

# Costruzioni teoriche

Paolo D'Arco  
pdarco@unisa.it

Università di Salerno

Elementi di Crittografia

- 1 Funzioni One-Way
- 2 Predicati Hard-core
- 3 Overview dei risultati

Pseudocasualità in tre forme

- Generatori PRG
- Funzioni PRF
- Permutazioni PRP

Blocchi di base per tutta la crittografia a chiave privata

**Funzioni one-way: facili da calcolare, difficili da invertire**

Se esistono, é possibile costruire PRG, PRF e PRP.

Si puó dimostrare che le funzioni one-way sono necessarie per la crittografia a chiave privata.

# One-way Function

Sia  $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$  una funzione. Siano  $A$  ppt,  $n$  par. di sicurezza.

$Invert_{A,f}(n)$

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

Nota:  $A$  non deve trovare necessariamente  $x$ . Basta una pre-immagine  $x'$  tale che  $f(x') = y = f(x)$

**Definizione 8.1.** Una funzione  $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$  é one way se

- 1 (Facile da calcolare): esiste un algoritmo di tempo polinomiale  $M_f$  per calcolare  $f$ , i.e., per ogni  $x$  risulta  $M_f(x) = f(x)$
- 2 (Difficile da invertire): per ogni  $A$  ppt, esiste una funzione trascurabile  $negl(n)$  tale che

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

Nota: il secondo requisito equivale a dire che:  $\forall A$  ppt,  $\exists negl(n)$  :

$$Pr_{x \leftarrow \{0,1\}^n}[A(1^n, f(x)) \in f^{-1}(f(x))] \leq negl(n)$$

# One-way Function

Una funzione che non é one-way **non** é necessariamente facile da invertire sempre. Piuttosto, l'inverso é:

$\exists$  A ppt ed una funzione  $\gamma$  non trascurabile tale A inverte  $f(x)$  con prob. almeno  $\gamma(n)$



$\exists$  un polinomio positivo  $p(\cdot)$  tale che, per infiniti valori di  $n$ , A inverte  $f(x)$  con prob. almeno  $1/p(n)$

Per esempio, se esiste A che inverte  $f$  con prob.  $n^{-10}$  per tutti i valori pari di  $n$  (ma fallisce sui dispari), allora  $f$  non é one-way.

# One-way Function

In tempo esponenziale, ogni funzione one-way può essere invertita in ogni punto, provando tutti i possibili  $x \in \{0, 1\}^n$



L'esistenza delle funzioni one-way é inerentemente una assunzione circa la difficoltà *computazionale*, i.e., riguarda un problema risolvibile in principio, supposto difficile da risolvere efficientemente

Spesso saremo interessati a funzioni one-way con proprietà strutturali aggiuntive

$f$  preserva la lunghezza se  $|f(x)| = |x|$ , per ogni  $x$

Una funzione one-way che preserva la lunghezza ed è 1-a-1 è una permutazione one-way



Ogni valore  $y$  ha un'unica pre-immagine  $x = f^{-1}(x)$ . Nonostante ciò, è ancora difficile trovare  $x$  in tempo polinomiale.

# One-way Function

La definizione precedente considera una singola funzione su dominio e codominio infiniti.

Tuttavia molte papabili funzioni one-way sono definite per certi parametri  $I$ .

La funzione  $f_I$  dovrebbe essere one-way sulle scelte di  $I$ .

Conviene considerare allora famiglie di funzioni (permutazioni) one-way.

**Definizione 8.2.** Una tripla  $\Pi = (Gen, Samp, f)$  di algoritmi ppt é una famiglia di funzioni se:

- 1 L'algoritmo di generazione dei parametri  $Gen(1^n)$  dá in output  $I$ , con  $|I| \geq n$ . Ciascun  $I$  definisce  $D_I$  ed  $R_I$  tali che

$$f_I : D_I \rightarrow R_I$$

- 2 L'algoritmo di campionamento  $Samp(I)$  dá in output un elemento uniformemente distribuito di  $D_I$
- 3 L'algoritmo di valutazione  $f$  deterministico, su input  $I$  ed  $x \in D_I$ , dá in output  $y \in R_I$ . Scriveremo  $y := f_I(x)$

# Famiglia di permutazioni

$\Pi$  é una famiglia di permutazioni se, per ogni  $I \leftarrow \text{Gen}(1^n)$ , risulta  $D_I = R_I$  e la funzione  $f_I : D_I \rightarrow R_I$  é una biiezione.

