Crittografia

Corso di Laurea Magistrale in Informatica

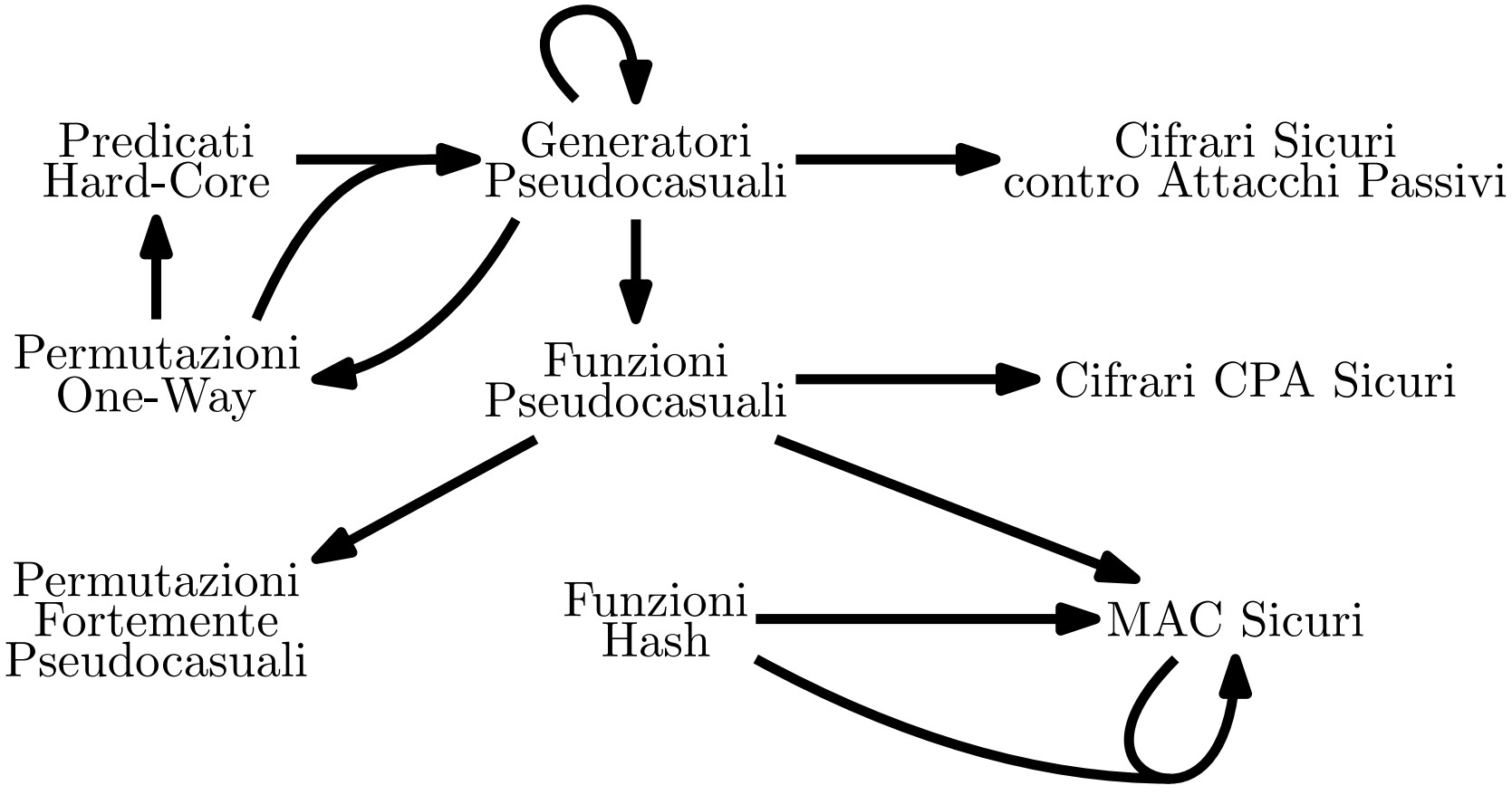
Algebra, Teoria dei Numeri e Relative Assunzioni

Ugo Dal Lago

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Anno Accademico 2018-2019

Dove Eravamo Rimasti?



Costruire Funzioni One-Way

- Abbiamo dato solo qualche esempio di funzione "presunta" one-way, senza discuterne più in dettaglio la natura.

Costruire Funzioni One-Way

- Abbiamo dato solo qualche esempio di funzione "presunta" one-way, senza discuterne più in dettaglio la natura.
- Le funzioni one-way più interessanti sono senza dubbio quelle **di natura "matematica"**.
  - In particolare, quelle che vengono dalla teoria dei numeri e dall'algebra.
- È venuto il momento di vedere un po' di cosa si tratti.
  - Faremo un ripasso veloce delle necessarie nozioni di algebra e teoria dei numeri, man mano che tali nozioni ci serviranno.
  - Un corso di crittografia pensato per studenti di matematica, tra l'altro, inizierebbe proprio qui.

Costruire Funzioni One-Way

- Abbiamo dato solo qualche esempio di funzione "presunta" one-way, senza discuterne più in dettaglio la natura.
- Le funzioni one-way più interessanti sono senza dubbio quelle **di natura "matematica"**.
  - In particolare, quelle che vengono dalla teoria dei numeri e dall'algebra.
- È venuto il momento di vedere un po' di cosa si tratti.
  - Faremo un ripasso veloce delle necessarie nozioni di algebra e teoria dei numeri, man mano che tali nozioni ci serviranno.
  - Un corso di crittografia pensato per studenti di matematica, tra l'altro, inizierebbe proprio qui.
- Le assunzioni che vedremo troveranno applicazione *diretta* nella crittografia a chiave pubblica, di cui parleremo successivamente.

