Distillato di Crittografia

Michele Sponsale

Luglio 2019

1 Casualità e Complessità di Kolmogorov

Sia S_i l'i-esimo *sistema di calcolo* in una certa enumerazione dei sistemi di calcolo[1]. Se p è un programma (stringa) di lunghezza $|p|$ per calcolare il numero h in S_i, indicato come $S_i(p) = h$, allora la **Complessita di Kolmogorov *relativa*** a S_i di h $K_{S_i}(h)$ è

$$K_{S_i}(h) = \min\{|x| : S_i(p) = h\}$$

Facoltativo: Esistono dei sistemi di calcolo universali S_U tali che possano emulare un programma p per il calcolo di h di qualunque S_i dato il programma $< p, i >$ (concatenazione di stringhe) di lunghezza $|p| + \log i$,[2] ossia $S_U(p, i) = h$.

La complessità di Kolmogorov su macchine universali o semplicemente **Complessità di Kolmogorov** di un numero h rispetta, per ogni i,

$$K(p) = K_{S_U}(p) \leq K_{S_i}(p) + c_i$$

dove c_i è una costante che dipende solo da i. Riassumendo meno formalmente

> La complessità di Kolmogorov di un numero è la lunghezza del più corto programma (senza input...) che lo calcola.

1.1 Numero casuale

Un numero h è detto (*algoritmicamente*) **casuale** se

$$K(h) \geq |h|$$

Statisticamente le proprietà di un numero che verifica la precedente sono le stesse di un numero che verifichi la seguente **definizione di numero *casuale***

$$K(h) \geq |h| - \lceil \log_2 |h| \rceil$$

[1] Tralasciamo la dimostrazione della numerabilità dei sistemi di calcolo Turing-equivalenti.
[2] La lunghezza di un numero naturale rappresentato in base B è $\log_B i$.

1.2 Cardinalità insieme numeri casuali - quanti ce ne sono

Fra i numeri di lunghezza (per esempio in bit) minore o uguale a n, la quantità dei numeri casuali è esponenziale alla lunghezza ed esistono di ogni lunghezza, come è possibile dimostrare con considerazioni sulla complessità di Kolmogorov. Dunque, fino a un numero fissato, sono esponenzialmente più frequenti i numeri casuali che quelli non casuali.

Stabilire se un numero sia casuale o meno (nel senso di Kolmogorov) è indecidibile, tuttavia possiamo testare le proprietà statistiche di un numero affinché siano simili a quelle di un numero casuale.

1.3 Test statistici di (pseudo)casualità

Superare un test statistico di casualità è condizione necessaria (ma non sufficiente) perché un numero sia casuale (dal paragrafo precedente non possono esistere classicamente condizioni sufficienti).

Test del prossimo bit Un numero supera il *test del prossimo bit* se il bit $k+1$ del numero non può essere previsto con probabilità maggiore di $\frac{1}{2}$ conoscendo i suoi precedenti k bit. Questo è il test più forte.

Consigliato ma non necessario

Questo test implica almeno altri 4 test statistici (nel senso che se n supera il test del prossimo bit supera anche questi):

Test di frequenza Ogni elemento della sequenza che rappresenta il numero compare quanto ogni altro.

Poker test Data una lunghezza fissata k, un numero passa il *poker test* se tutte le sottosequenze di lunghezza k (i *poker*) appaiono con la stessa frequenza.

Test di autocorrelazione Verifica il numero di elementi uguali ripetuti a distanze prefissate (ogni 5 elementi, ogni 3, etc.)

Run test Verifica se sottosequenze massimali di elementi identici[3] seguano una distribuzione esponenziale negativa nella lunghezza.

2 Generatori di numeri pseudo-casuali

Un **generatore di numeri pseudo-casuali** è un algoritmo o macchina fisica che genera numeri che superano alcuni test statistici di casualità. Nella pratica è utilizzato per generare una sequenza più o meno lunga di numeri casuali a partire da un singolo numero casuale ("amplificano" la casualità). Il *periodo* di

[3] E.g., in 111019011111111 le sottosequenze massimali sono $111, 0, 1, 9$ e 11111111 ma non 11111.

un generatore è il minimo valore della variabili indipendente, chiamata in questo contesto *seme*, dopo il quale l'immagine si ripete.

Esempi di generatori di numeri pseudo-casuali sono i **generatori polinomiali congruenti** $P_n(x) \mod m$ dove $P_n(x)$ è un polinomio di grado n, in cui tuttavia i coefficienti di $P_n(x)$ sono soggetti a vincoli affinché il generatore funzioni[4], oppure il **generatore BBS** (*Blum-Blum-Shub* dai nomi dei progettisti).

