Crittografia teoremi

Carmine D'Angelo

A.A. 2022/2023

1) Segretezza perfetta (Schema perfettamente segreto => |K| >= |M|): Si dimostri che in ogni schema di cifratura perfettamente segreto, l'insieme delle chiavi di cifratura deve avere cardinalità dell'insieme dei messaggi.

Teorema: Se (Gen, Enc, Dec) è uno schema di cifratura perfettamente segreto, con spazio dei messaggi M e delle chiavi K, allora |K| ≥ |M|.
Dim: Mostriamo che se fosse |K| ≤ |M|, lo schema non potrebbe essere perfettamente segreto. Sia quindi |K| ≤ |M|, sia M distribuito uniformemente e sia $c \in C$ tale che Pr[$C = c$]>0.

Def. M(c) = $\{m | m = Dec_k(c), \text{per qualche} k \in K\}$, chiaramente |M(c)| ≤ |K|. Se |K| < |M|, allora $\exists m' in M(c)$ tale che m' ∉ M. Ma allora risulta $Pr[M = m'|C = c] = 0 \neq Pr[M = m']$.
Pertanto lo schema non è perfettamente segreto. Quindi |K| ≥ |M|.

1.1) Inoltre si spieghi sotto quale condizione lo schema di Vigenere risulta perfettamente segreto. .
Lo schema di Vigenere è perfettamente segreto se il testo è senza ridondanza, la chiave è lunga e casuale (cambiare spesso la chiave)
2) Segretezza Perfetta. Si si spieghi cosa si intende per schema di cifratura "perfettamente segreto". Inoltre si formalizzi la nozione, discutendo le tre forme equivalenti che sono state presentate a lezione.

Con uno schema di cifratura perfettamente segreto quello che vogliamo ottenere è che un Adv che osserva un cifrato non dovrebbe ottenere nessuna informazione aggiuntiva riguardando il messaggio in chiaro. In poche parole il cifrato non deve rivelare nulla al di là di quello che già si sà.

Le tre forme equivalenti presentate a lezione sono le seguenti:

Definizione 1: 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 usato lo schema di cifratura, si richiede che potrà essere usato 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:

$$\mathbf{Pr[M = m \mid C = c] = Pr[M = m]} \tag{1}$$

Definizione 2 (Lemma 2.4): Possiamo fornire una definizione equivalente richiedendo che la distribuzione di probabilità dei cifrati non dipenda dal messaggio in chiaro.
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 $\forall c \in C$:

$$Pr[Enc_k(m) = c] = Pr[Enc_k(m') = c] \tag{2}$$

Questo ci dice che è impossibile distinguere una cifratura di un messaggio m da m' perché la distribuzione dei cifrati è la stessa. Un cifrato non rivela alcuna informazione sul messaggio in chiaro

Definizione 3: Si basa sull'esperimento che coinvolge Adv e Challenger C. 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';
- 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 $\frac{1}{2}$.
Quindi possiamo dire che:
Uno schema di cifratura (Gen,Enc,Dec) con spazio dei messaggi M è perfettamente indistinguibile se, per ogni A (non efficiente o efficiente), risulta: $Pr[Priv^{eav}_{A,\Pi} = 1] = \frac{1}{2}$
Uno schema di cifratura Π è perfettamente segreto se e solo se è perfettamente indistinguibile.

3) Segretezza Perfetta: Si dimostri che in ogni schema di cifratura perfettamente segreto, l'insieme delle chiavi di cifratura deve avere cardinalità maggiore o uguale alla cardinalità dell'insieme dei messaggi.
Risposta uguale alla domanda 1.

3.1) (Teorema 2.10-Shannon) Si spieghi perchè il one-time pad risulta insicuro rispetto alla trasmissione di messaggi multipli, per qualsiasi nozione significativa di sicurezza rispetto a messaggi multipli.

Se (Gen,Enc,Dec) è uno schema di cifratura perfettamente segreto con spazio dei messaggi M e spazio delle chiavi K, allora $|K| \geq |M|$.
Dim: Mostriamo che se fosse $|K| < |M|$ lo schema non potrebbe essere perfettamente segreto.
Sia $|K| < |M|$, sia 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:

$$Pr[M = m'|C = c] = 0 \neq Pr[M = m']. \tag{3}$$

Per lo schema non è perfettamente segreto. Quindi deve essere $|K| \geq |M|$.

Costruiamo ora il otp:
Sia $l > 0$ un intero. Siano M = K = C = $\{0,1\}^l$.

- Gen: sceglie $k < -\{0,1\}^l$
- Enc: dati $k \in \{0,1\}^l$ ed $m \in \{0,1\}^l$ manda in output $c := k \oplus m$
- Dec: dati $k \in \{0,1\}^l$ ed $c \in \{0,1\}^l$ manda in output $m := k \oplus c$

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

Proposizione 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. 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 cifrati 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:

se b=0 allora $c_1 = c_2 \Rightarrow A$da0

se b=1 allora $c_1 \neq c_2 \Rightarrow A$da1

Pertanto A vince sempre con probabilità 1 e quindi non è sicuro secondo la definizione.

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 negl tale che:

$$Pr[PrivK^{cpa}_{A,\Pi}(n) = 1] \leq \frac{1}{2} + negl(n) \tag{4}$$

L'esperimento è il seguente:

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

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.

Se uno schema di cifratura è deterministico, la cifratura dello stesso messaggio dà lo stesso cifrato => richiesta cifratura probabilistica. Altrimenti Se Π è uno schema di cifratura Enc() deterministica allora M non può avere cifrature multiple indistinguibili in presenza di un eav.

3.2) Necessità del rilassamento.
Sebbene la sicurezza perfetta sia un obiettivo ideale, spesso non è pratico o possibile da raggiungere nei sistemi crittografici reali. Uno dei possibili problemi è la distribuzione delle chiavi infatti il mittente e il destinatario devono condividere una chiave segreta lunga almeno quanto il messaggio da inviare. Questa chiave deve essere mantenuta completamente segreta e al riparo dagli attacchi. La distribuzioni delle chiavi può essere anche molto costosa. Un altro motivo è l'efficienza computazionale, è richiesta una chiave casuale lunga almeno quanto il messaggio da cifrare. La generazione e la distribuzione di tali chiavi possono essere impraticabili, specialmente per messaggi di grandi dimensioni o comunicazioni a lungo termine.

4) Sicurezza Semantica e Indistinguibilità: relazione di uguaglianza.
Per **sicurezza semantica** si intende la proprietà di uno schema Π in cui un attaccante non ottiene informazioni utili sul messaggio in chiaro leggendo il suo cifrato, anche se Adv ha accesso a più testi cifrati che utilizzano la stessa chiave K.
Per **Indistinguibilità** si intende la proprietà di uno schema crittografico Π in cui un Adv non riesce a distinguere tra due cifrati c che corrispondono a di versi messaggi in chiaro.

La **sicurezza semantica => indistinguibilità** questo perché se un Adv non può ottenere alcun informazione dal testo cifrato allora Adv non potrà nemmeno distinguere tra i vari cifrati che corrispondono a diversi messaggi in chiaro.
Non vale il contrario infatti uno schema che è indistinguibile potrebbe non essere necessariamente semanticamente sicuro. Ad esempio, uno schema di crittografia che restituisce sempre un testo cifrato fisso, indipendentemente dal messaggio in chiaro, sarebbe indistinguibile ma non semanticamente sicuro.