L'esperimento  $\text{Invert}_{A,\Pi}(n)$  é definito come segue:

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

- 1 Il challenger sceglie  $I \leftarrow \text{Gen}(1^n)$ ,  $x \leftarrow \text{Samp}(I)$  e calcola  $y := f_I(x)$ .
- 2  $A$  riceve in input  $I$  e  $y$  e dà in output  $x'$
- 3 L'output dell'esperimento è 1 se  $f_I(x') = y$ ; altrimenti, 0.

**Definizione 8.3.** Una famiglia di funzioni (permutazioni)  $\Pi = (Gen, Samp, f)$  é one way se, per ogni  $A$  ppt, esiste una funzione trascurabile  $negl(n)$  tale che

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

△

Allo stato attuale delle nostre conoscenze non sappiamo se esistono.

Congetturiamo (assumiamo) la loro esistenza.

Congettura basata sul fatto che diversi problemi computazionali naturali hanno ricevuto notevole attenzione ma non sono stati trovati algoritmi risolutivi di tempo polinomiale.

Problema piú famoso, fattorizzazione di interi.

É facile prendere due numeri e farne il prodotto.

Difficile prendere un intero grande e trovarne i fattori

$$f_{mult}(x, y) = x \cdot y$$

Se non poniamo alcuna restrizione,  $f_{mult}$  é facile da invertire.

Con alta probabilitá  $x \cdot y$  é pari e quindi  $(2, \frac{x \cdot y}{2})$  é un inverso per  $f_{mult}(x, y)$

Restringeremo il dominio a  $x$  e  $y$  *primi grandi e della stessa lunghezza*.

Un'altra funzione papabile é basata sul problema Subset Sum ed é definita da

$$f_{SS}(x_1, \dots, x_n, J) = (x_1, \dots, x_n, \sum_{j \in J} x_j \bmod 2^n),$$

dove

- $x_1, \dots, x_n$  sono stringhe di  $n$  bit viste come interi
- $J$  stringa di  $n$  bit che specifica un sottoinsieme di  $\{1, \dots, n\}$

Invertire  $f_{SS}$  su  $(x_1, \dots, x_n, y)$  significa trovare un

$$Z \subseteq \{1, \dots, n\} \text{ tale che } y = \sum_{j \in Z} x_j \bmod 2^n$$

Ricordo che Subset Sum é NP-Completo.

# $\mathcal{P} \neq \mathcal{NP}$ necessario ma non sufficiente

Nota però che:

$\mathcal{P} \neq \mathcal{NP}$  **non implica** che  $f_{SS}$  é one-way

$\mathcal{P} \neq \mathcal{NP} \Rightarrow$  ogni algoritmo di tempo polinomiale fallisce nel risolvere il problema Subset Sum su **almeno** un input.

D'altra parte  $f_{SS}$ , per essere one-way, richiede che ogni algoritmo di tempo polinomiale fallisca nel risolvere il problema (almeno per certi parametri) **quasi sempre**.

Pertanto, la nostra convinzione che  $f_{SS}$  sia one-way é basata sul fatto che non conosciamo algoritmi polinomiali che risolvono il problema **anche con probabilità bassa** su **istanze casuali**, piuttosto che sul fatto che il problema sia  $\mathcal{NP}$  Completo.

# Possibile famiglia di permutazioni one-way

Sia  $\Pi = (Gen, Samp, f)$  definita come segue

- $Gen(1^n) \rightarrow (p, g)$ , dove  $p$  é un primo di  $n$  bit e  $g \in \{2, \dots, p-1\}$  é un elemento speciale detto generatore di  $Z_p^*$  (ci torneremo ...)
- $Samp(p, g) \rightarrow x \in \{1, \dots, p-1\}$  restituisce un  $x$  scelto uniformemente
- $f_{p,g}(x) = [g^x \bmod p]$

$f_{p,g}$  risulta efficientemente calcolabile e 1-a-1, e dunque é una permutazione.