3 Predicati hard-core e generatore BBS

3.1 Accenno su funzioni one-way

Una funzione f è detta one-way se

$$f \in PTIME \wedge f^{-1} \in EXPTIME$$

(la funzione è calcolabile da algoritmi polinomiali ma la sua inversa solo da algoritmi esponenziali). Molto spesso a fini pratici per funzioni one-way si rilassa la condizione a $f \in P$ e $f^{-1} \in NPH$ o $f^{-1} \in NP$, e addirittura spesso si considerano per comodità pratica funzioni tali che si conoscano solo algoritmi NP, ma la cui appartenenza alla classe non è dimostrata (quindi possono essere scoperti algoritmi più efficienti). Riassumendo informalmente

> Le funzioni one-way sono funzioni facili da calcolare ma computazionalmente difficili da invertire.

3.2 Predicati hard-core

Un predicato $b(x)$ (cioé una funzione che restituisce 0 o 1) è detto **hard-core** per una funzione f se soddisfa la seguente definizione

> Un predicato b per una funzione f è detto **hard-core** *sse* $b(x) \in P$ e prevedere il valore di $\widetilde{b}(f(x)) = b(f^{-1}(f(x)))$ con una probabilità maggiore di $\frac{1}{2}$ è un problema NP.

Dunque se un predicato è facile da calcolare conoscendo x ma è difficile da prevedere conoscendo $f(x)$, allora è *hard-core* per quella funzione. In generale questi predicati esistono tipicamente per funzioni *one-way*, perché ereditano la difficoltà di previsione dalla difficoltà di dover invertire f per calcolarli.

Un tipico modo di utilizzare predicati *hard-core* è calcolare iterativamente da 0 a N una sequenza di coppie $f^i(x), b(f^i(x))$ (f applicata a se stessa i-volte), che è facile da calcolare per definizione, e successivamente restituire come valori le coppie nell'ordine inverso da $N, N-1, \ldots$ a 0. Restituendo i valori in ordine inverso una sequenza di bit generata in questo modo supera per definizione il **test del prossimo bit**.

[4] Questi vincoli sulle funzioni modulari ricompaiono per esempio nel *test di primalità di Miller-Rabin*.

3.3 Generatore BBS

Prendo due numeri primi grandi p e q, che soddisfino **entrambi**

1. $j \mod 4 = 3$
2. $2\lfloor \frac{j}{4} \rfloor + 1$ è primo

Allora, preso un numero y a caso (*seed* del **BBS**) uso la funzione one-way (perché invertirla è difficile)

$$x_0 \leftarrow y^2 \mod pq$$

$$x_{i+1} \leftarrow x_i^2 \mod (pq)$$

e ad ogni passo dell'algoritmo iterativo calcolo il predicato predicato hard-core

$$b(x_i) = x_{i-1} \mod 2$$

(che fa 0 se x_{i-1} calcolato al passo precedente è pari e 1 se dispari). Alla fine restituisco i valori del predicato di parità in ordine inverso, generando dunque un numero che supera il *test del prossimo bit* poiché il predicato è hard-core.

4 Test di primalità di Miller-Rabin

Il test di Miller-Rabin determina se un numero è primo con una probabilità di errore piccola a piacere scelta al momento dell'esecuzione (algoritmo Monte Carlo). Si basa sulle proprietà statistiche dei numeri primi conseguenza del Lemma di Miller-Rabin, ed è un metodo iterativo.

4.1 Lemma di Miller-Rabin

Se N è pari allora è sicuramente composto (non è un numero primo), altrimenti se è **dispari**

$$N - 1 \text{ è pari} \rightarrow N - 1 = 2^w z \text{ con z dispari}$$

poiché ovviamente qualsiasi numero pari si può fattorizzare come una potenza di 2 per un numero dispari.

Proprietà Dato un qualunque $y | 2 \leq y \leq N - 1$ allora se N è primo valgono le due proprietà (condizioni necessarie):

P1. $MCD(N, y) = 1$

P2. $y^z \mod N = 1 \vee \exists i (0 \leq i \geq w - 1) : y^{2^i z} \mod N = -1$

4.1.1 Lemma

Se N è composto il numero di interi in $[2, N - 1]$ che soddisfano entrambe le proprietà è minore di $N/4$.

4.2 Il test

Se per un y **SCELTO A CASO**[5] è falsificata P1 o P2 allora N è composto, ma anche se non si trovasse un y non potremo concludere la primalità di N, ma solo che la probabilità che N sia composto è minore di 1/4 (equivalentemente la probabilità che sia primo è maggiore di 3/4). In questo caso chiameremo y *certificato* (*probabilistico*) della primalità di N.

