza n, e sia Π′ = (Gen', Enc', Dec') uno schema di cifratura a chiave privata. Costruiamo uno schema di cifratura a chiave pubblica Π^{hy} =(Gen^{hy}, Enc^{hy}, Dec^{hy}) come segue:

- Gen^{hy}: Prende in input 1^n ed esegue Gen(1^n) e da in output le chiavi pubbliche e private (pk, sk);
- Enc^{hy}: prende in input la chiave pubblica pk e un messaggio m $\in \{0,1\}^*$ poi:
  1. Calcola (c, k) $\leftarrow$ $\text{Encaps}_{pk}(1^n)$;
  2. Calcola c' $\leftarrow$ Enc'_k(m);
  3. Da in output il testo cifrato $<$ c, c'$>$
- Dec^{hy}: prende in input la chiave privata sk e il cesto cifrato $<$ c, c'$>$ poi:
  1. Calcola k := Decaps_{sk}(c);
  2. Da in output il messaggio m := Dec'_k(c').

12.3.1 KEM CPA-sicuro

L'esperimento che presentiamo modella la capacità dell'avversario di capire se una stringa che gli si pone davanti ha una qualche relazione con l'incapsulamento, quindi una chiave incapsulata, o è una stringa totalmente casuale. Se l'avversario non ci riesce, vuol dire che una chiave incapsulata è indistinguibile da una stringa uniforme.

$\text{KEM}^{cpa}_{A,\Pi}(n)$:

1. Gen(1^n) viene eseguito per produrre le chiavi (pk, sk). Poi $\text{Encaps}_{pk}(1^n)$ viene eseguito per generare (c, k) con $k \in \{0,1\}^n$;
2. Viene scelto in modo uniforme $b \in \{0,1\}$. Se b = 0 imposta $\hat{k}$:= k, se b = 1 sceglie in modo uniforme $\hat{k} \in \{0,1\}^n$;
3. A riceve (pk, c, $\hat{k}$) e dà in output b'. L'output dell'esperimento è 1 se b' = b altrimenti 0.

Definizione 11.11 Un meccanismo di incapsulamento delle chiavi KEM Π è CPA-sicuro se per ogni avversario A PPT esiste una funzione trascurabile negl tale che:

$$Pr[KEM^{cpa}_{A,\Pi} = 1] \leq \frac{1}{2} + negl(n) \tag{12.6}$$

Teorema : Se Π è un KEM cpa-sicuro e Π′ è uno schema di cifratura simmetrica con cifratura indistinguibile rispetto ad un eav allora Π^{hy} a chiave pubblica è CPA-sicuro.

Nota: Se Π′ non è CCA-sicuro indipendentemente dalle proprietà del KEM utilizzato nella costruzione Π^{hy} non è CCA-sicuro, questo implica che la sicurezza CCA richiede che Π′ sia CCA-sicuro.

12.3.2 KEM CCA-sicuro

In questo contesto viene cambiato il potere che diamo all'avversario, oltre al fatto che nel KEM anziché di un oracolo di decifratura viene considerato un oracolo di decapsulamento:

$\text{KEM}^{cca}_{A,\Pi}(n)$

1. $\text{Gen}(1^n)$ viene eseguito per produrre le chiavi (pk, sk). Poi $\text{Encaps}_{pk}(1^n)$ viene eseguito per generare (c, k) con $k \in \{0,1\}$;
2. Viene scelto in modo uniforme $b \in \{0,1\}$. Se b = 0 imposta $\hat{k} :=$ k, se b = 1 sceglie in modo uniforme $\hat{k} \in \{0,1\}^m$;
3. A riceve (pk, c, $\hat{k}$) ed ha accesso ad un oracolo $\text{Decaps}_{sk}(\cdot)$, ma non chiede mai la decapsilazione di c;
4. A dà in output b'. L'output dell'esperimento è 1 se b' = b, altrimenti è 0.

Definizione 11.13 : un meccanismo di incapsulamento KEM Π è CCA-sicuro se per ogni avversario A PPT esiste una funzione trascurabile tale che:

$$Pr[KEM^{cca}_{A,\Pi}(n) = 1] \leq \frac{1}{2} + negl(n) \tag{12.7}$$

Teorema : Se Π è un KEM cca-sicuro e Π′ è uno schema di cifratura simmetrico cca-sicuro allora Π^{hy} è uno schema di cifratura a chiave pubblica CCA-sicuro.

12.4 Costruzioni basate sulle assunzioni CDH/DDH

12.4.1 Cifratura El Gamal

Supponiamo che Alice e Bob, tramite scambio chiavi Diffie-Hellman, stabiliscono la chiave k. Alice per cifrare il messaggio m manda c = m * k. Bob recupera il messaggio m con l'operazione m = c/k. Se K è indistinguibile da una chiave casuale, ed è quello che ci assicura lo scambio di chiavi Diffie-Hellman, abbiamo che c è indistinguibile da un valore casuale. Quindi c non da nessun informazione sul messaggio m.

LEMMA 11.15 : Sia G un gruppo finito e sia m ∈ G scelto uniformemente a caso. Scegliamo uniformemente k ∈ G e impostiamo k' := k · m avente la stessa distribuzione di k'. Differentemente per un qualsiasi $\hat{g} \in G$ abbiamo:

$$Pr[k \cdot m = \hat{g}] = \frac{1}{|G|} \tag{12.8}$$

Prova: In ogni gruppo sappiamo che ogni elemento ha un inverso. Sia $\hat{g} \in G$ scelto uniformemente. Allora:

$$Pr[k \cdot m = \hat{g}] = Pr[k = \hat{g} \cdot m^{-1}] \tag{12.9}$$

Finché k è uniforme, la probabilità che k è uguale a un elemento fissato $\hat{g} \cdot m^{-1}$ è esattamente $\frac{1}{|G|}$.

12.4.2 Costruzione El Gamal

Sia $f(1^n)$ un algoritmo PPT che su input 1^n dà in output la descrizione di un gruppo ciclico G di ordine q (con |q|=m) ed un generatore g:

Costruzione 11.16:
Uno schema di crittografia a chiave pubblica viene definito come segue:

- Gen: Prende in input 1^n ed ottiene (G, q, g). Sopo sceglie uniformemente $x \in Z_q$ e computa h := g^x. La chiave pubblica è <G, q, g, h> e la chiave privata è <G, q, g, x>. Il messaggio è preso dallo spazio G;
- Enc: prende in input la chiave pubblica pk=<G, q, g, x> e il messaggio $m \in G$, sceglie in modo uniforme $y \in Z_q$:

$$c =< g^y, h^y \cdot m > \tag{12.10}$$

- Dec: prende in input la chiave privata sh=<G, q, g, x> e il cifrato $<c_1, c_2>$, manda in output il messaggio:

$$\hat{m} := \frac{c_2}{c_1^x} \tag{12.11}$$

La correttezza di questo algoritmo è dovuta dal seguente motivo: Dati $<c_1, c_2> = <g^y, h^y \cdot m>$ con h = g^x, risulta $\hat{m} = \frac{c_2}{c_1^x} = \frac{h^y m}{(g^y)^x} = \frac{(g^x)^y \cdot m}{(g^y)^x} \frac{(g^x)^y \cdot m}{(g^x)^y}$= m
La prima componente della coppia $<g^y, h^y \cdot m>$ serve a chi decifra in quanto mette in condizione al decifratore di poter calcolare la maschera che deve essere tolta dalla seconda componente di questa coppia.

Teorema : Se DDH è difficile allora El-Gamal è CPA-sicuro
Sketch della prova:
Provare CPA-sicuro significa che per ogni avversario PPT la probabilità che tale avversario rispetto allo schema di El-Gamal riesca a vincere nell'esperimento $\text{KEM}^{cca}_{A,\Pi}(n)$ che consiste nel distinguere tra una cifratura di m_0 e una cifratura di m_1 scelti dall'avversario stesso, risulta $\leq \frac{1}{2}$ + negl(n).
Per provare tale affermazione, introduciamo uno schema fittizio $\tilde{\Pi}$ dove la chiave pubblica è esattamente lo schema Π quindi pk = (G, q, g, h) mentre l'algoritmo di cifratura viene modificato in modo tale che produca c = $<g^y, g^z \cdot m>$. Questo algoritmo non permette a Bob di decifrare perché non ha modo di rimuovere la maschera $g^z \cdot m$, non conoscendo il valore z che è stato utilizzato ma questo non interessa in quanto importa soltanto la cifratura. Grazie al fatto che scegliendo un gruppo finito, un elemento

arbitrario, k uniforme per poi applicare k $\cdot m$ i ottiene un valore la cui distribuzione è la stessa che si ottiene se si sceglie k uniformemente a caso, lo schema Π produce delle cifrature che sono distribuite uniformemente e indipendenti dal messaggio m, per cui:

$$Pr[PubK^{eav}_{A,\Pi}(n) = 1] = \frac{1}{2} \tag{12.12}$$

Costruiamo un distinguisher D per il problema decisionale di Diffie-Hellman che cerca di distinguere tra una tripla di Diffie-Hellman da una tripla casuale, servendosi dell'avversario A che invece supponiamo sia in grado di vincere nell'esperimento CPA.

D opera nel modo seguente:

1. In input riceve quella che è un'istanza del suo problema quindi (G, q, g, h_1, h_2, h_3);
2. Costruisce una chiave pubblica pk = (G, q, g, h_1) ed esegue A(pk). L'avversario A pensa di star eseguendo l'esperimento PubK$^{eav}_{A,\Pi}(n)$ per cui ad un certo punto sfida il challenger restituendo due messsaggi $m_0, m_1 \in G$;
3. Il distinguisher che sta simulando il challenger sceglie uniformemente a caso un bit b $\in \{0, 1\}$. Dopodiché costruisce le cifrature di sfida: $c_1 = h_2, c_2 = h_3 \cdot m_b$;
4. Restituisce $<c_1, c_2>$ all'avversario A e cerca di capire se corrisponde a una cifratura di m_0 e m_1 restituendo come scommessa b';
5. D controlla se b' = b. In caso affermativo, restituisce 1 (l'avversario è stato in grado di distinguere), altrimenti 0.

Per calcolare la probabilità di successo di questo distinguisher D occorre considerare due casi:

1. Il primo caso è quello in cui il distinguisher D riceve in input una tripla casuale quindi: $h_1 = g^x$,$h_2 2 = g^y$,$h_3 = g^z$ con x,y,z$\in Z_q$. Il cifrato che viene prodotto nel secondo passo c = $<g^y, g^z \cdot m>$ è esattamente il cifrato che viene prodotto dallo schema $\tilde{\Pi}$, per cui:

$$Pr[D(G, q, g, g^x, g^y, g^z) = 1] = Pr[PubK^{eav}_{A,\tilde{\Pi}}(n) = 1] = \frac{1}{2} \tag{12.13}$$

2. Nel secondo caso supponiamo che il distinguisher D riceve in input una tripla di Diffie-Hellman: $h_1 = g^x$,$h_2 2 = g^y$,$h_3 = g^z$ con x,y,z$\in Z_q$. Il cifrato che viene prodotto nel secondo passo c = $<g^y, g^{xy} \cdot m>$ è una cifratura in accordo alla realizzazione di una cifratura tramite lo schema $PubK^{eav}_{A,\Pi}$ per cui:

$$Pr[D(G, q, g, g^x, g^y, g^{xy}) = 1] = Pr[PubK^{eav}_{A,\Pi}(n) = 1] \tag{12.14}$$

   Se DDH è difficile, allora esiste negl(n):

$$|Pr[D(G, q, g, g^x, g^y, g^z) = 1] - Pr[D(G, q, g, g^x, g^y, g^{xy}) = 1]| \leq negl(n)$$
$$|Pr[PubK^{eav}_{A,\tilde{\Pi}}(n) = 1] - Pr[PubK^{eav}_{A,\Pi}(n) = 1]| \leq negl(n)$$
$$|\frac{1}{2} - Pr[PubK^{eav}_{A,\Pi}(n) = 1]| \leq negl(n)$$
$$Pr[PubK^{eav}_{A,\Pi}(n) = 1] \leq \frac{1}{2} negl(n)$$

12.4.3 El-Gamal per schemi ibridi

- Chi invia sceglie r ∈ G e lo cifra con El-Gamal e lo manda al ricevente;
- usa H : G → $\{0,1\}^n$ (funzione di derivazione della chiave) può calcolare K = H(r);
- cifra m con un cifrario simmetrico usando K.