Divisori, Numeri Primi, etc.

- L'insieme $\mathbb{Z}$ è l'insieme dei numeri interi, mentre $\mathbb{N}$ è l'insieme dei numeri naturali
- Dati due elementi $a, b \in \mathbb{Z}$, scriviamo $a|b$ quando esiste $c \in \mathbb{Z}$ con $ac = b$.
- Quando $a|b$ e a è positivo, diciamo che a è *divisore* di b, che quando $a \notin \{1, b\}$ è detto *fattore* di b.
- Un numero $p \in \mathbb{N}$ con $p > 1$ è detto *primo* se non ha fattori, in caso contrario *composto*.
- Dati $n, m \in \mathbb{N}$ dove $m > 1$, indichiamo con $n \mod m$ il resto della divisione di n per m.

Lemma

Se $n, m \in \mathbb{N}$ e $m > 1$, allora n è invertibile modulo m (esiste p tale che $np \mod m = 1$) ogniqualvolta $gcd(n, m) = 1$, ossia quando n, m sono primi tra loro.

Numeri Primi e Fattorizzazione

- Dato un numero naturale $N \in \mathbb{N}$, ci chiediamo se sia difficile determinare due interi $p, q \in \mathbb{N}$ tali che $N = pq$.
  - Esistono algoritmi che prendono tempo $O(\sqrt{N}(\lg(N))^k)$, ossia esponenziale nella lunghezza di N.
  - Questo è un problema che non ha nulla a che fare con la crittografia, ma che i matematici hanno studiato per secoli.

Numeri Primi e Fattorizzazione

- Dato un numero naturale $N \in \mathbb{N}$, ci chiediamo se sia difficile determinare due interi $p, q \in \mathbb{N}$ tali che $N = pq$.
  - Esistono algoritmi che prendono tempo $O(\sqrt{N}(\lg(N))^k)$, ossia esponenziale nella lunghezza di N.
  - Questo è un problema che non ha nulla a che fare con la crittografia, ma che i matematici hanno studiato per secoli.
- Possiamo quindi definire il seguente esperimento:

  $\mathsf{wFactor}_A(n)$:
  $(x, y) \leftarrow \mathbb{N} \times \mathbb{N}$ with $|x| = |y| = n$;
  $N \leftarrow x \cdot y$;
  $(z, w) \leftarrow A(N)$;
  Result: $z \cdot w = N$
- Al solito, la nostra assunzione potrebbe stare nel fatto che per ogni A PPT esiste $\varepsilon \in \mathcal{NGL}$ tale che

$$Pr(\mathsf{wFactor}_A(n) = 1) = \varepsilon(n)$$

Numeri Primi e Fattorizzazione

- L'assunzione che abbiamo appena dato, però, semplicemente *non vale*.
  - Con probabilità pari a $\frac{3}{4}$, il numero N sarà pari, perché basta che uno tra x e y sia pari perché N stesso lo sia.
  - Fattorizzare un numero pari è semplicissimo.

Numeri Primi e Fattorizzazione

- L'assunzione che abbiamo appena dato, però, semplicemente *non vale*.
  - Con probabilità pari a $\frac{3}{4}$, il numero N sarà pari, perché basta che uno tra x e y sia pari perché N stesso lo sia.
  - Fattorizzare un numero pari è semplicissimo.
- Occorre garantire che N non sia (in media) banalmente fattorizzabile.
  - Un'idea interessante è quella di modificare wFactor in modo che x e y siano sempre e solo numeri primi (rappresentabili in n bit).
  - Come facciamo a generare i numeri primi in modo casuale?
  - Che probabilità avrà un certo numero primo p di essere generato?

Gemerare Numeri Primi in Modo Efficiente

- Un modo plausibile per generare numeri primi rappresentabili in esattamente n bit è quello di andare per tentativi:

for $i \leftarrow 1$ to t **do**
  $r \leftarrow \{0,1\}^{n-1}$;
  $p \leftarrow 1||r$;
  if p *è primo* **then**
    Result: p
Result: `fail`

Gemerare Numeri Primi in Modo Efficiente

- Un modo plausibile per generare numeri primi rappresentabili in esattamente n bit è quello di andare per tentativi:

  for $i \leftarrow 1$ to t **do**
  - $r \leftarrow \{0,1\}^{n-1}$;
  - $p \leftarrow 1||r$;
  - **if** p *è primo* **then**
    - **Result**: p

  Result: `fail`

- Ci sono però due domande a cui occorre dare una risposta.
  1. **Come valorizzare il parametro t?**
     - Ci piacerebbe definire t come un polinomio in n, per ragioni di efficienza.
     - Vorremmo che per tale valore di t la probabilità di ottenere `fail` fosse trascurabile (in n).
  2. **Come testare la primalità di un numero?**
     - Vorremmo poter testare se p è primo in tempo polinomiale.

Come Valorizzare t?

Teorema

Esiste una costante c tale che per ogni $n > 1$ il numero di primi rappresentabili in esattamente n bit è almeno pari a $\frac{c \cdot 2^{n-1}}{n}$.

- Ad ogni iterata, quindi, la probabilità che p sia effettivamente primo sarà almeno pari a:

$$\frac{c \cdot 2^{n-1}/n}{2^{n-1}} = \frac{c \cdot 2^{n-1}}{n \cdot 2^{n-1}} = \frac{c}{n}$$

- Di conseguenza, se $t = \frac{n^2}{c}$, la probabilità di ottenere `fail` sarà al più pari a

$$\left(1 - \frac{c}{n}\right)^t = \left(\left(1 - \frac{c}{n}\right)^{\frac{n}{c}}\right)^n \leq (e^{-1})^n = e^{-n}$$

ogniqualvolta $n \geq c$

Come Testare la Primalità di Un Numero in Tempo Polinomiale?

