Appunti di Sicurezza delle reti

Lorenzo Prosseda, a.a. 2018-2019

(con correzioni di Kien Tuong Truong a.a. 2019-2020)

Indice

Parte 1

Crittografia

CAPITOLO 1

Teoria dei numeri

1.1. Proprietà degli interi

Da ora in avanti parleremo di numeri interi, positivi o negativi, operando all'interno dell'insieme $\mathbb{Z}$; enunciamo la proprietà di divisione nel modo seguente: presi due interi a, $b \in \mathbb{Z}$ non uguali ($a \neq b$) si dice che a divide b quando a è un divisore di b, ovvero

$$a \backslash b \implies \exists k \,:\, b = k \cdot a$$

Dalla precedente deduciamo che b dovrà essere un multiplo di a. Inoltre, otteniamo anche che:

$$\forall a \in \mathbb{Z} \,:\, a \backslash 0;\ \nexists a \in \mathbb{Z} \,:\, 0 \backslash a;\ \forall a \in \mathbb{Z} \,:\, a \backslash a$$

La relazione di divisione introdotta ammette la proprietà transitiva:

$$\forall a, b, c \in \mathbb{Z} \wedge a \neq b \neq c \,:\, a \backslash b \wedge b \backslash c \implies a \backslash c$$

1.2. Numeri primi

1.2.1. Definizione e proprietà. Un numero a si dice primo quando è divisibile solo per 1 e per sé stesso (ovvero se vale $\forall b \in \mathbb{Z} \,:\, b \backslash a \iff b = 1 \vee b = a$); un numero composto è scomponibile in un numero finito di fattori, e questa scomposizione è unica. Determinare la primalità di un numero tuttavia non è cosa facile; introduciamo il seguente teorema:

THEOREMA 1.1. *Sia $\pi(n) :=$ "numero di numeri primi fino a n", allora vale*

$$\pi(n) \sim \frac{n}{\ln(n)}$$

Teorema dei numeri primi

Alcuni algoritmi crittografici usano i numeri primi come "ingredienti" per creare le chiavi: in questi casi il teorema introdotto si dimostra molto utile per determinare la quantità di numeri primi che è possibile ottenere con una data quantità di cifre.

ESEMPIO 1.1. Determinare una stima della quantità di numeri primi che è possibile ottenere a partire da 100 cifre.

✓Usando il Teorema 1.1 possiamo scrivere la quantità di numeri primi con 100 cifre come

$$\pi\left(10^{100}\right) - \pi\left(10^{99}\right) = \frac{10^{100}}{100\ln(10)} - \frac{10^{99}}{99\ln(10)} \simeq \boxed{10^{97}}$$

□

Se fossimo nel contesto di un algoritmo di cifratura, pur sapendo che la chiave sia un numero primo di 100 cifre, dovremmo analizzare in ogni caso 10^{97} possibili candidati.

Prendiamo ora a, $b \in \mathbb{Z}$ tali che $\text{MCD}(a, b) = 1$: in tal caso diremo che a e b sono *coprimi* o *primi relativi*, indicando la loro relazione come $a \perp b$; da questa relazione segue che due numeri coprimi non hanno fattori in comune.

La funzione $\text{MCD}(a, b)$ indica il massimo comune divisore tra a e b

Escluso il numero 2, tutti i primi sono dispari, e sono divisi in due classi; preso un numero primo p, esso appartiene a una delle seguenti classi:

- $p \equiv 1 \pmod 4$
- $p \equiv 3 \pmod 4$

L'operatore $\equiv$ indica la congruenza in modulo: si dice che un numero $a \in \mathbb{Z}$ è *congruente a 1 modulo n* (e si scrive $a \equiv 1 \pmod n$) se il resto della divisione di a per n è 1.

Componendo le due classi osserviamo che tutti i numeri primi p possono essere indicati come

$$p = 6k \pm 1 \implies p \equiv \pm 1 \pmod 6,\ k \in \mathbb{Z} \tag{1.2.1}$$

Troveremo per esempio $p_{k=1} = 6 \pm 1 = \{5, 7\}$, $p_{k=2} = 12 \pm 1 = \{11, 13\}$, $p_{k=3} = 18 \pm 1 = \{17, 19\}$, ...; osserviamo che tutti i numeri della successione appena definita sono primi, tuttavia non tutti i primi appartengono a questa successione.

1.2.2. Test di primalità con classi. Possiamo usare la successione (1.2.1) per testare la primalità di un numero intero: sia $n > 0$ un numero del quale si vuole conoscere la primalità; allora definiamo un algoritmo iterativo che costruisce la successione (1.2.1) incrementando k, e per ogni primo p_k ottenuto, se non vale $p_k \backslash n$ fino a che $6k + 1 \leq \sqrt{n}$, allora n è primo.

1.3. Algoritmo di Euclide

1.3.1. Definizione della successione. La funzione principale di questo algoritmo, è calcolare il massimo comune divisore di due numeri; presi due interi m e n tali che $m < n$, vogliamo calcolare $\text{MCD}(m, n)$. Tramite questo algoritmo otteniamo il risultato desiderato, senza passare per la scomposizione in fattori primi di m e n; definendo la seguente successione:

$$\begin{aligned} \text{MCD}(m, n) &= \text{MCD}(n \bmod m, m) \\ &\vdots \\ &= \text{MCD}(0, d) = d \end{aligned} \tag{1.3.1}$$

Osserviamo come calcolare questa successione con un esempio.

Esempio 1.2. *Calcolare il massimo comune divisore tra 482 e 1180*

✓Procediamo applicando la definizione (1.3.1): per farlo dovremo scomporre il numero maggiore (n dalla definizione) usando il suo modulo rispetto al minore (m); in pratica prendiamo 1180 e lo dividiamo per 482, conservando il residuo dell'operazione da usare nel termine successivo della successione

$$\text{MCD}(482, 1180) \to 1180 = 2 \cdot 482 + 216$$

proseguiamo lavorando col resto del passo precedente (216) e col valore precedentemente usato per calcolare il modulo (482)

$$\begin{aligned} \text{MCD}(216, 482) &\to 482 = 2 \cdot 216 + 50 \\ \text{MCD}(50, 216) &\to 216 = 4 \cdot 50 + 16 \\ \text{MCD}(16, 50) &\to 3 \cdot 16 + \mathbf{2} = \boldsymbol{d} \\ \text{MCD}(2, 16) &\to 8 \cdot 2 + \boxed{0} \end{aligned}$$

La successione termina quando si ottiene resto zero; il resto chiamato d, ottenuto alla penultima riga (sopra a quella con resto 0, vale 2 in questo caso) è effettivamente il risultato della richiesta: $\text{MCD}(482, 1180) = 2$ □

1.3.2. Algoritmo in forma simbolica. Presi a, $b \in \mathbb{Z}$ tali che $a < b$, otteniamo $\text{MCD}(a, b) = d$ applicando (1.3.1) nel seguente modo:

Algoritmo di Euclide

$$\begin{aligned} b &= q_1 \cdot a + r_1 \\ a &= q_2 \cdot r_1 + r_2 \\ r_1 &= q_3 \cdot r_2 + r_3 \\ &\vdots \\ r_{k-2} &= q_k \cdot r_{k-1} + r_k \\ r_{k-1} &= q_{k+1} \cdot r_k + 0 \\ \boxed{r_k &= d} \end{aligned} \tag{1.3.2}$$

dove q_i è l'i-esimo quoziente e r_i è l'i-esimo resto; dall'algoritmo si può dedurre che, presi a, $b \neq 0$ e sia $d = \text{MCD}(a, b)$, allora è vero che $\exists x, y \in \mathbb{Z} \,:\, a \cdot x + b \cdot y = d$. Questo risulta chiaro se immaginiamo che i due interi cercati siano anche negativi; per trovare tali interi è necessario utilizzare una estensione del (1.3.2).

1.3.3. Algoritmo esteso. L'algoritmo di Euclide presentato nella sottosezione precedente può essere esteso, impiegando nel suo svolgimento due successioni $\{x_k\}$ e $\{y_k\}$: esse avranno i primi due valori ben definiti, come

$$x_0 = 0,\ x_1 = 1;\ y_0 = 1,\ y_1 = 0 \tag{1.3.3}$$

Facendo corrispondere ai passi (1.3.2) gli elementi delle successioni $\{x_k\}$ e $\{y_k\}$, possiamo ottenere gli elementi dal secondo in poi come:

Algoritmo di Euclide esteso

$$\begin{aligned} x_2 &= -q_1 \cdot x_1 + x_0 & y_2 &= -q_1 \cdot y_1 + y_0 \\ x_3 &= -q_2 \cdot x_2 + x_1 & y_3 &= -q_2 \cdot y_2 + y_1 \\ x_4 &= -q_3 \cdot x_3 + x_2 & y_4 &= -q_3 \cdot y_3 + y_2 \\ &\vdots & &\vdots \\ \boxed{x_{k+1}} &= -q_k \cdot x_k + x_{k-1} & \boxed{y_{k+1}} &= -q_k \cdot y_k + y_{k-1} \end{aligned} \tag{1.3.4}$$

Osserviamo che gli elementi delle successioni in x e y si ottengono in modo analogo, tuttavia le due successioni sono inizializzate in modo differente (1.3.3).

Si può dimostrare che, dati $a, b \in \mathbb{Z}$ con $d = \text{MCD}(a, b)$, vale l'identità di Bezout:

$$\exists s, t \in \mathbb{Z} | a \cdot s + b \cdot t = d \tag{1.3.5}$$

In particolare, i valori s e t corrispondono rispettivamente ai valori x_{k+1} e y_{k+1} trovati mediante l'algoritmo di Euclide esteso, che ci permette quindi di affermare che

$$d = a \cdot x_{k+1} + b \cdot y_{k+1}$$

Normalmente uno dei due coefficienti tra s e t è positivo e l'altro è negativo.

1.4. Congruenza in modulo

1.4.1. Definizione e proprietà. Nelle sezioni precedenti abbiamo introdotto il simbolo di congruenza '$\equiv$', che usato nel modo seguente implica che

$$a \equiv b \,(\text{mod}\, n) \implies a \bmod n = b \bmod n \implies a - b = k \cdot n$$

ovvero il fatto che a sia congruente in modulo n a b implica il fatto che a e b siano multipli; possiamo infatti dedurlo osservando l'ultima implicazione, riscritta come

$$a = b + k \cdot n$$

Per la congruenza in modulo valgono alcune proprietà simili a quelle dell'uguaglianza:

- $a \equiv 0 \,(\text{mod}\, n) \iff n \backslash a$

- $\forall a \in \mathbb{Z} \,:\, a \equiv a \,(\mathrm{mod}\, n)$
- $a \equiv b \,(\mathrm{mod}\, n) \iff b \equiv a \,(\mathrm{mod}\, n)$
- $a \equiv b \,(\mathrm{mod}\, n) \wedge b \equiv c \,(\mathrm{mod}\, n) \implies a \equiv c \,(\mathrm{mod}\, n)$

1.4.2. Insieme dei residui. Fin'ora abbiamo lavorato all'interno dell'insieme dei numeri interi $\mathbb{Z}$; introduciamo l'insieme dei residui modulo n, indicato come $\mathbb{Z}_n$, che contiene l'insieme dei valori da 0 a $n-1$ e al suo interno sono definite ciclicamente le operazioni di addizione, sottrazione e moltiplicazione; osserviamo un esempio sull'addizione.

ESEMPIO 1.3. *Determinare i risultati delle seguenti operazioni, all'interno dell'insieme* $\mathbb{Z}_{10}$*:* $4+5$, $5+5$, $2-3$, $3\cdot 4$.

✓Dalla definizione sappiamo che $\mathbb{Z}_{10} = \{0,1,2,3,4,5,6,7,8,9\}$, allora le operazioni richieste valgono:

$$4+5=9,\ 5+5=0,\ 2-3=9,\ 3\cdot 4=2$$

□

La divisione in un insieme dei residui non è definita in modo banale come le altre tre operazioni; introdurremo in seguito questa operazione.

1.4.3. Insieme ridotto dei residui. Dato un insieme dei residui $\mathbb{Z}_n$, possiamo definire il suo insieme ridotto $\mathbb{Z}_n^*$, che contiene gli elementi di $\mathbb{Z}_n$ che sono coprimi rispetto a n, ovvero

$$\forall z \in \mathbb{Z}_n^* \,:\, z \in \mathbb{Z}_n \wedge z \perp n$$

Procedendo con l'Esempio 1.3, l'insieme ridotto dei residui modulo 10 avrà i seguenti elementi al suo interno:

$$\mathbb{Z}_{10}^* = \{1,3,7,9\}$$

Notiamo che lo zero è sempre escluso dall'insieme ridotto, perché non è coprimo rispetto ad alcun intero; la cardinalità di $\mathbb{Z}_n^*$ è determinata da una funzione di n chiamata *toziente* o φ *di Eulero*; in particolare vale $\varphi(10) = 4$ (vedremo in seguito come calcolare questa funzione).

1.4.4. Equazioni congruenziali. Una equazione congruenziale è una relazione definita su un'insieme di residui; nel seguito vedremo delle proprietà che ci permetteranno di operare con queste relazioni.

PROPOSIZIONE 1.1. *Prendiamo un intero* $n \neq 0$ *e quattro interi* a,b,c,d *tali che* $a \equiv b \,(\mathrm{mod}\, n)$ *e* $c \equiv d \,(\mathrm{mod}\, n)$*; allora si verifica che*

$$a+c \equiv b+d \,(\mathrm{mod}\, n),\ a-c \equiv b-d \,(\mathrm{mod}\, n),\ a\cdot c \equiv b\cdot d \,(\mathrm{mod}\, n)$$

Possiamo usare queste proprietà per risolvere la seguente equazione congruenziale.

ESEMPIO 1.4. *Risolvere rispetto a* x *l'equazione* $x+7 \equiv 3 \,(\mathrm{mod}\, 17)$.

✓Possiamo sfruttare la seconda proprietà, per cui

$$\begin{gathered} x+7\mathbf{-7} \equiv 3\ \mathbf{-7} \,(\mathrm{mod}\, 17) \\ x \equiv -4 \,(\mathrm{mod}\, 17) \end{gathered}$$

Osservando che $17-4=13$, otteniamo infine (è equivalente a scrivere la congruenza con -4)

$$x \equiv 13 \,(\mathrm{mod}\, 17)$$

□

Ora introduciamo il concetto di divisione nell'insieme dei residui, con una seconda proposizione.

PROPOSIZIONE 1.2. *Prendiamo un intero $n \neq 0$ e tre interi a, b, c con $a \perp n$ e vale $a \cdot b \equiv a \cdot c \pmod{n}$; allora possiamo dire che $a \cdot b \equiv a \cdot c \pmod{n} \implies b \equiv c \pmod{n}$, tramite la moltiplicazione da ambo i lati per l'inverso a^{-1}*

Possiamo risolvere una forma di equazione congruenziale più elaborata.

ESEMPIO 1.5. *Risolvere rispetto a x l'equazione $2x + 7 \equiv 3 \pmod{17}$.*

✓Come nel caso precedente, possiamo sottrarre 7 da entrambi i lati, ottenendo

$$2x \equiv -4 \pmod{17}$$

Ora sfruttando il fatto che $2 \perp 17$ la precedente diventa

$$x \equiv -2 \pmod{17} = x \equiv 15 \pmod{17}$$

□

ESEMPIO 1.6. *Risolvere rispetto a x l'equazione $5x + 6 \equiv 13 \pmod{11}$.*

✓Sottraendo 6 da ambo i lati otteniamo

$$5x \equiv 7 \pmod{11}$$

Se ora usiamo la proprietà della divisione (vale $5 \perp 11$) dovremo scrivere

$$x \equiv \frac{\hat{7}}{5} \pmod{11}$$

dove la frazione $\hat{7/5}$ in realtà non esiste nell'insieme $\mathbb{Z}_{11}$; tuttavia sappiamo che $5x$ è congruente modulo 11 a 7, ma anche a (per la ciclicità dell'insieme dei residui) $7 + 11$, $7 + 22$, $\ldots$, $7 + k \cdot 11$; il primo numero che sia un multiplo di 5 si ha per $k = 3$; abbiamo ottenuto che $7 \equiv 40 \pmod{11}$ ma anche che $5 \backslash 40$, quindi possiamo scrivere

$$5x \equiv 40 \pmod{11}$$
$$x \equiv 8 \pmod{11}$$

Possiamo in un certo senso affermare che 8 si comporta come $7/5$ in $\mathbb{Z}_{11}$.

Esiste una seconda strada per risolvere questo esercizio; sapendo che 5 e 11 sono primi relativi:

$$5x \cdot \mathbf{5^{-1}} \equiv 7 \cdot \mathbf{5^{-1}} \pmod{11}$$

L'inverso $m \in \mathbb{Z}$ di un numero n è quel numero tale che $n \cdot m = 1$; dunque cerchiamo l'elemento di $\mathbb{Z}_{11}$ che moltiplicato per 5 risulta 1; Si può verificare che $m = 9$ è l'inverso di 5 in $\mathbb{Z}_{11}$, infatti $m \cdot 5 \equiv 1 \pmod{11}$, da cui segue che

$$x \equiv 7 \cdot 9 \pmod{11}$$

e dato che $7 \cdot 9 = 63 = 5 \cdot 11 + 8$ la precedente si scrive come

$$x \equiv 8 \pmod{11}$$

Abbiamo ottenuto il medesimo risultato a cui siamo arrivati attraverso il primo procedimento. □

OSSERVAZIONE 1.1. Presi $a \in \mathbb{Z}_n$ e $b \notin \mathbb{Z}_n^*$, non è definita la divisione $a \div b$ (deve infatti valere $b \perp n$, cioè $b \in \mathbb{Z}_n^*$).

OSSERVAZIONE 1.2. Se prendiamo l'equazione $a \cdot x \equiv b \pmod{n}$, essa ammetterà soluzione se vale $a \perp n$; in tal caso $\exists a^{-1} \in \mathbb{Z}_n^*$ e la soluzione sarà $x \equiv b \cdot a^{-1} \pmod{n}$.

Cosa possiamo concludere nel caso in cui, data l'equazione $E: a \cdot x \equiv b \,(\mathrm{mod}\, n)$, non sia vero che $a \perp n$, cioè nel caso in cui $\mathrm{MCD}\,(a,\, n) = d > 1$? In tal caso, l'equazione può non ammettere soluzione o ammetterne d.

Se $d \backslash b$ allora dividiamo per d tutte le quantità costanti, ottenendo

$$\overline{E}: \frac{a}{d}x \equiv \frac{b}{d} \left(\mathrm{mod}\, \frac{n}{d}\right)$$

questa equazione ha ora una soluzione, perché $\mathrm{MCD}\,(a,\, n) = d \implies \mathrm{MCD}\left(\frac{a}{d},\, \frac{n}{d}\right) = \frac{d}{d} = 1$ e ricadiamo nel caso dell'osservazione precedente.

Otteniamo la soluzione dell'equazione $\overline{E}$ e la chiamiamo x_0, la quale sarà in modulo n/d ($x_0 \in \mathbb{Z}_{n/d}$); dato che l'equazione di partenza E ha soluzioni in $\mathbb{Z}_n$,l'insieme delle soluzioni X corrisponderà a:

$$X = \left\{x_0,\, x_0 + \frac{n}{d},\, x_0 + 2\frac{n}{d},\, \dots\, ,\, x_0 + (d-1)\,\frac{n}{d}\right\}$$

Osserviamo l'esempio di un'equazione con queste caratteristiche.

ESEMPIO 1.7. *Risolvere rispetto a x l'equazione* $12x \equiv 21 \,(\mathrm{mod}\, 39)$.

✓Osserviamo subito che $\mathrm{MCD}\,(12,\, 39) = 3$, dunque l'equazione avrà 0 o 3 soluzioni; dividiamo le costanti per 3 e otteniamo

$$4x \equiv 7 \,(\mathrm{mod}\, 13)$$

da cui otteniamo $x \equiv \frac{\hat{7}}{4} \,(\mathrm{mod}\, 13)$ e trovando $4 \backslash (7 + 1 \cdot 13) = 20$ la precedente equazione fornisce una soluzione $x_0 \equiv 5 \,(\mathrm{mod}\, n)$; adesso possiamo ottenere tutte le soluzioni dell'equazione di partenza:

$$X = \left\{x_0 = \boxed{5},\, x_0 + \frac{39}{3} = \boxed{18},\, x_0 + \frac{2 \cdot 39}{3} = \boxed{31}\right\}$$

Possiamo immaginare le radici di E come dei punti su una circonferenza, a distanza d/n da x_0. □

Posto che valga $a \perp n$ (ovvero $\mathrm{MCD}\,(a,\, n) = 1$), come possiamo calcolare l'inverso a^{-1}? Potremmo tentare tutti i numeri da 1 a $n-1$ conducendo un'analisi esaustiva, tuttavia per n molto grande questo non è pratico; usiamo allora l'algoritmo di Euclide esteso (1.3.4): l'identità di Bezout (1.3.5) ci garantisce che:

$$\mathrm{MCD}\,(a,\, n) = 1 \implies \exists s, t \in \mathbb{Z}_n \,:\, a \cdot s + n \cdot t = 1$$

segue dalla precedente relazione che $a \cdot s = 1 - n \cdot t$, allora abbiamo $a \cdot s \equiv 1 \,(\mathrm{mod}\, n)$ e risolvendo l'equazione si ottiene

$$s \equiv a^{-1} \,(\mathrm{mod}\, n)$$

ovvero il numero s è proprio l'elemento finale x_{k+1} della sequenza di x dell'algoritmo; esso risulta essere anche il valore cercato dell'inverso a^{-1}.

1.5. Teorema cinese del resto

1.5.1. Applicazione ed enunciato. Mostriamo direttamente l'applicazione del teorema a un caso specifico: prendiamo $x \equiv 25 \,(\mathrm{mod}\, 42)$; possiamo esprimere $x = 25 + 6 \cdot (7 \cdot k) \rightleftarrows 25 + 7 \cdot (6 \cdot k)$, da cui ricaviamo

- per $x = 25 + 6 \cdot (7 \cdot k)$: $25 \bmod 6 = 1 \implies x \equiv 1 \,(\mathrm{mod}\, 6)$
- per $x = 25 + 7 \cdot (6 \cdot k)$: $25 \bmod 7 = 3 \implies x \equiv 4 \,(\mathrm{mod}\, 7)$

Dalle due congruenze più semplici ottenute possiamo scrivere il sistema

$$x \equiv 25 \,(\mathrm{mod}\, 42) \implies \begin{cases} x \equiv 1 \,(\mathrm{mod}\, 6) \\ x \equiv 4 \,(\mathrm{mod}\, 7) \end{cases}$$

Il teorema cinese del resto afferma che:

Teorema cinese del resto

THEOREMA 1.2. *Dati due interi n, m che siano primi relativi (deve valere $n \perp m$), e presi due interi a, b, allora il sistema delle congruenze*

$$\begin{cases} x \equiv a \,(\mathrm{mod}\, n) \\ x \equiv b \,(\mathrm{mod}\, m) \end{cases}$$

è equivalente alla singola congruenza

$$x \equiv c \,(\mathrm{mod}\, n \cdot m)$$

Il teorema afferma che, quanto introdotto all'inizio della sezione, può valere nel senso opposto in alcune condizioni specifiche. Osserviamo l'applicazione del teorema in un esempio.

ESEMPIO 1.8. *Sia dato il seguente insieme di congruenze:*

$$\begin{cases} x \equiv 3 \,(\mathrm{mod}\, 7) \\ x \equiv 5 \,(\mathrm{mod}\, 15) \end{cases}$$

Scrivere una forma equivalente, con una singola congruenza.

✓Dato che $7 \perp 15$, per il Teorema 1.2 possiamo affermare che esisterà una singola congruenza equivalente alle due della consegna; essa sarà scritta nella forma $x \equiv c \,(\mathrm{mod}\, 105)$. Per ottenere c possiamo provare i numeri multipli di 5, i quali in modulo 7 diano 3 come risultato; si ottiene facilmente $c = 80$ (possiamo scomporlo come $80 = 11 \cdot 7 + 3$).

Nel caso di numeri molto grandi, non è possibile scegliere di procedere per tentativi, ed è necessario usare gli strumenti della teoria dei numeri: se sappiamo che $x \equiv a \,(\mathrm{mod}\, n)$ e anche $x \equiv b \,(\mathrm{mod}\, m)$, allora possiamo scrivere

$$x = b + \overline{k} \cdot m \equiv a \,(\mathrm{mod}\, n)$$

risolvendo questa relazione si ottiene $a - b \equiv \overline{k} \cdot m \,(\mathrm{mod}\, n)$ da cui, rispetto a $\overline{k}$ otteniamo

$$\overline{k} \equiv (a - b) \cdot m^{-1} \,(\mathrm{mod}\, n)$$

Il valore $\overline{k}$ permette di ottenere $c = b + \overline{k} \cdot m$: esso infatti è congruente sia ad a in modulo n, sia a b in modulo m. □

1.5.2. Estensione. Il Teorema 1.2 ammette un'estensione a un caso generale, con N congruenze.

Teorema cinese del resto esteso

COROLLARIO 1.1. *Prendiamo N numeri interi, indicati come $m_i : i \in [1, N]$, tali che a coppie siano tutti coprimi ($\forall i, j \in [1, N] \wedge i \neq j : m_i \perp m_j$). Se abbiamo il seguente sistema di congruenze*

$$\begin{cases} x \equiv a_1 \,(mod\, m_1) \\ x \equiv a_2 \,(mod\, m_2) \\ \quad \vdots \\ x \equiv a_N \,(mod\, m_N) \end{cases}$$

è equivalente alla singola congruenza

$$x \equiv C \left(mod \prod_{i=1}^{N} m_i \right)$$

Tale congruenza si costruisce a partire dalle seguenti serie:

$$M = \prod_{i=1}^{N} m_i; \quad Z_i = \frac{M}{m_i}; \quad Y_i = Z_i^{-1} \bmod m_i$$

Otteniamo infine la seguente soluzione, equivalente a una singola congruenza:

$$X = \sum_{i=1}^{N} a_i Y_i Z_i \bmod M \tag{1.5.1}$$

1.6. Square & multiply

Tramite l'algoritmo chiamato square and multiply, è possibile calcolare il modulo di un numero che abbia molte cifre; mostriamo un esempio per illustrare l'idea dietro a questo algoritmo, usando numeri "piccoli".

ESEMPIO 1.9. *Si calcoli il risultato dell'operazione* $7^{11} \bmod 26$.

✓Eseguiamo i seguenti passi per cercare il valore desiderato:

LSB indica il least significant bit (bit meno significativo) e si riferisce al primo bit da destra

(1) Esprimiamo 11 (l'esponente) in forma binaria: $11_{10} = 1011_2 = 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$;
(2) Sostituire 11 con la sua rappresentazione binaria: $7^{11} = 7^{(8+2+1)} = 7^8 \cdot 7^2 \cdot 7$;
(3) Scriviamo i bit di 11 in colonna, al contrario (a partire dal LSB): $\begin{bmatrix}1 & 1 & 0 & 1\end{bmatrix}$;
(4) Scriviamo accanto a ciascun bit la potenza di 7 corrispondente, ottenuta dalla scomposizione al punto (1);
(5) Scriviamo accanto alle potenze 7^i il risultato dell'operazione $7^i \bmod 26$;
(6) Componiamo il risultato prendendo il prodotto dei valori scritti al punto (5) in corrispondenza di un 1, scritto al punto (3)

Otterremo infine:

$$\begin{array}{ccc} \mathbf{1} & 7^1 & \boxed{7} \\ \mathbf{1} & 7^2 & \boxed{23} \\ 0 & 7^4 & 9 \\ \mathbf{1} & 7^8 & \boxed{3} \end{array} \implies (7 \cdot 23 \cdot 3) \bmod 26 = 15$$

Il vantaggio di questo metodo è il fatto che si basa solo sull'operazione di elevamento al quadrato di numeri piccoli (relativamente al modulo, nell'esercizio siamo in $\mathbb{Z}_{26}$). In totale, saranno necessari (per questo caso dell'esercizio) $\lceil \log_2 (11) \rceil = 4$ passi. □

1.7. Teorema piccolo di Fermat

Teorema piccolo di Fermat

TEOREMA 1.3. *Sia p un intero primo, e a un intero tale che p non divida a, allora è vero che*

$$a^{p-1} \equiv 1 \,(\mathrm{mod}\, p) \tag{1.7.1}$$

Non vale l'implicazione inversa, infatti prendendo a ed n due interi qualsiasi, e osservando che $a^{n-1} \equiv 1 \,(\mathrm{mod}\, n)$, non possiamo dedurre che n sia primo.

Piuttosto possiamo usare il Teorema 1.3 per provare che un numero sia composto (non primo); tuttavia i numeri composti che danno resto 1 non ostante non siano primi sono pochi, e sono definiti *pseudo-primi*. Questi numeri sono sempre più rarefatti all'aumentare dell'ordine di grandezza.

Non ostante la scelta di una base a differente possa "smascherare" un numero pseudo-primo, esiste una categoria di interi chiamati *pseudo-primi assoluti*, i quali forniscono resto 1 con qualunque base scelta nel test del teorema piccolo di Fermat.

OSSERVAZIONE 1.3. Abbiamo concluso dal Teorema 1.3 che, se non è vero $p \backslash a$, allora vale $a^{p-1} \equiv 1 \pmod{p}$; possiamo scrivere questa congruenza in modo equivalente come $a \cdot a^{p-2} \equiv 1 \pmod{p}$, che può essere risolta con la regola della divisione nel modo seguente:

$$a^{p-2} \equiv a^{-1} \pmod{p} \tag{1.7.2}$$

Abbiamo appena ottenuto un nuovo metodo per calcolare l'inverso di un intero a.

Poniamoci nel caso generale, in cui abbiamo un intero qualunque a e un intero composto n: la condizione del teorema diventa che $a \perp n$; in queste condizioni possiamo enunciare il seguente Teorema 1.4.

TEOREMA 1.4. *Sia a un intero qualunque e n un intero composto, tali che $a \perp n$, allora è vero che*

Teorema di Eulero

$$a^{\varphi(n)} \equiv 1 \pmod{n} \tag{1.7.3}$$

La funzione *toziente* $\varphi(p)$ con p numero primo, è il numero di primi relativi rispetto a p, in questo caso $p-1$ numeri, tutti i predecessori di p, il quale essendo primo ha come fattore comune con essi solo 1; vale dunque $\varphi(p) = p-1$ per p numero primo.
In generale il toziente di un intero n qualunque si può calcolare in uno dei seguenti modi:

- $\varphi(n) = n \cdot \prod_{i \in [1,n-1],\, p_i \backslash n} \left(1 - \frac{1}{p_i}\right)$
- $n = \prod_{i=1}^{m} p_i^{r_i} \implies \varphi(n) = \prod_{i=1}^{m} \left(p_i^{r_i} - p_i^{r_i - 1}\right)$

Toziente

Vediamo un esempio in cui calcoliamo il toziente di due numeri adoperando entrambi i procedimenti:

ESEMPIO 1.10. *Calcolare il toziente di 1000.*

✓Adoperando il primo metodo, scriviamo la scomposizione di 1000 nei suoi fattori primi: $1000 = 2^3 \cdot 5^3$; scriviamo dunque la produttoria usando i due fattori $p_i = \{2, 5\}$ senza esponente

$$1000 \cdot \left(1 - \frac{1}{2}\right) \cdot \left(1 - \frac{1}{5}\right) = \boxed{400} = \varphi(1000)$$

Se adoperiamo il secondo metodo, scriviamo direttamente la produttoria a partire dai fattori primi 2^3 e 5^3:

$$\left(2^3 - 2^2\right) \cdot \left(5^3 - 5^2\right) = \boxed{400} = \varphi(1000)$$

Abbiamo ottenuto lo stesso toziente, mostrando che entrambi i procedimenti si equivalgono. □

OSSERVAZIONE 1.4. Se vogliamo calcolare l'inverso di un intero a modulo n non primo, posto che $a \perp n$ possiamo scrivere la congruenza

$$a^{-1} \equiv a^{\varphi(n)-1} \pmod{n} \tag{1.7.4}$$

Questa relazione ci permette di calcolare un inverso utilizzando il modulo di un elevamento a potenza (evitando la strada di Euclide esteso (1.3.4)), per esempio tramite square and multiply eseguito su un calcolatore.

Mettendo assieme quanto illustrato in questa sezione, possiamo rispondere alla richiesta mostrata nel seguente esempio.

ESEMPIO 1.11. *Calcolare le ultime tre cifre di* 7^{803}.

✓Le ultime tre cifre di un numero possono essere ottenute usando il resto della sua divisione per 10^3; allora dobbiamo trovare il risultato di $7^{803} \bmod 1000$.

Per il teorema di Eulero (1.7.3) sappiamo che $7^{\varphi(1000)} \equiv 1 \pmod{1000}$ e dal precedente esempio abbiamo già calcolato $\varphi(1000) = 400$, per cui possiamo scomporre 7^{803} nel modo seguente:

$$\left(7^{400} \cdot 7^{400} \cdot 7^3\right) \bmod 1000 = (1 \cdot 1 \cdot 343) \bmod 1000 = \boxed{343}$$

□

1.8. Principio fondamentale

Se abbiamo gli interi a, b, n, x, y tali che $a \perp n$ e $n > 0$, allora se $x \equiv y \pmod{\varphi(n)}$ vale

$$a^x \equiv b^y \pmod{n} \tag{1.8.1}$$

tuttavia non vale l'implicazione inversa.

Concludiamo che, in una congruenza modulo n, in presenza di elevamenti a potenza gli esponenti della stessa base lavorano in modulo di $\varphi(n)$.

Questo principio permette di risolvere rapidamente la richiesta dell'Esercizio 1.11: dato che $803 \equiv 3 \pmod{\varphi(1000) = 400}$ otteniamo direttamente $7^{803} \equiv 7^3 \pmod{1000}$; abbiamo usato (1.8.1) in cui $x = 803$ e $y = 3$.

1.9. Radici in aritmetica modulare

1.9.1. Radici primitive. Introduciamo il concetto di gruppo algebrico:

Gruppo algebrico

Chiamiamo gruppo *una struttura formata da un insieme e da un'operazione binaria definita su di esso, la quale soddisfi gli assiomi di associatività, esistenza dell'elemento neutro ed esistenza dell'elemento inverso*

Ad esempio, se prendiamo l'insieme dei numeri interi $\mathbb{Z}$ e l'operazione di somma $+$, possiamo formare il gruppo $(\mathbb{Z}, +)$ poiché la somma è associativa, l'elemento neutro è lo zero e l'inverso di qualunque elemento è sempre definito; anche prendendo $\mathbb{Z}_p^*$ (insieme dei residui interi modulo p, escluso lo 0) e l'operazione di prodotto, otteniamo di nuovo un gruppo $\left(\mathbb{Z}_p^*, \cdot\right)$. Non si ottiene un gruppo rispetto al prodotto per $\mathbb{Z}_p$ (residui modulo p, da 0 a $p-1$), poiché lo 0 non ha un inverso definito.