Cifrato complessivo: $<c_1, c_2>$, c = $Enc_K(m)$.
Esistono altri apporcci ibridi più efficienti: ad esempio il **KEM basato su El-Gamal**.

COSTRUZIONE 11.19:
Definiamo un KEM come segue:

- Gen: prende in input 1^n ed esegue G(1^n per ottenere (G, q, g), sceglie uniformemente x ∈ Z_q e imposta h := g^x. Quindi specifica una funzione H : G → $\{0,1\}^{l(n)}$ per una qualche funzione l. La chiave pubblica è <G, q, g, h. H> e la chiave privata è <G, q, g, x>;
- Encaps: prende in input la chiave pubblica pk = <G, q, g, h. H> e scegli uniformemente y ∈ Z_q e da in output il testo cifrato g^y e la chiave H(h^y);
- Decaps: prende in input la chiave privata sk = <G, q, g, x> e il testo cifrato c ∈ G, da in output la chiave H(c^x).

Il cifrato complessivo sarò $< g^y, Enc_k(m)>$

Teorema: DDH difficile e H "opportunamente scelta" => KEM CPA-sicuro

Come scegliere H? Abbiamo tre casi:

1. H: G → $\{0,1\}^l$ distribuisce quasi uniformente $\forall k \in \{0,1\}^l$, il # di elementi di G che producono k è approssimativamente lo stesso;
2. H è una funzione con chiave: la chiave è inclusa nella chiave pubblica del ricevente che si comporta come un estrattore forte;
3. H è un ROM: in quest'ultimo caso la costruzione può essere provata CPA-sicura nel modello ROM basandosi sull'assunzuine più debole CDH.

Mentre per la nozione CCA? Sfortunatamente El-Gamal è vulnerabile ad attacchi di tipo chosen-ciphertex:
Se A intercetta c = $<c_1, c_2>$ con pk = < G, q, g, h>, può costruire c' = $<c_1, c_2'>$ dove $c_2' = c_2 \cdot \alpha$ con $\alpha \in G$. Se c è una cifratura di m, cioè $<g^y, h^y \cdot m>$ allora c' = $<g^y, (h^y \cdot m) \cdot \alpha>$ risulta una cifratura $\alpha \cdot m$ e quindi c' = $<g^y, h^y \cdot (m \cdot \alpha)>$.
Quindi Adv può trasformare la cifratura di m (sconosciuto) in quella di $\alpha \cdot m$ (sconosciuto) con α noto.

Se H viene modellata come ROM si può dimostrare che il KEM è CCA-sicuro, am per farlo occorre una nuova assunzione crittografica.

12.4.4 Gap-Computational Diffie-Helmman (Gap-CDH)

CDH è ancora difficile anche se DDH è facile.

Dati g^x e g^y, per qualche generatore g, risulta computazionalmente impraticabile calcolare g^{xy} anche se si ha accesso ad un oracolo O tale che:

O(U, V) = 1 esattamente quando V = U^y

Teorema: Gap-CDH difficile e H ROM => KEM CCA-sicuro.
Mettendo insieme i risultati ottenuti fino ad ora abbiamo:

$$\left.\begin{array}{l}\Pi \text{ è un KEM CCA-sicuro} \\ \Pi' \text{ è uno schema di cifratura simmetrico CCA-sicuro}\end{array}\right\} => \text{ lo schema ibrido } \Pi^{hy} \text{ è CCA-sicuro} \tag{12.15}$$

Pertanto possiamo ottenre uno schema di cifratura ibrido a chiave pubblica CCA-sicuro usando:

- KEM basato su El-Gamal (Π_E);
- Uno schema simmetrico di cifratura autenticata (Π_M); quindi cifra il messaggio Enc e poi autentica il cifrato prodotto Mac;
- Cifratura: $< g^y$, Enc'$_{KE}(m)$, Mac$_{KM}$(c') $>$ dove la prima componente è prodotta dal KEM e le successive chiavi che vengono utilizzate per cifrare il messaggio m autenticare il cifrato c', prodotta dall'operazione di cifratura per il messaggio m, vengono in qualche modo derivate dal KEM.

La costruzione del nuovo scema ibrido è la seguente:

COSTRUZIONE 11.23
Sia Π_E = (Enc', Dec') uno schema di crittografia a chiave privata e sia Π_E = (Mac, Vrfy) un MAC. Si definisce uno schema di cifratura a chiave pubblica come segue:

- Gen: prende in input 1^* e restituisce (G, q, g) sceglie in modo uniforme $x \in Z_q$, imposta h := g^x, e specifica la funzione H:G→ $\{0,1\}^{2n}$. La chiave pubblica è <G, q,g, h, H> e la chiave privata è <G, q,g, x, H>;
- Enc: prende in input la chiave pubblica pk = <G, q,g, h,H>, sceglie in modo uniforme y $\in Z_q$, e imposta $k_E||k_M$:= H(h^y). Computa c'←Enc'$_{k_E}$(m), e manda in output il cifrato <g^y, c', Mac$_{k_M}$(c')>;
- Dec: prende in input la chiave privata sk = <G, q,g, x,H> e il cifrato <c, c', t>, dà in output $\perp$ se $c \notin G$. Altrimenti computa $k_E||k_M$:= H(c^x). Se Vrfy$_{k_M}$(c',t) $\neq 1$ allora manda in output $\perp$, altrimenti Dec'k_g(c').

12.5 DHIES / ECIES

Lo schema risultante corrisponde a quanto avviene negli schemi ibridi:

- DHIES (Diffie-Hellman Integrated Encryption Scheme), dove G è un sottogruppo ciclico di un campo finito;
- ECIES (Elliptic Curve Integrated Encryption Scheme), G è un gruppo di punti di una curva ellittica.

Corollario : Se Π_E è CPA-sicuro, Π_M è un MAC fortemente sicuro, H è un oracolo casuale e Gap-CDH è difficile relativamente a g, allora

$$\Pi^{hy}\text{(DHIES / ECIES) è CCA-sicuro} \tag{12.16}$$

Teorema: CDH difficile e H ROM => KEM è CPA-sciuro.
La prova sara-à un esempio non banale dell'uso del ROM.

DIM: Sia A PPT. Vogliamo mostrare che ∃ negl(n):

$$Pr[KEM^{cpa}_{A,\Pi}(n) = 1] \neq \frac{1}{2} + negl(n) \tag{12.17}$$

Nota: A riceve <pk, c, $\hat{k}$> e deve capire se è la chiave k o un valore random.

Sia pk = <G, q, g, h> e c = g^y e sia Query l'evento "A chiede l'hash di h$^y adH$ (cioè ottiene dall'oracolo H l'has del valore Diffie-Hellman, A per rispondere a questa query dovrebbe essere in grado di risolvere CDH).

Possiamo scrivere la probabilità di $Pr[KEM^{cpa}_{A,\Pi}(n) = 1]$ come segue:

$$Pr[KEM^{cpa}_{A,\Pi}(n) = 1] = Pr[KEM^{cpa}_{A,\Pi}(n) = 1 \wedge \overline{Query}] + Pr[KEM^{cpa}_{A,\Pi}(n) = 1 \wedge Query]$$
$$Pr[KEM^{cpa}_{A,\Pi}(n) = 1|\overline{Query}] + Pr[Query]$$

Nell'esperimento $KEM^{cpa}_{A,\Pi}(n)$ A dispone di <pk, c, $\hat{k}$ > dove $\hat{k}$ è o H(h^y) oppure un valore casuale.
Se l'evento Query non si verifica per la proprietà del ROM (1° proprietà) H(h^y) è uniformemente distribuita e quindi per A completamente indistinguibile da un valore casuale => $Pr[KEM^{cpa}_{A,\Pi}(n)|\overline{Query}] = \frac{1}{2}$ => $Pr[KEM^{cpa}_{A,\Pi}(n)] = \frac{1}{2} + Pr[Query]$

Possiamo far vedere che Pr[$Query$] ≤ $negl(n)$. Siamo nel ROM, quindi A può effettuare query all'oracolo H. Sia t il # polinomiale di query che A effettua.
Voglia sfruttare A (che gioca nell'esperimento $KEM^{cpa}_{A,\Pi}(n)$) per costruire A' che risolve il problema CDH.

Distinguisher A'

- Prende in input < (G, q, g), h, c> e calcola DH(h,c) = h^y;
- Pone la chiave pubblica pk = <G, q, g, h> e sceglie k (stringa di l-bit) uniformemente;
- Esegue A(pk, c, k). Quando A effettua query ad H, A' lo intercetta e simula le risposte inviando stringhe di l-bit scelte uniformemente a caso;
- Al termine dell'esecuzione di A, siano $w_1, w_2, ..., w_t$ le query che A ha fatto all'oracolo H. A' sceglie un indice i ∈ $\{1, ..., t\}$ e dà in output w_i.*

NOTA *: A' scommette che al passo i-esimo A abbia chiesto ad H l'hash del valore diffie-Hellman di h e c, che sarebbe appunto w_i. Equivalentemente possiamo dire che A' scommette che Query sia avvenuto alla i-esima query di A.

Qual'è quindi la $Pr[A'(G, q, g, h, c) = h^y]$ dove h^y = DH(h,c).
Osservazione: l'evento Query nella simulazione che A' realizza si verifica con la stessa probabilità con cui si verifica nell'esperimento $KEM^{cpa}_{A,\Pi}$. Quindi:

$$\text{Vista di A come soubroutine di A'} \equiv \text{vista di A in } KEM^{cpa}_{A,\Pi}(n).$$

Fino a quando non si verifica l'evento Query che in $KEM^{cpa}_{A,\Pi}(n)$ fa ricevere a A il valore reale H(h^y) uguale a K con probabilità $\frac{1}{2}$; In A' invece fa ricevere a A un valore uniforme.

Pertanto se l'evento Query si verifica:

$$h^y \in \{w_1, w_2, ..., w_t\} \tag{12.18}$$

Questo implica che A' dà in output il valore corretto con probabilità $\frac{1}{t}$. Pertanto:

$$Pr[A'(G, q, g, h, c) = h^y] \geq \frac{Pr[Query]}{t} \tag{12.19}$$

Ma se CDH è difficile, ∃ negl(n) tale che:

$Pr[A'(G, q, g, h, c) = h^y] \leq negl(n)$
$=> Pr[Query] \leq t \cdot negl(n) = negl(n)$(Questo perché polinomiale * negl fa negl)

12.6 Schemi basati sull'assunzione RSA

L'algoritmo RSA-plain nasce nel 1978:

GenRSA:
Input: Parametro di sicurezza 1^n
Output: N, e, d

(N, p, q) ←GenModulus(1^n)
$\phi(N)$:= (p-1)(q-1)
sceglie e > 1 tale che gcd(e, $\phi(N)$) =1
calcola d := $[e^{-1} mod \phi(N)]$ return N, e, d

Costruzione 11.26: Dato GenRSA, definiamo uno schema a chiave pubblica come segue:

- Gen: esegue GenRSA(1^n) per ottenere N, e, d. Da in output la chiave pubblica è <N, e> e la chiave privata è <N, d>;
- Enc: prende in input la chiave pubblica pk = <N, e> e un messaggio m ∈ Z_N^*, e calcola il cifrato:

$$c := [m^e \ mod \ N] \tag{12.20}$$

- Dec: prende in input la chiave privata sk: = <N, d> e il cifrato c ∈ Z_N^*, calcola il messaggio:

$$m := [c^d \ mod \ N] \tag{12.21}$$

La sicurezza di RSA si basa sul problema della fattorizzazione difficile e che la permutazione RSA sia difficile.

