Sicurezza: altre nozioni

Paolo D'Arco
pdarco@unisa.it

Universitá di Salerno

Elementi di Crittografia

Contenuti

Nozioni di sicurezza

Fino ad ora: Adv ascolta passivamente la trasmissione. Ha accesso ad *un* cifrato che due parti oneste si scambiano.

Sarebbe utile avere una nozione di sicurezza che permetta alle parti di inviare messaggi *multipli*.

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

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

Indistinguibilitá rispetto a messaggi multipli

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

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

dove la probabilitá é calcolata su

- randomness usata da A
- randomness usata nell'esperimento
  - scelta della chiave
  - scelta del bit b
  - random bit usati da $Enc_k(\cdot)$

Perché ci convince?

Rispetto al caso del messaggio singolo, i messaggi potrebbero essere legati tra di loro e le stesse cifrature potrebbero essere legate tra di loro e con i messaggi. E questi legami potrebbero essere efficientemente calcolabili e sfruttabili da un avversario.

Ma se l'avversario non riesce a capire se $(c_1, \ldots, c_t)$ corrisponda a $(m_{0,1}, \ldots, m_{0,t})$ o a $(m_{1,1}, \ldots, m_{1,t})$, ricordando l'equivalenza indistinguibilitá - semantica, allora lo schema di cifratura maschera molto bene i contenuti dei cifrati.

Indistinguibilitá rispetto a messaggi multipli

Osservazione immediata:

Π sicuro rispetto a $PrivK_{A,\Pi}^{eav-mult}(n) \Rightarrow \Pi$ sicuro rispetto a $PrivK_{A,\Pi}^{eav}(n)$ (caso speciale, un messaggio)

Possiamo far vedere che *non* é vero l'inverso con un controesempio.

Indistinguibilitá rispetto a messaggi multipli

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

Dim. Consideriamo lo schema **one-time pad** (in breve **otp**).

Segretezza perfetta $\Rightarrow$ cifrature indistinguibili.

Sia A l'Adv che segue, che attacca lo schema **otp** nell'esperimento $PrivK_{A,\text{otp}}^{eav-mult}(n)$

Indistinguibilitá rispetto a messaggi multipli

Adv A

1. Sceglie $M_0 = (0^\ell, 0^\ell)$ ed $M_1 = (0^\ell, 1^\ell)$ e li passa al Challenger
2. Riceve dal Challenger la lista di cifrati $c = (c_1, c_2)$
3. Se $c_1 = c_2$, dá in output $b' = 0$; altrimenti, $b' = 1$.

Analizziamo la probabilitá che $b' = b$. Lo schema **otp** é deterministico.

$$\begin{array}{lll} \text{se } b = 0, \text{ allora } c_1 = c_2 & \Rightarrow & A \text{ dá } 0 \\ \text{se } b = 1, \text{ allora } c_1 \neq c_2 & \Rightarrow & A \text{ dá } 1 \end{array}$$

Pertanto A vince con prob. 1 e **otp** *non* é sicuro rispetto alla Definizione 3.19.

Osservazione:

se uno schema di cifratura é deterministico, la cifratura dello stesso messaggio dá lo *stesso* cifrato $\Rightarrow$ la Definizione 3.19 richiede che la cifratura sia *probabilistica*.

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

Al momento sembra un requisito che confligge con l'operazione di decifratura $Dec()$

stesso messaggio cifrato piú volte $\Rightarrow$ cifrati diversi

... ma si puó fare!

Attacchi di tipo Chosen-Plaintext

Adv: ha l'abilitá di esercitare *controllo parziale* su ció che una parte onesta cifra

1. puó scegliere $m_1, m_2, \ldots, m_t$ e forzare Alice a cifrarli ed inviarli
2. puó osservare i cifrati corrispondenti $c_1, c_2, \ldots, c_t$ inviati da Alice sul canale
3. osserva un nuovo cifrato c, prodotto *autonomamente* da Alice, che diventa il target dell'attacco

Anche in questo caso, definiremo la sicurezza di uno schema in accordo alla nozione di indistinguibilitá

- al passo 3. il cifrato c corrisponde alla cifratura di uno tra $\{m_0, m_1\}$, *noti* ad Adv, e richiederemo l'incapacitá di Adv di capire *a quale dei due* effettivamente corrisponde

Attacchi di tipo Chosen-Plaintext

Nota: gli attacchi di tipo *known-plaintext* sono un caso particolare. Adv conosce ma *non sceglie* $m_1, \ldots, m_t$.