Se prendiamo un elemento $a \in \mathbb{Z}_p^*$ allora vale $a^{p-1} \equiv 1 \pmod{p}$ per il Teorema 1.3; chiamiamo inoltre ordine dell'elemento a (e lo indichiamo tramite $\text{ORD}(a)$) l'intero $n > 0$ più piccolo tale che

$$a^n \equiv 1 \pmod{p} \tag{1.9.1}$$

Il teorema piccolo di Fermat (1.7.1) ci assicura che vale sempre $n \leq p-1$, ma non è detto che sia proprio $n = p-1$. Chiamiamo quindi α l'elemento primitivo $a \in \mathbb{Z}_p^*$ se e solo se l'ordine di a è $p-1$, ovvero non esiste nessun altro intero n per cui $a^n \equiv 1 \pmod{p}$; in altri termini vale

$$a^n \equiv 1 \pmod{p} \iff n = p-1 \tag{1.9.2}$$

Se l'ordine di a fosse proprio $p-1$, allora preso α^i l'elemento primitivo, presi tutti gli interi $i \in [1, p-1]$, otteniamo dalla successione degli α^i tutti e soli gli elementi dell'insieme $\mathbb{Z}_p^*$ (in ordine anche differente); chiameremo l'elemento α la *radice primitiva*.

Radice primitiva

Chiamiamo α una radice primitiva per p se vale (1.9.2) e inoltre l'ordine di α è pari a $n = p - 1$.

ESEMPIO 1.12. *Trovare se $\alpha = 3$ sia una radice primitiva di $p = 7$.*

✓Per trovare se α sia effettivamente una radice primitiva di p, calcoliamo gli elementi:

$$\left\{\forall i \in [1,\, p-1] \,:\, \beta \equiv \alpha^i \,(\mathrm{mod}\, p)\right\}$$

Otteniamo quindi:

$$\begin{gathered} 3^1 \equiv 3 \,(\mathrm{mod}\, 7) \\ 3^2 \equiv 2 \,(\mathrm{mod}\, 7) \\ 3^3 \equiv 6 \,(\mathrm{mod}\, 7) \\ 3^4 \equiv 4 \,(\mathrm{mod}\, 7) \\ 3^5 \equiv 5 \,(\mathrm{mod}\, 7) \\ 3^6 \equiv 1 \,(\mathrm{mod}\, 7) \\ \vdots \end{gathered}$$

Abbiamo riottenuto tutti e soli gli elementi da 1 a $p - 1 = 6$ (i membri di $\mathbb{Z}_7^*$), con periodo $p - 1 = 6$. Si verifica dunque la condizione che rende un elemento primitivo α una radice per p; la radice primitiva è anche detta elemento generatore: infatti, tramite essa è possibile ottenere tutti i residui in modulo p di un certo insieme $\mathbb{Z}_p^*$. □

ESEMPIO 1.13. *Trovare se $\alpha = 2$ sia una radice primitiva di $p = 7$.*

✓Calcolando di nuovo gli α^i con $i \in [1,\, p-1]$ otteniamo

$$\begin{gathered} 2^1 \equiv 2 \,(\mathrm{mod}\, 7) \\ 2^2 \equiv 4 \,(\mathrm{mod}\, 7) \\ 2^3 \equiv 1 \,(\mathrm{mod}\, 7) \\ \vdots \end{gathered}$$

In questo caso abbiamo ottenuto periodo 3; inoltre l'ordine di α è $n = 3 \neq p - 1 = 6$, quindi $\alpha = 2$ non è radice primitiva di $p = 7$. □

OSSERVAZIONE 1.5. Se prendiamo l'elemento primitivo $a \in \mathbb{Z}_p^*$, esso avrà al massimo ordine $p - 1$; tuttavia potrà presentare anche ordini pari ai sottomultipli di $p - 1$, ma non ordini differenti da essi.

OSSERVAZIONE 1.6. Se α è un elemento primitivo di $\mathbb{Z}_p^*$ e vale $\beta \equiv \alpha^i \,(\mathrm{mod}\, p)$ con $1 \leq i \leq p - 1$, questi β sono tutti e soli gli elementi di $\mathbb{Z}_p^*$. Se ora prendiamo un elemento primitivo di $\mathbb{Z}_p^*$ espresso come β (tramite la precedente congruenza), il suo ordine sarà dipendente dall'esponente i usato per ottenerlo a partire da α, e dovrà valere

$$\mathrm{ORD}\,(\beta) = \frac{p-1}{\mathrm{MCD}\,(p-1,\, i)} \tag{1.9.3}$$

dove i è l'esponente a cui elevare α per ottenere β. Questo si verifica perché i β sono potenze di α, ed essendo $\mathrm{ORD}\,(\alpha) = p - 1$ (α è radice primitiva) necessariamente l'ordine di β sarà un sottomultiplo di $p - 1$, determinato da (1.9.3)

Gli elementi primitivi di $\mathbb{Z}_p^*$ sono in numero $\varphi\,(p-1)$; infatti saranno tutti gli elementi di $\mathbb{Z}_p^*$. coprimi rispetto a $p - 1$, per cui si verifica la condizione (1.9.3). Notiamo infatti che β è una radice primitiva di p (vale $\mathrm{ORD}\,(\beta) = p - 1$) quando si ha $\mathrm{MCD}\,(p-1,\, i) = 1$; concludiamo che il numero di elementi primitivi β di un insieme $\mathbb{Z}_p^*$ è il numero di elementi coprimi rispetto a $p - 1$ (valore ottenuto dalla funzione toziente di $p - 1$).

1.9.2. Test di primitività. Prendiamo un $\alpha \in \mathbb{Z}_p^*$ con p qualunque, e cerchiamo se α sia primitivo o meno rispetto a p; usiamo il seguente test di primitività: scomporre $p-1$ nei suoi fattori primi, in modo da avere una produttoria di quozienti elevati a un certo esponente $p-1 = \prod_i q_i^{r_i}$; diremo che α è primitivo rispetto a p se e solo se vale

Test primitività

$$\forall i \,:\, \alpha^{(p-1)/q_i} \not\equiv 1 \,(\mathrm{mod}\, p) \tag{1.9.4}$$

Si noti che p è primo (per definizione), di conseguenza $p-1$ è sempre pari.

ESEMPIO 1.14. *Ottenere se $\alpha = 2$ sia radice primitiva per $p = 19$.*

✓Usando il test (1.9.4), scomponiamo $19 - 1 = 18$ nei suoi fattori primi:

$$18 = 2 \cdot 3^2$$

Ora testiamo la congruenza per tutti i fattori primi (presi senza esponente):

$$2^{18/3} = 2^6 = 64 \equiv 7 \,(\mathrm{mod}\, 19)$$
$$2^{18/2} = 2^9 = 512 \equiv 18 \,(\mathrm{mod}\, 19)$$

Per nessun fattore primo si ha congruenza a 1 modulo 19, dunque $\alpha = 2$ è davvero una radice primitiva per $p = 19$.

Osserviamo infine che gli elementi primitivi di $\mathbb{Z}_{19}$ sono $\varphi(18) = 6$, distribuiti all'interno dell'insieme in modo non predicibile. □

1.9.3. Radici quadrate. Poniamoci nel caso di radici quadrate in modulo a un intero p primo; inoltre, consideriamo per tale intero solamente la metà dei numeri primi (si ricordi la separazione dei primi in classi di congruenza (1.2.1)):

$$p \equiv 3 \,(\mathrm{mod}\, 4) \tag{1.9.5}$$

Risolviamo, sotto queste condizioni, l'equazione $x^2 \equiv a \,(\mathrm{mod}\, p)$, ovvero cerchiamo $\sqrt{a} \in \mathbb{Z}_p$; si dimostra che, se esiste $\sqrt{a}$, allora vale:

Radice quadrata

$$\pm\, a^{(p+1)/4} = \sqrt{a} \;\vee\; \pm a^{(p+1)/4} = \sqrt{-a} \tag{1.9.6}$$

dove abbiamo l'unione di due condizioni che sono esclusive (se non esiste la radice di a allora esisterà quella di $-a$, e vale quanto indicato in (1.9.6), e vice versa).

OSSERVAZIONE 1.7. Se avessimo $p \not\equiv 3 \,(\mathrm{mod}\, 4)$ ma $p \equiv 1 \,(\mathrm{mod}\, 4)$ (p appartenente all'altra classe di congruenze), allora nel caso in cui non esista una delle due radici ($\sqrt{a}$ o $\sqrt{-a}$) non esiste nemmeno l'altra.

ESEMPIO 1.15. *Si calcoli, se esiste, la radice di 5 in modulo 11.*

✓Cercare $\sqrt{5} \,\mathrm{mod}\, 11$ equivale a risolvere l'equazione

$$x^2 \equiv 5 \,(\mathrm{mod}\, 11)$$

Dato che per 11 vale (1.9.5), possiamo usare la formula (1.9.6):

$$\pm 5^{(11+1)/4} = \pm 125 \,\mathrm{mod}\, 11 = \pm 4$$

Per determinare se in modulo 11 i numeri ± 4 siano la radice di 5 o di -5, effettuiamo l'elevamento a potenza della radice cercata:

$$4^2 \,\mathrm{mod}\, 11 = 5, \quad -4^2 = 7^2 \,\mathrm{mod}\, 11 = 5$$

Abbiamo ottenuto che la radice quadrata di 5 modulo 11 è ± 4. □

ESEMPIO 1.16. *Si calcoli, se esiste, la radice di 2 in modulo 11.*

✓Come prima, impostiamo così l'equazione da risolvere:

$$x^2 \equiv 2 \pmod{11}$$

Possiamo usare (1.9.6), dato che per 2 vale (1.9.5):

$$\pm 2^{(11+1)/4} = \pm 8 \bmod 11 = \pm 8$$

Infine effettuiamo l'elevamento a potenza della radice per ottenere il segno:

$$8^2 \bmod 11 = 9 \bmod 11 = -2$$

Abbiamo ottenuto che in modulo 11 non esiste la radice quadrata di 2; la radice di -2 però esiste, e vale ± 8. □

1.9.4. Test per segno della radice. Prendiamo un primo p dispari, un $a \neq 0$ in modulo p, allora

$$a^{(p-1)/2} \equiv \pm 1 \pmod{p} \tag{1.9.7}$$

Distinguiamo i due casi:

- se il segno in (1.9.7) è $+$, allora esiste $\sqrt{a}$ in modulo p;
- se il segno in (1.9.7) è $-$, allora esiste $\sqrt{-a}$ in modulo p.

Ricordando l'Osservazione 1.7, si nota che il test appena mostrato permette di affermare se la radice non sia definita, nel caso in cui $p \equiv 1 \pmod 4$ e in cui il segno in (1.9.7) sia $-$.

1.9.5. Radice quadrata modulo composto. Esaminiamo il problema $x^2 \equiv a \pmod n$ dove n sia un numero composto; dato che n sarà scomponibile in fattori primi, possiamo scomporre la congruenza esaminata in due congruenze più semplici in modulo p e q, sapendo che $n = p \cdot q$. Le due soluzioni ottenute si combinano infine tramite il Teorema 1.2.

Questo problema ha la sua difficoltà nella fattorizzazione di n, al punto che la complessità computazionale del calcolo della radice modulo n equivale a quella della sua scomposizione in fattori primi.

Esempio 1.17. *Si calcoli, se esiste, la radice di 71 in modulo 77.*

✓Notiamo subito che $77 = 7 \cdot 11$; per il Teorema 1.2 possiamo scomporre la congruenza in esame nelle due più semplici:

$$x^2 \equiv 71 \pmod{77} \implies \begin{cases} x^2 \equiv 1 \pmod 7 \\ x^2 \equiv 5 \pmod{11} \end{cases}$$

Nella precedente abbiamo già sostituito $71 \bmod 7 = 1$ e $71 \bmod 11 = 5$; possiamo risolvere le due congruenze in modo indipendente:

(1) la radice quadrata di 1 è comunque ± 1, modulo 7;
(2) la radice quadrata di 5 modulo 11 si può ottenere da (1.9.6) e risulta $x \equiv \pm 5^{(11+1)/4} \pmod{11} \implies x \equiv \pm 4 \pmod{11}$.

Abbiamo ottenuto due radici da ciascuna congruenza, che possono essere combinate in $2^2 = 4$ modi possibili, per costruire la soluzione tramite il Teorema 1.2.

- $\{x \equiv 1 \pmod 7 \,;\; x \equiv 4 \pmod{11}\}$
  Risolviamo la combinazione delle due congruenze analizzate:

$$\begin{gathered} x = 1 + k \cdot 7 \equiv 4 \pmod{11} \\ 7 \cdot k \equiv 3 \pmod{11} \\ k \equiv 3 \cdot 7^{-1} \pmod{11} \end{gathered}$$

Per trovare l'inverso utilizziamo (1.7.2), da cui $7^{-1} \bmod 11 = 7^9 \bmod 11 = 8$, da cui segue che $k = 24 \bmod 11 = 2$; sostituendo nella prima si ottiene

$$x = 14 + 1 \equiv 4 \,(\mathrm{mod}\, 11) \implies \boxed{x \equiv 15 \,(\mathrm{mod}\, 77)}$$

dal Teorema 1.2.

- $\{x \equiv 1 \,(\mathrm{mod}\, 7)\,;\ x \equiv -4 \,(\mathrm{mod}\, 11)\}$

$$x \equiv -15 \,(\mathrm{mod}\, 77)$$

- $\{x \equiv -1 \,(\mathrm{mod}\, 7)\,;\ x \equiv 4 \,(\mathrm{mod}\, 11)\}$

$$x \equiv 29 \,(\mathrm{mod}\, 77)$$

- $\{x \equiv -1 \,(\mathrm{mod}\, 7)\,;\ x \equiv -4 \,(\mathrm{mod}\, 11)\}$

$$x \equiv -29 \,(\mathrm{mod}\, 77)$$

Non è detto che le radici siano sempre 4, in questo caso esistevano tutte le radici per ciascuna congruenza in cui è stata scomposta quella iniziale. □

1.9.6. Residui quadratici. Abbiamo chiamato gli elementi di $\mathbb{Z}_p^*$ residui modulo p; i residui quadratici sono residui che corrispondono anche al quadrato di qualche elemento dello stesso insieme $\mathbb{Z}_p^*$.
Chiamiamo $a_q \in \mathbb{Z}_p^*$ i residui quadratici per cui valga

$$a_q \equiv (\pm b)^2 \,(\mathrm{mod}\, p)$$

con $b \in \mathbb{Z}_p^*$ un elemento dell'insieme a cui appartiene anche a_q.

Osserviamo che $b \in \left\{1, 2, \ldots, \frac{p-1}{2}\right\}$, poiché i residui a partire da $\frac{p-1}{2} + 1$ sono i residui negativi dei precedenti dal minore al maggiore ($-1 = p - 1$).

Esempio 1.18. *Trovare i residui quadratici dell'insieme* $\mathbb{Z}_{11}^*$.

✓Otteniamo subito che nell'insieme analizzato ci sono $\frac{p-1}{2} = \frac{11-1}{2} = 5$ residui quadratici; possiamo ottenerli tramite il test (1.9.7):

$$a^{(p-1)/2} \not\equiv +1 \,(\mathrm{mod}\, p) \implies a \textbf{ non } \text{è residuo quadratico!}$$

Effettuiamo questo test per ciascun elemento di $\mathbb{Z}_{11}^*$:

$$\begin{array}{ll} 1^5 \equiv \boxed{1} \,(\mathrm{mod}\, 11) & 6^5 \equiv -1 \,(\mathrm{mod}\, 11) \\ 2^5 \equiv -1 \,(\mathrm{mod}\, 11) & 7^5 \equiv -1 \,(\mathrm{mod}\, 11) \\ 3^5 \equiv \boxed{1} \,(\mathrm{mod}\, 11) & 8^5 \equiv -1 \,(\mathrm{mod}\, 11) \\ 4^5 \equiv \boxed{1} \,(\mathrm{mod}\, 11) & 9^5 \equiv \boxed{1} \,(\mathrm{mod}\, 11) \\ 5^5 \equiv \boxed{1} \,(\mathrm{mod}\, 11) & 10^5 \equiv -1 \,(\mathrm{mod}\, 11) \end{array}$$

Raccogliendo i risultati dei test con la congruenza a $+1$, possiamo scrivere l'insieme dei residui quadratici di $\mathbb{Z}_{11}^*$:

$$a_q = \{1, 3, 4, 5, 9\}$$

□

Osservazione 1.8. Un modo più rapido per ottenere i residui quadratici consiste nel sfruttare il fatto che essi saranno i quadrati dei primi $\frac{p-1}{2}$ elementi dell'insieme dei residui:

$$a_q = \left\{ 1^2, \ldots, \left(\frac{p-1}{2} \right)^2 \right\}$$

Riprendendo la consegna dell'Esempio 1.18, i residui quadratici di $\mathbb{Z}_{11}^*$ si ottengono come:

$$\begin{array}{ll} 1^2 \bmod 11 = 1 & 3^2 \bmod 11 = 9 \quad 5^2 \bmod 11 = 3 \\ 2^2 \bmod 11 = 4 & 4^2 \bmod 11 = 5 \end{array} \implies a_q = \{1, 4, 9, 5, 3\}$$

Otteniamo gli stessi residui quadratici trovati con i test, in ordine diverso.

CAPITOLO 2

Cifrari elementari

2.1. Introduzione

Facendo riferimento alla figura A.0.1, possiamo individuare la caratteristica del sistema di cifratura modellizzato: esso usa la stessa chiave k per la cifratura e la decifratura del messaggio; ammettiamo che essa sia consegnata al destinatario tramite un canale sicuro.

Il testo in chiaro è definito all'interno di tutti i possibili messaggi in chiaro componibili con l'alfabeto a disposizione ($p \in \mathscr{P}$), il messaggio cifrato è anch'esso definito all'interno di un insieme di ogni possibile messaggio cifrato componibile ($c \in \mathscr{C}$), infine la chiave appartiene all'insieme di tutte le possibili chiavi componibili ($k \in \mathscr{K}$).

Ogni chiave, all'interno dello spazio delle chiavi, è definita una regola di cifratura ($\forall k \in \mathscr{K} \ : \ E_k \in \mathscr{E}$) e anche una regola di decifratura ($\forall k \in \mathscr{K} \ : \ D_k \in \mathscr{D}$), entrambe dipendenti da k.

Deve valere che la cifratura sia invertibile con la decifratura, ovvero $\forall k \in \mathscr{K}$, $p \in \mathscr{P}$, $c \in \mathscr{C} \ : \ c = E_k(p) \wedge p = D_k(c)$; in caso contrario la decifratura non sarebbe più possibile (collisioni durante la cifratura di messaggi differenti).

2.2. Cifrari a scorrimento e sostituzione

2.2.1. Cifrario di Cesare. Il cifrario di Cesare utilizza uno scorrimento dell'alfabeto, la chiave stessa è un simbolo dell'alfabeto:

$$\mathscr{P} = \mathscr{C} = \mathscr{K} = \mathbb{Z}_n$$

Si verifica nel caso dell'alfabeto inglese vale $n = 26$; la funzione di cifratura e di decifratura si indicano come:

$$c = E_k(p) = (p + k) \bmod n$$
$$p = D_k(c) = (c - k) \bmod n$$

Possiamo attaccare questo cifrario con una ricerca esaustiva dello spazio delle chiavi, sapendo che $|\mathscr{K}| = n$, oppure possiamo confrontare un testo in chiaro noto con un testo cifrato per ottenere la chiave ($k = p - c$).

Chiamiamo questo tipo di cifrario *mono-alfabetico*, ovvero ad ogni carattere corrisponde un carattere.

2.2.2. Cifrario affine. Utilizza una combinazione lineare applicata al testo in chiaro; per questo cifrario vale:

$$\mathscr{P} = \mathscr{C} = \mathbb{Z}_n,\ \mathscr{K} = k(a,\, b)$$

La funzione di cifratura e di decifratura si indicano come:

$$c = E_k(p) = (ap + b) \bmod n$$
$$p = D_k(c) = (c - b)\, a^{-1} \bmod n$$

Bisogna prendere $a,\ b \in \mathbb{Z}_n$ e per la presenza dell'inversione di a, è necessario che esso sia primo relativo a n ($a \perp n$); se così non fosse, si verificherebbero delle collisioni nella cifratura.

2.2.3. Cifrario a sostituzione. Ogni carattere è sostituito da un'altro, secondo una tabella arbitraria; esso è la generalizzazione dei due precedenti cifrari (la chiave è una permutazione dell'alfabeto), e vale

$$\mathscr{P} = \mathscr{C} = \mathbb{Z}_n,\ |\mathscr{K}| = n!$$

La chiave è la tabella di sostituzione, e le funzioni di cifratura e decifratura sono determinate dalle sostituzioni indicate nella tabella.

Possiamo attaccare questo cifrario (e tutti gli altri cifrari mono-alfabetici) analizzando l'entropia dei singoli simboli dell'alfabeto: nel caso di messaggi cifrati in linguaggio naturale, l'analisi delle frequenze dei simboli può rivelare lettere molto usate.

Altri attacchi di questo tipo includono l'analisi della frequenza dei digrammi e trigrammi (coppie e triple di lettere).

2.2.4. Cifrario di Vigenère. Questo cifrario lavora su blocchi di h caratteri alla volta, e per questo viene detto poli-alfabetico. In questo caso abbiamo:

$$\mathscr{P} = \mathscr{C} = \mathscr{K} = (\mathbb{Z}_n)^h$$

La chiave, il testo in chiaro e il testo cifrato saranno vettori di h elementi:

$$k = (k_1, \ldots, k_h)\,,\ p = (p_1, \ldots, p_h)\,,\ c = (c_1, \ldots, c_h)$$

Le regole di cifratura e decifratura sono le stesse del cifrario di Cesare, applicate su ciascun elemento del vettore p o c:

$$\begin{aligned} c &= E_k(p) = (p_1 + k_1 \bmod n, \ldots, p_h + k_h \bmod n) \\ p &= D_k(c) = (c_1 - k_1 \bmod n, \ldots, c_h - k_h \bmod n) \end{aligned}$$

In questo cifrario gli scorrimenti per ciascun elemento del blocco dipendono dalla posizione al suo interno; la stessa lettera verrà cifrata in modo diverso a seconda della sua posizione nel blocco; l'analisi di frequenza si applica, suddividendo la frequenza di un simbolo su h possibili modi di cifrarlo (la difficoltà cresce all'aumentare di h).

2.3. Cifrario a permutazione

Si tratta di un cifrario a blocco (lavora su h simboli alla volta), che permuta le posizioni dei caratteri in ogni blocco. Al contrario del cifrario a sostituzione, la permutazione in ogni blocco è indipendente. La chiave adottata è una permutazione delle posizioni all'interno del blocco.

Esempio 2.1. *Dato il cifrario a permutazione con blocchi di dimensione $n = 6$ e permutazione $k = (3, 5, 1, 6, 4, 2)$, cifrare il messaggio $p =$ "she sells sea shells" (gli spazi tra le parole non sono caratteri dell'alfabeto).*

✓Identifichiamo i blocchi di $n = 6$ caratteri; il primo blocco da sinistra è $\mathbb{B}_1 =$ "shesel", che possiamo indicare come:

$$p = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ \text{s}, & \text{h}, & \text{e}, & \text{s}, & \text{e}, & \text{l} \end{pmatrix}$$

A questo punto applichiamo la permutazione k, che sposta i simboli del blocco nella seguente configurazione:

$$c = \begin{pmatrix} 3 & 5 & 1 & 6 & 4 & 2 \\ \text{e}, & \text{e}, & \text{s}, & \text{l}, & \text{s}, & \text{h} \end{pmatrix}$$

□

Questo cifrario risulta poli-alfabetico, e ogni blocco di lunghezza n può essere cifrato in $n!$ modi differenti.

2.4. Proprietà fondamentali dei cifrari a blocchi

In un cifrario a blocco di lunghezza n, si prende un messaggio in chiaro e a partire da esso si ottiene un messaggio cifrato, entrambi della stessa lunghezza ($|p| = |c| = n$). Nell'articolo[1] sulla crittografia pubblicato da Shannon, egli espone due proprietà fondamentali che ogni sistema crittografico dovrebbe avere:

Diffusione: Si ha diffusione perfetta se, cambiando 1 bit del blocco in chiaro, tutti i bit del blocco cifrato corrispondente cambiano in maniera apparentemente casuale, con probabilità 1/2. La perfetta diffusione vale anche nel senso opposto: cambiando 1 bit del blocco cifrato, tutti i bit del blocco in chiaro corrispondente cambiano con probabilità 1/2.

Confusione: La chiave deve avere effetto su tutti i bit del blocco cifrato: a parità di blocco in chiaro, se cambia 1 bit della chiave, tutti i bit del blocco cifrato cambieranno, con probabilità 1/2.

Queste due proprietà si riassumono nella seguente affermazione:

Proposizione 2.1. *Non solo ogni bit del blocco in chiaro hanno effetto su tutti i bit della chiave, ma anche tutti i bit della chiave hanno effetto su ogni bit del blocco in chiaro.*

[1] *"Communication Theory of Secrecy Systems"* (1949), Bell System Technical Journal

CAPITOLO 3

Cifrario DES

3.1. Introduzione

3.1.1. Nascita del DES come standard. L'algoritmo Data Encryption Standard (DES) nasce negli anni '70 per soddisfare la necessità degli Stati Uniti di usare un cifrario per le comunicazioni segrete; allora l'NBS (attuale NIST) pubblica un bando, a cui risponde IBM con l'algoritmo LUCIFER.

L'algoritmo di IBM conteneva delle tabelle di sostituzione, che furono modificate dall'NBS: questo portò a pensare, per molti anni, che il governo avesse introdotto una trapdoor nelle tabelle di sostituzione, tuttavia non fu mai trovata.

Infine il DES fu pubblicato come standard nel '77; col progredire della potenza di calcolo nei calcolatori, questo algoritmo è divenuto suscettibile a diversi attacchi, così sono state introdotte delle sue varianti (compatibilmente con l'hardware legacy), come il triplo DES (3DES).

Di per sé l'algoritmo DES è robusto, tuttavia la sua chiave di 56bit è relativamente corta ($|\mathscr{K}| = 2^{56} \simeq 2^6 \cdot 10^{15}$), e questo permette di effettuare in breve tempo un'analisi esaustiva dello spazio delle chiavi. Oltre a questo, DES sarebbe stato vulnerabile ad attacchi di crittoanalisi differenziale (si cerca un errore nella realizzazione della proprietà di perfetta diffusione, prendendo un blocco di testo in chiaro e cifrandolo, cambiando un bit alla volta e cercando evidenza statistica all'interno del blocco cifrato) se il suo algoritmo avesse avuto 14 *round*; tuttavia la specifica di IBM prevede 16 *round* (si veda 2.1).

DES è un cifrario a blocchi di 64bit, che nel modo più semplice lavora prendendo un blocco, cifrandolo e poi ripetendo il processo in modo indipendente per il blocco successivo (modo *ECB*).

3.1.2. Schema Feistel. Descriviamo l'idea dietro a questo schema con la procedura da implementare per realizzarlo:

- prendiamo un blocco di bit B_1, e dividiamolo in due metà, in modo da avere la sinistra L_1 e la destra R_1;
- usiamo la chiave K per generare una sotto-chiave di round K_i;
- permutiamo le due metà al round i-esimo, facendo passare la metà di sinistra attraverso una funzione fortemente non lineare che utilizzi la chiave: $B_i = R_{i-1} \parallel L_{i-1} \oplus f(R_{i-1}, K_i)$;
- se vi sono n round in totale, non si effettua la permutazione di L_i ed R_i al round n-esimo, e si ottiene $B_n = R_n \parallel L_n$

La notazione $A \parallel B$ indica la concatenazione di B dopo di A

Possiamo presumere che, continuando a permutare le due metà del blocco, grazie alla funzione $f()$, vi sarà un'ottima diffusione dei bit in ingresso; inoltre la confusione sarà garantita dalla presenza delle chiavi di round K_i, generate a partire dalla chiave iniziale.

Lo schema Feistel permette di decifrare con la stessa procedura della cifratura, utilizzando B_n e K_n come ingresso, e ottenendo B_1 alla fine degli n round.

3.2. Algoritmo DES

Il DES applica 16 volte (round) il blocco di Feistel, utilizzando la seguente procedura:

(1) si applica una permutazione iniziale (IP) fissata al blocco di 64bit in chiaro;
(2) si prende la chiave DES (da 64bit), si rimuovono gli 8bit di parità e si utilizzano i 56bit per produrre le successive chiavi di round K_i;
   (a) si permutano i bit della chiave da 56bit, poi essa viene divisa in due blocchi da 28bit, rispettivamente C_0 e D_0;
   (b) si produce la chiave di round secondo una tabella di shift fissa (funzione LS_i), che fa scorrere i bit delle due metà della chiave (C_i, D_i) in modo circolare di 1 o 2 posizioni, a seconda del numero i del round: $C_i = LS_i(C_{i-1})$, $D_i = LS_i(D_{i-1}) \implies K_i = C_i \parallel D_i$;
   (c) tramite una tabella di riduzione, si estraggono dalla chiave di round 48bit, da usare nella funzione $f()$.
(3) si prende la chiave di round K_i e il blocco sinistro di round R_{i-1} e si genera il blocco destro L_i tramite la funzione $f()$
   (a) il blocco R_{i-1} in ingresso viene espanso a 48bit tramite una tabella di espansione;
   (b) si somma modulo due la chiave di round al blocco espanso: $E(R_{i-1}) \oplus K_i$;
   (c) si divide il risultato da 48bit in 8 gruppi da 6bit;
   (d) ciascun gruppo viene modificato tramite una S-Box (8 in totale), che prende un blocco da 6bit e lo riduce a 4bit;
   (e) gli 8 gruppi da 4bit sono uniti in un blocco da 32bit, che viene permutato;
   (f) il blocco ottenuto viene infine usato assieme alla chiave nella funzione $f()$.

Initial Permutation (IP). Essa lavora su un blocco di 64bit, e produce un'altro blocco altrettanto lungo. Osserviamo che la permutazione iniziale, nota e fissa, non migliora diffusione o confusione; alcune ipotesi sostengono che poteva essere un modo per inizializzare l'hardware dell'epoca ('70).
La seguente tabella di look-up realizza la permutazione iniziale del DES:

Funzione f. Essa lavora su blocchi di 32bit, e garantisce ottima diffusione grazie alle permutazioni effettuate sui blocchi di bit.

S-Box. Tabella di sostituzione che prende un blocco da 6bit, e usando il primo e l'ultimo per indicare la riga e i 4 centrali per indicare la colonna, si ottiene un blocco da 4bit. Infatti, la tabella dell'S-Box contiene valori di massimo 4bit, che vengono indirizzati nel modo seguente: se $B_h = \overset{R_0}{0}\underset{C_0}{0}\underset{C_1}{1}\underset{C_2}{0}\underset{C_3}{0}\overset{R_1}{1}$, allora il valore ottenuto sarà l'elemento alla riga $01_2 = 1_{10}$ e alla colonna $0100_2 = 4_{10}$ della S-Box h (righe e colonne sono numerate a partire da 0); queste tabelle sono state progettate per realizzare una trasformazione non lineare sui blocchi.

Chiave di round. Ogni bit della chiave sarà utilizzato mediamente in 14 su 16 round; questo permette di realizzare ottima confusione. I bit di parità della chiave, che portano la sua lunghezza da 56 a 64bit, sono un metodo per controllarne l'integrità (per esempio per trasmettere la chiave senza errori, come controllo aggiuntivo).
Notiamo che il DES rispetta il principio di Kerchoffs (pagina 96) infatti la chiave deve rimanere privata, ed è l'unico "ingrediente" che permette di decifrare un blocco

cifrato; l'algoritmo è invece pubblico e ben definito. L'algoritmo è infatti progettato per rendere l'analisi esaustiva dello spazio delle chiavi l'unico tipo di attacco possibile.

3.3. Modi operativi

I cifrari a blocchi, come DES, supportano i modi operativi descritti nelle seguenti sotto-sezioni.

Electronic Code Book (ECB). Il messaggio in chiaro è diviso in blocchi, i quali sono cifrati in modo indipendente, utilizzando la stessa chiave:

$$c_i = E_k(p_i), \quad p_i = E_k(c_i)$$

Risulta facile individuare i blocchi ripetuti, inoltre è possibile costruire una tabella con le corrispondenze tra blocchi i chiaro e blocchi cifrati.

Cipher Block Chaining (CBC). Si tratta di un modo operativo concatenato: in qualche modo, ciascun blocco è legato agli altri, e si ottiene l'effetto di non avere blocchi ripetuti (questo vale per ciascun modo concatenato). CBC usa un vettore di inizializzazione (IV), come se fosse una chiave addizionale, come primo blocco cifrato c_0; a partire da esso, le funzioni di cifratura e decifratura sono definite come:

$$c_i = E_k(p_i \oplus c_{i-1}), \quad p_i = c_{i-1} \oplus D_k(c_i)$$

Si noti che IV può essere inviato anche in chiaro.

Cipher Feedback (CFB). Si tratta di un modo concatenato. Consiste nel cifrare il vettore di inizializzazione e poi sommarlo modulo 2 al blocco in chiaro; il risultato è usato come IV per il round successivo. Questo algoritmo crea una sequenza di bit pseudo-casuali e, per tale ragione, può essere considerato come un cifrario a flusso (stream cipher).

Per decifrare non è necessaria una funzione specifica, ma basta ripetere la somma in modulo 2 con i blocchi cifrati al posto dei blocchi in chiaro:

$$c_i = p_i \oplus E_k(c_{i-1}), \quad p_i = c_i \oplus E_k(c_{i-1})$$

Lo svantaggio di questo cifrario è l'amplificazione degli errori di trasmissione, da un blocco al suo successivo.

La modalità CFB può essere anche utilizzata in modalità 8 bit, in cui vengono generati 8 bit alla volta di testo cifrato, che viene poi riportato all'ingresso alla funzione di cifratura sotto forma degli 8 bit meno significativi.

Output Feedback Mode (OFB). Si tratta di un modo concatenato. Consiste nel cifrare il vettore di inizializzazione, sommarlo al blocco in chiaro, e poi usare l'IV cifrato come ingresso per il blocco successivo. La catena di cifrature effettuate sull'IV costituisce una PRBS (pseudo-random binary sequence, una sequenza pseudo-casuale di bit, rendendolo anch'esso uno stream cipher). L'unico modo per generare la stessa sequenza è conoscere la chiave; inoltre un errore di trasmissione non si propaga tra i blocchi.

Le funzioni di cifratura e decifratura sono definite come:

$$c_i = p_i \oplus E_k^{(i)}(\mathrm{IV}), \quad p_i = c_i \oplus E_k^{(i)}(\mathrm{IV})$$

dove $E_k^{(i)}(\mathrm{IV})$ è la funzione E_k applicata in modo ricorsivo per i volte all'IV.

Anche la modalità OFB può essere usato in modalità 8 bit, analogamente al CFB.

