Università degli Studi di Milano – Bicocca
Facoltà di Scienze Matematiche, Fisiche e Naturali
Corso di Laurea in Informatica

Appunti di Crittografia per il corso di Teoria dell'Informazione

Prof. Giancarlo Mauri

Appunti scritti dal Dott. Alberto Leporati

Anno Accademico 2001–2002

1 Introduzione – Crittosistemi monoalfabetici

La crittografia è un'arte antica e, contemporaneamente, una scienza molto giovane (sostituzioni monoalfabetiche fino al 1600 – 1700, poi ancora sostituzioni polialfabetiche e sistemi simmetrici in generale fino al 1975–76).

Solitamente, si indica con "crittografia moderna" tutta la crittografia sviluppata a partire dagli anni '70. La sua esplosione deriva dall'introduzione dei protocolli a chiave pubblica, che hanno portato a moltissime applicazioni nel campo della sicurezza informatica, e anche a nuovi modi di vedere/considerare lo scambio di informazioni in presenza di agenti ostili.

Lo scopo della crittografia è quello di studiare dei metodi che consentano di memorizzare, elaborare e trasmettere informazioni in presenza di agenti ostili. Lo schema di base della comunicazione tra due entità (agenti) in presenza di un'altra entità ostile è illustrato in Figura 1.

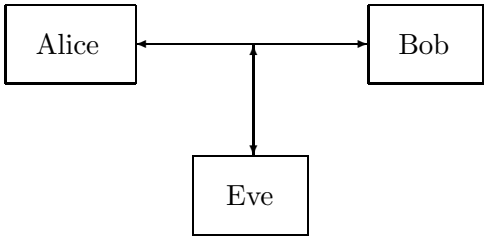


Figura 1: Schema di comunicazione in presenza di un agente ostile

Tradizionalmente, le due entità che vogliono comunicare si chiamano Alice e Bob, mentre l'entità che si mette in ascolto sul canale di comunicazione si chiama Eve (abbreviazione di "eavesdropper"). A seconda del modello di comunicazione che desideriamo studiare, si possono fare alcune ipotesi sulle capacità di Eve (ad esempio legate alla potenza computazionale a sua disposizione, alla capacità o meno di scrivere sul canale e/o di sostituire i messaggi che transitano sul canale, ecc.).

Si osservi che non è affatto detto che Alice e Bob siano i "buoni" e che Eve sia il "cattivo" (come avviene, ad esempio, nel caso in cui Alice e Bob stiano effettuando una transazione di denaro utilizzando una carta di credito, ed Eve ne approfitta per impossessarsi dei dati riguardanti la carta di credito usata): Alice e Bob potrebbero essere dei terroristi, ed Eve potrebbe essere l'FBI che cerca di scoprire dove avverrà il prossimo attentato. Allora la crittografia ha due obiettivi contrastanti:

- scoprire ed implementare crittosistemi *sicuri*, che consentano ad Alice e a Bob di comunicare tranquillamente, sapendo che anche se Eve riesce ad intercettare dei messaggi, non ci capisce nulla;

- analizzare i crittosistemi esistenti, al fine di scoprirne eventuali debolezze che possano essere usate per decifrare i messaggi cifrati con essi.

L'eterna "lotta" tra i due modi di intendere la crittografia (crittografi da una parte e crittoanalisti dall'altra) porta a migliorare i crittosistemi esistenti, a scartarne alcuni, a introdurne altri, un po' come avviene (o dovrebbe avvenire) con il software, dove alcuni programmi diventano obsoleti, alcuni errori vengono corretti, ecc.

Torniamo allo schema di comunicazione di Figura 1, e distinguiamo i seguenti casi:

- Eve non ha la possibilità né di leggere né di scrivere sul canale. In tal caso si dice che il canale è *sicuro*, e chiaramente Alice e Bob possono comunicare tranquillamente senza dover cifrare i messaggi. Osserviamo che, in realtà, canali perfettamente sicuri non esistono (se escludiamo i canali quantistici, di cui non ci occupiamo, che però hanno altri problemi); d'ora in poi, quindi, faremo l'ipotesi di avere a disposizione un canale *non sicuro*.

- Eve ha la possibilità di leggere ma non di scrivere sul canale. Pertanto, Eve può solamente intercettare i messaggi, ma non può ad esempio interagire con Bob facendo finta di essere Alice o, più semplicemente, modificare i messaggi che vanno da Alice a Bob (e/o viceversa) per trarne un qualche vantaggio. Questa è la situazione di gran lunga più comune.

- Eve ha la possibilità di scrivere ma non di leggere sul canale. Questa situazione può essere sfruttata per creare azioni di disturbo, ad esempio saturando il canale o immettendovi rumore per renderlo inutilizzabile. Alice e Bob devono quindi scegliere un altro canale, e questo è praticamente sempre possibile (ad esempio, è praticamente impossibile disturbare *tutte* le frequenze radio).

- Eve ha la possibilità sia di leggere che di scrivere sul canale. Questa è la situazione peggiore (dal punto di vista di Alice e Bob) perché consente ad Eve di sferrare i cosiddetti *attacchi dinamici* al protocollo di comunicazione usato, alterando i messaggi a proprio vantaggio e/o facendo finta di essere Alice o Bob nella comunicazione con l'altro.

Vediamo ora lo schema generale di un crittosistema. Supponiamo che Alice voglia mandare un messaggio m — che chiamiamo *testo in chiaro* — a Bob; Alice *cifra* il messaggio m utilizzando una *funzione di cifratura* $E(\cdot)$, che restituisce un nuovo messaggio c — il *testo cifrato* — e spedisce c a Bob. Questi, ricevuto il testo cifrato c, lo *decifra* applicando una *funzione di decifratura* $D(\cdot)$, la quale restituisce il testo in chiaro originale m.

Questo modo di procedere implica che Alice e Bob devono mettersi d'accordo in anticipo sulle funzioni $E(\cdot)$ e $D(\cdot)$ da usare, e che queste devono restare segrete. Ma l'esperienza insegna che questa non è una buona idea, dato che

è molto improbabile che $E(\cdot)$ e $D(\cdot)$ restino segrete. Già nel XIX secolo, A. Kerckhoffs enunciava il suo famoso principio, secondo il quale la segretezza non dovrebbe risiedere nella scelta degli algoritmi $E(\cdot)$ e $D(\cdot)$, ma piuttosto nella scelta della chiave. Definiamo allora in maniera un po' più formale il concetto di crittosistema.

Definizione 1.1 (Crittosistema). Un crittosistema è costituito dai seguenti elementi:

- uno spazio PT dei testi in chiaro, ovvero l'insieme di tutti i possibili messaggi in chiaro m;
- uno spazio K delle chiavi (di cifratura/decifratura);
- uno spazio CT dei testi cifrati, ovvero l'insieme di tutti i possibili messaggi cifrati c;
- una famiglia $E(\cdot,\cdot)$ di funzioni di cifratura:

$$E : PT \times K \to CT$$

- una famiglia $D(\cdot,\cdot)$ di funzioni di decifratura:

$$D : CT \times K \to PT$$

Ciascuna chiave k in K determina una funzione di cifratura $E(\cdot, k) : PT \to CT$ — solitamente indicata con $E_k(\cdot)$ — e una funzione di decifratura $D(\cdot, k) : CT \to PT$, solitamente indicata con $D_k(\cdot)$. Una prima richiesta che facciamo è che la decifratura della cifratura di un messaggio (effettuate con la medesima chiave) dia come risultato il messaggio stesso:

$$\forall m \, \forall k \quad D_k(E_k(m)) = m$$

Solitamente si prende $PT = \Sigma^*$, dove Σ è un opportuno alfabeto, e si pone $CT = \bigcup_{k \in K} E_k(PT)$, ovvero CT è l'unione delle immagini di PT rispetto alle funzioni $E_k(\cdot)$ (tale immagine può essere tutto Σ^* o un suo sottoinsieme). Senza perdere in generalità, si può sempre supporre che sia $\Sigma = \{0, 1\}$; per chiarezza, alcuni degli esempi che vedremo verranno svolti supponendo che Σ sia, ad esempio, l'alfabeto della lingua inglese.

Una seconda richiesta che facciamo è che lo spazio K delle chiavi non sia troppo piccolo; altrimenti, dato un testo cifrato c, Eve potrebbe calcolare i valori $D_{k_1}(c) = m_1$, $D_{k_2}(c) = m_2$, ..., $D_{k_l}(c) = m_l$, dove $K = \{k_1, k_2, \ldots, k_l\}$, e ispezionare i messaggi $m_1, m_2, \ldots, m_l$ alla ricerca di un messaggio di senso compiuto. Più precisamente, richiediamo che:

- per ogni messaggio in chiaro m e per ogni chiave k, il calcolo di $c = E_k(m)$ sia "facile" da effettuare;

- dato c, se si conosce il valore (o uno dei valori) di k tale che $c = E_k(m)$ allora il calcolo di $m = D_k(c)$ è "facile" da effettuare;
- dato c, se non si conosce il valore di k utilizzato per calcolare $c = E_k(m)$, è "difficile" trovare m.

Con i termini "facile" e "difficile" si intende, solitamente, che il calcolo può essere fatto in tempo polinomiale da una MdT deterministica ("facile"), e che non si conosce alcun algoritmo eseguibile in tempo polinomiale su una MdT deterministica che consente di svolgere il calcolo ("difficile"). Osserviamo quindi che stiamo introducendo nozioni derivanti dalla Teoria della Complessità; in effetti, quest'ultima svolge un ruolo fondamentale nella crittografia moderna. Torneremo su tali questioni quando parleremo di crittografia a chiave pubblica; per il momento, vediamo le tecniche di cifratura e decifratura a *chiave privata* (detti anche algoritmi *simmetrici*).