Definiamo la subroutine di utilità, dato un numero N (dispari) e un (papabile) certificato di primalità y

Verifica(N,y):
if not P1 or **not** P2:
return composto
else return indefinito

che ci permette di descrivere l'algoritmo del Test di Miller-Rabin

```
test_Miller-Rabin(N,k):
    for i<- 1 to k:
        y<-rand(2,N)
        if Verifica(N,y):
            return sicuramente-composto


    return forse-primo}
```

La probabilità che sia primo è esattamente $1 - \frac{1}{2^k}$[6]

4.3 Applicazione: generare numeri primi grandi

Un problema comune nelle applicazioni crittografiche è generare numeri primi grandi (per algoritmi che usano funzioni one-way basate sul problema della fattorizzazione). Sfruttando il test di Miller-Rabin possiamo definire un generatore di numeri primi.

La quantità di numeri primi minori di N tende asintoticamente a $N/\log N$, quindi in un intorno di raggio $\log N$ di N c'è mediamente un numero primo. Definiamo l'algoritmo per generare un numero primo di lunghezza n bit

genera_Primo(n,S):

$N \leftarrow 1S1$[7]

while test_Miller-Rabin(N,$\log N$) == composto:
$N \leftarrow N + 2$

return N

[5] Scegliendo non a caso il ragionamento non funziona.

[6] 1 meno la probabilità congiunta che sia composto pur avendo superato il test k-volte, che è il prodotto delle probabilità semplici se si scelgono y in modo casuale e indipendente.

[7] $1S1$ è la concatenazione di stringhe binarie.

5 Cifrari perfetti

Indicando con $m \in$**Msg** un messaggio nell'insieme messaggi possibili, con $c \in$**Critto** un crittogramma nell'insieme dei crittogrammi possibili, se $\mathcal{P}(M = m)$ è la probabilitò che il messaggio spedito sia m e $\mathcal{P}(M = m|C = c)$ la probabilità che il messaggio spedito sia m condizionata dal fatto che il crittogramma spedito C è stato rilevato essere c, allora la definizione di Shannon di **cifrario perfetto** è

> Un cifrario è **perfetto** *sse* per ogni messaggio $m \in$**Msg** e per ogni crittogramma $c \in$**Critto** vale
> $$\mathcal{P}(M = m \,|\, C = c) = \mathcal{P}(M = m)$$

dunque la probabilità che un attaccante possa individuare il messaggio non cambia dopo che questi ha intercetta il suo crittogramma.

5.1 Teorema sul numero delle chiavi in un cifrario pefretto

Il numero delle chiavi in un cifrario perfetto deve essere maggiore o uguale al numero dei messaggi possibili.

Il teorema si basa sul fatto che se ci fossero meno chiavi che messaggi dato un certo crittogramma, potremmo escludere alcuni messaggi non ottenibili decrittando quel crittogramma[8]. Così alteriamo la probabilità condizionata Sdel messaggio, e il cifrario non sarebbe perfetto (in particolare la probabilità condizionata sarebbe 0 mentre quella a priori non nulla, invece dovrebbero coincidere per definizione di cifrario perfetto).

Dimostrazione: Per ogni messaggio possibile m vale $\mathcal{P}(M = m) > 0$. Sia N_m il numero dei messaggi possibili, e sia N_k il numero delle chiavi.

Per assurdo $N_k < N_m$. A un crittogramma c corrispondono $s \leq N_k$ messaggi[9] ottenuti provando a decrittare c con tutte le chiavi. Poiché $s \leq N_k < N_m$ allora esiste almeno un messaggio m^* impossibile da ottenere da c[10], dunque $\mathcal{P}(M = m^* \,|\, C = c) = 0 \neq \mathcal{P}(M = m^*) > 0$ e il cifrario non sarebbe perfetto.

Dunque il numero delle chiavi deve essere maggiore o uguale al numero dei messaggi se il cifrario è perfetto.

5.2 Esempio di cifrario perfetto: One-Time Pad

Il **one-time pad** è un cifrario perfetto di tipo **xor**, in cui la chiave è una stringa binaria casuale lunga quanto il testo da cifrare. La cifratura avviene bit-a-bit come $c_i \leftarrow m_i \oplus k_i$ dove c_i, m_i, k_i sono rispettivamente lo i-esimo bit di crittogramma, messaggio e chiave. La decrittazione avviene simmetricamente con la chiave k bit-a-bit $m_i \leftarrow c_i \oplus k_i$.

[8] Esiste solo una decrittazione per ogni chiave, e sono minori del numero dei messaggi possibili.

[9] Non per forza distinti.

[10] La funzione decrittazione non è suriettiva

5.2.1 Teorema: il one-time pad è perfetto

Nell'ipotesi che ogni messaggio è possibile e che tutti i messaggi abbiano la stessa lunghezza (basta aggiungere un padding di 0 per esempio):

Calcoliamo la probabilità condizionata, ottenendo

$$\mathcal{P}(M = m \,|\, C = c) = \frac{\mathcal{P}(M = m,\, C = c)}{\mathcal{P}(M = m)} = \frac{\mathcal{P}(M = m)\mathcal{P}(C = c)}{\mathcal{P}(M = m)} = \mathcal{P}(M = m)$$

dalla definizione di probabilità congiunta di due eventi indipendenti $\mathcal{P}(M = m,\, C = c) = \mathcal{P}(M = m)\mathcal{P}(C = c)$ e dalla definizione di probabilità condizionale (al secondo membro dell'uguaglianza).