Counter Mode (CTR). Questo modo non è concatenato. Consiste nel cifrare un IV con la chiave, per poi sommarlo al blocco in chiaro; per ciascun blocco in chiaro i-esimo viene usato un $\mathrm{IV}^{(i)} := \mathrm{IV} + i$, e ogni blocco cifrato è prodotto in modo indipendente; la sicurezza di questo modo si basa unicamente su quella della funzione di cifratura (in caso di ottima diffusione e confusione, un attaccante non può sapere se a IV sta venendo sommato un intero e quale esso sia).

Le funzioni di cifratura e decifratura sono definite come:

$$c_i = p_i \oplus E_k\left(\mathrm{IV}^{(i)}\right), \quad p_i = c_i \oplus E_k\left(\mathrm{IV}^{(i)}\right)$$

3.4. Sicurezza del DES

3.4.1. Doppio DES. Per aumentare la sicurezza di DES potremmo provare ad utilizzare una chiave più lunga; in tal caso, ci chiediamo se esista una chiave $K' = K_1 \parallel K_2$ tale che $|K'| = 56 + 56 = 112\text{bit}$, e per essa si verifichi che $E_{K'}(p) \equiv E_{K_2}(E_{K_1}(c))$.

Se prendiamo la funzione di cifratura *affine* (si veda la sezione 2.2.2), dove abbiamo $K_1 = (a, b)$, allora possiamo cifrare un testo in chiaro p come $c_1 = E_{K_1}(p) = a \cdot p + b \bmod n$; prendiamo un'altra chiave $K_2 = (c, d)$ e con essa cifriamo di nuovo il messaggio p, ottenendo $c_2 = E_{K_2}(p) = c \cdot p + d \bmod n$.

Si nota subito che, per le caratteristiche della funzione di cifratura (combinazione lineare con la chiave), esiste $K' = (a \cdot c, b + d)$ per cui vale $c' = E_{K_2}(E_{K_1}(p)) \equiv E_{K'}(p) = ac \cdot p + b + d \bmod n$. Dato che il cifrario affine forma un gruppo algebrico (chiuso rispetto alla composizione), cifrando due volte con due chiavi K_1 e K_2 non viene aumentata la cardinalità dello spazio delle chiavi, dato che esiste in ogni caso una chiave equivalente K' di lunghezza identica alle singole chiavi.

Il DES non gode di questa proprietà: cifrando più volte il messaggio in chiaro con due chiavi differenti, aumenta di $2^{|K|}$ la cardinalità dello spazio delle chiavi. Nel caso del doppio DES (2DES), si ha una chiave $K = K_1 \parallel K_2$ lunga il doppio di una chiave singola, dunque $|\mathscr{K}| = 2^{112} \simeq 2^2 \cdot 10^{33}$.

Il doppio DES è suscettibile dell'attacco Meet-in-the-Middle (MiM): questa vulnerabilità permette di analizzare al più 2^{57} chiavi, invece delle 2^{112} totali; per ovviare al problema si è scelto di proseguire con la composizione, ottenendo il triplo DES.

3.4.2. Triplo DES. Questo cifrario è la composizione di tre chiavi DES, che portano la cardinalità dello spazio delle chiavi a 2^{168}; anche con un attacco MiM lo spazio delle chiavi si riduce al più a 2^{112}, e al giorno d'oggi è considerato sicuro almeno quanto l'AES, ma è meno veloce (bisogna cifrare 3 volte con DES).

Il metodo più intuitivo per implementare il 3DES è effettuare tre volte la cifratura con tre chiavi differenti: $c = E_{K_1}(E_{K_2}(E_{K_3}(p)))$; un approccio che utilizza solo due chiavi, senza ridurre lo spazio delle chiavi, è il seguente:

$$c = E_{K_1}(D_{K_2}(E_{K_1}(p))), \quad p = D_{K_1}(E_{K_2}(D_{K_1}(c)))$$

Questa implementazione del triplo DES è compatibile con l'hardware per il singolo DES.

Rivest ha proposto la seguente alternativa, chiamata DESX, che usa tre chiavi ma effettua una sola cifratura DES, ed è sicura quanto il 3DES standard:

$$c = K_3 \oplus E_{K_2}(K_1 \oplus p)$$

3.4.3. Attacco Meet-in-the-Middle. Abbiamo già osservato che DES non è un gruppo; ci chiediamo come possiamo affrontare questa caratteristica, per ridurre lo spazio delle chiavi nel caso di una composizione. Conduciamo un attacco di testo in chiaro noto, partendo da una coppia corrispondente di testo in chiaro e testo cifrato $p \leftrightarrows c$, senza conoscere la chiave.

Poniamoci nel caso di un testo cifrato ricavato tramite 2DES come $c = E_{K_2}(E_{K_1}(p))$, e vogliamo ottenere $K' = K_1 \parallel K_2$. Se quanto affermato sul testo cifrato è vero, allora applicando a c la decifratura con K_2 si ottiene

$$D_{K_2}(c) = E_{K_1}(p) \tag{3.4.1}$$

Effettuiamo una esplorazione esaustiva dello spazio delle chiavi K_1 memorizzando i risultati in una tabella di hash, per cui sarà necessario memorizzare 2^{56} valori da 8Byte (64bit) ovvero circa 0.5EB; poi effettuiamo la stessa analisi per lo spazio delle chiavi K_2, questa volta evitando di memorizzare i risultati ma confrontando ciascuna chiave trovata con i valori di K_1 salvati.

Notiamo che saranno necessarie al più $2 \cdot 2^{56} = 2^{57}$ operazioni di cifratura e decifratura con DES, e al più 2^{112} confronti.

3.5. Sicurezza delle password

3.5.1. Funzioni di hash. Per ragioni di sicurezza, una password non viene mai memorizzata in chiaro; in pratica, si usa una funzione di *hash*. Si tratta di una funzione con le seguenti caratteristiche:

- accetta un ingresso di lunghezza arbitraria, produce una uscita di lunghezza fissa;
- non è invertibile (ci sono infiniti messaggi con lo stesso hash in uscita dalla funzione);
- sia unidirezionale (presa una uscita $y(x)$, sia impossibile trovare uno qualunque degli infiniti x che produca quell'uscita);
- goda di ottima diffusione.

Usando una simile funzione, possiamo memorizzare lo hash della password, e confrontarlo con l'hash calcolato sulla password quando fornita.

È possibile attaccare queste funzioni provando un gran numero di ingressi casuali differenti, rispetto al numero delle possibili uscite; la sicurezza viene determinata dalla lunghezza dell'uscita.

Si può mitigare questo attacco scegliendo una funzione di hash particolarmente lenta.

3.5.2. Attacco del vocabolario. Dato che quasi nessuno sceglie una stringa veramente casuale come password, è possibile attaccare l'hash corrispondente generando tutti gli hash di tutte le parole di una lingua, apportando eventuali variazioni sulle singole parole (aggiunta di un numero, iniziale maiuscola, eccetera...) e poi confrontando gli hash ottenuti con quelli nel file delle password.

All'interno di sistemi con un gran numero di utenti, la probabilità di trovare una password "banale" per almeno qualche utenza è molto elevata.

La contromisura migliore è applicare un *salt* ("sale") alla password; esso è una stringa di bit, memorizzata in chiaro nel file delle password. L'hash "salato" è calcolato sulla stringa ottenuta dalla concatenazione del sale con la password.

Il sale non aumenta la complessità dell'attacco sul singolo utente, bensì ha il vantaggio che, nel caso di un attacco del vocabolario su numerose utenze, essi abbiano tutti un hash diverso, anche a parità di password, per via del sale (che sarà diverso per

ogni utente). Un attaccante dovrà calcolare l'hash per ogni parola del vocabolario, per ogni sale possibile.

Nei sistemi UNIX, le password sono salate con un salt a 12bit, e la funzione di hash è basata su DES; tradizionalmente si troncava la password utente a 64bit (8 caratteri), da questi si ottenevano i primi 7 bit di ciascun Byte (per rappresentare solo i caratteri stampabili), e i 56bit così ottenuti erano usati come chiave DES. Questa chiave K_h è poi usata per cifrare un blocco di 0 da 64bit, con 25 round di DES; a ogni round, i 12bit del sale sono usati per perturbare la funzione di cifratura (modificando le S-Box, rende inutile l'implementazione hardware a priori). Infine sale e blocco cifrato sono codificati in base 64.

La funzione così realizzata è reputata unidirezionale: infatti ottenere la password senza conoscere la chiave e con un DES da più di 16 round, per giunta perturbato dal sale, è un problema difficile.

CAPITOLO 4

Cifrario AES

4.1. Introduzione

Nel '97, quando DES era ancora in vigore, il NIST fece un bando per la creazione di un nuovo standard crittografico, destinato alle informazioni governative; l'algoritmo selezionato fu chiamato AES. Si tratta di un cifrario a blocchi da 128bit, con chiave a 128, 192 o 256bit.

La procedura per selezionare l'algoritmo fu gestita in modo pubblico e trasparente, attraverso delle conferenze; prima di selezionare l'algoritmo che sarebbe divenuto AES, arrivarono "in finale" i cinque seguenti algoritmi (oggi ancora usati, perché paragonabili all'AES nella sicurezza): MARS, RC6, Rijndael, Serpent, Twofish.

Nel 2000 viene infine selezionato Rijndael, che viene rinominato in AES; in particolare, Rijndael definisce una famiglia di cifrari: per AES viene selezionato una dimensione del blocco di 128bit e chiavi da 128 a 256bit.

Dal punto di vista matematico, AES non si basa su uno schema Feistel, ma su una rete di sostituzioni e trasformazioni; a seconda della dimensione della chiave, il Rijndael effettua 10, 12 o 14 cicli su ciascun blocco, e può essere utilizzato in tutti i modi operativi concatenati mostrati per il DES (vedere la sezione §3.3).

4.2. Algoritmo AES

Per maggiore semplicità, analizziamo il caso di Rijndael con chiave a 128bit, e 10 round. Osserviamo che i blocchi da 128bit sono suddivisi in 16Byte, organizzati in matrici 4×4; tutti gli elementi di queste matrici sono ottetti del campo $\mathcal{GF}(2^8)$ e rappresentano i polinomi al suo interno, dove il polinomio irriducibile per definire il campo è $P(x) = x^8 + x^4 + x^3 + x^2 + x + 1$. L'ottetto di zeri non è invertibile, tuttavia deve essere ammissibile nel cifrario: si consideri l'elemento nullo come inverso di sé stesso.

Ogni round del Rijndael è composto da 4 livelli, che operano sulla matrice 4×4 che rappresenta il blocco da 128bit:

- Substitute Byte (SB): tabella di sostituzione, simile alle S-Box del DES, ma costruita algebricamente in maniera trasparente;
- Shift Row (SR): scorrimento delle righe della matrice a sinistra;
- Mix Column (MC): scorrimento delle colonne a sinistra, evita la degenerazione del cifrario in 4 cifrari applicati alle singole colonne della matrice;
- Add Round Key (ARK): per ciascun ciclo si aggiunge una chiave di round, derivata da quella iniziale a 128bit.

L'algoritmo usa round e livelli nel modo seguente:

(1) ARK con chiave di round 0;
(2) nove round che usano i livelli SB $\to$ SR $\to$ MC $\to$ ARK, con le chiavi di round dalla 1 alla 9;
(3) un ultimo round che usa i livelli SB $\to$ SR $\to$ ARK, con la chiave di round 10.

Le differenze col DES riguardano l'assenza del Feistel, rispetto al rimescolamento dei bit dato dai quattro livelli di ciascun round; il risultato è una apparente diffusione perfetta già dal secondo round. Inoltre, tutte le operazioni sugli ottetti del blocco, sono all'interno del campo $\mathcal{GF}(256)$.

La notazione $\boldsymbol{A}$ in maiuscolo corsivo e grassetto, indica che A è una matrice

SB. Sostituzione non lineare applicata ai Byte (elementi) della matrice; secondo una matrice di sostituzione con 16 righe e 16 colonne, numerate a partire da 0, indirizzate tramite gli stessi bit degli ottetti, nel modo seguente: sia $\boldsymbol{A}$ la matrice che rappresenta il blocco di partenza, e sia $a_{1,0} = 10001011$; sia $\boldsymbol{B}$ la matrice ottenuta al termine del livello SB; allora $b_{1,0}$ si ottiene prendendo il valore indicato all'incrocio della riga 8 con la colonna 11, nella tabella di sostituzione, e sostituendolo al valore di un elemento di $\boldsymbol{A}$ (per esempio $a_{i,j} = \overbrace{1000}^{\text{riga}}\underbrace{1011}_{\text{colonna}} \rightarrow 61$).

Questa tabella si calcola nel modo seguente:

- l'input è un ottetto di bit $a_{i,j} = x_7x_6x_5x_4x_3x_2x_1x_0$
- calcoliamo il suo inverso in $\mathcal{GF}(256) \ni a_{i,j}^{-1} = y_7y_6y_5y_4y_3y_2y_1y_0$ (assumendo che 000000000 sia l'inverso di sé stesso)
- prendiamo l'ottetto $a_{i,j}^{-1}$, scriviamolo partendo dal bit meno significativo e come un vettore colonna $(y_0y_1y_2y_3y_4y_5y_6y_7)^{\mathrm{T}}$
- computiamo $b_{i,j} = z_7z_6z_5z_4z_3z_2z_1z_0$ e scriviamolo come $(z_0z_1z_2z_3z_4z_5z_6z_7)^{\mathrm{T}}$

per computare $b_{i,j}$ usiamo la seguente trasformazione, per diffondere e confondere ulteriormente i bit:

$$b_{i,j} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

SR. Spostamento a sinistra delle 4 righe rispettivamente di 0, 1, 2, 3 passi (la prima riga rimane immutata, la seconda è spostata di 1, ecc...); sia $\boldsymbol{B}$ la matrice proveniente dal livello SB, allora da questo livello otteniamo la seguente matrice $\boldsymbol{C}$:

$$C = \begin{bmatrix} c_{0,0} & c_{0,1} & c_{0,2} & c_{0,3} \\ c_{1,0} & c_{1,1} & c_{1,2} & c_{1,3} \\ c_{2,0} & c_{2,1} & c_{2,2} & c_{2,3} \\ c_{3,0} & c_{3,1} & c_{3,2} & c_{3,3} \end{bmatrix} = \begin{bmatrix} b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} \\ b_{1,1} & b_{1,2} & b_{1,3} & b_{1,0} \\ b_{2,2} & b_{2,3} & b_{2,0} & b_{2,1} \\ b_{3,3} & b_{3,0} & b_{3,1} & b_{3,2} \end{bmatrix}$$

MC. Prendiamo la matrice $\boldsymbol{C}$ ottenuta dal livello precedente ed effettuiamo il suo prodotto per una matrice M per ottenere $\boldsymbol{D} = \boldsymbol{M} \cdot \boldsymbol{C}$, nel modo seguente:

$$D = \begin{bmatrix} 00000010 & 00000011 & 00000001 & 00000001 \\ 00000001 & 00000010 & 00000011 & 00000001 \\ 00000001 & 00000001 & 00000010 & 00000011 \\ 00000011 & 00000001 & 00000001 & 00000001 \end{bmatrix} \cdot \begin{bmatrix} c_{0,0} & c_{0,1} & c_{0,2} & c_{0,3} \\ c_{1,0} & c_{1,1} & c_{1,2} & c_{1,3} \\ c_{2,0} & c_{2,1} & c_{2,2} & c_{2,3} \\ c_{3,0} & c_{3,1} & c_{3,2} & c_{3,3} \end{bmatrix}$$

Per garantire la decifratura, sarà necessario che la matrice $\boldsymbol{M}$ sia invertibile; l'inversa $\boldsymbol{M}^{-1}$ avrà più 1 che 0, e per questo la decifratura in AES è più lenta (di poco) della cifratura.

ARK. La chiave di round viene sommata bit a bit col blocco ottenuto al livello precedente: $\boldsymbol{E} = \boldsymbol{D} \oplus \boldsymbol{K}^{(i)}$. Ad ogni round i-esimo ricaviamo una chiave di 16Byte a partire da quella iniziale; se abbiamo chiavi da 128bit l'algoritmo prevede 10 round e quindi saranno necessarie 11 chiavi di round.

Esprimendo la chiave iniziale come matrice 4×4 chiamata $\boldsymbol{K}$, possiamo indicarla come la giustapposizione di 4 colonne di 4 elementi ciascuna; le 11 chiavi necessarie saranno costituite in totale da 44 colonne di 4Byte ciascuna. Allora le colonne 0, 1, 2, 3 appartengono alla chiave $\boldsymbol{K}^{(0)}$, le colonne 4, 5, 6, 7 costituiscono $\boldsymbol{K}^{(1)}$, e così via.

Le altre 40 colonne si ottengono dalle 4 iniziali nel modo seguente; sia $\vec{w}(i)$ la i-esima colonna:

$$\begin{cases} \vec{w}(i) = \vec{w}(i-4) \oplus \vec{w}(i-1) & i \bmod 4 \neq 0 \\ \vec{w}(i) = \vec{w}(i-4) \oplus T\left[\vec{w}(i-1)\right] & i \bmod 4 = 0 \end{cases}$$

La trasformazione T è definita nel modo seguente; sia $r(i) = 00000010^{(i-4)/4} \in \mathcal{GF}(2^8)$, e sia SB il livello Substitute Byte dell'algoritmo Rijndael:

$$T[(a,\, b,\, c,\, d)] = \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} \rightarrow \begin{bmatrix} b \\ c \\ d \\ a \end{bmatrix} \rightarrow \mathrm{SB} \rightarrow \begin{bmatrix} e \\ f \\ g \\ h \end{bmatrix} \rightarrow \begin{bmatrix} e \oplus r(i) \\ f \\ g \\ h \end{bmatrix}$$

In questo modo si ottiene la chiave di round come:

$$\boldsymbol{K}^{(i)} = \left[\vec{w}(4i),\, \vec{w}(4i+1),\, \vec{w}(4i+2),\, \vec{w}(4i+3)\right]$$

Decifratura. I quattro livelli di ciascun ciclo dell'algoritmo Rijndael sono invertibili:

- Inverted Substitute Byte (ISB): tabella di sostituzione inversa a quella del livello SB;
- Inverted Shift Row (ISR): scorrimento delle righe a destra invece che a sinistra;
- Inverted Mix Column (IMC): si usa l'inversa della matrice $\boldsymbol{M}$: $\boldsymbol{C} = \boldsymbol{M}^{-1} \cdot \boldsymbol{D}$;
- Add Round Key (ARK): questo livello, basato sulla somma bit a bit, è l'inverso di sé stesso.

Possiamo decifrare un blocco applicando all'inverso i 10 round:

(1) il primo round usa $\boldsymbol{K}^{(10)}$ e i livelli ARK $\rightarrow$ ISR $\rightarrow$ ISB;
(2) i nove round successivi usano le chiavi dalla 9 alla 1, con i livelli ARK $\rightarrow$ IMC $\rightarrow$ ISR $\rightarrow$ ISB;
(3) un ultimo round usa ARK con la chiave $\boldsymbol{K}^{(0)}$.

4.3. Sicurezza dell'AES

Al contrario del DES, le cui S-Box avevano fatto sorgere il dubbio della presenza di trapdoor segrete, in AES la tabella di sostituzione è ottenuta tramite una procedura algebrica definita; inoltre, il livello Substitute Byte consente di ottenere un elevato livello di non linearità, con la trasposizione dei vettori.

Ad oggi, gli unici attacchi che funzionano contro AES, sfruttano debolezze dell'implementazione, e non dell'algoritmo.

CAPITOLO 5

Successioni Pseudo-casuali

5.1. Sequenze binarie pseudo-casuali (PRBS)

Se pensiamo che l'AES sia sicuro, dobbiamo limitare questa osservazione alla pratica: infatti, anche se non è possibile trovare la chiave di cifratura in un periodo di tempo accettabile, dal punto di vista teorico e con un tempo infinito a disposizione, è possibile eludere la sicurezza di AES.

Esiste tuttavia un cifrario inviolabile anche dal punto di vista teorico: prendiamo una sequenza di bit da cifrare p e la sommiamo bit a bit con una sequenza casuale r, ottenendo la sequenza cifrata:

$$c = p \oplus r$$

La sequenza r deve essere veramente casuale (nessuna autocorrelazione del primo o secondo ordine) e impredicibile, ottenuta per esempio osservando un fenomeno aleatorio. Per decifrare un c ottenuto in questo modo, è necessario sommare bit a bit a c la stessa sequenza casuale r usata nella cifratura.

Osserviamo che la sequenza casuale r deve avere la stessa lunghezza dei dati da cifrare ($|r| = |p|$, per consentire la somma bit a bit con p), ed essa deve essere trasportata in modo sicuro al destinatario; inoltre, la sequenza r va usata una volta sola, per mantenere la sua impredicibilità. Questo sistema è comunemente denominato One Time Pad.

Questo sistema di cifratura ha però rilevanza pratica se possiamo generare in modo sicuro una sequenza r pseudo-casuale, ovvero a parità di inizializzazione, è possibile generare in modo indipendente la stessa sequenza r.

5.1.1. Generatore lineare congruenziale. Ricaviamo l'elemento i-esimo della sequenza pseudo-casuale attraverso una trasformazione lineare applicata al precedente, modulo un composto:

$$x_i = (a \cdot x_{i-1} + b) \bmod n$$

La sequenza viene generata a partire dai tre parametri a, b, n e il suo sviluppo è determinato dal seme iniziale, il primo elemento x_0.

Anche se il modulo aggiunge della complessità alla trasformazione lineare è triviale ottenere i tre parametri osservando abbastanza a lungo la sequenza r.

5.1.2. Funzione unidirezionale. Un metodo algoritmicamente sicuro ma computazionalmente non efficiente, consiste nell'impiego di funzioni unidirezionali nella generazione della sequenza. Le funzioni non invertibili $y = f(x)$ permettono di tentare di trovare un $\overline{y}$ provando per tentativi degli $\overline{x}$ tali che $\overline{y} = f(\overline{x})$, mentre le funzioni unidirezionali (invertibili o meno) non permettono di "tirare a indovinare" l'$\overline{x}$ che genera un $\overline{y}$ (anche provando per tentativi risulta quasi impossibile trovare $\overline{x}$).

Una sequenza pseudo-casuale del genere si inizializza con un valore iniziale (detto seme) x_0, mentre l'elemento i-esimo si ottiene come:

$$x_i = f(x_0 + i)$$

Se la funzione unidirezionale $f()$ è, per esempio, la funzione di cifratura del DES con chiave $K = x_0 + i$ (va applicata la funzione per generare ciascun bit i-esimo), abbiamo una buona funzione unidirezionale (genera una sequenza apparentemente casuale, per invertire la funzione sarebbe necessario trovare la chiave DES) che tuttavia risulta non pratica per l'elevato costo computazionale.

5.1.3. Generatore ai residui quadratici (Blum, Blum e Shub). Si prendano due primi p e q grandi, entrambi congruenti a 3 in modulo 4:

$$p \gg 1,\ q \gg 1\ :\ p \equiv 3 \pmod 4 \wedge q \equiv 3 \pmod 4$$

Definiamo il prodotto $n = p \cdot q$, e prendiamo un numero casuale che non abbia p o q tra i suoi fattori ($x \perp n$).

La successione viene inizializzata a partire da un residuo quadratico $x_0 \equiv x^2 \pmod n$; l'i-esimo elemento viene calcolato come:

$$x_i \equiv x_{i-1}^2 \pmod n$$

mentre il bit i-esimo della sequenza sarà il bit meno significativo dell'elemento x_i ($b_i = \text{LSB}(x_i)$):

Normalmente, la sequenza di bit $\{b_i\}$ ha una statistica ben fatta (probabilità circa $1/2$ che esca 0 o 1), ha autocorrelazione nulla, ed assenza di memoria (b_i non dipende da b_{i-1}). La differenza con una sequenza casuale vera è che essa è periodica (dovrà essere garantito un periodo lungo); inoltre si tratta di un generatore lento (per il calcolo di un quadrato per ciascun bit).

Con una sequenza generata in questo modo, è possibile prevedere un bit successivo b_{i+k} una volta noti un x_i e n; tuttavia è impossibile ottenere un b_{i-h}, poiché sarebbe necessario calcolare la radice quadrata modulo n, e quindi sono necessari i fattori p e q (e con p e q scelti adeguatamente il problema della fattorizzazione risulta impraticabile).

ESEMPIO 5.1. *Usare il generatore di Blum, Blum e Shub con $p = 43$ e $q = 31$, cercando di determinare il suo periodo.*

✓Ricaviamo subito $n = p \cdot q = 1333$; scegliamo inoltre $x_0 = 50 \perp 1333$. Applichiamo l'algoritmo:

i	x_i	b_i
0	$50^2 \equiv \mathbf{1167} \pmod{1333}$	1
1	$1167^2 \equiv \mathbf{896} \pmod{1333}$	0
2	$896^2 \equiv \mathbf{350} \pmod{1333}$	0
3	$350^2 \equiv \mathbf{1197} \pmod{1333}$	1
4	$1197^2 \equiv \mathbf{1167} \pmod{1333}$	1

Notiamo che il periodo vale 4 (il generatore continuerà a produrre 1, 0, 0, 1 . . .). □

5.2. Periodo del generatore ai residui quadratici

5.2.1. Calcolare il periodo. Gli studi di Blum, Blum e Shub hanno mostrato che il periodo τ della sequenza del loro generatore, è un divisore della funzione della funzione di Carmichael, del prodotto $n = p \cdot q$:

$$\tau \backslash \lambda(\lambda(n)),\ n = p \cdot q$$

La funzione di Carmichael di un numero n è, a sua volta, un divisore del toziente di n ($\lambda(n) \backslash \varphi(n)$); essa si calcola nel modo seguente; preso un intero composto positivo n, esso può essere espresso come prodotto di fattori primi p_i, ciascuno con la propria potenza a_i:

$$n = p_1^{a_1} \cdot p_2^{a_2} \cdot \ldots \cdot p_r^{a_r}$$

La funzione di Carmichael è pari al minimo comune multiplo degli ordini (tozienti) di ciascuno dei $p_i^{a_i}$, per i da 1 a r (elementi del gruppo moltiplicativo degli elementi invertibili di $\mathbb{Z}_n$):

Funzione di Carmichael

$$\lambda(n) = \operatorname{mcm}\left(\left\{\lambda\left(p_i^{a_i}\right)\right\},\ i \in [1,\ r]\right) \tag{5.2.1}$$

Dove, per il teorema di Carmichael, si ha che:

Teorema di Carmichael

$$\lambda\left(p^k\right) = \begin{cases} \dfrac{\varphi(p^k)}{2} & p = 2 \wedge k \geq 3 \\ \varphi(p^k) & \text{altrimenti} \end{cases}$$

Prendiamo in mano le ipotesi del Teorema 1.3, il quale afferma che preso un intero a e un primo p, vale $a^{p-1} \equiv 1 \pmod p$; se p non fosse primo, il Teorema 1.4 ci dice che $a^{\varphi(p)} \equiv 1 \pmod p$. Nel primo caso, ci chiediamo infine se esista un intero $q < p-1$ per cui il residuo modulo p di a vale sempre 1 ($a^q \equiv 1 \pmod p$); in generale questo dipende da a, e chiamiamo questo intero $\operatorname{ord}(a) = q$.

Nel secondo caso, ci chiediamo se esista un divisore di $\varphi(p)$ per cui il residuo modulo p di a sia sempre 1; questo accade per $\lambda(p)$, che possiamo considerare come l'analogo dell'ordine di a.

Facciamo un paragone tra ordine e funzione di Carmichael:

- $\operatorname{ord}(a) \in \mathbb{Z}_p$ è il minimo intero n tale che $a^n \equiv 1 \pmod p$; l'ordine di a è massimo $p-1$ o comunque un suo divisore.
- $\lambda(n) \in \mathbb{Z}_p$ è il minimo intero N tale che $a^N \equiv 1 \pmod p$; $\lambda(n)$ è al massimo $\varphi(n)$ o comunque un suo divisore.

Esempio 5.2. Calcolare il periodo del generatore BBS (Blum, Blum e Shub), con $p = 23$, $q = 19$, $x_0 = 7$.

✓Sia p che q sono congruenti a 3 modulo 4, inoltre x_0 essendo primo è sicuramente primo relativo rispetto a $p \cdot q = n = 437$. Calcoliamo la successione (si ricorda che tutti i conti per la colonna x_i sono quadrati in modulo n, mentre la colonna b_i contiene l'LSB corrispondente):

i	x_i	b_i	i	x_i	b_i
0	49	1	6	87	1
1	216	0	7	140	0
2	334	0	8	372	0
3	121	1	9	292	0
4	220	0	10	$\boxed{49}$	1
5	330	0			

Dalla successione costruita osserviamo che il periodo vale 10. Sappiamo che il periodo sarà un divisore di $\lambda(\lambda(n))$; calcoliamo $\lambda(n)$:

$$\begin{aligned} \lambda(n) = \lambda(437) = \lambda(23 \cdot 19) &= \operatorname{mcm}(\varphi(23),\ \varphi(19)) \\ &= \operatorname{mcm}(22,\ 18) = \operatorname{mcm}\left(2 \cdot 11,\ 2 \cdot 3^2\right) = 11 \cdot 2 \cdot 9 \\ &= 198 \end{aligned}$$

Ora possiamo calcolare $\lambda(\lambda(n))$:

$$\begin{aligned} \lambda(198) = \lambda\left(2 \cdot 3^2 \cdot 11\right) &= \operatorname{mcm}(\varphi(2),\ \varphi(9),\ \varphi(11)) \\ &= \operatorname{mcm}(1,\ 2 \cdot 3,\ 2 \cdot 5) = 2 \cdot 3 \cdot 5 = 30 \end{aligned}$$

Concludiamo che il periodo τ sarà uno tra i possibili divisori di 30, a seconda dal x_0 selezionato: $\tau \in \{1, 2, 3, 5, 6, 10, 15, 30\}$; presi $p = 23$, $q = 19$ il periodo è un divisore di 30; con $x_0 = 7$ si ha esattamente $\pi = 10$. □

5.2.2. Massimizzare il periodo. Per avere un periodo lungo di una sequenza BBS, è necessario scegliere un x_0 tale che $\pi = \lambda(\lambda(n)) \gg 1$; perché ciò si verifichi, scegliamo p e q nel modo seguente:

- prendiamo un primo p_2, tale che anche un $p_1 = 2p_2 + 1$ sia primo; scegliamo il primo $p = 2 \cdot p_1 + 1$
- prendiamo un primo q_2, tale che anche un $q_1 = 2q_2 + 1$ sia primo; scegliamo il primo $q = 2 \cdot q_1 + 1$
- deve valere $p \neq q$, e $p, q \gg 1$

Scegliendo i due primi col criterio appena mostrato, si otterrà il periodo più lungo possibile.

5.3. Successioni LFSR

Si tratta di successioni lineari di bit, con aritmetica modulo 2; definiamo una sequenza PRBS con la seguente costruzione:

$$x_{n+5} \equiv x_n + x_{n+2} \pmod 2$$

Una notazione equivalente per la precedente congruenza è:

$$x_n \equiv x_{n-5} + x_{n-3} \pmod 2 \tag{5.3.1}$$

Chiamiamo successione LFSR con memoria 5 la congruenza (5.3.1).

In generale, data una memoria di M bit (la quale genera una ricorrenza di ordine M), possiamo definire una successione LFSR come:

$$x_{n+M} \equiv c_M \cdot x_n + c_{M-1} \cdot x_{n+1} + \ldots + c_1 \cdot x_{n+M-1} \pmod 2$$

Come prima, una costruzione alternativa (vedremo che essa semplificherà la sintesi di un circuito sequenziale) per la precedente è:

Successione LFSR

$$x_n \equiv c_M \cdot x_{n-M} + c_{M-1} \cdot x_{n-M+1} + \ldots + c_1 \cdot x_{n-1} \pmod 2 \tag{5.3.2}$$

Possiamo descrivere la ricorrenza M relativa alla successione (5.3.2) tramite il seguente polinomio, che ha per coefficienti i c_i della ricorrenza:

$$P(Y) = c_M Y^M + c_{M-1} Y^{M-1} + \ldots + c_1 Y + c_0 \tag{5.3.3}$$

Il polinomio è nella variabile Y per distinguere chiaramente la sua variabile da quella della successione (5.3.2); usiamo un polinomio per descrivere la ricorrenza, per lavorare con i campi di Galois.

Il polinomio (5.3.3) ha grado M e descrive una successione con periodo al più $2^M - 1$; i polinomi di grado esattamente M sono in numero $2^M - 1$: infatti possiamo affermare che $c_0 = 1$ (riferito all'elemento presente) e $c_M = 1$ (altrimenti il polinomio non arriverebbe al grado M), sottraendo due gradi di libertà dal numero totale di polinomi di grado M.

Per avere una ricorrenza di periodo massimo, dovremo scegliere un polinomio *irriducibile*, da cercare tra i $2^M - 1$ disponibili.

Questa ricorrenza può essere implementata via hardware, tramite dei registri a scorrimento; per esempio per avere una ricorrenza con periodo di un milione serviranno almeno 20 celle di memoria ($2^{20} - 1 \simeq 10^6$). La sequenza $\{x_M\}$ generata in questo modo potrà essere usata come sequenza pseudo-casuale r, nel sistema di cifratura introdotto all'inizio de la sezione §5.1. In tal caso la chiave sarà costituita da:

- i coefficienti del polinomio (5.3.3), tranne il primo e l'ultimo — sappiamo che essi valgono sempre 1 — ovvero $M-1$ bit in totale, a rappresentare i rimanenti coefficienti (che essendo residui modulo 2 valgono 0 o 1);
- i valori degli x con i quali inizializzare il polinomio, ovvero M bit.

Questo metodo è vulnerabile all'attacco del testo in chiaro, ed è usato quando la velocità di cifratura va privilegiata rispetto alla sicurezza.

ESEMPIO 5.3. *Consideriamo la successione LFSR definita con la congruenza seguente:*

$$x_n \equiv x_{n-2} + x_{n-3} \pmod 2$$

Implementare la ricorrenza in esame con un circuito sequenziale.