5) Che cosa è 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 $Dec_k(Enc_k(m)) = (k \oplus (k \oplus m)) = m$

5.1) Dimostrare che opt è perfettamente segreto Lo schema di cifratura one-time pad è perfettamente segreto.
Dim
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.

5.2) Segretezza Computazionale: Si spieghi perché è necessario, nella formulazione della nozione, rilassare le assunzioni utilizzate per la segretezza perfetta – avversari di potere illimitato e probabilità di errore zero – considerando, invece, avversari ppt e ammettendo probabilità di errore trascurabili. Inoltre, si diano le definizioni di schema di cifratura simmetrico CPA-sicuro e CCA-sicuro, rispettivamente.
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. Per ottenere una nozione di segretezza più debole ma utile nella pratica, sono stati definiti due aspetti:

- avversario computazionalmente quantificabile, 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.

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:

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

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

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

6) Generatore pseudocasuale: Si fornisca la definizione di generatore pseudocasuale.
Un generatore pseudocasuale G è un algoritmo deterministico efficiente per trasformare una stringa uniforme corta, chiamata seme, in una più lunga che sembra uniforme.
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:

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

6.1): Si consideri il seguente generatore: G:$\{0,1\}^{nm} \rightarrow \{0,1\}^{n(m+1)}$. Il generatore prende in input m stringhe $x_1, ..., x_m$ di n bit e dà in output

$$\text{G}(\text{x}_1, ..., \text{x}_m) = x_1...x_m y, \text{ dove y} = XOR_i x_i \tag{5}$$

dove y è ottenuto facendo l'XOR delle m stringhe $x_1, ..., x_m$ di n bit.

Il generatore non rispetta la proprietà dello pseudocasualità perché esiste un Distinguisher che è in grado di capire, data una stringa w di m+1 bit, se w è stata prodotta dal generatore oppure selezionata uniformemente a caso tra le stringhe di m+1 bit. Il Distinguisher stampa in output 1 se e solo se il bit finale di w è l'xor di tutti i bit precedenti ed è costruito come segue:

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

Poiché la differenza è

$$|Pr[D(G(s)) = 1] - Pr[D(r) = 1]| = |1 - \frac{1}{2}| = \frac{1}{2} \tag{6}$$

Dato che avremo come probabilità pari a $\frac{1}{2}$, si tratta di una funzione costante che è molto lontana da una funzione non trascurabile per cui G non è un generatore pseudocasuale.

6.2) Si consideri il seguente generatore G:$\{0,1\}^{nm} \to \{0,1\}^{n(m+1)}$.
Il generatore interpreta la stringa di input come la rappresentazione di **m** interi x_i di **n** bit e dà in output la rappresentazione degli stessi m interi più quella di un ulteriore intero y, dato dalla somma **mod** 2^n dei valori cx_i, diversi per i = 1, ... , n, il valore cx_i è il complemento mod 2^n di x_i. Precisamente

$$G(x_i, ..., x_m y) = x_i, ..., x_m y, \quad \text{dove t} = \sum_i cx_i \ \ mod\, 2^n \tag{7}$$

È G un generatore pseudocasuale? Si supporti la risposta con un argomento rigoroso.
Sia G:$\{0,1\}^{nm} \to \{0,1\}^{n(m+1)}$ tale che

$$G(x_i, ..., x_m y) = x_i, ..., x_m y, \quad \text{dove t} = \sum_i cx_i \ \ mod\, 2^n \tag{8}$$

Il distinguisher D su input ω da in output 1 se e solo se y è il complemento mod 2^n degli x_i.
Risulta:

- Se input di D è G(s), allora Pr$[G(s) = 1] = 1$;
- Se input di D è r, allora Pr$[G(r) = 1] = \frac{1}{2}$, la probabilità che l'ultimo bit sia uguale al complemento mod 2^n degli x_i precedenti è $\frac{1}{2}$ perché r viene scelto in modo uniforme su $\{0,1\}^n$

$$|Pr[G(s) = 1] - Pr[D(r) = 1]| = |1 - \frac{1}{2}| = \frac{1}{2} \tag{9}$$

Complemento (non trascurabile) G non è pseudocasuale perchè la probabilità non è trascurabile.
6.3 Cifratura sicura con PRG.
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:

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

7) Riduzioni: metodologia: Si descriva la struttura generale di una riduzione di sicurezza, evidenziando le motivazioni alla base dell'approccio e le proprietà che soddisfa.
Sia $\tilde{\Pi} = (\tilde{Gen}, \tilde{Enc}, \tilde{Dec})$ costruito a partire da $\Pi = (Gen, Enc, Dec)$

- $\tilde{\Pi}$ usa f $\in$ Func$_n$ scelta uniformemente a caso
- Π usa F_k dove k p scelta uniformemente a caso

Per ogni Adv A PPT sia q(n) tutte il numero di query che A(1^n) rivolge all'oracolo di cifratura.
Esiste una funzione trascurabile tale che

$$|Pr[PrivK^{cpa}_{A,\Pi}(n) = 1] - Pr[PrivK^{cpa}_{A,\tilde{\Pi}}(n) = 1]| \leq negl(n) \tag{10}$$

Usiamo A per costruire un distinguisher D per la funzione pseudocasuale F.

Se A ha successo => D distingue

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

Distinguisher D:
Prende in input 1^n e ha accesso all'oracolo O: $\{0,1\}^n -> \{0,1\}^n$:

1. Esegue A(1^n). A effettua query all'oracolo scegliendo m $\in \{0,1\}^n$, risponde:
   a Sceglie uniformemente r $\in \{0,1\}^n$;
   b Query O(r) e ottiene y;
   c Restituisce < r, y $\oplus$ m > ad A
2. Quando A da in output $m_0, m_1 \in \{0,1\}^n$, sceglie un bit uniforme b $\in \{0,1\}$:
   a Sceglie r $\in \{0,1\}^n$;
   b Query O(r) e ottiene y;
   c Restituisce il cifrato challange <r, y $\cdot$ m$_b$> ad A
3. A effettua query all'oracolo fino a quando da in output b'. Se b' = b da in output 1 , altrimenti 0.

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:

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

7.1) Caso di esempio. Se F è una funzione pseudocasuale, allora lo schema di cifratura che associa il cifrato:

$$c :=< r, f_k(r) \oplus m >, \qquad \text{al messaggio m} \tag{11}$$

dove r e la chiave k sono scelti **uniformemente** a caso , è uno schema di cifratura CPA-sicura.

Assumiamo che F è pseudocasuale, questo 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:

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

Nell'esperimento che stiamo considerando un messaggio 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:

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

- 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:

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

Pertanto Π è CPA-sicuro.
7.2) Caso di esempio: Si dimostri che se G è un PRG, allora lo schema di cifratura c = G(s) $\oplus m$, con s scelto uniformemente a caso e $\oplus$ a denotare l'operazione di XOR bit a bit, è EAV-sicuro.
Teorema 3.18 (Cifratura con PRG) 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 EAV.

Passi per la dimostrazione: Esistono due idee che possono essere sfruttate per dimostrare tale teorema:

1. Se lo schema di cifratura usasse un pad uniforme invece di G(k), lo schema sarebbe identico allo schema one-time pad ed A vincerebbe nell'esperimento con probabilità $\frac{1}{2}$ perché one-time pad è perfettamente segreto, dalla nozione di segretezza perfetta, per cui nell'esperimento di indistinguibilità è richiesto che A riesca a distinguere la cifratura tra i due messaggi con probabilità esattamente $\frac{1}{2}$.
2. Se l'avversario fosse in grado nel caso in cui lo schema utilizza uno generatore pseudocasuale, di vincere con probabilità significativamente maggiore di $\frac{1}{2}$, allora A potrebbe essere usato per distinguere G(k) da una stringa uniforme.