La difficultá di inversione é basata sulla difficultá congetturata del **problema del logaritmo discreto**.

Dire che una funzione é one-way significa dire che é difficile da invertire: dato  $y = f(x)$ , il valore  $x$  non puó essere calcolato **nella sua interezza** in tempo polinomiale.

Non significa che **nulla** possa essere calcolato su  $x$  in tempo polinomiale!

Per esempio, se  $g$  é one-way, allora

$$f(x_1, x_2) = (x_1, g(x_2)), \quad \text{dove } |x_1| = |x_2|$$

risulta one-way anche se rivela metà del suo input.

In molte applicazioni abbiamo bisogno di identificare un pezzo di informazione relativo ad  $x$  "nascosto da  $f(x)$ ."

Un predicato hard-core  $hc : \{0, 1\}^* \rightarrow \{0, 1\}$  di  $f$  ha la proprietà che  $hc(x)$  è difficile da calcolare con probabilità significativamente migliore di  $1/2$ , dato  $f(x)$ .

In modo più formale:

**Definizione 8.4.** Una funzione  $hc : \{0, 1\}^* \rightarrow \{0, 1\}$  è un predicato hard-core di una funzione  $f$  se  $hc$  può essere calcolato in tempo polinomiale e, per ogni  $A$  ppt, esiste una funzione trascurabile  $negl(n)$  tale che

$$Pr_{x \leftarrow \{0,1\}^n} [A(1^n, f(x)) = hc(x)] \leq 1/2 + negl(n)$$

Non è facile costruire predicati hard-core.

Si consideri

$$hc(x) = \bigoplus_{i=1}^n x_i \quad \text{dove } x = x_1 \dots x_n$$

Idea alla base:  $f$  one-way  $\Rightarrow$  almeno un bit di  $x_i$  é nascosto da  $f(x)$



$hc$  é un predicato hard-core **per qualsiasi** funzione one-way.

Purtroppo non funziona ...

Sia  $g$  una funzione one-way e sia

$$f(x) = (g(x), \bigoplus_{i=1}^n x_i)$$

$f$  é one-way ma non nasconde  $\bigoplus_{i=1}^n x_i$ , che é parte del suo output!

Pertanto,  $hc$  non é un predicato hard-core per  $f$

Nota: alcune funzioni hanno predicati hard-core banali

$$f(x_1, \dots, x_n) = x_1, \dots, x_{n-1} \quad (\text{butta via l'ultimo bit di input})$$

É difficile determinare  $x_n$  dato  $f(x_1, \dots, x_n)$  poiché  $x_n$  é indipendente da  $f$ .

La funzione  $f$  non é neanche one-way.

Nelle costruzioni crittografiche, predicati di questo tipo non sono utili.

Domanda: *esiste un predicato hard-core per qualsiasi funzione one-way?*

Non ha ancora risposta, ma possiamo mostrare qualcosa di piú debole, utile ai nostri scopi

**Teorema 8.5 (Goldreich-Levin).** Supponiamo che le funzioni (permutazioni) one-way esistano. Allora esiste una funzione (permutazione) one-way  $g$  ed un predicato hardcore  $hc$  di  $g$ .

Sia  $f$  una funzione one-way. Le funzioni  $g$  ed  $hc$  sono costruite come segue

$$g(x, r) = (f(x), r), \quad \text{dove} \quad x = x_1 \dots x_n, \quad r = r_1 \dots r_n$$

$$hc(x, r) = \bigoplus_{i=1}^n x_i \cdot r_i, \quad \text{dove } x_i \text{ (resp. } r_i) \text{ é } i\text{-esimo bit di } x \text{ (resp. } r)$$

Se  $r$  é uniforme, allora  $hc(x, r)$  é l'xor di un sottoinsieme casuale di bit di  $x$ .  
Essenzialmente il teorema di Goldreich-Levin afferma che, se  $f$  é one-way, allora  $f(x)$  nasconde l'xor di un sottoinsieme casuale dei bit di  $x$ .