✓Possiamo rappresentare la successione col seguente polinomio:

$$P(Y) = Y^3 + Y^2 + 1$$

Infatti l'ordine della ricorrenza è 3 (ottenuto osservando gli indici delle x), sappiamo che $c_3 = c_0 = 1$ e infine i termini x_{n-2} e x_{n-3} rappresentano Y^2 e Y^3 rispettivamente; il periodo massimo della sequenza vale $2^3 - 1 = 7$ (sarebbe sicuramente massimo se $P(Y)$ fosse irriducibile).
Implementiamo la successione LFSR col circuito ne la figura 5.3.1. □

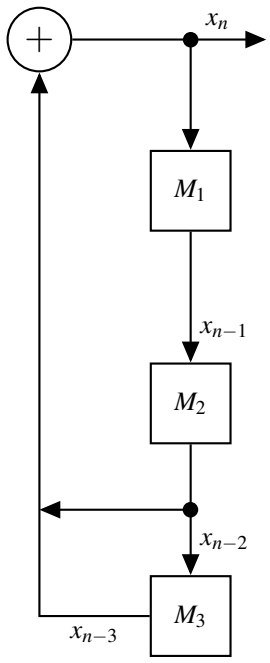

Figura 5.3.1. Registro a scorrimento con 3 celle di memoria

OSSERVAZIONE 5.1. Abbiamo già concluso che il periodo di una successione LFSR definita dal polinomio (5.3.2) *irriducibile*, sarà al massimo $2^M - 1$, oppure un suo sottomultiplo (può essere un numero diverso se $P(Y)$ non è irriducibile); tuttavia, se $2^M - 1$ è *primo*, il suo unico sottomultiplo è lui stesso (si ottiene il periodo massimo possibile). In breve vale la seguente condizione:

$$P(Y) \text{ irriducibile} \wedge 2^M - 1 \text{ primo} \implies \{x_M\} \text{ periodo massimo } \left(2^M - 1\right)$$

5.4. Stato della successione

Supponiamo di prendere il polinomio $P(Y)$ irriducibile (5.3.3), allora possiamo definire il campo di Galois con questo polinomio:

Tutti i conti effettuati in questa sezione sono in modulo $P(Y)$

$$\mathcal{GF}\left(2^M\right) = \mathbb{Z}_2[Y] \bmod P(Y)$$

I polinomi appartenenti a questo campo, che chiamiamo $s_j(Y)$, avranno al massimo grado $M-1$ e gli M coefficienti di ciascun polinomio rappresentano lo stato al passo j-esimo di un circuito sequenziale che genera una successione LFSR, che chiameremo *scrambler*.

Possiamo anche considerare $s_j(Y) \in \mathcal{GF}\left(2^M\right)$ come un vettore a M dimensioni, dove le coordinate valgono in modulo 2 (0 o 1).

Se moltiplichiamo per Y un polinomio $s_j(Y)$ — il quale rappresenta lo stato delle M celle di memoria — otteniamo lo shift in avanti di un passo del circuito combinatorio:

$$Y \cdot s_j(Y) = s_{j+1}(Y)$$

A questo punto, possiamo scrivere lo stato al passo j-esimo come un residuo in modulo $P(Y)$, rispetto allo stato iniziale:

$$s_j(Y) \equiv s_0(Y) \cdot Y^j \pmod{P(Y)}$$

Chiediamoci dunque sotto quali condizioni $s_j(Y) = s_0(Y)$, o in altre parole quanto vale il passo al quale inizia il secondo periodo; questo si verifica quando $Y^j \equiv 1 \pmod{P(Y)}$ Ci interessa il j per cui questa congruenza è vera, quindi ci interessa l'intero minimo per cui Y (elemento del campo) elevato a quell'intero dia 1 come residuo: in pratica cerchiamo l'ordine dell'elemento Y:

$$s_j(Y) = s_0(Y) \implies j = \text{ORD}(Y),\ Y \in \mathcal{GF}\left(2^M\right)$$

PROPOSIZIONE 5.1. *L'ordine di un elemento del campo sarà del tipo* $\left(2^M - 1\right)/d$*, con* d *un qualche intero; si verifica* $d = 1$ *(ordine massimo, pari a* $2^M - 1$*) quando* Y *è un elemento generatore del campo:*

$$\text{ORD}(Y) = 2^M - 1 \implies Y \text{ elemento generatore}$$

PROPOSIZIONE 5.2. *Condizione sufficiente perché il periodo della ricorrenza sia* $2^M - 1$ *è che il numero* $2^M - 1$ *sia primo; in tal caso tutti gli elementi del campo* $\mathcal{GF}\left(2^M\right)$ *saranno generatori.*

5.5. Scrambler

Consideriamo un apparato di telecomunicazioni: esso avrà un canale che trasporta una sequenza di bit; in generale le proprietà statistiche della sequenza di bit, emessi dalla sorgente, non saranno uniformi. È possibile che una sorgente trasmetta sequenze lunghe di 0 o 1 (sia per volontà degli utenti, che per segnalare una condizione o un evento, o uno stato dell'infrastruttura).

Il modo più banale per codificare bit su un canale di comunicazione, per esempio sulla fibra ottica, è quello di emettere potenza (luce) quando si vuole inviare un 1, e non emettere nulla quando si vuole inviare uno 0. In ricezione, si pone il problema di capire se sia arrivato un 1 o uno 0: si misura la potenza ricevuta, a intervalli di tempo regolari T (periodo di cifra), e si determina dell'intensità del segnale in arrivo è maggiore di una certa soglia e può essere considerata la codifica di un 1 piuttosto che di uno 0.

Tuttavia i componenti che costituiscono i circuiti del trasmettitore e del ricevitore, a causa della propria tolleranza di fabbricazione, non possono essere allineati in partenza sulla frequenza di trasmissione (sarà sempre presente uno sfasamento del segnale ricevuto, chiamato *jitter*, o rumore di fase). Si usa quindi un circuito chiamato *clock recovery*, che estrae il sincronismo dal segnale ricevuto; questo è usato come base per generare un segnale di clock che pilota un campionatore, facendolo agire esattamente a metà del periodo di cifra ($T/2$); per estrarre il clock si utilizzano i fronti di salita e discesa del segnale in ricezione.

Un test a livello fisico che si effettua sul ricevitore è chiamato CID (consecutive identical digits, ovvero cifre identiche consecutive)

Il problema del clock recovery è l'incapacità di estrarre il sincronismo da una sequenza di cifre uguali (tutte 1 o 0). Vi sono due strade per ovviare al problema:

(1) usare una modulazione differente per codificare il segnale nel canale (per esempio HDB3 trasmette transizioni di potenza a prescindere dalla cifra codificata);
(2) su fibra, dove si trasmette con la modulazione on/off, si usa uno *scrambler*.

La seconda strada prevede l'impiego dello scrambler, che genera una sequenza pseudo-casuale (non ci saranno più sequenze lunghe di cifre ripetute); esso dovrà avere le seguenti proprietà:

- equalizza la statistica del primo ordine (la *media* degli 1 e 0 trasmessi sia *uguale*);
- equalizza la statistica del secondo ordine (l'autocorrelazione degli 1 e degli 0 deve essere uguale, ovvero la sequenza deve avere *assenza di memoria* — essere non periodica).

Per realizzare uno scrambler si usa una successione LFSR (si veda la sezione §5.3); possiamo realizzare gli scrambler in due modi.

5.5.1. Scrambler auto-sincronizzante. Si prenda la sequenza $\{I_k\}$ dei dati da confondere, la si somma in modulo 2 al contenuto di una serie di celle di memoria, collegando in retroazione solo quelle identificate dal polinomio caratteristico che descrive la successione LFSR usata. Si ottiene la sequenza $\{U_k\}$ dei dati confusi.

Esempio 5.4. *Consideriamo la successione LFSR definita dal polinomio:*

$$P(Y) = Y^3 + Y + 1$$

Implementare uno scrambler descritto da $P(Y)$.

✓Possiamo rappresentare il polinomio con la congruenza:

$$x_n \equiv x_{n-1} + x_{n-3} \pmod 2$$

Infatti l'ordine della ricorrenza è 3 (ottenuto osservando il grado di $P(Y)$), sappiamo che $c_3 = c_0 = 1$ e infine i termini Y e Y^3 rappresentano rispettivamente x_{n-1} e x_{n-3}; il periodo massimo della sequenza vale $2^3 - 1 = 7$ (sarebbe sicuramente massimo se $P(Y)$ fosse irriducibile).
Lo scrambler è implementato dal circuito ne la figura 5.5.1. □

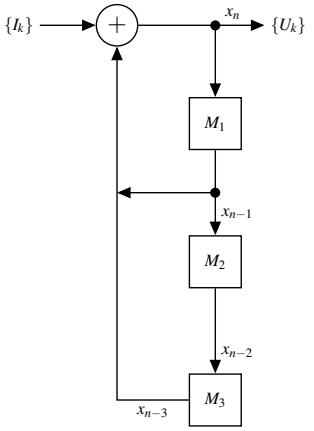

Figura 5.5.1. Scrambler auto-sincronizzante di ordine 3

Uno scrambler come quello realizzato nell'esempio, avrà periodo massimo 7, nel caso in cui l'ingresso sia costituito da una sequenza di cifre uguali.

Per effettuare l'operazione inversa dello scrambling, basta invertire il senso delle frecce che escono da $\{I_k\}$ e entrano in $\{U_k\}$ (grazie alle somme modulo 2, bisogna solamente invertire la direzione di ingresso e uscita); inoltre non è necessario conoscere il contenuto di M_1, M_2 e M_3, tuttavia saranno necessari 3 passi iniziali per inizializzare lo stato dello scrambler. In pratica sarà necessario conoscere unicamente il polinomio $P(Y)$.

5.5.2. Scrambler additivo.

Il funzionamento di questo scrambler è simile al precedente, con la differenza che l'ingresso viene spento (sostituito da zeri), e il circuito LFSR genera una sequenza casuale $\{R_k\}$ (come un generatore PRBS), che viene infine sommata all'ingresso per generare l'uscita. In questo caso è necessario conoscere anche l'inizializzazione delle celle M_i, oltre al polinomio $P(Y)$.
Lo scrambler additivo sopperisce a un difetto di quello auto-sincronizzante: quest'ultimo infatti propaga gli errori; un bit sbagliato ricevuto da uno scrambler autosincronizzante interrompe la sequenza, e sono necessari M passi per inizializzare nuovamente il registro a scorrimento.

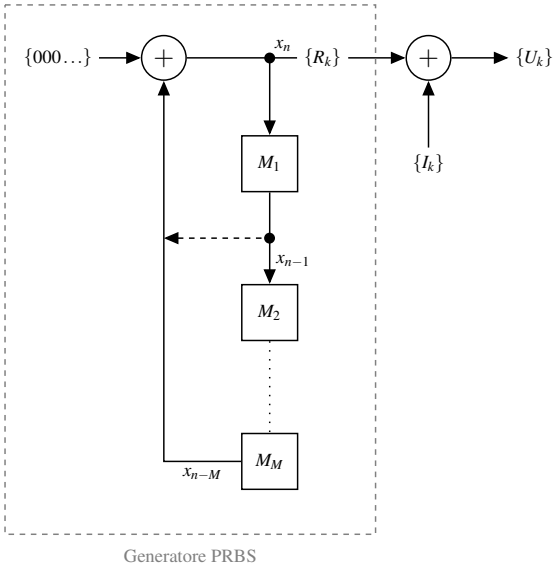

Figura 5.5.2. Scrambler additivo di ordine M

Per effettuare il *descrambling* (operazione inversa dello scrambling), si effettua di nuovo l'inversione delle frecce collegate all'ingresso e all'uscita.

FATTO 5.1. *Tutti i dati trasmessi da un utente, prima di essere mappati nel canale trasmissivo tramite un protocollo di livello 2 o 1, passano attraverso un scrambler auto-sincronizzante con polinomio* $P(Y) = Y^{43} + 1$*; questa configurazione comporta una propagazione di eventuali errori, tuttavia garantisce maggiore sicurezza, impedendo che un attaccante manipoli l'uscita* $\{U_k\}$ *dello scrambler (sarebbe necessario conoscere lo stato del registro a scorrimento, è possibile ottenerlo tramite* 2^{43} *tentativi — circa* 10^{12}*).*

In tutti i sistemi di comunicazione si disaccoppiano i dati trasmessi dall'utente da quelli che circolano sul mezzo trasmissivo

CAPITOLO 6

RSA

6.1. Introduzione

Trattiamo l'algoritmo di cifratura a chiave pubblica Rivest-Shamir-Adleman (RSA). La peculiarità degli algoritmi a chiave pubblica è la condivisione di una chiave pubblica con chiunque voglia cifrare un messaggio, il quale potrà essere decifrato solo dalla chiave privata, custodita dal destinatario. La generazione della coppia di chiavi pubblica e privata è realizzata mediante una funzione unidirezionale (si veda la sezione 3.5.1); in questo modo, si ricava la chiave pubblica da quella privata. In questo modo chi entrerà in possesso della chiave pubblica non sarà in grado di ricavarne la chiave privata.

Nel caso dell'RSA, la funzione unidirezionale è la seguente. Prendiamo una coppia di primi p e q molto grandi, e scelti secondo un criterio (mostrato più avanti); calcoliamo $n = p \cdot q$, e a questo punto avremo un intero composto computazionalmente impossibile da fattorizzare. Il prodotto dei due primi costituisce una funzione invertibile, e se essi sono scelti correttamente è anche unidirezionale; il composto n sarà pubblico, e i primi p, q saranno mantenuti segreti.

6.2. Algoritmo

Prendiamo un composto $n = p \cdot q$, costituito da due primi p e q scelti adeguatamente; prendiamo un intero e, che chiamiamo esponente di cifratura, tale che $e \perp \varphi(n)$. Dato che n è costituito da due primi, vale $\varphi(n) = (p-1)(q-1)$.

Il messaggio in chiaro m e quello cifrato c sono entrambi elementi di $\mathbb{Z}_n$; le funzioni di cifratura e decifratura sono definite come:

$$\begin{aligned} c &\equiv m^e \pmod{n} \\ m &\equiv c^d \pmod{n} \end{aligned} \tag{6.2.1}$$

L'intero d, chiamato esponente di decifratura, è legato a e nel modo seguente:

$$e \cdot d \equiv 1 \pmod{\varphi(n)} \implies \boxed{d \equiv e^{-1} \pmod{\varphi(n)}}$$

Dalla congruenza con 1 modulo $\varphi(n)$ e il prodotto degli esponenti di cifratura, si deduce che:

$$e \cdot d = 1 + k \cdot \varphi(n)$$

A questo punto dalla funzione di decifratura (6.2.1) otteniamo:

$$c^d = m^{e \cdot d} = m^{1 + k \cdot \varphi(n)} \overset{*}{=} \boxed{m} \cdot 1^k$$

Nella relazione $*$ abbiamo usato (1.7.3); si è voluto mostrare che esiste una corrispondenza univoca tra m e c, per l'ipotesi che $e \perp \varphi(n)$ (esiste un solo d che sia inverso di e in modulo $\varphi(n)$, e vice versa).

La *chiave pubblica* è costituita da n, e mentre la *chiave privata* è d, insieme a p, q oppure a $\varphi(n)$.

ESEMPIO 6.1. *Siano dati due primi* $p_{\mathbb{S}} = 3$, $q_{\mathbb{S}} = 11$ *e un intero* $e_{\mathbb{P}} = 7$, *cifrare tramite RSA il messaggio* $m = 19$.

All'interno di questo esempio, abbiamo usato i pedici $_{\mathbb{P}}$ e $_{\mathbb{S}}$ per indicare un elemento della chiave pubblica o della chiave privata

✓Calcoliamo il composto $n_{\mathbb{P}} = p \cdot q = 33$; calcoliamo inoltre $\varphi(n)_{\mathbb{S}} = \varphi(33) = (3-1)(11-1) = 2 \cdot 10 = 20$. A questo punto possiamo calcolare d:

$$d \equiv e^{-1} \pmod{\varphi(n)}$$
$$d \equiv e^{\varphi(\varphi(n))-1} \pmod{\varphi(n)}$$
$$d \equiv 7^7 \pmod{20}$$
$$\boxed{d_{\mathbb{S}} \equiv 3 \pmod{20}}$$

Nella precedente relazione abbiamo usato (1.7.4), e abbiamo calcolato $\varphi(\varphi(n)) = (2^2-2)(5-1) = 8$.

Cifriamo il messaggio $m = 19$ applicando (6.2.1):

$$c = 19^7 \bmod 33 = 13$$

Si noti che possiamo riottenere $m = 13^3 \bmod 33 = 19$. □

OSSERVAZIONE 6.1. Per effettuare la decifratura (operazione adibita alla chiave privata) è necessario l'esponente d, il quale a sua volta viene calcolato tramite $\varphi(n)$, che a sua volta deriva dai due primi p, q. Ebbene, la conoscenza dei primi p, q o del toziente del loro prodotto $\varphi(n)$, compromette nello stesso modo la sicurezza dell'algoritmo, e permette di calcolare d (la chiave privata).

ESEMPIO 6.2. *Sia dato il composto $n = p \cdot q = 11413$, prodotto di due primi, e il suo toziente sia noto $\varphi(n) = (p-1)(q-1) = 11200$. Calcolare i due primi p e q.*

✓Analizziamo la relazione $n - \varphi(n) + 1$, con le seguenti riscritture:

$$n - \varphi(n) + 1 = p \cdot q - (p-1)(q-1) + 1$$
$$214 = p + q$$

Con la conoscenza della somma dei primi cercati, possiamo risolvere la seguente equazione:

$$(x-p)(x-q) = 0$$
$$x^2 - \overbrace{(p+q)}^{n-\varphi(n)+1} x + \overbrace{p \cdot q}^{n} = 0$$
$$x^2 - 214x + 11413$$

Nella formula risolutiva deve valere $\Delta > 0$, altrimenti n e $\varphi(n)$ non sarebbero corretti

Otteniamo le seguenti soluzioni:

$$x = \frac{214 \pm \sqrt{214^2 - 4 \cdot 11413}}{2} = \{101, 113\}$$

Abbiamo ottenuto $p = 101$, $q = 113$. Avremmo ottenuto il risultato con una bassa complessità computazionale (risolvere un'equazione di secondo grado), anche nel caso di p e q molto grandi. □

6.3. Sicurezza di RSA

In teoria, l'RSA correttamente implementato è sicuro, fintanto che la scomposizione in fattori primi molto grandi rimane un problema difficile; tuttavia esistono delle insidie che influenzano la corretta realizzazione, di cui parleremo in questa sezione.

Debolezza sulle cifre dei primi in RSA

TEOREMA 6.1. *Sia m il numero di cifre di $n = p \cdot q$ ($m = \lceil \log_{10}(p \cdot q) \rceil$); se si conoscono almeno le prime o le ultime $m/4$ cifre di uno dei due primi p o q, allora è possibile fattorizzare in modo efficiente n.*

OSSERVAZIONE 6.2. Se n ha m cifre, e p e q sono confrontabili, allora essi avranno ciascuno $m/2$ cifre.

Il Teorema 6.1 può essere sfruttato se il metodo che il mittente utilizza per ricavare i primi p e q è noto e predicibile: provando per tentativi le cifre dei primi in un intorno di candidati, è possibile trovare le prime o le ultime $m/4$ cifre di p o q.

COROLLARIO 6.1. *Consideriamo una chiave pubblica RSA costituita da n, e, sia m il numero di cifre di n; se si conoscono almeno le ultime $m/4$ cifre di d (esponente di decifratura, segreto) allora è possibile fattorizzare in modo efficiente d.*

OSSERVAZIONE 6.3. Dato che $d \equiv e^{-1} \pmod{\varphi(n)}$, esso avrà da 1 a m cifre, per il limite imposto dal modulo di $\varphi(n)$ (il toziente avrà al più il numero di cifre di n).

Abbiamo enunciato nelle sezioni precedenti che la scelta di p e q deve essere fatta prendendo due primi grandi (nella pratica si usano numeri di 150 cifre, ottenendo un n di 300 cifre). per la scelta dell'esponente di cifratura, bisogna evitare invece un e troppo piccolo, il quale favorirebbe la velocità di cifratura ma sicuramente causerebbe la comparsa di un d molto grande. Nella scelta di e bisogna anche tenere a mente l'ipotesi che esso sia $e \perp \varphi(n)$.

Un valore per l'esponente di cifratura comunemente usato e comprovato è il seguente:

$$e_{\text{RSA}} = 2^{16} + 10 = 65537$$

Esso è un numero primo (è altamente improbabile che abbia fattori in comune con $p-1$ o $q-1$, possiamo verificarlo con (1.3.4)); inoltre è facile calcolare il suo valore, tramite square and multiply (si veda la sezione §1.6).

Un altro approccio consiste nello scegliere un d sufficientemente robusto (50 cifre sono sufficenti), e a partire da esso si calcola e effettuando l'inverso in modulo $\varphi(n)$.

6.4. Test di primalità

Normalmente testare la primalità di un numero e fattorizzarlo sono problemi differenti: è molto più semplice determinare che un numero sia primo o meno. Vi sono per esempio numeri molto grandi di cui si sa con certezza che sono composti, ma nessuno è stato in grado di determinare i loro fattori.

6.4.1. Test di Fermat. Richiamiamo il Teorema 1.3: $a \perp p \wedge p$ primo $\implies a^{p-1} \equiv 1 \pmod{p}$; usando questo risultato, ci chiediamo se, preso un n intero da testare e un $a \perp n$ qualunque, valga

Si ricorda che è possibile testare che due interi siano primi relativi ($a \perp b$) tramite Euclide Esteso (1.3.4)

$$a^{n-1} \equiv 1 \pmod{n} \tag{6.4.1}$$

In caso di residuo esattamente 1, possiamo essere abbastanza sicuri che n sia un primo: lo chiameremo *pseudo-primo di Fermat rispetto alla base a*. Se il residuo della congruenza fosse diverso da 1, potremo affermare che n sia composto.

OSSERVAZIONE 6.4. Gli interi primi diventano rarefatti tanto più si cercano con un valore grande, quindi per $n \gg 1$ la probabilità che uno pseudo-primo di Fermat sia davvero un primo è elevata.

ESEMPIO 6.3. *Applicare il test di Fermat all'intero $n = 341$.*

✓Scegliamo una base che sia prima relativa rispetto a n; dato che 341 è dispari, prendiamo $a = 2$. Applichiamo (6.4.1):

$$2^{340} \bmod 341 = 1$$

Possiamo affermare che 341 sia uno pseudo-primo di Fermat rispetto alla base 2; con un'analisi veloce tuttavia, otteniamo che $11 \backslash 341$ (infatti $341 = 11 \cdot 31$).

Gli pseudo-primi di Fermat rispetto alla base 2 sono detti anche *numeri di Poulet, Sarrus o Fermatians*

Ripetiamo il test per un'altra base, rispettando l'ipotesi che sia prima relativa rispetto a n; scegliamo $a' = 3$, dal test si ottiene:

$$3^{340} \bmod 341 = 56 \neq 1$$

Dato che il candidato al test n non passa il test di Fermat per qualunque base a, possiamo affermare che si tratta di un composto. La condizione appena enunciata è rappresentata da:

$$\exists a \in \mathbb{Z} \,:\, a^{n-1} \not\equiv 1 \,(\mathrm{mod}\, n) \implies n \text{ composto}$$

□

Esempio 6.4. Applicare il test di Fermat all'intero $n = 561$.

✓Prendiamo la base $a = 2$, e applichiamo il test:

$$2^{560} \bmod 561 = 1$$

Possiamo quindi affermare che 561 è uno pseudo-primo di Fermat rispetto alla base 2; proviamo a usare un'altra base $a = 3$, in tal caso il test vale:

$$3^{560} \bmod 561 = 375$$

Da quest'ultimo risultato deduciamo che 561 è composto; tuttavia quello che abbiamo appena effettuato non era un test di Fermat! Infatti 3 è un fattore di 561 ($561 = 3 \cdot 11 \cdot 17$); inoltre l'intero 561 appare pseudo-primo per tutte le basi, ad eccezione dei suoi fattori. □

Numeri di Carmichael

Definizione 6.1. Un intero pseudo-primo di Fermat rispetto ad ogni base, ad eccezione di quelle che sono i suoi fattori, è chiamato *pseudo-primo assoluto*, oppure *numero di Carmichael*.

Osservazione 6.5. La distribuzione dei primi di Carmichael è estremamente rarefatta: il primo elemento è proprio 561, il secondo è 41041, ecc... Il numero $\mathcal{C}(x)$ dei primi di Carmichael da 0 a x è limitato superiormente nel modo seguente:

$$\mathcal{C}(x) < x \cdot e^{\frac{-k \cdot \ln(x) \cdot \ln(\ln(\ln x))}{\ln(\ln(x))}} \tag{6.4.2}$$

Osservazione 6.6. Non esistono pseudo-primi che siano forti e assoluti; tuttavia, sia $\mathbb{B}$ un insieme finito di basi, b un suo elemento, e $\#_{\mathrm{F}}(b)$ il numero di pseudo-primi forti rispetto a una base b, allora

$$\forall b \in \mathbb{B} \,:\, \#_{\mathrm{F}}(b) = \infty$$

6.4.2. Test del principio fondamentale. Richiamiamo il principio fondamentale (1.8.1); possiamo usare questo risultato per dedurre:

$$\exists a,\, b \in \mathbb{Z} \,:\, a^2 \equiv b^2 \,(\mathrm{mod}\, n) \wedge a \not\equiv \pm b \,(\mathrm{mod}\, n) \implies n \text{ composto} \tag{6.4.3}$$

Il test (6.4.3) ci fornisce anche la seguente informazione: il massimo comune divisore tra $a - b$ e n è un fattore non banale di n (diverso da 1 e da n), ovvero vale:

$$\mathrm{MCD}(a - b,\, n) \neq \{1,\, \mathrm{n}\}$$

Tramite la conoscenza di un fattore non banale, si può scomporre n e ridurre il problema alla fattorizzazione di un intero più piccolo.

6.4.3. Test di Miller-Rabin. Dato un candidato n da testare, esso viene prima testato con Fermat (6.4.1); se n si rivela uno pseudo-primo di Fermat, viene sottoposto al test del principio fondamentale (6.4.3); gli interi che passano anche il secondo test sono chiamati *pseudo-primi forti* (non sono sicuramente dei primi, ma lo sono con alta probabilità). Gli interi che non passano il test di Miller-Rabin sono sicuramente composti.

Se prendiamo il candidato $n = 561$ dell'Esempio 6.4, esso non passa il test di Miller-Rabin, quindi è uno pseudo-primo assoluto ma non uno pseudo-primo forte.

6.4.4. Densità dei primi. Vogliamo trovare un intero primo che abbia 100 cifre; la densità dei numeri primi intorno a 10^{100} può essere calcolata usando la quantità di primi dal Teorema 1.1:

$$\frac{\#\text{ primi}}{\#\text{ interi}} = \frac{\pi\left(10^{100}\right)}{10^{100}} = \frac{1}{\ln\left(10^{100}\right)} = \frac{1}{100 \cdot \ln\left(10\right)} \simeq \frac{1}{230}$$

Possiamo scrivere in generale, chiamando $\delta(x)$ la densità di primi intorno a x:

Densità di interi primi

$$\delta(x) = \frac{1}{\ln(x)} \tag{6.4.4}$$

La densità calcolata attorno a 10^{100} si traduce nella seguente osservazione: prendendo a caso un intero di 100 cifre, la probabilità che sia primo è circa $1/230$.

6.5. Fattorizzazione

Consideriamo un numero n che sappiamo essere composto (possiamo prendere un intero che non abbia passato il test di Miller-Rabin §6.4.3); possiamo adottare dei metodi algoritmici per trovare i suoi fattori primi, se n soddisfa certe condizioni, da evitare nella scelta dei primi per RSA.

6.5.1. Metodo di fattorizzazione di Fermat. Proviamo ad esprimere il numero che sappiamo essere composto, come differenza di due quadrati; otteniamo:

$$n = x^2 - y^2 = \left(\overbrace{x+y}^{p}\right)\left(\overbrace{x-y}^{q}\right)$$

Costruiamo la sequenza $\{x^2\}$ seguente, e ci fermiamo quando $n + y^2$ è un quadrato perfetto:

Fattorizzazione di Fermat

$$\left\{x^2\right\} = \left\{y \in [0, \infty) \,:\, n + y^2 = x^2\right\} \tag{6.5.1}$$

Una volta trovato un quadrato perfetto, siamo anche a conoscenza della x e della y, con le quali possiamo calcolare i fattori di n: $p = (x + y)$ e $q = (x - y)$.

L'efficienza computazionale di questo metodo dipende da quanto siano vicini i fattori di n; infatti se $n = p \cdot q$ e $p \sim q$, allora avremo un y piccolo che permetterà di arrivare velocemente alla soluzione.

Esempio 6.5. *Fattorizzare l'intero $n = 295927$ col metodo di Fermat.*

✓Costruiamo la successione (6.5.1):

$$295927 + 1 \neq x^2$$
$$295927 + 4 \neq x^2$$
$$295927 + 9 = 544^2$$

Possiamo affermare che $x = 544$, $y = 3$ e vale $n = (544 + 3)(544 - 3)$. Si noti che i fattori ottenuti non sono necessariamente primi. □

6.5.2. Algoritmo di Pollard ($p-1$). Consideriamo un primo p, in tal caso $p-1$ sarà pari, quindi composto:

$$p-1 = 2 \cdot \prod_{i=1}^{i=\dots} q_i$$

Possiamo sfruttare questa osservazione solo quando i fattori q_i sono piccoli: se abbiamo almeno un fattore q_i grande, l'algoritmo di Pollard non semplifica il problema della fattorizzazione.

Dato un intero n da fattorizzare, si scelga una base $a > 1$, poi si sviluppi la successione $\{b_j\}$ come:

$$\begin{gathered} b_1 \equiv a \pmod n \\ b_2 \equiv b_1^2 \pmod n \\ b_3 \equiv b_2^3 \pmod n \\ \vdots \end{gathered}$$

Algoritmo di Pollard

Formalizziamo la successione con le seguenti scritture:

$$b_j \equiv b_{j-1}^j \pmod n, \quad d_j = \text{MCD}\,(b_j - 1,\, n) \tag{6.5.2}$$

Continuiamo per ogni j fino a che $d_j \neq 1$; in tal caso d_j è un fattore non banale di n, e l'algoritmo termina.

ESEMPIO 6.6. *Fattorizzare l'intero $n = 11413$ con l'algoritmo di Pollard.*

✓Scegliamo la base $a = 2$, e costruiamo la successione (6.5.2):

$$\begin{aligned} b_1 &\equiv 2 \pmod{11413}, & d_1 &= \text{MCD}\,(1,\, 11413) = 1 \\ b_2 &\equiv 2^2 \pmod{11413}, & d_2 &= \text{MCD}\,(3,\, 11413) = 1 \\ b_3 &\equiv 4^3 \pmod{11413}, & d_3 &= \text{MCD}\,(63,\, 11413) = 1 \\ b_4 &\equiv 64^4 \pmod{11413}, & d_4 &= \text{MCD}\,(105,\, 11413) = 1 \\ b_5 &\equiv 106^5 \pmod{11413}, & d_5 &= \text{MCD}\,(-309,\, 11413) = 1 \\ b_6 &\equiv -309^6 \pmod{11413}, & d_6 &= \text{MCD}\,(9992,\, 11413) = 1 \\ b_7 &\equiv 9993^7 \pmod{11413}, & d_7 &= \text{MCD}\,(5085,\, 11413) = \boxed{113} \end{aligned}$$

Otteniamo che un fattore non banale di n è proprio $p = d_7 = 113$, mentre il secondo fattore di $q = n/p = 101$. □

CAPITOLO 7

Firma digitale

7.1. Logaritmo discreto

7.1.1. Definizione. Consideriamo la congruenza seguente:

$$\beta \equiv \alpha^x \pmod{p} \tag{7.1.1}$$

dove p è un primo e α, $\beta \in \mathbb{Z}_p^*$; essa è una equazione congruenziale nell'incognita x, a cui dovremo assegnare il valore a cui elevare α per ottenere β.

Dunque, abbiamo un problema analogo a quello del logaritmo; osserviamo inoltre che l'equazione (7.1.1) ha sempre soluzione se la base α è radice primitiva di $\mathbb{Z}_p^*$, infatti in tal caso si ha $\text{ORD}(\alpha) = p - 1$, quindi elevando α a tutti gli interi da 1 a $p - 1$ si ottengono tutti gli elementi di $\mathbb{Z}_p^*$; dal momento che anche β è contenuto in $\mathbb{Z}_p^*$, sarà certo produrre β da una alle potenze di α.

Chiamiamo *logaritmo discreto* questo problema, e lo indichiamo come:

Logaritmo discreto

$$x = \mathcal{L}_\alpha(\beta) \tag{7.1.2}$$

Risolvendo un logaritmo discreto, si ottengono infinite soluzioni — una classe di congruenze del tipo $x \equiv \mathcal{L}_\alpha(\beta) \pmod{\text{ORD}(\alpha)}$; il logaritmo discreto gode delle stesse proprietà dei logaritmi.

ESEMPIO 7.1. *Calcolare la soluzione del logaritmo discreto in base $\alpha = 2$ di $\beta = 9$, nell'insieme dei residui modulo $p = 11$.*

✓Le soluzioni apparterranno alla classe di congruenze:

$$x \equiv \mathcal{L}_2(9) \pmod{11}$$

In questo caso, essendo 10 le possibili soluzioni, possiamo effettuare una esplorazione esaustiva dello spazio delle radici; provando ad elevare 2 a tutti gli interi da 1 a 10 si ottiene:

$$x \equiv 6 \pmod{11}$$

Possiamo porci nello stesso caso, considerando $\alpha = 3$; il problema sarà allora:

$$3^x \equiv 9 \pmod{11} \implies x \equiv \mathcal{L}_3(9) \pmod{11}$$

Effettuando di nuovo un'analisi esaustiva dello spazio delle radici otteniamo che:

$$x \equiv 2 \pmod{5}$$

Notiamo che la classe di residui non è in modulo 11 questa volta, poiché 3 non è una radice primitiva in $\mathbb{Z}_{11}^*$. □

OSSERVAZIONE 7.1. In generale non abbiamo un algoritmo efficiente che risolva il logaritmo discreto per un primo p arbitrario. Quindi la funzione logaritmo discreto z è unidirezionale; infatti, dato β, non si riesce a calcolare x in modo semplice. Questo rende il logaritmo discreto una funzione adatta a essere usata in un cifrario a chiave pubblica.

7.1.2. Parità del logaritmo discreto. Possiamo ottenere se la soluzione di un logaritmo discreto sia pari o meno: consideriamo il primo p, gli interi α, $\beta \in \mathbb{Z}_p^*$ e l'intero $x \in [0,\, p-1]$, tali che:

$$\beta \equiv \alpha^x \,(\mathrm{mod}\, p)$$

Vogliamo trovare il valore di x.

Notiamo che, con la seguente riscrittura, se α è radice primitiva in $\mathbb{Z}_p^*$ (implicazione $\star$), otteniamo una congruenza a 1:

$$\left(\alpha^{\frac{p-1}{2}}\right)^2 \equiv \alpha^{p-1} \equiv 1 \,(\mathrm{mod}\, p) \overset{\star}{\implies} \alpha^{\frac{p-1}{2}} \equiv \pm 1 \,(\mathrm{mod}\, p)$$

Dato che $p-1$ è l'esponente minore che verifica la congruenza a $+1$, consideriamo nella precedente solo "$\equiv -1 \,(\mathrm{mod}\, p)$".

Elevando entrambi i membri della prima congruenza a $(p-1)/2$ si ottiene:

$$\beta^{\frac{p-1}{2}} \equiv \alpha^{\frac{p-1}{2} \cdot x} \equiv (-1)^x \,(\mathrm{mod}\, p)$$

Nella precedente, se abbiamo $\beta^{(p-1)/2} \equiv +1 \,(\mathrm{mod}\, p)$ allora x è pari, altrimenti sarà dispari.