Il primo crittosistema a chiave privata che vediamo si chiama CAESAR, ed è effettivamente stato utilizzato da Giulio Cesare. Dal punto di vista della sicurezza lascia molto a desiderare. CAESAR è un sistema *monoalfabetico per sostituzione*: ovvero, detto $\Sigma = \{\mathtt{A}, \mathtt{B}, \mathtt{C}, \ldots, \mathtt{Z}\}$ l'alfabeto della lingua inglese, CAESAR sostituisce ogni lettera con un'altra; inoltre, ogni lettera viene *sempre* sostituita con *la stessa* lettera (da cui il termine "monoalfabetico"). Non è importante il modo in cui fissiamo gli spazi CT e PT: possiamo benissimo supporre, ad esempio, che sia $PT = CT = \Sigma$, ed estendere le funzioni $E_k(\cdot)$ e $D_k(\cdot)$ in modo che funzionino su *stringhe* di caratteri nel modo seguente:

$$\forall\, w = a_1 a_2 \cdots a_n \in \Sigma,$$
$$D_k(w) = D_k(a_1 a_2 \cdots a_n) = D_k(a_1) \circ D_k(a_2) \circ \ldots \circ D_k(a_n)$$
$$E_k(w) = E_k(a_1 a_2 \cdots a_n) = E_k(a_1) \circ E_k(a_2) \circ \ldots \circ E_k(a_n)$$

dove il simbolo $\circ$ indica la concatenazione tra stringhe (i simboli dell'alfabeto possono essere considerati delle stringhe di lunghezza 1).

Lo spazio K delle chiavi è costituito dai numeri interi compresi tra 0 e 25:

$$K = \{0, 1, 2, \ldots, 25\}$$

La funzione di cifratura $E_k : \Sigma \to \Sigma$ è definita nel modo seguente: data una lettera $\sigma \in \Sigma$, $E_k(\sigma)$ è la lettera ottenuta avanzando di k posti lungo l'alfabeto, partendo da σ e ricominciando da capo se si supera la lettera `Z`. Quindi abbiamo, ad esempio:

$$E_3(\mathtt{TRYAGAIN}) = \mathtt{WUBDJDLQ}$$
$$E_{25}(\mathtt{IBM}) = \mathtt{HAL}$$

Analogamente, la funzione di decifratura $D_k : \Sigma \to \Sigma$ è definita nel modo seguente: data una lettera $\sigma \in \Sigma$, $D_k(\sigma)$ è la lettera ottenuta retrocedendo

di k posti lungo l'alfabeto, partendo da σ e ricominciando da `Z` se si arriva alla lettera `A`.

Il motivo per cui CAESAR non è per niente sicuro è che lo spazio delle chiavi è troppo piccolo. È pertanto possibile (anche a mano!) applicare la funzione $D_k(\cdot)$ su una porzione del testo cifrato, provando tutte le possibili chiavi e arrestandosi quando si ottiene un messaggio di senso compiuto. Nonostante ciò il crittosistema CAESAR è utile da un punto di vista didattico poiché ci consente di illustrare alcune interessanti proprietà di cui gode. Una di queste è la *commutatività*: l'ordine di applicazione delle funzioni non conta (ovvero, come si dice, è *immateriale*). Ad esempio,

$$E_3 D_7 E_6 D_{11} = E_3 E_6 D_7 D_{11} = E_{17} = D_9$$

Inoltre, valgono le seguenti proprietà; per ogni k compreso tra 1 e 25:

$$\begin{aligned} D_k &= E_{26-k} \\ E_k &= D_{26-k} \\ D_k E_k &= E_0 = D_0 \end{aligned}$$

L'ultima proprietà esprime il fatto che le funzioni $E_k(\cdot)$ e $D_k(\cdot)$ si elidono a vicenda, come dovrebbero.

Un altro crittosistema a chiave segreta (privata) è dovuto a Hill, e si basa sull'algebra lineare. Numeriamo le lettere da `A` a `Z` con i numeri interi da 0 a 25; d'ora in poi, tutte le operazioni aritmetiche indicate vengono svolte modulo 26. Scegliamo un intero $d \geq 2$; per semplicità, in questo esempio scegliamo $d = 2$. Consideriamo ora una matrice quadrata M di ordine d, i cui elementi siano tutti interi compresi tra 0 e 25, e che inoltre sia invertibile nella nostra aritmetica, ovvero tale che esista M^{-1} (basta che sia $\det(M) \neq 0$) e che i suoi elementi siano tutti interi compresi tra 0 e 25. Ad esempio,

$$M = \begin{bmatrix} 3 & 3 \\ 2 & 5 \end{bmatrix} \quad \text{e} \quad M^{-1} = \begin{bmatrix} 15 & 17 \\ 20 & 9 \end{bmatrix}$$

Svolgendo le operazioni modulo 26, si verifica facilmente che $M \cdot M^{-1} = \mathbb{I}_2$. In questo crittosistema gli spazi PT e CT sono scelti entrambi uguali a Σ^d, dove Σ è l'insieme dei numeri interi da 0 a 25. La cifratura di un messaggio in chiaro m avviene nel modo seguente: m viene suddiviso in blocchi di d lettere:

$$m = P_1 P_2 \cdots P_l \quad \text{con } |P_i| = d \quad \forall\, i = 1, 2, \ldots, l$$

(se la lunghezza di m non è un multiplo di d, si possono aggiungere delle lettere fittizie in fondo al messaggio in modo da verificare questa condizione), e quindi ogni blocco viene cifrato separatamente:

$$M P_i = C_i \qquad \forall\, i = 1, 2, \ldots, l$$

Il testo cifrato risulta dalla concatenazione dei blocchi $C_1, C_2, \ldots, C_l$ ottenuti cifrando i singoli blocchi. Ad esempio, se vogliamo cifrare $m = \texttt{HELP}$ con la matrice M di cui sopra (che costituisce la chiave segreta k), otteniamo:

$$P_1 = \begin{bmatrix} \texttt{H} \\ \texttt{E} \end{bmatrix} = \begin{bmatrix} 7 \\ 4 \end{bmatrix} \quad \text{e} \quad P_2 = \begin{bmatrix} \texttt{L} \\ \texttt{P} \end{bmatrix} = \begin{bmatrix} 11 \\ 15 \end{bmatrix}$$

e quindi:

$$C_1 = M \cdot P_1 = \begin{bmatrix} 3 & 3 \\ 2 & 5 \end{bmatrix} \cdot \begin{bmatrix} 7 \\ 4 \end{bmatrix} = \begin{bmatrix} 7 \\ 8 \end{bmatrix} = \begin{bmatrix} \texttt{H} \\ \texttt{I} \end{bmatrix}$$

$$C_2 = M \cdot P_2 = \begin{bmatrix} 3 & 3 \\ 2 & 5 \end{bmatrix} \cdot \begin{bmatrix} 11 \\ 15 \end{bmatrix} = \begin{bmatrix} 0 \\ 19 \end{bmatrix} = \begin{bmatrix} \texttt{A} \\ \texttt{T} \end{bmatrix}$$

ovvero $C = C_1 C_2 = \texttt{HIAT}$.

Consideriamo ora il compito del crittoanalista. Questi può effettuare diversi tipi di attacco al crittosistema; quattro di questi, posti in ordine crescente di vantaggio, sono i seguenti:

1. **Cryptotext only**. In questo caso, il crittoanalista ha a disposizione solamente il testo cifrato, e deve risalire al testo in chiaro. Per la maggior parte dei crittosistemi, più lungo è il testo cifrato e meglio è, dato che così si possono effettuare analisi statistiche sul testo, ottenendo risultati significativi.

2. **Known plaintext**. In questo caso, il crittoanalista conosce alcune coppie $(pt, E_k(pt))$; dato un nuovo testo cifrato c, che si sa essere ottenuto come $c = E_k(m)$ (cioè, con la stessa chiave), egli deve risalire al testo in chiaro m. Si osservi che la conoscenza delle coppie $(pt, E_k(pt))$ può aiutare considerevolmente nell'analisi del testo cifrato c.

3. **Chosen plaintext**. Il crittoanalista è in grado di scegliere, anche in maniera adattiva, alcuni testi in chiaro pt, e osservare i corrispondenti testi cifrati $E_k(pt)$. Questo significa che può fare delle ipotesi sulle caratteristiche della funzione $E_k(\cdot)$ e verificarle. Gli attacchi di tipo "chosen plaintext" sono sempre possibili nei crittosistemi a chiave pubblica; in quelli a chiave privata, invece, richiedono che Eve sia in grado di mascherare la propria identità, assumendo quella di uno degli utenti legittimi del sistema. Sul libro di testo [1] viene mostrato un esempio in cui, utilizzando il crittosistema di Hill basato sull'algebra lineare, il crittoanalista ha un effettivo vantaggio nell'utilizzare un attacco di tipo chosen plaintext rispetto ad un attacco di tipo known plaintext.