Ma RSA-plain ha dei problemi:

- se il messaggio m non è scelto uniformemente in Z_N^*;
- se Adv è interessato ad informazioni parziali;
- l'output è deterministico.

Quindi si è passati dall'assunzione di RSA-plain a RSA-randomizzato

12.6.1 RSA-randomizzato

COSTRUZIONE 11.30: Sia l una funzione con l(n)≤ 2n - 4 per tutti gli n. Si definisce uno schema di crittografia a chiave pubblica:

- Gen: prende in input 1^n, esegue GenRSA(1^n) ed ottiene (N, e , d). Da in output la chiave pubblica è <N, e> e la chiave privata è <N, d>;
- Enc: prende in input la chiave pubblica pk = <N, e> e un messaggio m ∈ $\{0,1\}^{|N|-1(n)-2}$, sceglie in modo uniforme $r \in \{0,1\}^{l(n)}$ e interpreta $\hat{m}$:= r || m come un elemento di $Z*_N$. Manda in output il cifrato:

$$c := [\hat{m}^e \; mod \; N] \tag{12.22}$$

- Dec: prende in input la chiave privata sk: = <N, d> e il cifrato c ∈ Z_N^*, calcola il messaggio:

$$\hat{m} := [c^d \; mod \; N] \tag{12.23}$$

  E manda in output i |N|-l(n)-2 bits meno significativi di $\hat{m}$.

L'algoritmo di RSA permette di vedere i messaggi come stringhe binarie di lunghezza al più |N|-l(n)-2. La sicurezza di Padded RSA dipende da l. Se l è troppo piccolo, possono essere applicati tutti i possibili valori di r e applicare l'attacco che sfrutta la conoscenza parziale di $\hat{m}$ per decifrare c. Quindi l deve essere fissato per un valore tale che non consenta la ricerca esaustiva di tutti i possibili valori di r a cui possono essere sfruttati per avere il messaggio originale m.
D'altra parte, se il PAD è il più grande possibile, per esempio con m che è un singolo bit, allora Padded RSA è CPA-sicuro. Tuttavia, per i casi intermedi, casi in cui $\hat{m} = r||m$ con |r| = $\frac{1}{2}$ e |m| = $\frac{1}{2}$ la situazione non è chiara perché non ci sono prove che attacchi PPT non possono esistere se vale l'assunzione RSA ma al momento non se ne conoscono.

12.6.2 RSA PKCS#1 v1.5

Nato nel 1993, utilizza una variante del padding descritto precedentemente. Precisamente data pk = <N, e> definiamo:

- k : lunghezza di N in byte, vale a dire $2^{8(k-1)} \leq N < 2^{8k}$;
- D : lunghezza del messaggio m in byte. Deve essere: $1 \leq D \leq k - 11$.

La cifratura di un messaggio m allora diviene:

$$\text{c} = (\text{0x00} \;||\; \text{0x02} \;||\; \text{r} \;||\; \text{0x00} \;||\; \text{m})^e \bmod \text{N}$$

Dove:
- r è una stringa casuale di K-D-3 byte;
- m sono D byte.

Poiché $D \leq K - 11$ => r + di almeno 8 byte. Purtroppo non è CPA-sicuro.
Un Adv può calcolare la parte iniziale di un messaggio m di cui si sa che ha molti 0 alla fine.
Pertanto Adv applicando gli attacchi per messaggi brevi oppure gli attacchi basati su conoscenza parziale di m riesce a calcolare un bit b del messaggio.

Occorre avare quindi valori di r più grandi.
Se r è circa la metà del messaggio si congettura che PKCS#1 v1.5 risulti CPA-sicuro. Tuttavia è noto

un attacco CCA che ha indotto ad introdurre nuovi standard.

Descriviamo quindi un nuovo schema di cifratura CPA-sicuro basato sull'assunzione RSA:

- descriviamo un predicato Hard-core specifico per la permutazione RSA;
- mostriamo come usare il predicato per cifrare un messaggio di un singolo bit.

Dato un algoritmo GenRSA ed un Adv definiamo l'esperimento $RSA - lsb_{A,GenRSA}(n)$

12.6.3 RSA-lsb

Definiamo l'esperimento $RSA - lsb_{A,GenRSA}(n)$ come segue:

1. Esegue GenRSA(1^n) e ottiene (N, e, d);
2. sceglie uniformemente x $\in Z_N^*$ e calcola y := $[x^e \, mod \, N]$;
3. A prende in input N, e, y e da in output un bit b;
4. L'output dell'esperimento è 1 (Adv vince) se e solo se lsb(x) = b.

lsb(x) è il bit meno significativo.

NOTA: lsb(x) è un bit uniforme quando $x \in Z_N^*$ viene scelto uniformemente a caso. A può indovinare lsb(x) con probabilità $\frac{1}{2}$
Teorema: RSA difficile relativamente a GenRSA => ∀A PPT ∃ negl(n) tale che:

$$Pr[RSA\_lsb_{A,GenRSA}(n) = 1] \leq \frac{1}{2} + negl(n) \tag{12.24}$$

In altre parole lsb(x) è un predicato hardcore per la permutazione RSA.
Intuizione sulla prova del teorema:
Se esistesse un algoritmo efficiente che calcola sempre lsb(r) da N, ed r^e mod N, allora potrebbe essere usato per calcolare efficientemente x da N, e ed x^e mod N. Corrisponde ad invertire RSA.

Iniziamo fissando la chiave pubbluca (N, e) e prendiamo un A tale che A($[r^e$ mod N]) = lsb(r).
Dati N, e ed y = $[x^e$ mod N], possiamo calcolare i bit di x uno dopo l'altro, dal meno significativo al più significativo. Infatti per determinare lsb(x) eseguiremo A. Distinguiamo due casi:

- sia lsb(x) = 0: Sappiamo che $\frac{y}{2^e} = (\frac{x}{2})^e$ mod N e poiché x è pari si divede x per 2. Questo corrisponde a uno shift a destra di un posto di x, quindi lsb($\frac{x}{2}$) è il secondo bit meno significativo di x. SI calcola quindi facendo:

  $$y' = [\frac{y}{2^e} \text{ mod N}] \tag{12.25}$$

  E poi si calcola A(y')

- sia lsb(x) = 1: In questo caso notiamo che:

  $$[\frac{x}{2} \text{mod N}] = \frac{(x+N)}{2} \tag{12.26}$$

  Quindi lsb($[\frac{x}{2}$mod N]) risulta uguale a 2sb(x+N) (secondo bit meno significativo),
  2sb(x+N) = 1 ⊕ 2sb(N) ⊕ 2sb(x).

1 rappresenta un riporto perché sia x che N sono dispari.
Pertanto calcoliamo:

$$y' = [[\frac{y}{2^e} \text{mod N}]] \tag{12.27}$$

alloar risulta

$$2sb(x) = A(y') \oplus 1 \oplus 2sb(N) \tag{12.28}$$

Procedendo in questo modo si possono ricavare tutti i bit di x.

Mostriamo ora una costruzione che mostri come usare RSA_lsb.

Teorema: RSA difficile => Costrruzione CPA-sicura (relativamente a GenRSA)

COSTRUZIONE 11.32:
Definiamo uno schema Π di cifratura a chiave pubblica per $RSA - lsb_{A,\Pi}(n)$ che usi GenRSA come segue:

- Gen: essegue GenRSA(1^n) per ottenre (N, e, d). da in output la chiave pubblica pk = <N, e> e la chiave privata sk = <N, d>;
- Enc: prende in input la chiave pubblica pk = <N, e> e il messaggio m $\in \{0,1\}$, sceglie uniformemente r $\in Z_N^*$ con il vincolo che lsb(r) = m.
  Da in output il cifrato c := [r^e mod N];
- Dec: prende in input la chiave privata sk = <N, d> e il cifrato c, calcola r := [c^d mod N] e da in output lsb(r).

Dimostrazione: Π ha cifratue indistinguibili in presenza di un Adv che ascolta => Π è CPA-sicuro.

Sia A PPT $m_0 = 0$ ed $m_1 = 1$ in $\text{Pub}^{eav}_{A,\Pi}$(n).

$$\begin{aligned} Pr[Pub^{eav}_{A,\Pi}(n) = 1] = &\frac{1}{2} pr[A(N,e,c) = 0 | \text{c è una cifratura di} m_0] \\ &+ \frac{1}{2} pr[A(N,e,c) = 1 | \text{c è una cifratura di} m_1] \end{aligned}$$

NOTA: $\frac{1}{2}$ rappresenta la scleta uniformemente a caso del bit b per cifrare m_b.

Se supponiamo di eseguire A nell'esperimento RSA_lsb:

$$Pr[RSA\_lsb_{A,GenRSA}(n) = 1] = Pr[A(n,e), [r^e \text{ mod N}] = lsb(r)] \tag{12.29}$$

Poiché Pr[$lsb(r) = 1$] = $\frac{1}{2}$, risulta

$$\begin{aligned} Pr[RSA\_lsb_{A,GenRSA}(n) = 1] = &\frac{1}{2} Pr[A(n,e), [r^e \text{ mod N}] = 0 | lsb(r) = 0] \\ &+ \frac{1}{2} Pr[A(n,e), [r^e \text{ mod N}] = 1 | lsb(r) = 1] \end{aligned}$$

Notando che cifrare m $\in \{0,1\}$ corrisponde a scegliere un r uniforme che soddisfi lsb(r) = m, discende che:

$Pr[A(N,e,c) = b|$ c è un cifrato di b] = $Pr[A(n,e), [r^e \text{ mod N}] = b | lsb(r) = b]$

$$=> Pr[Pub^{eav}_{A,\Pi}(n) = 1] = Pr[RSA\_lsb_{A,G}$$

Avendo supposto che lsb(r) è un predicato hardcore per la permutazione RSA relativamente a GenRSA, allora ∃negl(n):

$$Pr[RSA\_lsb_{A,GenRSA}(n) = 1] \leq \frac{1}{2} + negl(n)$$
$$Pr[Pub^{eav}_{A,\Pi}(n) = 1] \leq \frac{1}{2} + negl(n)$$

Pertanto Π è CPA-sicuro.

12.6.4 KEM basato su cifratura con predicato hardcore

- Una prima soluzione è scegliere una chiave k di n bit cifrati uno per uno.
  Problema: un cifrato troppo lungo implica n elementi di Z^*_N.

- Seconda soluzione: sceglie c uniformemente a caso in Z^*_N, calcola $c^e, c^{e^2}, c^{e^3}, ..., c^{e^n}$ questa è un'applicazione ripetuta di RSA, successivamente k = $lsb(c)lsb(c^e)...lsb(c^{e^{n-1}})$

La chiave può essere recuperata decifrando c^{e^n} successivamente:

$c^{e^{n-1}} corrispondealsb$
$c^{e^{n-2}} corrispondealsb$
...
$c^e corrispondealsb$
$ccorrispondealsb$

COSTRUZIONE 11.34:
Definiamo un KEM come segue:

- Gen: esegue GenRSA(1^n) per ottenere (N, e, d). Calcola poi d' := [d^n mod$\Phi(N)$]. da in output pk = (N, e) e sk = (N, d');
- Encaps: prende in input pk = (N, e) e 1^n, sceglie uniformemente $c_1 \in Z^*_N$.
  Successivamente for i = 1, ..., n do:
  1. Calcola k_i := lsb(c_i)
  2.
  3. Calcola $c_i + 1$:= [c_i^e mod N]

  Da in output il cifrato c_{n+1} e la chiave k = k_1 ... k_n
- Decaps: prende in input sk = (N, d') e il testo cifrato c, calcola c_1 := [$c^{d'}$ mod N.
  Successivamente for i = 1, ..., n do:
  1. Calcola k_i := lsb(c_i)
  2.
  3. Calcola $c_i + 1$:= [c_i^e mod N]

  Da in output la chiave k = k_1 ... k_n.

