

PER ALTRI APPUNTI CONSULTARE IL SITO:

https://luigi-v.github.io/Appunti_Universita/

Ripasso definizione schema di cifratura:

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

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

La condizione di **correttezza perfetta** che ci aspetta da uno schema di cifratura è:

Per ogni $k \in K$, per ogni $m \in M$ e qualsiasi $c \leftarrow \text{Enc}_k(m)$, risulta $\text{Dec}_k(c) = m$, con probabilità 1.

Definizione 2.3: uno schema di cifratura (Gen, Enc, Dec) con spazio dei messaggi M è **perfettamente segreto** se:

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

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

Ripasso esperimento cifratura simmetrica EAV (perfetta indistinguibilità):

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

$$\text{PrivK}_{A,\Pi}^{\text{eav}}$$

1. L'avversario A sceglie i due messaggi in chiaro, m e m' , mandandoli in output;
2. Il Challenger C genera k tramite $\text{Gen}()$ e *sceglie uniformemente a caso* il bit in $\{0,1\}$;
3. C calcola $c = \text{Enc}_k(m_b)$, detto **cifrato di sfida**, e lo passa ad A ;
4. A dà in output un bit b' ;
5. 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 $> 1/2$. La strategia migliore che un avversario può adottare è prevedere in maniera casuale il messaggio con probabilità di un $1/2$.

Definizione 2.5. Uno schema di cifratura (Gen, Enc, Dec) con spazio dei messaggi M è **perfettamente indistinguibile** se, per ogni A (non efficiente o efficiente), risulta:

$$\Pr[\text{PrivK}_{A,\Pi}^{\text{eav}} = 1] = 1/2$$

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. (*)

Ogni schema perfettamente segreto deve avere uno spazio delle chiavi almeno tanto grande quanto lo spazio dei messaggi.

Teorema 2.10. Se (Gen, Enc, Dec) è uno schema di cifratura perfettamente segreto con spazio dei messaggi M e spazio delle chiavi K , allora $|K| \geq |M|$

Dimostrazione.

Dimostriamo **per assurdo** che sia $|K| < |M|$. Sia M una distribuzione uniforme e sia $c \in C$ tale che $\Pr[C=c] > 0$. Definiamo:

$$M(c) \stackrel{\text{def}}{=} \{m \mid m = \text{Dec}_k(c), \text{ per qualche } k \in K\}.$$

Chiaramente $|M(c)| \leq |K|$. Se $|K| < |M|$, allora $\exists m' \in M$ tale che $m' \notin M(c)$. Ma allora risulta:

$$\Pr[M=m' \mid C=c] = 0 \neq \Pr[M=m']$$

Pertanto, lo schema non è perfettamente segreto in quanto, avendo la cardinalità K minore di M , esiste un messaggio che non può essere cifrato con un dato c . Quindi deve essere $|K| \geq |M|$.

Teorema 2.11 (Shannon bound). Sia (Gen, Enc, Dec) uno schema di cifratura con spazio dei messaggi M per cui $|M| = |K| = |C|$.

Lo schema è perfettamente segreto se e solo se:

- Gen sceglie ogni chiave $k \in K$ con probabilità uguale a $1/|K|$;
- Per ogni $m \in M$ ed ogni $c \in C$, esiste un'unica chiave $k \in K$ tale che $\text{Enc}_k(m) = c$

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

Segretezza Perfetta. (*) Inoltre, si descriva una generalizzazione del one-time pad che usi n cifre (con $n > 2$) e risulti perfettamente segreto.

Costruzione One-time pad:

Sia $l > 0$ un intero. Siano $M=K=C=\{0,1\}^l$. Quindi lo spazio dei messaggi, della chiave e dei cifrati coincidono:

- Esiste un generatore che sceglie uniformemente a caso la chiave $k \in \{0,1\}^l$;
- Un algoritmo di cifratura Enc che dati $k \in \{0,1\}^l$ e $m \in \{0,1\}^l$ manda in output il cifrato tramite l'operazione di XOR:
$$c := k \oplus m$$
- Un algoritmo di decifratura Dec che dati $k \in \{0,1\}^l$ e $c \in \{0,1\}^l$ manda in output il messaggio tramite l'operazione di XOR:
$$m := k \oplus c$$

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

Teorema 2.9. Lo schema di cifratura one-time pad è perfettamente segreto.

Segretezza Perfetta. (*) Inoltre, si spieghi perché il one-time pad risulta insicuro rispetto alla trasmissione di messaggi multipli, per qualsiasi nozione significativa di sicurezza rispetto a messaggi multipli.

Nello schema di cifratura one-time pad, la chiave è **tanto lunga quanto il messaggio** che si intende cifrare ed è **sicura soltanto per un determinato messaggio**. Quindi tale chiave non può essere utilizzato per cifrare un altro messaggio perché fare lo XOR con due messaggi diversi e chiave uguale permetterebbe all'avversario di capire esattamente in quale posizione i due messaggi risultano diversi. Dato che:

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

Quindi l'one-time pad non è perfettamente segreto per qualsiasi nozione di segretezza perfetta per messaggi multiplo.

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 $\Pi = (Gen, Enc, Dec)$ ha cifrature indistinguibili rispetto ad attacchi di tipo chosen-plaintext (**CPA-sicuro**) se, per ogni avversario A PPT, esiste una funzione trascurabile *negl* tale che:

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

Definizione 3.33. Uno schema di cifratura a chiave privata $\Pi = (Gen, Enc, Dec)$ ha cifrature indistinguibili rispetto ad attacchi di tipo chosen ciphertext (**CCA-sicuro**) se, per ogni avversario A PPT, esiste una funzione trascurabile *negl* tale che:

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

Ripasso esperimento cifratura simmetrica EAV:

Nel contesto computazionale abbiamo che l'avversario è PPT, può vincere con una probabilità trascurabile e l'esperimento dipende dal parametro di sicurezza n .

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

1. A ottiene 1^n e dà in output $m_0, m_1 \leftarrow A$ tali che $|m_0| = |m_1|$;
2. Il Challenger calcola $c \leftarrow Enc_k(m_b)$ dove $b \leftarrow \{0,1\}$ e $k \leftarrow Gen(1^n)$;
3. $A(1^n)$ riceve c e dà in output $b' \in \{0,1\}$;
4. Se $b=b'$ l'output dell'esperimento è 1 altrimenti 0.

Definizione 3.8. Uno schema di cifratura a chiave privata $\Pi = (Gen, Enc, Dec)$ ha cifrature indistinguibili in presenza di un avversario che ascolta (eavesdropper) o è EAV-sicuro se, per ogni Adv A PPT, esiste una funzione trascurabile *negl* tale che:

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

Generatori pseudocasuali. Si fornisca la definizione di generatore pseudocasuale. Inoltre, si consideri il seguente generatore

$$G: \{0,1\}^{nm} \rightarrow \{0,1\}^{n(m+1)}$$

Il generatore prende in input m stringhe $x_1 \dots x_m$ di n bit e dà in output

$$G(x_1 \dots x_m) = x_1 \dots x_m y, \text{ dove } y = \text{XOR}_i x_i$$

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

È G un generatore pseudocasuale? Si supporti la risposta con un argomento rigoroso.

Un **generatore pseudocasuale** G è un algoritmo deterministico efficiente per trasformare una stringa uniforme corta, chiamata **seme**, in una più lunga che sembra **uniforme**.

Sia un generatore $G: \{0,1\}^n \leftarrow \{0,1\}^l$ e sia $Dist$ la distribuzione sulle stringhe di l bit ottenuta scegliendo uniformemente a caso $s \in \{0,1\}^n$ e calcolando l'output che si ottiene applicando $G(s)$, G è un **PRG** se e solo se $Dist$ è pseudocasuale.

Risulta **pseudocasuale** se è impossibile per ogni algoritmo PPT, ovvero per ogni algoritmo che esegue in tempo al più polinomiale, dire con chance significativamente se la stringa derivi da $Dist$ o dalla distribuzione uniforme.

Definizione 3.14. Sia $l(n)$ un polinomio e G un algoritmo deterministico di tempo polinomiale, tale che per ogni n e $s \in \{0,1\}^n$, $G(s)$ è una stringa di $l(n)$ bit. Diremo che G è un PRG se valgono le seguenti condizioni:

1. **Espansione:** per ogni n risulta $l(n) > n$ (significa che il seme s di n bit viene esteso ad una qualche stringa che ha lunghezza maggiore di n : esempio $n \rightarrow p(n)$);
2. **Pseudocasualità:** per ogni algoritmo D PPT (per ogni test efficiente al quale si può pensare), esiste una funzione trascurabile $negl(n)$ tale che:

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

La differenza è data dalla probabilità che il test efficiente ricevendo in input una stringa che è l'output del generatore pseudocasuale sul seme s dà 1, e dalla probabilità che lo stesso test ricevendo in input una stringa che è scelta in accordo alla distribuzione uniforme, dà lo stesso valore uno. Dire che la differenza di probabilità tra questi due eventi è trascurabile risulta che l'avversario non è in grado di determinare quale dei due eventi è stato realizzato: se il generatore pseudocasuale o la selezione uniforme della stringa.

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 $Pr[D(r)=1]=1/2$, la probabilità che l'ultimo bit sia uguale allo xor dei precedenti è $1/2$ perché r viene scelta in modo uniforme in $\{0,1\}^{m+1}$

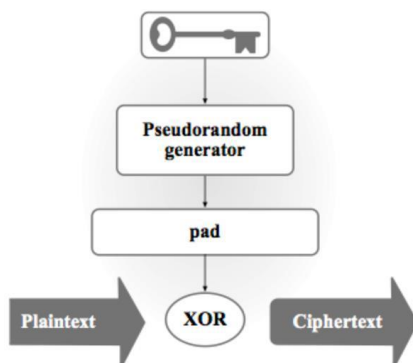
Poiché la differenza è

$$|Pr[D(G(s))=1] - Pr[D(r)=1]| = |1 - 1/2| = 1/2$$

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

Ripasso cifratura sicura con PRG:

L'idea alla base della costruzione dello schema di cifratura è ispirato al procedimento del one-time pad in cui cifrava i messaggi tramite $m \oplus r$, con r stringa scelta uniformemente a caso. In questo caso cifriamo tramite $m \oplus G(s)$ con $G(s)$ stringa pseudocasuale con le parti che devono condividere il seme s (chiave segreta).



Costruzione 3.17. Sia G un generatore pseudocasuale con fattore di espansione l . Definiamo uno schema di crittografia a chiave privata per i messaggi di lunghezza l come segue:

- **Gen:** prende in input 1^n , sceglie in modo uniforme $k \in \{0,1\}^n$ e la manda in output come chiave;
- **Enc:** prende in input la chiave $k \in \{0,1\}^n$ e il messaggio $m \in \{0,1\}^{l(n)}$, manda in output il cifrato:

$$c := G(k) \oplus m$$
- **Dec:** prende in input la chiave $k \in \{0,1\}^n$ e il cifrato $c \in \{0,1\}^{l(n)}$, e manda in output il messaggio:

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

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

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

Riduzioni: metodologia. Si descriva concisamente la **struttura generale** di una riduzione di sicurezza, evidenziando le **motivazioni** alla base dell'approccio e le **proprietà** che soddisfa. (*)

Nella crittografia moderna si ha bisogno di **definizioni** che stabiliscono in modo chiaro e non ambiguo lo schema di cifratura. Successivamente bisogna **dimostrare** che lo schema raggiunga effettivamente la definizione e creare una qualche **prova** che dimostra che la cifratura risulti essere sicura, dove nessun attaccante avrà successo. Queste prove vengono accompagnate da **assunzioni** che assumono alcuni concetti per rendere facile le dimostrazioni, in quanto risulta difficile dimostrare in maniera incondizionata che la costruzione risulta essere sicura.

