Sicurezza e Crittografia

Anno Accademico 2018-2019

Homework 2

Matteo Berti

27 settembre 2019

Esercizio 1.
Quello che si cerca di costruire è un generatore pseudocasuale in cui: (per qualsiasi n), $\forall s \in \{0,1\}^n$, si ha $G_F^p(s) : \{0,1\}^n \to \{0,1\}^{p(n)}$. Si può costruire $G_F^p(s)$ come:

$$G_F^p(s) = F_s(1^n) \,||\, F_s(2^n) \,||\, ... \,||\, F_s(\left\lfloor \frac{p(n)}{n} \right\rfloor^n)$$

La dicitura W^n con $1 \leq W \leq \frac{p(n)}{n}$ negli input delle funzioni pseudocasuali, prevede che il numero naturale W sia convertito in una stringa binaria lunga esattamente n bit. Quindi si avranno $\frac{p(n)}{n}$ blocchi concatenati di lunghezza n ottenendo $|G_F^p(s)| = p(n)$. Vengono passati in input alle funzioni pseudocasuali gli elementi da 1 a $\frac{p(n)}{n}$ in modo che producano sempre output diversi (tra loro) e che per un $p' < p$ la stringa $G_F^{p'}(s)$ sia un prefisso di $G_F^p(s)$.
Un problema che può sorgere è il caso in cui il polinomio p non sia divisibile per n (banalmente $p(n) = n+5$). In questo caso si prende il resto della divisione $p(n)\ mod\ n = q$ e si aggiunge in coda a $G_F^p(s)$ un'ulteriore funzione pseudocasuale che copra i q bit rimanenti: ... $||\ F_{cut(s)}([\left\lfloor \frac{p(n)}{n} \right\rfloor + 1]^q)$ in cui chiaramente $cut(s)$ restituisce i primi q bit di s.

Quel che si vuole dimostrare ora è che, dato un distinguitore D che distingua tra l'output di generatori pseudocasuali e stringhe realmente casuali valga:

$$|Pr(D(r) = 1) - Pr(D(G_F^p(s)) = 1)| \leq \epsilon(n)$$

Dati r una stringa realmente random tale che $|r| = p(n)$, s tale che $|s| = n$ ed $\epsilon(n)$ trascurabile.
Si sa che un distinguitore che distingue funzioni pseudocasuali da funzioni casuali utilizza necessariamente un oracolo (in quanto la descrizione di una funzione random ha lunghezza esponenziale e il distinguitore può lavorare al massimo in tempo polinomiale). Si può costruire un distinguitore D_* simile a quello appena citato, con la differenza che invece di utilizzare l'oracolo una sola volta sull'input, concatena $\frac{p(n)}{n}$ chiamate all'oracolo, con n passato come parametro nella forma 1^n. In questo modo si ottiene una stringa w lunga $p(n)$ come risultato delle $\frac{p(n)}{n}$ chiamate all'oracolo. Poiché F_s è una funzione pseudocasuale, vale:

$$|Pr(D_*^{f(\cdot)}(1^n) = 1) - Pr(D_*^{F_s(\cdot)}(1^n) = 1)| \leq \epsilon(n)$$

Se supponiamo che $G_F^p(s)$ non sia un generatore pseudocasuale valido, allora il distinguitore D riuscirebbe facilmente a distinguere tra il risultato di $G_F^p(s)$ e una stringa realmente casuale. Tuttavia la probabilità che D_* riesca a distinguere una funzione pseudocasuale, da una realmente casuale, concatenando i risultati dell'oracolo può essere rappresentata come la probabilità che il

distinguitore di generatori pseudocasuali distingua i risultati concatenati di $\frac{p(n)}{n}$ chiamate a funzioni pseudocasuali sugli stessi input usati per le chiamate all'oracolo:

$$= |Pr(D(f(1^n) \mid\mid ... \mid\mid f([\tfrac{p(n)}{n}]^n) \mid\mid f([\left\lfloor \tfrac{p(n)}{n} \right\rfloor + 1]^q)) = 1) \; -$$
$$Pr(D(F_s(1^n) \mid\mid ... \mid\mid F_s([\tfrac{p(n)}{n}]^n) \mid\mid F_{cut(s)}([\left\lfloor \tfrac{p(n)}{n} \right\rfloor + 1]^q)) = 1)|$$

Che equivale a distinguere il risultato del generatore pseudocasuale $G_F^p(s)$ da una concatenazione di valori realmente casuali. Tuttavia si è assunto $G_F^p(s)$ fosse facilmente distinguibile, ovvero fosse possibile dinguerlo da valori casuali con probabilità non trascurabile, in questo modo si è appena visto avere uguale probabilità trascurabile rispetto all'utilizzo di D_*, si ha quindi l'assurdo.