7.1.3. Algoritmo di Pohlig-Hellman. Osserviamo che $p-1$ sarà pari, dato un primo p: allora $p-1$ dovrà essere un composto. Scriviamolo come:

$$p - 1 = 2 \cdot \prod_{i=1}^{h} p_i^{r_i}$$

Dato che la soluzione del logaritmo discreto è della forma $x \equiv \mathcal{L}_\alpha(\beta) \,(\mathrm{mod}\ (p-1))$, possiamo scomporre il problema col Teorema 1.2 in h pezzi diversi, ognuno in modulo $p_i^{r_i}$.

Questi problemi sono più facili da risolvere perché più piccoli; se tuttavia è presente un fattore $p_i^{r_i}$ molto grande, allora non vi è riduzione della complessità; concludiamo che la difficoltà del problema del logaritmo discreto modulo $p-1$ dipende strettamente dalla grandezza del fattore più grande di $p-1$.

7.1.4. Algoritmo baby step, giant step. Consideriamo le ipotesi per il problema descritto dall'equazione (7.1.1), e scegliamo un intero $N : N^2 \geq p-1$; segue che:

$$N = \left\lfloor \sqrt{p-1} \right\rfloor + 1$$

Costruiamo ora due successioni $\{\alpha^j\}$ e $\left\{\beta \cdot \alpha^{-N \cdot k}\right\}$ tali che $0 \leq j < N$ e $0 \leq k < N$; le successioni così fatte saranno sviluppate come:

$\boldsymbol{j}$	$\boldsymbol{\alpha^j}$	$\boldsymbol{k}$	$\boldsymbol{\beta \cdot \alpha^{-N \cdot k}}$
0	1	0	β
1	α	1	$\beta \cdot \alpha^{-N}$
2	α^2	2	$\beta \cdot \alpha^{-2N}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$

Tutti i termini delle successioni α e β sono in modulo p; inoltre, una volta sviluppata la successione α, cerchiamo la prima corrispondenza con uno dei termini della successione β. Una corrispondenza tra le due successioni vuol dire aver trovato dei valori per j e k che verificano:

$$\alpha^j \equiv \beta \cdot \alpha^{-N \cdot k} \,(\mathrm{mod}\, p) \implies \alpha^{j+N \cdot k} \equiv \beta \,(\mathrm{mod}\, p)$$

Per il principio fondamentale, dalla precedente possiamo scrivere la congruenza degli esponenti modulo $p-1$, trovando la risposta al problema del logaritmo discreto (a quale esponente elevare α per ottenere β):

$$x \equiv j + N \cdot k \,(\mathrm{mod}\ (p-1))$$

Osservazione 7.2. Con questo algoritmo esprimiamo il valore del logaritmo discreto in base N; la complessità computazionale è proporzionale a $\sqrt{p}$, rispetto all'analisi esaustiva delle soluzioni del logaritmo discreto, che può essere condotta con $p-1$ elevamenti a potenza. Infatti dovremo certamente sviluppare la successione di α fino a $j = N-1$, e nel peggiore dei casi anche la successione di β fino a $k = N-1$.

7.2. Pattuizione della chiave Diffie-Hellman

Il metodo Diffie-Hellman permette di scambiare una chiave simmetrica attraverso un canale non sicuro, sfruttando la complessità del problema del logaritmo discreto. Consideriamo un mittente **A** e un destinatario **B**; ipotizziamo che un attaccante **O** possa osservare tutto il traffico che transita sul canale tra **A** e **B**; nel seguente elenco sono indicati i messaggi scambiati tra **A** e **B**:

(1) $\mathbf{A} \leftrightarrows \mathbf{B}$: mittente e destinatario concordano un primo p e una radice primitiva $\alpha \in \mathbb{Z}_p^*$ (sia α che p sono pubblici);
(2) $\mathbf{A} \to \mathbf{B}$: **A** sceglie a caso un intero $x \in [1,\, p-2]$, e invia a **B** $\alpha^x \bmod p$;
(3) $\mathbf{A} \leftarrow \mathbf{B}$: **B** sceglie a caso un intero $y \in [1,\, p-2]$, e invia ad **A** $\alpha^y \bmod p$;
(4) $K_{\mathbf{A}\to\mathbf{B}}$: la chiave che **A** usa per scrivere a **B** è costruita come $(\alpha^y)^x \bmod p$;
(5) $K_{\mathbf{A}\leftarrow\mathbf{B}}$: la chiave che **B** usa per rispondere ad **A** è costruita come $(\alpha^x)^y \bmod p$.

Si verifica facilmente che le chiavi generate con questo algoritmo sono uguali:

$$K_{\mathbf{A}\to\mathbf{B}} \equiv K_{\mathbf{A}\leftarrow\mathbf{B}} \equiv \alpha^{x\cdot y} \,(\mathrm{mod}\, p)$$

Consideriamo il punto di vista di un attaccante **O** che osserva il traffico sul canale: egli non potrà ricavare x e y a meno che non risolva il problema del logaritmo discreto; è quindi necessario concordare un primo p che renda intrattabile il problema del logaritmo discreto in $\mathbb{Z}_p^*$.

Problema 7.1. Problema computazionale di Diffie-Hellman. Dati α^x e α^y, entrambi residui modulo p, calcolare $\alpha^{x\cdot y} \bmod p$ non è più difficile di calcolare il logaritmo discreto $\mathcal{L}_{\mathrm{x\cdot y}}(\alpha)$. Risolvere il logaritmo discreto è quindi condizione sufficiente per il problema computazionale di Diffie-Hellman.

Problema 7.2. Problema decisionale di Diffie-Hellman. Dati α^x, α^y e un intero c, tutti residui modulo p, determinare se si verifica $c \equiv \alpha^{x\cdot y} \,(\mathrm{mod}\, p)$. Risolvere il problema computazionale permette di risolvere anche quello decisionale, tuttavia non è vero il contrario: non sappiamo se risolvere il problema decisionale possa ridurre a priori la complessità di quello computazionale.

7.3. Crittosistema a chiave pubblica El Gamal

7.3.1. Algoritmo. Consideriamo un mittente **A** e un destinatario **B**; ipotizziamo che un attaccante **O** possa osservare tutto il traffico che transita sul canale tra **A** e **B**, e il messaggio m che **A** desidera inviare a **B**:

(1) **B**: sceglie un primo p tale che il problema del logaritmo discreto in $\mathbb{Z}_p^*$ sia intrattabile e una radice primitiva α in tale insieme (pubblici); inoltre sceglie un intero $a \in [1,\, p-2]$ (segreto); infine, egli computa $\beta \equiv \alpha^a \,(\mathrm{mod}\, p)$;
(2) $\mathbf{A} \leftarrow \mathbf{B}$: **B** invia ad **A** la propria chiave pubblica, costituita da $(\alpha,\, \beta,\, p)$;

(3) **A**: sceglie un intero $k \in [1,\, p-2]$, e calcola:

$$r \equiv \alpha^k \,(\mathrm{mod}\, p)\,, \quad t \equiv \beta^k \cdot m \,(\mathrm{mod}\, p)$$

(4) $\mathbf{A} \to \mathbf{B}$: **A** invia il messaggio cifrato costituito da $(r,\, t)$ a **B**;
(5) **B**: decifra il messaggio ricevuto computando $t \cdot r^{-a} \bmod p$:

$$t \cdot r^{-a} \equiv \left(\beta^k \cdot m\right)\left(\alpha^k\right)^{-a} \equiv \left(\cancel{\alpha^{ak}} \cdot m\right)\left(\cancel{\alpha^{ak}}\right) \equiv \boxed{m} \,(\mathrm{mod}\, p)\,.$$

Osservazione 7.3. Mentre a rimane il segreto di **B** fin dall'inizio della comunicazione, il segreto k di **A** invece è casuale ed è bene che sia usato una sola volta; esso è un segreto effimero chiamato *nonce* (dalla crasi di number e once, un numero da usare una sola volta).

Osservazione 7.4. Notiamo che il messaggio in chiaro m appartiene a $\mathbb{Z}_p^*$, tuttavia il messaggio cifrato è dato dalla coppia di numeri $(r,\, t) \in \mathbb{Z}_p^* \times \mathbb{Z}_p^*$.

Osservazione 7.5. Possiamo calcolare r^{-a} sia passando per l'inverso di r elevato ad a, sia considerando che $r^{-a} \equiv r^{p-1-a} \,(\mathrm{mod}\, p)$, per il Teorema 1.3.

7.3.2. Attacco del nonce ripetuto. Ipotizziamo di utilizzare lo stesso nonce due volte; chiamiamo m_i il testo in chiaro, c_i il messaggio cifrato, e $m_1 \neq m_2$, allora dalla cifratura si ottiene:

$$E\,(m_1) = c_1\,(r,\, t_1)\,, \quad E\,(m_2) = c_2\,(r,\, t_2)$$

I due interi r per i messaggi cifrati coincidono, e questo permette all'attaccante (se egli possiede anche un testo in chiaro noto, per esempio m_1 associato a c_1) di provare a computare:

$$\frac{t_1}{m_1} \equiv \frac{t_2}{m_2} \,(\mathrm{mod}\, p) \implies m_2 \equiv \frac{t_2}{t_1} \cdot m_1 \,(\mathrm{mod}\, p)$$

Dalla precedente congruenza è possibile ricavare tutti i messaggi successivi alla ripetizione di un nonce, sfruttando la relazione di proporzionalità mostrata.

Osservazione 7.6. Se avessimo un dispositivo che risolve il Problema 7.1, potremmo usarlo per decifrare i messaggi cifrati tramite il crittosistema di El Gamal; se avessimo un dispositivo che risolve il Problema 7.2, potremmo usarlo per decidere se un messaggio cifrato $(r,\, t)$ sia compatibile con un messaggio in chiaro m (permette di attuare l'attacco del testo in chiaro noto), non ostante il segreto effimero k. Questa è la differenza con l'RSA nel quale, dato un messaggio cifrato e uno in chiaro, è possibile verificare direttamente la loro corrispondenza.

7.4. Funzioni di Hash

7.4.1. Definizione e proprietà. Una funzione di hash è una funzione matematica che prende in ingresso un messaggio di lunghezza arbitraria, e produce una uscita (di solito corta) di lunghezza prefissata.

Le funzioni di hash sono state introdotte nella sottosezione §3.5.1

La funzione matematica selezionata deve:

- essere veloce da calcolare;
- avere uscita pseudo-casuale (deve godere delle proprietà descritte dalla Proposizione 2.1);
- essere non-invertibile (non c'è corrispondenza univoca tra ingresso e uscita, risulta non-invertibile per definizione);
- essere unidirezionale (non deve essere possibile trovare nemmeno una uscita delle infinite che producono un certo ingresso);

La proprietà di unidirezionalità è anche detta *resistenza alla contro-immagine*

- sia computazionalmente intrattabile il problema di trovare una coppia di messaggi differenti $m_1 \neq m_2$, tali che i loro hash siano uguali, ovvero (detta $h(m)$ la funzione di hash del messaggio m)
$$\forall (m_1, m_2) : m_1 \neq m_2 \implies h(m_1) \neq h(m_2)$$
se questo si verifica, diremo che la funzione $h()$ è *fortemente resistente alle collisioni*;
- dato un messaggio m, sia computazionalmente intrattabile il problema di trovare un messaggio diverso $m' \neq m$ per il quale gli hash dei due messaggi siano uguali, ovvero
$$\forall m \nexists m' : m' \neq m \implies h(m) = h(m')$$
se questo si verifica, diremo che la funzione $h()$ è *debolmente resistente alle collisioni*.

Ottenere la resistenza debole alle collisioni è più difficile: infatti essa fornisce un grado di libertà in meno rispetto alla resistenza forte (uno dei due messaggi viene fissato).

ESEMPIO 7.2. *Determinare se le seguenti funzioni siano invertibili, unidirezionali e prive di collisioni:*

(1) $h(x) = x \bmod n$ con n intero composto qualunque;
(2) $h(x) = \alpha^x \bmod p$ con α radice primitiva di $\mathbb{Z}_p$;
(3) $h(x) = x^2 \bmod n$ con $n = p \cdot q$ prodotto di due primi;
(4) $h(x) = \text{DES}_x(000\ldots)$ con chiave $x \in [0, 2^{56}]$.

✓Analizziamo una alla volta le quattro funzioni, nelle sottosezioni seguenti.

Funzione (1). Questa funzione non è invertibile, poiché esistono infiniti messaggi che producono la stessa uscita della funzione ($\forall k : h(x) = (x + k \cdot n) \bmod n$); per quanto appena osservato, tale funzione non è unidirezionale, poiché è possibile trovare uno qualunque di quegli infiniti messaggi che producono la stessa uscita (basta trovare un $x + k \cdot n$). Per lo stesso motivo, è possibile trovare una collisione (per esempio $h(x) = h(x + n)$).

Funzione (2). Questa funzione non è invertibile, poiché esistono infiniti numeri nella classe di congruenza $k \equiv \overline{x} \pmod{(p-1)}$ che danno resto k; l'inverso di questa funzione ricade nel problema del logaritmo discreto (7.1.2): con un p scelto in modo opportuno, la funzione risulta unidirezionale.

Possiamo ottenere una collisione prendendo x_1 e $x_2 = x_1 + k(p-1)$, da cui si ottiene la stessa uscita ($h(x_1) = h(x_2)$).

Funzione (3). Questa funzione non è invertibile, poiché esistono infiniti valori di x per cui la congruenza in modulo n possa essere soddisfatta; presi p e q tali da rendere computazionalmente intrattabile la fattorizzazione di n, questa funzione risulta unidirezionale.

Possiamo ottenere una collisione usando il fatto che un quadrato abbia due radici opposte: $h(x) = h(-x)$.

Funzione (4). Cifriamo un blocco di 64 zeri col DES, usando il messaggio come chiave di 56bit. Dato che il messaggio usato come chiave è da 56bit, e l'uscita è un blocco da 64bit, la funzione non è invertibile; grazie alle proprietà del DES la funzione risulta unidirezionale (non è possibile ottenere la chiave x che produca un'uscita $h(x)$ desiderata), infatti DES resiste all'attacco del testo in chiaro noto.

Per quanto appena affermato, non è possibile trovare collisioni (equivale a trovare due chiavi che cifrano nello stesso modo lo stesso blocco).

Osserviamo che non possiamo usare questa funzione come una funzione di hash: infatti essa non rispetta la proprietà sulla lunghezza arbitraria dell'ingresso, che è limitato a un massimo di 56bit. □

7.4.2. Algoritmo SHA. Si tratta di una famiglia di algoritmi classificati, in base alla lunghezza dell'uscita, la lunghezza dei blocchi e il numero di round, in SHA-0 (160bit), SHA-1 (160bit), SHA-2 (256bit), SHA-3 (512bit, standard approvato dal NIST). Il loro funzionamento può essere schematizzato nel modo seguente; sia dato un messaggio m, di cui si vuole calcolare l'hash:

Inizializzazione

- concatena al messaggio m dei bit di "padding", in modo che la sua lunghezza ℓ sia inferiore di 64 bit rispetto ad un multiplo di 512;
- concatena al messaggio m la sua lunghezza ℓ, sotto forma di intero senza segno a 64bit (questo pone un limite intrinseco alla lunghezza dell'ingresso), rendendo il tutto di lunghezza pari ad un multiplo di 512;
- inizializza 5 registri $H_0, \ldots, H_4$ con dei valori costanti (sono parte della specifica dell'algoritmo).

Esecuzione

(1) dividiamo il messaggio m in blocchi da 512bit chiamati M_i;
(2) dividiamo ciascun blocco M_i in 16 blocchi da 32bit, che chiamiamo $W_{i,j}$;
(3) per j da 16 a 79:
  (a) $W_{i,j} = (W_{i,j-3} \oplus W_{i,j-8} \oplus W_{i,j-14} \oplus W_{i,j-16}) \hookleftarrow 1$;
(4) assegniamo $A = H_0$, $B = H_1$, $C = H_2$, $D = H_3$, $E = H_4$;
(5) per j da 0 a 79:
  (a) $T = (A \hookleftarrow 5) + f_j(B, C, D) + E + W_j + K_j$;
  (b) $E = D$;
  (c) $D = C$;
  (d) $C = (B \hookleftarrow 30)$;
  (e) $B = A$;
  (f) $A = T$;
(6) assegniamo $H_0 = H_0 + A$, $H_1 = H_1 + B$, $H_2 = H_2 + C$, $H_3 = H_3 + D$, $H_4 = H_4 + E$;
(7) il risultato dell'algoritmo è la stringa $H_0 \parallel H_1 \parallel H_2 \parallel H_3 \parallel H_4$, il valore a 160bit dell'hash.

La funzione $f_j()$ e il valore K_j sono specifici dell'algoritmo SHA-1; la funzione $B \hookleftarrow r$ fa scorrere in modo ciclico i bit di B a sinistra di r posizioni

7.4.3. Paradosso del compleanno. Consideriamo una coppia di persone: la probabilità che esse non abbiano il compleanno nello stesso giorno è pari a $\frac{364}{365} \simeq 99.7\%$ (stiamo escludendo gli anni bisestili). Prendiamo ora un campione di 100 persone; esse possono formare in totale $100 \cdot 99 = 990$ coppie differenti; allora la probabilità che nessuna delle coppie abbia il compleanno nello stesso giorno si ottiene come:

$$\left(\frac{364}{365}\right)^{990} \simeq 6.6\%$$

Calcoliamo ora la probabilità che, all'interno del campione di 100 persone, almeno due abbiano il compleanno nello stesso giorno:

$$1-\left(1-\frac{1}{365}\right)\cdot\left(1-\frac{2}{365}\right)\cdot\ldots\cdot\left(1-\frac{99}{365}\right) \underset{N\gg r}{\sim} 1-e^{-r^2/2N} \to 1-e^{-10000/730} \simeq 0.999999$$

Nel caso in esame si ha $r = 100$ e $N = 365$; se invece vogliamo ottenere il numero di persone r che renda $1/2$ la probabilità di avere almeno una coppia con lo stesso

compleanno, possiamo usare al contrario la formula approssimata appena enunciata:

$$1 - e^{-r^2/2N} = 0.5 \to \frac{r^2}{2N} = \ln(2) \to r \simeq 1.17\sqrt{N}$$

Quando r è circa uguale alla radice di N, la probabilità di collisioni è superiore al 50%.

Consideriamo un terzo caso, più rilevante dal punto di vista crittografico; prendiamo due insiemi di r persone, ognuna delle quali ha il compleanno in uno degli N giorni dell'anno (le ripetizioni vengano considerate): si dimostra che la probabilità che vi sia una corrispondenza per i compleanni di una coppia di persone, ciascuna da un gruppo differente, è pari a:

Collisione nel paradosso del compleanno

$$1 - e^{-r^2/N} \tag{7.4.1}$$

Possiamo adottare questo principio per attaccare la firma digitale tramite hash. Sia data una funzione di hash con lunghezza 60bit: vi saranno in totale $2^{60} \simeq 10^{19} = N$ "compleanni" (hash possibili); la probabilità di ottenere un hash desiderato scegliendo il messaggio è pari a $1/2^{60}$ (pressoché nulla).

Se prendiamo due insiemi di r documenti, e calcoliamo l'hash di ciascuno di essi, troveremo una collisione al 50% di probabilità se r vale:

$$r \simeq \sqrt{2^{60}} = 2^{30} \simeq 10^{10}$$

In pratica, nel caso di un hash a 60bit, possiamo generare dieci miliardi di variazioni per un messaggio legittimo e un miliardo di variazioni per un messaggio malevolo (modificando virgole, spazi o lettere maiuscole, in modo che un osservatore distratto non noti la differenza); dalla (7.4.1) sappiamo che otterremo una collisione dell'hash con probabilità circa 63.2%; in questo modo è possibile ottenere un messaggio malevolo che appaia firmato dall'autore del messaggio legittimo.

ESEMPIO 7.3. *Sia $h_n(x)$ una funzione di hash con ingresso x e lunghezza dell'uscita n (espressa in bit). Calcolare le probabilità di collisione per $h_4(x)$, $h_8(x)$ e $h_{12}(x)$.*

✓Nella seguente tabella sono riassunte le considerazioni per le tre funzioni di hash:

$$\begin{array}{llll} h_4(x) \implies 16 \text{ hash} & \mathbb{P}(\text{collisione}) = \frac{1}{16} & 1 - \left(1 - \frac{1}{16}\right)^{16} = 0.6439 \\ h_8(x) \implies 256 \text{ hash} & \mathbb{P}(\text{collisione}) = \frac{1}{256} & 1 - \left(1 - \frac{1}{256}\right)^{256} = 0.6328 \\ h_{12}(x) \implies 4096 \text{ hash} & \mathbb{P}(\text{collisione}) = \frac{1}{4096} & 1 - \left(1 - \frac{1}{4096}\right)^{4096} = 0.6321 \end{array}$$

Nella terza colonna è stata valutata la probabilità p di trovare la collisione desiderata in 2^n tentativi (con n il numero di bit dell'uscita della funzione); osservando i valori di probabilità ottenuti, possiamo dimostrare che essi tendono a un valore limite, che possiamo descrivere nel seguente modo:

$$\lim_{n\to\infty} \left(1 - \frac{1}{n}\right)^n \to \frac{1}{e} \implies p = 1 - \frac{1}{e}$$

□

7.4.4. Attacco del compleanno al logaritmo discreto. Riprendiamo le ipotesi del problema del logaritmo discreto (7.1.2):

$$\alpha^x \equiv \beta \pmod{p} \implies x \equiv \mathcal{L}_\alpha(\beta) \pmod{p}$$

Per risolvere il logaritmo discreto abbiamo introdotto l'algoritmo "baby step, giant step" (§7.1.4), che consiste nel generare due liste lunghe al più $\sqrt{p}$; attuiamo l'attacco

del compleanno costruendo prima di tutto due liste: la prima contiene $\{\alpha^k \bmod p\}$, la seconda $\{\beta \cdot \alpha^{-\ell} \bmod p\}$, dove $k, \ell \in \mathbb{Z}_{p-1}^*$ sono scelti a caso.

Troviamo una corrispondenza tra le due liste al 50% di probabilità, calcolando $\sqrt{p}$ valori possibili per k ed ℓ; indicando con $\overline{k}$ ed $\overline{\ell}$ i valori che permettono di ottenere una corrispondenza tra le liste, possiamo scrivere:

$$\alpha^{\overline{k}} \equiv \beta \cdot \alpha^{-\overline{\ell}} \pmod p \implies \beta \equiv \alpha^{\overline{k}+\overline{\ell}} \pmod p \implies \overline{k} + \overline{\ell} \equiv \mathcal{L}_\alpha(\beta) \pmod p$$

In pratica possiamo affermare che il valore $\overline{k} + \overline{\ell}$ sia il valore che risolve il logaritmo discreto cercato, tuttavia osserviamo che esso non è più efficace dell'algoritmo "baby step, giant step"; infatti, per applicare tale algoritmo non è necessario compilare completamente la seconda lista, ma basta fermarsi alla prima corrispondenza, inoltre i valori di $\overline{k}$ ed $\overline{\ell}$ sono casuali e non possono essere computati progressivamente. Infine, la probabilità trovare una corrispondenza non è migliore di quella ottenuta dall'applicazione del "baby step, giant step".

7.5. Firma digitale

La firma digitale è una stringa di bit associata al messaggio da firmare, che include un segreto del firmatario. In pratica si utilizza lo hash del messaggio da firmare, al quale viene aggiunto un segreto del firmatario, cosicché solo chi è a conoscenza del segreto (il firmatario del messaggio) è in grado di creare la firma associata; inoltre la firma deve essere verificabile pubblicamente — è l'opposto della cifratura in un crittosistema a chiave pubblica: firmando con la chiave segreta si permette a tutti di verificare la firma decifrandola con la chiave pubblica; questo garantisce che solo chi è in possesso della chiave privata associata alla firma, possa generarla.

Si firma l'hash del messaggio per evitare di avere una firma molto lunga (pari o maggiore della lunghezza del messaggio) e questo non compromette la sicurezza della firma, per le proprietà della funzione di hash.

Autenticazione di questo tipo viene classificata come *HMAC*: Hash-based Message Autentication Code

Altre applicazioni delle funzioni di hash sono il controllo di integrità (cambiando anche un bit di un messaggio, il suo hash cambia) e l'autenticazione del messaggio (si usa una chiave simmetrica insieme al messaggio, per generare l'hash).

7.5.1. Firma RSA. Consideriamo due parti **A** e **B**, e **B** voglia inviare un messaggio m ad **A** in modo che lo firmi. L'algoritmo per la firma RSA in questo caso si svolge come segue:

- **A** pubblica n ed e_A, mantiene segreto d_A
  - sceglie due primi grandi p, q e li usa per calcolare $n = p \cdot q$ in modo da rendere questo intero computazionalmente non fattorizzabile;
  - sceglie la sua chiave di cifratura pubblica come $e_A \in (1, \varphi(n))$;
  - crea la sua chiave di decifratura come $d_A \in (1, \varphi(n))$, che deve essere l'inverso di e_A ($d_A \cdot e_A \equiv 1 \pmod{\varphi(n)}$);
- **B** invia ad **A** il messaggio m da firmare;
- **A** firma il messaggio, producendo $\text{SIG}_A(m) = m^{d_A} \bmod n$;
- **B** verifica la firma di **A** provando che valga $\text{SIG}_A(m)^{e_A} \bmod n \stackrel{?}{=} m$.

Negli intervalli di definizione, 1 e $\varphi(n)$ sono esclusi poiché e_A e d_A sono esponenti (elevando a 1 non cambia la base, ed elevare al toziente di n un residuo modulo n dà 1)

OSSERVAZIONE 7.7. In RSA, per decifrare un messaggio cifrato da **A**, conoscendo solo i dati pubblici n ed e_A, dobbiamo ottenere l'esponente di decifratura $d_A \equiv e_A^{-1} \pmod{\varphi(n)}$; per calcolare questo inverso possiamo usare (1.7.3) oppure (1.3.4) ma in entrambi i casi avremo bisogno di $\varphi(n) = (p-1)(q-1)$.

La firma digitale RSA funziona nel modo inverso, usando l'esponente di decifratura per firmare e quello di cifratura per verificare la firma.

OSSERVAZIONE 7.8. Questo tipo di firma nasconde il messaggio al suo interno; inoltre intercettando una qualunque firma $\text{SIG}_\text{A}(m)$ ed elevandola ad e_A modulo n, otteniamo un messaggio valido firmato da A; tuttavia esso risulterà una serie casuale di bit.

7.5.2. Firma cieca. Consideriamo la seguente variazione dell'algoritmo della firma digitale RSA. **B** vuole che **A** firmi un messaggio m, senza che **A** conosca il suo contenuto:

- **A** pubblica n ed e_A, mantiene segreto d_A (tutti e tre gli interi sono costruiti come in §7.5.1);
- **B** sceglie un intero casuale $k \perp n$, da usare una volta sola (nonce);
  - **B** calcola $t \equiv m \cdot k^{e_\text{A}} \pmod n$;
  - **B** invia ad **A** il messaggio cifrato t;
- **A** firma il messaggio calcolando $s \equiv t^{d_\text{A}} \pmod n = \text{SIG}_\text{A}(t)$;
  - **A** invia la firma s a **B**;
- **B** verifica la firma calcolando $s \cdot k^{-1} \bmod n$, ottenendo

$$\overbrace{m^{d_\text{A}} \cdot k^{e_\text{A} \cdot d_\text{A}}}^{s} \cdot k^{-1} \equiv m^{d_\text{A}} \cdot \cancel{k} \cdot \cancel{k^{-1}} \pmod n$$

  ovvero **B** computa proprio la firma di **A** del messaggio m (secondo l'algoritmo della firma digitale RSA).

7.5.3. Firma di El Gamal. Seguendo l'applicazione della firma RSA, l'algoritmo per la firma di El Gamal si applica firmando con la chiave privata e verificando la firma tramite la chiave pubblica; in questo caso la chiave privata può essere ottenuta a partire da quella pubblica solo risolvendo il problema del logaritmo discreto.

Consideriamo le stesse premesse delle sezioni precedenti — **B** ha un messaggio m che vuole far firmare ad **A** (se il messaggio è più lungo di p, deve essere spezzato blocchi di dimensione p, dato che l'algoritmo opera in $\mathbb{Z}_p^*$):

- **A** pubblica un primo p e un intero α;
  - p tale da rendere intrattabile il problema del logaritmo discreto in $\mathbb{Z}_p^*$;
  - α sia una radice primitiva di $\mathbb{Z}_p^*$ (per garantire l'esistenza di una soluzione al problema del logaritmo discreto);
- **A** sceglie un esponente intero segreto $a \in (1,\, p-2)$;
  - **A** calcola e pubblica $\beta \equiv \alpha^a \pmod p$ (nella pratica è impossibile ottenere il segreto a risolvendo questa equazione con un logaritmo discreto);
- **A** sceglie un esponente casuale $k \in (1,\, p-2)$ tale che $k \perp p-1$, da considerarsi come nonce;
  - A produce la firma $\text{SIG}_\text{A}(m) = (r,\, s)$;
  - $r \equiv \alpha^k \pmod p$;
  - $s \equiv k^{-1}(m - a \cdot r) \pmod{(p-1)}$;
- **B** verifica la firma $(r,\, s)$ usando solo dati pubblici;
  - la firma risulta corretta se si verifica la congruenza $\beta^r \cdot r^s \stackrel{?}{\equiv} \alpha^m \pmod p$;

Mostriamo ora come la congruenza appena enunciata sia una verifica corretta per la firma di El Gamal; dalla definizione di s, moltiplichiamo per k entrambi i lati della

congruenza:

$$k \cdot s \equiv k \cdot k^{-1} (m - a \cdot r) \pmod{(p-1)}$$
$$m \equiv k \cdot s + a \cdot r \pmod{(p-1)}$$
$$\alpha^m \overset{\star}{\equiv} \overbrace{\alpha^{k \cdot s + a \cdot r}}^{*} \pmod{p}$$
$$\overbrace{\underbrace{(\alpha^a)^r}_{\blacktriangle} \cdot \underbrace{(\alpha^k)^s}_{\blacksquare}}^{*} \equiv \overbrace{\beta^r}^{\blacktriangle} \cdot \overbrace{r^s}^{\blacksquare} \pmod{p}$$
$$\alpha^m \equiv \beta^r \cdot r^s \pmod{p}$$

Nella relazione $\star$ abbiamo usato (1.8.1), dato che gli esponenti di α sono congruenti in modulo p; l'ultima congruenza è proprio quella che verifica la firma di El Gamal.

Osservazione 7.9. La firma generata in questo modo non è sempre la stessa, fissato il messaggio: infatti essa dipende dal segreto effimero k; quindi ci sono $p-2$ firme valide per lo stesso messaggio m. Rispetto alla firma cieca, in questo modo non è neanche possibile capire se è lo stesso messaggio ad essere firmato due volte.

Osservazione 7.10. La firma RSA è lunga quanto il messaggio firmato; la firma di El Gamal è lunga il doppio del messaggio firmato (essa è espressa da due interi r ed s, con lo stesso ordine di grandezza); questo riconferma la necessità di firmare l'hash del messaggio e non il messaggio stesso, altrimenti la firma avrebbe una dimensione troppo grande rispetto al messaggio.

7.5.4. Attacco del nonce ripetuto. Un attaccante è in grado di firmare con El Gamal un messaggio al posto di **A** in modo che **B** possa comunque verificare correttamente la sua firma; questo accade quando **A** utilizza due volte lo stesso k per firmare.

Dal punto di vista di un osservatore esterno, quando A emette due firme differenti $\text{SIG}_\text{A}(m_1) = (r_1, s_1) \neq \text{SIG}_\text{A}(m_2) = (r_2, s_2)$ tali che $r_1 = r_2 = r'$, si può ricavare il nonce k nel modo seguente:

$$\begin{cases} -a \cdot r' \equiv s_1 \cdot k - m_1 \pmod{(p-1)} \\ -a \cdot r' \equiv s_2 \cdot k - m_2 \pmod{(p-1)} \end{cases} \implies (s_1 - s_2)\, k \equiv m_1 - m_2 \pmod{(p-1)}$$

La seconda relazione è il frutto della differenza tra le prime due; possiamo calcolare k dall'equazione lineare in una incognita:

$$k \equiv (m_1 - m_2) \cdot (s_1 - s_2)^{-1} \pmod{(p-1)} \tag{7.5.1}$$

Per risolverla dobbiamo ottenere l'inverso di $s_1 - s_2$ modulo $p-1$ (non è detto che esista): calcoliamo quindi $\text{MCD}(s_1 - s_2, p-1) = d$; esso dovrebbe essere un numero relativamente piccolo e rappresenta il numero di soluzioni dell'equazione.

Una volta ottenuti d valori di k, possiamo ricavare il segreto a risolvendo una equazione congruenziale lineare in una incognita:

$$a \equiv r^{-1} (m - k \cdot s) \pmod{(p-1)}$$

Di nuovo, otterremo non uno ma n possibili valori di a che risolvono l'equazione.

Esercizi

Equazioni congruenziali

Esercizio 7.1. Risolvere la seguente equazione congruenziale:

$$28x \equiv 16 \pmod{412}$$

▶ *Esercizio: equazione congruenziale, modulo composto*

Soluzione 7.1. Procediamo calcolando $\text{MCD}(28, 412) = 4$; allora possiamo dividere per 4 tutti i termini dell'equazione, ottenendo

$$7x \equiv 4 \pmod{103}$$

L'equazione originale avrà 4 soluzioni, e la prima si ricava dall'equazione ridotta appena trovata:

$$x_0 \equiv 4 \cdot 7^{-1} \pmod{103}$$

Per calcolare l'inverso di 7 modulo 103 possiamo usare il Teorema 1.3 oppure l'algoritmo (1.3.4).

Teorema piccolo di Fermat
Il teorema afferma che, in questo caso, vale

$$7^{-1} \equiv 7^{101} \pmod{103}$$

dato che 103 è primo; adesso effettuiamo l'operazione di $7^{101} \bmod 103 = 59$ (possiamo usare l'algoritmo descritto ne la sezione §1.6), da cui segue che

$$7^{-1} \equiv 59 \pmod{103}$$

Algoritmo di Euclide Esteso
Usiamo l'algoritmo per ottenere che $7 \perp 103$ e in tal caso anche 7^{-1}; cominciamo a costruire la sequenza (1.3.2):

$$\begin{array}{cc} 103 = 14 \cdot 7 + 5 & q_1 = 14 \\ 7 = 1 \cdot 5 + 2 & q_2 = 1 \\ 5 = 2 \cdot 2 + \boxed{1} & q_3 = 2 \\ 2 = 2 \cdot 1 + 0 & q_4 = 1 \end{array}$$

Otteniamo che $\text{MCD}(7, 103) = 1$, e per quanto riguarda l'inverso costruiamo la sequenza (1.3.4):

$$\begin{array}{c} x_0 = 0 \\ x_1 = 1 \\ x_2 = -q_1 x_1 + x_0 = -14 \\ x_3 = -q_2 x_2 + x_1 = 15 \\ x_4 = -q_3 x_3 + x_2 = \boxed{-44} \end{array}$$

Segue che $-44 \bmod 103 = 59 = 7^{-1}$.