4. **Chosen ciphertext**. In questo caso, il crittoanalista ha a disposizione il dispositivo decifrante (che conosce quindi il valore k corretto per applicare $D_k(\cdot)$) per un numero al più polinomiale di decifrature (rispetto ad un

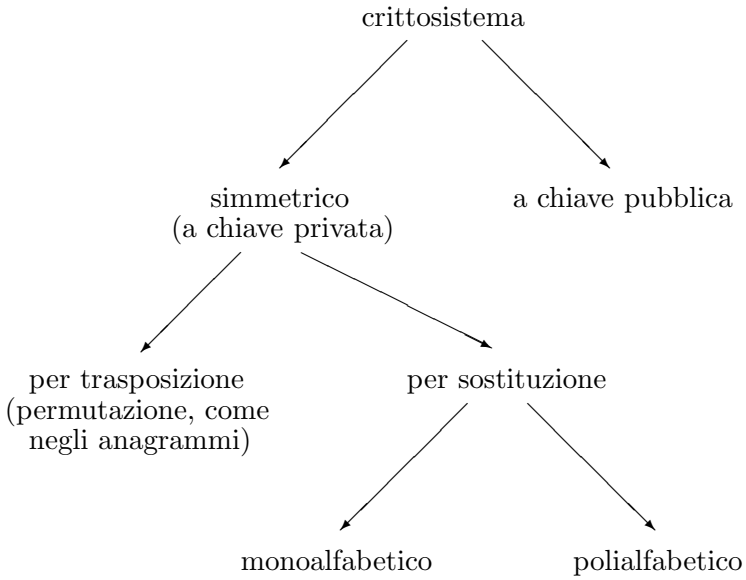


Figura 2: Una prima classificazione dei crittosistemi.

parametro n di sicurezza), su testi cifrati da lui scelti, anche in maniera adattiva. Pertanto, è in grado di fare ipotesi sul valore di k o sulle caratteristiche di $D_k(\cdot)$, e di verificarle. Dopodiché, egli riceve il testo cifrato c da decifrare, e da questo deve tentare di risalire ad m. Questo tipo di attacco risulta essere più potente soprattutto nei crittosistemi a chiave pubblica, dove le chiavi usate per cifrare e decifrare sono diverse.

In Figura 2 possiamo trovare una prima classificazione dei crittosistemi.

In un crittosistema (simmetrico) *per sostituzione*, le lettere del testo in chiaro vengono rimpiazzate con dei sostituti; in un crittosistema *monoalfabetico*, il sostituto di ciascuna lettera rimane sempre lo stesso, mentre in un crittosistema *polialfabetico* il sostituto di una data lettera può cambiare nel corso della cifratura.

È evidente che i crittosistemi per trasposizione, effettuando semplicemente una permutazione (anagramma) delle lettere che compongono il testo in chiaro, o sono molto facili da *rompere* (con questo termine indichiamo l'attività del crittoanalista, consistente nell'individuare una debolezza del sistema che consente di ricavare la chiave k, oppure di decifrare c anche senza conoscere k), oppure sono molto difficili da ricordare. È inoltre evidente che, anche grazie all'uso dei computer, questi metodi non sono molto resistenti ad una crittoanalisi accurata.

I crittosistemi per sostituzione monoalfabetica, poi, sono generalmente molto semplici da rompere: la crittoanalisi consiste semplicemente nell'analisi delle frequenze delle lettere del testo cifrato, e nel successivo confronto di tali frequenze con quelle tipiche del linguaggio in cui si suppone che il testo in chiaro sia stato

scritto. Ad esempio, nella Tabella 1 sono mostrate le frequenze con le quali compaiono le lettere nella lingua inglese.

Alte		Medie		Basse	
E	12.31%	L	4.03%	B	1.62%
T	9.59%	D	3.65%	G	1.61%
A	8.05%	C	3.20%	V	0.93%
O	7.94%	U	3.10%	K	0.52%
N	7.19%	P	2.29%	Q	0.20%
I	7.18%	F	2.28%	X	0.20%
S	6.59%	M	2.25%	J	0.10%
R	6.03%	W	2.03%	Z	0.09%
H	5.14%	Y	1.88%		

Tabella 1: Frequenze delle lettere nella lingua inglese.

Sul libro [1], alle pagine 16 e 17 ci sono altre tabelle per i più diffusi linguaggi europei.

Questa tecnica di crittoanalisi funziona perché, anche sostituendo le lettere del testo in chiaro con altri simboli, le frequenze non cambiano. Oltre alle tabelle generiche, esistono pure delle tabelle che si riferiscono all'inglese tecnico, militare, ecc. Il metodo dell'analisi delle frequenze può essere reso più complicato dal fatto che vengono scelte le parole (nel testo in chiaro) in modo da falsare le frequenze. Tuttavia, due metodi molto più semplici per raggiungere il medesimo obiettivo sono quelli di usare le cosiddette *nulle* e le cosiddette *omofone*[1]. L'uso delle nulle consiste semplicemente nell'inserire nel testo in chiaro delle lettere — scelte tra quelle poco frequenti — che non fanno parte del messaggio vero e proprio, e che hanno come unico scopo quello di appiattire le frequenze. Il decifratore legittimo del testo cifrato, una volta ottenuto il testo in chiaro, sarà naturalmente in grado di capire se una data lettera fa parte del messaggio vero e proprio oppure se è solamente una nulla. L'uso delle omofone, invece, opera come segue: supponiamo di adottare $\Sigma = \{\mathtt{A}, \mathtt{B}, \mathtt{C}, \ldots, \mathtt{Z}\}$ come alfabeto per il testo in chiaro, e $\Gamma = \{0, 1, 2, \ldots, 99\}$ come alfabeto per il testo cifrato. Dato che la lettera `E` compare circa il 12% delle volte, scegliamo a caso dodici "simboli" di Γ (gli "omofoni" di `E`) e, ogni volta che troviamo la lettera `E` nel testo in chiaro, la sostituiamo con uno di questi dodici simboli di Γ, scelto a caso con probabilità uniforme o, ancora meglio, in modo tale da rendere perfettamente piatte le frequenze del testo cifrato. Nonostante ciascuna lettera del testo in chiaro possa essere sostituita con più di un simbolo, l'uso delle omofone viene ancora considerato un sistema monoalfabetico in quanto a ciascun simbolo del

[1] La parola *omofona* si pronuncia con l'accento sulla seconda "o", cioè *omòfona*.

testo cifrato corrisponde sempre un'unica lettera dell'alfabeto Σ del testo in chiaro.

2 Crittosistemi polialfabetici

Ricordiamo che un crittosistema per sostituzione si dice *polialfabetico* se una data lettera appartenente all'alfabeto Σ del testo in chiaro può essere sostituita con più di un simbolo dell'alfabeto Γ del testo cifrato, nel corso dell'operazione di cifratura. Una classificazione più fine prevede la distinzione tra *crittosistemi polialfabetici in senso stretto e in senso lato*; per apprezzare la differenza, si consideri un crittosistema che mappa *coppie* di lettere del testo in chiaro in coppie di simboli del testo cifrato: ebbene, tale mappatura può essere monoalfabetica se consideriamo le coppie di lettere (ovvero, ogni coppia di lettere del testo in chiaro viene sempre mappata nella medesima coppia del testo cifrato), mentre perde questa caratteristica se si considerano le singole lettere (ad esempio, una `A` del testo in chiaro viene mappata in maniera differente a seconda che sia seguita da una `B` oppure da una `F`). In tal caso, si parla di crittosistema monoalfabetico in senso lato: diventa un sistema monoalfabetico in senso stretto con un opportuno cambiamento dell'alfabeto Σ del testo in chiaro. Un esempio di crittosistema monoalfabetico in senso lato è quello di Hill basato sull'algebra lineare: una data coppia di lettere del testo in chiaro, ad esempio `AL`, verrà cifrata sempre nello stesso modo, a meno che la `A` e la `L` appartengano a due blocchi diversi (stiamo supponendo che sia $d = 2$). In questo caso, il crittoanalista potrebbe tentare di decifrare un messaggio segreto facendo l'analisi delle frequenze di tutte le possibili coppie di lettere (digrammi), e confrontando poi i risultati ottenuti con le tabelle esistenti in letteratura.

Vediamo ora due crittosistemi polialfabetici. Il primo si chiama PLAYFAIR; si inizia sistemando le lettere dell'alfabeto inglese, esclusa la `J`, in un quadrato 5×5, come segue:

```
S Y D W Z
R I P U L
H C A X F
T N O G E
B K M Q V
```

La cifratura avviene nel modo seguente: il testo in chiaro viene diviso in blocchi di due lettere ciascuno. Occorre fare in modo che in nessun blocco vi siano due occorrenze della stessa lettera, e che il testo abbia lunghezza pari; se una di queste condizioni non è verificata, occorre modificare il testo in chiaro (anche solo inserendo qualche errore di ortografia facilmente individuabile). Si cifra poi ogni blocco come segue: se le due lettere del blocco non sono sulla stessa riga o colonna del quadrato, si considerano gli angoli del rettangolo individuato dalle due lettere; così, ad esempio, la coppia `AE` viene cifrata con `FO` (l'ordine in `FO` è determinato dalla condizione che `F` è sulla stessa riga di `A`, e `O` è sulla

stessa riga di E). Se le due lettere sono sulla stessa riga (rispettivamente, sulla stessa colonna), ci si sposta invece di una posizione verso destra (rispettivamente, verso il basso), in maniera ciclica. Queste sono solo convenzioni, naturalmente: nessuno ci vieta di usare altre matrici e/o altre regole di cifratura. La decifratura avviene in maniera ovvia, eseguendo al contrario le operazioni effettuate per la cifratura.