Sono una preoccupazione realistica? Sí!

Nel libro di testo vengono riportati due esempi storici:

- Tedeschi: seconda guerra mondiale (mine in *posizioni note* da parte degli Inglesi, *cifrate* dai tedeschi)
- Us Navy, battaglia di Midway, 1942 (ipotesi AF cifratura di Midway, confermata inducendo la cifratura di un messaggio ad hoc)

Leggeteli!

Esempio moderno: utente che digita dati ad un terminale, il terminale cifra prima di inviarli.

Attacchi di tipo Chosen-Plaintext

Come facciamo a modellare la capacitá di Adv di disporre di cifrati corrispondenti a messaggi di propria scelta?

Esperimenti

Abbiamo giá usato esperimenti per definire le nozioni di indistinguibilitá perfetta e computazionale

Molte definizioni in Crittografia vengono fornite utilizzando degli esperimenti, in cui un Challenger sfida un Adv che cerca di aver successo in un determinato task

Gli esperimenti permettono di *astrarre* e *modellare* scenari reali in modo semplice

Per modellare lo scenario di un attacco chosen-plaintext abbiamo bisogno di uno stratagemma per fornire ad *Adv* i cifrati corrispondenti ai messaggi che sceglie.

Esperimenti

Useremo un **oracolo** O: scatola nera che cifra messaggi usando una chiave k

- Adv non conosce k
- Adv invia richieste di cifratura, dette *query*, ad O specificando m ed ottenendo in risposta $Enc_k(m)$.
- se $Enc_k()$ é randomizzato, O usa random bit *nuovi* ogni volta che riceve una query
- Adv puó inviare, adattivamente, quante query vuole

Esperimenti

Siano $\Pi = (Gen, Enc, Dec)$, Adv A, n parametro di sicurezza.

Indichiamo con $A^{O(\cdot)}$ un Adv (algoritmo) che ha accesso all'oracolo $O(\cdot)$.

$PrivK^{cpa}_{A,\Pi}(n)$ (Gestito da un Challenger)

1. Genera $k \leftarrow Gen(1^n)$ e *setta* l'oracolo $O(\cdot)$
2. $A^{O(\cdot)}(1^n)$ dá in output m_0 ed 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 dá in output $b' \in \{0, 1\}$
5. Se $b = b'$, l'output dell'esperimento é 1 ($A^{O(\cdot)}(1^n)$ vince); altrimenti, 0.

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 Adv A PPT, esiste una funzione trascurabile *negl* tale che:

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

dove la probabilitá é calcolata su

- randomness usata da A
- randomness usata nell'esperimento

Sicurezza CPA per cifrature multiple

Per la formalizzazione, usiamo un approccio diverso dal precedente.

Un Adv ha accesso ad un oracolo *left-or-right*, denotato con $LR_{k,b}$, che, su input m_0 ed m_1, restituisce $c \leftarrow Enc_k(m_b)$

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

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

Sicurezza CPA per cifrature multiple

Differenze con l'approccio precedente:

- Adv ottiene le cifrature di m inviando la coppia (m, m)
- le coppie $(m_{0,i}, m_{1,i})$ sono scelte *adattivamente* invece che in un sol colpo.

Sicurezza cpa per messaggi multipli

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

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

dove la probabilitá é calcolata su

- randomness usata da A
- randomness usata nell'esperimento

Ovviamente,

$$\Pi \text{ CPA-sicuro per cifrature multiple} \Rightarrow \Pi \text{ CPA-sicuro}$$

Vale anche l'inverso!