La **struttura generale** di una riduzione è la seguente:

Assunzioni $x, y \dots$ valgono, allora la Costruzione Π soddisfa la definizione Z .

dove l'assunzione x e y possono essere rispettivamente l'assunzione della fattorizzazione e l'assunzione della funzione logaritmo discreto, che sono problemi difficili, ovvero non esistono algoritmi che risolvono questi problemi in tempo polinomiale. Mentre la definizione Z specifica il **security goal**, indipendentemente da qualsiasi informazioni l'attaccante possa già avere, un cifrato non dovrebbe rilasciare nessuna informazione aggiuntiva circa il sottostante messaggio in chiaro, e il **thread model**, in cui stabilisce la sicurezza della costruzione rispetto ad un avversario dal potere illimitato o limitato e che tipo di attacco.

(++)Riduzioni: metodologia. (*) Inoltre, come caso d'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. Se G è un PRG, la costruzione 3.17 realizza uno schema di cifratura a chiave privata per messaggi di lunghezza fissa che ha cifrati indistinguibili in presenza di un eavesdropper (EAV-sicuro).

Passi per la dimostrazione:

Sono due le idee che possiamo sfruttare 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 avversario A PPT, esiste una funzione trascurabile $negl(n)$ tale che:

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

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 $\tilde{\Pi} = (Gen, Enc, Dec)$ in 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 $Gen(1^n)$ riceve in input il parametro di sicurezza n e dà in output una chiave uniforme $k \in \{0,1\}^{l(n)}$.

Per la proprietà di segretezza perfetta (perfetta indistinguibilità) abbiamo che:

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

- Se w è una stringa uniforme in $\{0,1\}^{l(n)}$ allora la vista di A quando è eseguito come subroutine da D è uguale alla vista di A in $\text{PrivK}_{A,\tilde{\Pi}}^{eav}(n)$, poiché D dà in output 1 quando A ha successo, risulta

$$\Pr[D(w)=1] = \Pr[\text{PrivK}_{A,\tilde{\Pi}}^{eav}(n) = 1] = \frac{1}{2}.$$

- Se, invece, $w=G(k)$, dove k è una stringa uniforme in $\{0,1\}^n$ allora la vista di A quando è eseguito come subroutine da D è uguale alla vista di A in $\text{PrivK}_{A,\tilde{\Pi}}^{eav}(n)$, poiché D dà in output 1 quando A ha successo, risulta

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

G per ipotesi è un PRG e D è PPT. Pertanto, esiste $negl(n)$ tale che:

$$\begin{aligned} &= |\Pr[D(w)=1] - \Pr[D(G(k))=1]| \leq negl(n) \\ &= |\Pr[\text{PrivK}_{A,\tilde{\Pi}}^{eav}(n) = 1] - \Pr[\text{PrivK}_{A,\tilde{\Pi}}^{eav}(n) = 1]| \leq negl(n) \\ &= |\frac{1}{2} - \Pr[\text{PrivK}_{A,\tilde{\Pi}}^{eav}(n) = 1]| \leq negl(n) \\ &= \Pr[\text{PrivK}_{A,\tilde{\Pi}}^{eav}(n) = 1] \leq \frac{1}{2} + negl(n) \end{aligned}$$

Poiché A è PPT, possiamo concludere che Π ha cifrature indistinguibili in presenza di un EAV.

Ripasso esperimento cifratura simmetrica CPA:

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

Siano $\Pi=(Gen, Enc, Dec)$, un avversario A , 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:

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

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 tali che $|m_0|=|m_1|$;
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. Studiando 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 $\Pi=(Gen, Enc, Dec)$ ha cifrature indistinguibili rispetto ad attacchi di tipo chosen-plaintext (**CPA-sicuro**) se, per ogni avversario A PPT, esiste una funzione trascurabile *negl* tale che:

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

Funzioni Pseudocasuali. Si spieghi cosa sono informalmente e se ne fornisca una definizione formale. Inoltre, si consideri la funzione $F(k,x)=x^2 \oplus k$, dove k, x e $F(k,x)$ sono stringhe di n bit e $x^2 = x^2 \bmod 2^n$. Rappresenta F una funzione pseudocasuale?

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 e fissato. Per cui si ha:

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

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

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

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

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 $O(\cdot)$ 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 $O(\cdot)$ una valutazione su x e ricevendo $O(x)$ potrebbe calcolare $F_k(x)$ e controllare che $F_k(x)=O(x)$. Se l'uguaglianza sussiste, D con altissima probabilità sta interagendo con F_k .

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 $y_1=O(x_1^2)$ 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 $O \equiv F_k$, per ogni k , D dà in output 1 con probabilità 1 poiché:

$$y_1 \oplus y_2 = (x_1^2 \oplus k) \oplus (x_2^2 \oplus k) = x_1^2 \oplus x_2^2$$

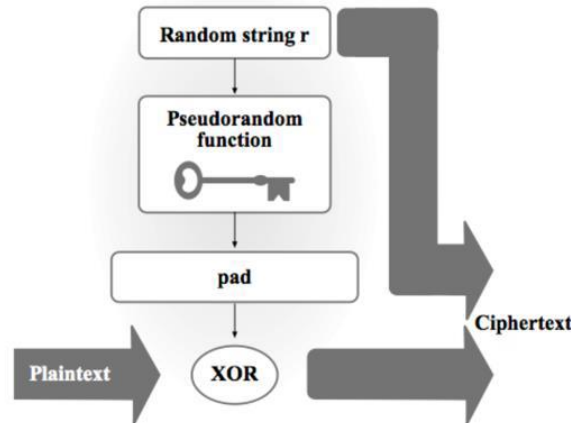
- Se $O \equiv f$, D dà in output 1 con probabilità:

$$\Pr[y_1 \oplus y_2 = x_1^2 \oplus x_2^2] = \Pr[y_2 = x_1^2 \oplus x_2^2 \oplus y_1] = 1/2^n$$

La differenza $|1 - 1/2^n|$ non è trascurabile, quindi F non è pseudocasuale.

Ripasso cifratura CPA con PRF:

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



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

- **Gen:** prende in input 1^n , sceglie in modo uniforme $k \in \{0,1\}^n$ e lo manda in output;
- **Enc:** prende in input la chiave $k \in \{0,1\}^n$ e il messaggio $m \in \{0,1\}^n$, sceglie uniforme $r \in \{0,1\}^n$ e da in output il cifrato:
$$c := \langle r, F_k(r) \oplus m \rangle$$
- **Dec:** prende in input la chiave $k \in \{0,1\}^n$ e il cifrato $c = \langle r, s \rangle$, e manda in output il messaggio:
$$m := F_k(r) \oplus s$$

(++) Riduzioni: metodologia. (*) Inoltre, come caso d'esempio, si dimostri che:

• se F è una funzione pseudocasuale, allora lo schema di cifratura che associa il cifrato $c := \langle r, f_k(r) \oplus m \rangle$ al messaggio m , dove r e la chiave k sono scelti **uniformemente** a caso, è uno schema di cifratura **CPA sicuro**.

Teorema 3.31. Se F è una funzione pseudocasuale, allora la costruzione 3.30 realizza uno schema di cifratura CPA-sicuro per messaggi di lunghezza n .

Dimostrazione:

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

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

Nel primo passo si costruisce uno schema di cifratura ipotetico in cui al posto della funzione pseudocasuale F_k che viene usata da Π , viene usata una funzione f , scelta uniformemente a caso.

Quindi sia $\tilde{\Pi} = (\tilde{Gen}, \tilde{Enc}, \tilde{Dec})$ costruito a partire da $\Pi = (Gen, Enc, Dec)$ tale che:

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

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

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

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

- D ha accesso all'oracolo $O(\cdot)$ e deve scoprire se la funzione è " F_k , per k uniforme in $\{0,1\}^n$ " o " $f \in Func_n$ uniforme";
- D emula l'esperimento $\text{PrivK}_{A,\tilde{\Pi}}^{cpa}(n)$ per A ed osserva se A ha successo, D deduce che A sta interagendo con un oracolo che memorizza una funzione pseudocasuale altrimenti una funzione scelta uniformemente a caso.

Codice:

Si costruisce un **Distinguisher D** che prende in input 1^n e ha accesso ad un oracolo $O: \{0,1\}^n \rightarrow \{0,1\}^n$:

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

Spiegazione del codice:

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

1. Il Distinguisher esegue l'algoritmo A come subroutine che gioca nell'esperimento $\text{PrivK}_{A,\Pi}^{cpa}$, quindi A può interrogare l'oracolo di cifratura su un messaggio. Quando A invia una query all'oracolo, verrà intercettato da D che risponde scegliendo uniformemente a caso una stringa r di n bit per poi mandare una query $O(r)$ all'oracolo ricevendo come risposta il messaggio cifrato y . Dopodiché ritorna ad A il cifrato contenente una coppia avente la stringa r e il messaggio cifrato tramite lo XOR tra y e m . Questo passo va avanti per tutte le volte in cui l'avversario chiede cifrature di messaggi;
2. Ad un certo punto l'avversario A darà in output i due messaggi, m_0 e m_1 , sui quali vuole essere sfidato. A questo punto D fa esattamente quello che avrebbe fatto il Challenger, ovvero sceglie un bit uniformemente a caso e al contempo sceglie una stringa r uniformemente a caso. Manda una query di questa stringa r all'oracolo ricevendo come risposta il cifrato y per poi ritornare all'avversario un cifrato contenente una coppia avente la stringa r e il messaggio cifrato tramite l' XOR del messaggio m_b e y . Successivamente, l'avversario potrebbe continuare a interagire col suo oracolo di cifrature e D continuerebbe a simulare le risposte, così come fatto al passo 1;
3. Quando A si sente pronto stampa in output b' . Quindi D verifica se l'avversario A ha indovinato andando a confrontare i due bit, b e b' . Se sono uguali, l'avversario ha vinto altrimenti no.

Analisi:

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

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

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

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

$$\Pr[D^{f(\cdot)}(1^n)=1] = \Pr[\text{PrivK}_{A,\Pi}^{cpa}(n)=1].$$

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

$$\begin{aligned} &= |\Pr[D^{F_k(\cdot)}(1^n)=1] - \Pr[D^{f(\cdot)}(1^n)=1]| \leq \text{negl}(n) \\ &= |\Pr[\text{PrivK}_{A,\Pi}^{cpa}(n)=1] - \Pr[\text{PrivK}_{A,\Pi}^{cpa}(n)=1]| \leq \text{negl}(n) \end{aligned}$$

Assicurato che possiamo analizzare lo schema ipotetico, il secondo passo ovvero cercare di capire qual è la probabilità che l'avversario A possa avere successo. Mostriamo che:

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

Nell'esperimento che stiamo considerando $\text{PrivK}_{A,\Pi}^{cpa}(n)$, viene scelta una stringa uniforme $r \in \{0,1\}^n$ e il cifrato si ottiene come coppia $\langle r, f(r) \oplus m \rangle$.