Possiamo costruire le matrici da utilizzare con il sistema PLAYFAIR utilizzando come chiavi delle frasi; ad esempio la "frase":

CORSO DI CRITTOGRAFIA

può essere utilizzata nel modo seguente: anzitutto si rimuovono le occorrenze multiple di ciascuna lettera:

C O R S D I T G A F

e poi si completa con le altre lettere dell'alfabeto, ad esempio in ordine alfabetico:

C O R S D I T G A F B E H K L M N P Q U V W X Y Z

Si osservi che abbiamo ignorato gli spazi, e abbiamo saltato la lettera J. La matrice che otteniamo è, pertanto:

```
C O R S D
I T G A F
B E H K L
M N P Q U
V W X Y Z
```

Naturalmente, una matrice siffatta si costruisce e si ricorda molto più facilmente di una matrice qualsiasi.

Per quanto riguarda la crittoanalisi del sistema PLAYFAIR, c'è un esempio molto esteso nel libro di testo [1]; in questa sede osserviamo solamente che, tra le altre cose, viene effettuata un'analisi delle frequenze dei digrammi (poiché se una data coppia di lettere del testo in chiaro forma un blocco allora viene sempre associata alla medesima coppia di lettere del testo cifrato).

Il secondo crittosistema polialfabetico che vediamo è il (falso) VIGENÈRE[2] (XVI secolo). Il VIGENÈRE è come il CAESAR, dove però la chiave varia ad ogni passo. Si costruisce un quadrato (pag. 29 del libro [1]), che viene usato sia per la cifratura che per la decifratura. Ogni colonna può essere vista come un sistema di Cesare, con la chiave k che varia da 0 a 25. Scelta una parola chiave, la si ripete tante volte quanto basta per ottenere la lunghezza del testo in chiaro; dopodiché si cifra ciascuna lettera del testo in chiaro considerando la corrispondente lettera della chiave, e guardando all'intersezione tra la riga della

[2]In francese gli accenti non andrebbero indicati sulle parole scritte in maiuscolo; qui lo mettiamo unicamente per dare un'indicazione della pronuncia.

tabella che inizia con la lettera del testo in chiaro, e la colonna che inizia con la lettera della chiave. In pratica, se la colonna che inizia con la lettera della parola chiave è la k–esima da sinistra, partendo a contare da 0, la lettera del testo in chiaro viene cifrata con il crittosistema CAESAR utilizzando come chiave il valore k; con lo stesso sistema vengono cifrate anche le lettere che distano da questa per un multiplo della lunghezza della parola chiave. Le azioni necessarie a decifrare un testo cifrato sono esattamente le opposte di quelle necessarie a cifrare. Ad esempio, la cifratura del testo in chiaro `PURPLE` con la parola chiave `CRYPTO` è:

testo in chiaro	`PURPLE`
chiave	`CRYPTO`
testo cifrato	`RLPEES`

Poiché diverse occorrenze della stessa lettera vengono cifrate nello stesso modo se capitano in corrispondenza della medesima lettera della chiave, un crittoanalista che si trovi a dover decifrare un testo cifrato senza conoscere la chiave può anzitutto fare delle ipotesi sulla lunghezza di questa. Supponiamo che la lunghezza della chiave sia 5; riscrivendo il testo cifrato su 5 colonne come segue:

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
⋮	⋮	⋮	⋮	⋮

dove i numeri indicano la posizione della lettera nel testo cifrato, osserviamo che i simboli che compaiono nella prima colonna sono stati ottenuti utilizzando la prima lettera della chiave, e così via per le altre colonne. Ovvero, ogni colonna è stata ottenuta attraverso un particolare crittosistema di Cesare. Anche se il numero di chiavi in un sistema di Cesare è molto basso, il numero di combinazioni che si possono fare con il VIGENÈRE è molto elevato, soprattutto se la chiave considerata è molto lunga; già nel nostro caso abbiamo $26^5 = 11.881.376$ combinazioni possibili. Tuttavia, è sempre possibile fare l'analisi delle frequenze per ciascuna colonna, dato che CAESAR è un crittosistema monoalfabetico; se il testo non è di tipo molto particolare, solitamente questo approccio consente di individuare con facilità la chiave, e quindi di decifrare il testo cifrato. L'unico problema, quindi, sembra essere quello di individuare la lunghezza della chiave. Ma questo può essere fatto molto spesso in maniera efficace utilizzando il *metodo di Kasiski* (1860 circa); tale metodo prevede di cercare occorrenze della stessa parola nel testo cifrato. Supponiamo che la parola `PUXUL` appaia due volte, separata da 15 lettere:

... `PUXUL` 15 lettere `PUXUL` ...

Questo potrebbe essere un fatto puramente accidentale, ma potrebbe anche essere dovuto al fatto che la stessa porzione di testo in chiaro è stata cifrata con la

stessa porzione di chiave (o meglio, partendo dalla stessa posizione all'interno della chiave). Se ciò fosse vero, la distanza tra le due `P`, ovvero 20, sarebbe un multiplo della lunghezza della chiave; allora la lunghezza della chiave potrebbe essere solamente (1), 2, 4, 5, 10 o 20. Avendo parecchie congetture sulla lunghezza della chiave, molto spesso si riesce ad individuare la lunghezza effettiva prendendo le lunghezze comuni dalle congetture. Chiaramente, più lunghe sono le parole che si ripetono, meglio è; anche la situazione in cui la stessa parola si ripete più di due volte risulta essere molto spesso vantaggiosa. Si osservi che, fino alla scoperta del metodo di Kasiski, il VIGENÈRE era da tutti considerato un crittosistema molto sicuro.

I crittosistemi che abbiamo visto finora erano tutti per sostituzione; come abbiamo già osservato, i metodi per trasposizione sono in generale molto difficili da ricordare quando non sono banali. Naturalmente nessuno vieta di applicare entrambi i metodi, effettuando quelle che sono note come *sovracifrature*: ad esempio, prima si effettua una trasposizione (permutazione) delle lettere del testo in chiaro, e poi si applica una cifratura per sostituzione sul testo così ottenuto. Questi metodi di cifratura misti sono particolarmente efficaci, in quanto elevano a dismisura il numero di possibili chiavi e sono particolarmente ostici da crittoanalizzare, come ben sanno gli inglesi che hanno lavorato a Bletchley Park durante la II Guerra Mondiale.

3 Rotori e DES

I crittosistemi visti finora possono essere resi più complicati, e allo stesso tempo più sicuri, grazie all'uso di *macchine crittografiche*, che rendono le operazioni di cifratura e decifratura molto più veloci, e consentono di lavorare con sistemi aventi un numero enorme di chiavi (ad esempio, nell'ordine di 10^{100}). Tali macchine fanno uso molto spesso di *rotori*, ciascuno dei quali è suddiviso in un certo numero di caselline, ciascuna contenente una lettera della chiave. Le macchine cifranti sono fatte in modo tale che una lettera del testo in chiaro viene cifrata tenendo conto delle lettere indicate su *tutti* i rotori. Dopo aver cifrato una lettera, uno dei rotori avanza di una posizione; quando il rotore ha fatto un giro completo, una linguetta fa in modo che il rotore successivo avanzi anch'esso di una posizione, e così via. La lunghezza effettiva della chiave è allora data dal prodotto tra i numeri di caselline contenute in ciascun rotore. Chiaramente, prima di cifrare un testo in chiaro occorrerà posizionare i rotori in una configurazione iniziale; le macchine cifranti sono solitamente costruite in modo tale che, posizionati i rotori nella medesima posizione, e inserito il testo cifrato, queste restituiscono il testo in chiaro.

Una macchina a rotori implementa una sostituzione polialfabetica. Detto N il numero delle possibili configurazioni dei rotori, è evidente che due occorrenze del medesimo carattere che compaiono nel testo in chiaro ad una distanza di N caratteri verranno cifrate nello stesso modo; tuttavia, questo non accade mai in

pratica, dato che N può essere *molto* grande (ad esempio, dell'ordine di 26^{10}). Ne consegue che anche il metodo di Kasiski è destinato al fallimento.

Non ci dilunghiamo oltre sulle macchine a rotori; per chi fosse interessato, sul libro di testo [1] c'è una buona descrizione della macchina C–36. Parliamo piuttosto dell'algoritmo di cifratura simmetrica più usato dagli anni '70 agli anni '90: il DES (Data Encryption Standard). All'inizio degli anni '70, la situazione riguardante gli algoritmi crittografici usati era molto caotica: ognuno usava un algoritmo diverso da quelli usati dagli altri per cifrare e decifrare. Ciò creava non poche difficoltà quando due sistemi (tipicamente due computer appartenenti a banche diverse, dato che allora si cominciavano ad utilizzare in maniera estesa mainframe e minicomputer nell'industria e nel commercio) dovevano dialogare tra loro. In poche parole, mancava uno *standard* per gli algoritmi di cifratura. Il DES è stato il primo di tali standard (US FIPS 46); pubblicato nel 1977, è abbastanza complicato da richiedere l'impiego di un dispositivo di calcolo per cifrare e per decifrare.

Cifratura e decifratura avvengono come segue: si sceglie a caso una chiave di 56 bit; questa serve sia per cifrare che per decifrare, e deve restare naturalmente segreta. Quindi si aggiungono otto bit alla chiave, nelle posizioni 8, 16, 24, 32, ..., 64, in modo tale che ciascun byte abbia parità dispari; ciò è utile per individuare errori di trasmissione durante la distribuzione e la memorizzazione delle chiavi. Si ottengono così 64 bit, di cui i 56 bit della chiave vera e propria sono nelle posizioni:

$$1, 2, \ldots, 7, 9, 10, \ldots, 63$$

Questi 56 bit vengono sottoposti alla seguente permutazione:

57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

La permutazione determina due blocchi C_0 e D_0 di 28 bit ciascuno (C_0 è costituito dai primi 28 bit, e D_0 dai successivi 28 bit). Avendo costruito i blocchi C_{n-1} e D_{n-1}, per $n = 1, 2, \ldots, 16$, costruiamo i blocchi C_n e D_n applicando *uno o due shift* (o meglio, *rotazioni*) *verso sinistra* a C_{n-1} e D_{n-1}, secondo la seguente tabella:

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Numero di shift a sinistra	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Ad esempio, i blocchi C_6 e D_6 vengono ottenuti applicando due shift verso sinistra ai blocchi C_5 e D_5. Lo shift a sinistra va inteso come una *rotazione* dei 28 bit che compongono ciascun blocco: dopo l'operazione, i bit ottenuti sono quelli che precedentemente occupavano le posizioni 2, 3, ..., 28, 1.