Mostriamo come un predicato hard-core di una permutazione one-way possa essere usato per costruire un generatore pseudo-casuale

**Teorema 8.6.** Sia  $f$  una permutazione one-way, e sia  $hc$  un predicato hardcore di  $f$ . Allora

$$G(s) = f(s) || hc(s)$$

è un generatore pseudocasuale con fattore di espansione  $\ell(n) = n + 1$ .

Perché funziona? Intuizione:

- $s$  uniforme  $\Rightarrow f(s)$  uniformemente distribuito, essendo  $f$  una permutazione, ovvero i primi  $n$  bit di  $G(s)$  sono uniformi
- $hc(s)$  predicato hard-core di  $f \Rightarrow hc(s)$  sembra casuale dato il valore di  $f(s)$

Pertanto, l'intero output di  $G$  è pseudocasuale

Per applicazioni abbiamo bisogno di PRG con espansione maggiore

**Teorema 8.7.** Se esiste un PRG con fattore di espansione  $\ell(n) = n + 1$  allora, per ogni polinomio  $poly(n)$ , esiste un PRG con fattore di espansione  $poly(n)$ .

Pertanto, PRG con fattori di espansione arbitraria possono essere costruiti a partire da una qualsiasi permutazione one-way.

Ricordo che

- PRG sono sufficienti per costruire schemi di cifratura a chiave privata EAV-sicuri
- Per schemi di cifratura CPA-sicuri e schemi di autenticazione di messaggi occorrono PRF.

**Teorema 8.8.** Se esiste un PRG con fattore di espansione  $\ell(n) = 2n$ , allora esiste una PRF.

In realtà possiamo mostrare un risultato piú forte

**Teorema 8.9.** Se esiste una PRF, allora esiste una PRP forte.

Discendono immediatamente i seguenti corollari

**Corollario 8.10.** Assumendo l'esistenza di **permutazioni one-way**, esistono PRG con fattore di espansione polinomiale, PRF e PRP forti

$$OWP \Rightarrow PRG, PRF, SPRP$$

**Corollario 8.11.** Assumendo l'esistenza di **permutazioni one-way**, esistono schemi di cifratura a chiave privata *CCA*-sicuri e schemi di autenticazione di messaggi sicuri

$$OWP \Rightarrow CCA - \text{secure encryption, secure} - MAC$$

È possibile dimostrare (ma è più complicato) che gli stessi risultati valgono a partire dalla sola esistenza di **funzioni one-way** (OWF invece di OWP).

**Teorema 8.5 (Goldreich-Levin).** Supponiamo che le funzioni (permutazioni) one-way esistano. Allora esiste una funzione (permutazione) one-way  $g$  ed un predicato hardcore  $hc$  di  $g$ .

Il risultato viene provato "per passi".

Se esiste  $A$  ppt che calcola **sempre correttamente**

$$gI(x, r) = hc(x, r) \text{ data } g(x, r) = (f(x), r),$$

per ogni  $n$  e per ogni  $x, r \in \{0, 1\}^n$ , allora esiste  $A'$  tale che

$$A'(1^n, f(x)) = x \quad \forall n \text{ e } \forall x \in \{0, 1\}^n$$

Questo é il caso semplice.

$A'(1^n, y)$  calcola  $x_i = A(y, e^i)$ , per  $i = 1, \dots, n$ , e dá in output  $x_1, \dots, x_n$

# Teorema di Goldreich-Levin

Infatti, essendo  $e_i$  la stringa di  $n$  bit con un solo 1 in posizione  $i$ ,

$$x_i = A(f(x^*), e^i) = gl(x^*, e^i) = \bigoplus_{j=1}^n x_j^* \cdot e_j^i = x_i^*$$

Pertanto,  $x_i = x_i^*$ , per tutti gli  $i$ , ed  $A'$  dá in output l'inversa corretta  $x$ .

Il risultato generale viene provato mostrando

- prima il caso in cui  $A$  calcola con **prob.**  $> 3/4$  il predicato  $gl(x, r)$ . Questa analisi é complicata
- poi, affinando ulteriormente la prova, che se  $A$  calcola con **prob. non trascurabile**  $gl(x, r)$ , allora  $A'$  inverte con prob. non trascurabile.

Se interessati, i dettagli sono nella sottosezione 8.3.