NOTA BENE

Anche nel caso dei linguaggi naturali in cui non tutte le stringhe di n bit sono sensate e appartengono al linguaggio, il one-time pad rimane perfetto. Il numero minimo di chiavi (cardinalità dello spazio delle chiavi) richiesto è inferiore ma sempre esponenziale nella lunghezza del messaggio, dunque il one-time pad in questo caso non ha un numero di chiavi *minimale*.

6 Euclide Esteso

Dati due numeri a, b cerchiamo la tripla di interi $(MCD(a,b), x, y)$ dove $ax + by = MCD(a,b)$:

Euclide_Esteso(a, b):
  if $b == 0$://caso base
    return $(a, 1, 0)$
  (mcd,x, y) $\leftarrow$ Euclide_Esteso$(b, a \mod b)$
//^ scambio a con b e b con il modolo di a
  return mcd,$y, x - \lfloor \frac{a}{b} \rfloor y$ [11]

---

7 Cifrario a chiave pubblica

Per n utenti bastano n coppie di chiavi pubblica-privata (contro le circa n^2 dei cifrari simmetrici[12]).

Chiavi $k[pub]$ per cifrare verso un destinatario e $k[prv]$ usata dal destinatario per decifrare. La $k[pub]$ è conosciuta da tutti, la $k[prv]$ solo dal destinatario.
Mitt invia a *Dest* il messaggio m come $C(m, k[pub]) = c$ # computazionalmente facile
Dest decritta c con $D(c, k[prv])$ # anche questo facile
L'eventuale funzione $D(c, k[pub])$ che un *attaccante* potrebbe sfruttare è invece difficile computazionalmente.

Questi cifrari sono tuttavia più lenti dei simmetrici (in cui $k[pub] = k[prv]$).

[11] $\lfloor \frac{a}{b} \rfloor$ è il quoziente intero della divisione di $\frac{a}{b}$.
[12] $n(n-1)/2$

Sono esposti ad audit chosen-plain-text: se voglio sapere se un messaggio specifico $m*$ viene inviato a un certo Dest mi basta calcolare $C(m*, k[pub])$ e controllo se questo crittogramma appaia sul canale di comunicazione.

8 RSA

RSA:
È un cifrario a blocchi a chiave pubblica basato sulla funzione *one-way* esponenziale modulare.

k[pub]=(e,n) con $n = pq$ e p e q primi e $e < \Phi(n)$[13]
k[prv]= e_Φ^{-1}, l'inverso di e modulo $\Phi(n)$

Cifratura: Divido il messaggio in blocchi m di $\lfloor \log_2 n \rfloor$ e applico
C(m,k[pub])= $m^e \mod n$

Decifrazione: D(c,k[prv])=$c^{e_\Phi^{-1}} \mod n$

Algoritmo di generazione delle chiavi:

---

0. Scelgo p, q primi di (almeno) un migliaio di bit (per esempio con Miller-Rabin).
   Calcolo le chiavi in 3 fasi:

1. $n \leftarrow pq$
   $\Phi(n) \leftarrow (p-1)(q-1)$
2. Scelgo un $e < \Phi(n)$, coprimi fra loro[14]
   Se non fossero coprimi la funzione di cifratura non sarebbe invertibile
3. Calcolo $e_{\Phi(n)}^{-1}$ # inverso di $e \mod \Phi(n)$, cioè $e\, e_{\Phi(n)}^{-1} \mod \Phi(n) = 1$, calcolato con Euclide Esteso
4. k[pub]$\leftarrow$ (e,n), k[prv]$\leftarrow e_{\Phi(n)}^{-1}$

8.1 Dimostrazione di correttezza

Poiché abbiamo scelto e coprimo, dobbiamo dimostrare solo che

$$c^{e_\Phi^{-1}(n)} \mod n = (m^e \mod n)^{e_\Phi^{-1}(n)} \mod n = m$$

Ci sono due casi:

p e q non dividono m: Usando le proprietà dell'esponenziazione modulare, $(m^e \mod n)^{e_\Phi^{-1}(n)} \mod n = m^{e\, e_\Phi^{-1}(n)} \mod n$ ma $e\, e_\Phi^{-1}(n) = 1 + r\Phi(n)$ per definizione, dunque il precedente è uguale a m per le proprietà delle congruenze per $m < n$.