Applichiamo poi 16 *selezioni permutate* K_n, con $1 \leq n \leq 16$, dei bit che costituiscono la chiave. Ciascun K_n consiste di 48 bit, ottenuti dai bit di C_nD_n nell'ordine seguente:

14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

Osserviamo che in K_n mancano 8 dei 56 bit di C_nD_n. Dall'applicazione di queste selezioni permutate otteniamo i sedici blocchi K_1, K_2, ..., K_{16}, di 48 bit ciascuno. Con questo si conclude la fase *preliminare* del DES; viene ora la cifratura vera e propria. Anzitutto, suddividiamo il testo in chiaro in blocchi di 64 bit ciascuno; chiamiamo w uno di tali blocchi. Il blocco w viene anzitutto sottoposto alla seguente *permutazione iniziale*:

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

Chiamiamo w' il blocco così ottenuto, e scriviamo $w' = L_0R_0$, dove L_0 ed R_0 consistono di 32 bit ciascuno. Avendo definito L_{n-1} ed R_{n-1}, per $1 \leq n \leq 16$, definiamo L_n ed R_n come:

$$\begin{aligned} L_n &= R_{n-1} \\ R_n &= L_{n-1} \oplus f(R_{n-1}, K_n) \end{aligned} \tag{1}$$

dove $\oplus$ denota lo XOR bit a bit, ed f è definita più avanti. La cifratura c del blocco w di testo in chiaro viene ottenuta applicando l'*inversa della permutazione iniziale* al blocco di 64 bit $R_{16}L_{16}$.

Prima di definire la funzione f, vediamo come funziona la decifratura. Dato un blocco c di 64 bit del testo cifrato, applichiamo anzitutto la permutazione

iniziale, ottenendo così il blocco $R_{16}L_{16}$. Poiché le equazioni (1) possono essere scritte come segue:

$$\begin{aligned} R_{n-1} &= L_n \\ L_{n-1} &= R_n \oplus f(L_n, K_n) \end{aligned}$$

a partire da L_{16} ed R_{16} possiamo ricavarci L_{15}, R_{15}, L_{14}, R_{14}, ..., L_0, R_0. Detto w' il blocco di 64 bit ottenuto accostando L_0 ed R_0, applicando l'inversa della permutazione iniziale otteniamo il blocco w del testo in chiaro corrispondente al blocco c del testo cifrato.

La funzione f prende in ingresso il blocco L_n (o R_{n-1}, che è lo stesso) di 32 bit e il blocco K_n di 48 bit, e produce un blocco di 32 bit come segue. Il blocco L_n di 32 bit viene espanso in un blocco di 48 bit, sistemando i bit come segue:

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

Dopo questa espansione, i due blocchi di 48 bit (quello appena ottenuto e K_n) vengono sommati modulo 2 (XOR) bit a bit. Il blocco B risultante viene diviso in 8 blocchi di 6 bit ciascuno:

$$B = B_1\, B_2 \cdots B_8$$

Ciascuno di questi 8 blocchi B_i viene ora trasformato in un blocco B_i' di 4 bit utilizzando la tabella S_i (S–box) appropriata, tra quelle qui di seguito elencate:

S_1

14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

S_2

15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

S_3

10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

S_4

7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

S_5

2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

S_6

12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

S_7

4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

S_8

13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

La trasformazione avviene come segue. Si supponga, ad esempio, che sia $B_7 = \texttt{110010}$. Il primo e l'ultimo bit (`1` e `0`) rappresentano un numero intero x, con $0 \leq x \leq 3$. Analogamente, i 4 bit centrali (`1001`) rappresentano un numero intero y, con $0 \leq y \leq 15$. Nel nostro esempio, $x = 2$ e $y = 9$. Indicizzando le righe e le colonne di S_7 con i numeri x e y, abbiamo che la coppia (x, y) determina un unico numero; nel nostro caso, tale numero è 15. Considerandone la rappresentazione binaria, abbiamo $B'_7 = \texttt{1111}$. Detto B' il blocco di 32 bit

ottenuto accostando B'_1, B'_2, ..., B'_8, il valore di f è ottenuto applicando a B' la seguente permutazione:

16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25

Questo completa la descrizione del DES.

Il DES è un algoritmo molto veloce, utilizzando un hardware appropriato: le operazioni richieste sono solamente XOR, permutazioni e rotazioni di gruppi di bit, e indicizzazione di matrici prestabilite. D'altra parte, la crittanalisi porta a numerosi sistemi di equazioni non lineari e a problemi NP–completi. Nonostante ciò, il DES è praticamente stato abbandonato in quanto non viene più considerato un metodo sicuro (nel senso di cifratura forte): il fatto che utilizzi chiavi di "soli" 56 bit ha fatto sì che fosse possibile costruire una macchina dedicata (la DES Cracking Machine), massicciamente parallela, che esplora tutto lo spazio delle 2^{56} possibili chiavi in maniera esaustiva, alla velocità di una chiave ogni microsecondo. Il prezzo stimato di tale macchina varia considerevolmente, a seconda del numero di processori utilizzati (e anche perché ne è stato costruito un unico esemplare); una stima fatta a metà degli anni '90 parla di circa 250.000 dollari.

Sempre a metà degli anni '90 è stata avviata una gara internazionale per scegliere il successore di DES come standard per la cifratura simmetrica. Nel frattempo, appurata l'insicurezza del DES, si è pensato di adottare il seguente stratagemma: anziché utilizzare semplicemente il DES con una chiave di 56 bit si scelgono due chiavi k_1 e k_2 di 56 bit ciascuna. Per cifrare un blocco w di 64 bit del testo in chiaro si procede come segue: si cifra w con la chiave k_1, ottenendo un blocco cifrato c_1; dopodiché si *decifra* c_1 utilizzando la chiave k_2, e infine si cifra con la chiave k_1 il blocco c_2 così ottenuto. Questo procedimento, consistente nell'applicare tre volte il DES, è noto come *Triple–DES* (solitamente abbreviato con 3DES). Come si può notare, il 3DES risulta in pratica essere un algoritmo di cifratura con chiavi da 112 bit.

Tra i diversi algoritmi proposti come successore del DES è stato scelto Rijndael[3], scritto da due ricercatori belgi, J. Daemen e V. Rijmen. Tale crittosistema è divenuto standard (con il nome di Advanced Encryption Standard, AES) nel novembre del 2001 (US FIPS 197). Rijndael è un crittosistema che trasforma blocchi da 128 bit del testo in chiaro in blocchi cifrati da 128 bit. Le lunghezze possibili per le chiavi previste nello standard sono tre: 128, 192 e 256

[3]Gli autori dell'algoritmo precisano che la pronuncia corretta è molto simile a quella di "rain doll" in inglese.

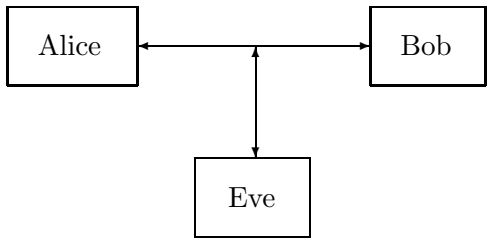
bit; tuttavia, l'algoritmo è in grado di funzionare con chiavi di altre lunghezze, come anche su blocchi di lunghezza maggiore.

A differenza del DES, che nel corso dell'esecuzione agisce su singoli bit della chiave e del testo in chiaro, Rijndael opera sempre su interi byte. Ciò facilita enormemente le implementazioni software; per quanto riguarda invece le implementazioni hardware, sono possibili diversi compromessi tra la dimensione del circuito e il tempo impiegato per cifrare e decifrare: qui menzioniamo solo il fatto che avendo a disposizione 4 KByte di ROM è possibile implementare l'algoritmo come la consultazione di 4 tabelle prefissate da 1 KByte ciascuna e con 4 XOR (da effettuarsi tra i 4 valori ottenuti dalle tabelle e la chiave). Tutte queste caratteristiche fanno si che — a meno che non venga scoperta qualche vulnerabilità — il nuovo AES risulti l'algoritmo di cifratura simmetrica più conveniente da usare per molti anni a venire.

4 Sicurezza alla Shannon e One Time Pad

La rottura del DES porta a porci la seguente domanda: esiste un crittosistema che sia assolutamente e dimostrabilmente sicuro? Questa domanda, a sua volta, porta alla seguente: cosa intendiamo quando diciamo che un dato sistema è *sicuro*?