Ora supponiamo che r^* sia la stringa usata per produrre il cifrato di sfida, cioè $c^* := \langle r^*, f(r^*) \oplus m_b \rangle$.

Possiamo verificarsi due casi:

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

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

- 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 $c^* := \langle r^*, f(r^*) \oplus m_b \rangle$, 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 $c := \langle r^*, s \rangle = \langle r^*, f(r^*) \oplus m \rangle$, 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 che $\Pr[\text{PrivK}_{A,\Pi}^{cpa}(n)=1]$ è uguale a:

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

si ha:

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

Pertanto, Π è CPA-sicuro.

Modalità operative. Si spieghi in modo chiaro e conciso

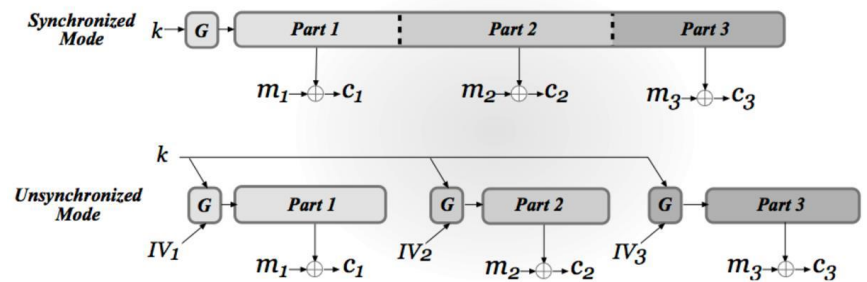
▪ Quali sono le principali modalità operative utilizzate (anche usando delle rappresentazioni)

Le modalità operative vengono impiegate per cifrare in maniera sicura ed efficiente 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**.

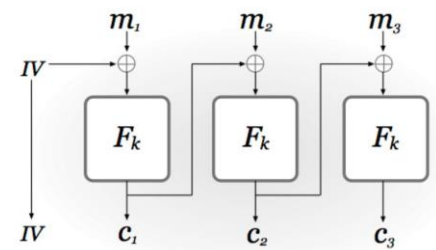


Per le **modalità operative dei cifrari a blocchi**, supponiamo di avere F un cifrario a blocchi con lunghezza di blocco n . Sia:

$$m = m_1 m_2 \dots m_l \quad m_i \in \{0,1\}^n \quad \text{per ogni } i=1, \dots, l$$

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: $|m_l|=n$.

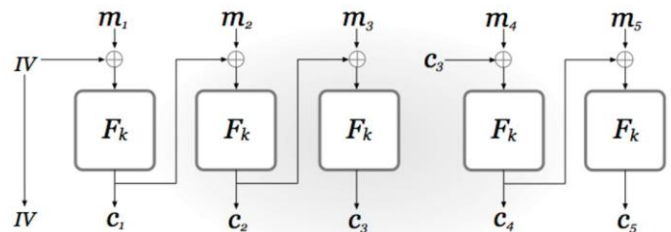
La prima modalità è **Cipher Block Chaining (CBC)**. Supponiamo che il messaggio sia costituita da un certo numero di blocchi. L'idea è scegliere un vettore di inizializzazione, uniformemente a caso, per poi effettuare l'operazione XOR di questo con il messaggio stesso, dove il risultato verrà usato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

Una variante di **CBC** utilizzata è la modalità con **concatenazione**.

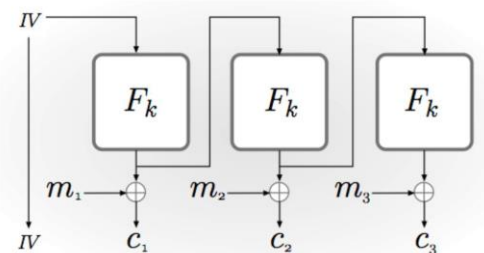
Invece di usare un nuovo IV per ciascuna cifratura, l'ultimo blocco c_i del cifrato precedente viene utilizzato come IV per cifrare il prossimo blocco del cifrato. Questa modalità è efficiente perché nel momento in cui Alice vuole mandare un secondo messaggio a Bob non c'è bisogno che venga inviato anche il nuovo vettore di inizializzazione perché c_1 sarà già a disposizione da parte di Bob. Quindi si tratta di una variante con stato.



Tuttavia, non è così sicura quanto CBC in quanto è vulnerabile ad attacchi di tipo chosen-plaintext in cui attacco si basa sul fatto che il vettore IV è noto prima che la seconda cifratura abbia luogo.

La modalità più importante è **Output Feedback Chaining, OFB**.

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

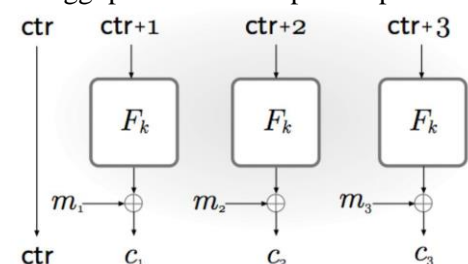


per generare un nuovo pad. Inoltre, non è necessario che il messaggio abbia una lunghezza multipla di n , in quanto la stringa pseudocasuale può essere troncata dove serve. Essendo i pad totalmente sganciati dai messaggi possono essere precomputati.

Un'altra modalità che è simile alla precedente è **Counter mode (CTR)**.

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_i . 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.



Ripasso esperimento cifratura simmetrica CCA:

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

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

1. Il Challenger sceglie $k \leftarrow \text{Gen}(1^n)$;
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 $c^* \leftarrow \text{Enc}_k(m_b)$;
4. A (1^n) riceve c^* , continua ad accedere agli oracoli $\text{Enc}_k(\cdot)$ e $\text{Dec}_k(\cdot)$. Non può chiedere la decifrazione 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.

Ripasso 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 necessità dell'oracolo di decifrazione $\text{Dec}_k(\cdot)$. 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. Nella fase di decifrazione 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.

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 $c = \langle c_0, c_1, c_2 \rangle = \langle IV, c_1, c_2 \rangle$. 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 decifrazione 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 decifrazione. Se fallisce, la decifrazione 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.

MAC. Si spieghi in modo chiaro e conciso

- cos'è
- come si prova la sicurezza di un Message Authentication Code;
- come si costruisce uno schema MAC sicuro.

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 $\text{Vrfy}(k, m, t)$ verifica che il tag t sia stato generato a partire dal messaggio m con la chiave k stampando come risultato "valido" o "non valido".

Definizione 4.1. Un MAC consiste di algoritmi PPT ($\text{Gen}, \text{Mac}, \text{Vrfy}$) dove:

- $k \leftarrow \text{Gen}(1^n)$, $|k| \geq n$: in cui genera le chiavi di autenticazioni di lunghezza n che rappresenta il parametro di sicurezza;
- $t \leftarrow \text{Mac}_k(m)$, $m \in \{0,1\}^*$: dove t è il tag associato a m ;
- $b := \text{Vrfy}_k(m, t)$, $b \in \{0,1\}$ in cui verifica la validità o meno del tag sul messaggio, tali che, per ogni n , per ogni k dato in output da $\text{Gen}(1^n)$ e per ogni $m \in \{0,1\}^*$ si ha $\text{Vrfy}_k(m, \text{Mac}_k(m)) = 1$.

Una classe importante è quella degli schemi deterministici in cui:

- $\text{Mac}_k(\cdot)$ è deterministica;
- La verifica si dice **canonica** in cui consiste nel ricalcolo del tag da parte di Bob e nella verifica che sia uguale a quello ricevuto, quindi *data la coppia (m, t) , ricalcola $\text{Mac}_k(m) = t'$ e verifica che $t' = t$* .

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 creare una coppia (m', t') valida, siccome non conosce k . Si può definire il seguente esperimento:

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

Sia $\Pi = (\text{Gen}, \text{Mac}, \text{Vrfy})$ uno schema MAC, un avversario A generico ed n il parametro di sicurezza:

1. Il Challenger genera $k \leftarrow \text{Gen}(1^n)$;
2. L'avversario $A^{\text{Mac}_k(\cdot)}$ viene mandato in esecuzione avendo l'accesso all'oracolo che il challenger ha settato con la chiave k , chiedendo l'autenticazione di messaggi di propria scelta ricevendo come risposta il tag t corrispondente. Quindi sia $Q = \{m_1, \dots, m_t\}$ l'insieme delle query richieste. Alla fine di questa interazione, l'avversario stamperà una coppia (m, t) ;
3. L'avversario ha successo se e solo se $\text{Vrfy}_k(m, t) = 1$, ovvero si tratta di un tag lecito, e $m \notin Q$, ovvero non è un messaggio che l'avversario ha precedentemente richiesto all'oracolo quindi il tag è stato prodotto manualmente dall'avversario. In questo caso, l'output dell'esperimento è 1, altrimenti è 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 $\text{Vrfy}_k(m, t) = 1$, 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**.

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 $1/2^n$.

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 .

(+) Cifratura autenticata. Si spieghi in modo chiaro e conciso

- cos'è e perché è utile
- come si formalizza tale nozione
- con quale approccio generico può essere ottenuta
- che relazione sussiste con la nozione di "schema di cifratura simmetrico CCA-sicuro"

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

Definizione 4.17. Uno schema di cifratura a chiave privata Π è uno schema di cifratura autentica se è CCA-sicuro e non falsificabile.

Uno schema di cifratura autenticata lo si realizza combinando schemi di cifratura e MAC sicuri, la combinazione più sicura è:

Cifra e poi autentica: Il mittente trasmette il cifrato $\langle c, t \rangle$ calcolato come

$$c \leftarrow \text{Enc}_k(m) \text{ e } t = \text{Mac}_k(c)$$

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

Costruzione 4.18. Sia $\Pi_E = (\text{Enc}, \text{Dec})$ uno schema di crittografia a chiave privata e sia $\Pi_M = (\text{Mac}, \text{Vrfy})$ 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 $(\text{Gen}', \text{Enc}', \text{Dec}')$ 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 $c \leftarrow \text{Enc}_{k_E}(m)$ e $t \leftarrow \text{Mac}_{k_M}(c)$. Manda in output $\langle c, t \rangle$.
- *Dec'*: prende in input la chiave (k_E, k_M) e il cifrato $\langle c, t \rangle$, prima controlla se $\text{Vrfy}_{k_M}(c, t) = 1$, allora manda in output $\text{Dec}_{k_E}(c)$, 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 $\text{PrivK}_{A, \Pi}^{\text{cpa}}$.

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.

Ripasso 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 stampa in output s ;
- *H* prende in input s ed una stringa $x \in \{0,1\}^*$ e stampa in output $H^s(x) \in \{0,1\}^{l(n)}$.

L'obiettivo è far sì che data una funzione hash, sia difficile trovare le collisioni. Definiamo la sicurezza attraverso un esperimento. Siano $\Pi = (\text{Gen}, H)$, l'avversario A e n parametro di sicurezza, denotato con il seguente termine:

$$\text{Hash-coll}_{A, \Pi}(n)$$

1. Il Challenger esegue la funzione $\text{Gen}(1^n)$ per individuare una specifica funzione della famiglia;
2. L'avversario A riceve il parametro s e stampa in output due valori del dominio, x e x' .
3. Il Challenger controlla che $x \neq x'$ e $H^s(x) = H^s(x')$. In caso affermativo stampa 1 che varrebbe a dire che l'avversario A ha trovato una collisione, altrimenti 0.