Dimostrazione: Dimostriamo che, per ogni Adv A PPT, esiste una funzione trascurabile negl(n) tale che:

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

La probabilità che l'avversario riesca a distinguere una cifratura di un m_0 da un m_1 è trascurabilmente migliore di un $\frac{1}{2}$. quindi l'avversario non sarà in grado di capire a quale dei due messaggi corrisponde la cifratura.
Si costruisce un **distinguisher D** che riceve in input una stringa $w \in \{0,1\}^{l(n)}$ ed opera nel modo seguente:

- Esegue A(1^n) per ottenere $m_0, m_1 \in \{0,1\}^{l(n)}$;
- sceglie uniformemente $b \in \{0,1\}$ e pone c:= $m_b \oplus w$;
- dà c ad A e ottiene da A il bit b';
- se b' = b, dà in output 1, altrimenti 0.

Quindi se A ha avuto successo, D capisce che la stringa w è stata costruita mediante un generatore pseudocasuale mentre se A fallisce, D capisce che la stringa w è stata selezionata in maniera uniformemente a caso. L'algoritmo D è PPT se A è PPT. Definiamo lo schema $\tilde{\Pi} = (\tilde{Gen}, \tilde{Enc}, \tilde{Dec})$ n cui l'algoritmo di generazione della chiave non genera il seme ma direttamente una stringa di l bit quindi corrisponde esattamente al one-time pad, dove $\tilde{Gen}(1^n)$ riceve in input il parametro di sicurezza n e dà in output una chiave uniforme $k1in\{0,1\}^{l(n)}$.
La segretezza 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 vista 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.

8) definizione CPA sicuro simmetrico
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($\cdot$)
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:

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

9) Funzioni Pseudocasuali: Si spieghi cosa sono informalmente e se ne fornisca una definizione formale.
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 di tempo polinomiale per calcolare F(k, x) dati k e x. Negli usi tipici, k viene scelto fissato. Per cui si ha:

$$F_k(x) = F(k, x) \text{ ovvero } F_k : \{0,1\}^* x \{0,1\}^* \rightarrow \{0,1\}^* \tag{12}$$

F è pseudocasuale se F_k, per k scelta uniformemente a caso è indistinguibile da una funzione scelta uniformemente a caso dall'insieme di tute le funzioni aventi lo stesso dominio e codominio. **Formalmente invece:**

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:

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

La differenza di probabilità è costituita dalla 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. Un algoritmo D che ha accesso ad un oracolo O(·) che implementa f uniforme o F_k, per k uniforme, può chiedere il valore della funzione su un numero polinomiale di input x. Inoltre, al termine deve cercare di capire se all'interno dell'oracolo è stato implementato f o F_k.
L'algoritmo D non riceve la chiave K, in quanto richiedendo all'oracolo O(·) 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.

9.1): Si consideri la funzione F(k,x) = $x^2 \oplus$ k, x e F(k,x) sono stringhe di n bit e $x^2 = x^2 \bmod 2^n$. Rappresenta F una funzione pseudocasuale?

F non è pseudocasuale poiché i suoi valori su ogni coppia di punti sono correlati. D chiede all'oracolo valutazioni su x_1^2 e x_2^2 ottenendo quindi $y_i = O(x_1^2)$ e $y_2 = O(x_2^2)$.
Se $y_1 \oplus y_2 = x_1^2 \oplus x_2^2$, stampa in output 1 altrimenti 0.

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

$$y_1 \oplus y_2 = (x_1^2 \oplus k) \oplus (x_2^2 \oplus k) = x_1^2 \oplus x_2^2 \tag{13}$$

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

$$Pr[y_1 \oplus y_2 = x_1^2 \oplus x_2^2] = Pr[y_2 = x_1^2 \oplus x_2^2 \oplus y_1] = \frac{1}{2^n} \tag{14}$$

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

9.2) Cifratura CPA con funzione pseudo casuale :
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:

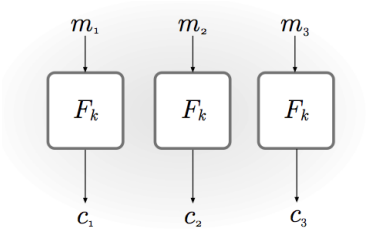
$$m := F_k(r) \oplus s$$

9.3): Se F è una funzione pseudocasuale, allora lo schema di cifratura che associa il cifrato c := < r, F_k(r) ⊕ m > al messaggio m, dove r e la chiave k sono scelti uniformemente a caso, è uno schema di cifratura CPA sicuro. (Vedere domanda 7).

10) Modalità operative: Si spieghi in modo chiaro e conciso quali sono le principali modalità operative utilizzate.

Le modalità operative sono utilizzate per la cifratura di messaggi molto lunghi.
Le modalità operative degli **stream cipher** avvengono in due modi, sincrono e asincrono. Gli stream cipher possono essere visti come PRG flessibili in quanto non hanno nessun fattore di espansione fisso e utilizzano una produzione dinamica della stringa come output, quindi un bit/byte alla volta. Immaginiamo che G sia il PRG. Il pad lo possiamo generare dando al generatore il seme e la lunghezza di esso sarà lunga quanto possibile in maniera da cifrare ogni parte del messaggio con una parte dello stesso pad, avendo quindi una forte sincronicità. Mentre nel contesto asincrono, ogni volta che bisogna cifrare un messaggio, utilizzo il generatore dando non solo la chiave ma anche il vettore di inizializzazione 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.

Per le **modalità operative dei cifrari a blocchi**, si supponga di avere una PRF F e una lunghezza del blocco n. Sia: m = $m_1 m_2 ... m_i \quad m_i \in \{0,1\}^n$ per ogni i = 1, ..., l.
Un messaggio è composto da l blocchi in cui ognuno ha una lunghezza di n bit. Quello che può accadere è che l'ultimo blocco non ha esattamente n bit e ciò viene colmato tramite una tecnica di padding: $|\ m_i| = n$.
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.



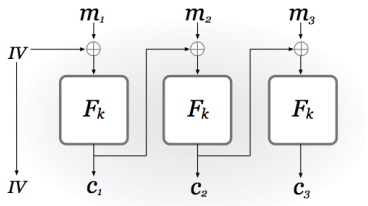
CBC (Cipher Block Chaining) Supponiamo che il messaggio sia costituita da un certo numero di blocchi. L'idea è scegliere un vettore di inizializzazione, uniformemente a caso, per poi effettuare l'operazione XOR di questo con il messaggio stesso, dove il risultato verrà usato come input dalla funzione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

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

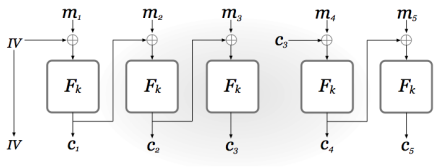
Questa modalità è probabilistica perché il vettore di inizializzazione viene scelto ogniqualvolta si cifra un blocco del messaggio, in maniera uniformemente a caso. Quindi con 10 blocchi uguali avrei 10 cifrature diverse. Unico problema è che la cifratura deve essere effettuata sequenzialmente, per calcolare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

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

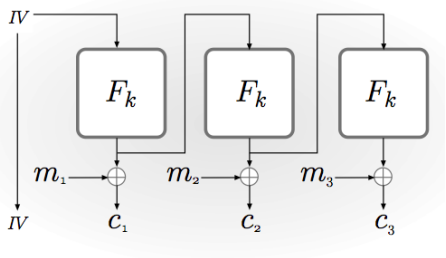
Nota: Se IV non è scelto uniformemente a caso la cifratura non è più sicura.