- Esistono algoritmi **deterministici** per il test di primalità che prendono tempo polinomiale.
  - Il primo, il cosiddetto *algoritmo AKS*, è stato ideato da due studenti di Master dell'IIT-Kanpur, in India.
  - Il grado del polinomio è *troppo alto* perché questi algoritmi possano essere considerati interessanti da un punto di vista pratico.
- Il cosiddetto **test di Miller-Rabin** è invece probabilistico, ma è PPT.
  - Se l'input è un numero primo, il test di Miller-Rabin restituisce `prime` con probabilità 1.
  - Se l'input p è invece composto, il test di Miller-Rabin restituisce `composite` con probabilità $1 - \varepsilon(|p|)$, dove $\varepsilon \in \mathcal{NGL}$.
  - Il grado del polinomio che limita superiormente la complessità del test è relativamente basso.

L'Assunzione, Opportunamente Formalizzata

- L'assunzione che stiamo cercando di definire sarà parametrizzata su un algoritmo, chiamato GenModulus che, su input 1^n, produce in output una tripla (N, p, q) dove $N = pq$ e p, q siano primi ad n bits.
- L'esperimento wFactor diventa il seguente:

  $\mathsf{Factor}_{A,\mathsf{GenModulus}}(n)$:
  $(N, p, q) \leftarrow \mathsf{GenModulus}(1^n)$;
  $(r, s) \leftarrow A(N)$;
  Result: $r \cdot s = N$
- Diciamo che *la fattorizzazione è difficile rispetto a* GenModulus sse per ogni algoritmo A che sia PPT esiste una funzione trascurabile $\varepsilon \in \mathcal{NGL}$ tale che

$$Pr(\mathsf{Factor}_{A,\mathsf{GenModulus}}(n) = 1) = \varepsilon(n)$$

- Quest'assunzione è sufficiente a derivare una funzione one-way, ma non a dimostrare la sicurezza di schemi a chiave pubblica.

Teoria dei Gruppi

- Un **gruppo** è una struttura algebrica $(\mathbb{G}, \circ)$ dove $\circ$ è un'operazione binaria che sia associativa, con identità e e in cui ogni $g \in \mathbb{G}$ abbia un inverso g^{-1}.
- Un gruppo finito $(\mathbb{G}, \circ)$ è detto avere **ordine** pari a $|\mathbb{G}|$.
- Un gruppo si dice **abeliano** se $\circ$ è un'operazione commutativa.
- L'operazione binaria è spesso indicata:
  - Con il simbolo di addizione $+$; in tal caso il gruppo si chiama **additivo**, e se $m \in \mathbb{N}$ si può usare la notazione

$$mg = \underbrace{g + \cdots + g}_{m \text{ volte}}$$

  - Con il simbolo di moltiplicazione $\cdot$; il tal caso il gruppo si chiama **moltiplicativo**, e se $m \in \mathbb{N}$ si può scrivere

$$g^m = \underbrace{g \cdot \ldots \cdot g}_{m \text{ volte}}$$

Elevamento a Potenza

- Il calcolo di mg o di g^m può essere eseguito tramite in un numero di operazioni polinomiale in $|m|$, ossia logaritmico in m.
  - Basta procedere seguendo la rappresentazione binaria di m. Ad esempio:

    $$g^{11} = g^8 \cdot g^2 \cdot g^1 = g^{2^3} \cdot g^{2^1} \cdot g^{2^0}$$

    e ciascuno dei fattori, che sono al più $|m|$, può essere calcolato in tempo lineare in $|m|$.

Elevamento a Potenza

- Il calcolo di mg o di g^m può essere eseguito tramite in un numero di operazioni polinomiale in $|m|$, ossia logaritmico in m.
  - Basta procedere seguendo la rappresentazione binaria di m. Ad esempio:

$$g^{11} = g^8 \cdot g^2 \cdot g^1 = g^{2^3} \cdot g^{2^1} \cdot g^{2^0}$$

    e ciascuno dei fattori, che sono al più $|m|$, può essere calcolato in tempo lineare in $|m|$.

Teorema

Se $(\mathbb{G}, \cdot)$ *ha ordine* m, *allora per ogni* $g \in \mathbb{G}$, *esiste* $g^m = 1_{\mathbb{G}}$.

Corollario

Se $(\mathbb{G}, \cdot)$ *ha ordine* $m > 1$, *allora per ogni* $g \in \mathbb{G}$ *e per ogni* i, $g^i = g^{[i \mod m]}$.

Esempi di Gruppi Finiti

- L'insieme $\mathbb{Z}_N = \{0, \ldots, N-1\}$ è un gruppo se la sottostante operazione è l'addizione modulo N ovvero la mappa che a (a, b) fa corrispondere $a + b \mod N$.
  - L'identità è 0;
  - L'inverso di $n \in \mathbb{Z}_N$ è $N - n$.

Esempi di Gruppi Finiti

- L'insieme $\mathbb{Z}_N = \{0, \ldots, N-1\}$ è un gruppo se la sottostante operazione è l'addizione modulo N ovvero la mappa che a (a, b) fa corrispondere $a + b \mod N$.
  - L'identità è 0;
  - L'inverso di $n \in \mathbb{Z}_N$ è $N - n$.
- L'insieme $\mathbb{Z}_N$ diventa un gruppo con la moltiplicazione modulo N quando:
  - Eliminiamo 0 (che non è invertibile) dal gruppo.
  - N è primo. Ciò garantisce che ogni $1 < n < N$ sia invertibile modulo N.