Ora che abbiamo l'inverso di 7 in modulo 103, siamo in grado di scrivere la soluzione dell'equazione ridotta:

$$x_0 \equiv 4 \cdot 59 \,(\mathrm{mod}\, 103)$$
$$x_0 \equiv 30 \,(\mathrm{mod}\, 103)$$

Le soluzioni successive saranno altre tre, a 103 di distanza da x_0, esse inoltre saranno in modulo 412:

$$X = \{30,\, 133,\, 236,\, 339\} \bmod 412$$

□

▶ *Esercizio: sistema di 2 congruenze*

ESERCIZIO 7.2. Trovare la congruenza equivalente al seguente sistema di congruenze:

$$\begin{cases} x \equiv 5 \,(\mathrm{mod}\, 11) \\ x \equiv 2 \,(\mathrm{mod}\, 20) \end{cases}$$

SOLUZIONE 7.2. Esisterà una sola congruenza equivalente, per il Teorema 1.2, infatti vale $11 \perp 20$; per riassumere le due congruenze in una sola, cerchiamo un valore x della successione $5 + k \cdot 11$ che sia anche congruente a 2 in modulo 20:

$$x = 5 + k \cdot 11 \equiv 2 \,(\mathrm{mod}\, 20)$$

Questo sarà vero per un valore di k pari a

$$k \cdot 11 \equiv 2 - 5 \,(\mathrm{mod}\, 20)$$
$$k \equiv -3 \cdot 11^{-1} \,(\mathrm{mod}\, 20)$$

Troviamo l'inverso di 11 in modulo 20 usando il Teorema 1.4:

$$11^{-1} \equiv 11^{\varphi(20)-1} \,(\mathrm{mod}\, 20)$$

sapendo che la funzione toziente di 20 vale $\varphi\,(20) = (2^2 - 2^1)\,(5^1 - 5^0) = 2 \cdot 4 = 8$, e che $11^{8-1} \bmod 20 = 11$, abbiamo la seguente congruenza per k:

$$k \equiv -33 \,(\mathrm{mod}\, 20)$$
$$k \equiv 7 \,(\mathrm{mod}\, 20)$$

Ritornando alla congruenza per x, si ottiene dalla precedente (con $k = 7$):

$$x = 5 + 77 \equiv 2 \,(\mathrm{mod}\, 20 \cdot 11)$$
$$x \equiv 82 \,(\mathrm{mod}\, 220)$$

dove abbiamo usato il Teorema 1.2 per comporre la congruenza in modulo 220. □

▶ *Esercizio: sistema di 3 congruenze*

ESERCIZIO 7.3. Trovare il valore della variabile x, che compare nel sistema di congruenze:

$$\begin{cases} x \equiv 1 \,(\mathrm{mod}\, 10) \\ x \equiv 2 \,(\mathrm{mod}\, 11) \\ x \equiv 0 \,(\mathrm{mod}\, 3) \end{cases}$$

SOLUZIONE 7.3. Mettiamo assieme la prima congruenza con la seconda, ottenendo:

$$x = 1 + k \cdot 10 \equiv 2 \,(\mathrm{mod}\, 11)$$

Risolviamo la precedente rispetto a k:

$$k \cdot 10 \equiv 1 \,(\mathrm{mod}\, 11)$$

Notando che $100 = 9 \cdot 11 + 1 \implies 100 \bmod 11 = 1$ possiamo affermare subito che $k = 10$; per il Teorema 1.2 scriviamo:

$$x \equiv 101 \,(\mathrm{mod}\, 110)$$

Unendo quanto ottenuto con la terza congruenza del sistema, si ha:

$$x = 101 + k \cdot 110 \equiv 0 \,(\mathrm{mod}\, 3)$$

Effettuando l'operazione di modulo 3 sui termini dell'equazione, avremo:

$$x = 2 + h \cdot 2 \equiv 0 \,(\mathrm{mod}\, 3)$$

Risolviamo la precedente rispetto ad h:

$$\begin{aligned} 2 \cdot h &\equiv -2 \,(\mathrm{mod}\, 3) \\ h &\equiv 1 \cdot 2^{-1} \,(\mathrm{mod}\, 3) \end{aligned}$$

Osserviamo che l'inverso di 2 in $\mathbb{Z}_3$ è proprio 2, infatti $2 \cdot 2 = 4 \bmod 3 = 1$; allora h è congruente a 2 in modulo 3, e sostituendo nella congruenza di x otteniamo:

$$\begin{aligned} x &\equiv 101 + 220 \,(\mathrm{mod}\, 110 \cdot 3) \\ x &\equiv 321 \,(\mathrm{mod}\, 330) \end{aligned}$$

Abbiamo trovato che x deve valere $321 + k \cdot 330$ per generare le tre congruenze esaminate. □

Elementi primitivi

ESERCIZIO 7.4. Quanti elementi primitivi ha l'insieme $\mathbb{Z}_{31}^*$, e quali sono?

► *Esercizio: elementi primitivi di un insieme dei residui*

SOLUZIONE 7.4. Gli elementi primitivi sono in numero $\varphi(31-1) = \varphi(30) = (2^1 - 2^0)(3^1 - 3^0)(5^1 - 5^0) = 1 \cdot 2 \cdot 4 = 8$ (applichiamo una delle formule a pagina 15 per calcolare il toziente $\varphi(\circ)$).

Per capire quali siano gli elementi primitivi, usiamo il test (1.9.4) con i quozienti $q_i = \{2, 3, 5\}$, per ciascun elemento di $\mathbb{Z}_{31}^*$ (da 1 a 30); facendo le prove col test rispetto ai tre quozienti, si ottengono i seguenti elementi primitivi:

$$\alpha = \{3, 11, 12, 13, \ldots\}$$

□

ESERCIZIO 7.5. Trovare gli elementi primitivi dell'insieme $\mathbb{Z}_{13}^*$

► *Esercizio: elementi primitivi di un insieme dei residui*

SOLUZIONE 7.5. Usiamo un secondo metodo, a partire da un elemento sicuramente primitivo $\alpha \in \mathbb{Z}_{13}^*$; proviamo con 2, sottoponendolo al test (1.9.4):

$$2^6 \equiv -1 \,(\mathrm{mod}\, 13)\,, \quad 2^4 \equiv 3 \,(\mathrm{mod}\, 13)$$

Allora $\alpha = 2$ è un elemento primitivo rispetto a $p = 13$; possiamo elevare α a tutte le potenze, per ottenere tutti gli elementi dell'insieme. Tuttavia solo le potenze positive pari restituiranno un altro elemento primitivo rispetto a p che è un quadrato:

$$\begin{array}{lll} \alpha^1 \equiv 2 \,(\mathrm{mod}\, 13) & \alpha^5 \equiv 6 \,(\mathrm{mod}\, 13) & \alpha^9 \equiv 5 \,(\mathrm{mod}\, 13) \\ \alpha^2 \equiv \boxed{4} \,(\mathrm{mod}\, 13) & \alpha^6 \equiv \boxed{12} \,(\mathrm{mod}\, 13) & \alpha^{10} \equiv \boxed{10} \,(\mathrm{mod}\, 13) \\ \alpha^3 \equiv 8 \,(\mathrm{mod}\, 13) & \alpha^7 \equiv 11 \,(\mathrm{mod}\, 13) & \alpha^{11} \equiv 7 \,(\mathrm{mod}\, 13) \\ \alpha^4 \equiv \boxed{3} \,(\mathrm{mod}\, 13) & \alpha^8 \equiv \boxed{9} \,(\mathrm{mod}\, 13) & \alpha^{12} \equiv \boxed{1} \,(\mathrm{mod}\, 13) \end{array}$$

Le radici primitive in $\mathbb{Z}_{13}^*$ sono:

$$a_q = \{1, 3, 4, 9, 10, 12\}$$

Si noti che ci troviamo nel caso $p \equiv 1 \,(\mathrm{mod}\, 4)$, per cui se non esiste $\sqrt{a}$ nemmeno $\sqrt{-a}$ è definita rispetto a p. □

Scrambling

▶ *Esercizio: scrambler auto-sincronizzante*

ESERCIZIO 7.6. Si consideri uni scrambler auto-sincronizzante con polinomio caratteristico $P(Y) = Y^3 + Y + 1$; esso è inizializzato come $M_1 M_2 M_3 = 000$, in ingresso si ha una sequenza di soli 1 ($\{I_k\} = 111\ldots$); determinare la sequenza di uscita $\{U_k\}$.

SOLUZIONE 7.6. Innanzitutto cerchiamo il periodo della successione, analizzando la riducibilità del polinomio $P(Y)$; esso non è divisibile per Y, tuttavia può esserlo per $Y + 1$; dalla divisione tra polinomi si ottiene:

Si ricorda che le operazioni su questo polinomio sono tutte in modulo 2

$$\begin{array}{l|l} Y^3 + Y + 1 & (\div Y + 1) \\ \hline Y^3 + Y^2 & Y^2 + Y \\ \hline Y^2 + Y + 1 & \\ \hline Y^2 + Y & \\ \hline 1 & \end{array}$$

Ottenendo resto non nullo, confermiamo che il polinomio non può essere diviso nemmeno per $Y + 1$; in tal caso possiamo affermare che $P(Y)$ sia irriducibile, e quindi il periodo della sequenza pseudo-casuale generata sarà determinato dal suo grado $M = 3$ come $\pi = 2^3 - 1 = 7$.

Lo scrambler descritto dal polinomio $P(Y)$ può essere implementato col circuito ne la figura 5.5.1; l'andamento della sua uscita si ricava riempiendo la seguente tabella (viene mostrato anche il descrambling della sequenza $\{U_k\}$ ottenuta):

k	$\overset{\bullet}{I}_k$	$\overset{\bullet}{M}_{1,k}$	$M_{2,k}$	$\overset{\bullet}{M}_{3,k}$	U_k
0	1	*0*	*0*	*0*	1
1	1	1	0	0	0
2	1	0	1	0	1
3	1	1	0	1	1
4	1	1	1	0	0
5	1	0	1	1	0
6	1	0	0	1	0
7	1	0	0	0	1

Tabella 7.5.1. Uscita dello scrambler

k	$\overset{\bullet}{U}_k$	$\overset{\bullet}{M}_{1,k}$	$M_{2,k}$	$\overset{\bullet}{M}_{3,k}$	I_k
0	1	*1*	*0*	*0*	**0**
1	0	1	1	0	**1**
2	1	0	1	1	**0**
3	1	1	0	1	1
4	0	1	1	0	1
5	0	0	1	1	1
6	0	0	0	1	1
7	1	0	0	0	1

Tabella 7.5.2. Uscita del descrambler

La tabella indica lo stato al passo k-esimo delle celle di memoria e dell'uscita; la consegna prevede di assegnare all'ingresso 1 in ciascun passo ($\forall k \in [0, \infty) : I_k = 1$). Le colonne con un cerchio pieno sopra l'etichetta di una colonna relativa a una cella di memoria, indica che essa è collegata in retroazione; in pratica l'uscita sarà influenzata dalla somma modulo 2 dell'ingresso con tutte le celle $\overset{\bullet}{M}_{i,k}$.

Possiamo notare che l'uscita ha proprio periodo 7; infatti lo stato delle celle di memoria al passo 7 è uguale a quello al passo 0.

Si noti che, nella tabella del descrambler, le tre celle di memoria sono state inizializzate in modo arbitrario ($M_1 M_2 M_3 = 100$), inoltre sono stati necessari 3 passi per avere la traduzione corretta della sequenza — lo scrambler auto-sincronizzante descritto da $P(Y)$ impiega $M = 3$ passi per raggiungere da solo la configurazione corretta. □

Cifrari a chiave pubblica

▶ *Esercizio: cifratura RSA*

ESERCIZIO 7.7. Adottiamo il sistema a chiave pubblica RSA, pubblicando l'intero $n = 323$ e l'esponente di cifratura $e = 17$; verificare la correttezza dei dati forniti.

Provare a decifrare il messaggio cifrato $c = 55$, senza conoscere l'esponente di decifratura. Cifrare il messaggio in chiaro $p = 55$.

SOLUZIONE 7.7. Controlliamo che l'esponente di cifratura sia primo relativo al toziente dell'intero n:

$$e \perp \varphi(n) \implies 17 \perp \varphi(323) = \text{MCD}(17, \varphi(323)) \stackrel{?}{=} 1$$

Per calcolare l'ultima uguaglianza è necessario ottenere il toziente di 323, e a questo scopo sarà necessario fattorizzare l'intero n; anche se in generale non sarebbe possibile, nel nostro caso abbiamo un intero piccolo, fattorizzabile per tentativi (dividiamo 323 per tutti i primi da 3 a $\sqrt{323}$); otteniamo che $17 \backslash 323$, e vale $323 = 17 \cdot 19$.

Il suo toziente può essere calcolato come:

$$\varphi(323) = (17-1)\cdot(19-1) = 16 \cdot 18 = \boxed{288} = 2^5 \cdot 3^2$$

Tra i fattori di 288 non compare $e = 17$, dunque l'esponente di cifratura sarà primo relativo rispetto al toziente dell'intero n.

Possiamo decifrare un messaggio cifrato qualunque, data la facilità di fattorizzazione dell'intero n. In generale, in RSA si decifra un messaggio cifrato tramite l'esponente di decifratura, secondo la seguente congruenza:

$$p \equiv c^d \pmod{n} \longrightarrow p \equiv 55^d \pmod{323}$$

Possiamo ottenere l'esponente di decifratura se conosciamo il toziente dell'intero n (che abbiamo calcolato in precedenza, fattorizzando n):

$$d \equiv e^{-1} \pmod{\varphi(n)} \stackrel{\star}{\equiv} e^{\varphi(\varphi(n))-1} \pmod{\varphi(n)} \longrightarrow d \equiv 17^{\varphi(288)-1} \pmod{288}$$

Nella congruenza $\star$ abbiamo usato l'osservazione (1.7.4) relativa al Teorema 1.4; a questo punto vogliamo calcolare il toziente di 288, che risulta $\varphi(288) = (2^5 - 2^4)\cdot(3^2 - 3) = 16 \cdot 6 = 96$ e sostituendo nelle precedenti congruenze si ha:

$$d = 17^{95} \bmod 288 = 17$$
$$p = 55^{17} \bmod 323 = \boxed{123}$$

Dato che l'esponente di decifratura è l'inverso dell'esponente di cifratura modulo $\varphi(323)$, la cifratura o la decifratura di 55 produce sempre 123 (quindi $p = 55$ viene cifrato in $c = 123$). □

Firma digitale

ESERCIZIO 7.8. Dati i seguenti parametri: un intero $p = 43$, un segreto $a = 10$, un nonce $k_1 = 11$ e un messaggio $m_1 = 15$, calcolare la firma di El Gamal per il messaggio m_1 e poi verificarla.

► *Esercizio: firma di El Gamal*

SOLUZIONE 7.8. Controlliamo le ipotesi del sistema di firma El Gamal: p è un primo e inoltre vale $a,\ k \in (1,\ p-2)$, infine $k \perp p-1$ ($\text{MCD}(11, 42) = 1$), quindi le ipotesi sono verificate.

Per calcolare la firma con El Gamal abbiamo ancora bisogno di una radice primitiva $\alpha \in \mathbb{Z}_p^*$; proviamo con $\alpha = 3$, e controlliamo che si tratta di una radice primitiva col test (1.9.4):

$$3^6 \equiv 41 \pmod{43}$$
$$3^{14} \equiv 36 \pmod{43}$$
$$3^{21} \equiv 42 \pmod{43}$$

Abbiamo la conferma che $\alpha = 3$ è una radice primitiva di $\mathbb{Z}_{43}^*$; ora possiamo calcolare $\beta \equiv \alpha^a \pmod p \implies 10 \equiv 3^{10} \pmod{43}$, e infine pubblichiamo i dati $(p,\, \alpha,\, \beta) = (43,\, 3,\, 10)$.

Firmiamo il messaggio $m_1 = 15$ costruendo i due interi r_1 ed s_1:

$$\begin{aligned} r_1 &\equiv \alpha^{k_1} \pmod p = 3^{11} \bmod 43 = 30 \\ s_1 &\equiv k_1^{-1} \cdot (m_1 - a \cdot r_1) \pmod{42} = 23 \cdot (15 - 10 \cdot 30) \bmod 42 = 39 \end{aligned}$$

Abbiamo calcolato l'inverso di k modulo 42 utilizzando la relazione (1.7.4) $k_1^{-1} \equiv k_1^{\varphi(42)-1} \pmod{42} = 11^{11} \bmod 42 = 23$; la firma del messaggio m_1 corrisponde a $(r_1,\, s_1) = (30,\, 39)$.

Verifichiamo ora la firma tramite la congruenza:

$$10^{30} \cdot 30^{39} \equiv 3^{15} \pmod{43} \implies 22 \equiv 22 \pmod{43} \checkmark$$

□

▶ *Esercizio: attacco del nonce ripetuto*

ESERCIZIO 7.9. Dati i seguenti parametri: un primo $p = 43$, una radice primitiva $\alpha = 3 \in \mathbb{Z}_{43}^*$, un nonce $k_1 = 11$ e un messaggio $m_1 = 15$, otteniamo $\text{SIG}(m_1) = (30,\, 39)$; se abbiamo firmato anche $m_2 = 20$ senza variare il nonce k_1, la sue firma sarà $\text{SIG}(m_2) = (30,\, 28)$. Dal punto di vista di un attaccante che osserva le firme dei messaggi, effettuare l'attacco del nonce ripetuto per ricavare k_1.

SOLUZIONE 7.9. Per prima cosa, possiamo verificare la validità delle firme dei due messaggi tramite i dati pubblici; per m_1 abbiamo effettuato la verifica nell'esercizio precedente, per m_2 si ha:

$$10^{30} \cdot 30^{28} \equiv 3^{20} \pmod{43} \implies 14 \equiv 14 \pmod{43}$$

Quindi entrambe le firme sono valide; notando che il valore di $r_1 = r_2$ possiamo affermare che il nonce abbia lo stesso valore in entrambi i casi ($k_1 = k_2 = k'$) e ad essere cambiato è il messaggio ($m_1 \neq m_2 \implies s_1 \neq s_2$).

Sapendo che $-a \cdot r \equiv s \cdot k - m \pmod{(p-1)}$, scriviamo la formula per entrambe le firme e i messaggi associati:

$$\begin{cases} -a \cdot r_1 \equiv s_1 \cdot k' - m_1 \pmod{(p-1)} \\ -a \cdot r_1 \equiv s_2 \cdot k' - m_2 \pmod{(p-1)} \end{cases}$$
$$(s_1 - s_2)\, k' \equiv m_1 - m_2 \pmod{(p-1)}$$
$$k' \equiv (15 - 20) \cdot (39 - 28)^{-1} \pmod{42}$$
$$k' = -5 \cdot 11^{-1} \bmod 42$$

Calcolando $\text{MCD}(11,\, 42) = 1$ possiamo affermare che l'equazione in k' abbia 1 soluzione; sapendo che vale $11^{-1} \equiv 23 \pmod{42}$ (come calcolato nell'esercizio precedente), proseguiamo valutando l'ultima delle precedenti congruenze:

$$k' = -5 \cdot 23 \bmod 42 = 11$$

Abbiamo ottenuto il valore di k_1, il nonce che è stato ripetuto nella firma dei due messaggi m_1 ed m_2. □

Parte 2

Comunicazione sicura

CAPITOLO 8

Instaurazione e distribuzione della chiave

8.1. Identificare la chiave

8.1.1. Introduzione. La matematica che è stata applicata alla crittografia è abbastanza solida da garantire l'impossibilità pratica di decifrare un messaggio se gli algoritmi sono implementati correttamente; in particolare richiamiamo il Teorema A che è valido sia per chiavi simmetriche che per chiavi asimmetriche.

Infatti teoricamente è possibile ricavare la chiave privata da quella pubblica, ma in pratica è possibile solo a patto di conoscere un'informazione "speciale" — come i due fattori primi in RSA, o l'esponente usato per l'elevamento a potenza in El Gamal.

In generale, mittente e destinatario devono scambiare un segreto, e possono comunicare solo a distanza, tramite un canale non sicuro; sembrerebbe che la soluzione sia usare un crittosistema a chiave pubblica, tuttavia:

- gli algoritmi per implementare i crittosistemi a chiave asimmetrica sono computazionalmente più onerosi delle controparti a chiave simmetrica.

Una seconda possibilità consiste, per esempio, nell'usare un crittosistema a chiave asimmetrica per scambiare in modo sicuro una chiave simmetrica, quest'ultima sarà utilizzata per la successiva cifratura della comunicazione; in questo caso:

- non è possibile fidarsi di una chiave pubblica pubblicata sul web, senza una certificazione adeguata.

Allora è necessario fidarsi di un *certificato*, che autentica una chiave pubblica associandola con l'identità dell'utente che l'ha pubblicata; si tratta di un documento emesso da un ente di certificazione di cui gli utenti si fidano (solitamente si tratta di un ente che attribuisce valore economico alla sua credibilità, dunque la fiducia nei suoi confronti è giustificata).

Distinguiamo e analizziamo nelle sezioni successive gli algoritmi delle seguenti categorie:

(1) **distribuzione** della chiave: una autorità centrale si occupa di inviare tutte le chiavi a tutti gli utenti (nel caso di n utenti e di chiavi simmetriche, bisognerà distribuire circa $n^2/2$ chiavi differenti);
(2) **instaurazione** della chiave: mittente e destinatario scambiano, attraverso un canale non sicuro, gli elementi necessari a costruire la chiave privatamente, senza che chi ne venga a conoscenza possa a sua volta costruirla.

8.1.2. Attacco man-in-the-middle. Richiamiamo l'algoritmo di pattuizione della chiave Diffie-Hellman (§7.2); esso presenta il seguente problema di sicurezza: un attaccante potrebbe fingersi il mittente o il destinatario dei messaggi. In effetti, non è possibile sapere a priori se il mittente o il destinatario sia effettivamente chi dice di essere.

Introduciamo l'attacco *man-in-the-middle* (letteralmente "uomo nel mezzo"), che consiste nell'impersonare uno degli interlocutori; nel caso dell'algoritmo Diffie-Hellman, abbiamo un attaccante **E** che agisce nel modo seguente:

- **E** sceglie un esponente $z \in [1,\, p-2]$ (compatibilmente con le informazioni pubblicate da **A** e **B**);
- **E** intercetta α^x e α^y provenienti da **A** e **B**;
- **E** invia α^z ad **A** (crede di ricevere α^y) e a **B** (crede di ricevere α^x);
- **E** calcola $K_{\mathrm{A}\rightleftarrows\mathrm{E}} \equiv (\alpha^x)^z \pmod p$ e $K_{\mathrm{E}\rightleftarrows\mathrm{B}} \equiv (\alpha^y)^z \pmod p$
- **A** calcola $K_{\mathrm{A}\rightleftarrows\mathrm{E}}$ indotto da **E**, e anche **B** calcola $K_{\mathrm{E}\rightleftarrows\mathrm{B}}$ indotto da **E** (invece di calcolare entrambi $K_{\mathrm{A}\rightleftarrows\mathrm{B}}$);
- Quando **A** invia un messaggio a **B** cifrato tramite $K_{\mathrm{A}\rightleftarrows\mathrm{E}}$ **E** lo intercetta, lo decifra e poi lo cifra con $K_{\mathrm{E}\rightleftarrows\mathrm{B}}$, infine lo inoltra a **B**.

Per ovviare a questo problema, è stata introdotta una modifica dell'algoritmo Diffie-Hellman, formalizzata nel protocollo *Station-to-station*.

8.1.3. Station-to-station. Si tratta di una variante dell'algoritmo di pattuizione della chiave Diffie-Hellman, con l'aggiunta del concetto delle firme digitali, per autenticare gli interlocutori. Supponiamo che A e B vogliano instaurare una chiave simmetrica K da usare nella funzione di cifratura $E_K()$; inoltre, ammettiamo che gli algoritmi di verifica della firma per un dato utente U siano disponibili pubblicamente, e sia **T** un'autorità fidata che certifichi che un algoritmo di verifica ver_U sia adatto a verificare la firma SIG_U dell'utente U, e non dell'attaccante **E**; essi procedono nel modo seguente:

(1) entrambi scelgono un primo p molto grande, e una radice primitiva $\alpha \in \mathbb{Z}_p$ tale che il problema del logaritmo discreto in $\mathbb{Z}_p$ sia intrattabile;
(2) **A** sceglie un intero casuale $x \in [1,\, p-2]$, **B** sceglie un intero casuale $y \in [1,\, p-2]$;
(3) **A** calcola $\alpha^x \bmod p$, **B** calcola $\alpha^y \bmod p$;
(4) **A** invia α^x a **B**;
(5) **B** calcola $K \equiv (\alpha^x)^y \pmod p$;
   (a) **B** invia α^y e $E_K(\mathrm{SIG}_B(\alpha^y,\, \alpha^x))$ ad **A**;
   (b) **A** calcola $K \equiv (\alpha^y)^x \pmod p$;
   (c) **A** decifra $E_K(\mathrm{SIG}_B(\alpha^y,\, \alpha^x))$ ottenendo $\mathrm{SIG}_B(\alpha^y,\, \alpha^x)$;
   (d) **A** chiede a **T** di verificare che ver_B sia l'algoritmo di verifica per la firma di **B;**
(6) **A** usa ver_B per verificare SIG_B;
   (a) **A** invia $E_K(\mathrm{SIG}_A(\alpha^x,\, \alpha^y))$ a B;
   (b) **B** decifra $E_K(\mathrm{SIG}_A(\alpha^x,\, \alpha^y))$ ottenendo $\mathrm{SIG}_A(\alpha^x,\, \alpha^y)$;
   (c) **B** chiede a **T** di verificare che ver_A sia l'algoritmo di verifica per la firma di **A**;
(7) **B** usa ver_A per verificare SIG_A.

Osservazione 8.1. Il ruolo dell'autorità di certificazione e la fiducia riposta in essa sono centrali in questo algoritmo; infatti la sicurezza della comunicazione tra gli interlocutori si fonda ampiamente sulla fiducia nella verifica di **T**.

8.2. Distribuire la chiave

Anche se siamo in possesso di un algoritmo crittografico perfettamente implementato, la debolezza si nasconde nella password utilizzata per la cifratura: in questo senso, la maggior parte degli attacchi prende come bersaglio la password stessa, poiché computazionalmente fattibile.

Un'altra problematica riguarda la sicurezza della trasmissione dei segreti condivisi (per esempio chiavi simmetriche) dalle due parti che vogliono comunicare, quando

esse sono distanti. Con la crittografia a chiave pubblica il problema della trasmissione dei segreti è risolto, tuttavia è impossibile fidarsi della provenienza di una chiave pubblica senza ulteriori informazioni correlate.

Affrontiamo ora il problema della distribuzione delle chiavi pubbliche a un gruppo di utenti, considerando che il canale di comunicazione sia osservato da **O**, il quale può anche ripetere i messaggi ascoltati sul canale.

8.2.1. Shamir-Massey-Omura (3-pass protocol). Questo protocollo permette di trasferire una chiave segreta $\bar{K}$ tra due parti A, B tramite un canale non sicuro:

(1) **A** sceglie e pubblica un primo p tale da rendere difficile il problema del logaritmo discreto in $\mathbb{Z}_p^*$ e abbastanza grande da rappresentare $\bar{K}$;
(2) **A**, **B** scelgono ciascuno un numero segreto casuale, coprimo rispetto a $p-1$ ($\text{MCD}(a, p-1) = 1 \wedge \text{MCD}(b, p-1) = 1$);
(3) **A**, **B** calcolano rispettivamente $a^{-1} \bmod (p-1)$ e $b^{-1} \bmod (p-1)$;
   (a) $\mathbf{A} \to \mathbf{B}$ invia $K_1 \equiv \bar{K}^a \pmod p$;
   (b) $\mathbf{A} \leftarrow \mathbf{B}$ invia $K_2 \equiv K_1^b \equiv \bar{K}^{ab} \pmod p$;
   (c) $\mathbf{A} \to \mathbf{B}$ invia $K_3 \equiv K_2^{a^{-1}} \equiv \bar{K}^{aba^{-1}} \pmod p$;
(4) **B** calcola $\bar{K} \equiv K_3^{b^{-1}} \equiv \bar{K}^{aba^{-1}b^{-1}} \pmod p$.

Possiamo generalizzare il protocollo usando RSA (o un'altra funzione crittografica che goda di proprietà commutativa) invece dell'elevamento a potenza descritto, usando l'intero n prodotto di due primi invece del primo p.

8.2.2. Schema di Blom per pre-distribuzione. Utilizziamo questo algoritmo quando dobbiamo distribuire a un gran numero di utenti di una rete altrettante chiavi, tramite un'autorità fidata **T**:

(1) Assegniamo a ogni utente un numero pubblico $r_U \in \mathbb{U} \bmod p$;
(2) **T** sceglie tre numeri a, b, c segreti, casuali e residui modulo p;
(3) **T** calcola, per ogni utente r_U, le quantità:

$$a_U \equiv a + b \cdot r_U \pmod p, \qquad b_U \equiv b + c \cdot r_U \pmod p$$

   e li invia all'utente r_U su un canale sicuro;
(4) ciascun utente, usando il proprio r_U, calcola il polinomio

$$g_U(r_U) = a_U + b_U \cdot r_U$$

Ora consideriamo due utenti A e B con i rispettivi numeri r_A e r_B:

- se A vuole comunicare con B, deve computare $g_A(r_B) = K_{A \to B}$;
- se B vuole comunicare con A, deve computare $g_B(r_A) = K_{A \leftarrow B}$;

inoltre si può dimostrare che $K_{A \to B} \equiv K_{A \leftarrow B}$, per cui la chiave può essere usata per un crittosistema a chiave simmetrica.

8.2.3. Cooperazione degli utenti. Se due utenti A e B scambiano i propri numeri ricevuti da T, entrambi vengono a conoscenza di a_A, b_A, a_B, b_B: in tal caso potranno computare i tre numeri segreti a, b, c selezionati da T, e quindi scoprire le chiavi per comunicare con tutti gli altri utenti.

8.3. Distribuzione autenticata

Trattiamo qui il secondo problema introdotto nella sezione precedente, ovvero la fiducia nelle chiavi ricevute dall'autorità fidata **T**: infatti un osservatore **O** che intercetta e ripete un messaggio inviato da **T** sarà scambiato per quest'ultimo dal destinatario del messaggio, il quale potrebbe essere indotto a instaurare una chiave con **O**.

Per contrastare un simile attacco è opportuno autenticare gli utenti; inoltre si impiegano delle *key encryption key* (KEK) per cifrare la comunicazione delle chiavi instaurate con l'autorità fidata.

Un timestamp è una stringa che rappresenta l'informazione sul tempo di sistema di un dispositivo

Un'altro metodo consiste nell'identificazione dei messaggi, tramite numero di sequenza (bisogna tenerne traccia), *timestamp* (le parti che vogliono comunicare devono essere sincronizzate) o nonce (è necessario uno schema di *challenge-response*).

8.3.1. Protocollo della rana dalla bocca larga. Si tratta di un protocollo di distribuzione autenticata della chiave tramite un'autorità fidata **T**, che sfrutta i timestamp e si basa sull'ipotesi che le KEK tra gli utenti e **T** siano già state distribuite:

(1) **A** sceglie una chiave di sessione K_{AB} per comunicare con **B**, e chiede a **T** di inviargliela in modo sicuro;
(2) **A** invia a **T** un timestamp t_A, un identificativo del destinatario ID_B e la chiave che vuole trasferire K_{AB}, cifrando il messaggio con la chiave per comunicare con **T**:

$$E_{K_{AT}}\left[t_A \| \mathrm{ID}_B \| K_{AB}\right];$$

(3) **T** invia a **B** un timestamp t_T, un identificativo del mittente ID_A e la chiave da trasferire $K_{A\to B}$, cifrando il messaggio con la chiave per comunicare con **B**:

$$E_{K_{BT}}\left[t_T \| \mathrm{ID}_A \| K_{AB}\right];$$

Il timestamp serve per evitare gli attacchi di replica (l'attaccante dovrebbe conoscere le KEK per modificare il contenuto dei messaggi); il fatto che l'autorità fidata si occupi di rinnovare il timestamp rappresenta una potenziale minaccia.

8.3.2. Man-in-the-middle al protocollo della rana dalla bocca larga. Immaginiamo di avere sulla rete un altro utente malevolo **M**, il quale vuole estendere la validità della KEK: una chiave di sessione come la KEK deve avere durata limitata, sia per evitare di concedere all'attaccante il tempo per trovarla, sia perché se compromessa è desiderabile sostituirla il prima possibile.

(1) **M** osserva uno scambio di messaggi tra **A** e **T** e tra **T** e **B**;
(2) **M** impersona **B**, e replicando il messaggio inviato da **T** a **B**, chiede a **T** di condividere nuovamente la stessa chiave K_{AB} con **A**:

$$E_{K_{BT}}\left[t_T \| \mathrm{ID}_A \| K_{AB}\right];$$

(3) **T** riceve un messaggio valido e quindi invia la chiave ad **A**, aggiornando il timestamp del messaggio:

$$E_{K_{AT}}\left[t'_T \| \mathrm{ID}_B \| K_{AB}\right]$$

A crede che il messaggio ricevuto da **T** sia valido, quindi la chiave K_{AB} rimane valida per un tempo $t'_T > t_T$, e per un certo intervallo di tempo successivo;

(4) **M** impersona **A**, e invia nuovamente a **T** il messaggio appena osservato (quello col timestamp t'); **B** riceve il messaggio contenente la chiave K_{AB} e un timestamp $t''_T > t'_T$:

$$E_{K_{BT}}\left[t''_T \| \mathrm{ID}_A \| K_{AB}\right]$$

così facendo la chiave K_{AB} rimarrà valida per un intervallo di tempo successivo a t''_T;

(5) **M** ripete i passi dal (2) al (4).

Con questa procedura, è possibile indurre gli utenti a usare in modo indefinito la stessa chiave K_{AB}.

8.3.3. Protocollo Needham-Schroeder. Poniamo il caso di avere un'autorità fidata **T**, che abbia già distribuito agli utenti **A** e **B** le KEK:

(1) **A** invia a **T** in chiaro il suo ID, quello dell'utente col quale vuole comunicare (**B**) e un nonce r_1:

$$\mathrm{ID}_A \| \mathrm{ID}_B \| r_1;$$

(2) **T** risponde ad **A** cifrando con la KEK relativa, inviando una chiave di sessione estratta K_S, lo stesso nonce , l'ID del destinatario e il messaggio che egli riceverà (cifrato con la KEK relativa):

$$E_{K_{AT}}\left[K_S \| \mathrm{ID}_B \| r_1 \| E_{K_{BT}}\left[K_S \| \mathrm{ID}_A\right]\right];$$

(3) **A** estrae l'ultima stringa $E_{K_{BT}}\left[K_S \| \mathrm{ID}_A\right]$ (che non può decifrare), e la manda a **B**, il quale possiede la KEK per estrarre dal messaggio la chiave di sessione K_S che userà per comunicare con **A** (il messaggio contiene ID_A);

(4) **B** invia ad **A** un nonce cifrato con la chiave di sessione: $E_{K_S}\left[r_2\right]$;

(5) **A** risponde a **B** col nonce meno 1: $E_{K_S}\left[r_2 - 1\right]$.