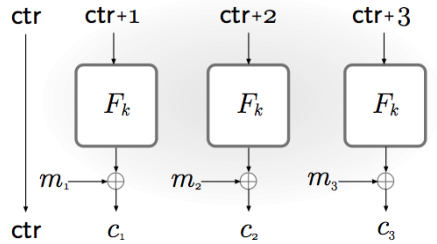
Chained CBC mode Una variante di CBC utilizzata è la modalità con concatenazione. Invece di usare un nuovo IV per ciascuna cifratura, l'ultimo blocco c_l del cifrato precedente viene utilizzato come IV per cifrare il prossimo blocco del cifrato. Questa modalità è efficiente perché nel momento in cui 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, l'attacco si basa sul fatto che il vettore IV è noto prima che la seconda cifratura abbia luogo.



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.



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 $i \in \{0,1\}^{n/4}$

11) definizione CCA sicuro simmetrico 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 $\text{Dec}_k(\cdot)$. Dà 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, dà in output $b' \in \{0,1\}$;
5. Se b = b', l'output dell'esperimento è 1 (A vince), altrimenti 0.

12) 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.

13) MAC cos'è:
Il MAC (message authentication code) è una primitiva crittografica che permette di identificare ed autenticare il mittente e controllare l'integrità del messaggio. Nel contesto della cifratura simmetrica, ogniqualvolta Alice manda un messaggio a Bob, attacca al messaggio una stringa chiamata Tag prodotta attraverso un algoritmo che utilizza la chiave segreta k: t $\leftarrow \text{Mac}_k$(m). Dall'altro lato, una volta che Bob ha ricevuto il messaggio e il tag associato, attraverso la funzione Vrfy(k,m,t) verifica che il tag t sia stato generato a partire dal messaggio m con la chiave k stampando come risultato "valido" o "non valido".

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;
- $t \leftarrow Mac_k(m)$ con 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.

13.1) MAC come si prova la sicurezza di un Message Authentication Code:
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. Si può definire il seguente esperimento "Mac-forge$_{A,\Pi}(n)$.
Sia $\Pi = (Gen, Mac, Vrfy)$ e A un Adv generico ed n il parametro di sicurezza:

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**.

13.2) MAC come si costruisce uno schema MAC sicuro: Se lo schema MAC è deterministico, quindi ad un messaggio m corrisponde un unico tag t, per verificare la correttezza si applica la verifica canonica, ovvero rigenerare il Mac per poi controllare l'uguaglianza con quello che ha inviato il messaggio allora lo schema è sicuro e anche fortemente sicuro.
Possiamo costruire MAC sicuri utilizzando le funzioni pseudocasuali, in quanto la probabilità di individuare il valore t=F$_k$(m) è molto prossimo a un valore uniformemente distribuito su stringe di n bit, cioè $\frac{1}{2^n}$
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)

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.

14) Cifratura autenticata : Si spieghi in modo chiaro e conciso cos'è e perché è utile:
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.

14.1): come si formalizza tale nozione (definizione CCA-sicuro simmetrico):
Definizione 4.17: Uno schema di cifratura a chiave privata Π é uno schema di cifratura autenticata se é CCA-sicuro e non falsificabile.
Uno schema di cifratura autenticata lo si realizza combinando schemi di cifratira e MAC sicuri, la combinazione più sicura è **Cifra e poi autentica**: Il mittente trasmette il cifrato <c, t> calcolato come:

$$c \leftarrow Enc_k(m) \text{ e } t = Mac_k(c) \tag{15}$$

Il ricevente verifica il tag t. Se risulta corretto, decifra c e dà in output m. Altrimenti restituisce $\perp$.

14.2) con quale approccio generico può essere ottenuta:
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 $\text{Priv}^{cpa}_{A,\Pi}$.

14.3): che relazione sussiste con la nozione di "schema di cifratura simmetrico CCA-sicuro"
Gli schemi di cifratura visti all'inizio non erano in grado di soddisfare la nozione di sicurezza CCA, ovvero attacchi di tipo chosen-chipertext. Con l'introduzione del concetto di cifratura autenticata si è in grado di produrre schemi che risultano CCA-sicuri e al contempo non falsificabili. Di conseguenza, uno schema di cifratura autenticata potrebbe essere più forte di uno schema CCA-sicuro, in quanto, uno schema CCA-sicuro non garantisce l'integrità dei dati.

15) Funzioni hash: Esperimento collisioni hash:
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 $H^s(x) \in \{0,1\}^{l(n)}$

L'obiettivo è far si che data una funzione hash, sia difficile trovare collisioni. Definiamo la sicurezza attraverso un esperimento. Siano Π = (Gen, H), Adv A e n parametro di sicurezza, definiamo l'esperimento trovare una collisione hash $\textbf{Hash-coll}_{A,\Pi}(n)$

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.

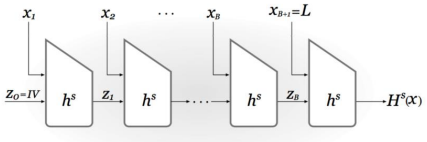
Definizione 5.2: Una funzione Π = (Gen, H) è resistente a collisioni se, per ogni avversario PPT A, esiste una funzione trascurabile negl tale che:

$$Pr[\text{Hash-coll}_{A,\Pi}(n) = 1] \leq negl(n) \tag{16}$$

15.1):Si descriva la trasformata di Merkle-Damgard per estendere il dominio di una funzione di compressione e si provi che trovare efficientemente collisioni per la funzione estesa implica trovare efficientemente collisioni per la funzione di compressione sottostante.
La costruzione delle funzioni hash 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.
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.



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':

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

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:

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

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.

16) HMAC Si spieghi in modo chiaro e conciso, cos'è e quale problema risolve:

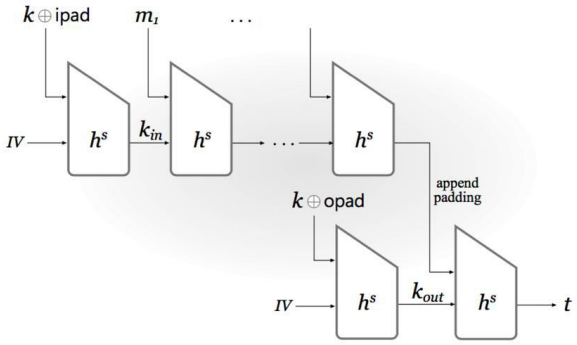
HMAC permette di autenticare messaggi di lunghezza arbitraria usando delle funzioni hash. L'idea è calcolare l'hash del messaggio per poi applicare il Mac sul hash calcolato. La funzione H^s è resistente a collisioni:

$$m \in \{0,1\}^*, y = H^s(m), Mac_k(y) \tag{17}$$

16.1) HMAC Come funziona:
È possibile costruire uno schema Mac sicuro per messaggi di lunghezza arbitraria, basandosi direttamente su una funzione hash, ma bisogna porre dell'attenzione alla costruzione perché considerare 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 $\text{Mac}_k(m) = H(k||m)$.
L'idea che invece funziona è quella di utilizzare due livelli di hash:

- Un primo livello per creare il **digest**;
- Un secondo livello per **autenticare**;

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.