Esempi di Gruppi Finiti

- L'insieme $\mathbb{Z}_N = \{0, \ldots, N-1\}$ è un gruppo se la sottostante operazione è l'addizione modulo N ovvero la mappa che a (a, b) fa corrispondere $a + b \mod N$.
  - L'identità è 0;
  - L'inverso di $n \in \mathbb{Z}_N$ è $N - n$.
- L'insieme $\mathbb{Z}_N$ diventa un gruppo con la moltiplicazione modulo N quando:
  - Eliminiamo 0 (che non è invertibile) dal gruppo.
  - N è primo. Ciò garantisce che ogni $1 < n < N$ sia invertibile modulo N.
- Se consideriamo $\mathbb{Z}_N$ con N composto, c'è un modo per rendere questo insieme un gruppo rispetto alla moltiplicazione modulo N:
  - Basta considerare $\mathbb{Z}_N^* \subseteq \mathbb{Z}_N$ definito come $\{n \in \mathbb{Z}_N \mid gcd(n, N) = 1\}$.

Esempi di Gruppi Finiti

- L'insieme $\mathbb{Z}_N = \{0, \ldots, N-1\}$ è un gruppo se la sottostante operazione è l'addizione modulo N ovvero la mappa che a (a, b) fa corrispondere $a + b \mod N$.
  - L'identità è 0;
  - L'inverso di $n \in \mathbb{Z}_N$ è $N - n$.
- L'insieme $\mathbb{Z}_N$ diventa un gruppo con la moltiplicazione modulo N quando:
  - Eliminiamo 0 (che non è invertibile) dal gruppo.
  - N è primo. Ciò garantisce che ogni $1 < n < N$ sia invertibile modulo N.
- Se consideriamo $\mathbb{Z}_N$ con N composto, c'è un modo per rendere questo insieme un gruppo rispetto alla moltiplicazione modulo N:
  - Basta considerare $\mathbb{Z}_N^* \subseteq \mathbb{Z}_N$ definito come $\{n \in \mathbb{Z}_N \mid gcd(n, N) = 1\}$.
- Tutti i gruppi considerati sono *abeliani*.

Sulla cardinalità di $\mathbb{Z}_N^*$

- La funzione che ad ogni naturale $N > 1$ associa la cardinalità di $\mathbb{Z}_N^*$ è detta *funzione di Eulero* ed è indicata con Φ:

$$\Phi(N) = |\mathbb{Z}_N^*|$$

- Chiaramente $1 \leq \Phi(N) < N$. Ma possiamo dire qualcosa di più?
  - Se N è un numero primo p, allora $\Phi(N) = p - 1$ perché ogni n compreso tra 1 e $p-1$ è primo con p, ovvero $gcd(n, p) = 1$.
  - Se N è il prodotto di due primi p e q, allora $gcd(a, N) \neq 1$ precisamente quando $p|a$ oppure $q|a$. Quindi:

$$\begin{aligned}\Phi(N) &= N - 1 - (p-1) - (q-1) = pq - p - q + 1 \\ &= p(q-1) - 1(q-1) = (p-1)(q-1)\end{aligned}$$

Classi Resto ed Elevamento a Potenza

Teorema

Sia $N > 1$. *Per ogni naturale* $e > 0$, *definiamo* $f_e : \mathbb{Z}_N^* \to \mathbb{Z}_N^*$ *ponendo* $f_e(x) = x^e \mod N$. *Se* $gcd(e, \Phi(N)) = 1$, *allora* f_e *è una permutazione. Inoltre se* d *è l'inverso di* e *(modulo* $\Phi(N)$*), allora* f_d *è l'inversa di* f_e.

Classi Resto ed Elevamento a Potenza

Teorema

Sia $N > 1$. Per ogni naturale $e > 0$, definiamo $f_e : \mathbb{Z}_N^* \to \mathbb{Z}_N^*$ ponendo $f_e(x) = x^e \mod N$. Se $gcd(e, \Phi(N)) = 1$, allora f_e è una permutazione. Inoltre se d è l'inverso di e (modulo $\Phi(N)$), allora f_d è l'inversa di f_e.

- Osserviamo come dati e ed N, il valore di $f_e(x)$ sia efficientemente calcolabile a partire a partire da x.
  - Basta applicare l'algoritmo di esponenziazione.

Classi Resto ed Elevamento a Potenza

Teorema

Sia $N > 1$. *Per ogni naturale* $e > 0$, *definiamo* $f_e : \mathbb{Z}_N^* \to \mathbb{Z}_N^*$ *ponendo* $f_e(x) = x^e \mod N$. *Se* $gcd(e, \Phi(N)) = 1$, *allora* f_e *è una permutazione. Inoltre se* d *è l'inverso di* e *(modulo* $\Phi(N)$*), allora* f_d *è l'inversa di* f_e.

- Osserviamo come dati e ed N, il valore di $f_e(x)$ sia efficientemente calcolabile a partire a partire da x.
  - Basta applicare l'algoritmo di esponenziazione.
- Osserviamo anche dati e e $\Phi(N)$, l'inverso d di e moduleo $\Phi(N)$ sia efficientemente calcolabile.
  - L'inversione modulo un qualunque intero è un problema trattabile.

Classi Resto ed Elevamento a Potenza

Teorema

Sia $N > 1$. Per ogni naturale $e > 0$, definiamo $f_e : \mathbb{Z}_N^* \to \mathbb{Z}_N^*$ ponendo $f_e(x) = x^e \mod N$. Se $gcd(e, \Phi(N)) = 1$, allora f_e è una permutazione. Inoltre se d è l'inverso di e (modulo $\Phi(N)$), allora f_d è l'inversa di f_e.