# PRG con espansione polinomiale

Supponiamo di disporre di un PRG  $G : \{0, 1\}^n \rightarrow \{0, 1\}^{n+1}$

Usiamo  $G$  "ripetutamente" per costruire  $\hat{G}$  come segue. Per capire l'idea, costruiamo  $\hat{G} : \{0, 1\}^n \rightarrow \{0, 1\}^{n+2}$

Dato  $s \in \{0, 1\}^n$ , calcola  $t_1 = G(s)$ .

Degli  $n + 1$  bit di  $t_1$ , cioè  $t_1^1 \dots t_1^n t_1^{n+1}$ , i primi  $n$  vengono usati come seme per  $G$ .

Vale a dire, ponendo  $t = t_1^1 \dots t_1^n$ , calcola  $w_2 = G(t)$ .

Agli  $n + 1$  bit di  $w_2$ , cioè  $w_2^1 \dots w_2^{n+1}$ , viene concatenato  $t_1^{n+1}$ .

Pertanto,

$$\hat{G}(s) = w_2^1 \dots w_2^{n+1} t_1^{n+1}.$$

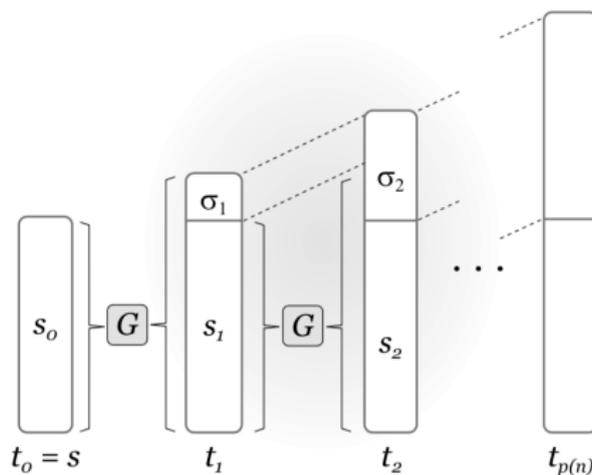
Indicando con  $p(n)$  il fattore di espansione voluto, il caso generale diventa:

1. Set  $t_0 := s$ . For  $i = 1, \dots, p(n)$  do:
  - (a) Let  $s_{i-1}$  be the first  $n$  bits of  $t_{i-1}$ , and let  $\sigma_{i-1}$  denote the remaining  $i - 1$  bits. (When  $i = 1$ ,  $s_0 = t_0$  and  $\sigma_0$  is the empty string.)
  - (b) Set  $t_i := G(s_{i-1})\|\sigma_{i-1}$ .
2. Output  $t_{p(n)}$ .

Ogni iterazione allunga "il suffisso"  $\sigma_i$  di un bit. L'ultima iterazione aggiunge  $n + 1$  bit "in testa".

# PRG con espansione polinomiale

Graficamente:



Se usiamo come PRG di partenza il generatore

$$G(s) = f(s) \| hc(s) \quad (f \text{ perm. one-way e } hc \text{ predicato hardcore})$$

allora la costruzione generale precedente dá luogo a

$$\hat{G} = f(f^{\ell-1}(s)) \| hc(f^{\ell-1}(s)) \| \dots \| hc(f^i(s)) \| \dots \| hc(s),$$

dove  $f^i$ , per  $i = 1, \dots, \ell$ , denota l'applicazione iterata  $i$  volte di  $f$ .

Il fattore di espansione del generatore é  $\ell + n$  bits.

Mostreremo come costruire una PRF a partire da un qualsiasi PRG che raddoppia la lunghezza dell'input.