[13] Funzione Totiente di Eulero: numero di primi minori dell'argomento. Nel caso del prodotto di due primi vale $(p-1)(q-1)$, altrimenti $\prod 1 - \frac{1}{p}$ dove p è un fattore primo di n.

[14] e chiamato così perché è l'esponente nella cifratura

solo uno fra p e q divide m: Sfrutto il fatto di poter dividere la congruenza $c \mod (pq)$ fra $c \mod p$ e $c \mod q$ e riapplico poi l'argomento precedente

Questo vale comunque finché cifro e decifro il messaggio in blocchi di $\lfloor \log_2 n \rfloor$ bit (cioé il valore numerico del messaggio rappresentato in bit è minore n), altrimenti l'inversione delle congruenze non sarebbe possibile in modo univoco.

8.2 Attacchi a RSA

La sicurezza di RSA è equivalente alla fattorizzazione di della chiave pubblica Oltre ai chosen-plain-text a cui tutti i cifrari a chiave pubblica sono soggetti **RSA** può subire alcuni altri attacchi per la sua struttura matematica. La sicurezza del cifrario si fonda sulla difficoltà di fattorizzare n quando è un numero composto molto grande, perché estrarre la radice modulare e-esima è un problema altrettanto difficile. Un altro attacco potrebbe essere calcolare $\Phi(n)$ e trovare dalla parte e di k[pub] la chiave privata, ma questo ancora una volta è altrettanto difficile che fattorizzare n, dimostrazione:

Supponiamo che esista un algoritmo per calcolare $\Phi(n)$ senza scomporre prima n in fattori primi. Tuttavia possiamo calcolare la scomposizione in fattori di n da $\Phi(n)$ in tempo polinomiale, quindi questo problema è computazionalmente equivalente alla scomposizione, infatti: $Phi(n) = (p-1)(q-1) = pq-(p+q)+1 = n-(p+q)+1$ dunque $p+q = n-\Phi(n)+1$ e $(p-q)^2 = (p+q)^2 - 4n$ da cui possiamo calcolare p e q in tempo polinomiale.

Accorgimenti per i casi vulerabili Se si sceglie la chiave privata nel modo sbagliato rompere il **RSA** diventa piuttosto semplice. Oltre a richiedere numeri lunghi almeno qualche migliaio di bit, a causa dell'efficienza corrente dell'aritmetica hardware, bisogna scegliere p e q distanti fra loro: se $|p-q|$ fosse piccolo $\frac{p+q}{2}$ sarebbe vicino a $\sqrt{n}$, e dal valore di questa radice basterebbe cercare un valore z in $[\sqrt{n}, n)$ tale che $z^2 - n$ è un quadrato perfetto, e testarlo eventualmente sul crittogramma. Nella pratica funziona molto velocemente.

Altri attacchi più sofisticati sono possibili se il MCD di $p-1$ e $q-1$ (che sono sicuramente composti perché p, q sono primi) è grande.

Esistono vincoli di sicurezza anche sulla scelta dell'esponente e, per esempio se questo non induca alcun tipo di cifratura sul messaggio (per $e = 1 + \Phi(n)/k$ dove k è un divisore di $\Phi(n)$).

Un problema peggiore è dato dal **teorema cinese del resto**, se più utenti scelgono uno stesso e piccolo: si passa così da dover forzare il cifrario con la radice e-esima modulare a fare semplicemente la radice e-esima aritmetica.

Possibilmente anche n deve essere diverso fra tutti gli utenti per evitare attacchi basati sull'applicazione dell'algoritmo di Euclide Esteso.

Inoltre RSA è a blocchi, per cui per eliminare le sue periodicità possibili è necessario usare un *Cifrario a composizione di Blocchi.*

9 Diffie-Hellman classico

Scambio di chiavi di sessione per cifrari ibridi Diffie-Hellman è un protocollo di scambio *chiavi di sessione*. Scambiate le chiavi, con metodi a chiave *pubblica*, il resto della conversazione continuerà con un cifrario simmetrico di chiave pari alla chiave di sessione. È basato sul logaritmo discreto (calcolare potenze modulari è facile, $\mathcal{O}(\log k)$ moltiplicazioni, tramite il metodo iterativo delle quadrature successive), trovare l'esponente in $y = a^e mod n$ è difficile sapendo y, n o a

1. k[pub]$\leftarrow (p, g)$
   accordo pubblico su p primo grande e g generatore di Z_p^* (che esiste perché p è primo)

2a. Alice sceglie $k_A[prv] \leftarrow x < p$, invia $k_A[pub] \leftarrow g^x \mod p \leq p - 1$

2b. Bob sceglie $k_B[prv] \leftarrow y < p$, invia $k_B[pub] \leftarrow g^y \mod p$

3a. Alice calcola $k_A[prv] \leftarrow k_B[pub]^{k_A[prv]} \mod p$

3b. Bob calcola $k_B[prv] \leftarrow k_A[pub]^{k_B[prv]} \mod p$

ma $k_A[prv] = k_B[prv] = k[prv] = g^(xy) \mod p$, dunque il protocollo è corretto.
Attacchi: attacco attivo *man-in-the-middle*, *Eve* si frappone fra Alice e Bob, invia false chiavi e si finge il destinatario con ognuno. Una *Certification Authority* può mitigare il problema.