- Osserviamo come dati e ed N, il valore di $f_e(x)$ sia efficientemente calcolabile a partire a partire da x.
  - Basta applicare l'algoritmo di esponenziazione.
- Osserviamo anche dati e e $\Phi(N)$, l'inverso d di e moduleo $\Phi(N)$ sia efficientemente calcolabile.
  - L'inversione modulo un qualunque intero è un problema trattabile.
- Infine, osserviamo come dato N prodotto di due primi p e q il valore di $\Phi(N)$ *non sia* invece facilmente calcolabile.
  - Lo sarebbe se si potesse (efficientemente) fattorizzare N.

L'Assunzione RSA

- Nella cosiddetta Assunzione RSA quello che facciamo è predicare la *difficoltà di invertire* $f_e : \mathbb{Z}_N^* \to \mathbb{Z}_N^*$ quando N è il prodotto di due primi p e q e $gcd(e, \Phi(N)) = 1$.
- A differenza dell'assunzione sulla fattorizzazione, però, il problema che stiamo assumendo difficile *diventa facile* se
  - oltre a N si conoscono anche p e q;
  - oppure se si conosce un inverso d di e (modulo $\Phi(N)$).
- L'assunzione RSA è parametrizzata su una routine GenRSA che, dato un input 1^n ritorna:
  - Un naturale N che sia prodotto di due primi p e q con $|p| = |q| = n$.
  - Un naturale e tale che $gcd(e, \Phi(N)) = 1$.
  - Un naturale d tale che $ed \mod \Phi(N) = 1$.

L'Assunzione RSA

- L'esperimento RSAInv è parametrizzato su un avversario e sulla routine GenRSA:

  $\mathsf{RSAInv}_{A,\mathsf{GenRSA}}(n)$:
  $(N, e, d) \leftarrow \mathsf{GenRSA}(1^n)$;
  $y \leftarrow \mathbb{Z}_N^*$;
  $x \leftarrow A(N, e, y)$;
  Result: $x^e \mod N = y$

L'Assunzione RSA

- L'esperimento RSAInv è parametrizzato su un avversario e sulla routine GenRSA:

  $\mathsf{RSAInv}_{A,\mathsf{GenRSA}}(n)$:
  $(N, e, d) \leftarrow \mathsf{GenRSA}(1^n)$;
  $y \leftarrow \mathbb{Z}_N^*$;
  $x \leftarrow A(N, e, y)$;
  Result: $x^e \mod N = y$

- Diciamo che *il problema RSA è difficile rispetto a* GenRSA sse per ogni avversario A che sia PPT esiste $\varepsilon \in \mathcal{NGL}$ tale che

$$Pr(\mathsf{RSAInv}_{A,\mathsf{GenRSA}}(n) = 1) = \varepsilon(n)$$

L'Assunzione RSA

- L'esperimento RSAInv è parametrizzato su un avversario e sulla routine GenRSA:

  $\mathsf{RSAInv}_{A,\mathsf{GenRSA}}(n)$:
  $(N, e, d) \leftarrow \mathsf{GenRSA}(1^n)$;
  $y \leftarrow \mathbb{Z}_N^*$;
  $x \leftarrow A(N, e, y)$;
  Result: $x^e \mod N = y$

- Diciamo che *il problema RSA è difficile rispetto a* GenRSA sse per ogni avversario A che sia PPT esiste $\varepsilon \in \mathcal{NGL}$ tale che

$$Pr(\mathsf{RSAInv}_{A,\mathsf{GenRSA}}(n) = 1) = \varepsilon(n)$$

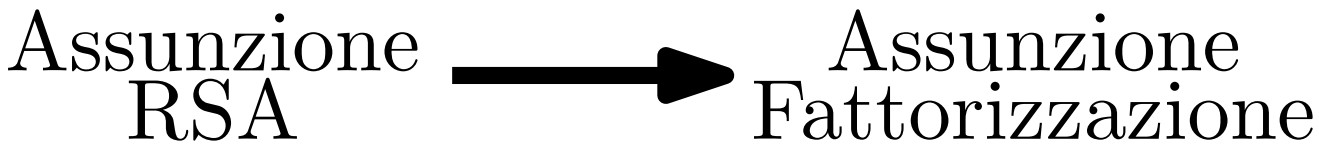
- Come abbiamo già avuto modo di argomentare, se A riuscisse a fattorizzare N, riuscirebbe a calcolare $\Phi(N)$ e quindi a calcolare d, invertendo di conseguenza f_e.

Come Costruire GenRSA?

- Come possiamo costruire l'algoritmo GenRSA in modo che sia efficientemente calcolabile?
- Certamente, GenModulus è d'aiuto, ma ovviamente non basta.
- Potremmo per esempio procedere come segue:

$$
\begin{aligned}
&(N, p, q) \leftarrow \mathsf{GenModulus}(1^n);\\
&M \leftarrow (p-1)(q-1);\\
&e \leftarrow \{1, \ldots, M\} \text{ tale che } gcd(e, M) = 1;\\
&d \leftarrow e^{-1} \mod M;\\
&\textbf{Result}: (N, e, d)
\end{aligned}
$$

- Se GenRSA è costruita in questo modo a partire da GenModulus, è possibile dimostrare che dall'Assunzione RSA *segue* l'Assunzione sulla Fattorizzazione dei Numeri Primi.


Assunzione
RSA
Assunzione
Fattorizzazione

Gruppi Ciclici

- Consideriamo un gruppo moltiplicativo finito $(\mathbb{G}, \cdot)$, un suo elemento $g \in \mathbb{G}$ e costruiamo

$$\langle g \rangle = \{g^0, g^1, g^2, \ldots\} \subseteq \mathbb{G}$$

Gruppi Ciclici

- Consideriamo un gruppo moltiplicativo finito $(\mathbb{G}, \cdot)$, un suo elemento $g \in \mathbb{G}$ e costruiamo

$$\langle g \rangle = \{g^0, g^1, g^2, \ldots\} \subseteq \mathbb{G}$$

- Sappiamo che $g^m = 1_{\mathbb{G}}$, per cui possiamo certo scrivere che $\langle g \rangle = \{g^1, \ldots, g^m\}$.