Definizione 5.2. Una funzione $\Pi = (\text{Gen}, H)$ è resistente a collisioni se, per ogni avversario PPT A , esiste una funzione trascurabile *negl* tale che:

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

(++) Funzioni hash. 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.

Funzioni HASH. Si spieghi in modo chiaro e conciso:

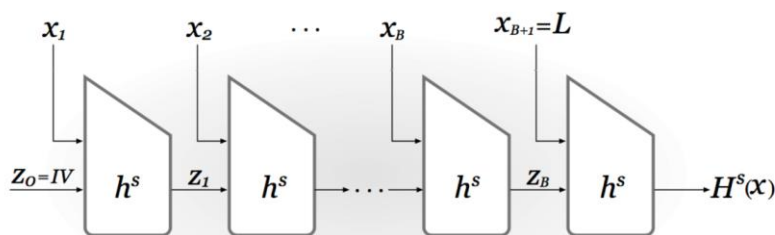
- L'estensione di dominio di Merkle-Damgard
- Si dimostri la sicurezza dello schema.

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 Merkle-Damgard**.

Il suo funzionamento è che: 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_i prodotto dalla funzione di compressione precedente, ad eccezione del primo blocco che prende come secondo input un IV. Alla fine della trasformata, avremo come risultato il valore hash per la stringa x .



Teorema 5.4. Se (Gen, h) è resistente rispetto a collisioni, allora anche (Gen, H) lo è.

Dimostrazione:

Dimosteremo 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'

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

tali che $H^s(x) = H^s(x')$.

Ci sono 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:

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

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:

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

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 uguale per entrambi $x_{B+1} = x'_{B+1}$.

Per dimostrare tale caso, introduciamo delle notazioni con lo scopo di renderlo facile. Siano:

z_1, \dots, z_{B+1} i valori prodotti dal calcolo di $H^s(x)$.

I_1, \dots, I_{B+1} gli input per h^s , cioè $I_i = z_{i-1} \| x_i$, per $i = 1, \dots, B+1$

z'_1, \dots, z'_{B+1} i valori prodotti dal calcolo di $H^s(x')$.

I'_1, \dots, I'_{B+1} gli input per h^s , cioè $I'_i = z'_{i-1} \| x'_i$, per $i = 1, \dots, B+1$

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 $x_i \neq x'_i$ quindi, come conseguenza a ciò, esiste N .

D'altra parte, dato che

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

deve essere $N \leq B+1$ dato che c'è l'uguaglianza per I_{B+2} e I'_{B+2} .

Per definizione, N è l'indice più grande per cui $I_N \neq I'_N$. 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 .

HMAC. Si spieghi in modo chiaro e conciso

- cos'è e quale problema risolve
- come funziona (anche un diagramma commentato va bene)
- perché è stato progettato in quel modo (paradigma di riferimento)
- perché si ritiene sicuro

HMAC. Si descriva in modo chiaro e conciso lo schema di autenticazione HMAC e se ne discuta la sicurezza.

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.

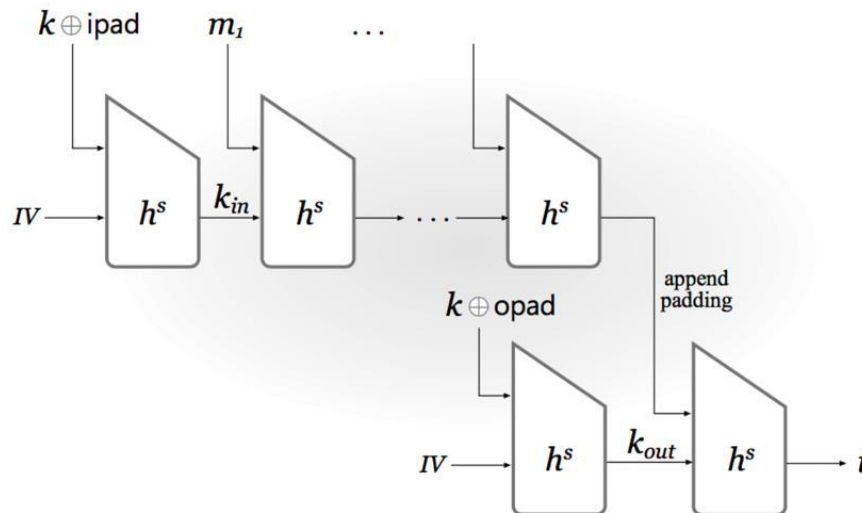
$$m \in \{0,1\}^*, y = H^s(m), \text{Mac}_k(y)$$

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 consiste nel prendere semplicemente 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:

1. Un primo livello per creare il *digest*;
2. 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-Damgård; 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 sottoposta 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.



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'istanziatura 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 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 deboli).

Nell'esperimento $\text{Hash-coll}_{A,\Pi}$ l'avversario una volta che ha ricevuto s , che specifica la funzione di compressione, si calcola da solo tutti gli hash che vuole e provare 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 oracolare.

Questo è importante perché se H è collision-resistance \rightarrow allora H è weakly collision-resistance.

Ripasso Random Oracle Model (ROM):

Per diverse costruzioni che utilizzano funzioni hash non si è grado di fornire una prova o una riduzione di sicurezza. La soluzione è utilizzare un modello idealizzato, ovvero il ROM. In questo modello si assume la presenza di una **funzione totalmente casuale pubblica** in cui associa una stringa arbitraria ad una stringa di l bit:

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

che può essere valutata soltanto attraverso 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**.

Applicazioni di funzioni HASH. Si spieghi in modo chiaro e conciso il funzionamento di:

- L'autenticazione mediante Merkle-Tree
- Schema commitment

Consideriamo lo scenario in cui un utente non ha dello spazio sufficiente sulla macchina locale per la memorizzazione di un file grande per cui carica questo 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 effettuata la funzione hash H su questo per poi andare a verificare l'uguaglianza tra il valore appena calcolato con quello presente già sul computer. In questo modo, viene verificata l'integrità del file scaricato.

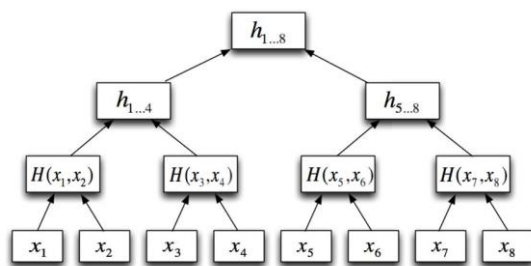
Questo non è un metodo efficiente, in quanto, se consideriamo di memorizzare una sequenza di file x_1, \dots, x_t , dovremo memorizzare per ognuno di essi un hash $H(x_1), \dots, H(x_t)$. Quindi lo svantaggio è dovuto alla memorizzazione dei digest che cresce in maniera proporzionale al numero di file che vengono caricati sul cloud.

Esiste una tecnica efficiente che permette di definire una memorizzazione

“autenticata” ed “efficiente” dell'informazione, ovvero il **Merkle-Tree**.

L'idea è quella di memorizzare i digest ispirandosi alla struttura dell'albero, avremo come n foglie quanti sono i file che vengono caricati sul cloud per poi costruire i nodi mediante la concatenazione a coppia di questi digest. Avremo come risultato finale un valore hash situato sulla **radice dell'albero** calcolato concatenando l'hash dei due sottoalberi.

In questa maniera basterebbe soltanto che l'utente memorizzi il valore hash presente all'interno del nodo radice dell'albero $h_{1\dots 8}$ per poi così uploadare i file x_1, \dots, x_8 nel cloud.



Successivamente, quando l'utente vuole verificare l'integrità del file x_i riceve dal cloud il file x_i e tutti i valori hash associati ai nodi adiacenti alla path da x_i alla radice. Per esempio, quando l'utente richiede il file x_3 , riceverà dal cloud, oltre il file x_3 , il file x_4 , $h_{1,2}$ e $h_{5\dots 8}$. Dopodiché questo effettua i dovuti controlli per assicurare l'integrità del file scaricato. Questo perché le informazioni aggiuntive che vengono fornite dalle path adiacenti sono una sorta di prova che sottintendono un buon mantenimento dell'informazione dando la possibilità all'utente di ricalcolare il valore radice.

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 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.

Formalmente, uno **schema di commitment** è una tripla di algoritmi $\Pi = (Gen, Com, Decom)$ tale che:

- Gen , probabilistico che genera i *params* pubblici;
- $Com(params, r, m) \rightarrow com$ che genera il commitment value *com*, su input *params*, una stringa random r ed il messaggio m . Crea la busta per poi inserire il messaggio;
- $Decom(com) \rightarrow (r, m)$ stampa in output la stringa casuale r ed il messaggio m usati per calcolare *com*. Apre la busta per leggere ciò che c'è dentro.

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

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

Reti SPN e reti di Feistel. Si spieghi in modo chiaro e conciso:

- cosa si intende per **confusione** e **diffusione**, e come vengono ottenute
- cosa sono e perché sono importanti nella progettazione di cifrari simmetrici

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:

$$x \rightarrow y, x' \rightarrow z \neq y$$

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 $1/2$. 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 casuali.

$$F_K(x) = f_1(x_1) \parallel \dots \parallel f_n(x_n)$$

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 resto dei byte uguali avremo che i due output $F(x)$ e $F(x')$ saranno differenti soltanto nel primo 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**.

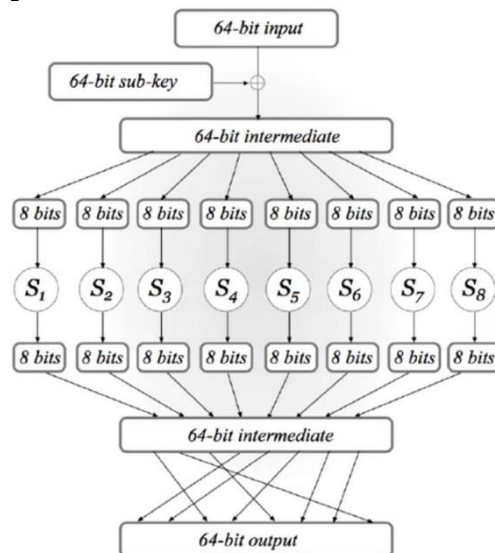
Questo metodo viene chiamato **paradigma della confusione e diffusione** potrebbe essere un modo attraverso il quale si cerca di riprodurre l'astrazione della pseudocasualità dell'output. Quindi, attraverso passi che realizzano confusione e diffusione globale, si potrebbe, con un certo numero di round, 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**.

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:

$$f(x) = S(k \oplus x)$$

Consideriamo una rete a sostituzione e permutazione con un blocco di 64 bit, basata su una collezione di S -box S_1, \dots, 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 divisa 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.



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

Nelle **reti di Feistel bilanciate**, la i -esima funzione di round \hat{f}_i .

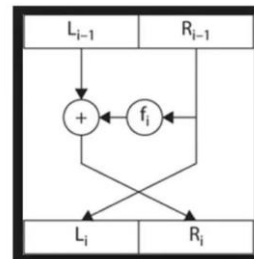
- Prende in input una sottochiave K_i ed una stringa R di $l/2$ bit dove l è la lunghezza della stringa x ;
- Stampa in output una stringa di $l/2$ bit.