10 Crittografia su curve ellittiche e logaritmo discreto

10.1 Generalità su curve ellittiche

Consigliato
La *Crittografia su curve ellittiche* **ECC** sfrutta la difficoltà dell'inversione di alcune operazioni semplici sulle strutture algebriche definite sui punti di particolari curve dette ellittiche (per ragioni storiche).

Dato un campo K, definiamo *caratteristica del campo* il minimo numero di volte per cui sommando l'elemento neutro moltiplicativo (1) con se stesso si ottiene l'elemento neutro additivo (0) del campo. Prendendo curve su campi di caratteristica maggiore di 3, considereremo curve ellittiche (sempre esprimibili) nella *forma normale di Weiestrass*

$$y^2 = x^3 + ax + b$$

In particolare possiamo definire curve ellittiche sul campo degli interi modulo p $\mathbb{Z}_p$ con p numero primo (maggiore di 3 per avere la caratteristica del campo

compatibile con la forma normale), dunque avremo una *curva ellittica prima* di primo p e coefficienti a, b

$$E_p(a,b) = \{(x,y) |\, y^2 \mod p = x^3 + ax + b \mod p\}$$

(simmetrica rispetto a $y = \frac{p}{2}$, *non importante*), e definiremo l'**ordine** della curva come il suo numero di punti, per il quale vale il seguente *teorema di* ***Hasse*** (senza dimostrazione): *l'ordine* N *di una* $E_p(a,b)$ *verifica* $|N - (p+1)| \leq 2\sqrt{p}$.

Somma su curve ellittiche e LOGARITMO DISCRETO ELLITTICO La somma fra punti di una curva ellittica, definita in un modo particolare, forma un gruppo abeliano[15].

È possibile inoltre definire dalla somma il prodotto kP di un punto P per un intero k ricorsivamente come $0P = O$, $(k+1)P = P + kP$.

Un punto ha **ordine** n se è possibile generare n diversi punti della curva ellittica sommando quel punto a se stesso un numero finito di volte (dunque l'*ordine* di qualsiasi punto è minore all'*ordine della curva*, cioè al suo numero totale di punti).

Così siamo in grado definire l'analogo ellittico del logaritmo discreto: dati due punti Q, P trovare il minimo k tale che $Q = kP$, ossia il logaritmo discreto in base P di Q $\log_P Q$ è il minimo intero verificante

$$Q = \log_P(Q)\, P$$

e la sua esistenza è quindi intimamente connessa all'ordine di P.

Raddoppi successivi e complesssità Algoritmo di $\mathcal{O}(\log k)$ prodotti per calcolare kP con i raddoppi successivi: binary(k,i) è l'i-esimo bit dal meno significativo al più significativo della rappresentazione binaria di k

---

```
RaddoppiSuccesivi(P, k):
    tmp ← P
    sum ← binary(k, 0) tmp
    for i ← 1 to ⌊log_2 k⌋
        tmp ← 2 tmp
        sum ← binary(k, i) tmp
/* sommo solo quando l'i-esimo bit di k non è nullo */
    return sum
```

Vi è una identificazione informale fra la difficoltà dei problemi di aritmetica modulare e su curve ellittiche: ai prodotti modulari corrispondono le somme su curve ellittiche, e alle potenze modulari i prodotti su curve ellittiche (con i rispettivi inversi, come detto per il *logaritmo discreto*).

[15] Si comporta come la somma fra numeri interi.

11 Diffie-Hellman ellittico

Nella variante ellittica, Alice e Bob si scambiano punti su una curva ellittica. Alla fine questi punti verranno interpretati come un numero tipicamente prendendo l'ascissa x_P del punto e facendone il modulo per 256: blocco $\leftarrow x_p \mod 256$.

k[pub]: $E_p, B \in E_p$

Il punto B si comporta come il generatore in *DH classico* Se N è l'ordine di E_p, allora poniamo $n < N$ l'ordine del punto B

Alice:
k_A[prv]$\leftarrow n_A < n$
k_A[pub]$\leftarrow n_A B$
Bob:
k_B[prv] $\leftarrow n_B < n$
k_B[pub] $\leftarrow n_B B$

Chiave privata k[prv]$\leftarrow S = n_A n_B B = n_A k_B[pub] = n_B k_A[pub]$

Nel protocollo Alice e Bob si scambiano le chiavi pubbliche e calcolano successivamente la chiave privata di sessione come sopra. Come il **DH** classico è soggetto ad attacchi attivi (tipo *man-in-the-middle*).