Torniamo allo schema di comunicazione tra Alice e Bob:



Detta X una variabile casuale a valori in PT (lo spazio dei testi in chiaro) e Y una variabile casuale a valori in CT (lo spazio dei testi cifrati), abbiamo che Alice sceglie un valore $x \in PT$, lo cifra — ottenendo un valore $y \in CT$ — e spedisce y a Bob. Per semplicità, supponiamo che x venga scelto in maniera uniforme su PT. Se Eve non è in ascolto sul canale, tutto ciò che può dire riguardo x è che è stato scelto in maniera casuale; in altre parole, l'unica cosa che Eve conosce è la $\Pr[X = x]$. D'altra parte, se Eve ascolta il messaggio y che transita sul canale conosce non solo $\Pr[X = x]$, ma anche la probabilità condizionata $\Pr[X = x \mid Y = y]$; questo significa che Eve, osservando y, anche se non viene a conoscenza del valore effettivo di x acquisisce tuttavia un'informazione che prima non aveva. Possiamo allora definire un crittosistema sicuro come un crittosistema nel quale Eve, avendo osservato y, non acquisisce alcuna informazione in più su x.

Definizione 4.1. [Sicurezza alla Shannon] Il crittosistema è *sicuro* se e solo se:

$$\forall\, x \in X,\ \forall\, y \in Y \quad \Pr[X = x] = \Pr[X = x \mid Y = y]$$

Dal punto di vista della Teoria dell'Informazione, questo significa che:

$$H(X) = H(X \mid Y)$$

In altre parole, le variabili aleatorie X e Y sono *indipendenti*.

Questa definizione è stata data da Shannon, e fa parte dei suoi lavori svolti nel corso della II Guerra Mondiale. La definizione (insieme ad altri risultati) è stata pubblicata nel 1949, in seguito ad una declassificazione del lavoro avvenuta probabilmente per sbaglio.

Ora ci chiediamo: come fanno le variabili X ed Y ad essere indipendenti, dato che i valori di Y sono cifrature dei messaggi in X? Detto queto, esistono dei crittosistemi che sono sicuri nel senso dato da Shannon? Ebbene, la risposta è affermativa (!): è possibile dimostrare che l'*unico* crittosistema sicuro — a meno di isomorfismi — è il cosiddetto One Time Pad, noto anche come *Cifrario di Vernam*.

Le idee alla base del crittosistema One Time Pad sono le seguenti:

- la chiave deve essere lunga almeno quanto il messaggio che deve essere cifrato;
- la chiave deve essere utilizzata una sola volta.

Se anche uno solo di questi requisiti non viene soddisfatto, si dimostra che il crittosistema non è più sicuro.

Le funzioni di cifratura e di decifratura del One Time Pad sono definite come segue: detta k la chiave, x un messaggio in chiaro da cifrare e y un messaggio da decifrare, abbiamo:

- $E(x, k) = x \oplus k$
- $D(y, k) = y \oplus k$

Supponendo che x, y e k siano stringhe in $\{0,1\}^n$, abbiamo quindi che la stessa funzione (lo XOR bit a bit) viene utilizzata sia per cifrare che per decifrare. Osserviamo che vale:

$$\forall\, x, k \in \{0,1\}^n \qquad D\big(E(x,k), k\big) = E(x,k) \oplus k = x \oplus k \oplus k = x$$

ovvero, cifrando e successivamente decifrando un dato messaggio in chiaro x, usando lo stesso valore di k per entrambe le operazioni, riotteniamo il messaggio di partenza.

Diamo lo schema della dimostrazione della sicurezza (nel senso dato da Shannon) del crittosistema One Time Pad. Ricordando che $|x| = |y| = |k|$, abbiamo che:

$$\forall\, y \in Y,\ \forall\, x \in X \quad \exists\,!\, k \text{ tale che } E(x,k) = y$$

Quindi, dati y e x, il valore di k resta univocamente determinato; d'altra parte, dato il solo valore y, per ogni possibile valore di k esiste un x tale che $E(x,k) = y$. Questo significa che anche se conosciamo y non possiamo dedurre nulla riguardo x, poiché y potrebbe venire da un *qualunque* valore di x, scegliendo opportunamente il valore di k.

Se $|k| < |x| = |y|$, invece, le cose stanno diversamente. Supponiamo che sia $|k| = |x| - 1 = n - 1$. Allora, poiché i messaggi in chiaro vengono scelti dal mittente con probabilità uniforme, abbiamo:

$$\Pr[X = x] = \frac{1}{2^n}$$

mentre, osservato y, i valori di k utilizzabili per ricavare x sono al più 2^{n-1}, e quindi:

$$\Pr[X = x \mid Y = y] = \frac{1}{2^{n-1}} \neq \Pr[X = x]$$

In alternativa, possiamo associare (ad esempio) a ciascun k il valore di x corrispondente la cui sequenza di bit inizia con 0 (dato che il crittosistema funziona solo per $|k| = |x|$)[4]; per la metà dei possibili valori di x abbiamo:

$$\Pr[X = x \mid Y = y] = 0$$

È evidente che le due condizioni di cui sopra rendono molto difficoltoso l'utilizzo del crittosistema proposto. In particolare, il problema della distribuzione e della gestione delle chiavi non è affatto banale: le chiavi devono essere distribuite periodicamente da parte di una persona fidata (cioè, in pratica, attraverso un canale sicuro), e inoltre il loro utilizzo deve essere sincronizzato tra Alice e Bob. Per quanto riguarda il problema della distribuzione delle chiavi, vedremo che la crittografia a chiave pubblica risolve il problema in maniera molto elegante. Resta tuttavia il problema della *generazione* delle chiavi: l'ideale sarebbe che le chiavi vengano scelte in maniera completamente casuale; ciò, però, è in generale un compito lungo e non facile: occorre affidarsi a una qualche sorgente naturale di casualità (ad esempio, l'emissione di particelle in un materiale radioattivo, oppure le estrazioni del Lotto) ed effettuare la misurazione (rilevazione) di un qualche evento ad essa associato (ad esempio, il numero di particelle emesso in un'unità di tempo, oppure la sequenza dei numeri estratti su una certa ruota). Per motivi di semplicità e di efficienza, ci piacerebbe che le chiavi fossero generate da particolari algoritmi deterministici, che chiamiamo *generatori pseudocasuali*.

[4] In quanto precede è stata implicitamente utilizzata la possibilità di replicare la chiave. In particolare, abbiamo supposto che si possa — ad esempio — cifrare il primo e l'ultimo bit di x con il primo bit di k.

5 Generatori pseudocasuali

L'idea intuitiva che sta dietro ad un generatore pseudocasuale è che, a partire da una stringa "veramente" casuale x, detta *seme*, esso produce una stringa che è *più lunga* di x, e che è *indistinguibile* da una stringa della stessa lunghezza scelta a caso. Con il termine *indistinguibile* intendiamo dire che nessun algoritmo eseguibile in tempo polinomiale su una Macchina di Turing (MdT) *probabilistica* è in grado di decidere se la stringa emessa dal generatore è veramente casuale oppure no.

La definizione di pseudocasualità che stiamo dando è quindi di tipo *computazionale*: siamo disposti a sostituire i numeri (stringhe di bit) casuali — difficili e costosi da ottenere — con numeri generati da un algoritmo *deterministico* in maniera tale che nessun algoritmo *probabilistico* riesca ad accorgersi della sostituzione. Poiché è ragionevole supporre che l'avversario, Eve, disponga di capacità computazionali che sono al più quelle di una MdT probabilistica (dato che Eve non è un essere soprannaturale, e le MdT probabilistiche sono tradizionalmente usate per separare i problemi computazionalmente trattabili da quelli intrattabili), stiamo automaticamente supponendo che neanche Eve sia in grado di capire se stiamo usando numeri casuali o pseudocasuali.

Inoltre, la definizione che stiamo per dare solleva un'interessante quesito: cosa significa che un evento è casuale? Supponiamo di mettere alcune palline bianche e nere in un'urna, di mescolarle e di estrarre una pallina. Ci hanno sempre insegnato che il risultato di questo esperimento è casuale; ma supponiamo di essere in grado, osservando il modo in cui le palline vengono immesse nell'urna, di calcolare per ogni istante l'esatta posizione di ciascuna pallina, e quindi di prevedere con assoluta certezza quale sarà la pallina estratta: è evidente che in questo caso il risultato dell'esperimento non sarà più casuale. Ma sappiamo benissimo che — a parte la difficoltà nel misurare l'esatta traiettoria delle palline mentre vengono immesse nell'urna — è praticamente impossibile calcolare per ogni istante la posizione di ciascuna pallina nell'urna mentre questa viene rimescolata: i parametri da considerare sono semplicemente troppi, e inoltre i sistemi di equazioni differenziali che ne risultano non sono in generale risolvibili analiticamente. Il tutto si riduce allora alla nostra incapacità nel calcolare (approssimazioni numeriche delle) soluzioni di tali sistemi: sembrerebbe pertanto che la casualità derivi dalla nostra limitata capacità computazionale[5].

Fatte queste considerazioni, diamo la definizione formale di generatore pseudocasuale.

Definizione 5.1. Un *generatore pseudocasuale* è un algoritmo G eseguibile in tempo polinomiale da una MdT deterministica che, data in ingresso una stringa

[5] In realtà, considerando sistemi di dimensione microscopica, per i quali le leggi della meccanica classica non sono più applicabili, per descrivere il sistema è necessario passare alla Meccanica Quantistica, nella quale la casualità sembra essere ineliminabile.

x (il *seme*) con $|x| = k$, emette in uscita una stringa y di lunghezza $\ell(k) > k$, e che inoltre gode della seguente proprietà: per ogni algoritmo D eseguibile in tempo polinomiale da una MdT probabilistica, per ogni polinomio $p(\cdot)$ e per tutti gli interi k sufficientemente grandi, vale:

$$\left| \Pr\left[x \leftarrow \{0,1\}^k; r \leftarrow G(x) : D(r) = 1\right] - \Pr\left[r \leftarrow \{0,1\}^{\ell(k)} : D(r) = 1\right] \right| < \frac{1}{p(k)}$$

■

L'algoritmo D, detto "distinguisher", emette un 1 in output se la stringa r ricevuta in input è — secondo lui — veramente casuale, ed emette uno 0 in caso contrario. La definizione di cui sopra può essere parafrasata dicendo che la probabilità che l'algoritmo D non sia in grado di distinguere l'output di G da una stringa r veramente casuale tende a zero (al crescere di k) più velocemente dell'inverso di un qualsiasi polinomio $p(\cdot)$ prefissato. Si osservi che la distanza tra le due distribuzioni di probabilità menzionate nella formula corrisponde alla "capacità di inganno" di G.