16.2) HMAC perché si ritiene sicuro:
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.
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).
Nell'esperimento 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 $\rightarrow$ allora H è weakly collision-resistance.

17) Random Oracle Model
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. Il modello si basa sull'avere una funzione H totalmente casuale:

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

questa funzione è pubblica e può essere valutata soltanto attraverso delle query ad un oracolo.
Le parti oneste e l'avversario possono inviare query x all'oracolo ricevendo H(x) come se fosse una scatola nera dal punto di vista dell'avversario. Le query sono private quindi nessun altro conosce la query x fatta da una parte e nessun viene a conoscenza che una parte ha inviato una query. Inoltre, le query vengono risposte consistentemente, nel senso che con le stesse query si hanno le stesse risposte. Distinguere la funzione totalmente casuale e pseudocasuale veniva fatto mediante il paragone tra questi per definire le proprietà di un oggetto, ovvero un avversario nel capire quale delle due funzioni veniva utilizzato dall'oracolo per la cifratura dei messaggi si basava sullo studio delle query. Invece nel random oracle model, la funzione random H è parte della costruzione stessa.

18) Applicazioni di funzioni HASH: Autenticazione mediante 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.

18.1): Hash schema commitment
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.

19) Reti SPN e reti di Feistel: cosa si intende per confusione e diffusione, e come vengono e come vengono ottenute
In una permutazione cambiare un singolo bit nell'input significa ottenere un output quasi del tutto indipendente dall'output associato all'input precedente perché lo scopo della permutazione è dare dei valori diversi per ogni input diverso:

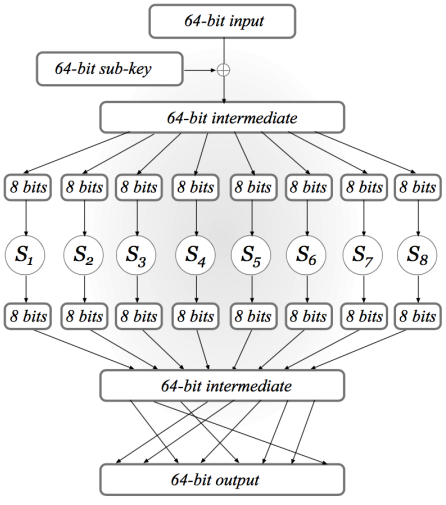
$$x \to y,\ x' \to z \neq y \tag{18}$$

Un cifrario a blocchi è uno strumento che serve per realizzare una permutazione pseudocasuale (PRP). 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 $\frac{1}{2}$. 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 casuiali*: $F_K(x) = f_1(x_1)||...||f_n(x_n)$.
Le funzioni f_i, 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 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.

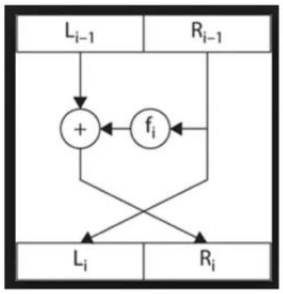
19.1) Reti SPN:
Una rete a sostituzione e permutazione (SPN) è un'implementazione diretta del paradigma della confusione e della diffusione. L'idea di fondo è che invece di usare una porzione della chiave k per scegliere una f_i fissiamo una funzione di sostituzione pubblica S. Quindi diremo che S è una S-box e useremo la chiave k o una porzione di essa per specificare la funzione f come:

$$f(x) = S(k \oplus x) \tag{19}$$

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 S-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.



19.2) Reti di Feistel:
La rete 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:

$$f_i : \{0,1\}^{l/2} \to \{0,1\}^{l/2} \text{ come } \quad f_i(R) \stackrel{\text{def}}{=} \hat{f}(K_i, R) \tag{20}$$

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:

$$L_i = R_{i-1} \text{ concatenato con } R_i = L_{i-1} \oplus f_i(R_{i-1}) \tag{21}$$

Le reti di Feistel sono invertibili.

20) Funzioni one-way: Cosa sono e come si definiscono
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

$$Pr[Invert_{A,f}(n) = 1] \leq negl(n) \tag{22}$$

20.1): Perchè sono sufficienti per realizzare tutta la crittografia simmetrica?
Si può dimostrare che le funzioni one-way permettono di realizzare tutte le astrazioni che sono necessarie per la crittografia a chiave privata. Infatti, se un PRG perdesse la proprietà di essere invertibile ne va a inficiare la proprietà di essere pseudocasuale in quanto l'avversario tramite l'inversione della funzione potrebbe calcolare il seme per sovvertire le proprietà che caratterizzano la sicurezza di uno schema.
Corollario 8.10. Assumendo l'esistenza di permutazioni one-way:

- esistono PRG con fattore di espansione polinomiale, PRF e PRP forti;
- esistono schemi di cifratura a chiave privata CCA-sicuri e schemi di autenticazioni di messaggi sicuri.

21) Primalità: Come possono essere generati numeri primi casuali di n bit?
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).

21.1): Cosa ci assicura che riusciamo a trovarne con alta probabilità con un numero di tentativi polinomiale in n?
Il teorema dei numeri primi ci dice che

$$\lim_{x \to \infty} \frac{\Pi(x)}{x/lnx} = 1 \tag{23}$$

quindi al crescere di x, x/ln x è una stringa ragionevole per $\Pi(x) \Rightarrow 1/\ln x$ stima della probabilità che x scelto a caso sia primo.

21.2) Come funziona il test di Miller e Rabin e quali risultati della teoria dei numeri utilizza.
Il test utilizza due algoritmi:

- 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.

La procedura di Miller e Rabin, Miller-Rabin(n, s), è quindi 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. Vengono utilizzate due proprietà della teoria dei numeri:

- **Teorema di Fermat**: Per qualsiasi primo p e per ogni a $\in Z_p^*$ risulta: $a^{p-1} \equiv 1 \bmod p$;
- **Radici quadrate dell'unità**: Se p è un primo dispari ed e ≥ 1 allora: $x^2 \equiv 1 \bmod p^e$ ha solo due soluzioni banali, x =1 e x = -1.

22) Gruppi ciclici e assunzioni crittografiche Cosa sono?
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.

22.1) Come sono definiti i problemi DL (logaritmo discreto) ,CDH DDH.
Il problema del logaritmo discreto è definto nel seguente modo:
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 $g^x = h$, dove g^x denota l'applicazione dell'operazione x volte. La x esponente di g prende il nome di **logaritmo discreto** in base g di h $\rightarrow log_g h$. Prendiamo in considerazione quindi l'esperimento
$\text{Dlog}_{A,GenG}$(n)

1. Il challenger 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.

Def: Relativamente a GenG(1^n), il problema del logaritmo discreto + difficile se per ogni algoritmo PPT A esiste una funzione trascurabile tale che:

$$Pr[\text{Dlog}_{A,GenG}(n) = 1] \leq negl(n) \tag{24}$$

Oltre al problema del logaritmo discreto, ci sono i problemi di Diffie-Hellman, una decisionale e una computazionale. Il setting è lo stesso del problema del logaritmo discreto quindi abbiamo un algoritmo di generazione di gruppi GenG() PPT che fornisce la rappresentazione del gruppo G, un generatore g e l'ordine q, quindi (G,g,q).
L'esperimento che consideriamo viene denotato:
$\text{CDH}_{A,GenG}$(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
$$h_1 = g^{x_1} \in G \text{ e } h_2 = g^{x_2} \in G \tag{25}$$
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:

$$Pr[\text{CDH}_{A,GenG}(n) = 1] \leq negl(n) \tag{26}$$

La variante decisionale del problema è chiamata Diffie-Hellman decisionale e consiste nel riuscire a distinguere data una tripla (h_1, h_2, h_3) se l'elemento h_3 è un elemento di Diffie-Hellman quindi g^{x_1,x_2} o un elemento scelto uniformemente a caso.