Conseguenze

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

Discende che:

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

$$\Pi = (Gen, Enc, Dec) \text{ CPA-sicuro per messaggi di 1 bit}$$

$$\Downarrow$$

$$\Pi' = (Gen', Enc', Dec') \text{ CPA-sicuro per messaggi di lunghezza arbitraria}$$

Conseguenze

Struttura dello schema per messaggi di lunghezza arbitraria

- $Gen' = Gen$
- $Enc'_k(m) = Enc_k(m_1) \dots Enc_k(m_\ell)$, dove $m = m_1 \dots m_\ell$, ed $m_i \in \{0, 1\}$
- $Dec'_k(c) = Dec_k(c_1) \dots Dec_k(c_\ell)$, dove $c = c_1 \dots c_\ell$, e $c_i \in C$ per ogni i

Osservazione:

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

Costruzione di schemi CPA-sicuri

Abbiamo bisogno di uno strumento nuovo

- le funzioni pseudocasuali (pseudorandom function, o PRF in breve)

Generalizzano la nozione di generatore pseudocasuale

- i PRG producono *stringhe che sembrano casuali*
- le PRF sono *funzioni che sembrano casuali*

Non ha senso parlare di *una* funzione fissata

Dobbiamo considerare una *distribuzione di funzioni*.

Funzioni *parametrizzate da una chiave* (keyed function) inducono naturalmente una distribuzione di funzioni

Funzioni pseudocasuali

Una funzione parametrizzata da una chiave

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

é una funzione con due input, in cui il primo é chiamato chiave e viene denotato con k.

F é efficiente se $\exists$ un algoritmo di tempo polinomiale per calcolare $F(k,x)$, dati k e x

Funzioni pseudocasuali

In usi tipici, k viene scelto e fissato. Per cui

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

Nella nostra trattazione, il parametro di sicurezza n parametrizza tre funzioni:

- $\ell_{key}(n)$ lunghezza della chiave
- $\ell_{in}(n)$ lunghezza dell'input
- $\ell_{out}(n)$ lunghezza dell'output

Funzioni pseudocasuali

Per ogni $k \in \{0,1\}^{\ell_{key}(n)}$, F_k é definita solo per $x \in \{0,1\}^{\ell_{in}(n)}$ ed ha output $y \in \{0,1\}^{\ell_{out}(n)}$.

Siano $\ell_{key}(n) = \ell_{in}(n) = \ell_{out}(n) = n$ (F *preserva la lunghezza*).

Una funzione $F(\cdot,\cdot)$ parametrizzata da una chiave induce una distribuzione di funzioni

- scegliendo una chiave uniforme $k \in \{0,1\}^n$
- considerando la funzione di una singola variabile F_k risultante

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

Funzioni pseudocasuali

Per formalizzare la nozione, abbiamo bisogno di chiarire alcuni aspetti.

Per esempio, cosa significa *scegliere una funzione a caso*?

$$\text{Sia } Func_n = \{ \text{ tutte le funzioni } f : \{0,1\}^n \rightarrow \{0,1\}^n \}$$

Una funzione puó essere rappresentata con una tabella con 2^n righe, ciascuna di n bit.

$f \in Func_n \rightarrow$

1	$f(1)$
2	$f(2)$
...	...
i	$f(i)$
...	...
2^n	$f(2^n)$

$\leftarrow$ valore della funzione sull'i-esima stringa

Funzioni pseudocasuali

Se concatenassimo tutte le righe della tabella, potremmo vedere f come una stringa di $2^n \cdot n$ bit, cioé:

$f(1)$	$\ldots$	$f(i)$	$\ldots$	$f(2^n)$

Pertanto,

$$|Func_n| = 2^{2^n \cdot n} \quad \leftarrow \text{ numero di funzioni di } n\text{-bit in } n\text{-bit}$$

Si noti che lo stesso insieme di funzioni $Func_n$ puó essere visto come una tabella.

scegliere una funzione a caso

$\approx$

scegliere una riga della tabella a caso

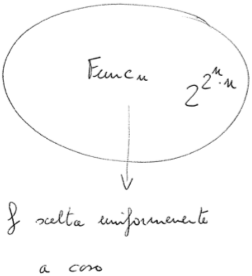
1	$f_1(\cdot)$
$\ldots$	$\ldots$
i	$f_i(\cdot)$
$\ldots$	$\ldots$
$2^{2^n \cdot n}$	$f_{2^{2^n \cdot n}}(\cdot)$

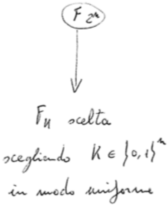
Funzioni pseudocasuali

D'altra parte, quest'ultima operazione é equivalente a vedere la riga della tabella come *una riga di elementi scelti al volo*, ogni volta che f viene valutata su un nuovo input.