Osservazione: la costruzione ricorda l'approccio usato per costruire un generatoe pseudocasuale a parire da una permutazione one-way.
Per esempio f(x) = [x^e mod N] con (N,e) chiave pubblica.

$$\text{Sicurezza CPA della costruzione} \iff \text{pseudocasualità di } f^n(c)lsb(f^{n-1}(c))...lsb(c) \tag{12.30}$$

Quindi possiamo presentiamo il seguente teorema:
Teorema: RSA difficile relativamente a genRSA => KEM CPA-sicuro.

Sicurezza CCA Tutte le costruzione basate su RSA viste fino ad ora non sono CCA-sicure. Plain RSA è malleabile:
Data c = m^e mod N, per qualche m risulta:

$$c' = (2^e)c \text{ mod N} = (2m)^e \text{ mod N} => \text{ c' è una cifratura di 2 m} \tag{12.31}$$

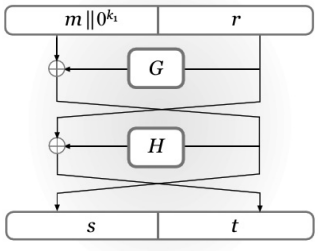
Mentre per RSA PKCS #1 v1.5 è soggetto ad un attacco CCA importante proposto da Daniel Bleichenbacher nel 1998.
Anche il Kem basato sugli RSA visti fino ad ora è debole ad attacchi di tipo CCA.

12.6.5 RSA OAEP (optimal asymetric encryotion padding)

è uno schema di cifratura che soddisfa la nozione di CCA-sicuro. L'idea è basata su RSA con padding. Il messaggio m viene trasformato attraverso un mapping randomizzato in un elemento di Z_N^* per poi applicare la permutazione RSA:

$$m \to \hat{m} \in Z_N^* \to c = [\hat{m}^e mod N] \tag{12.32}$$

L'idea è concatenare al messaggio m, con del padding, una stringa randomizzata r. La trasformazione consiste nell'applicazione di una rete di Feistel a due round e due funzioni hash G e H.
Nel primo passaggio, il parametro r viene sottoposto alla funzione G(r) per poi fare l'operazione XOR di questo risultato con m' = m || 0^{k^l}. Nel secondo passaggio, il risultato di questa operazione viene stampato in output come parte sinistra del messaggio e al contempo come input per la funzione H. Da qui viene fatto l'operazione di XOR con il parametro r e s=m' $\oplus$ G(r). Il risultato t = r $\oplus$ H(s) viene così stampato come parte destra del messaggio. Tramite G(r) si protegge m', mentre tramite H(s) protegge il seme r.



COSTRUZIONE 11.36:
Sia G : $\{0,1\}^{k_0} \leftarrow \{0,1\}^{l+k_1}$ e H : $\{0,1\}^{l+k_1} \leftarrow \{0,1\}^{k_0}$, Costruiamo uno schema di cifratura a chiave pubblica come segue:

- Gen: Esegue GenRSA(1^n) per ottenere (N, e, d). La chiave pubblica è (N, e) e òa chiave privata è (N, d);
- Enc: prende in input la chiave pubblica (N, e) e il messaggio m $\in \{0,1\}^l$, setta m' := m || 0^{k^1} e sceglie r $\in \{0,1\}^{k_0}$ uniformemente. Poi calcola:

$$s := m' \oplus G(r), \quad t := r \oplus H(s) \tag{12.33}$$

  e imposta $\hat{m}$:= s || t. Da in output il testo cifrato c := [$\hat{m}^e$ mod N]
- Dec: prende in input la chiave privata (N, d) e il cifrato c $\in Z_N^*$, calcola $\hat{m}$:= [c^d mod N]. Se || $\hat{m}$|| > l + k_0 + k_1, e da in output $\perp$. Altrimenti analizza $\hat{m}$ come s||t con s $\in \{0,1\}^{l+k_1}$ e t $\in \{0,1\}^{k_0}$. Calcola r:= H(s) $\oplus t$ e m' := G(r) $\oplus$s. Se i k_1 bit meno significativi di m' non non sono tutti 0 da in output $\perp$. Altrimenti da in output gli l bit meno significativi di $\hat{m}$.

Si può dimostrare che RSA relativamente a GenRSA + H e G ROM $\leftarrow$ RSA-OAEP CCA sicura. L'intuizione del risultato è la seguente, durante la cifratura il mittente calcola:

$$m' = m||0^{k_1}, s = m' \oplus G(r) et = r \oplus H(s), \text{ con r scelto in modo uniforme} \tag{12.34}$$

Il cifrato risultante è

$$c = [(s||t)^e \text{ mod N}] \tag{12.35}$$

Se l'avversario non chiede r all'oracolo G, G(r) uniforme per l'avversario e, quindi, m' protetto da G come se fosse cifrato dal ONE-TIME PAD in S in quanto viene effettuato uno XOR tra il messaggio m' e una stringa totalmente casuale. Quindi l'unico modo per l'avversario di calcolare il messaggio m quello di riuscire ad inviare la query r a G perché nel random oracle model, fin quando non viene chiesto la query r all'oracolo G, il valore G(r) uniforme. Si noti che r è protetto da H(s) in t. Quindi se l'avversario non chiede la query s all'oracolo H, r è protetto come se fosse cifrato dal ONE-TIME PAD in t. l'alternativa sarebbe provare a indovinare scegliendo a caso, ma se la lunghezza di r sufficientemente grande, la probabilità che l'avversario indovini è trascurabile. Pertanto, l'unica cosa che l'avversario può fare è calcolare:

$$s \text{ da } [(s||t)^e \text{ mod N}] \tag{12.36}$$

per poter poi inviare la query s all'oracolo H così da calcolare r=t $\oplus$ H(s).
Tale strategia sembra plausibile perché se l'assunzione RSA afferma che dal cifrato c computazionalmente inammissibile calcolare $\hat{m}$ (cioè s||t), si sta solo cercando di recuperare una parte di $\hat{m}$ quindi non viola l'assunzione RSA.
Fortunatamente, si può dimostrare che il recupero di s tramite A PPT implica il recupero di t in tempo polinomiale, quindi significa dire che se fossimo in grado di calcolare soltanto la prima parte di $\hat{m}$. saremmo in grado di invertire l'intera stringa. Possiamo concludere, quindi, che recuperare s è computazionalmente inammissibile.

12.6.6 KEM CCA-sicuro basato su RSA

COSTRUZIONE 11.37:
Sia GenRSA come al solito e costruiamo il KEM come segue:

- Gen: prende in input 1^n ed esegue GenRSA(1^n) ed ottiene (N, e, d). Manda in output la chiave pubblica pk = (N, e) e la chiave privata sk = (N, d). La parte che genera la chiave, specifica una funzione H: $Z_N^* \leftarrow \{0,1\}^n$, ma questo non ha nessun impatto;
- Encaps: prende in input la chiave pubblica pk = (N, e) il parametro di sicurezza 1^n, sceglie in modo uniforme r $\in Z_N^*$. Manda in output c:= [r^e mod N] e la chiave k := H(r);
- Decaps: prende in input la chiave privata sk = (N, d) e un cifrato c $\in Z_N^*$, calcola r:=[c^d mod N] e manda in output la chiave k := H(r).

Si è quindi in grado di costruire uno schema di cifratura ibrido efficiente e CCA-sicuro perché possiamo considerare l'unione di KEM con uno schema di cifratura simmetrico.

Teorema: Se il problema RSA è difficile relativamente a GenRSA la funzione H scelta è ROM allora il KEM risultante è CCA-sicuro.
Dimostrazione (sketch)
Sia A un avversario PPT e consideriamo $KEM^{cca}_{A,\Pi}(n)$. L'esperimento svolge al mondo seguente:

1. Il challenger usa GenRSA(1^n) per preparare l'esperimento. Quindi genera chiave pubblica e chiave privata per poi scegliere una funzione casuale H: $Z_N^* \leftarrow \{0,1\}^n$ per eseguire l'operazione di derivazione;
2. Seleziona un valore casuale r $\in Z_N^*$ calcola il cifrato c:= [r^e mod N] e la chiave k = H(r);
3. Il passo precedente serve per dare una tripla di sfida all'avversario, quindi, completata tale operazione sceglie un bit uniformemente a caso b $\in \{0,1\}$. Se b=0 allora setta $\hat{k} = k$. Se b=1 allora seleziona in maniera uniforme k $\in \{0,1\}^n$;
4. L'avversario A riceve la chiave pubblica pk = (N, e), c e $\hat{k}$. L'avversario potrebbe interrogare l'oracolo H($\cdot$) l'oracolo di decapsulazione Decaps(N, d)$^{(\cdot)}$ con l'eccezione che non potrà chiedere la decifratura del cifrato c, altrimenti non ha più senso l'esperimento;
5. Ad un certo punto, dopo varie interazioni con gli oracoli, l'avversario A stampa il bit di sfida b' (lo scopo è capire se la chiave è stata scelta tramite una funzione casuale o casualmente). L'output di tale esperimento è 1 se b' = b, altrimenti 0.

La condizione che permetterebbe all'avversario di vincere l'esperimento dipende dalla query che effettua. Quindi se questo, una volta ricevuto la tripla sfida <pk, c, $\hat{k}$>, effettua una query giusta all'oracolo H(r) = $\hat{k}$, ottenendo la chiave corrispondente per la query r per poi calcolare c = r^e mod N sarebbe certo che la chiave $\hat{k}$ è la chiave di incapsulamento del cifrato c.
Indichiamo gli eventi Query = { A fa query r all'oracolo H } e Success = { b'=b ovvero l'evento in cui l'avversario vince }.
La probabilità dell'evento Success può essere definita con la probabilità dei due eventi complementari:

$$\begin{aligned} Pr[Success] &= Pr[Success \wedge \overline{Query}] + Pr[Success \wedge Query] \\ &\leq Pr[Success \wedge \overline{Query}] + Pr[Query] \end{aligned}$$

Condizionato dall'evento $\overline{Query}$, Adv non fa query r all'oracolo H, e il valore k = H(r) è uniformemente distribuito perché H, essendo un oracolo casuale, per la prima proprietà del ROM afferma che fin quando

non viene effettuato la query all'oracolo, il valore corrispondente a questo è uniformemente distribuito. Pertanto, fino a quando Query non si verifica, si ha che:

$$\leq Pr[Success|\overline{Query}] = \frac{1}{2} \tag{12.37}$$

Mentre per dimostrare che $Pr[Query] \leq$ negl(n), i vuole vedere la costruzione di A' in grado di invertire la permutazione RSA per cui se A vince con una probabilità non trascurabile nell'esperimento, A' riesce ad invertire con una probabilità non alquanto trascurabile. Faremo una riduzione del problema RSA al problema di vittoria dell'avversario in $KEM^{cca}_{A,\Pi}(n)$.
per far ciò A' esegue A e:

- Risponde alle query di A per H. Ogni volta che H riceve una query p risponde con H(p) scelto uniformemente a caso, quindi A' farà la stessa cosa;
- Risponde alle query di A per Decaps. Questo rappresenta un problema in quanto Decaps funziona che quando riceve r, quindi decifra e calcola H(r), utilizza la chiave privata sk = <N, d>, cosa di cui A' non dispone, così gli invia valori casuali.

A' però deve fare attenzione a rispondere consistentemente alle query per H e Decaps:

- per ogni $\hat{r}, \hat{c}$ con $\hat{r}^e = \hat{c}$ mod N. risulta h($\hat{r}$) = decaps ($\hat{c}$)
- A' può quindi usare due liste:
  - L_H = risposte alle query per H;
  - L_{Decaps} = risposte alle query per Decaps;

Algoritmo A':
L'algoritmo prende in input (N, e, c):

1. Inizializza due liste vuote L_H, L_{Decaps}, sceglie uniformemente k $\in \{0,1\}^n$ e salva (c, k) in L_{Decaps};
2. Sceglie uniformemente un bit b $\in \{0,1\}$. Se b = 0 setta $\hat{k} := k$, se b = 1 sceglie uniformemente $\hat{k} \in \{0,1\}^n$. Esegue A sull'input (N,e), c, $\hat{k}$. Corrisponde ad A' che simula $KEM^{cca}_{A,\Pi}(n)$ per A.
   Quando A fa una query H($\tilde{r}$), risponde come segue:
   - Se nella lista L_H è presente $(\tilde{r}, k)$ per un k, allora ritorna k;
   - Altrimenti, sia $\tilde{c}$:= [$\tilde{r}^e$ mod N]. Se nella lista L_{Decaps} è presente $(\tilde{c}, k)$ per un k, ritorna k e salva in L_H la coppia $(\tilde{r}, k)$.