Consideriamo l'esperimento $\text{DDH}_{A,GenG}$(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

$$h_1 = g^{x_1} \in G \text{ e } h_2 = g^{x_2} \in G \tag{27}$$

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

$$Pr[\text{DDH}_{A,GenG}(n) = 1] \leq negl(n) \tag{28}$$

Le relazioni fra questi problemi sono le seguenti:

- 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).

22.2): Perché sono importanti i gruppi di ordine primo in crittografia?
I gruppi ciclici che vengono preferiti in crittografia sono quelli di ordine primo perché:

- Il problema DL è più difficile da risolvere dagli algoritmi noti per il calcolo del DL;
- Il problema DDH sembra essere difficile mentre è noto che risulta facile se l'ordine del gruppo q ha fattori primi piccoli;
- Trovare un generatore del gruppo è banale, essendo ogni elemento del gruppo eccetto l'unità un generatore;
- Rendono più facile le prove di sicurezza. Infatti, in alcune costruzioni crittografiche le prove di sicurezza richiedono il calcolo degli inversi moltiplicativi di certi esponenti. Nei gruppi di ordine primo q ogni esponente non nullo è invertibile.
- Quando si usa il problema DDH, se l'ordine del gruppo è primo, si può dimostrare che la distribuzione dei valori Diffie-Hellman $g^{x_1 x_2}$, dove x_1 e x_2 sono scelti a caso in Zq, è quasi uniforme.

23) Collision-resistance (DL difficile rispetto a Gen() => costruzione hash coll-res) La teoria dei numeri permette di realizzare funzioni hash collision-resistant. Si descriva la costruzione su gruppi ciclici presentata a lezione e si dimostri che, se il problema DL è difficile relativamente al generatore di gruppi prescelto, allora la costruzione risulta resistente a collisioni.

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:

$$H^s : \{0,1\}^{2(n-1)} \leftarrow \{0,1\}^l, \text{ con l} < 2\text{(n-1)} \tag{29}$$

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 $\epsilon(n)$:

$$Pr[\text{Hash-coll}_{A,\Pi}(n) = 1] = \epsilon(n) \tag{30}$$

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 $x'_2 - x_2 \neq 0 \bmod q$, 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 $x'_2 - x_2 \neq 0 \bmod q$. Poiché q è primo esiste l'inverso moltiplicativo di $x'_2 - x_2 \neq 0 \bmod q$:

$$\begin{aligned} (x'_2 - x_2)^{-1} \bmod q \leftarrow g^{(x_1 - x'_1)(x_2 - x'_2)^{-1}} &= h^{(x_1 - x'_1)(x_2 - x_2)^{-1}} = \\ &= h^1 = h \leftarrow log_2 h = (x_1 - x'_1)(x'_2 - x_2)^{-1} \end{aligned}$$

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.

24) Gestione chiavi simmetriche e pubbliche Si descriva l'esperimento dello 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:

$$Pr[KE^{eav}_{A,\Pi}(n) = 1] \leq \frac{1}{2} + negl(n) \tag{31}$$

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 $K_B := h^y_A$;
3. Alice riceve h_B e dà in output $K_A := h^x_B$.

facile vedere che il protocollo è corretto perché: $K_B = h^y_A = (g^x)^y) = g^{xy}$ e $K_A = h^x_B = (g^y)^x) = g^{yx}$. 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 $\rightarrow \overline{K} = H(K)$.

24.1): se DDH è difficile in G, allora lo scambio di chiavi Diffie-Hellman è EAV-sicuro
Teorema 10.3: Se il problema DDH è difficile relativamente a G, allora lo scambio di chiavi Diffie-Hellman Π è EAV-sicuro.
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:

$$|Pr[A(G, q, g, g^x, g^y, g^z) = 1] - Pr[A(G, q, g, g^x, g^y, g^{xy}) = 1]| \leq negl(n)$$

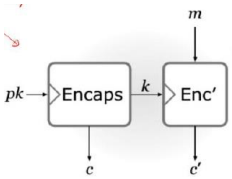
Discende quindi che:

$$Pr[\hat{KE}^{eav}_{A,\Pi} = 1] \leq \frac{1}{2} + \frac{1}{2} \cdot negl(n) \tag{32}$$

25) KEM: definizione KEM
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 := 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 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).



25.1): KEM CPA
KEM$^{cpa}_{A,\Pi}(n)$:

1. Gen(1^n) viene eseguito per produrre le chiavi (pk, sk). Poi 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.

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.

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:

$$Pr[KEM^{cpa}_{A,\Pi} = 1] \leq \frac{1}{2} + negl(n) \tag{33}$$

25.2) KEM CPA:
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: KEM$^{cca}_{A,\Pi}(n)$

1. Gen(1^n) viene eseguito per produrre le chiavi (pk, sk). Poi 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 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:

$$Pr[KEM^{cca}_{A,\Pi}(n) = 1] \leq \frac{1}{2} + negl(n) \tag{34}$$

26) Crittosistemi a chiave pubblica: Schema di cifratura asimmetrico
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.

26.1): Cifratura a chiave pubblica CCA-Sicura
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 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:

$$Pr[PubK^{cca}_{A,\Pi}(n) = 1] \leq \frac{1}{2} + negl(n) \tag{35}$$

26.2) Si spieghi in modo chiaro e coinciso che cosa si intende per crittosistema a chiave pubblica EAV-sicuro.

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. Il challenger sceglie uniformemente un bit e calcola la cifratura del messaggio selezionato $\leftarrow Enc_{pk}(m_b)$ e lo consegna ad A;
4. A sceglie un bit b $\in \{0,1\}$ scelto uniformemente;
5. 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:

$$Pr[PubK^{eav}_{A,\Pi}(n) = 1] \leq \frac{1}{2} + negl(n) \tag{36}$$

26.3): Si fornisca la motivazione dell'uguaglianza tra EAV e CPA sicurezza in crittografia asimmetrica.
Nel contesto pubblico, la definizione coincide con quella di sicurezza rispetto ad attacchi CPA perché nel contesto simmetrico l'esperimento CPA prevede che l'avversario potesse interrogare un oracolo, disponendo della chiave, il quale restituiva all'avversario cifratura di messaggi a scelta. Mentre in questo contesto l'avversario riceve la chiave pubblica, è come se l'oracolo fosse già incorporato nella chiave pubblica, quindi, può da solo produrre tutti i cifrati che vuole. Proposizione. Di conseguenza, possiamo dire che se uno schema di cifratura a chiave pubblica ha cifrature indistinguibili in presenza di un avversario che ascolta, allora è CPA-sicuro.

26.4) si fornisca un esempio di crittosistema che soddisfa tale definizione. In particolare, si descriva il funzionamento del crittosistema scelto e si fornisca uno sketch della prova di EAV-sicurezza. (EL-GAMAL)
Uno dei primi schemi di crittografia a chiave pubblica è lo schema di cifratura El Gamal, e funziona come segue:
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.
L'algoritmo di El-Gamal sfrutta le assunzioni di Diffie-Hellman, quindi, è un algoritmo che opera su un gruppo ciclico.