Gruppi Ciclici

- Consideriamo un gruppo moltiplicativo finito $(\mathbb{G}, \cdot)$, un suo elemento $g \in \mathbb{G}$ e costruiamo

$$\langle g \rangle = \{g^0, g^1, g^2, \ldots\} \subseteq \mathbb{G}$$

- Sappiamo che $g^m = 1_{\mathbb{G}}$, per cui possiamo certo scrivere che $\langle g \rangle = \{g^1, \ldots, g^m\}$.
- Ci potrebbe però essere un $i < m$ tale che $g^i = 1_{\mathbb{G}}$. Per ovvie ragioni $\langle g \rangle = \{g^1, \ldots, g^i\}$, e quindi $\langle g \rangle$ contiene al più i elementi.
  - In realtà ne ha *esattamente* i; se $1 \leq k < j < i$, allora

$$g^j = g^k \implies g^j \cdot g^{-k} = 1_{\mathbb{G}} \implies g^{j-k} = 1_{\mathbb{G}}.$$

Gruppi Ciclici

- Consideriamo un gruppo moltiplicativo finito $(\mathbb{G}, \cdot)$, un suo elemento $g \in \mathbb{G}$ e costruiamo

$$\langle g \rangle = \{g^0, g^1, g^2, \ldots\} \subseteq \mathbb{G}$$

- Sappiamo che $g^m = 1_{\mathbb{G}}$, per cui possiamo certo scrivere che $\langle g \rangle = \{g^1, \ldots, g^m\}$.
- Ci potrebbe però essere un $i < m$ tale che $g^i = 1_{\mathbb{G}}$. Per ovvie ragioni $\langle g \rangle = \{g^1, \ldots, g^i\}$, e quindi $\langle g \rangle$ contiene al più i elementi.
  - In realtà ne ha *esattamente* i; se $1 \leq k < j < i$, allora

$$g^j = g^k \implies g^j \cdot g^{-k} = 1_{\mathbb{G}} \implies g^{j-k} = 1_{\mathbb{G}}.$$

- L'**ordine** di $g \in \mathbb{G}$ è il più piccolo naturale i tale che $g^i = 1$, ossia la cardinalità di $\langle g \rangle$.
- Un gruppo finito $(\mathbb{G}, \cdot)$ si dice **ciclico** se esiste $g \in \mathbb{G}$ con $\langle g \rangle = \mathbb{G}$. Un tale g si dice **generatore** di $\mathbb{G}$.

Gruppi Ciclici e Ordine

Lemma

Se $\mathbb{G}$ *ha ordine* m *e* $g \in \mathbb{G}$ *ha ordine* i*, allora* $i|m$

Gruppi Ciclici e Ordine

Lemma

Se $\mathbb{G}$ ha ordine m e $g \in \mathbb{G}$ ha ordine i, allora $i|m$

- Se così non fosse, avremmo che $m = ik + j$ con $j < i$. Ma quindi

$$g^j = g^{j+ik-ik} = g^{m-ik} = g^m(g^i)^{-k} = 1_{\mathbb{G}}(1_{\mathbb{G}})^{-k} = 1_{\mathbb{G}}$$

Gruppi Ciclici e Ordine

Lemma

Se $\mathbb{G}$ ha ordine m e $g \in \mathbb{G}$ ha ordine i, allora $i|m$

- Se così non fosse, avremmo che $m = ik + j$ con $j < i$. Ma quindi

$$g^j = g^{j+ik-ik} = g^{m-ik} = g^m(g^i)^{-k} = 1_{\mathbb{G}}(1_{\mathbb{G}})^{-k} = 1_{\mathbb{G}}$$

Teorema

Se $\mathbb{G}$ ha ordine primo allora $\mathbb{G}$ è ciclico e ogni $g \in \mathbb{G}$ con $g \neq 1_{\mathbb{G}}$ genera $\mathbb{G}$.

Assunzione Del Logaritmo Discreto

- Se $\mathbb{G}$ è un gruppo moltiplicativo ciclico, allora esiste una corrispondenza biunivoca "naturale" tra $\mathbb{G}$ e $\mathbb{Z}_{|\mathbb{G}|}$.
  - Ad ogni $h \in \mathbb{G}$ si può mettere in corrispondenza *un unico* $x \in \mathbb{Z}_{|\mathbb{G}|}$, ossia quello tale per cui $g^x = h$. Chiamiamo tale x il **logaritmo discreto** di h rispetto a g, che scriviamo $\log_g h$.
- Il *problema del logaritmo discreto* è semplicemente il problema di calcolare $\log_g h$ dati un gruppo ciclico $\mathbb{G}$, un generatore g per $\mathbb{G}$ e un elemento casuale h.
- L'esperimento con cui formalizzeremo l'assunzione del logaritmo discreto è parametrizzato, al solito, da una routine GenCG che dato 1^n costruisce un gruppo $\mathbb{G}$, di ordine q con $|q| = n$ e un generatore $g \in \mathbb{G}$.