   Quando A fa una query Decaps($\tilde{c}$), risponde come segue:
   - Se nella lista L_{Decaps} è presente $(\tilde{c}, k)$ per un k, ritorna k;
   - Altrimenti per ogni coppia $(\tilde{r}, k) \in L_H$ contolla se $\tilde{r}^e = \tilde{c}$ mod N e se è così da in output k;
   - Altrimenti sceglie uniformemente un k $\in \{0,1\}^n$ e ritorna k, salvando $(\tilde{c}, k)$ in L_{Decaps}.
3. Alla fine dell'esecuzione di A' se nella lista L_H è presente (r,k) per cui r^e = c mod N ritornerà r.

Pertanto, considerando che A' ha la capacità di simulare completamente l'ambiente reale per A, possiamo dire che se il problema RSA è difficile relativamente a genRSA allora avremo che:

$$Pr[Query] \leq negl(n) \tag{12.38}$$

Pertanto RSA difficile => $Pr[Query] \leq negl(n)$ relativamente a GenRSA.

Capitolo 13

Firma Digitale

Possono essere viste come analogia dei codici per autenticazione dei messaggi (MAC) ma utilizzata in un contesto asimmetrico invece che simmetrico.

Funzionamento: A firma un messaggio utilizzando la sua chiave privata: SK_A e produce la firma σ, invia a B la coppia (messaggio, σ). B utilizzando la chiave pubblica di a: PK_A verifica la firma.

Le firme digitali rispetto ai MAC sono più lunghe e di 2 o 3 ordini di grandezza più lente da generare/verificare.

Definizione 12.1: Uno schema di firma digitale consiste in tre algoritmi polinomiali (Gen, Sign, Vrfy) definiti come segue :

- Gen: prende in input il parametro di iscurezza 1^n e da in output la coppia di chiavi (pk, sk);
- Sign: prendi in input sk a un messaggio m $\in \{0,1\}^*$ scelto uniformemente a caso. Da in output la firma σ. Indicheremo questo processo come segue: $\sigma \leftarrow Sign_{sk}(m)$
- Vrfy: prende in input pk, un messaggio m, e la firma σ. Da in output un bit b, se b = 1 vuol dire che è valida se b = 0 vuol dire che non è valida. Indicheremo questo processo con b := Vrfy_{pk}(m, σ).

Definizione 12.2: Una schema di firma Π = (Gen, Sign, Vrfy) è non falsificabile verso un attacco di tipo chosen-message PPT per ogni avversario A se esiste negl(n):

$$Pr[\text{Sig-forge}_{A,\Pi}(n) = 1] \leq negl(n) \tag{13.1}$$

Definiamo l'esperimento come segue:

$\text{Sig-forge}_{A,\Pi}(n)$:

1. Gen(1^n) è eseguito per ottenere la coppia di chiavi (pk, sk);
2. Adv A prende pk e ha accesso all'oracolo $\text{Sign}_{sk}(\cdot)$. L'avversario da in output (m, σ). Denotiamo Q come l'insieme delle query fatte da A verso l'oracolo;
3. A ha successo se e solo se Vrfy_{pk}(m, σ) = 1 e m $\notin Q$.

13.1 Firma digitale ibrida

Anche nella firma digitale un approccio ibrido permette d i avere schemi performanti. Nel caso della firma si utilizza il paradigma hash_and_sign.

Paradigma hash_and_sign:

Sia $\Pi = (Gen, Sign, Vrfy)$ uno schema di firma per un messaggio di lunghezza l(n) e sia $\Pi_H = (Gen_H, H)$ una funzione hash con output di lunghezza l(n)

. Costruiamo uno schema $\Pi' = (Gen', Sign', Vrfy')$ come segue:

- Gen': esegue Gen(1^n) per ottenere (pk, sk) e esegue $\text{Gen}_H(1^n)$ per ottenere s;
- Sign': prende in input la coppia <sk, s> e il messaggio $m \in \{0,1\}^*$, da in output $\sigma \leftarrow Sign_{sk}(H^s(m))$;
- Vrfy': prende in input la coppia <pk, s> e un messaggio m $\in \{0,1\}^*$, e una firma σ, da in output 1 se e solo se $\text{Vrfy}_{pk}(\text{H}^s(\text{m}), \sigma) \stackrel{\text{def}}{=} 1$

Teorema: Se Π è uno schema di firma digitale per messaggi di lunghezza l sicuro e Π_H è una funzione hash resistente a collisione, allora la costruzione hash-and-sign è sicura per messaggi di lunghezza arbitraria.

COSTRUZIONE 12.5: Si definisce uno schema di firma come segue:

- Gen: GenRSa(1^n) è eseguito per ottenere (N, e, d), dove pk =(N, e) e sk = (N, d);
- Sign: prende in input la chiave sk e il messaggio $m \in Z_N^*$, computa la firma:

$$\sigma := [m^d \text{ mod N}] \tag{13.2}$$

- Vrfy: prende in input pk, il messaggio $m \in Z_N^*$ e una firma $\sigma \in Z_N^*$, manda in output 1 se e solo se:

$$m := [\sigma^d \text{ mod N}] \tag{13.3}$$

Poiché il problema RSA è ritenuto difficile, si potrebbe pensare che sia difficile produrre contraffazioni. In realtà, è possibile produrre contraffazioni:

- Attacco senza messaggi: con RSA,si può effettuare un attacco che utilizza solo la chiave pubblica. Data la chiave pubblica (N, e), sceglie un valore a caso $\sigma \in Z_N^*$, calcola il messaggio m = $[\sigma^e \text{mod n}]$ e dà in output (m, σ). Questo è una contraffazione perché per come è costruita l'algoritmo di verifica, richiede che data σ si verifichi se $\sigma^e = m$. Per cui la firma σ è valida su m e non è stata generata da legittimo firmatario.
- Contraffazione di un messaggio arbitrario: L'attacco dimostra che un avversario rispetto allo schema di firma può produrre delle contraffazioni, ma utilizzando l'oracolo di firma ha la possibilità di firmare ciò che vuole. Per produrre una firma di un messaggio dell'interesse dell'avversario, vengono utilizzate due firme. Vengono scelti due messaggi tale che m = $m_1 \cdot m_2$ mod N, per poi inviarli come query all'oracolo di firma ottenendo le firme $\sigma_1 e \sigma_2$ L'avversario prende le due firme

per fare l'operazione modulare $\sigma = [\sigma_1\sigma_2 \bmod N]$ su m. Quindi la contraffazione sarà (m, σ). Risulta:

$$\sigma^e = (\sigma_1 \cdot \sigma_2)^e = (m_1^d \cdot m_2^d)^e = (m_1^{de} \cdot m_2^{de}) = m_1 \cdot m_2 = m \bmod N \tag{13.4}$$

In questo modo siamo riusciti a firmare un messaggio tramite l'utilizzo di due firme diversi.

L'attacco può essere generalizzato.
Dato un insieme di q firme su $M = \{m_1, ..., m_q\}$, Adv può produrre firme su $2^q - q$ alti messaggi. Devastante!

13.1.1 RSA-FDH

un algoritmo di firma digitale che utilizza il paradigma hash-and-sign.

COSTRUZIONE 12.6: Sia uno schema di firma digitale che utilizza GenRSA e costruito come segue:

- Gen: GenRSa(1^n) è eseguito per ottenere (N, e, d), dove pk =(N, e) e sk = (N, d). La parte che genera la chiave specifica una funzione H:$\{0,1\}^* \to Z_N^*$, ma questo non ha nessun impatto;
- Sign: prende in input la chiave sk e il messaggio $m \in Z_N^*$, computa la firma:

$$\sigma := [H(m)^d \bmod N] \tag{13.5}$$

- Vrfy: prende in input pk, il messaggio m e una firma σ, manda in output 1 se e solo se:

$$\sigma^e := [H(m)^d \bmod N] \tag{13.6}$$

Le proprietà che deve avere la funzione H è che deve essere: difficile da invertire, non ammettere relazioni moltiplicative e difficile trovare collisioni. Se la funzione H viene modellata come oracolo casuale, le tre condizioni sono soddisfatte.
Teorema: Se RSA è un problema difficile relativamente al GenRSA, che scegliamo nella costruzione dello schema di firma, e se H si comporta come un oracolo casuale allora la costruzione RSA-FDH è sicuro.

Sketch: Supponendo che esista un A PPT in grado di produrre contraffazioni, ovvero coppie (m, σ) false, allora si potrebbe costruire un A altrettanto PPT che è in grado di risolvere il problema RSA, supposto difficile, con probabilità non trascurabile. L'idea è che A' simula l'esperimento Sign-forge$_{A,\Pi}(n)$ per A rispondendo alle sue query. Quindi oltre all'oracolo di firma $Sign_{sk}(\cdot)$ avremo anche l'oracolo H con cui l'avversario può interagire. L'avversario può mandare una query per il messaggio m all'oracolo H così da ricevere H(m) per poi inviare questo come query all'oracolo di firma così da avere $\sigma = Sign_{sk}(H(m))$.
Caso semplice, solo chiave pubblica. L'avversario A invia solo query a H. A' risponde alle query come segue:
Su input (N, e, y), dove rispettivamente sono la chiave pubblica, l'esponente e la challenge di cui A' dovrebbe essere in grado di calcolare la pre-immagine x, A' manda in esecuzione la subroutine A e per tutte le query, tranne una, che A effettua a quello che pensa essere l'oracolo H, A' risponde scegliendo valori casuali in Z_N^* , quando riceve m_1 risponde con r_1(H(m_1)). A' opera in questo modo tranne nel"i-esimo caso, il valore i lo sceglie uniformemente a caso prima di mandare in esecuzione A, dove invece di selezionare un valore casuale, prende la sua Challenge, e la manda come u=(H(m_i)) all'avversario A. L'obiettivo di A' è quello di scommettere sull'i-esimo messaggio in cui l'avversario A produrrà una

contraffazione. Se la scommessa va a buon fine, l'avversario A calcola un x tale che $x^e = y$. Quindi A risolve per A' il problema del calcolo della pre-immagine, ovvero **l'inversione di RSA**.
Poiché H è un oracolo casuale, i valori H(m) sono uniformemente distribuiti. Per cui visto che A' sceglie gli r_j con $j \neq i$ uniformi, A non nota alcuna differenza sulla distribuzione dei valori che riceve. Per quanto riguarda il caso della query i-esima, poichè x è scelto uniformemente nell'esperimento che definisce il problema RSA, y è distribuito uniformemente a caso, per cui A non si accorge di nulla. Pertanto, A produce una contraffazione (m, σ) su un messaggio m i cui ha chiesto H(m) con probabilità $\frac{1}{q}$, in qunato A' sceglie i in maniera uniforme, questo messaggio m è m_i. Pertanto, se A' dà in output σ come soluzione al problema RSA, risulta:

$$\sigma^e = H(m) = H(m_i) = y \text{ mod N} \tag{13.7}$$

Con probabilità che l'avversario A' ha scelto esattamente l'indice i, che corrisponde al messaggio per il quale A produce una contraffazione per un qualcosa di non trascurabile:

$$\frac{1}{q} \cdot \text{ non-negl(n)} \tag{13.8}$$

Quindi con questa probabilità, l'avversario A' è in grado di invertire RSA perché il valore σ he viene dato come output è la pre-immagine di y. D'altra parte. A' è efficiente se A è efficiente e risolvere il problema RSA con probabilità non trascurabile.
Caso dell'oracolo di firma:
La dimostrazione deve essere estesa in modo da considerare il caso in cui l'avversario A utilizzi anche l'oracolo di firma. In questo caso, A' dato che non conosce l'esponente d, non avrà la possibilità di firmare i messaggi richiesto dal calcolo:

$$\sigma = H(m_i)^d \text{ mod N} \tag{13.9}$$

Quindi per simulare l'oracolo di firma, l'avversario A', per m_j, sceglie un valore casuale σ_j e calcola $\text{H}(m_j) = \sigma_j^e$ mod N dove e è l'esponente pubblico. Da notare che σ_j è uniforme quindi si ha che $\text{H}(m_j) = \sigma_j^e$ mod N è uniforme. Quindi A', alla prima query per m_j indirizzata sia all'oracolo H che per l'oracolo Sign, genera e memorizza la tripla $(m_j, \sigma_j, \sigma_j^e)$ che sono rispettivamente m, H(m)^d, firma e H(m). Con questa strategia A' in grado di rispondere consistentemente ad entrambi i tipi di query. Dopodiché se la query che riceve è per un hash, invia σ_j^d, e invece riceve una query di firma per l'hash σ_j^d va a ricercare nella tabellina prendendo così la firma associata al messaggio che ha precomputato precedentemente. Ancora una volta, l'avversario A non si accorge di nulla perché l'hash che riceve in risposta alle query sono distribuiti uniformemente e le firme che ottiene successivamente sono consistenti. Pertanto, A' in grado con probabilità non trascurabile più piccola della probabilità di contraffazione di invertire la permutazione RSA.