Restringeremo l'attenzione a funzioni che preservano la lunghezza, i.e., per ogni  $k \in \{0, 1\}^n$ ,

$$F_k : \{0, 1\}^n \rightarrow \{0, 1\}^n$$

Per familiarizzare con la costruzione, consideriamo un caso semplice. Sia  $n = 2$ . Disponiamo di  $G : \{0, 1\}^2 \rightarrow \{0, 1\}^4$  e vogliamo realizzare

$$F : \{0, 1\}^2 \times \{0, 1\}^2 \rightarrow \{0, 1\}^2$$

Sia  $G(k) = G_0(k) || G_1(k)$  (cioé,  $G_0$  prima metà di  $G(k)$ ,  $G_1$  seconda)

Definiamo  $F$  come segue

$$F_k(00) = G_0(G_0(k)), \quad F_k(10) = G_0(G_1(k))$$

$$F_k(01) = G_1(G_0(k)), \quad F_k(11) = G_1(G_1(k))$$

Intuitivamente  $F$  é pseudocasuale perché, se  $G$  é un PRG, allora

- $G_0(k) || G_1(k)$  é indistinguibile da  $K_0 || K_1$  uniforme
- e, parimenti,

$$G_0(G_0(k)) || G_0(G_1(k)) || G_1(G_0(k)) || G_1(G_1(k))$$

é indistinguibile da

$$G_0(K_0) || G_0(K_1) || G_1(K_0) || G_1(K_1).$$

Il caso generale diventa quindi:

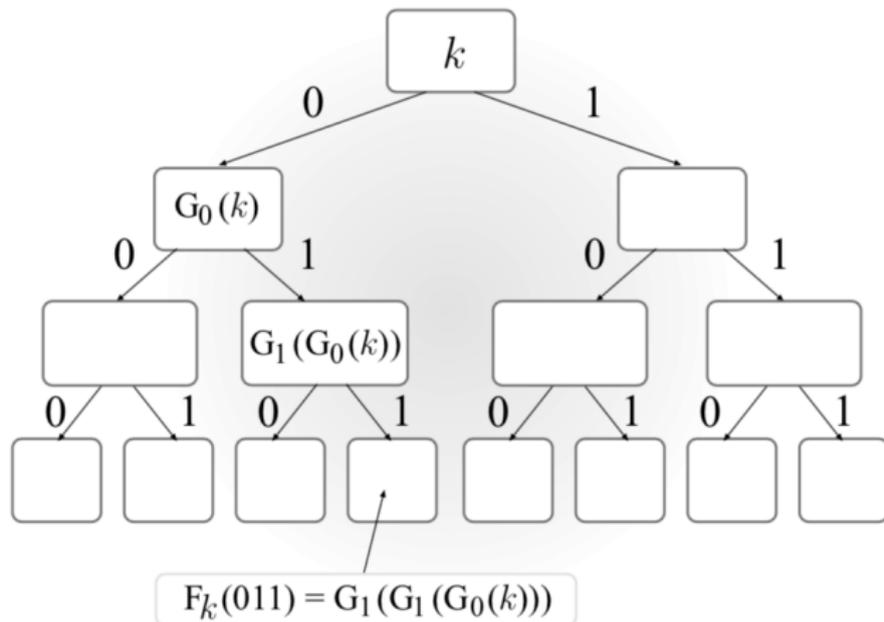
## **CONSTRUCTION 8.20**

Let  $G$  be a pseudorandom generator with expansion factor  $\ell(n) = 2n$ , and define  $G_0, G_1$  as in the text. For  $k \in \{0, 1\}^n$ , define the function  $F_k : \{0, 1\}^n \rightarrow \{0, 1\}^n$  as:

$$F_k(x_1 x_2 \cdots x_n) = G_{x_n} (\cdots (G_{x_2} (G_{x_1} (k))) \cdots).$$

# Costruzione di Goldreich-Goldwasser-Micali

Graficamente:



La taglia dell'albero é esponenziale in  $n$

Nonostante ciò si noti che per calcolare  $F_k$  **non** occorre costruire tutto l'albero e memorizzarlo. Bastano  $n$  valutazioni di  $G$

**Teorema 7.22.** Se  $G$  é un PRG con fattore di espansione  $\ell(n) = 2n$ , allora la Costruzione 7.21 produce una funzione pseudocasuale.

La dimostrazione utilizza una tecnica di prova elegante detta "dell'argomento ibrido."

É possibile costruire permutazioni pseudocasuali e permutazioni pseudocasuali forti usando una funzione pseudocasuale  $F$  e reti di Feistel a 3 round (PRP) e a 4 round (SPRP).

Per dettagli si vedano il Teorema 8.22 e la Costruzione 8.23.