Assunzione Del Logaritmo Discreto

- L'esperimento DLog è definito come segue:

  $\mathsf{DLog}_{A,\mathsf{GenCG}}(n)$:
  $(\mathbb{G}, q, g) \leftarrow \mathsf{GenCG}(1^n)$;
  $h \leftarrow \mathbb{G}$;
  $x \leftarrow A(\mathbb{G}, q, g, h)$;
  Result: $g^x = h$

- Al solito, diciamo che l'*assunzione del logaritmo discreto vale rispetto a* GenCG sse per ogni avversario A PPT esiste $\varepsilon \in \mathcal{NGL}$ tale che

$$Pr(\mathsf{DLog}_{A,\mathsf{GenCG}}(n) = 1) = \varepsilon(n)$$

Assunzione Computazionale di Diffie-Helmann

- Dato un gruppo ciclico $\mathbb{G}$ e un generatore $g \in \mathbb{G}$ per esso, definiamo la funzione $DH_g : \mathbb{G} \times \mathbb{G} \to \mathbb{G}$ come segue:

$$DH_g(h, j) = g^{(\log_g h) \cdot (\log_g j)}$$

- Osserviamo come:

$$DH_g(g^x, g^y) = g^{x \cdot y} = (g^x)^y = (g^y)^x.$$

- Il problema CDH consiste nel calcolare efficientemente DH_g dati un gruppo $\mathbb{G}$ e un generatore g per esso (prodotti tramite GenCG.
- L'assunzione CDH (rispetto a GenCG) vale quando il problema CDH è difficile (rispetto allo stesso GenCG).
- Ogni algoritmo efficiente per il logaritmo discreto induce un algoritmo efficiente per CDH.
  - Basta calcolare i logaritmi di h e j, moltiplicare i risultati ottenuti ed elevare g per il prodotto.

Assunzione Decisionale di Diffie-Helmann

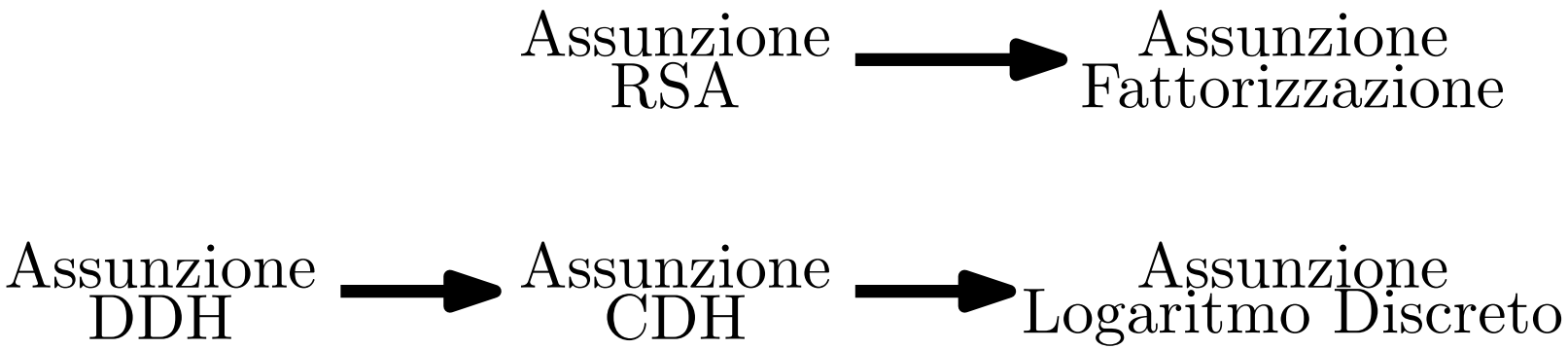
- Informalmente, il problema DDH consiste nel distinguere $DH_g(h, j)$ da un arbitrario elemento del gruppo $\mathbb{G}$, dati ovviamente h e j.

Assunzione Decisionale di Diffie-Helmann

- Informalmente, il problema DDH consiste nel distinguere $DH_g(h, j)$ da un arbitrario elemento del gruppo $\mathbb{G}$, dati ovviamente h e j.
- Formalmente, diciamo che il problema DDH è difficile (o che l'assunzione DDH è valida rispetto a GenCG) sse per ogni A PPT esiste ε trascurabile tale che

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

  dove $(\mathbb{G}, q, g)$ è il risultato di $\mathsf{GenCG}(1^n)$ e $x, y, z \in \mathbb{Z}_q$ sono casuali.
- Ogni algoritmo efficiente per CDH induce trivialmente un algoritmo efficiente per DDH.
  - Basta che A calcoli $DH_g(h, j)$ dove h e j sono il quarto e quinto parametro. Il risultato va poi confrontato con il sesto parametro.


Assunzione RSA
Assunzione Fattorizzazione
Assunzione DDH
Assunzione CDH
Assunzione Logaritmo Discreto

Assunzioni di DH su Gruppi Specifici

- Prima di tutto è da osservare come l'utilizzo di gruppi con un numero primo di elementi è preferibile, questo perché:
  - Perché testare se un elemento è o meno un generatore è triviale.
  - Perché è possibile dimostrare che se $\mathbb{G}$ è un gruppo di ordine primo q con $|q| = \Theta(2^n)$ allora

$$Pr(DH_g(h, j) = y) = \frac{1}{q} + \varepsilon(n)$$

- Prendiamo poi in considerazione i gruppi $\mathbb{Z}_p^*$ dove p è un numero *primo*.
  - Un algoritmo GenCG che generi gruppi di questo tipo esiste ed è efficiente. L'assunzione del logaritmo discreto vale per questo gruppo.
  - DDH *non* si crede essere difficile per questi gruppi.
  - Esiste però un diverso algoritmo GenCG' che restitutisce un sottogruppo di $\mathbb{Z}_p^*$ e per il quale anche DDH si crede essere difficile.

Dalla Fattorizzazione ad alle Funzioni One-Way

- Sia data una funzione $\mathsf{GenModulus}$ che utilizzi al più $p(n)$, dove p è polinomio, bit casuali su input di lunghezza n.
- Costruiamo un algoritmo che calcoli una funzione $f_{\mathsf{GenModulus}}$ come segue:
  - L'input è una stringa x;
  - Calcoliamo prima di tutto un intero n tale che $p(n) \leq |x| \leq p(n+1)$;
  - Calcola (N, p, q) come il risultato di $\mathsf{GenModulus}(1^n)$ usando come bit casuali quelli in x, che sono sufficienti.
  - Restituisci N.