I generatori pseudocasuali possono essere sia *uniformi* che *non uniformi*; specificare meglio cosa questo significhi esula dagli obiettivi del corso: basti sapere che un *generatore pseudocasuale non uniforme* è definito come sopra, sostituendo al posto di "per ogni algoritmo D eseguibile in tempo polinomiale da una MdT probabilistica" la frase "per ogni algoritmo D eseguibile in tempo polinomiale da una MdT probabilistica *non uniforme*". In altre parole, un generatore pseudocasale non uniforme è in grado di imbrogliare perfino una MdT non uniforme, che è in generale più potente di una MdT uniforme.

Esempio 5.1. Dati tre numeri interi a, b ed m, con $0 \leq a, b < m$, e un seme (anch'esso intero) s, con $0 \leq s < m$, il seguente sistema di equazioni:

$$\begin{aligned} x_0 &= s \\ x_i &= ax_{i-1} + b \mod m \end{aligned}$$

si chiama *generatore congruenziale lineare*. Osserviamo che esso *non è* un generatore pseudocasuale secondo la Definizione 5.1, dato che produce una sequenza di numeri (stringhe di bit) tutti della medesima lunghezza. Per questo e per altri motivi, il generatore congruenziale lineare non viene praticamente mai utilizzato in crittografia. ■

Abbiamo visto che un generatore pseudocasuale è un algoritmo che, in qualche modo, "allunga" la stringa di ingresso di k bit, e restituisce una stringa di $\ell(k) > k$ bit che ha la proprietà di essere indistinguibile (almeno, avendo a disposizione al più una MdT probabilistica) da una stringa di $\ell(k)$ bit veramente casuale. La prima domanda che ci poniamo è la seguente: "dobbiamo costruire tanti generatori pseudocasuali, uno diverso dall'altro, per ogni possibile funzione

$\ell(\cdot)$ che riusciamo a concepire?". Fortunatamente la risposta è negativa: a partire da un unico generatore pseudocasuale $H : \{0,1\}^k \to \{0,1\}^{k+1}$, che allunga l'input di 1 bit, possiamo costruire qualsiasi generatore pseudocasuale $G : \{0,1\}^k \to \{0,1\}^{\ell(k)}$ per qualsiasi funzione $\ell : \mathbb{N} \to \mathbb{N}$ tale che $\ell(k) > k$:

$$\begin{aligned}
&\text{Generatore } G(x_0)\\
&\qquad x_1\sigma_1 \leftarrow H(x_0)\\
&\qquad x_2\sigma_2 \leftarrow H(x_1)\\
&\qquad\quad \vdots\\
&\qquad x_{\ell(k)}\sigma_{\ell(k)} \leftarrow H(x_{\ell(k)-1})\\
&\text{Output } (\sigma_1, \sigma_2, \ldots, \sigma_{\ell(k)})
\end{aligned}$$

dove $x_i \in \{0,1\}^k$ e $\sigma_i \in \{0,1\}$ per ogni $i = 1, 2, \ldots, \ell(k)$.

In altre parole, data la stringa $x_0 \in \{0,1\}^k$, per costruire una stringa pseudocasuale di lunghezza $\ell(k) > k$ è sufficiente applicare $\ell(k)$ volte il generatore pseudocasuale H, ottenendo ogni volta un bit della stringa di output e un nuovo seme per applicare H. Ci resta quindi "solamente" da mostrare come sia possibile costruire un generatore pseudocasuale che allunga di 1 bit la stringa data in input.

Prima di fare ciò, però, illustriamo un notevole risultato di A. C. Yao del 1985. Il problema principale nel dimostrare che un certo generatore è pseudocasuale è il fatto che dobbiamo dimostrare che è in grado di resistere a tutti i possibili test. Ciò, naturalmente, è impossibile, dato che i test sono infiniti! (Ad ogni "distinguisher" D corrisponde un test). Fortunatamente, Yao ha dimostrato che un generatore è pseudocasuale se e solo se è in grado di resistere ad un unico test particolare, detto il *test universale*, che riguarda l'impredicibilità della stringa generata, e che è definito come segue.

Definizione 5.2. Sia $G : \{0,1\}^k \to \{0,1\}^{\ell(k)}$ un generatore. G è *impredicibile* se e solo se per ogni $i = 1, 2, \ldots, \ell(k)$, per ogni polinomio $p(\cdot)$ e per ogni algoritmo A eseguibile in tempo polinomiale da una MdT probabilistica, esiste un intero $k_{A,p}$ (che dipende cioè da A e da p) tale che per tutti i $k > k_{A,p}$:

$$\begin{aligned}
\Pr\Big[x \leftarrow \{0,1\}^k;\ \ (\sigma_1 \cdots \sigma_{\ell(k)}) = G(x);\ \ \hat{\sigma}_i \leftarrow A(\sigma_1 \cdots \sigma_{i-1}) :\\
\sigma_i = \hat{\sigma}_i\Big] \leq \frac{1}{2} + \frac{1}{p(k)}
\end{aligned}$$

■

Parlando in maniera informale, l'impredicibilità della sequenza $(\sigma_1 \cdots \sigma_{\ell(k)})$ emessa dal generatore consiste nell'intrattabilità computazionale del compito di indovinare correttamente l'i–esimo bit della sequenza a partire dai primi $i-1$ (o almeno, di indovinarlo in maniera significativamente migliore di quanto si possa fare lanciando una moneta non truccata).

Come dicevamo, Yao ha dimostrato il seguente teorema.

Teorema 5.1 (Yao). *Un generatore G è pseudocasuale se e solo se è impredicibile.* ∎

Torniamo ora al problema di costruire un generatore pseudocasuale che allunga di 1 la lunghezza della stringa data in ingresso. Gli strumenti che ci servono per fare ciò sono le cosiddette *funzioni one–way* e i *predicati hard–core*. Intuitivamente, una funzione $f : \mathbb{N} \to \mathbb{N}$ viene detta one–way se, dato $x \in \mathbb{N}$, il problema consistente nel calcolare $f(x)$ è *computazionalmente trattabile* mentre, dato $f(x)$, il problema che consiste nel calcolare un valore $x' \in \mathbb{N}$ tale che $f(x) = f(x')$ è *computazionalmente intrattabile*. Più formalmente, possiamo definire le funzioni one–way come segue.

Definizione 5.3 (Funzione one–way). Una funzione $f : \mathbb{N} \to \mathbb{N}$ è *one–way* se e solo se:

- f è calcolabile in tempo polinomiale (rispetto alla lunghezza dell'input) da una MdT deterministica;
- esistono due polinomi $p(\cdot)$ e $q(\cdot)$ tali che:

$$p(|x|) \leq |f(x)| \leq q(|x|);$$

- per ogni algoritmo A che può essere eseguito in tempo polinomiale da una MdT probabilistica, e per ogni polinomio $p(\cdot)$ esiste un numero naturale $n_{A,p}$ (che dipende cioè da A e da p) tale che:

$$\forall\, n \geq n_{A,p} \quad \Pr\left[x \leftarrow \{0,1\}^n;\ \ x' \leftarrow A\big(f(x)\big) :\ \ f(x) = f(x')\right] \leq \frac{1}{p(n)}$$

∎

Osserviamo che l'ultima disuguaglianza afferma che la probabilità che un qualsiasi algoritmo probabilistico A riesca ad invertire la funzione f tende a zero più velocemente di qualsiasi polinomio fissato. Insieme al fatto — espresso dal primo punto della definizione — che la funzione f è "facile" da calcolare, abbiamo la formalizzazione dell'idea intuitiva di funzione one–way. Il secondo punto della definizione è più delicato: mentre la disuguaglianza $|f(x)| \leq q(|x|)$ è ridondante, dato che se A è eseguibile in tempo polinomiale allora non può produrre un output di lunghezza superpolinomiale, la richiesta che sia $p(|x|) \leq |f(x)|$ è invece necessaria, dato che se l'output della funzione f fosse "troppo corto" nessun algoritmo A potrebbe calcolare una preimmagine in tempo polinomiale. Ad esempio, se consideriamo la funzione $f : x \to \log x$, un algoritmo che dovesse calcolare una preimmagine di $f(x) = \log x$ richiederebbe un numero esponenziale di passi in $|f(x)|$.

Osserviamo anche che se $P = NP$ allora non possono esistere funzioni one–way; la contronominale di questa affermazione, che è quindi anch'essa vera, è

che se esiste almeno una funzione one–way allora $P \neq NP$. Sull'affermazione contraria, invece, non possiamo dire nulla: il fatto che sia $P \neq NP$ non ci consente di dire niente riguardo l'esistenza di funzioni one–way: infatti, un certo problema in NP potrebbe essere difficile da risolvere per alcune istanze (il che è sufficiente per fare in modo che sia $P \neq NP$), ma potrebbe nondimeno essere facile da risolvere *in media*, ovvero *per istanze scelte a caso* (il che renderebbe falsa la disuguaglianza nella definizione di funzione one–way).