Essenzialmente la riga viene riempita *volta per volta*.

Viceversa, F_k, per k uniforme, viene scelta su un insieme $\mathcal{F}$ di al piú 2^n funzioni.





Funzioni pseudocasuali

Dire che F é pseudocasuale significa che, *nonostante* la notevole differenza evidenziata, il *comportamento* di f e di F_k sembra lo stesso a qualsiasi algoritmo PPT che cerca di distinguere tra i due casi.

Come formalizzare *distinguere*?

Prima idea: dare all'algoritmo D che distingue (PPT) le descrizioni di F_k ed f.

D dovrebbe dare in output 1 all'incirca con la stessa probabilitá nei due casi.

Ma ... f ha *lunghezza esponenziale* ($2^n \cdot n$ bit) e D, che é PPT, *non* puó neanche leggere la sua descrizione!

Funzioni pseudocasuali

Seconda idea: dare a D accesso ad un *oracolo* $O(\cdot)$ che o implementa F_k, per k uniforme, o implementa f, per f uniforme.

- D puó chiedere il valore della funzione su un numero polinomiale di input x
- non chiede mai due volte il valore per lo stesso x
- al termine deve decidere se ha interagito con F_k o f

Definizione

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

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

dove la prima probabilitá é calcolata su

- scelta uniforme di k
- random bit di D

e la seconda su

- scelta uniforme di f
- random bit di D

Nota: Ovviamente D *non* riceve la chiave k!

Altrimenti risulterebbe banale per D distinguere.

Infatti, chiedendo 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 probabilitá sta interagendo con F_k. Piú valutazioni corroborerebbero l'ipotesi.

$$\Downarrow$$

Se k diventa noto, la pseudocasualitá non c'é piú!

Esempio di funzione non pseudocasuale

Esempio 3.6. Sia F una funzione con chiave che preserva la lunghezza, definita da

$$F(k, x) = k \oplus x.$$

Per ogni input x, il valore $F_k(x)$ é uniformemente distribuito quando k viene scelto in modo uniforme.

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

Infatti, D:

- chiede all'oracolo valutazioni su x_1 e x_2
- ottiene $y_1 = O(x_1)$ e $y_2 = O(x_2)$
- se $y_1 \oplus y_2 = x_1 \oplus x_2$, dá in output 1; altrimenti, dá 0.

Esempio di funzione non pseudocasuale

É facile vedere che:

- se $O \equiv F_k$, per ogni k, D dá in output 1 con probabilitá 1, poiché

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

- se $O \equiv f$, D dá in output 1 con probabilitá

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

La differenza $|1 - 1/2^n|$ ovviamente non é trascurabile

$$\Downarrow$$

F *non* é pseudocasuale!

Permutazioni pseudocasuali

Sia $Perm_n$ l'insieme di tutte le permutazioni su $\{0,1\}^n$.

Nota che

- $f \in Perm_n$ puó essere vista ancora come una tabella
- ogni due righe della tabella sono differenti
- $|Perm_n| = 2^n!$

F é una permutazione con chiave se $\ell_{in}(n) = \ell_{out}(n)$ e per ogni $k \in \{0,1\}^{\ell_{key}(n)}$ la funzione

$$F_k : \{0,1\}^{\ell_{in}(n)} \rightarrow \{0,1\}^{\ell_{out}(n)}$$

é uno a uno.

Il valore $\ell_{in}(n)$ si dice anche *lunghezza del blocco* di F.

Considereremo il caso in cui $\ell_{key}(n) = \ell_{in}(n) = \ell_{out}(n) = n$.