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

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

Le \hat{f}_i 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} dove l'output viene calcolato in: $L_i = R_{i-1}$ concatenato con $R_i = L_{i-1} \oplus f_i(R_{i-1})$.



(+) **Funzioni one-way.** Si spieghi in modo chiaro e conciso

- cosa sono e come si definiscono
- perché sono sufficienti per realizzare tutta la “crittografia simmetrica”

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**. Consideriamo $f: \{0,1\}^* \rightarrow \{0,1\}^*$ una funzione. Sia l'avversario A un algoritmo efficiente ed indichiamo con n il parametro di sicurezza. Si può definire l'esperimento

$$\text{Invert}_{A,f}(n)$$

- Il challenger sceglie uniformemente $x \in \{0,1\}^n$ e calcola $y=f(x)$;
- A riceve in input 1^n e y , e dà in output x' ;
- L'output dell'esperimento è 1 se $f(x')=y$, altrimenti 0.

L'avversario vince l'esperimento se è stato in grado di trovare una pre-immagine x' tale che $f(x')=y=f(x)$.

Definizione 7.1. Una funzione $f: \{0,1\}^* \rightarrow \{0,1\}^*$ è **one way** se:

- Facile da calcolare:** esiste un algoritmo di tempo polinomiale M_f per calcolare f . Per ogni x risulta $M_f(x)=f(x)$.
- Difficile da invertire:** per ogni avversario A PPT, esiste una funzione trascurabile $\text{negl}(n)$ tale che la probabilità che l'avversario riesca ad invertire è trascurabile:

$$\Pr[\text{Invert}_{A,f}(n)=1] \leq \text{negl}(n)$$

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 7.10. Assumendo l'esistenza di **permutazioni one-way**, esistono PRG con fattore di espansione polinomiale, PRF e PRP forti.

Corollario 7.11. Assumendo l'esistenza di **permutazioni one-way**, esistono schemi di cifratura a chiave privata CCA-sicuri e schemi di autenticazioni di messaggi sicuri.

(+++++) **Primalità.** Si spieghi in modo chiaro e conciso

- come possono essere generati numeri primi casuali di n bit
- cosa ci assicura che riusciamo a trovarne con alta probabilità con un numero di tentativi polinomiale in n
- come funziona il test di Miller e Rabin e quali risultati della teoria dei numeri utilizza

Il **test di Miller e Rabin** utilizza due algoritmi:

- Sia $\text{Random}(1, n-1)$ un algoritmo che restituisce un numero a compreso tra 0 ed $n-1$, scelto in modo casuale;
- Sia $\text{Witness}(a, n)$ un algoritmo che restituisce true se e solo se il valore a è un testimone della compostezza di n .

La procedura di Miller e Rabin, **Miller-Rabin**(n, s), è una ricerca probabilistica parametrizzata di una prova che n è composto. Sceglie s valori casuali e, se una di queste scelte risulta essere un testimone che n è composto, allora la procedura restituisce composto. D'altra parte, se nessun testimone viene trovato negli s tentativi, assume che ciò accada perché non vi sono testimoni e, di conseguenza, restituisce primo.

Vengono utilizzate due proprietà degli elementi della teoria dei numeri:

- Teorema 24 (Teorema di Fermat).** Per qualsiasi primo p e per ogni $a \in \mathbb{Z}_p^*$, risulta
$$a^{p-1} \equiv 1 \pmod{p}.$$

- Teorema 27 (Radici quadrate dell'unità).** Se p è un primo dispari ed $e \geq 1$, allora
$$x^2 \equiv 1 \pmod{p^e}$$

ha solo due soluzioni banali, $x = 1$ e $x = -1$.

(++) **Gruppi ciclici.** Si spieghi in modo chiaro e conciso

- Cosa sono;
- Come sono definiti i problemi DL e DH (Computazionale e decisionale)
- Perché sono importanti i gruppi di ordine primo in crittografia.

Gruppi ciclici e assunzioni crittografiche. Si spieghi concisamente cosa si intende per gruppo ciclico e si descrivano le assunzioni DL e DDH definite su tali gruppi, evidenziando le relazioni note tra di esse.

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.

Il **problema del logaritmo discreto** può essere definito nel modo seguente:

Supponiamo che $\text{GenG}()$ sia un algoritmo PPT per la generazione di gruppi ciclici. L'output di questo è (G, g, q) dove G è un insieme di elementi, g è un generatore e q denota l'ordine del gruppo (che sarebbe $p-1$).

Supponiamo che l'operazione definita su questo gruppo \oplus è efficientemente calcolabile e che anche la verifica dell'appartenenza di un elemento al gruppo sia efficientemente calcolabile.

Se G è un gruppo ciclico, sappiamo che tutti gli elementi di G possono essere scritti $G = \{g^0, \dots, g^{q-1}\}$. Allora per ogni elemento del gruppo, ovvero $\forall h \in G$ esiste un unico esponente $x \in \mathbb{Z}_q$ tale che $g^x = h$, dove g^x denota l'applicazione dell'operazione x volte. La x che è esponente di g prende il nome di **logaritmo discreto** in base g di $h \rightarrow \log_g h$.

Consideriamo, quindi, l'esperimento:

$$DLog_{A,GenG}(n)$$

1. Il challenger C esegue $GenG(1^n)$ in modo da ottenere (G, g, q) ;
2. C sceglie in modo uniforme $h \in G$;
3. L'avversario A riceve (G, g, q) e h e stampa in output $x \in \mathbb{Z}_q$;
4. Il challenger controlla se $g^x = h$. In caso affermativo C stampa in output 1, altrimenti 0.

Definizione. Relativamente a $GenG()$, il problema DL è difficile se, per ogni avversario A PPT, esiste una funzione trascurabile tale che:

$$\Pr[DLog_{A,GenG}(n)=1] \leq \text{negl}(n)$$

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:

$$CDH_{A,GenG}(n)$$

1. Il challenger C esegue $GenG(1^n)$ per ottenere (G, g, q) ;
2. C sceglie in modo uniforme $x_1 \in \mathbb{Z}_q, x_2 \in \mathbb{Z}_q$ e calcola $h_1 = g^{x_1} \in G, h_2 = g^{x_2} \in G$;
3. L'avversario riceve $(G, g, q), h_1, h_2$ e stampa in output $h_3 \in G$;
4. Se $h_3 = g^{x_1 x_2}$ allora il challenger stampa in output 1 altrimenti 0.

Il problema **Diffie-Hellman computazionale** consiste nel calcolare $h_3 = g^{x_1 x_2}$ per elementi scelti a caso h_1, h_2 .

Definizione. Relativamente a $GenG(1^n)$, il problema Diffie-Hellman computazionale è difficile se, per ogni avversario A PPT, esiste una funzione trascurabile $\text{negl}(n)$ tale che:

$$\Pr[CDH_{A,GenG}(n)=1] \leq \text{negl}(n)$$

La variante decisionale del problema, che si chiama **Diffie-Hellman decisionale** 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.

Pertanto, possiamo considerare l'esperimento in cui non cambia niente rispetto al problema computazionale di Diffie-Hellman circa il setting. L'esperimento viene denotato con il seguente termine:

$$DDH_{A,GenG}(n)$$

1. Il challenger C esegue $GenG(1^n)$ per ottenere (G, g, q) ;
2. C sceglie in modo uniforme $x_1 \in \mathbb{Z}_q, x_2 \in \mathbb{Z}_q$ e calcola $h_1 = g^{x_1} \in G, h_2 = g^{x_2} \in G$;
3. Il challenger sceglie un bit b . Se $b=1$, calcola $h_3 = g^{x_1 x_2}$. Altrimenti, sceglie x_3 uniformemente a caso e calcola $h_3 = g^{x_3}$;
4. L'avversario A riceve (G, g, q) e (h_1, h_2, h_3) e stampa in output al challenger C, b' ;
5. Il challenger controlla se $b'=b$. In caso affermativo, stampa 1 altrimenti 0.

Definizione. Relativamente a $GenG(1^n)$, il problema Diffie-Hellman decisionale è difficile, se per ogni avversario A PPT, esiste una funzione trascurabile $\text{negl}(n)$ tale che:

$$\Pr[DDH_{A,GenG}(n)=1] \leq \frac{1}{2} + \text{negl}(n)$$

dove $\frac{1}{2}$ rappresenta la scelta casuale da parte dell'avversario.

Per quanto riguarda le **relazioni tra questi problemi**, è facile vedere che:

- Se DL fosse facile $\rightarrow CDH$ facile \rightarrow dati h_1 e h_2 , si calcola $x_1 = \log_g h_1$ e poi $h_2^{x_1} = h_3$ perché $(h_2)^{x_1} = (g^{x_2})^{x_1} = h_3$.

Purtroppo, non sappiamo se vale l'inverso, ovvero se il problema DL è difficile, non è detto che CDH lo sia.

D'altra parte, è facile vedere che:

- Se CDH fosse facile $\rightarrow DDH$ è facile \rightarrow data la tripla (h_1, h_2, h_3) , si calcola h_3' da h_1 e h_2 per poi controllare se $h_3' = h_3$, allora la tripla sarebbe di tipo Diffie-Hellman, altrimenti casuale.

In questo caso, l'inverso non sembra essere vero, ovvero CDH difficile $\rightarrow DDH$ difficile.

Ci sono dei gruppi in cui DL e CDH , allo stato attuale delle nostre conoscenze, sembrano essere difficili, mentre sappiamo che DDH è facile. Quindi abbiamo degli algoritmi efficienti per distinguere triple di Diffie-Hellman da triple casuali in \mathbb{Z}_p^* ma non abbiamo algoritmi efficienti per farlo nel sottogruppo di ordine q di \mathbb{Z}_p^* , posto che p sia scelto in modo opportuno.

Ci sono varie classi di gruppi ciclici in cui i problemi DL e Diffie-Hellman sono ritenuti difficili.

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 \mathbb{Z}_q , è *quasi uniforme*.

(+) Collision-resistance. 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, g, q) . 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 $(x_1, x_2) \in \mathbb{Z}_q \times \mathbb{Z}_q$ restituisce l'hash della stringa $x = x_1 || x_2 \rightarrow H^s(x_1, x_2) = g^{x_1} h^{x_2} \in G$.

Note: 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 H^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:

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

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 $Hash-call_{A, \Pi}(n)$. 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 $\epsilon(n)$:

$$\Pr[Hash-call_{A, \Pi}(n)=1] = \epsilon(n)$$

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 specifica opera come segue:

1. A' riceve in input la descrizione del gruppo ciclico, data dalla terna (G, g, q) e h , elemento di cui calcolare il DL;
2. Utilizza l'input per definire una chiave $s = \langle G, g, q, h \rangle$ 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:

a. x come (x_1, x_2) con $x_1, x_2 \in \mathbb{Z}_q$

b. x' come (x'_1, x'_2) con $x'_1, x'_2 \in \mathbb{Z}_q$

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 $Hash-call_{A, \Pi}$ 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 $Hash-call_{A, \Pi}$ 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:

$$\begin{aligned} H^s(x_1, x_2) = H^s(x'_1, x'_2) &\leftrightarrow g^{x_1} h^{x_2} = g^{x'_1} h^{x'_2} \leftrightarrow (g^{x_1} h^{x_2}) g^{-x'_1} h^{-x'_2} = (g^{x'_1} h^{x'_2}) g^{-x'_1} h^{-x'_2} \\ &\leftrightarrow g^{x_1 - x'_1} = h^{x'_2 - x_2} \end{aligned}$$

Si nota che $x_2' - x_2 \neq 0 \bmod q$, altrimenti sarebbe che $g^{x_1 - x'_1} = h^0 = 1 = g^0 \rightarrow x_1 = x'_1$ 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 $x_2' - x_2 \neq 0 \bmod q$:

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

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.