Per questo si può confermare che $G_F^p(s)$ sia un generatore pseudocasuale valido.

Esercizio 2.
Per verificare la sicurezza di Π_F^2 è necessario costruire un avversario A^*, che usi $MacForge_{A^*,\Pi}(n)$ e che abbia una probabilità di riuscita non trascurabile. Sia $MacForge_{A^*,\Pi}(n)$ l'esperimento tale per cui se $Pr(MacForge_{A^*,\Pi}(n) = 1) \leq \epsilon(n)$ allora un MAC Π si dice sicuro.

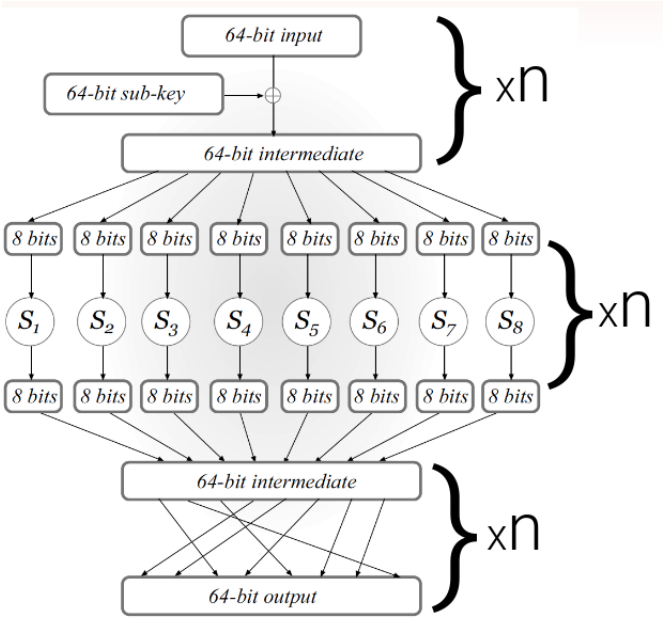
A^* nell'esperimento $MacForge_{A^*,\Pi}(n)$ deve forgiare un tag corretto per un messaggio $m = m_0 \mid\mid m_1$ con $|m_0| = |m_1| = |k| - 1$. Quello che fa, avendo a disposizione l'oracolo $Mac_k(\cdot)$, è chiamare $Mac_k(m_0 \mid\mid m_1^*) = t_0$ in cui m_1^* altro non è che m_1 a cui è stato invertito un bit a caso. Il risultato t_0 ottenuto va ripulito della seconda metà del tag, che è incorretta, quindi lo si divide a metà e se ne prende solo la prima parte: $t_0 = getFirstHalf(t_0)$;. Il secondo passaggio è chiamare l'oracolo con $Mac_k(m_0^* \mid\mid m_1) = t_1$, anche qui m_0^* altro non è che m_0 a cui è stato invertito un bit a caso. Nuovamente, va ripulito t_1 togliendo questa volta la prima parte e tenendo la seconda metà: $t_1 = getSecondHalf(t_1)$;.
Infine è sufficiente concatenare $t = t_0 \mid\mid t_1$ per ottenere il tag corretto per il messaggio $m = m_0 \mid\mid m_1$. Il tutto senza aver mai chiamato l'oracolo $Mac_k(m)$ sul messaggio di cui si vuole forgiare il tag.

La probabilità di successo dell'esperimento è quindi:

$$Pr(MacForge_{A^*,\Pi}(n) = 1) = 1$$

Ciò rende Π_F^2 insicuro.

Esercizio 3.



Nello schema sopra, la fase di mixing dell'input con la chiave viene eseguita n volte consecutive, dato un messaggio M il messaggio intermedio M_{inter} che esce da questa fase avrà la forma:

$$M_{inter} = M \oplus K_{sub_1} \oplus K_{sub_2} \oplus ...$$

Viene infatti applicato lo xor tra il messaggio ed ogni sotto-chiave.
Successivamente saranno applicate ad M_{inter} anche le n iterazioni di $S-BOX$ e successivamente le n permutazioni.

Si sa che le S-BOX e le permutazioni sono invertibili e per il principio di Kerckhoffs il loro funzionamento deve essere noto. Questo permette di invertire il procedimento di entrambe e risalire ad M_{inter}. Un avversario banalmente non farà altro che applicare all'output n permutazioni inverse, e successivamente n $S-BOX$ inverse per così ottenere M_{inter}, il quale se eseguito lo xor con il messaggio M inizialmente passato come input restituirà la chiave completa K:

$$K = M \oplus M_{inter} = M \oplus M \oplus K_{sub_1} \oplus K_{sub_2} \oplus ...$$