Permutazioni pseudocasuali

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

$$\Updownarrow$$

F efficientemente calcolabile ed invertibile, data k.

La pseudocasualitá é definita esattamente come per le funzioni.

Nota: quando la lunghezza del blocco é sufficientemente lunga, una permutazione casuale é indistinguibile da una funzione casuale.

$$\Downarrow$$

Permutazioni pseudocasuali

$$\text{Funzione uniforme} \quad \overset{\text{"appare identica"}}{\approx} \quad \text{Permutazione uniforme}$$

... a meno che il distinguisher D non trovi x ed y tali che $f(x) = f(y)$.

La probabilitá di un tale evento é trascurabile utilizzando un numero polinomiale di query.

Proposizione 3.27. Se F é una permutazione pseudocasuale e $\ell_{out}(n) \geq n$, allora F é anche una funzione pseudocasuale.

Alcune costruzioni crittografiche richiedono alle parti oneste di usare anche F_k^{-1}. Pertanto, Adv puó conoscere anche questi valori

Abbiamo bisogno di una nozione forte, che tenga in conto anche questa possibilitá dell'Adv, che possiamo modellare con un *accesso ad un oracolo* per F_k^{-1}.

Permutazioni pseudocasuali forti

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

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

dove la prima probabilitá é calcolata su

- scelta uniforme di k
- random bit di D

e la seconda su

- scelta uniforme di f
- random bit di D

Permutazioni pseudocasuali forti

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

Nella pratica i *cifrari a blocchi* vengono progettati per essere istanziazioni sicure di permutazioni pseudocasuali forti, con una lunghezza della chiave e del blocco fissate.

Funzioni pseudocasuali e generatori pseudocasuali

Un generatore pseudocasuale G da una funzione pseudocasuale F si costruisce facilmente:

$$G(s) = F_s(1)||F_s(2)||\ldots||F_s(\ell)$$

per ogni valore di ℓ desiderato.

Idea della prova: se sostituiamo F_s con $f \in Func_n$

$$G'(s) = f(1)||f(2)||\ldots||f(\ell) \qquad \text{(uniforme)}$$

$$\Downarrow$$

$$G(s) = F_s(1)||F_s(2)||\ldots||F_s(\ell) \qquad \text{(pseudocasuale),}$$

perché, se non lo fosse, esisterebbe un D PPT che distingue F_s da f.

Piú in generale, possiamo costruire uno stream cipher a partire da F.

CONSTRUCTION 3.29

Let F be a pseudorandom function. Define a stream cipher (Init, GetBits), where each call to GetBits outputs n bits, as follows:

- Init: on input $s \in \{0,1\}^n$ and $IV \in \{0,1\}^n$, set $\mathsf{st}_0 := (s, IV)$.
- GetBits: on input $\mathsf{st}_i = (s, IV)$, compute $IV' := IV + 1$ and set $y := F_s(IV')$ and $\mathsf{st}_{i+1} := (s, IV')$. Output (y, st_{i+1}).

Osservazione: se F viene implementata attraverso un cifrario a blocchi, allora la costruzione precedente usa al suo interno il cifrario a blocchi.

Anche se esistono costruzioni *dirette* per stream cipher, la costruzione precedente é raccomandata.

Generatori pseudocasuali e funzioni pseudocasuali

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

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

un generatore con fattore di espansione $2^{t(n)} \cdot n$.

Per calcolare $F_k(i)$:

- calcoliamo $G(k)$
- rappresentiamo l'output con una tabella
- prendiamo la i-esima riga tra le $2^{t(n)}$ di n bit

r_1
$\dots$
r_i
$\dots$
$r_{2^{t(n)}}$

Generatori pseudocasuali e funzioni pseudocasuali

Questa costruzione é efficiente *solo se* $t(n) = O(\log n)$.

Infatti
$$2^{t(n)} \cdot n = 2^{c \log n} \cdot n = n^c \cdot n = poly(n)$$

Nota che
$$F : \{0,1\}^n \times \{0,1\}^{c \log n} \to \{0,1\}^n$$

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

Una costruzione generale esiste ma é piú complicata. Ci torneremo nel seguito.