CAPITOLO 9

Autenticazione e autorizzazione

9.1. Caratteristiche dell'autenticazione

Autenticare vuol dir garantire l'identità di un utente, di un dispositivo o di una sorgente di informazioni; autorizzare vuol dire determinare le risorse e i servizi che una entità autenticata è in grado di usare (l'autorizzazzione implica l'autenticazione).

La definizione di autenticazione è enunciata nel RFC 4949 (IETF)

L'autenticazione è costituita da tre fasi principali:

(1) REGISTRAZIONE: l'utente deve essere registrato (per esempio in una base di dati) e associato a delle credenziali;
(2) IDENTIFICAZIONE: il candidato si presenta al verificatore (per esempio il server) come un certo utente, fornendogli un identificatore;
(3) VERIFICA: il candidato deve provare di essere chi afferma, (per esempio tramite password, rispondendo a una sfida, ecc...).

Possiamo classificare l'autenticazione in base al tipo di connessione col servizio richiesto:

- locale (l'utente accede localmente alle risorse);
- diretta (l'utente accede a un sistema remoto);
- indiretta (l'autenticazione è delegata a una terza parte);
- off-line (il successo dell'autenticazione viene determinato senza contattare l'utente o una terza parte).

Inoltre, l'autenticazione può essere effettuata attraverso i seguenti mezzi (*fattori*):

- qualcosa che sai (PIN, password);
- qualcosa che hai (smart card, token hardware);
- qualcosa che sei (impronta digitale, scansione retina);
- qualcosa che fai (timbro vocale, scrittura a mano).

Spesso per migliorare la sicurezza di questi sistemi si combinano due o tre fattori di autenticazione.

Per garantire la sicurezza dell'autenticazione è necessario che: le credenziali non siano trasmesse in chiaro (un osservatore del canale può intercettarle), e le credenziali non siano salvate in chiaro (se il sistema viene compromesso, l'attaccante le ottiene).

Ci si riferisce ai dati trasmessi e salvati rispettivamente con *"in transit"* e *"at rest"*

9.2. Username e password

9.2.1. Debolezze della password. Sistema più comune di autenticazione, che incarna il Teorema A: il nome utente può essere pubblicato, ma la password va mantenuta segreta. Per questo motivo viene trasmessa e memorizzata sotto forma di *digest* (prodotto di una funzione di hash).

In pratica, per ottenere la password che corrisponde al digest memorizzato o trasmesso, dobbiamo tentare tutte le possibili stringhe come input della funzione di hash coinvolta, fino ad ottenere una collisione o la password corretta, e questo richiede un numero di tentativi proporzionale al numero di bit del digest.

Pur procedendo a caso, se abbiamo un digest di n bit e calcoliamo 2^n hash per altrettante stringhe in input differenti, la probabilità di trovare la corrispondenza è comunque inferiore a 1.

Ovviamente, la probabilità di successo si innalza drasticamente quando la password è la parola di una lingua e ci sono le condizioni per mettere in pratica un attacco del vocabolario.

9.2.2. Entropia della password. Una password è tanto più sicura quanta più informazione segreta essa contiene; dalla teoria dell'informazione di Shannon, sappiamo che la misura di informazione in un messaggio è la sua entropia (misura del "disordine"); a grandi linee possiamo affermare che:

> *L'entropia di un messaggio è l'impredicibilità del suo stato o, in modo equivalente, del suo contenuto informativo, o del contenuto informativo medio di una sorgente di informazioni; un messaggio contiene tanta più informazione quanto più è impredicibile.*

DEFINIZIONE 9.1. Consideriamo una sorgente X (priva di memoria, la probabilità di ottenere un determinato messaggio è indipendente dai messaggi già estratti) dalla quale estraiamo il messaggio $\overline{x}$, definito su un alfabeto finito di N simboli:

$$\overline{x} \in \{x_1, \ldots, x_N\}$$

La probabilità che il messaggio assuma valore $\overline{x} = x_i$ per $i \in [1, \ldots, N]$ sarà definita come:

$$p_i = P(\overline{x} = x_i)$$

la relativa distribuzione di probabilità quasi sempre non è uniforme e dipende soprattutto dal linguaggio che usa l'alfabeto.

Chiamiamo quantità di *informazione*, e la misuriamo in bit, la seguente funzione relativa a un messaggio $\overline{x} = x_i$:

$$\mathbb{I}(\overline{x}) \coloneqq -\log_2(p_i) \geq 0 \tag{9.2.1}$$

L'informazione media di una sorgente, per carattere sull'alfabeto, è chiamata *entropia di sorgente* e si indica come:

$$\mathbb{H}(X) \coloneqq -\sum_{i=1}^{N} p_i \cdot \log_2(p_i) \geq 0 \tag{9.2.2}$$

American Standard Code for Information Interchange, si pronuncia /ˈæski:/

ESEMPIO 9.1. *Consideriamo una password alfabetica, che usi caratteri definiti tramite 7 bit; dei $2^7 = 128$ caratteri ASCII, 32 sono di controllo (non stampabili e non usati nelle password) e i restanti 95 simboli saranno l'alfabeto $\vec{a}$ che useremo; ottenere l'entropia di una password così formata.*

Applicando la formula (9.2.2) su una sorgente di informazione A con tale alfabeto, in cui i simboli in uscita abbiano distribuzione di probabilità uniforme, otteniamo:

$$\mathbb{H}(A) = \log_2(95) \simeq 6.57 \,\text{bit/carattere}$$

Segue che una password di 10 caratteri ricavata dalla sorgente A avrà circa 65.7 bit di informazione; inoltre possiamo verificare che, usando un alfabeto latino di 26 lettere più lo spazio, la quantità d'informazione relativa sarà di circa 4.75 bit/carattere.

Infine, osservando che i simboli non hanno tutti la stessa probabilità di apparire all'interno di una determinata lingua (si può determinare la distribuzione di probabilità di una certa lingua, da usare nella formula per l'entropia della sorgente), possiamo ottenere un'entropia media per carattere ancora più bassa. □

È ragionevole aspettarsi che caratteri successivi estratti da una sorgente siano correlati (digrammi, trigrammi): questo abbassa ancora di più l'entropia finale di un determinato messaggio; prendendo un testo in lingua italiana sufficientemente lungo come "I promessi sposi" (A. Manzoni) ed estraendo cinque simboli dalla sorgente P (che usa alfabeto di 21 lettere) otteniamo:

$$\mathbb{H}_5(P) \simeq 1.87\,\text{bit/carattere}$$

Shannon dimostrò che, preso un testo in lingua inglese come sorgente E, l'entropia media che ne risulta è:

$$0.6 < \mathbb{H}(E) < 1.3\,\text{bit/carattere}$$

L'entropia di una passphrase o password ad oggi è in media circa 2 bit/carattere.

9.2.3. Attacchi alle password. Possiamo ottenere una password attaccando la sua memorizzazione o trasmissione, nei seguenti modi:

- *sniffing* (intercettazione del traffico sul canale dove avviene l'autenticazione);
- on-line *guessing* (generazione e invio di molte password al server per l'autenticazione);
- off-line *guessing* (generazione di password per attaccare la memorizzazione non sicura, teoricamente senza limiti di tempo e spazio);
- altro (social engineering, shoulder surfing, *phishing*, *keylogging*, ecc...).

9.3. Sfida e risposta

9.3.1. Motivazioni e implementazione. Risolve il problema della trasmissione sicura delle credenziali, che in questo protocollo non viene effettuata: un segreto viene condiviso in qualche maniera tra client e server, e il client deve dimostrare di conoscerlo, rispondendo in maniera adeguata a una sfida.

La sfida viene risolta nel modo seguente:

(1) client e server condividono lo stesso segreto K;
(2) Il server invia al client un numero pseudo-casuale S;
(3) il client invia indietro una risposta costruita come
   (a) cifratura simmetrica della sfida $E_K(S)$, tramite segreto K;
   (b) il digest della sfida concatenata al segreto $h(S\|K)$;
   (c) cifratura della sfida tramite chiave privata K_p, $E_{K_p}(S)$ (il server deve possedere la chiave pubblica associata K_ℓ).

Oltre alla procedura che usa un segreto condiviso, un protocollo di sfida e risposta (challenge-response) può essere implementato tramite chiavi di sessione mono-uso; si pone tuttavia il problema della generazione e del trasferimento sicuro di queste chiavi da server a client.

Chiamiamo OTP (One Time Password) una password mono-uso

9.3.2. Schema di Lamport per la catena di hash. Consideriamo un client **A** e un server **B**, quest'ultimo vuole autenticare **A**; il client conosce il proprio nome sul server e la password p_A, mentre il server conosce il digest della funzione di hash calcolata n volte ricorsivamente su p_A (dove n è un numero grande, e che indichiamo con $h_n(p_A)$); è evidentemente necessaria una fase di instaurazione delle chiavi nella quale al server viene inviato il nome dell'utente A accoppiato col digest $h_n(p_A)$.

L'autenticazione si svolge nel modo seguente:

(1) **A** invia il proprio nome utente in chiaro a **B**;
(2) **B** risponde ad **A** inviando n in chiaro;

(3) **A** invia $h_{n-1}(p_A)$ a **B**;
(4) **B** computa $h_n(p_A) = h(h_{n-1}(p_A))$, e lo confronta col valore abbinato al nome utente A;
(5) in caso di successo, **B** aggiorna il valore associato al nome utente A, rimpiazzandolo con $h_{n-1}(p_A)$, e alla prossima richiesta di autenticazione da **A** invierà l'intero $n-1$.

La sicurezza dell'algoritmo si basa sull'unidirezionalità della funzione di hash $h(\circ)$ e sulla grandezza di n.

Schema di Lamport con due server di autenticazione. In un contesto con due server **B** e **C** a cui l'utente **A** si vuole autenticare, è possibile che ciascuno di essi invii al client un intero n differente; in tal caso, un osservatore sul canale vedrà le coppie n_B, $h_{n_B-1}(p_A)$ e n_C, $h_{n_C-1}(p_A)$; se effettivamente i due interi sono diversi e inoltre siamo nel caso $n_B = n_C + 1$, allora l'osservatore potrà presentarsi al server **C** inviando il nome utente A e il digest $h_{n_C-1}(p_A)$.

Per far fronte a questa debolezza basta usare un sale unico per ciascun server, per salvare le password p degli utenti; in questo modo i digest destinati a un server saranno rifiutati dagli altri.

Attacco del piccolo n. Quando il server sceglie l'intero n, determina il tempo di vita dell'algoritmo; supponiamo che un osservatore **O** sul canale riesca a impersonare il server **B**: egli attende la richiesta di autenticazione da parte di **A**, e risponde con un intero piccolo (per esempio $n = 50$).

L'osservatore otterrà l'informazione $h_{49}(p_A)$ e, se il server vero sta usando $n = 1000$, allora **O** potrà usarla per generare tutti i digest fino a $h_{1000}(p_A)$, per autenticarsi le prossime $1000 - 50 = 950$ volte col server vero al posto di **A**.

9.3.3. Token di sicurezza. Si tratta di un dispositivo fisico o di un software che genera una password mono-uso basandosi sul tempo (OTP valida entro una finestra di tempo, usata per sincronizzare il tempo del token col server) o su un contatore (generato in modo incrementale secondo una sequenza, permette di sincronizzarla col server).

Per aumentarne la sicurezza, i token sono spesso abbinati al fattore di autenticazione della conoscenza (PIN o password da abbinare alla OTP); questa procedura viene chiamata autenticazione a due fattori e si indica con la sigla 2FA.

9.3.4. Biometrica. Si utilizza per evitare all'utente di ricordare una passphrase lunga, tuttavia si tratta di sistemi ancora deboli e non in grado costituire il singolo fattore di autenticazione di un sistema.

Le applicazioni di questo fattore sono le seguenti:

- autenticazione (confermare la dichiarata identità di un utente);
- identificazione (associare un campione dato con un utente o un piccolo gruppo dalle caratteristiche simili);
- unicità (dato un campione, possiamo determinare se l'utente sia già registrato).

Il fattore biometrico viene a sua volta suddiviso in base alle modalità di acquisizione del campione:

- statico (qualcosa che sei, caratteristiche fisiche):
- dinamico (qualcosa che fai, pattern di comportamento).

La ricerca dei campioni nella base di dati degli utenti registrati viene effettuata per corrispondenza migliore: dal momento che l'informazione biometrica digitalizzata è molto ingombrante, su di essa viene calcolato un modello semplificato basato su caratteristiche comuni a tutti i campioni, e poi confrontato coi dati registrati.

Il confronto viene effettuato calcolando la distanza tra campione e dato registrato, e il campione viene accettato solo se questa distanza è inferiore a una certa soglia di accettazione.

Le caratteristiche fisiche che vengono campionate come fattore biometrico possono essere: impronte digitali, forma della mano, iride o retina, volto, immagine termica, analisi dell'odore, DNA, (fattori statici) oppure riconoscimento vocale, dinamica e pattern di scrittura (fattori dinamici), ecc...

Accuratezza e prestazioni del fattore biometrico.

DEFINIZIONE 9.2. Chiamiamo tasso di falsa accettazione (false acceptance rate) e lo indichiamo con la sigla FAR, la misura della probabilità che il sistema permetta l'accesso a un utente non autorizzato (falso positivo); il suo valore si ottiene come:

$$\text{FAR} := \frac{\text{falsi positivi accettati}}{\text{tentativi di identificazione}}$$

DEFINIZIONE 9.3. Chiamiamo tasso di falso rifiuto (false rejection rate) e lo indichiamo con la sigla FRR, la misura della probabilità che il sistema neghi l'accesso a un utente autorizzato e registrato (falso negativo); il suo valore si ottiene come:

$$\text{FRR} := \frac{\text{falsi negativi rifiutati}}{\text{tentativi di identificazione}}$$

Fissato un metodo per acquisire campioni biometrici, e una metrica per modellizzarli ed effettuare confronti tra loro calcolandone la distanza, possiamo regolare la soglia di decisione basandoci sui parametri FAR e FRR: il minimo della somma o del prodotto delle due probabilità (punto di incontro delle relative curve in $\mathbb{R}^2$) viene scelto come soglia di accettazione per il sistema.

I sistemi biometrici attuali non ammettono valori FAR e FRR tali da permettere di usare questo fattore di autenticazione da solo; i problemi di accuratezza di cui soffre sono classificati come segue:

- rumore nella misurazione e inconsistenza dei dati acquisiti (sensore per impronte digitali sporco o polpastrello ferito);
- variazioni intra-classe (la foto del volto cambia a seconda della luce ambientale);
- similitudine inter-classe (persone con volti molto simili);
- non-universalità (non tutti gli individui presentano qualità adeguata per le caratteristiche fisiche misurate, come l'assenza di impronte digitali per una persona priva di mani);
- *spoofing* (ingannare il sensore inducendo un falso positivo).

9.4. Kerberos

Questo protocollo fornisce un servizio centralizzato per la distribuzione delle chiavi e l'autenticazione indiretta in un ambiente distribuito e non fidato (si dà per scontata la presenza di osservatori sul canale e il tentativo di impersonare gli utenti legittimi).

I suoi requisiti di implementazione sono:

- gli utenti accedono ai servizi tramite la rete;

- i server limitano l'accesso agli utenti autorizzati e autenticano le richieste per i singoli servizi;
- basato sul protocollo Needham-Schroeder (§ 8.3.3), con l'impiego di un server fidato per l'autenticazione;
- usa solo crittografia simmetrica (nessuna chiave pubblica);
- richiede la sincronizzazione del tempo su tutte le macchine coinvolte.

L'idea alla sua base è avere un server per autenticare gli utenti e un server per autenticare l'accesso al servizio richiesto.

9.4.1. Richiesta di autenticazione. Consideriamo un utente **C** che vuole autenticarsi tramite il server di autenticazione **A** (quest'ultimo deve essere stato inizializzato correttamente con le chiavi simmetriche):

(1) **C** invia ad **A** il proprio ID, la propria password e l'ID del server al quale vuole accedere ($\mathrm{ID}_C \| p_C \| \mathrm{ID}_S$);
(2) **A** risponde a **C** con un *ticket* $\bar{t}$ (messaggio che solo il server di autenticazione può creare ed è valido per uno specifico utente ID_C su uno specifico server ID_S);
(3) **C** invia ad **S** (server identificato da ID_S) il ticket e il proprio nome ($\mathrm{ID}_C \| \bar{t}$).

Il primo problema di questa implementazione è rappresentato dall'invio della password dell'utente in chiaro; inoltre con questa implementazione ogni volta che si vuole richiedere il servizio è necessario usare un ticket (sarebbe più comodo per l'utente avere a disposizione una sessione al cui interno usare liberamente il servizio).

9.4.2. Ticket granting server. Introduciamo i seguenti cambiamenti all'algoritmo mostrato nella precedente sezione:

- non trasmettiamo la password dell'utente in chiaro;
- usiamo un ticket granting server (**T**), che distribuisce a un utente autorizzato un ticket per ottenere un altro servizio richiesto;
  - ticket granting ticket ($\bar{t}_G$) permette di richiedere il ticket di un servizio;
  - ticket di servizio ($\bar{t}_S$) permette di richiedere uno specifico servizio.

Implementiamo il protocollo con i nuovi elementi introdotti:

(1) l'utente **C**, che si vuole autenticare, chiede un ticket al server **A**, inviandogli $\mathrm{ID}_C \| \mathrm{ID}_T$;
(2) il server di autenticazione **A** risponde con un ticket $\bar{t}_G$ cifrato con la chiave simmetrica condivisa con **C** ($E_{K_{C,A}}(\bar{t}_G)$);
(3) **C** chiede a **T** un ticket per usare un servizio sul server **S**, inviandogli $\mathrm{ID}_C \| \mathrm{ID}_S \| \bar{t}_G$ (il ticket ricevuto da **T** può essere estratto solo dal vero **C**, poiché è necessario decifrare usando la chiave simmetrica $K_{C,A}$ condivisa tra utente e server di autenticazione);
(4) il server **T** risponde con il ticket $\bar{t}_S$ se ha ricevuto un $\bar{t}_G$ coretto;
(5) l'utente **C** si rivolge al server **S** che fornisce il servizio desiderato, inviandogli il ticket e il proprio nome ($\mathrm{ID}_C \| \bar{t}_S$).

Ogni volta che un utente inizia una nuova sessione di autenticazione, dovrà ripetere i passi 1-2; ogni volta che l'utente vuole accedere a un servizio differente, ripeterà i passi 3-4; per ogni sessione di servizio avviata dall'utente, esso dovrà ripetere il passo 5.

I due ticket sono costruiti come segue:

- $\bar{t}_G = E_{K_{T,A}}(\mathrm{ID}_C \| \mathrm{ID}_A \| \mathrm{ID}_T \| t_{s1} \| \ell_1)$;
- $\bar{t}_S = E_{K_{S,T}}(\mathrm{ID}_C \| \mathrm{ID}_A \| \mathrm{ID}_S \| t_{s2} \| \ell_2)$.

Nella struttura dei ticket abbiamo t_{si} che rappresenta il timestamp i-esimo (data di rilascio del ticket), e ℓ_i che rappresenta il tempo di vita i-esimo di un ticket. Se i tempi di vita dei ticket sono troppo corti o troppo lunghi, l'utente dovrà fornire spesso la password oppure ci si espone ad attacchi di replica:

- il ticket $\bar{t}_G$ può essere osservato sul canale e replicato per richiedere il ticket $\bar{t}_S$, impersonando l'utente;
- il ticket $\bar{t}_S$ può essere osservato sul canale e replicato per richiedere un servizio, impersonando l'utente;
- basta usare l'indirizzo IP dell'utente **C** per impersonarlo.

Per evitare questo tipo di attacchi bisogna autenticare client con server e vice versa.

9.4.3. Mutua autenticazione (Kerberos v4). Introduciamo i seguenti cambiamenti all'algoritmo mostrato nella precedente sezione:

- utilizziamo i timestamp anche in fase di autenticazione;
- inviamo i ticket assieme a un timestamp e cifriamo il tutto;
- usiamo delle stringhe contenenti la cifratura di di ID del client e del server di autenticazione assieme a un timestamp, e chiamiamole *authenticator*.

Implementiamo il protocollo con i nuovi elementi introdotti:

(1) **C** invia ad **A** il messaggio $\mathrm{ID}_C \| \mathrm{ID}_T \| t_{s1}$;
(2) **A** risponde a **C** con $E_{K_{C,A}} \left(K_{C,T} \| \mathrm{ID}_T \| t_{s2} \| \ell_2 \| \bar{t}_G \right)$;
(3) **C** invia a **T** il messaggio $\mathrm{ID}_S \| \bar{t}_G \| \mathbb{A}_{C1}$;
(4) **T** risponde con $E_{K_{C,T}} \left(K_{C,S} \| \mathrm{ID}_S \| t_{s4} \| \bar{t}_S \right)$;
(5) **C** invia ad **S** il messaggio $\bar{t}_S \| \mathbb{A}_{C2}$;
(6) **S** risponde con $E_{K_{C,S}} \left(t_{s5} + 1 \right)$.

I due authenticator sono costruiti come:

- $\mathbb{A}_{C1} = E_{K_{C,T}} \left(\mathrm{ID}_C \| \mathrm{ID}_A \| t_{s3} \right)$;
- $\mathbb{A}_{C2} = E_{K_{C,S}} \left(\mathrm{ID}_C \| \mathrm{ID}_A \| t_{s5} \right)$.

9.4.4. Reami. Un reame Kerberos consiste in un server di autenticazione, un TGS, uno o più server che espongono servizi e infine un gruppo di utenti che li richiedono; possiamo configurare la rete per contenere più di un reame e per effettuare l'autenticazione inter-reame, permettendo a utenti di un reame di usare servizi situati in un'altro reame.

9.4.5. Miglioramenti e limitazioni di Kerberos v5. Al contrario della versione 4 che usa la cifratura simmetrica DES, la versione successiva permette di adottare qualunque sistema di cifratura; inoltre Kerberos v5 non dipende dal protocollo IP, usa uno stile fisso per l'ordinamento dei Byte nel messaggio, specifica esplicitamente inizio e fine validità per i token e infine permette l'inoltro dell'autenticazione (un server a cui è autenticato un client può autenticarsi a sua volta a un'altro server).

CAPITOLO 10

Certificati e protocolli di rete

10.1. Infrastruttura a chiave pubblica (PKI)

10.1.1. PKI e certificati. Nella cifratura a chiave pubblica, la chiave usata per cifrare è pubblica e disponibile a tutti; la chiave per decifrare è privata (non viene condivisa) e anche se ricavabile dalla pubblica, il costo computazionale rende il problema irrisolvibile in pratica. Il punto debole di questo meccanismo è la fiducia nella chiave pubblica: proprio perché si tratta di una informazione pubblica, è necessario che sia autenticata.

Questo problema viene risolto dall'infrastruttura a chiave pubblica (public key infrastructure, PKI), un insieme di procedure e regole per generare, pubblicare e certificare le chiavi pubbliche; essa si basa su *certificati* (documenti elettronici) che legano la chiave pubblica all'entità che l'ha generata; un certificato garantisce quindi l'autenticità di una chiave.

Il processo che determina la validità di un certificato si chiama *validazione*; questo implica che sarà necessario fidarsi di un'autorità di certificazione (certification authority, CA) che emette certificati firmati da lei stessa.

Possiamo classificare i certificati nel modo seguente:

- i *certificati d'identità* forniscono e garantiscono le informazioni sull'identità di un'entità (indirizzo, chiave pubblica, ecc...);
- i *certificati credenziali* forniscono e garantiscono le informazioni sui diritti di accesso e di autorizzazione per un servizio o risorsa.

10.1.2. Fidarsi di un certificato. Tra le informazioni contenute in un certificato, quelle principali sono il nome del soggetto del certificato, la sua chiave pubblica e una firma (digest) delle due informazioni appena enunciate, firmata con la chiave privata della CA (tutti hanno la sua chiave pubblica e possono verificare che sia stata la CA a firmare):

$$\textsc{cert}\,(A) \coloneqq E_{CA}\left(A,\, K_A,\, E_{K_{CA}^{-1}}\left(h\,(A,\, K_A)\right)\right)$$

La chiave pubblica delle CA di livello più alto viene inserita nel browser o nel sistema operativo, dunque bisognerà riporre fiducia in questi software per fidarsi delle chiavi con le quali sono distribuiti; in ogni caso, fidarsi della CA implica soltanto il riporre fiducia nelle informazioni che essa ha firmato, non vuol dire fidarsi della CA in quanto praticamente affidabile.

Dal momento che le CA di alto livello sono poche e le entità che vogliono essere certificate sono numerose, spesso le CA firmano certificati per delle autorità di registrazione (registration authority, RA), le quali a loro volta possono firmare i certificati di RA di livello più basso; si viene così a formare una struttura ad albero, che racchiude catene di fiducia (catene di certificati), e permette di fidarsi del certificato foglia se si ripone fiducia nel certificato radice.

10.1.3. Certificati X.509. Si tratta di uno standard raccomandato dall'Unione Internazionale per le Telecomunicazioni (ITU); al contrario degli standard, non si incorre in sanzioni se non si seguono (è illegale vendere un prodotto non conforme agli standard approvati dall'ente nazionale di competenza).

La serie di raccomandazioni dell'ITU X.500 specifica il servizio di rubrica per gli utenti, e X.509 definisce i metodi di autenticazione per tali utenti, tramite certificati.

I certificati X.509 hanno una struttura costituita dai seguenti campi:

- versione
- numero di serie (ogni certificato ha uno univoco, rispetto alla sua CA emittente);
- nome della CA;
- periodo di validità;
- nome del soggetto (entità di cui si certifica la chiave pubblica);
- chiave pubblica del soggetto;
- firma (specifica l'algoritmo di firma).

Tramite il numero di serie è possibile revocare un certificato che non sia ancora scaduto, ponendolo in una lista di revoca; questo pone il problema di dover controllare le revoche effettuate, attraverso la lista di revoca che viene salvata periodicamente in una *cache*.

ESEMPIO 10.1. *Sia dato il certificato dell'utente A, costruito secondo le raccomandazioni X.509:*

$$\text{CERT}(A) = \left(A,\, K_A,\, h_{K_{CA}^{-1}}(A,\, K_A)\right)$$

Sappiamo che cifratura e firma usano l'algoritmo RSA con parametri $n = 221$, $K_{CA} = 35$, $K_A = 25$, $A = 200$; inoltre la funzione di hash $h(\circ)$ è definita come:

$$\forall x,\, y \in \mathbb{Z}_n \,:\, h(x,\, y) = (x \oplus (y \ll 3) \oplus (y \ll 4)) \bmod n$$

Verificare la correttezza dei dati nel contesto dell'algoritmo RSA.

✓Controlliamo che n sia valido, controllando che sia costituito da due fattori primi p e q:

$$n = 221 = 13 \cdot 17$$

Calcoliamo ora il suo toziente e il toziente del toziente:

$$\varphi(221) = 12 \cdot 16 = 2^6 \cdot 3 = 192; \qquad \varphi(\varphi(221)) = \varphi(192) = \left(2^6 - 2^5\right) \cdot (3 - 1) = 64$$

Verifichiamo che le chiavi pubbliche K_{CA}, K_A siano invertibili in modulo $\varphi(n)$:

$$\overbrace{7 \cdot 5}^{35} \perp \overbrace{2^6 \cdot 3}^{192} \implies 35^{-1} \equiv 11 \,(\text{mod}\, 192)$$
$$\overbrace{5^2}^{25} \perp \overbrace{2^6 \cdot 3}^{192} \implies 25^{-1} \equiv 169 \,(\text{mod}\, 192)$$

Calcolare la firma digitale associata al certificato.

✓Per prima cosa dobbiamo calcolare il digest della funzione $h(A,\, K_A)$ esprimendo nome utente e chiave pubblica associata su 8 bit, applicando la funzione di hash proposta dal quesito:

$$\begin{array}{ll} 11001000_2 = 200_{10} & (A) \\ 00011001_2 = 25_{10} & (K_A) \\ 11001000 & K_A \ll 3 \\ 10010001 & K_A \ll 4 \end{array}$$

Calcoliamo ora il digest applicando la somma bit a bit:

$$\begin{array}{ll} \cancel{11001000} & \oplus \\ \cancel{11001000} & \oplus \\ \underline{10010001} & = \\ 10010001 & (145) \end{array}$$

Infine possiamo ottenere la firma di $h\,(200,\,25) = 145$ usando la chiave $K_{CA}^{-1} \equiv 25^{-1} \equiv 11 \,(\mathrm{mod}\, 192)$:

$$145^{11} \equiv 202 \,(\mathrm{mod}\, 221)$$

Concludiamo che i valori che costituiscono il certificato sono $C_A = (200,\, 25,\, 202)$; possiamo verificare la chiave pubblica tramite la firma contenuta nel certificato:

$$\left[h_{K_{CA}^{-1}}\,(A,\, K_A)\right]^{K_{CA}} \equiv h\,(A,\, K_A) \,(\mathrm{mod}\, n) \to 202^{35} \equiv 145 \,(\mathrm{mod}\, 221)$$

Abbiamo ottenuto nuovamente il digest, verificando la congruenza e la validità della firma (solo la CA poteva calcolare la firma poiché solo essa conosce l'inverso di $K_{CA} \,\mathrm{mod}\, n$). □

10.2. Protocolli di rete sicuri

I protocolli di rete sono organizzati a strati, per questo motivo abbiamo le seguenti alternative nel localizzare i protocolli di sicurezza sullo *stack di rete*:

- livello di rete (IPSec);
  - trasparente alle applicazioni e agli utenti;
  - soluzione generale che garantisce la cifratura di tutto il traffico;
  - cifra solamente il traffico selezionato, evitando overhead non necessario.
- livello di trasporto (TLS);
  - usa connessioni TCP per inviare i dati cifrati;
  - deve essere integrato nelle applicazioni specifiche che usano TCP.
- livello applicativo.

10.2.1. Transport Level Security (TLS). Usa TCP per trasportare i dati cifrati, sfruttando le garanzie di integrità che esso offre; esso è implementato come un insieme di protocolli basati su TCP, di cui il principale è chiamato *Record Protocol* (fornisce servizi di trasporto sicuro ai livelli superiori). È documentato nell'RFC 5246.

Gli altri servizi che si basano sul Record Protocol e costituiscono TLS sono:

- Handshake Protocol (negozia le chiavi di sessione);
- Change Cipher Spec Protocol (cambia o seleziona algoritmo di cifratura per una connessione);
- Alert Protocol (distribuisce avvisi sulla connessione TLS agli altri peer);
- Heartbit Protocol (mantiene attiva la connessione, ritardando il timeout).

Il TLS implementa, oltre al concetto di connessione (già presente in TCP) anche il concetto di *sessione*: con l'Handshake Protocol si inizia una sessione, la quale può avere più connessioni al proprio interno; inoltre la sessione mantiene un set di chiavi che possono essere usate dalle connessioni instaurate nel suo contesto. Queste chiavi saranno rinnovate all'apertura di una nuova sessione; il concetto di sessione evita la negoziazione costosa dei parametri di sicurezza ad ogni nuova connessione.

Lo stato della sessione è costituito da:

- identificatore della sessione;
- certificato X.509 del peer (opzionale);

- algoritmo di compressione (usato sui dati prima della cifratura);
- specifica del cifrario (algoritmo per la cifratura);
- master secret (segreto da 48 Byte condiviso tra client e server);

Le singole connessioni possiedono uno stato definito dai seguenti parametri:

- numero casuale per server/client (generato per ogni connessione);
- segreto MAC per il server (usato per autenticare i messaggi del server);
- segreto MAC per il client (usato per autenticare i messaggi del client);
- chiave simmetrica del server (chiave simmetrica per cifrare sul server e decifrare sul client);
- chiave simmetrica del client (chiave simmetrica per cifrare sul client e decifrare sul server);
- vettori di inizializzazione;
- numeri di sequenza.

I servizi che il TLS Record Protocol sono i seguenti:

Confidenzialità lo Handshake Protocol definisce una chiave segreta condivisa usata per la crittografia del *payload* TLS;

Integrità lo Handshake Protocol inoltre definisce una chiave segreta usata come codice per autenticare i messaggi (MAC).

In ordine le operazioni che il Record Protocol effettua sono:

(1) segmentazione dei dati applicativi da inviare sul canale sicuro (c'è una finestra massima per i pacchetti TLS);
(2) i segmenti vengono compressi con l'algoritmo di compressione negoziato tramite Handshake Protocol;
(3) al segmento compresso viene concatenato un HMAC calcolato a partire dal messaggio da inviare, usando il segreto pattuito con Handshake Protocol;
(4) il frammento compresso con HMAC viene cifrato con l'algoritmo pattuito tramite Handshake Protocol;
(5) al segmento cifrato viene anteposto un *header* TLS per Record Protocol: esso sarà a sua volta il payload di un pacchetto TCP.

L'header TLS contiene il campo `Content type` (tipo di contenuto), che può indicare HTTPS o uno dei quattro protocolli su cui si basa Record Protocol.

10.2.2. Handshake TLS. L'instaurazione di una sessione TLS è preceduta dal seguente procedimento, chiamato *handshake*:

(1) client e server stabiliscono i protocolli per l'instaurazione della chiave, per la cifratura e per l'autenticazione dei messaggi;
(2) il server si autentica (eventualmente inviando il proprio certificato) e invia le chiavi per la cifratura al client;
(3) il client si autentica (eventualmente inviando il proprio certificato) e invia le chiavi per la cifratura al server;
(4) client e server scelgono l'algoritmo di cifratura con cui usare le chiavi instaurate, e l'inizializzazione viene finalizzata.

In generale, considerati un client **C** e un server **S**, le chiavi crittografiche vengono generate con la seguente procedura:

- **C** invia ad **S** in chiaro un numero casuale r_C di 32 Byte, di cui i primi 4 siano un timestamp;
- **S** invia a **C** in chiaro un numero casuale r_S di 32 Byte, di cui i primi 4 siano un timestamp;

- **C** invia ad **S** un numero casuale S_{PM} di 48 Byte cifrato con RSA e chiamato *pre-master secret*;
- **C** e **S** calcolano il numero S_{M} che dipende dai tre numeri scambiati e si ottiene concatenando i 3 blocchi seguenti:

$$S_{\text{M}} = \begin{cases} \text{MD5}\left(S_{\text{PM}} \| \text{SHA1}\left(A \| S_{\text{PM}} \| r_C \| r_S\right)\right) \\ \text{MD5}\left(S_{\text{PM}} \| \text{SHA1}\left(BB \| S_{\text{PM}} \| r_C \| r_S\right)\right) \\ \text{MD5}\left(S_{\text{PM}} \| \text{SHA1}\left(CCC \| S_{\text{PM}} \| r_C \| r_S\right)\right) \end{cases}$$

  dove ciascuno dei blocchi è costituito da 128 bit (16 Byte);
- **C** e **S** calcolano il proprio blocco di chiavi di sessione a partire dal segreto S_{M} (6 chiavi segrete, la metà per cifrare da client a server e vice-versa);
  - chiave di cifratura;
  - chiave per MAC;
  - vettore di inizializzazione se il cifrario è usato in modo concatenato (CBC).