Ripasso esperimento scambio chiavi KE:

Un protocollo di sicurezza di scambio chiavi è sicuro se la chiave che Alice e Bob danno in output è totalmente imprevedibile 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 formalizzato richiedendo che l'avversario, dopo aver ascoltato una esecuzione, non è in 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 **imprevedibilità** 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:

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

1. Le due parti eseguono il protocollo Π su input 1^n . Sia $trans$ il transcript della comunicazione e sia K che danno in output;
2. Viene scelto $b \leftarrow \{0,1\}$. Se $b=0$, si pone $\hat{K} := K$ (valore che hanno realmente calcolato), altrimenti $\hat{K} \leftarrow \{0,1\}^n$ uniforme;
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:

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

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 $x \leftarrow \mathbb{Z}_q$ 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 $y \leftarrow \mathbb{Z}_q$ e $h_B := g^y$. Invia h_B ad Alice e stampa in output $K_B := h_A^y$;
3. Alice riceve h_B e dà in output $K_A := h_B^x$.

È facile vedere che il protocollo è corretto perché: $K_B = h_A^y = (g^x)^y = g^{xy}$ e $K_A = h_B^x = (g^y)^x = g^{yx}$.

Il protocollo permette ad Alice e Bob di calcolare un elemento comune del gruppo. Viene applicata una trasformazione all'elemento in comune del gruppo $K = g^{xy}$ usando un'appropriata funzione di derivazione $H \rightarrow \bar{K} = H(K)$.

(+) Riduzioni: metodologia. (*) Inoltre, come caso d'esempio, si dimostri che:

- se DDH è difficile nel gruppo 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:

Nota: Sia $\widehat{KE}_{A,\Pi}^{eav}$ l'esperimento $KE_{A,\Pi}^{eav}(n)$ in cui A distingue tra K e un elemento di G scelto uniformemente a caso.

Sia A un avversario PPT.

Poiché $\Pr[b=0] = \Pr[b=1] = \frac{1}{2}$, risulta:

$$\Pr[\widehat{KE}_{A,\Pi}^{eav}(n)=1] = \frac{1}{2} \cdot \Pr[\widehat{KE}_{A,\Pi}^{eav}(n)=1 \mid b=0] + \frac{1}{2} \cdot \Pr[\widehat{KE}_{A,\Pi}^{eav}(n)=1 \mid b=1]$$

Nell'esperimento $\widehat{KE}_{A,\Pi}^{eav}$, 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 .

Nota che distinguere tra i due casi (h_A, h_B, K) e (h_A, h_B, u) è equivalente a risolvere DDH.

Pertanto, la $\Pr[\widehat{KE}_{A,\Pi}^{eav}(n)=1]$ risulta uguale a:

$$\begin{aligned} & \frac{1}{2} \cdot \Pr[\widehat{KE}_{A,\Pi}^{eav}(n)=1 \mid b=0] + \frac{1}{2} \cdot \Pr[\widehat{KE}_{A,\Pi}^{eav}(n)=1 \mid b=1] \\ &= \frac{1}{2} \cdot \Pr[A(G, q, g, g^x, g^y, g^{xy})=0] + \frac{1}{2} \cdot \Pr[A(G, q, g, g^x, g^y, g^z)=1] \\ &= \frac{1}{2} \cdot (1 - \Pr[A(G, q, g, g^x, g^y, g^{xy})=1]) + \frac{1}{2} \cdot \Pr[A(G, q, g, g^x, g^y, g^z)=1] \\ &= \frac{1}{2} + \frac{1}{2} \cdot (\Pr[A(G, q, g, g^x, g^y, g^z)=1] - \Pr[A(G, q, g, g^x, g^y, g^{xy})=1]) \\ &\leq \frac{1}{2} + \frac{1}{2} \cdot |\Pr[A(G, q, g, g^x, g^y, g^z)=1] - \Pr[A(G, q, g, g^x, g^y, g^{xy})=1]| \end{aligned}$$

Se DDH è difficile relativamente a f , allora \exists $negl$ 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:

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

Ripasso definizione 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 generazione 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 $m := Dec_{sk}(c)$.

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 .

Ripasso esperimento cifratura asimmetrica CCA:

$$PubK_{A,\Pi}^{cca}$$

1. Il challenger genera tramite $Gen(1^n)$ una coppia di chiavi (pk, sk) ;
2. L'avversario A riceve come input pk e l'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). Restituisce una coppia di messaggi, m_0 e m_1 , della stessa lunghezza;
3. Il challenger sceglie uniformemente un bit $b \in \{0,1\}$ per poi applicare l'algoritmo di cifratura al messaggio selezionato $c \leftarrow Enc_{pk}(m_b)$. Restituisce tale risultato all'avversario 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 selezionato dal challenger, quindi restituisce b' ;
5. Il challenger controlla se $b' = b$. In caso affermativo, stampa 1, altrimenti 0.

Definizione 11.8. Uno schema di cifratura a chiave pubblica $\Pi = (Gen, Enc, Dec)$ 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:

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

Ripasso definizione KEM:

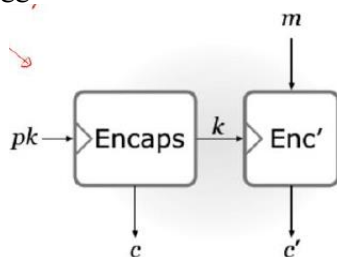
Vengono combinati schemi di cifratura a chiave pubblica e schemi di cifratura a chiave privata, in modo da realizzare i **cifrari ibridi**. È possibile farlo introducendo il concetto di **KEM**, ovvero meccanismo di incapsulamento della chiave.

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 di 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 produrre l'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).



Ripasso esperimento KEM CPA:

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

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 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:

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

Ripasso esperimento KEM CCA:

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.

$$KEM_{A,\Pi}^{cca}(n)$$

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}) ed ha accesso ad un oracolo $Decaps_{sk}(\cdot)$, ma non chiede mai la decapsulazione di c ;
4. A dà in output b' . L'output dell'esperimento è 1 se $b' = b$, altrimenti 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:

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

Crittosistemi a chiave pubblica. Si spieghi in modo chiaro e conciso che cosa si intende per crittosistema a chiave pubblica EAV-sicuro. Si fornisca la motivazione dell'uguaglianza tra EAV e CPA sicurezza in crittografia asimmetrica. Inoltre, 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.

(++) Crittosistemi a chiave pubblica. Si spieghi in modo chiaro e conciso che cosa si intende per crittosistema a chiave pubblica CPA-sicuro. Inoltre, 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 CPA-sicurezza.

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

1. L'algoritmo di generazione $Gen(1^n)$ produce una coppia di chiave pubblica pk e chiave privata sk ;
2. L'avversario A riceve la chiave pubblica pk , e restituisce una coppia di messaggi, m_0 e m_1 della stessa lunghezza;
3. Il challenger sceglie uniformemente un bit e calcola la cifratura del messaggio selezionato $c \leftarrow Enc_{pk}(m_b)$ per poi consegnarlo all'avversario A . Chiameremo c il cifrato di sfida;
4. L'avversario osserva il cifrato, effettua delle computazioni per poi dare alla fine quella che è la sua scommessa b' ;
5. Il challenger verifica che $b'=b$. In caso affermativo, stampa 1 (l'avversario ha vinto), altrimenti 0.

Utilizzando questo esperimento possiamo creare la seguente definizione:

Definizione 11.2. Uno schema di cifratura a chiave pubblica $\Pi=(Gen,Enc,Dec)$ ha una cifratura indistinguibile in presenza di un avversario che ascolta se per ogni avversario A PPT esiste una funzione trascurabile tale che:

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

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.

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 $c=m*k$, un'operazione che maschera il messaggio con un elemento k casuale. Bob recupera il messaggio $m=c/k$.

Se K è indistinguibile da una chiave casuale, ed è quello che ci assicura lo scambio di chiavi Diffie-Hellman, ovvero c è indistinguibile da un valore casuale. Quindi c non dà 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) . Dopo sceglie in modo uniforme $x \in \mathbb{Z}_q$ e computa $h:=g^x$. La chiave pubblica è $\langle G,q,g,h \rangle$ e la chiave privata è $\langle G,q,g,x \rangle$. Il messaggio è preso dallo spazio G ;
- **Enc:** prende in input la chiave pubblica $pk=\langle G,q,g,h \rangle$ e il messaggio $m \in G$, sceglie in modo uniforme $y \in \mathbb{Z}_q$ e manda in output il cifrato:

$$c=\langle g^y, h^y \cdot m \rangle$$

- **Dec:** prende in input la chiave privata $sk=\langle G,q,g,x \rangle$ e il cifrato $\langle c_1, c_2 \rangle$, manda in output il messaggio:

$$\hat{m}:=c_2/c_1^x$$

La correttezza di questo algoritmo è dovuta dal seguente motivo:

$$\text{Dati } \langle c_1, c_2 \rangle = \langle g^y, h^y \cdot m \rangle \text{ con } h=g^x, \text{ risulta } \hat{m} = c_2/c_1^x = h^y \cdot m / (g^y)^x = (g^y)^x \cdot m / (g^y)^x = m$$

La prima componente della coppia $\langle g^y, h^y \cdot m \rangle$ 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.

-Sketch della prova di CPA-sicurezza di El-Gamal nella domanda successiva...

Crittosistemi a chiave pubblica. Si dimostri che se il problema **DDH è difficile** nel gruppo G , allora lo schema di cifratura di El Gamal è CPA-sicuro. Inoltre, si spieghi come costruire un KEM CCA-sicuro, nel random oracle model, usando le idee dello schema di El Gamal.