13.1.2 Probabilistic Signature Scheme (RSA-FDH randomizzato)

La particolarità di questi schemi è che ogni messaggio può avere più firme.

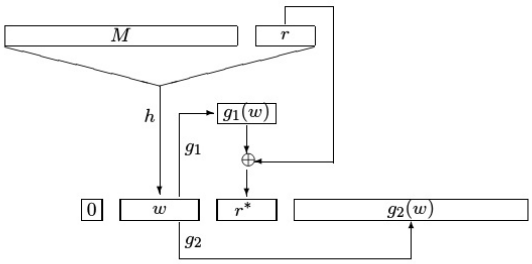


Figure 1: PSS: Components of image $y = 0 \parallel w \parallel r^* \parallel g_2(w)$ are darkened. The signature of M is $y^d \bmod N$.

Algoritmi di firma e verifica

SignPSS (M)
$r \xleftarrow{R} \{0,1\}^{k_0}$; $w \leftarrow h(M \parallel r)$; $r^* \leftarrow g_1(w) \oplus r$
$y \leftarrow 0 \parallel w \parallel r^* \parallel g_2(w)$
return $y^d \bmod N$

VerifyPSS (M, x)
$y \leftarrow x^e \bmod N$
Break up y as $b \parallel w \parallel r^* \parallel \gamma$. (That is, let b be the first bit of y, w the next k_1 bits, r^* the next k_0 bits, and γ the remaining bits.)
$r \leftarrow r^* \oplus g_1(w)$
if ($h(M \parallel r) = w$ and $g_2(w) = \gamma$ and $b = 0$) **then return** 1
else return 0

Perchè usare PSS?
La riduzione di sicurezza per RSA-FDH non è stretta.
Informalmente se A rompe Π in tempo t con probabilità ϵ => A' risolve X in tempo $\approx t$ con probabilità $\approx \epsilon$ e quindi la riduzione è stretta.

Nella pratica: abbiamo N modulo di K bit, se sappiamo che X richiede tempo t per una probabilità di successo ϵ, allora ogni A di tempo t rompe Π con probabilità di successo al più ϵ:

$$\epsilon(K) = \frac{t(K)}{2^{K/4}} \tag{13.10}$$

Quindi se RSA (t' ϵ')-sicuro => RSA-FDH(t, ϵ)-sicuro dove:

$$\begin{cases} t(K) = t'(K) - [q_h + q_s] \cdot K^{\epsilon} \\ \epsilon(K) = q_h + q_s] \cdot \epsilon'(K) \end{cases} \tag{13.11}$$

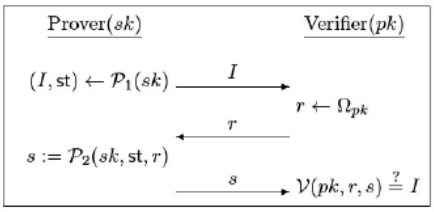
Per rendere il valore di $\epsilon(K)$ piccolo si deve scegliere K sempre più grande.

Confronto tra RSA-FDH e PSS:

- la riduzione del problema non è stretta;
- PSS è stato introdotto con l'obiettivo di ottenere una riduzione stretta;
- Una riduzione migliore per RSA-FDH è stata prodotta qualche anno dopo: RSA e Rabin, Gennaro-halevi-Rabin.

13.2 Schemi di identificazione

Uno schema di identificazione a chiave pubblica è un protocollo interattivo che permette ad una parte di provare la propria identità, chiamata Prover, ad un'altra parte, chiamata Verifier. La struttura generale di uno schema di identificazione in tre round funziona come segue:



Il Prover ha in input la propria chiave privata, mentre il Verifier ha in input la chiave pubblica del Prover. Nel primo round del protocollo, il Prover attraverso l'algoritmo P_1 e utilizzando la chiave privata, produce un valore I e un'informazione di stato st. Il valore I viene trasmesso al Verifier dove sceglie uniformemente a caso all'interno di un dominio, che è definito in qualche modo dalla chiave pubblica, un valore casuale r di cui lo possiamo interpretare come una sorta di Challenge. Questo inviato nel secondo round al Prover in cui utilizzando l'algoritmo P_2 che prende come parametri la chiave privata sk, st e r produce una risposta di sfida s che viene inviata al Verifier. Quest'ultimo, utilizzando la chiave pubblica pk, r e s e attraverso il proprio algoritmo di verifica V, se il risultato di questo è uguale al valore I, inviato al primo round, l'identificazione è avvenuta correttamente altrimenti si è verificato qualche problema.

13.2.1 Sicurezza dello schema

Per la correttezza di uno schema di identificazione, un eventuale avversario non avendo a disposizione della chiave privata non può essere in grado di farsi identificare da V, ovvero non può impersonare P. D'altra parte, l'avversario, prima di tentare di indentificarsi da V, potrebbe ascoltare diverse esecuzioni del protocollo tra P e V con lo scopo di acquisire ulteriori informazioni. Quindi si dà all'avversario un oracolo al quale può chiedere trascrizioni della comunicazione, indicando con $Trans_{sk}$ l'oracolo che, invocato senza input, esegue il protocollo tra due parti e restituisce all'avversario la trascrizione dei messaggi (transcript) che si sono scambiati, ovvero (I, r, s).

Supponiamo che $\Pi = (Gen, P_1, P_2, V)$ sia uno schema di identificazione e A un avversario PPT, l'esperimento $Ident_{A,\Pi}$ è definito come segue:

1. Viene eseguito Gen(1^n) per ottenre le chiavi (pk, sk);
2. L'avversario A prende pk e ha accesso all'oracolo Trans_{sk} che può interrogare tutte le volte che vuole;
3. In qualsiasi momento durante l'esperimento, A dà in output un messaggio I. Viene scelto in modo uniforme $r \in \Omega_{pk}$ e dato ad A, che risponde con alcune s (A può continuare a interrogare Trans anche dopo aver ricevuto r);
4. L'output dell'esperimento è 1 se e solo se V(pk, r, s) = 1.

Definizione 12.8: Uno schema di identificazione $\Pi = (Gen, P_1, P_2, V)$ è sicuro contro un attacco passivo, o semplicemente sicuro se per ogni avversario PPT, se esiste una funzione trascurabile negl tale che:

$$Pr[Ident_{A,\Pi}(n) = 1] \leq negl(n) \tag{13.12}$$

Possono essere considerati anche Adv attivi, ma nello schema non vengono richiesti per produrlo.

13.2.2 Fiat e Shamir

La trasformazione di Fiat e Shamir offre un metodo per convertire uno schema di identificazione interattivo in uno schema di firma non interattivo. L'idea è questa: il Prover esegue il protocollo di identificazione da solo, rimuovendo l'interazione usando una funzione hash, cioè la challenge che di solito si riceve dal Verifier viene generato in maniera autonoma dal Prover tramite la funzione hash.

Costruzione 12.9: Sia (Gem_{id}, P_1, P_2, P_1) uno schema di identificazione, uno schema di firma digitale è costruito come segue:

- Gen: prende in input 1^n, esegue $\text{Gen}_{id}(1^n)$ ottenendo le chiavi (pk, sk). La chiave pubblica pk specifica l'insieme di challenge Ω_{pk}. La parte che genera la chiave specifica la funzione H: $\{0,1\}^* \to \Omega_{pk}$, ma non ha un grosso impatto;
- Sign: prende in input la chiave privata sk e un messaggio m $\in \{0,1\}^*$, e :
  1. Computa (I, st) $\leftarrow P_1(sk)$;
  2. Computa $r := H(I, m)$; (passo in cui l'interazione viene rimossa)
  3. Computa s := $P_2(sk, st, r)$

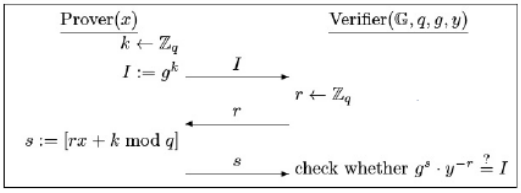
  Manda in output la firma (r,s)
- Vrfy: rende in input la chiave pubblica pk, un messaggio m e la firma (r,s), computa I := V(pk, r, s) e manda in output 1 se e solo se H(I, m) = r.

Si sta utilizzando lo schema di identificazione per produrre come firma i valori r e s dove r, che rappresenta la challenge, è stata calcolata rimuovendo l'interazione con il Verifier e utilizzando una funzione hash avente come input il messaggio che si vuole firmare e il valore I prodotto nel primo passo dello schema di identificazione.
Una firma (r, s) è legata ad un messaggio specifico m perché r è una funzione sia di I che di m quindi ogni volta che si va a cambiare m si ottiene un r diverso. Inoltre, se H viene modellata come un oracolo casuale che mappa gli input uniformemente su Ω_{pk}, allora r è uniformemente distribuito. Pertanto, possiamo concludere che per l'avversario è tanto difficile trovare una firma valida (r,s) su un messaggio m quanto lo sarebbe impersonare il Prover in una esecuzione onesta dal protocollo di identificazione per il motivo che in entrambi i casi il valore r viene scelto uniformemente a caso.

Teorema: Sia Π uno schema di indentificazione, Sia Π′ uno schema di firma ottenuto da Π applicando la trasformazione di Fiat e Shamir. Se Π è sicuro e H è un ROM => Π′ è sciuro.

13.2.3 Schema di identificazione di Schnorr

Il Prover, utilizzando un algoritmo di gruppi ciclici G(1^n) genera una tripla (G, q, g) dove G è un gruppo, q ordine del gruppo e g generatore, per poi scegliere $x \in Z_q$ uniformemente a caso. infine calcola y = g^x. La chiave pubblica e privata sono pk=(G, q, g, xy e sk =(G, q, g, x):



Il Prover sceglie un valore k uniformemente a caso per poi calcolare I := g^k. I è un elemento distribuito in maniera uniforme all'interno del gruppo in quanto g viene scelto uniformemente a caso. Il Verifier, una volta ricevuto I, sceglie un valore r uniformemente a caso per poi inviarlo al Prover. Quest'ultimo risponde con un valore s := $[rx+k \bmod q]$ dove x è la chiave privata. Il Verifier verifica che $g^s \cdot y^{-r}$ sia uguale a I ottenuto al primo passo e y^{-r} rappresenta l'inverso moltiplicativo. Questo funziona perché:

$$g^s \cdot y^{-r} = g^{rx+k} \cdot (g^x)^{-r} = g^{rx+k-rx} = g^k = I \tag{13.13}$$

Per quanto riguarda la sicurezza, quando l'avversario gioca può saltare la fase di interrogazione dell'oracolo, usato nell'esperimento $\text{Ident}_{A,\Pi}(n)$, tentare direttamente di identificarsi, perché l'avversario può simulare transcript di esecuzioni oneste del protocollo da solo, sfruttando soltanto la chiave pubblica e non conoscendo la chiave privata.
Idea: può essere effettuata invertendo l'ordine dei passi, quindi potrebbe scegliere prima indipendentemente ed uniformemente a caso i valori r,s $\in Z_q$ per poi calcolare I= $g^s \cdot y^{-r}$. Questo è possibile in quanto, le differenze di distribuzione tra il caso in cui l'avversario utilizza Trans e tra l'avversario in cui genera da solo le trascrizioni, le triple generate sarebbero distribuite esattamente allo stesso modo. Quindi dal punto di vista dell'avversario non ci sarebbe alcuna differenza e ciò significa che l'avversario può tranquillamente evitare di interrogare l'oracolo.