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 $y \in Z_q$:

$$c =< g^y, h^y \cdot m > \tag{37}$$

- Dec: prende in input la chiave privata sh=<G, q, g, x> e il cifrato $<c_1, c_2>$, manda in output il messaggio:

$$\hat{m} := \frac{c_2}{c_1^x} \tag{38}$$

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. (Prova nella domanda 26.5).

26.5): DDH difficile in G -> El Gamal CPA-sicuro: **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:

$$Pr[PubK^{eav}_{A,\Pi}(n) = 1] = \frac{1}{2} \tag{39}$$

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 $\text{PubK}^{eav}_{A,\Pi}(n)$ 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:

$$Pr[D(G, q, g, g^x, g^y, g^z) = 1] = Pr[PubK^{eav}_{A,\tilde{\Pi}}(n) = 1] = \frac{1}{2} \tag{40}$$

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:

$$Pr[D(G, q, g, g^x, g^y, g^{xy}) = 1] = Pr[PubK^{eav}_{A,\Pi}(n) = 1] \tag{41}$$

Se DDH è difficile, allora esiste negl(n):

$$|Pr[D(G, q, g, g^x, g^y, g^z) = 1] - Pr[D(G, q, g, g^x, g^y, g^{xy}) = 1]| \leq negl(n)$$
$$|Pr[PubK^{eav}_{A,\tilde{\Pi}}(n) = 1] - Pr[PubK^{eav}_{A,\Pi}(n) = 1]| \leq negl(n)$$
$$|\frac{1}{2} - Pr[PubK^{eav}_{A,\Pi}(n) = 1]| \leq negl(n)$$
$$Pr[PubK^{eav}_{A,\Pi}(n) = 1] \leq \frac{1}{2} negl(n)$$

26.6: si spieghi come costruire un KEM CCA-sicuro, nel random oracle model, usando le idee dello schema di El Gamal.
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$\rightarrow \{0,1\}^{2n}$. 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'$\leftarrow$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').

26.7): DHIES e ECIES

- 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.

Teorema: CDH difficile e H ROM => KEM è CPA-sicuro.

DIM: Sia A PPT. Vogliamo mostrare che $\exists$ negl(n):

$$Pr[KEM^{cpa}_{A,\Pi}(n) = 1] \neq \frac{1}{2} + negl(n) \tag{42}$$

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:

$$[noitemsep]Pr[KEM^{cpa}_{A,\Pi}(n) = 1] = Pr[KEM^{cpa}_{A,\Pi}(n) = 1 \wedge \overline{Query}] + Pr[KEM^{cpa}_{A,\Pi}(n) = 1 \wedge Query]$$
$$Pr[KEM^{cpa}_{A,\Pi}(n) = 1|\overline{Query}] + Pr[Query]$$

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*] $\leq negl(n)$. 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 $\in \{1, ..., t\}$ 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:

Vista di A come subroutine di A' ≡ 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:

$$h^y \in \{w_1, w_2, ..., w_t\} \tag{43}$$

Questo implica che A' dà in output il valore corretto con probabilità $\frac{1}{t}$. Pertanto:

$$Pr[A'(G, q, g, h, c) = h^y] \geq \frac{Pr[Query]}{t} \tag{44}$$

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)

27) RSA randomizzato: Costruzione 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:
Per utilizzare l'assunzione RSA bisogna passare dalla versione RSA-plain ad una versione di RSA randomizzata.

- 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:

$$c := [\hat{m}^e \; mod \; N] \tag{45}$$

- Dec: prende in input la chiave privata sk: = <N, d> e il cifrato c ∈ Z^*_N, calcola il messaggio:

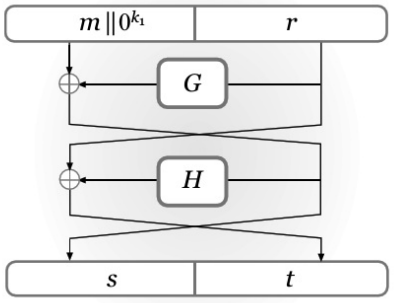
$$\hat{m} := [c^d \; mod \; N] \tag{46}$$

  E manda in output i |N|-l(n)-2 bits meno significativi di $\hat{m}$.

27.1): RSA con random pad
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.

27.2): RSA-OAEP
RSA-OAEP 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:

$$m \to \hat{m} \in Z_N^* \to c = [\hat{m}^e mod N] \tag{47}$$



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 || $0^{k'}$. 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.
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:

$$m' = m||0^{k_1}, s = m' \oplus G(r) et = r \oplus H(s), \text{ con r scelto in modo uniforme} \tag{48}$$

Il cifrato risultante è:

$$c = [(s||t)^e \text{ mod N}] \tag{49}$$

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:

$$s \text{ da } [(s||t)^e \text{ mod N}] \tag{50}$$

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.

27.2) (RSA difficile -> KEM con ROM e RSA CCA-sicuro) Si descriva il KEM che usa una funzione hash e la permutazione RSA. Inoltre, si provi che risulta CCA-sicuro nel random oracle model, assumendo che il problema RSA sia difficile.
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 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:

$$\leq Pr[Success|\overline{Query}] = \frac{1}{2} \tag{51}$$

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:

$$Pr[Query] \leq negl(n) \tag{52}$$

Pertanto RSA difficile => $Pr[Query] \leq negl(n)$ relativamente a GenRSA.

28) Schemi di firme digitali: Descrizione paradigma hash-and-sign
Gli schemi di firma digitale possono essere visti come l'analogo dei MAC nel contesto simmetrico. Quando Alice intende inviare uno un messaggio firmato a Bob calcola sul messaggio m un secondo valore σ tilizzando la chiave privata. La parte B, una volta ricevuta la coppia (m, σ), utilizza la chiave pubblica di A per verificare che l'elemento σ è la firma che è stata posta da A sul messaggio m. Le chiavi quindi sono utilizzate nel seguente modo:

- La chiave privata viene utilizzata per produrre la firma σ per poi inserirla sul messaggio m;
- La chiave pubblica viene utilizzata per verificare la validità della firma σ presente sul messaggio m.

La chiave privata viene indicata come chiave di firma e la chiave pubblica viene indicata come chiave di verifica.

Se si volessero firmare messaggi lunghi in maniera diretta con uno schema di firma il costo sarebbe veramente elevato. Quindi, come per gli schemi di cifratura a chiave pubblica l'approccio ibrido permette di guadagnare efficienza, così per gli schemi di firma un approccio ibrido permette di ottenere schemi più performanti. Nel caso delle firme, questo approccio è costituito dal paradigma hash-and-sign in cui procede sulla falsariga dell'approccio hash-and-mac:

$$m \in \{0,1\}^* , H(m) \in \{0,1\}^l , \ Sign_{sk}(H(m)) \tag{53}$$

In pratica, una volta considerato un messaggio di lunghezza generica, si applica una funzione hash per produrre una stringa di lunghezza fissata, per esempio di l bit, per poi firmare questo hash prodotto.