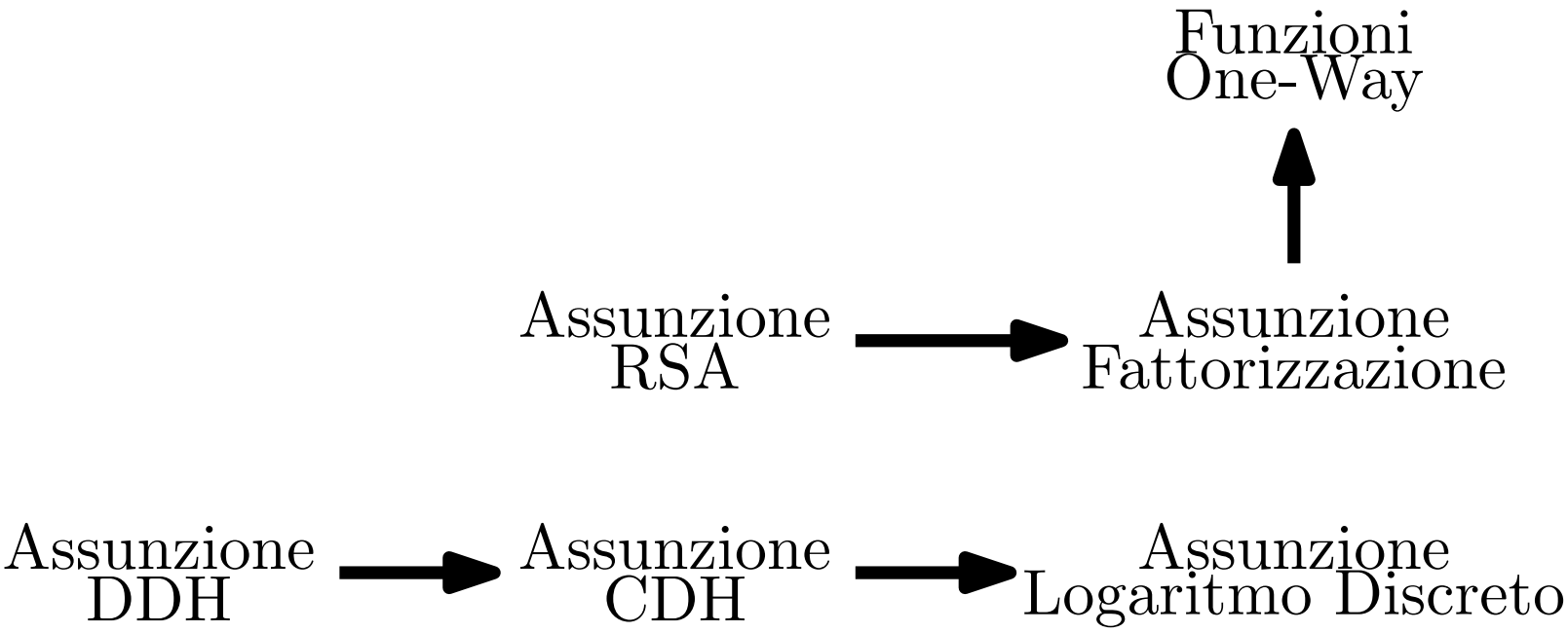
Dalla Fattorizzazione ad alle Funzioni One-Way

- Sia data una funzione $\mathsf{GenModulus}$ che utilizzi al più $p(n)$, dove p è polinomio, bit casuali su input di lunghezza n.
- Costruiamo un algoritmo che calcoli una funzione $f_{\mathsf{GenModulus}}$ come segue:
  - L'input è una stringa x;
  - Calcoliamo prima di tutto un intero n tale che $p(n) \leq |x| \leq p(n+1)$;
  - Calcola (N, p, q) come il risultato di $\mathsf{GenModulus}(1^n)$ usando come bit casuali quelli in x, che sono sufficienti.
  - Restituisci N.
- Osserviamo a questo punto come le seguenti distribuzioni sono identiche per ogni $m \in \mathbb{N}$
  - Il risultato N di $f_{\mathsf{GenModulus}}(x)$ dove $x \in \{0,1\}^m$ è scelta in modo casuale;
  - Il risultato N di $\mathsf{GenModulus}(1^n)$ dove $p(n) \leq m \leq p(n+1)$.

Dalla Fattorizzazione ad alle Funzioni One-Way

- Sia data una funzione GenModulus che utilizzi al più $p(n)$, dove p è polinomio, bit casuali su input di lunghezza n.
- Costruiamo un algoritmo che calcoli una funzione $f_{\mathsf{GenModulus}}$ come segue:
  - L'input è una stringa x;
  - Calcoliamo prima di tutto un intero n tale che $p(n) \leq |x| \leq p(n+1)$;
  - Calcola (N, p, q) come il risultato di $\mathsf{GenModulus}(1^n)$ usando come bit casuali quelli in x, che sono sufficienti.
  - Restituisci N.
- Osserviamo a questo punto come le seguenti distribuzioni sono identiche per ogni $m \in \mathbb{N}$
  - Il risultato N di $f_{\mathsf{GenModulus}}(x)$ dove $x \in \{0,1\}^m$ è scelta in modo casuale;
  - Il risultato N di $\mathsf{GenModulus}(1^n)$ dove $p(n) \leq m \leq p(n+1)$.

Teorema

Se la fattorizzazione è difficile rispetto a GenModulus, *allora* $f_{\mathsf{GenModulus}}$ *è una funzione one-way.*

Riassumendo