Supponiamo ora un avversario che dispone della chiave pubblica y in cui invia I al Verifier da cui riceverà come risposta r. Da qui risponderà con s tale che $g^s \cdot y^{-r} = I$.
Se l'avversario fosse in grado di calcolare valori di s giusti efficientemente e con alta probabilità, allora sarebbe in grado di calcolare risposte corrette s_1 e s_2 ad almeno due sfide $r_1, r_2 \in Z_q$. Ma questo significa che l'avversario sarebbe in grado di calcolare il logaritmo discreto, quindi la chiave segreta, perché per come è stata definita l'equazione di verifica:

$$g^{S_1} \cdot y^{-r_1} = I = g^{S_2} \cdot y^{-r_2} \rightarrow g^{S_1 - S_2} = y^{r_1 - r_2} \tag{13.14}$$

Da questo risultato si può ricavare che l'avversario può calcolare esplicitamente, moltiplicando:

$$g^{(S_1-S_2)(r_1-r_2)^{-1}} y^{(r_1-r_2)(r_1-r_2)^{-1}} \rightarrow y = g^{(S_1-S_2)(r_1-r_2)^{-1}}$$
$$\rightarrow \log_g y = [(S_1 - S_2)(r_1 - r_2)^{-1} \bmod q]$$

ovvero il logaritmo discreto di y, contraddicendo la difficoltà del problema DL.

Teorema: Se DL è difficile in G allora lo schema di Schnorr è sicuro.

Essendo DL difficile e utilizzando la trasformata di Fiat-Shamir, si può produrre lo schema di firma di Schnorr.
Costruzione 12.12 Sia:

- Gen: Esegue G(1^n) ottenendo (G, q, g). Sceglie in modo uniforme x $\in Z_q$ e imposta y := g^x. La chiave privata è x e la chiave pubblica è (G,q,g,y). La parte che genera la chiave specifica la funzione H:$\{0,1\}^* \to Z_q$ ma non ha un grosso impatto;
- Sign: prende in input la chiave privata x e un messaggio $m \in \{0,1\}^*$, sceglie in modo uniforme k$\in Z_q$ e imposta I := g^k. Poi computa r := H(I, m), seguito da s:= $[rx+k \bmod q]$. Da in output la firma (r, s);
- Vrfy: prende in input la chiave pubblica (G,q,g,y), un messaggio m e la firma (r,s), computa I = $g^s \cdot y^{-r}$ e dà in output 1 se H(I, m) = r.

13.3 DSA (Digital Signature Algorithm) ed ECDSA (Elliptic Curve Digital Signature Algorithm)

DSA e ECDSA sono anche loro come sappiamo basati sul problema DL anche se su classi di gruppi differenti. Quindi possono essere visti come costuiti da uno schema di identificazione:

1. Il Prover ha sk =(x, y=g^x) e pk =(G,q,g,y). Sceglie k $\in Z_q$ scelto uniformemente a caso, calcola I = g^k e lo invia al Verifier;
2. Il Verifier sceglie $\alpha, r \in Z_q$ uniformemente a caso e lo invia come challenger al Prover;
3. Il Prover calcola S := $[k^{-1} \cdot (\alpha + xr) \bmod q]$ e lo invia al Verifier;
4. Il Verifier accetta se $S \neq 0$ e $g^{\alpha S^{-1}} \cdot y^{r \cdot S^{-1}} = I$.

Si noti che risulta S $\neq 0$ a meno che $\alpha = -xr$ mod q che accade con probabilità trascurabile.

Pertanto S^{-1} esiste e risulta:

$$\begin{aligned} g^{\alpha S^{-1}} \cdot y^{r \cdot S^{-1}} &= g^{\alpha S^{-1}} \cdot (g^x)^{r \cdot S^{-1}} = g^{(\alpha + xr) S^{-1}} \\ &\text{Poniamo } s = K^{-1} \cdot (\alpha + xr) \bmod q) \text{ allora abbiamo} \\ &= g^{(\alpha+xr)(\alpha+xr)^{-1} \cdot K} = g^K = I \end{aligned}$$

Quindi lo schema è corretto.
Si può dimostrare sicuro se il problema DL è difficile elativamente a f(1^n).
Sketch della prova:

- Transcript di esecuzioni oneste possono essere simulate scegliendo uniformemente a caso: $\alpha, r \in Z_q$, $s \in Z_q^*$, I = $g^{\alpha S^{-1}} \cdot y^{r \cdot S^{-1}}$
- Se Adv dà in output I per cui può dare risposte corrette $S_1, S_2 \in Z_q^*$ al challenger in coppie distine $(\alpha, r_1)(\alpha, r_2)$ allora:

$$\begin{aligned} & g^{\alpha \cdot S_1^{-1}} \cdot y^{r_1 \cdot S_1^{-1}} = I = g^{\alpha S_2^{-1}} \cdot y^{r_2 \cdot S_2^{-1}} \\ & => g^{\alpha(S_1^{-1} - S_2^{-1})} = y^{r_2 S_2^{-1} - r_1 S_1^{-1}} \\ & => \text{ posso quindi calcolare } \log_g Y \end{aligned}$$

Vale anche per challenge (α_1, r_1), (α_2, r_1)

Gli schemi di firma DSA/ECDSA sono costruiti facendo collassare lo schema di identificazione in un algoritmo non-interattivo eseguito dal firmante.
Rispetto alla trasformazione di Fiat e Shamir, operiamo come segue:

- α = H(m), m è un messaggio e H una funzione hash;
- r = F(I), F : $F \to Z_q$, F una funzione semplice

In DSA, G è un sottogruppo di ordine q di Z_P^* dove p è primo, F(I) $\stackrel{\text{def}}{=}$ [I mod q].
In ECDSA, G è un sottogruppo di ordine q di una curva ellittica E(Z_P), p primo con $F[(x,y)] \stackrel{\text{def}}{=} [xmodq]$. **NOTA:** (x,y) rappresenta un punto della curva.

Entrambi gli schemi possono essere descritti in modo stretto come segue:

COSTUZIONE 12.13:

- Gen: esegue $G(1^n)$ per ottenere (G, q, g). Sceglie uniformemente x $\in Z_q$ e y := g^x. La chiave pubblica è (G, q, g, y) e la chiave privata x.
  Due funzioni fanno parte della generazione delle chiavi: H:$\{0.1\}^* \to Z_q$ e F: G $\to Z_q$;
- Sign: prende in input la chiave privata e il messaggio m $\in \{0,1\}^*$, sceglie uniformemente a caso $k \in Z_q^*$ e r:= F(g^k). Calcola s:= $[k^{-1} \cdot (H(m) + xr) \text{ mod q}]$. (Se r = 0 o s = 0 si riparte con un k). Da in output la firma (r, s);
- Vrfy: Prende in input la chiave pubblica (G, q, g, y) e un messaggio m $\in \{0,1\}^*$, e la firma (r, s) con r, s $\neq$ 0 mod q, da in output 1 se e solo se

$$r \stackrel{?}{=} F(g^{H(m)\cdot s^{-1}} y^{r\cdot s^{-1}} \tag{13.15}$$

Se il problema DL è difficile relativamente a f(1^n) ed H ed F sono modellati come oracoli casuali, allora la costruzione generica è sicura.

Osservazioni: Il valore K $\in Z_q^*$ deve essere scelto uniformemente a caso. Se una sorgente povera di randomness porta ad un K predicibile, Adv può calcolare da (r, s) la chiave privata x!

Infatti preso

$$S = K^{-1}(H(m) + xr) \text{ mod q} \tag{13.16}$$

S, H(m) e r sono sempre noti, se lo fosse anche K ci rimane come unica incognita x. Basta che K sia stato usato per produrre 2 firme.
Siano quindi (r, S_1) e (r, S_2) le firme per m_1 ed m_2 si può procedere come segue:

$$\begin{aligned}
&S_1 = K^{-1}(H(m) + xr) \text{ mod q}\\
&S_2 = K^{-1}(H(m) + xr) \text{ mod q}\\
&\Rightarrow S_1 - S_2 = K^{-1}(H(m_1) - H(m_2)) \text{ mod q}\\
&\Rightarrow \text{K può essere calcolato}\\
&\Rightarrow \text{x può essere calcolato}
\end{aligned}$$

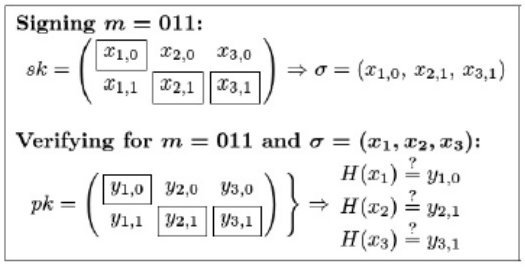
Attacco applicando ralmente per ottenrer la masterr key della PS3 nel 2010.

13.4 Firme digitali tramite funzioni hash

Uno schema di firma digitale può essere ottenuto unando funzioni hash crittografiche, senza necessità di assunzioni di teoria dei numeri.

13.4.1 Schema di Lamport

Sicuro per un singolo uso (one-time signatue)



L'algoritmo funziona nel seguente modo:

- La chiave privata è costruita su due righe e n colonne quando la lunghezza del messaggio che si vuole inviare, quando si vuole firmare un messaggio per ogni bit si sceglie la cella corrispondente come riportato sopra e li si dà in output.

- La chiave pubblica è costruita nello stesso modo ma inserendo nelle celle i valori hash delle firme, quando si vuole controllare la firma del messaggio ricevuto, si effettua il controllo con l'hash dei bit della firma ricevuta con quelli della chiave pubblica ricevuta .

Definiamo l'esperimento onte-time signature Sig-forge$^{1-time}_{A,\Pi}$ come segue:

1. Si esegue Gen(1^n) per ottenere le chiavi pk e sk;
2. Adv A prende in input pk e chiede una singola query m' al suo oracolo di firma Sign$_{sk}(\cdot)$. A da in output (m, σ) insiema a m $\neq m'$;
3. L'output dell'esperimento è 1 se e solo se Vrfy(m, $\sigma == 1$

Definizione 12.14: Uno schema di firma $\Pi = (Gen, Sign, Vrfy)$ è sicuro alla contraffazione verso un attacco a singolo messaggio, oppure è uno schema di firma one-time sicuro se per ogni Adv A PPT esiste una funziona trascurabile tale che:

$$Pr[\text{Sig-forge}^{1-time}_{A,\Pi}(n) = 1] \leq negl(n) \tag{13.17}$$

Sia H: $\{0.1\}^* \to \{0.1\}^*$ una funzione. Costruiamo uno schema di firma per messaggi di lunghezza l = l(n) come segue:

- Gen: Su input 1^n procede come segue: per i $\in \{1, ..., l\}$
  1. Sceglie uniformemente $x_{i,0}, x_{i,1} \in \{0,1\}^n$;
  2. Calcola $y_{i,0} := H(x_{i,0}$ e $y_{i,1} := H(x_{i,1}$

  La chiave pubblica pk e la chiave privata sk sono

$$sk = \begin{pmatrix} x_{1,0} & x_{2,0} & ... & x_{l,0} \\ x_{1,1} & x_{2,1} & ... & x_{l,1} \end{pmatrix} pk = \begin{pmatrix} y_{1,0} & y_{2,0} & ... & y_{l,0} \\ y_{1,1} & y_{2,1} & ... & y_{l,1} \end{pmatrix} \tag{13.18}$$

- Sign: prende in input sk e un messaggio $m \in \{0,1\}^l$, con m = $m_1...m_l$ da in output la firma $(x_{1,m_1}, ..., x_{l,m_l})$;
- Vrfy: prende in input pk e un messaggio $m \in \{0,1\}^l$, con m = $m_1...m_l$ e la firma $\sigma = (x_1, ..., x_l)$ da in output 1 se e solo se H(x_i) = $y_{i,mi}$ per ogni 1 $1leqi \leq$l.