12 Algoritmo di Koblitz e protocollo di El Gamal

12.1 Algoritmo di Koblitz

Non è nota una funzione $f \in P$ per associare un ascissa (per esempio la codifica binaria di un messaggio) a un punto di una E_p, necessario per tradurre efficientemente[16] problemi numerici in problemi su punti di curve ellittiche. Esiste però un algoritmo randomizzato in tempo polinomiale RP per questo problema conosciuto come *Algoritmo di Koblitz*.

La probabilità $prob = p(P_m \in E_p(a,b))$ di trovare un punto di una curva $E_p(a,b)$ che abbia un certo valore m come ascissa è la probabilità che esista un y il cui quadrato sia uguale a $m^3 + am + b(\mod p)$ (in tal caso il polinomio nella m è chiamato *residuo quadratico*).

Si può dimostrare che $prob = \frac{1}{2}$ (dal teorema di Hasse $|N - (p+1)| \leq 2\sqrt{p}|$, *senza dimostrazione*)

Dunque prendo (aleatoriamente) un h t.c. $(m+1)h = mh + h < p$

/*scorro le ascisse in $[mh, mh+h)$ finché non trovo una associata a $P \in E_p$*/
for $i \in [0, h)$:
$x \leftarrow mh + i$

[16] Altrimenti si perde la semplicità di cifratura.

if $x^3 + ax + b(\mod p)$ è un *residuo quadratico*
return x

Probabilità di successo $1 - 2^{-h}$ (tende a 1 esponenzialmente)[17].

Decrittazione $P_m = (x, y) \to m = \lfloor \frac{x}{h} \rfloor$ dove m è la codifica numerica del messaggio.

12.2 El Gamal

Un protocollo crittografico che sfrutta algoritmi simili al precedente è per esempio il cifrario asimmetrico di *El Gamal*

k[pub] $= (E_p(a, b), B)$ con B di ordine n elevato

Scelta chiave: $n_{user} < n$, $P_{user} = n_{user}B$

Cifratura:

1. $m \mapsto P_m$ con per esempio *Koblitz*
2. prendo a caso r: $V \leftarrow rB, W \leftarrow P_m + rP_{dest}$ dove P_{dest} è chiave pubblica del destinatario

Decifrazione: con n_D la chiave privata in mano al destinatario, ricordando $P_D = n_D B$

$$P_m = W - n_D V = P_m + rP_D - n_D rB$$

Se un crittoanalista intercetta $(V = rB, W = P_m + rP_D)$ dovrebbe calcolare r per conoscere il messaggio, dunque risolvere un problema di logaritmo discreto su EC.

13 Sicurezza su curve ellittiche vs altri

Per il problema della *fattorizzazione* e per il *logaritmo discreto modulare* esistono algoritmi **subesponenziali**[18]. Per il *logaritmo discreto* **su curve ellittiche** il miglior algoritmo è invece ancora pienamente esponenziale[19].

Inoltre i problemi su **curve ellittiche**, a parità di una complessità dell'implementazione con **RSA**, permettono chiavi nell'ordine di centinaia di bit, contro le diverse migliaia dei cifrari asimmetrici non basati sulle curve ellittiche per sicurezza simile.

Possiamo rifarci, per comparazione a una **misura** "fisica" **universale di sicurezza**: l'energia[20] per forzare e.g. **RSA** a 228 bit è pari circa a quella impiegata per bollire un cucchiaio d'acqua, mentre quella per forzare un cifrario basato sulle **curve ellittiche** a pari lunghezza di chiave è circa l'energia necessaria a far bollire tutta l'acqua presente sulla Terra.

[17] La probabilità di trovare ogni volta numeri che non siano residui quadratici, se h è casuale, è il prodotto delle probabilità a priori che non lo sia, che per il teorema citato è 0.5, dunque $0.5^{numero\ tentativi}$. Poi è stata presa la probabilità complementare.

[18] *index calculus*, $\mathcal{O}(2^{\sqrt{n \log n}}$ contro lo $\mathcal{O}(2^n)$ di bruteforce

[19] Al 2014 il *Pollard* $\rho \in \mathcal{O}(2^{\frac{n}{2}})$ che è ancora pienamente esponenziale perché $2^{\frac{n}{2}} = \sqrt{2}^n$.

[20] Quantomeno allo stato attuale dell'arte.

14 Zero-knowledge Proof

Nelle **zero-knowledge proof** le computazioni, in questo caso i protocolli crittografici, sono viste come *dimostrazioni interattive* (ossia c'è uno scambio di messaggi durante l'esecuzione) *a conoscenza zero*. Illustriamo il concetto con un esempio.