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 $KEM_{A,\Pi}^{cca}(n)$ 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 $c=\langle g^y, g^z * m \rangle$. Questo algoritmo non permette a Bob di decifrare perché non ha modo di rimuovere la maschera $g^z * m$, 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 $k*m$ si ottiene un valore la cui distribuzione è la stessa che si ottiene se si sceglie k uniformemente a caso, lo schema $\tilde{\Pi}$ produce delle cifrature che sono distribuite uniformemente e indipendenti dal messaggio m , per cui

$$\Pr[PubK_{A,\tilde{\Pi}}^{eav}(n)=1] = \frac{1}{2}$$

Costruiamo un **distinguisher** D per il problema decisionale di Diffie-Hellman che cerca di distinguere tra una tripla di Diffie-Hellman da una tripla casuale, servendosi dell'avversario A che invece supponiamo sia in grado di vincere nell'esperimento CPA. D opera nel modo seguente:

1. In input riceve quella che è un'istanza del suo problema quindi (G, q, g, h_1, h_2, h_3) ;
2. Costruisce una chiave pubblica $pk = (G, q, g, h_1)$ ed esegue $A(pk)$. L'avversario A pensa di star eseguendo l'esperimento $PubK_{A, \Pi}^{eav}(n)$ per cui ad un certo punto sfida il challenger restituendo due messaggi $m_0, m_1 \in G$;
3. Il distinguisher che sta simulando il Challenger sceglie uniformemente a caso un bit $b \in \{0, 1\}$. Dopodiché costruisce le cifrature di sfida: $c_1 = h_2, c_2 = h_3 * m_b$;
4. Restituisce $\langle c_1, c_2 \rangle$ all'avversario A e cerca di capire se corrisponde a una cifratura di m_0 o m_1 restituendo 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. **• caso)** Il primo caso è quello in cui il distinguisher D riceve in input una tripla *casuale* quindi: $h_1 = g^x, h_2 = g^y, h_3 = g^z$ con $x, y, z \in \mathbb{Z}_q$. Il cifrato che viene prodotto nel secondo passo $c = \langle g^y, g^z * m \rangle$ è esattamente il cifrato che viene prodotto dal cifrato Π , per cui:

$$\Pr[D(G, q, g, g^x, g^y, g^z) = 1] = \Pr[PubK_{A, \Pi}^{eav}(n) = 1] = 1/2$$

2. **• caso)** Supponiamo che in questo caso il distinguisher D riceve in input una *tripla di Diffie-Hellman*: $h_1 = g^x, h_2 = g^y, h_3 = g^{xy}$ con $x, y \in \mathbb{Z}_q$. Il cifrato che viene prodotto nel secondo passo $c = \langle g^y, g^{xy} * m \rangle$ è una cifratura in accordo alla realizzazione di una cifratura tramite lo schema $PubK_{A, \Pi}^{eav}$ per cui:

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

Se DDH è difficile, allora $\exists \text{negl}(n)$:

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

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

$$|1/2 - \Pr[PubK_{A, \Pi}^{eav}(n) = 1]| \leq \text{negl}(n)$$

$$\Pr[PubK_{A, \Pi}^{eav}(n) = 1] \leq 1/2 + \text{negl}(n)$$

Possiamo ottenere 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 $\rightarrow \langle g^y, Enc'_{KE}(m), Mac_{KM}(c') \rangle$ dove la prima componente è prodotta dal KEM e le successive chiavi che vengono utilizzate per cifrare il messaggio m e autenticare il cifrato c' , prodotta dall'operazione di cifratura per il messaggio m , vengono in qualche modo derivate dal KEM.

Costruzione 11.23. Sia $\Pi_E = (Enc', Dec')$ uno schema di crittografia a chiave privata e sia $\Pi_E = (Mac, Vrfy)$ un MAC. Si definisce uno schema di cifratura a chiave pubblica come segue:

- **Gen:** prende in input 1^n e restituisce (G, q, g) , sceglie in modo uniforme $x \in \mathbb{Z}_q$, imposta $h := g^x$, e specifica la funzione $H: G \rightarrow \{0, 1\}^{2n}$. La chiave pubblica è $\langle G, q, g, h, H \rangle$ e la chiave privata è $\langle G, q, g, x, H \rangle$;
- **Enc:** prende in input la chiave pubblica $pk = \langle G, q, g, h, H \rangle$, sceglie in modo uniforme $y \in \mathbb{Z}_q$ e imposta $k_E || k_M := H(h^y)$. Computa $c' \leftarrow Enc'_{k_E}(m)$, e manda in output il cifrato $\langle g^y, c', Mac_{k_M}(c') \rangle$;
- **Dec:** prende in input la chiave privata $sk = \langle G, q, g, x, H \rangle$ e il cifrato $\langle c, c', t \rangle$, 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 $Dec'_{k_E}(c)$.

Ripasso RSA-randomizzato:

Per utilizzare l'assunzione RSA bisogna passare dalla versione RSA-plain ad una versione di **RSA randomizzata**.

Costruzione 11.30. Sia l una funzione con $l(n) \leq 2n-4$ per tutti gli n . Si definisce uno schema di crittografia a chiave pubblica:

- **Gen:** prende in input 1^n , esegue $GenRSA(1^n)$ ed ottiene (N, e, d) . Manda in output la chiave pubblica $pk = \langle N, e \rangle$ e la chiave privata $sk = \langle N, d \rangle$;
- **Enc:** prende in input la chiave pubblica $pk = \langle N, e \rangle$ e un messaggio $m \in \{0, 1\}^{\lceil N \rceil - l(n) - 2}$, sceglie in modo uniforme $r \in \{0, 1\}^{l(n)}$ e interpreta $\hat{m} := r || m$ come un elemento di \mathbb{Z}_N^* . Manda in output il cifrato:

$$c := [\hat{m}^e \bmod N]$$

- **Dec:** prende in input la chiave privata $sk = \langle N, d \rangle$ e un cifrato $c \in \mathbb{Z}_N^*$, computa:

$$\hat{m} := [c^d \bmod N]$$

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

Crittosistemi a chiave pubblica. Si *discuta* la sicurezza delle versioni randomizzate del crittosistema RSA (con random pad) e si *presenti* lo standard RSA-OAEP, *discutendone* la sicurezza.

L'algoritmo di RSA permette di vedere i messaggi come stringhe binarie di lunghezza al più $\lceil N \rceil - l(n) - 2$. 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 .

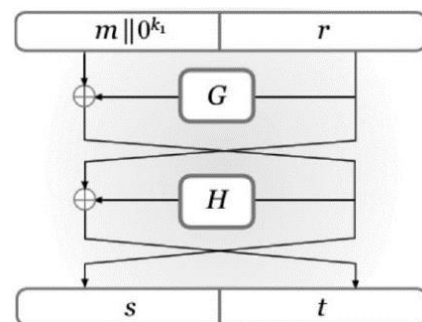
D'altra parte, se il PAD è il più grande possibile, per esempio con m che è un singolo bit, allora Padded RSA è CPA-sicuro. Tuttavia, per i casi intermedi, casi in cui $\hat{m} = r || m$ con $|r| = 1/2$ e $|m| = 1/2$ la situazione non è chiara perché non ci sono prove che attacchi PPT non possono esistere se vale l'assunzione RSA ma al momento non se ne conoscono.

Lo **standard RSA-OAEP** (Optimal asymmetric encryption padding - schema di padding ottimale per la cifratura simmetrica) 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:

$$m \rightarrow \hat{m} \in Z_N^* \rightarrow c = [\hat{m}^e \bmod N]$$

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 $m' = m || 0^{k_1}$. 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 \rightarrow RSA-OAEP CCA sicura.

L'intuizione del risultato è la seguente, durante la cifratura il mittente calcola:

$$m' = m || 0^{k_1}, s = m' \oplus G(r) \text{ e } t = r \oplus H(s), \text{ con } r \text{ scelto in modo uniforme.}$$

Il cifrato risultante è:

$$c = [(s || t)^e \bmod N]$$

Se l'avversario non chiede r all'oracolo G , $G(r)$ è uniforme per l'avversario e, quindi, m' è 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 è 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 $G(r)$ è 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 . Di conseguenza l'avversario non ha speranze nel recuperarlo finché non invia la query s all'oracolo H . A questo punto, l'alternativa sarebbe provare a indovinare scegliendo a caso, ma se la lunghezza di r è sufficientemente grande, la probabilità che l'avversario indovini è trascurabile. Pertanto, l'unica cosa che l'avversario può fare è calcolare:

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

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 è computazionalmente inammissibile calcolare \hat{m} (cioè $s || t$), si sta solo cercando di recuperare una parte di \hat{m} e 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.

Crittosistemi a chiave pubblica. 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 , esegue $GenRSA(1^n)$ ed ottiene (N, e, d) . Manda in output la chiave pubblica $pk = \langle N, e \rangle$ e la chiave privata $sk = \langle N, d \rangle$;
La parte che genera la chiave, specifica una funzione $H: Z_N^* \rightarrow \{0, 1\}^n$, ma questo non ha nessun impatto.
- **Encaps:** prende in input la chiave pubblica $pk = \langle N, e \rangle$ e il parametro di sicurezza 1^n , sceglie in modo uniforme $r \in Z_N^*$. Manda in output $c := [r^e \bmod N]$ e la chiave $k := H(r)$;
- **Decaps:** prende in input la chiave privata $sk = \langle N, d \rangle$ e un cifrato $c \in Z_N^*$, computa $r := [c^d \bmod N]$ 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 $GenRSA$ e la funzione H scelta è ROM (random oracle model) allora il KEM risultante è CCA-sicuro.

Dimostrazione:

Sia A un avversario PPT e consideriamo $KEM_{A, \Pi}^{cca}(n)$. 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 $H: Z_N^* \rightarrow \{0, 1\}^n$ per eseguire l'operazione di derivazione;
2. Seleziona un valore casuale $r \in Z_N^*$, calcola il cifrato $c := [r^e \bmod N]$ 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 $H(\cdot)$ e 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 $\langle pk, c, \hat{k} \rangle$, effettua una query giusta all'oracolo $H(r) = \hat{k}$, ottenendo la chiave corrispondente per la query r per poi calcolare $c = r^e \bmod N$ sarebbe certo che la chiave \hat{k} è la chiave di incapsulamento del cifrato c .

Indichiamo gli eventi $Query = \{A \text{ fa la query } r \text{ all'oracolo } H\}$ e $Success = \{b' = b, \text{ ovvero l'evento in cui l'avversario vince}\}$.

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 all'evento \overline{Query} , l'avversario non fa query r all'oracolo H , 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:

$$\leq \Pr[Success \mid \overline{Query}] = 1/2$$

Mentre per dimostrare che $\Pr[Query] \leq \text{negl}(n)$, si 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 $KEM_{A,\Pi}^{cca}(n)$.

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 $sk = \langle N, d \rangle$, cosa che A' non dispone così gli invia valori casuali;

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:

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

Unito con il risultato della dimostrazione dell'evento $\Pr[Success \mid \overline{Query}]$ avremo che il KEM è CCA-sicuro.

Ripasso schemi di firma digitale, paradigma hash-and-sign e produzione di una contraffazione:

Gli **schemi di firma digitale** possono essere visti come l'analogo dei MAC nel contesto simmetrico. Quando Alice intende inviare un messaggio firmato a Bob calcola sul messaggio m un secondo valore σ utilizzando la chiave privata. La parte B, una volta ricevuta la coppia (m, σ) , utilizza la chiave pubblica di A pk_A per verificare che l'elemento σ è la firma che è stata apposta da A sul messaggio m . Quindi le chiavi vengono utilizzate nel modo seguente:

- 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 volessimo firmare lunghi messaggi in maniera diretta con uno schema di firma, il costo sarebbe 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*:

$$m \in \{0,1\}^*, H(m) \in \{0,1\}^l, \text{Sign}_{sk}(H(m))$$

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.

Schemi di firme digitali. Si spieghi in modo chiaro e conciso 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 $\text{Vrf}_{pk}(\cdot)$ dà esito positivo e il messaggio non appartiene alle query precedenti effettuati dall'avversario all'oracolo.

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

1. Viene eseguito $\text{Gen}(1^n)$ per ottenere (pk, sk) ;
2. L'avversario A prende pk e ha accesso ad un oracolo $\text{Sign}_{sk}(\cdot)$. L'avversario manda in output (m, σ) . Sia Q l'insieme di tutte le query che A chiede all'oracolo;
3. A ha successo se e solo se $\text{Vrf}_{pk}(m, \sigma) = 1$ e $m \notin Q$.