10.2.3. HTTPS. Si tratta di un protocollo applicativo che combina HTTP con TLS per garantire una comunicazione sicura tra client e server; è documentato nell'RFC 2818. Quando viene richiesta una risorsa tramite HTTPS, vengono cifrati i seguenti elementi:

- l'URL del documento richiesto;
- il contenuto del documento;
- i contenuti dei moduli del browser;
- i *cookie* inviati dal client al server e vice-versa;
- l'header HTTP.

L'agente che ha il ruolo di client HTTP agisce anche come client TLS, prima avviando un handshake TLS e poi effettuando la richiesta HTTP tramite il canale sicuro; una connessione HTTP ha tre "strati":

- HTTP
  - un client richiede una connessione HTTP a un server, e tale richiesta è inviata allo strato inferiore;
- TLS
  - una sessione TLS è stabilita tra client e server;
  - la richiesta per stabilire la connessione comincia con una connessione allo strato inferiore;
- TCP
  - una connessione è stabilita tra le entità TCP di client e server.

Infine, possiamo chiudere una connessione HTTPS partendo dallo "strato" superiore e terminando ciascuna connessione in cascata verso gli strati inferiori; a livello di HTTP, la connessione si chiude tramite l'header `Connection: close`, mentre a livello di TLS si usa l'Alert Protocol per inviare l'avviso di chiusura.

Se la connessione HTTPS si interrompe in modo inaspettato sullo strato TCP, questo può indicare un errore di implementazione o un possibile attacco: in ogni caso è opportuno mostrare un errore.

10.3. Posta elettronica sicura

10.3.1. Simple Mail Transfer Protocol (SMTP). All'interno dell'RFC 5598 vi è la specifica dell'architettura per l'*Internet Email*, fondata su agenti chiamati MTA (Mail Transfer Agent), che trasferiscono i messaggi tra loro fino a consegnarli alla destinazione corretta. I messaggi di posta elettronica scambiati usando questa

architettura vengono inviati in chiaro, attraverso una connessione TCP, tramite il protocollo SMTP (Simple Mail Transfer Protocol).

In pratica, il protocollo si basa su una conversazione tra client e server, basata su comandi testuali (in codifica ASCII); la porta predefinita per le connessioni in ingresso a un server SMTP è la numero 25.

10.3.2. STARTTLS. La prima strategia che è stata usata per rendere SMTP sicuro si basava sull'uso di TLS per cifrare la comunicazione col server: l'RFC 3207 definisce l'estensione dell'SMTP sicuro sopra TLS.

Questa estensione si basa sull'implementazione del comando `STARTTLS`, utilizzato nel modo seguente:

- il server risponde all'handshake del client con un codice `250 STARTTLS`, suggerendo al client l'uso dell'estensione sicura;
- il client invia al server il comando STARTTLS per avviare la connessione sicura tramite TLS; il server assegnerà la porta 587 alla connessione `SMTP + TLS`;
- se il server risponde col codice `220`, viene instaurata una connessione TLS tra le due parti.

Esiste un approccio differente (deprecato) implementato sotto forma di SMTPS, che utilizza una connessione TLS sulla porta 465 del server e all'interno di essa viene effettuata la comunicazione SMTP.

10.3.3. Post Office Protocol (POP3). Se vogliamo interrogare il server con la nostra casella email (MTA) per scaricare i messaggi di posta e cancellarli dal server, possiamo usare il protocollo per la ricezione POP3; esso sfrutta una connessione sulla porta standard 110, trasferendo in chiaro ciascuna informazione (anche la password!); per questo motivo è stata introdotta l'estensione con l'implementazione del comando STARTTLS, la quale permette di instaurare una connessione cifrata dopo l'handshake iniziale.

Esiste anche la variante POP3S, che permette di collegarsi al server tramite una connessione TLS sulla porta 995.

In generale, IMAP offre più funzioni e autenticazione più forte rispetto a POP3

10.3.4. Internet Mail Access Protocol (IMAP). Se vogliamo controllare i messaggi di posta presenti nella casella email senza rimuoverli dal server (MTA), allora possiamo usare il protocollo IMAP, connettendoci con TCP alla porta 143 del server e inviandogli i comandi adatti; è opportuno utilizzare l'estensione sicura di IMAP, che sfrutta l'implementazione del comando STARTTLS.

Possiamo inoltre usare la variante IMAPS, che permette di collegarci al server tramite una connessione TLS sulla porta 993.

10.3.5. Struttura dei messaggi. L'RFC 5322 definisce gli standard per i messaggi testuali di posta elettronica, caratterizzati dalla suddivisione del contenuto (solo testuale, ASCII a 7bit) in *envelope* (comprende i metadati come data di invio, mittente, destinatario, ecc...) e *content* (comprende il testo del mesaggio vero e proprio).

Successivamente, per risolvere il problema dell'invio di file binari che avessero bisogno di una codifica a 8bit rispetto ai 7bit concessi da ASCII, è stata aggiunta l'estensione **MIME** (Multipurpose Internet Mail Extension) al protocollo SMTP: essa permette di convertire una codifica in 8bit in un'altra solo testuale a 7bit, permettendo il trasferimento di file binari e digitali attraverso il protocollo SMTP.

10.3.6. Minacce alla sicurezza della posta elettronica. Possiamo riassumere le minacce alla sicurezza dell'Internet Email nelle seguenti categorie:

(1) Autenticità: il mittente potrebbe non essere quello rappresentato dal metadato nell'header del messaggio;
(2) Integrità: il messaggio che è arrivato al destinatario potrebbe essere stato modificato durante l'inoltro tra i Mail Transfer Agent;
(3) Confidenzialità: il contenuto del messaggio potrebbe essere stato letto da un soggetto differente da quello associato all'account del destinatario;
(4) Disponibilità: potrebbe essere possibile negare agli utenti la disponibilità del servizio di inoltro e consegna dei messaggi.

Per risolvere o mitigare questi problemi, sono state introdotte le seguenti estensioni o varianti dei protocolli mostrati in precedenza:

- usare STARTTLS sia per l'SMTP sia per la ricezione dei messaggi (POP / IMAP), garantendo autenticità, integrità, confidenzialità e non-ripudio;
- usare S/MIME per trasmettere file binari garantendo l'integrità del contenuto del messaggio;
- usare DNSSEC per garantire autenticità e integrità della comunicazione col servizio DNS per determinare il Mail Transfer Agent col quale comunicare;
- usare DANE per autenticare le chiavi pubbliche associate ai nomi dei domini;
- usare SPF per autorizzare solo a un determinato intervallo di indirizzi IP per essere mittenti verso un server mail specifico;
- usare DKIM per permettere a un MTA di firmare campi specifici dell'header e il contenuto di un messaggio, e di verificare la sua integrità;
- usare DMARC per segnalare ai mittenti l'efficacia delle loro politiche SPF e DKIM, e segnalare ai destinatari quali azioni è opportuno intraprendere nel caso di vari scenari di attacchi.

10.3.7. Pretty Good Privacy (PGP). Sistema alternativo all'S/MIME, che offre delle funzionalità simili ma le implementa diversamente; in particolare S/MIME si basa su certificati per autenticare gli utenti, mentre PGP usa il principio del "*web of trust*" (si si fida della chiave pubblica firmata da qualcuno di cui ci si fida).

PGP permette di autenticare e cifrare i messaggi, combinando anche queste funzionalità; esse sono realizzate come segue.

Autenticazione del messaggio.

(1) **A** calcola il digest $\overline{m}$ del messaggio m da autenticare (per esempio) usando l'hash SHA-1;
(2) **A** firma il digest, usando (di solito) RSA, elevando $\overline{m}$ all'esponente segreto $d \bmod n$;
(3) **A** invia a **B** in chiaro la firma del digest $\overline{m}^d \bmod n$;
(4) **B** si fida della chiave pubblica di **A**, e verifica la firma ricevuta elevandola all'esponente pubblico $e \bmod n$ e poi confrontando il risultato col digest del messaggio;
(5) Se $\left(\overline{m}^d\right)^e \bmod n$ corrisponde al digest del messaggio m e **B** si fida della chiave pubblica di **A**, il messaggio è considerato autenticato e viene accettato.

Cifratura del messaggio.

(1) **A** genera la chiave di sessione simmetrica k_s come un numero casuale di 128 bit (essa sarà usata una volta sola);
(2) **A** cifra il messaggio m con la chiave di sessione con 3DES, poi cifra la chiave di sessione k_s con RSA;
(3) **A** invia m e k_s cifrati a **B**;
(4) **B** decifra la chiave di sessione k_s e a sua volta usa la chiave di sessione per decifrare il messaggio m.

In questo algoritmo di cifratura viene usata una chiave simmetrica per cifrare il messaggio, per velocizzare la decifratura (la sicurezza dell'algoritmo non viene compromessa da questa scelta, poiché la chiave simmetrica è comunque inviata dopo essere stata cifrata con RSA). Si osserva che non è richiesta la fiducia nel mittente del messaggio.

Autenticazione e cifratura.

Scegliamo di firmare prima e cifrare poi per mitigare alcuni tipi di attacchi; per maggiori informazioni fare riferimento al seguente articolo: *Defective Signs & Encryption*[1]

(1) **A** calcola il digest $\overline{m}$ del suo messaggio m, e firma il digest con RSA, ottenendo $\text{sig}(\overline{m}) \coloneqq \overline{m}^d \bmod n$;
(2) **A** genera una chiave di sessione simmetrica k_s con 128 bit casuali;
(3) **A** cifra $\text{sig}(\overline{m}) \| m$ tramite chiave di sessione k_s usando (per esempio) 3DES, poi cifra k_s tramite RSA;
(4) **A** invia messaggio con firma e chiave di sessione cifrati a **B**;
(5) **B** decifra la chiave di sessione k_s (tramite la propria chiave privata), e a sua volta la usa per decifrare il messaggio con firma;
(6) **B** verifica che la firma RSA $\text{sig}(\overline{m})$ corrisponda a quella del messaggio m (usando la chiave pubblica di **A**).

10.4. IPSec

Protocollo di sicurezza per il livello di rete, che fornisce il servizio di trasporto dei pacchetti IP in maniera sicura; risulta trasparente ai livelli superiori dello stack di rete. Questo protocollo ricopre tre aree funzionali:

- *autenticazione*: verifica del mittente e dell'integrità dei pacchetti;
- *confidenzialità*: il contenuto dei pacchetti è cifrato;
- *gestione delle chiavi*: scambio sicuro delle chiavi affidabili.

IPSec prevede varie applicazioni e tipi di trasporto differenti; noi siamo interessati al tunneling.

10.4.1. Tunneling. Consiste nel prendere un pacchetto IP, cifrarlo, autenticarlo, e usarlo come payload di un'altro pacchetto IP; questo nuovo pacchetto ha la struttura di un pacchetto IP, dunque può viaggiare in una rete di dispositivi che non conoscono necessariamente IPSec.

La comunicazione in *tunnel mode* viene realizzata da una funzione combinata di autenticazione e cifratura chiamata *ESP* (Encapsulating Security Payload, il pacchetto ESP diventa payload del pacchetto autenticato), assieme allo scambio delle chiavi.

In pratica la tunnel mode realizza una *VPN* sicura (Virtual Private Network, rete privata virtuale) attraverso Internet.

Oltre a poter mettere in sicurezza tutto il traffico perimetrale di una rete se implementato su un *firewall* ed essere trasparente a utenti e applicazioni, IPSec viene applicato per mettere in sicurezza il traffico dei protocolli di routing (come OSPF).

[1] `http://world.std.com/~dtd/sign_encrypt/sign_encrypt7.html`

Transport mode. In questa modalità di funzionamento, IPSec fornisce protezione ai protocolli di livello superiore, ovvero il payload dei pacchetti IP viene estratto e cifrato. Al contrario, la tunnel mode protegge l'intero pacchetto IP, compreso l'header e altri metadati.

Tunnel mode. Per realizzare questa modalità di funzionamento la specifica prevede di:

- aggiungere al pacchetto IP campi addizionali per il protocollo ESP;
- l'intero pacchetto IP insieme ai campi addizionali sono cifrati e considerati il payload di un nuovo pacchetto IP;
- nessun dispositivo intermedio è in grado di osservare il payload di questo pacchetto;
- ESP cifra e (opzionalmente) autentica il pacchetto IP interno (payload).

Se i due nodi che comunicano via IPSec sono parte di una SA (Security Association, per esempio due *gateway*, router o firewall che implementano IPSec), i nodi ad essi connessi sulla rete locale non hanno necessità di implementare IPSec.

10.4.2. Security policy. La cifratura e l'autenticazione in IPSec viene determinata pacchetto per pacchetto, in base a un'insieme di regole chiamato *security policy*; queste regole sono determinate dall'interazione tra due database:

- SAD (Security Association Database): memorizza le SA registrate e i parametri SPI corrispondenti;
- SPD (Security Policy Database): memorizza le regole relative a ciascuna SA, sotto forma di sottoinsiemi di traffico IP e relazioni uno-a-molti.

In ciascun pacchetto IP, la SA corrispondente è identificata da indirizzo di destinazione e i parametri SPI (Security Parameters Index) nell'header dell'estensione ESP;

10.4.3. Internet Key Exchange (IKE). Insieme di protocolli per generare e trasferire le chiavi necessarie alla comunicazione sicura con IPSec; in particolare, vengono usati certificati X.509 e lo scambio Diffie-Hellman per instaurare un segreto di sessione condiviso, derivando due paia di chiavi per ciascuna comunicazione (una chiave per trasmissione e una per ricezione, due paia per ridondanza).

Per configurare una SA si usano:

- il protocollo di instaurazione della chiave di Oakley (generico, simile a Diffie-Hellman);
- il protocollo di instaurazione delle SA chiamato ISAKMP (RFC 2408), che fornisce un *framework* per lo scambio delle chiavi e per l'autenticazione.

APPENDICE A

Introduzione alla crittografia

Cenni storici. Il termine crittografia deriva dal greco $\kappa\rho\upsilon\pi\tau\acute{o}\zeta$ (kryptós - segreto) e $\gamma\rho\alpha\varphi\acute{\eta}$ (grafi - scrittura); tra gli esempi di crittografia dal passato il più famoso è il cifrario di Cesare: si tratta di un cifrario a scorrimento ciclico, che consisteva nello scrivere le lettere dell'alfabeto su due anelli per poi ruotarne uno rispetto all'altro di $k = +3$ posizioni; in questo modo si ottiene $A \to D$, $B \to E$, $\ldots Z \to C$. Non si trattava di un cifrario robusto, ma veniva usato quando gli avversari dell'Impero Romano erano i Galli: si rivelò un metodo più che sufficiente.

Comunicazione sicura. In generale una comunicazione sicura tra due parti si svolge nel modo seguente: sia A il mittente del messaggio, B il destinatario, ed E un intruso che abbia accesso al canale di comunicazione. L'intruso può essere *passivo* (intercetta i messaggi senza modificarne il flusso) oppure *attivo* (modifica il contenuto dei messaggi).

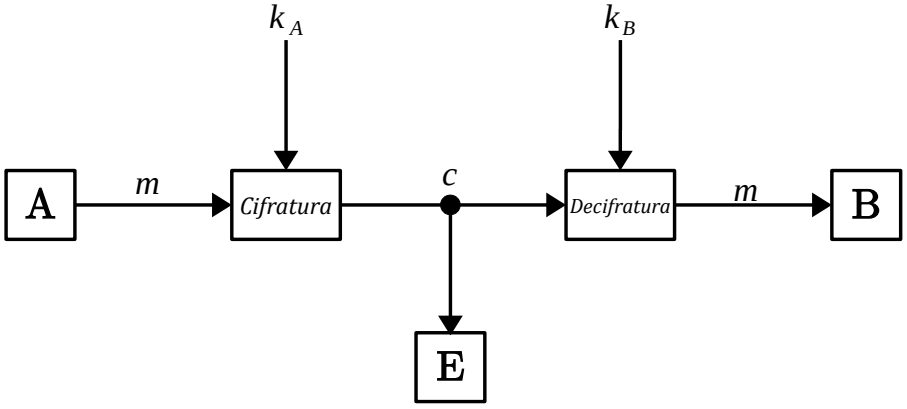

Figura A.0.1. Scenario fondamentale di comunicazione

A invia un *plaintext* m (testo in chiaro), lo codifica usando una funzione di cifratura (un algoritmo crittografico) che prende in ingresso anche la sua chiave k_{A} e ottiene un *ciphertext* c (messaggio cifrato); il messaggio c giunge a ***B***, il quale usa una funzione di de-cifratura — tramite la sua chiave k_{B}, che è associata in qualche modo alla chiave k_{A} — per ottenere nuovamente il *plaintext* m inviato da ***A***.

E potrebbe avere le seguenti intenzioni malevole rispetto alla comunicazione tra ***A*** e ***B***:

- leggere il messaggio e comprenderne il contenuto;
- ottenere la chiave;
- corrompere il contenuto del messaggio;
- impersonare A senza che B se ne accorga.

L'intruso può mettere in atto i seguenti tipi di attacchi sull'algoritmo di cifratura usato nella comunicazione:

- CIPHERTEXT-ONLY: avendo a disposizione il testo cifrato, si cerca di ricavarne delle informazioni (attacco più comune);

- KNOWN PLAINTEXT: avendo a disposizione una coppia di testo cifrato e testo in chiaro corrispondente, si confrontano i due cercando di ottenere informazioni sulla chiave;
- CHOSEN PLAINTEXT: avendo a disposizione la stessa implementazione dell'algoritmo utilizzato per cifrare il messaggio, si scelgono dei testi in chiaro da cifrare e si osservano i testi cifrati in uscita, per cercare di ricavare informazioni sull'implementazione;
- CHOSEN CIPHERTEXT: avendo a disposizione la stessa implementazione dell'algoritmo utilizzato per decifrare il messaggio, si scelgono dei testi cifrati da decifrare e si osservano i testi in chiaro in uscita, per cercare di ricavare informazioni sull'implementazione.

Sicurezza e segretezza. Giulio Cesare basava la sicurezza del proprio algoritmo di cifratura sul fatto che i possibili avversari non ne conoscessero il funzionamento; il crittografo olandese Auguste Kerckhoffs, enunciò ne *'La cryptographie militaire'* (1883) il principio di Kerchoffs:

Principio di Kerchoffs

La sicurezza di un sistema di cifratura è basata sulla segretezza della chiave (assumere sempre che il nemico conosca l'algoritmo di cifratura)

Da questa considerazione segue che la chiave utilizzata deve essere lunga, complessa, e in generale essere costruita per evitare che sia possibile indovinarla.

Claude Elwood Shannon, che scrisse cinque articoli che cambiarono la storia della comunicazione dell'informazione, tra cui un articolo sulla crittografia[1], espresse lo stesso principio in maniera molto incisiva con le parole "Il nemico conosce il sistema" (frase nota come massima di Shannon).

È interessante notare come si è passati dal fondare la sicurezza del sistema sulla segretezza dell'algoritmo alla segretezza della chiave; il passo successivo fu il sistema a *chiave pubblica*: gli algoritmi usati sono noti e accessibili a tutti, e una chiave del mittente (quella pubblica) è resa nota a tutti; tramite la chiave pubblica è possibile cifrare i messaggi, tuttavia la chiave per decifrare, associata alla chiave pubblica, è mantenuta riservata (si parla di *chiave privata*).

Fondamentale è il fatto che, non ostante esista una regola (formula o algoritmo) che permetta di associare la chiave pubblica a quella privata, per un intruso qualunque è impossibile, dal punto di vista computazionale, risalire alla chiave privata attraverso quella pubblica. Solo il mittente che possiede la chiave privata è in grado di computare questa associazione, poiché egli deve aver ricavato la chiave pubblica a partire da quella privata (l'operazione inversa risulta molto più difficile).

Algoritmi noti. Gli algoritmi a chiave simmetrica hanno una coppia di chiavi, per cifratura e de-cifratura, che sono entrambe segrete: DES, AES; si pone il problema di scambiare col destinatario la chiave di de-cifratura, utilizzando un canale sicuro.

Con i sistemi di cifratura a chiave pubblica questo problema non si pone, tuttavia si pone il nuovo problema dell'autenticità delle chiavi pubbliche in circolazione; è stato introdotto il meccanismo dei certificati, da associare alle chiavi pubbliche, per garantire la loro provenienza e affidabilità.

[1] *"Communication Theory of Secrecy Systems"* (1949), Bell System Technical Journal

Numeri interi grandi. Lavoreremo prevalentemente con numeri interi (positivi e negativi) di elevato ordine di grandezza; ecco un esempio per effettuare un calcolo approssimato.

ESEMPIO A.1. *Calcolare in modo approssimato il valore di* 2^{35}.

✓Sfruttando le proprietà delle potenze e la costante informatica $2^{10} \sim 1000 = 10^3$, possiamo ragionare nel modo seguente:

$$2^{35} = 2^{30} \cdot 2^5 = \left(2^{10}\right)^3 \cdot 32 \simeq \left(10^3\right)^3 \cdot 32 = \boxed{32 \times 10^9}$$

□

Trattare interi grandi è importante nell'ambito degli algoritmi di cifratura: prendendo l'algoritmo a chiave simmetrica DES come esempio, è ragionevole pensare che una chiave di 56 bit non sia sufficientemente sicura; usando le considerazioni fatte nell'Esempio A.1 otteniamo che $2^{56} \simeq 10^{16}$, ed essendo in possesso di una macchina in grado di ottenere una chiave in $1ns$, allora sarebbero necessari 27 mesi per ottenere questa chiave; ovviamente si può ridurre questo tempo aumentando il numero di macchine impiegate.

Al giorno d'oggi sono considerate sicure chiavi a 265 bit ($\sim 10^{77}$): per analizzare in modo esaustivo un simile spazio delle chiavi sarebbero necessari 10^{60} anni, nelle condizioni descritte in precedenza!

APPENDICE B

Campi di Galois

B.1. Costruzione con polinomi

Per comprendere la struttura e le proprietà dei campi di Galois, analizziamo il seguente esempio: sia dato l'insieme $\mathcal{GF}(4) = \{0, 1, \omega, \omega^2\}$,che chiamiamo "campo di Galois 4"; su questo insieme siano definite le seguenti proprietà, posto $\forall x \in \mathcal{GF}(4)$:

(1) Ogni elemento del campo ha un inverso additivo: $x + 0 = x$;
(2) $x + x = 0$;
(3) Ogni elemento del campo ha un inverso moltiplicativo: $1 \cdot x = x$;
(4) $\omega + 1 = \omega^2$;
(5) somma e prodotto godono delle proprietà commutativa, associativa e distributiva.

Usando queste proprietà possiamo calcolare $\omega^2 + 1 \xrightarrow{4} \omega + 1 + 1 \xrightarrow{2} \omega$. Notiamo dalle proprietà enunciate, che con 0 indichiamo l'elemento neutro rispetto alla somma e con 1 l'elemento neutro rispetto al prodotto; rispetto alla somma inoltre, ogni elemento è l'inverso di sé stesso. L'inverso moltiplicativo invece, non è definito per 0, e vale 1 per l'elemento 1; negli altri casi, (ω e ω^2) si ha che l'inverso è ω^2; infatti si ricava che

$$\omega \cdot \omega^2 = \omega \cdot (\omega + 1) = \omega + \omega^2 = \omega + \omega + 1 = 1 \implies \omega \cdot \omega^2 = 1 \implies \boxed{\omega^{-1} = \omega^2}$$

Vale alche l'opposto, ovvero l'inverso di ω^2 è ω.

Possiamo enunciare la definizione di *campo*:

Campo

DEFINIZIONE B.1. Un campo è un insieme che abbia almeno una operazione di somma e una di prodotto, includa come minimo gli elementi 0 e 1, dove 0 è l'elemento neutro rispetto alla somma e 1 l'elemento neutro rispetto al prodotto; l'insieme deve soddisfare inoltre le proprietà (1), (3) e (5) tra quelle elencate in precedenza.

Questo implica che un insieme che soddisfa la definizione di campo sarà chiuso rispetto alle operazioni di somma e prodotto.

ESEMPIO B.1. L'insieme dei numeri reali forma un campo, infatti in esso sono contenuti 0 e 1, e per ciascun numero reale esiste un inverso definito all'interno dell'insieme.

L'insieme degli interi non è un campo, poiché l'inverso degli elementi non appartiene all'insieme.

Consideriamo l'insieme $\mathbb{Z}_p$ con p un primo, e l'insieme $\mathbb{Z}_n$ con n composto: il primo è un campo, mentre il secondo no, dato che i suoi elementi hanno inverso moltiplicativo solo se primi relativi rispetto a n.

Infine, l'insieme delle matrici 2×2 non forma un campo, perché non gode della proprietà commutativa e non tutti i suoi elementi sono invertibili. □

Vale in generale la seguente proprietà, che lega i campi di Galois alla cardinalità degli insiemi di interi:

DEFINIZIONE B.2. Per ogni intero, esprimibile come potenza di un primo, esiste esattamente un solo campo finito di p^n elementi, chiamato campo di Galois p^n:

$$\forall p^n \exists! \, \mathcal{GF}\left(p^n\right) \, : \, \left|\mathcal{GF}\left(p^n\right)\right| = p^n \tag{B.1.1}$$

Possiamo domandarci se l'insieme dei residui modulo p^n ($\mathbb{Z}_{p^n}$) sia un campo di Galois: esso non lo è, perché solo i suoi elementi primi relativi rispetto a p^n ammettono inverso. Usiamo la seguente procedura, che permette di costruire in generale un insieme che sia un campo $\mathcal{GF}\left(p^n\right)$ tramite l'impiego dei polinomi (avremmo potuto rappresentare il campo di Galois tramite qualsiasi simbolo di qualunque alfabeto).

Usiamo l'insieme dei residui modulo p, e l'insieme di tutti i polinomi nella variabile x, di qualunque grado, con coefficienti modulo p:

$$\mathbb{Z}_p, \quad \mathbb{Z}_p\left[x\right] = \sum_{i=0}^{n} a_i x^i \, \forall a_i \in \mathbb{Z}_p$$

prendiamo un polinomio $P\left(x\right)$ di grado n, con coefficienti a_i in $\mathbb{Z}_p$, che sia irriducibile (impossibile da scomporre nel prodotto di polinomi di grado inferiore).

ESEMPIO B.2. Prendiamo $p = 2$, $n = 2$. Scriviamo allora $P\left(x\right) = x^2 + 1 \bmod 2$

Notiamo che $P\left(x\right)$ è riducibile, infatti $x^2 + 1 \equiv \left(x + 1\right) \cdot \left(x + 1\right) \left(\bmod 2\right)$.

Invece, possiamo usare il polinomio irriducibile $x^2 + x + 1$ per definire $\mathcal{GF}\left(4\right)$. Per provare che un polinomio sia riducibile o meno è necessario provare la divisione tra polinomi, tenendo presente che le operazioni sono modulo p, provando tutti i possibili divisori fino al grado $n/2$. □

OSSERVAZIONE B.1. Adesso definiamo il campo di Galois p^n nel modo seguente:

$$\mathcal{GF}\left(p^n\right) = \mathbb{Z}_p\left[x\right] \bmod P\left(x\right) \tag{B.1.2}$$

Possiamo osservare un'analogia con l'insieme dei resti interi modulo p:

primo p	irriducibile $P\left(x\right)$
$\mathbb{Z}_p$	$\mathbb{Z}_p\left[x\right] \bmod P\left(x\right)$
residui della divisione per un primo p	residui della divisione per un polinomio irriducibile $P\left(x\right)$ di grado n
cardinalità p	cardinalità p^n

Un generico polinomio ottenuto secondo questa costruzione è al massimo di grado $n - 1$, essendo il resto di una divisione per un polinomio di grado n; possiamo scrivere il polinomio $P\left(x\right)$ nella seguente forma:

$$P\left(x\right) = a_{n-1} \cdot x^{n-1} + a_{n-2} \cdot x^{n-2} + \ldots + a_1 \cdot x + a_0$$

I possibili polinomi di questa forma sono p^n, dato che ogni singolo coefficiente è in modulo p, e si hanno in totale n coefficienti.

OSSERVAZIONE B.2. Qualunque polinomio appartenente a un campo di Galois ammette inverso. Prendiamo dunque $a\left(x\right) \in \mathcal{GF}\left(p^n\right)$, definito come descritto in questa sezione, l'inverso di $a\left(x\right)$ è l'elemento $a^{-1}\left(x\right) \in \mathcal{GF}\left(p^n\right)$ tale che

$$a^{-1}\left(x\right) \cdot a\left(x\right) \equiv 1 \left(\bmod P\left(x\right)\right)$$

Questo vale se $\text{MCD}\left(a\left(x\right), \, P\left(x\right)\right) = 1$, ma ciò è sicuramente vero perché $P\left(x\right)$ è definito irriducibile; se avessimo un polinomio composto (riducibile) al posto di $P\left(x\right)$, avremmo un'analogia con l'insieme $\mathbb{Z}_n$ (n composto), nel quale solo gli elementi primi relativi rispetto a n hanno l'inverso.

Osservazione B.3. Per definire un campo di Galois p^n abbiamo bisogno di un polinomio irriducibile di grado n; tuttavia vi è più di un solo polinomio irriducibile, fissato il grado. Si verifica che, usando polinomi irriducibili diversi dello stesso grado, si ottiene lo stesso campo di Galois da entrambi: esso è uno e uno solo, da (B.1.1).

Dal punto di vista algebrico, si ottengono due campi *isomorfi*, ovvero le relazioni biunivoche tra elementi di uno stesso insieme coincidono con quelle dell'altro insieme.

Definizione B.3. Un campo di Galois può essere definito tramite i resti $R(x)$ che derivano dalla divisione dei polinomi di qualunque grado $\mathbb{Z}_p[x]$ per il polinomio irriducibile $P(x)$:

Campo di Galois

$$\mathcal{GF}(p^n) \doteq R(x) \equiv \mathbb{Z}_p[x] \pmod{P(x)} \tag{B.1.3}$$

B.2. Polinomi come elementi generatori

Come visto nell'Osservazione B.3, il grado massimo di $R(x)$ sarà $n-1$, mentre i possibili polinomi di grado fino a $n-1$ sono in totale p^n. Nella stessa Osservazione, abbiamo mostrato un'analogia tra l'insieme dei residui modulo p e i campi di Galois definiti come residui della divisione dei polinomi con coefficienti modulo p, per $P(x)$.

Possiamo estendere questa analogia al concetto di radice primitiva (1.9.1), ovvero anche in $\mathbb{Z}_p[x] \bmod P(x)$ ci saranno dei polinomi $g(x)$, chiamati elementi generatori, tali che

$$g(x)^k \equiv 1 \pmod{P(x)}$$

Il minimo n per il quale si verifica la precedente condizione è chiamato ordine (1.9.1) del campo, e vale:

$$\text{ord}(g(x)) = p^n - 1 \tag{B.2.1}$$

Questo polinomio $g(x)$ sarà una radice primitiva del campo; il numero di radici primitive di un campo sarà determinato dal toziente dell'ordine, come $\varphi(p^n-1)$

$\mathbb{Z}_p$	$\mathbb{Z}_p[x] \bmod P(x)$
$\text{ord}(\alpha) = p-1$	$\text{ord}(g(x)) = p^n-1$
$\varphi(p-1)$	$\varphi(p^n-1)$

Osservazione B.4. Chiediamoci se esiste un caso in cui tutti gli elementi di un insieme $\mathbb{Z}_p$ sono anche elementi generatori; dovrebbe valere $\varphi(p-1) = p-1$, ma questo è impossibile, poiché p è primo e $p-1$ risulta pari.

Nel campo $\mathbb{Z}_p[x] \bmod P(x)$ invece deve valere $\varphi(p^n-1) = p^n-1$, e ciò è possibile poiché p^n-1 è primo per alcuni n, a patto che valga $p=2$ (per esempio, con $p=2$ e $n=5$, vale $p^n-1=31$ e il campo $\mathcal{GF}(2^5)$ avrà tutti gli elementi che sono anche generatori del campo).

Esempio B.3. Consideriamo il campo $\mathcal{GF}(256)$, ovvero $p=2$ e $n=8$; possiamo definire questo campo a partire da qualunque polinomio irriducibile $P(x)$ di grado 8, con i coefficienti modulo 2: prendiamo

Il polinomio di $\mathcal{GF}(256)$ mostrato in questo esempio è quello utilizzato nel cifrario AES

$$P(x) = x^8 + x^4 + x^3 + x^2 + x + 1$$

Gli elementi del campo, resti della divisione con $P(x)$, avranno grado massimo 7; possiamo scriverli come

$$R(x) = b_7 \cdot x^7 + b_6 \cdot x^6 + \ldots + b_1 \cdot x + b_0$$

I coefficienti di un polinomio $R(x) \in \mathcal{GF}(256)$ sono 8 e sono tutti in modulo 2 (valgono 0 o 1): essi possono essere rappresentati da un Byte, e l'algoritmo dell'AES usa questa rappresentazione per effettuare operazioni tra polinomi, all'interno di $\mathcal{GF}(256)$.

B.3. Cardinalità dei polinomi irriducibili

Abbiamo già concluso che il numero di elementi di un campo di Galois p^n è proprio p^n ($|\mathcal{GF}(p^n)| = p^n$); per definire il campo è necessario un polinomio irriducibile $P(x)$ di grado n coi coefficienti modulo p. Ci chiediamo il numero di tali polinomi; indichiamo allora il numero di polinomi irriducibili con coefficienti modulo p e grado n come:

Numero di polinomi irriducibili

$$N_{irr}(p,\, n) = \frac{1}{n} \sum_{i=1,\, i \backslash n}^{n} \mu\left(\frac{n}{i}\right) \cdot p^i \tag{B.3.1}$$

La funzione $\mu()$ si chiama funzione di Möbius ed è definita nel modo seguente:

Funzione di Möbius

$$\mu(x) = \begin{cases} 0 & x = \text{ prodotto di primi, almeno uno ripetuto} \\ 1 & x = 1 \\ (-1)^k & x = \text{ prodotto di } k \text{ primi distinti} \end{cases} \tag{B.3.2}$$

Esempio B.4. *Prendiamo i polinomi con coefficienti in modulo 2 e grado 6: determinare il numero di polinomi irriducibili tra di essi.*

✓Sappiamo che il numero totale di polinomi con queste caratteristiche è $p^n = 2^6 = 64$; usiamo la formula (B.3.1):

$$\begin{aligned} N_{irr}(2,\, 6) &= \frac{1}{6}\left(\mu(6) \cdot 2^1 + \mu(3) \cdot 2^2 + \mu(2) \cdot 2^3 + \mu(1) \cdot 2^6\right) \\ &= \frac{1}{6}\left(2 - 2^2 - 2^3 + 2^6\right) = \frac{54}{6} = \boxed{9} \end{aligned}$$

Abbiamo usato nella sommatoria i valori da 0 a 6 che dividono 6, ovvero $i \in \{1,\, 2,\, 3,\, 6\}$. □