Immaginiamo di avere una deficienza nella visione dei colori[21] e che un normovedente voglia dimostrarci che le due palle che abbiamo in mano, che a noi sembrano perfettamente identiche, siano di diverse. Il normovedente ne vede i colori, che distingue e conosce solo lui.

Possiamo essere ***probabilmente*** sicuri, con una probabilità grande a piacere, del fatto che non menta e che trovi delle differenze nelle due palle se il normovedente indica un certo numero consecutivo di volte fissato da noi quale palla era originariamente a sinistra o destra dopo che noi le abbiamo scambiate in segreto da lui: se almeno una volta non indovinasse saremmo sicuri che mentiva, mentre la probabilità in k ripetizioni che indovini quale fosse la palla originariamente a destra o sinistra se le palle sono uguali è $\frac{1}{2^k}$.

In tutto questo, se effettivamente il normovedente ne riconosce i colori non abbiamo potuto provare che mentiva, verificando di volta in volta la correttezza delle sue indicazioni. Se invece è disonesto potrà ingannarci solo con una probabilità infinitesima. In ogni caso così non saremo venuti a conoscenza dei colori delle sfere.

Una **zero-knowledge proof** vede due utenti, il provatore P e il verificatore V. P cerca di *provare* a V che conosce un certo *segreto* (una informazione) senza rivelarlo a V, mentre V *verifica* che gli argomenti di P siano logicamente corretti. Una **Z-K Proof** ha tre caratteristiche:

> **Completezza**: se una affermazione q di P è vera, allora passerà sempre la verifica di V.
>
> **Correttezza** (probabilistica, asintotica): se una affermazione q di P è falsa, la probabilità che possa passare la verifica di V è $\leq \frac{1}{2^k}$ per un k arbitrario scelto di volta in volta da V.
>
> **Zero-Knowledge**: se P possiede davvero un segreto s, anche se V fosse disonesto[22] nell'uso del protocollo sarebbe strutturalemente impossibile per lui ottenere dei bit del segreto di P.

Un ottimo esempio di impiego di un sistema simile, per l'affidabilità illimitata e la *conoscenza zero* sono i sistemi di identificazione, classicamente soggetti a diversi tipi di furto delle informazioni specie se nel caso di autorità centrali disoneste.

Un sistema *zero-knowledge* necessità comunque dell'utilizzo di funzioni *one-way*, rendendo però il protocollo a *conoscenza zero*.

[21] Quindi non distinguiamo i colori ma solo le forme

[22] Facesse qualsiasi azione non standard per tentare di forzare il protocollo.

14.1 Protocollo Fiat-Shamir

Il protocollo *zero-knowledge Fiat-Shamir* ha come principale obiettivo l'*autenticazione* (verifica dell'identità di un utente) senza che vi siano *Certification Authority* e senza comunicare alcun bit dei propri dati segreti (chiavi private). Questo permette di risolvere problemi di sicurezza insiti in molti protocolli di autenticazione.

Utilizza la funzione one-way $s^2 \mod n$ (dove n è prodotto di due primi grandi), perché calcolare $s^2 \mod n$ è semplice mentre l'estrazione della radice (in questo caso quadrata) modulare è difficile[23].

Sia $n = pq$ con p e q primi Sia $s < n$ k[prv]= s k[pub]=$(n, s^2 \mod n = S)$

V (verificatore) conosce la chiave pubblica.

Il protocollo consiste in un numero arbitrario di iterazioni deciso da V che si svolgono come la seguente interazione:

Inizio interazione
P sceglie un $r < n$ poi invia $r^2 \mod n = R$ a V
V risponde con un bit casuale $e \in \{0, 1\}$ inviandolo a P
P deve ora calcolare $rs^e \mod n$ e inviarlo a V
fine interazione

V può ora verificare[24] che P sia in possesso del valore s conoscendo solo k[pub] facendo:

---

if $RS \mod n == (rs^e \mod n)^2 \mod n$
    return OK
else return *non verificato*

La ***completezza*** del protocollo discende dal fatto che se P conosce davvero s (cioè la radice quadrata modulare di S della chiave pubblica), qualunque sia r scelto all'inizio e qualunque sia il bit e inviato da V, $RS \mod n = (r^2 \mod n)(s^2 \mod n) \mod n = (rs)^2 \mod n$ [25] e sarà dunque uguale a $(rs^e \mod n)^2 \mod n$

[23] Confrontare con **RSA** dove il problema difficile era l'estrazione di una generica radice n-esima. Non facendo trapelare bit sulla chiave segreta in *Fiat-Shamir* evitiamo i calssici problemi di **RSA**.

[24] Con un errore $\frac{1}{2}$, che tenderà a 0 aumentando le iterazioni

[25] Dalla proprietà dell'aritmetica modulare $ab \mod n = (a \mod nb \mod n) \mod n$ e dalle proprietà delle potenze.