Definizione 12.2. Uno schema di firma digitale $\Pi = (\text{Gen}, \text{Sign}, \text{Vrf})$ è essenzialmente non falsificabile da un attacco adattivo del messaggio scelto, (quindi non falsificabile rispetto ad attacchi **adattivi di tipo chosen-message**), o semplicemente sicuro, se per ogni avversario A PPT, esiste una funzione trascurabile negl tale che:

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

Ripasso produzione contraffazioni di firme con RSA:

Costruzione 12.5. Si definisce uno schema di firma come segue:

- *Gen*: prende in input 1^n esegue $GenRSA(1^n)$ per ottenere (N, e, d) . La chiave pubblica $pk = \langle N, e \rangle$ e la chiave privata $sk = \langle N, d \rangle$;
- *Sign*: prende in input la chiave privata $sk = \langle N, d \rangle$ e il messaggio $m \in \mathbb{Z}_N^*$, computa la firma:
$$\sigma := [m^d \bmod N]$$
- *Vrfy*: prende in input la chiave pubblica $pk = \langle N, e \rangle$, un messaggio $m \in \mathbb{Z}_N^*$ e una firma $\sigma \in \mathbb{Z}_N^*$, manda in output 1 se e solo se:
$$m = [\sigma^e \bmod N].$$

Poiché il problema RSA è ritenuto difficile, si potrebbe pensare che sia difficile **produrre contraffazioni**. In realtà, è possibile produrre contraffazioni rispetto a RSA, mediante un attacco che non utilizza l'oracolo di firma, ma solo la chiave pubblica.

Data la chiave pubblica (N, e) , sceglie un valore a caso $\sigma \in \mathbb{Z}_N^*$, calcola il messaggio $m = [\sigma^e \bmod N]$ 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. 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 $m = m_1 * m_2 \bmod N$ per poi inviarli come query all'oracolo di firma ottenendo le firme σ_1 e σ_2 . L'avversario prende le due firme per fare l'operazione modulare $\sigma = [\sigma_1 \sigma_2 \bmod N]$ su m . Quindi la contraffazione sarà (m, σ) . Risulta:

$$\sigma^e = (\sigma_1 \sigma_2)^e = (m_1^d * m_2^d)^e = (m_1^{de} * m_2^{de}) = m_1 * m_2 = m \bmod N$$

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

(+++) **Schemi di firme digitali.** 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.

L'**RSA Full Domain Hash (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*: prende in input 1^n esegue $GenRSA(1^n)$ per ottenere (N, e, d) . La chiave pubblica $pk = \langle N, e \rangle$ e la chiave privata $sk = \langle N, d \rangle$. La parte che genera la chiave, specifica una funzione $H: \{0, 1\}^* \rightarrow \mathbb{Z}_N^*$, ma questo non ha nessun impatto.
- *Sign*: prende in input la chiave privata $\langle N, d \rangle$ e un messaggio $m \in \{0, 1\}^*$, computa:
$$\sigma := [H(m)^d \bmod N]$$
- *Vrfy*: prende in input la chiave pubblica $\langle N, e \rangle$, un messaggio m e una firma σ . Restituisce 1 se e solo se
$$\sigma^e = H(m) \bmod N.$$

Le **proprietà** che deve avere la **funzione H** è che deve essere: difficile da invertire, non ammettere relazioni moltiplicative e difficile trovare collisioni. Se la **funzione H** viene modellata come oracolo casuale, le tre condizioni sono soddisfatte.

Teorema. Se RSA è un problema difficile relativamente al *GenRSA*, che scegliamo nella costruzione dello schema di firma, e se H si comporta come un oracolo casuale allora la costruzione RSA-FDH è sicuro.

Sketch:

Supponendo che esista un A PPT in grado di produrre contraffazioni, ovvero coppie (m, σ) false, allora si potrebbe costruire un A altrettanto PPT che è in grado di risolvere il problema RSA, supposto difficile, con probabilità non trascurabile.

L'idea è che A' simula l'esperimento $Sig\text{-}forge_{A, \Pi}(n)$ 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 \mathbb{Z}_N^* , quando riceve m_1 risponde con $r_1 = (H(m_1))$.

A' opera in questo modo tranne nell' 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 $y = (H(m_i))$ 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, poiché 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 di cui ha chiesto $H(m)$ con probabilità $1/q$, in quanto 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:

$$\sigma^e = H(m) = H(m_i) = y \bmod N$$

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:

$$1/q * \text{non-negl}(n)$$

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 risolve 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:

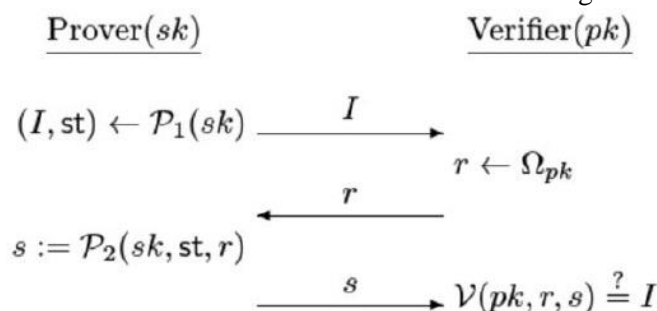
$$\sigma = H(m_j)^d \bmod N.$$

Quindi per simulare l'oracolo di firma, l'avversario A' , per m_j , sceglie un valore casuale σ_j e calcola $H(m_j) = \sigma_j^e \bmod N$ dove e è l'esponente pubblico. Da notare che σ_j è uniforme quindi si ha che $H(m_j) = \sigma_j^e \bmod N$ 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 $H(m)$ (hash). Con questa strategia, A' è in grado di rispondere consistentemente ad entrambi i tipi di query. Dopodiché se la query che riceve è per un hash, invia σ_j^e , se invece riceve una query di firma per l'hash σ_j^e 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' è in grado con probabilità non trascurabile più piccola della probabilità di contraffazione di invertire la permutazione RSA.

(+) Schemi di identificazione. Si spieghi in modo chiaro e conciso che cosa si intende per schema di identificazione in un sistema interattivo. Si fornisca l'esempio di uno schema interattivo in tre 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.

Ripasso esperimento $Ident_{A,\Pi}$:

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 identificarsi 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

$$Ident_{A,\Pi}$$

- Viene eseguito $Gen(1^n)$ per ottenere le chiavi (pk, sk) ;
- L'avversario A prende pk e ha accesso ad un oracolo $Trans_{sk}$ che può interrogare tutte le volte che vuole;
- 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);
- 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 A PPT, esiste una funzione trascurabile **negl** tale che:

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

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

Ripasso trasformazione di Fiat e Shamir:

La **trasformazione di Fiat e Shamir** offre un metodo per convertire uno schema di identificazione interattivo in uno schema di firma non interattivo. L'idea è questa: il Prover esegue il protocollo di identificazione da solo, rimuovendo l'interazione usando una funzione hash, cioè la challenge che di solito si riceve dal Verifier viene generato in maniera autonoma dal Prover tramite la funzione hash.

Costruzione 12.9. Sia (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 $H: \{0,1\}^* \rightarrow \Omega_{pk}$, 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:** prende 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.

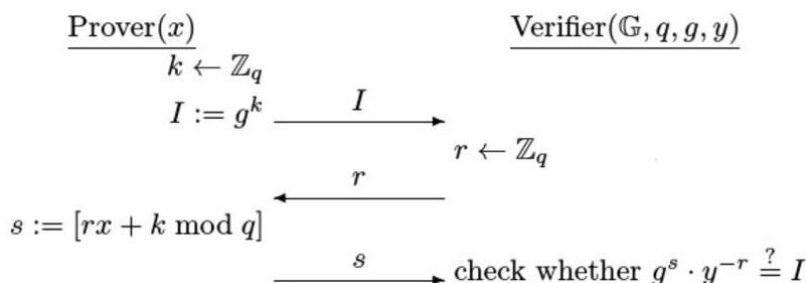
Una firma (r, s) è legata ad un messaggio specifico m perché r è una funzione sia di I che di m , quindi ogni volta che si va a cambiare m si ottiene un r diverso. Inoltre, se H viene modellata come un oracolo casuale che mappa gli input uniformemente su Ω_{pk} , allora r è uniformemente distribuito. 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.

Schemi di identificazione. Si descriva lo schema di identificazione di Schnorr e se ne discuta la sicurezza. Inoltre, si spieghi come possa essere convertito in uno schema di firma digitale.

Lo **schema di identificazione di Schnorr** funziona come segue:

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 $x \in \mathbb{Z}_q$ uniformemente a caso. Infine, calcola $y = g^x$. La chiave pubblica e privata sono:

$$pk = (G, q, g, y) \quad sk = (G, q, g, x)$$



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:

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

Per quanto riguarda la **sicurezza**, quando l'avversario gioca può saltare la fase di interrogazione dell'oracolo, usato nell'esperimento $Ident_{A, \mathcal{H}}(n)$, e 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.

L'idea può essere effettuata invertendo l'ordine dei passi, quindi potrebbe scegliere prima indipendentemente ed uniformemente a caso i valori $r, s \in \mathbb{Z}_q$ 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 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 $r_1, r_2 \in \mathbb{Z}_q$. 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:

$$g^{s_1} \cdot y^{-r_1} = I = g^{s_2} \cdot y^{-r_2} \rightarrow g^{s_1 - s_2} = y^{r_1 - r_2}$$

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

$$g^{(s_1-s_2)(r_1-r_2)^{-1}} = y^{(r_1-r_2)(r_1-r_2)^{-1}} \rightarrow y = g^{(s_1-s_2)(r_1-r_2)^{-1}}$$

$$\rightarrow \log_g y = [(s_1 - s_2)(r_1 - r_2)^{-1} \bmod q]$$

ovvero il logaritmo discreto di y , contraddicendo la difficoltà del problema DL.

Teorema. Se DL è difficile in G allora lo schema di Schnorr è sicuro.

Essendo DL difficile e utilizzando la trasformata di Fiat-Shamir, si può produrre lo **schema di firma di Schnorr**.

Costruzione 12.12. Sia:

- **Gen:** esegue $G(1^n)$ ottenendo (G, q, g) . Sceglie in modo uniforme $x \in \mathbb{Z}_q$ 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 $H: \{0,1\}^* \rightarrow \mathbb{Z}_q$, 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 $k \in \mathbb{Z}_q$ e imposta $I := g^k$. Poi computa $r := H(I, m)$, seguito da $s := [rx + k \bmod q]$. Da in output la firma (r, s) ;
- **Vrfy:** prende in input la chiave pubblica (G, q, g, y) , un messaggio m e la firma (r, s) , computa $I = g^s * y^{-r}$ e da in output 1 se $H(I, m) = r$.

Schemi di identificazione. Si descriva lo schema di identificazione di Schnorr e se ne discuta la sicurezza.

Opzionale: guardando lo schema come un sistema di prova, in cui il provatore dà prova della propria identità, se il verificatore è **onesto**, cioè esegue il protocollo scegliendo la challenge r in accordo alla distribuzione *uniforme*, risulta lo schema, per questo caso, a conoscenza zero?

Argomentare la risposta.

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 $\langle P, V^* \rangle$ deve essere indistinguibile dal transcript che produce il simulatore S .