Teorema: Se H è one-way allora la costruzione è sicura.

Capitolo 14

Prova a conoscenza zero e condivisione di segreti

14.1 Prova a conoscenza zero

In crittografia una prova a conoscenza zero è un metodo interattivo utilizzato da un provatore per dimostrare ad un verificatore che una affermazione (solitamente matematica) è vera, senza rivelare nient'altro oltre alla veridicità della stessa.
Una dimostrazione a conoscenza zero deve soddisfare tre proprietà:

- **Completezza**: se l'affermazione è vera, un provatore (P) onesto potrà convincere del fatto un verificatore (V) onesto (cioè chi segue esattamente il protocollo);
- **Correttezza**: se l'affermazione è falsa, nessun provatore imbroglione potrà convincere il verificatore onesto che essa è vera, la probabilità di riuscirci è molto bassa;
- **Conoscenza zero**: se l'affermazione è vera, nessun verificatore imbroglione potrà sapere altro che tale informazione. Questo viene provato mostrando che a ogni verificatore imbroglione viene associato un simulatore da cui può ricavare un transcript per l'affermazione vera, e che l'output dato dal simulatore è indistinguible da quello prodotto dal provatore e dal verificatore.

Un sistema a conoscenza zero permette al verificatore di acquisire un unico bit di conoscenza. Se consideriamo un transcript reale e un transcript simulato, costruito attraverso un esperimento mentale da parte del verificatore, si ha che dal punto di vista del verificatore, i transcript che ha generato sono distribuiti esattamente come nel protocollo reale. In sintesi, il verificatore riesce da solo a costruirsi l'interazione con il provatore, nel caso in cui il teorema fosse vero.
Se il verificatore è onesto riesce a costruire da solo ciò che vedrebbe in una interazione reale, allora nell'interazione non c'è nulla che gli permetta di acquisire ulteriore conoscenza.

Formalizziamo:

Un sistema di prova a conoscenza zero è una coppia (P, V) dove P ha potere computazionale illimitato, mentre V ha potere computazionale polinomiale.
Devono essere soddisfatte 3 proprietà:

- Completeness: Se il teorema è verificato allora si potrà convincere il verificatore;
- Soundness: Se il teorema è falso allora non si riuscirà a convincere il verificatore;
- Zero Knowledge:$\forall$ V* (anche malizioso) PPT $\exists$ S (simulatore) PPT tale che $\forall$ teorema vero X deve accadere che il transcript generato dalla coppia <P, V*> deve essere indistinguibile dal transcript prodotto dal simulatore S(X).

Le prove a conoscenza zero sono molto utili per dimostrare soluzioni a problemi nella classe NP. È stato anche dimostrato che possono essere utilizzati per provare qualsiasi teorema.

14.2 Condivisione di segreti

Sfruttano la distribuzione dei polinomi:

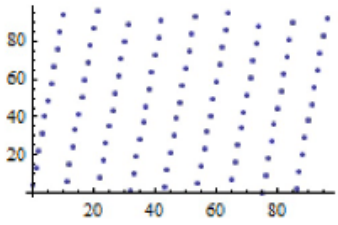


Grafico di una retta (mod 97)

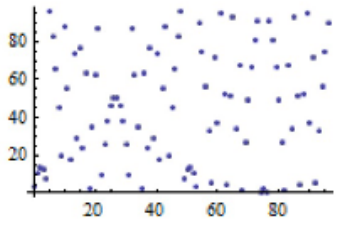


Grafico di una parabola (mod 97)

Matematicamente il loro comportamento è simile ad un polinomio.
Un polinomio che ha grado d ha al più d radici. Un polinomio di grado d-1 è univocamente determinato da d punti distinti:

$$(x_0, f(x_0))...(x_{d-1}, f(x_{d-1})) \qquad f(x) = a_0 + a_1 x + ...a_{d-1}x^{d-1} \tag{14.1}$$

Possiamo scrivere il polinomio sotto forma di una matrice di Vandermonde (non singolare), e le incognite sotto forma di un vettore; il loro prodotto darà in output un vettore di valori noti:

$$\begin{pmatrix} 1 & x_0 & ... & x_0^{d-1} \\ 1 & x_1 & ... & x_1^{d-1} \\ . & ... & ... & . \\ . & ... & ... & . \\ . & ... & ... & . \\ 1 & ... & ... & x_{d-1}^{d-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ . \\ . \\ . \\ a_{d-1} \end{pmatrix} = \begin{pmatrix} f(x_0) \\ f(x_1) \\ . \\ . \\ . \\ f(x_{d-1}) \end{pmatrix} \tag{14.2}$$

Il polinomio interpolato può essere calcolato più efficientemente ed espresso attraverso la formula di Langrage:

$$f(x) = \sum_{i=0}^{d-1} f(x_i) \cdot L_i(x) \tag{14.3}$$

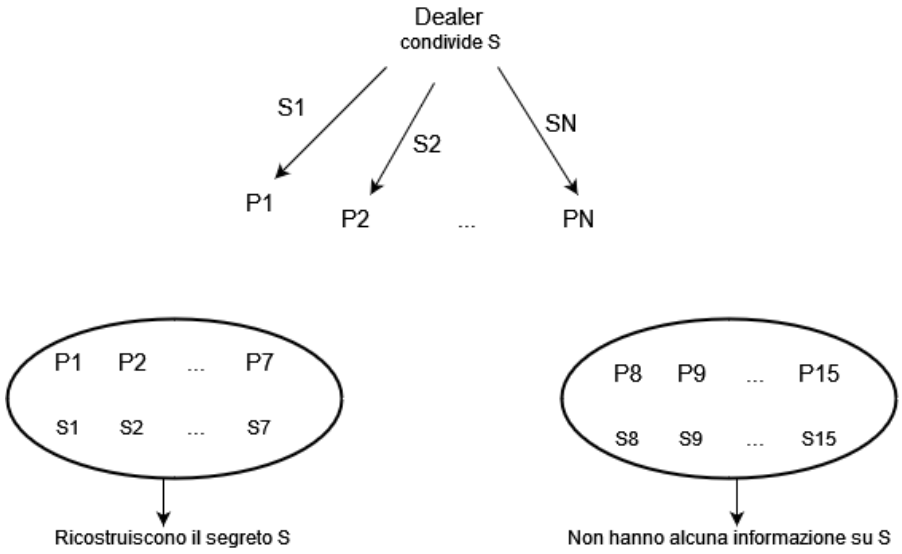
dove $L_i(x) = \prod_{j=0, j\neq i}^{d-1} \frac{(x-x_i}{(x_i-x_j)}$ $\quad x_i \neq x_j, \forall i \neq j$
NOTA:$L_i(x_i) = 1$ e $L_j(x_i) = 0 \quad \forall j \neq i$ => f(x) vale esattamente f(x_i) con ognuno degli x_i.

14.2.1 Funzionamento dello schema

Inizialmente abbiamo un **Dealer** che vuole condividere un segreto **S** con gli n partecipanti in modo tale che:

- un sottoinsieme di qualificati restituiscano S;
- un sottoinsieme di proibiti non deve avere nessun informazione su S

La reconstruction Phase può essere condotta o da partecipanti o tramite un combiner.



Formalizzazione:
S $\in \{0,1\}^n$ ed è una stringa di n bit. Il Dealer genera dei valori totalmente casuali $r_i \in \{0,1\}^n$ detti share, e li dà ai partecipanti. Il partecipante P_n avrà nella sua stringa casuale il seguente valore:

$$P_n = r_1 \oplus ... \oplus r_{n-1} \oplus S \tag{14.4}$$

Per ottenere S basta che i partecipanti mettano insieme le share:

$$S = r_1 \oplus r_2 \oplus ... \oplus r_{n-1}(r_1 \oplus ... \oplus r_{n-1} \oplus S) \tag{14.5}$$

Se manca solo uno dei partecipanti la maschera non può essere rimossa.
NOTA: Lo schema può essere facilmente riprodotto in altri gruppo al posto di $(\{0,1\}^n, \oplus)$, per esempio $(Z_m, +_m)$.

14.2.2 Forma generale dello schema

Nella forma più generale gli insiemi qualificati e proibiti di partecipanti definiscono una "struttura d'accesso" al segreto:
A = (Q, F) su P = $\{1, 2, ..., n\}$, dove:

- A è la struttura d'accesso;

- Q è la famiglia di insiemi qualificati;
- F è la famiglia di insiemi proibiti;
- P è l'insieme dei partecipanti.

Le strutture di accesso che ci interessano sono quelle monotone; cioè:

$$A \in Q, A \subset B => B \in Q \tag{14.6}$$

14.2.3 Schema di Shamir

Realizza una struttura d'accesso a soglia:

- Q = $\{S \subseteq \{1, ...n\} : |S| \geq t\}$: ogni sottoinsieme di almeno t partecipanti ricostruisce;
- F = $\{S \subseteq \{1, ...n\} : |S| < t\}$: ogni sottoinsieme di t_1 o meno partecipanti non ottiene alcun informazione.

Uno schema a soglia (t, n) è descritto come segue:
Sia s $\in Z_p$ (p primo, p > n) deve quindi avere almeno n punti distinti.

- **Sharing phase**: Il Dealer sceglie uniformemente a caso un polinomio a(x) di grado al più t-1 tale che a(0) = S. Per i = 1, ..., n, invia $S_i = a(i)$ ai partecipanti P_i;
- **Reconstruction phase**: Ogni sottoinsieme di t partecipanti Q ricostruisce il segreto delle proprie share usando l'interpolazione di Lagrange.

$$S = \sum_{i \in Q} s_i \cdot \lambda_{Q,i} \qquad \lambda_{Q,i} = \prod_{j \in Q \setminus \{i\}} \frac{i}{j - i} \tag{14.7}$$

NOTA: non ricostruisco a(x). Calcolo direttamente il suo valore in 0.

- **Correttezza**: l'interpolazione di Lagrange garantisce che ogni sottoinsieme qualificato ricostruisca S;
- **Sicurezza**: Cosa ci garantisce che un sottoinsieme di al più t-1 partecipanti non ottiene alcuna informazione su a(0)? Senza perdita di generalità consideriamo $\{P_1, ..., P_{t-1}$ interpolando i partecipanti con la loro share:

$$\begin{pmatrix} 1 & 0 & ... & 0 \\ 1 & 1 & ... & 1^{t-1} \\ 1 & 2 & ... & 22^{t-1} \\ . & ... & ... & . \\ . & ... & ... & . \\ 1 & t-1 & ... & (t-1)^{t-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ . \\ . \\ . \\ a_{t-1} \end{pmatrix} = \begin{pmatrix} S_0 \\ S_1 \\ . \\ . \\ . \\ S_{t-1}) \end{pmatrix} \tag{14.8}$$

  Per ogni scelta di $S_0 \in Z_p$ il sistema ammette un'unica soluzione perché i coefficienti $a_1, ..., a_{t-1}$ sono scelti uniformemente a caso, ogni valore di S_0 è equivalente probabile.

Osservazione: lo schema iniziale è uno schema a soglia (n, n). Può essere esteso per realizzare un (t,n):

- per ogni sottoinsieme di t partecipanti si usa uno schema (t,t) indipendente dagli altri.
- **Inefficiente**: P_i appartiene a $\binom{n-1}{t-1}$ sottoinsiemi e riceve $\binom{n-1}{t-1}$ share, una per sottoinsieme.

Shamir invece dà solo una share a P_i.
Lo schema iniziale però può essere utile per realizzare strutture d'accesso generali nel caso in cui non sappiamo fare di meglio.