28.1) Si spieghi in modo chiaro e coinciso che cosa si intende per schema di firme digitali sicuro rispetto ad un adaptive chosen message attack.
Uno schema di firma è sicuro se nessuno è in grado di apporre una firma al posto di un altro. Per definire la sicurezza usiamo l'esperimento in cui l'avversario ha la possibilità di vedere delle firme su messaggi, tramite un oracolo di firma $\text{Sign}_{sk}(\cdot)$. La condizione di vittoria è data dalla produzione di una coppia nuova e contraffatta da parte dell'avversario, ovvero quando l'algoritmo $\text{Vrfy}_{pk}(\cdot)$ dà esito positivo e il messaggio non appartiene alle query precedenti effettuati dall'avversario all'oracolo.
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 $\notin Q$.

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):

$$Pr[\text{Sig-forge}_{A,\Pi}(n) = 1] \leq negl(n) \tag{54}$$

28.2): Contraffazione di firme con RSA
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:

$$\sigma := [m^d \bmod \text{N}] \tag{55}$$

- 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:

$$m := [\sigma^d \bmod \text{N}] \tag{56}$$

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 = [σ^emod 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 $\sigma = [\sigma_1 \sigma_2$ mod N] su m. Quindi la contraffazione sarà (m, σ). Risulta:

$$\sigma^e = (\sigma_1 \cdot \sigma_2)^e = (m_1^d \cdot m_2^d)^e = (m_1^{de} \cdot m_2^{de}) = m_1 \cdot m_2 = m \text{ mod N} \tag{57}$$

  In questo modo siamo riusciti a firmare un messaggio tramite l'utilizzo di due firme diversi.

28.3) Si descriva il funzionamento dello schema di firme RSA-FDH e si fornisca uno sketch della prova di sicurezza. In particolare, si spieghi come la produzione efficiente di contraffazioni, implichi l'inversione efficiente della permutazione RSA.

RSA-FDH è 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:

$$\sigma := [H(m)^d \text{ mod N}] \tag{58}$$

- Vrfy: prende in input pk, il messaggio m e una firma σ, manda in output 1 se e solo se:

$$\sigma^e := [H(m)^d \text{ mod N}] \tag{59}$$

Le proprietà che deve avere la funzione H è che deve essere difficile da invertire, non ammettere relazioni moltiplicative e deve essere 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:

$$\sigma^e = H(m) = H(m_i) = y \text{ mod N} \tag{60}$$

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:

$$\frac{1}{q} \cdot \text{ non-negl(n)} \tag{61}$$

Quindi con questa probabilità, l'avversario A' è in grado di invertire RSA perché il valore σ che 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:

$$\sigma = H(m_i)^d \text{ mod N} \tag{62}$$

Quindi per simulare l'oracolo di firma, l'avversario A', per m_j, sceglie un valore casuale σ_j e calcola H(m_j) = σ_j^e mod N dove e è l'esponente pubblico. Da notare che σ_j è uniforme quindi si ha che H(m_j) = σ_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.

28.4) RSA-FDH Randomizzato

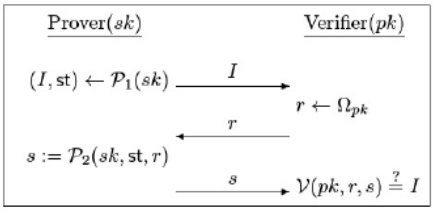
RSA-FDH randomizzato è un sistema crittografico a chiave pubblica che utilizza l'algoritmo RSA in combinazione con una funzione di hash a dominio completo (FDH) randomizzata.

- il processo di cifratura di RSA-FDH randomizzato prevede che il mittente scelga casualmente un numero, di lunghezza pari a quella della chiave RSA utilizzata, e che lo utilizzi come input per l'FDH. Il risultato dell'FDH viene quindi cifrato utilizzando la chiave pubblica RSA del destinatario, ottenendo così il messaggio cifrato.
- Il processo di decrittazione prevede invece che il destinatario decifri il messaggio cifrato utilizzando la sua chiave privata RSA e poi applichi l'FDH all'output decodificato. In questo modo si ottiene il messaggio originale.

Permette ad un messaggio di avere più firme.

29) Schemi di identificazione Che cosa si intende per schema di identificazione in un sistema interattivo, lo si spieghi con un esempio di schema a 3 round.

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.

29.1): Correttezza di uno schema di identificazione

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:

$$Pr[Ident_{A,\Pi}(n) = 1] \leq negl(n) \tag{63}$$

Viene considerato attacco passivo perchè nell'esperimento non viene eseguita nessuna cattiva azione che va a modificare le trascrizioni delle esecuzioni oneste tra P e V.

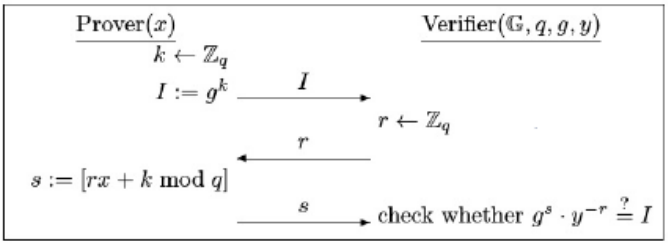
29.2): Trasformazione di Fiat Shamir
La trasformazione di Fiat e Shamir offre un metodo per convertire uno schema di identificazione interattivo in uno schema di firma non interattivo. L'idea è questa: il Prover esegue il protocollo di identificazione da solo, rimuovendo l'interazione usando una funzione 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 (Gen$_{id}$, P_1, P_2, V) uno schema di identificazione, uno schema di firma digitale è costruito come segue:

- Gen: prende in input 1^n, esegue 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.

29.3): Si descriva lo 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:

$$g^s \cdot y^{-r} = g^{rx+k} \cdot (g^x)^{-r} = g^{rx+k-rx} = g^k = I \tag{64}$$

29.4): Sicurezza dello schema d'identificazione di Schnorr
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:

$$g^{S_1} \cdot y^{-r_1} = I = g^{S_2} \cdot y^{-r_2} \rightarrow g^{S_1 - S_2} = y^{r_1 - r_2} \tag{65}$$

Da questo risultato si può ricavare che l'avversario può calcolare esplicitamente, moltiplicando:

$$g^{(S_1-S_2)(r_1-r_2)^{-1}} y^{(r_1-r_2)(r_1-r_2)^{-1}} \rightarrow y = g^{(S_1-S_2)(r_1-r_2)^{-1}}$$
$$\rightarrow \log_g y = [(S_1 - S_2)(r_1 - r_2)^{-1} \text{mod q}]$$

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.

29.5): Si spiehi come uno schema di identificazione di schnorr possa essere convertito in uno schema di firma digitale
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:$\{0,1\}^* \rightarrow Z_q$ 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:= $[rx + k \text{ mod q}]$. 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.

30) Sistema a conoscenza zero Cho cosa è?
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.
La proprietà **Zero Knowledge** (Conoscenza zero) può essere formalizzata richiedendo che per ogni verificatore V* esiste un simulatore S PPT, tale che, per ogni teorema vero X deve accadere che il transcript generato dalla coppia <P,V*> deve essere indistinguibile dal transcript che produce il simulatore S.

Il funzionamento dello schema è il seguente: 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 i 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:

$$P_n = r_1 \oplus ... \oplus r_{n-1} \oplus S \tag{66}$$

Per ottenere S basta che i partecipanti mettano insieme le share:

$$S = r_1 \oplus r_2 \oplus ... \oplus r_{n-1}(r_1 \oplus ... \oplus r_{n-1} \oplus S) \tag{67}$$

Se manca solo uno dei partecipanti la maschera non può essere rimossa.