D'ora in poi, per semplicità, considereremo solamente *permutazioni one–way*, ovvero funzioni one–way biunivoche da un insieme (tipicamente, $\{0,1\}^n$) in se stesso.

Definiamo le nozioni di *hard–core bit* e di *hard–core predicate*. I due concetti sono strettamente correlati: in effetti, un hard–core bit può essere visto come il valore prodotto da un hard–core predicate, ovvero da una funzione che mappa gli elementi di $\{0,1\}^n$ in $\{0,1\}$. L'idea è che, dato $f(x)$, se f è una permutazione one–way allora deve esistere qualche bit di x che è difficile da calcolare. Chiaramente, la situazione in cui vi è un solo bit di x che è difficile da calcolare non è molto significativa, dato che si può risalire al valore del bit per tentativi; pertanto, richiediamo che una parte sostanziale dei bit che compongono x (ad esempio, i primi $\frac{n}{4}$ e gli ultimi $\frac{n}{4}$) siano difficili da calcolare. Formalizziamo questa idea attraverso la definizione del concetto di *predicato hard–core* (hard–core predicate).

Definizione 5.4 (Predicato hard–core). Sia $\mathbb{B} : \{0,1\}^n \to \{0,1\}$ un predicato computabile in tempo polinomiale da una MdT deterministica, e sia $f : \{0,1\}^n \to \{0,1\}^n$ una permutazione one–way. $\mathbb{B}$ è un *predicato hard–core per* f se e solo se per ogni algoritmo A computabile in tempo polinomiale da una MdT probabilistica e per ogni polinomio $p(\cdot)$ esiste un numero naturale $n_{A,p}$ (che dipende sia da A che da p) tale che:

$$\forall\, n \geq n_{A,p} \quad \Pr\big[x \leftarrow \{0,1\}^n;\ \ b \leftarrow A\big(f(x)\big) :\ \ b = \mathbb{B}(x)\big] \leq \frac{1}{2} + \frac{1}{p(n)}$$

■

In altre parole, Eve non è in grado di trovare alcun algoritmo A che sia in grado di indovinare il valore del predicato $\mathbb{B}(x)$ — ovvero di indovinare il valore del corrispondente hard–core bit di x — sostanzialmente meglio di quanto farebbe lanciando una moneta non truccata.

Utilizzando una permutazione one–way f e un predicato hard–core $\mathbb{B}$ per f, possiamo finalmente costruire un generatore pseudocasuale G che, data in ingresso una stringa di k bit, ne produce in uscita una di $k+1$ bit:

$$G(\underbrace{x}_{k \text{ bit}}) = \underbrace{f(x)}_{k \text{ bit}} \circ \underbrace{\mathbb{B}(x)}_{1 \text{ bit}}$$

Questa costruzione è valida in generale, qualunque sia la scelta della permutazione one–way f; è sufficiente che $\mathbb{B}$ sia un predicato hard–core per f.

È evidente che l'output di G è impredicibile: infatti, il valore $\mathbb{B}(x)$ è impredicibile per la definizione stessa di predicato hard–core, mentre se x ed f sono scelti a caso, il valore $f(x)$ è casuale e quindi impredicibile.

Resta ora da vedere qualche esempio di funzione (o permutazione) one–way, e di predicato hard–core per essa. Ricordiamo che, parlando di funzioni one–way, stiamo automaticamente assumendo che sia $P \neq NP$; ricordiamo inoltre che questa è una condizione necessaria ma non sufficiente per l'esistenza di funzioni one–way.

Il primo esempio di funzione one–way è la cosiddetta *esponenziazione modulare*. Sia p un numero primo, e sia $\mathbb{Z}_p$ il campo (di Galois) delle classi dei resti modulo p. Indichiamo con $\mathbb{Z}_p^*$ il gruppo moltiplicativo in $\mathbb{Z}_p$ (ricordiamo che in $\mathbb{Z}_p^*$ non compare lo zero del campo); si dimostra facilmente che $\mathbb{Z}_p^*$ è un gruppo ciclico, ovvero esiste un elemento g di $\mathbb{Z}_p^*$ che lo genera: in altre parole, possiamo scrivere $\mathbb{Z}_p^* = \{g^0, g^1, \ldots, g^{p-2}\}$. La funzione di esponenziazione modulare è definita nel modo seguente:

$$f(z) = g^z$$

ovvero, è la funzione di esponenziazione che si studia in Analisi Matematica, calcolata però a valori in $\mathbb{Z}_p^*$.

Esempio 5.2. Dato $\mathbb{Z}_5 = \langle\{[0]_5, [1]_5, [2]_5, [3]_5, [4]_5\}, +, \cdot, [0]_5, [1]_5\rangle$ abbiamo che $\mathbb{Z}_5^* = \langle\{[1]_5, [2]_5, [3]_5, [4]_5\}, \cdot, [1]_5\rangle$ è il gruppo moltiplicativo di $\mathbb{Z}_5$. $\mathbb{Z}_5^*$ è un gruppo ciclico; infatti, l'elemento $[2]_5$ è un generatore, e possiamo scrivere: $\mathbb{Z}_5^* = \{[2]_5^0, [2]_5^1, [2]_5^2, [2]_5^3\}$. Allora, possiamo definire la seguente funzione di esponenziazione modulare:

$$f(z) = [2]_5^z$$

■

È evidente che, dato z, il calcolo di $f(z)$ può essere fatto in tempo polinomiale (rispetto al numero di bit necessari per rappresentare gli elementi di $\mathbb{Z}_p^*$) da una MdT deterministica. Il calcolo dell'inversa di f (che è ben definita, in quanto f è biunivoca) costituisce il cosiddetto *problema del logaritmo discreto*, e si congettura che sia intrattabile, ovvero che non esista alcun algoritmo efficiente che consente di calcolare il valore di z a partire da g^z. Esistono algoritmi molto efficienti, che però funzionano solo per alcuni casi particolari; nel caso generale, gli algoritmi noti non sono sostanzialmente migliori di quello che prova a calcolare g^0, g^1, g^2, …, e si arresta quando trova il valore g^z dato. Per farsi un'idea della difficoltà del calcolo del logaritmo discreto, si consiglia di provare a cimentarsi a calcolare (a mano!) i logaritmi in $\mathbb{Z}_{113}^*$; una volta appurata la difficoltà del compito, si osservi che $\mathbb{Z}_{113}^*$ è ridicolmente piccolo per applicazioni crittografiche reali, e che tabulare gli elementi di $\mathbb{Z}_p^*$ con i relativi logaritmi richiede in generale $\mathcal{O}(p)$ passi; posto $n = \log_2 p$ (il numero di bit necessari a rappresentare p, e quindi anche gli elementi di $\mathbb{Z}_p$), questo significa che il calcolo del logaritmo discreto in $\mathbb{Z}_p^*$ richiede in generale $\mathcal{O}(2^n)$ passi.

Vediamo ora un hard–core predicate per l'esponenziazione modulare. Blum e Micali hanno dimostrato che, dato $y = g^z \in \mathbb{Z}_p^*$, il calcolo di $\mathrm{msb}(z)$, ovvero del bit più significativo di z, è altrettanto difficile del calcolo di z. In altre parole, $\mathrm{msb}(z)$ è un predicato hard–core per l'esponenziazione modulare. Questo significa che possiamo costruire un generatore pseudocasuale come segue:

$$G(z) = g^z \circ \mathrm{msb}(z)$$

Un'altra funzione one–way, che come vedremo è molto famosa, dato che viene utilizzata nell'RSA, che è il crittosistema a chiave pubblica più famoso, è la funzione di *moltiplicazione*: dati due numeri primi p e q, è immediato verificare che il calcolo di $n = p \cdot q$ può essere svolto in tempo polinomiale (rispetto al numero di bit utilizzati per rappresentare p e q) da una MdT deterministica; tuttavia, il calcolo di p e di q a partire da n dà luogo al cosiddetto problema della *fattorizzazione*, che si congettura essere intrattabile. Anche in questo caso, come per il logaritmo discreto, sembra che la cosa migliore che si possa fare per un n generico sia quella di provare a moltiplicare tutte le possibili coppie di numeri primi minori di $\sqrt{n}$, fino a che si trova la coppia giusta (che è tra l'altro unica, a meno dell'ordine dei fattori, come ci assicura il Teorema Fondamentale dell'Aritmetica); equivalentemente, si può provare a dividere n per tutti i numeri primi compresi tra 2 e $\sqrt{n}$. Quest'ultimo approccio richiede $\mathcal{O}(\sqrt{n})$ passi; detto $k = \log_2 n$ il numero di bit necessari per rappresentare n, otteniamo $\mathcal{O}(\sqrt{2^k}) = \mathcal{O}(2^{k/2})$ passi.

Concludiamo il discorso sulla costruzione dei generatori pseudocasuali enunciando il seguente teorema, di Goldreich e Levin.

Teorema 5.2 (Goldreich–Levin). *Sia $f(x)$ una permutazione one–way, e sia $g(x, y) = f(x) \circ y$, con $|x| = |y| = n$. Allora la funzione:*

$$\mathbb{B}(x, y) \stackrel{\text{def}}{=} \langle x, y \rangle$$

è un predicato hard–core per g. ∎

Osserviamo anzitutto che se f è una permutazione one–way allora anche g lo è: è infatti facile verificare che è un'applicazione biunivoca, e che è one–way in quanto il calcolo di x a partire da $g(x, y)$ implica il calcolo dell'inversa di f.

Il simbolo $\langle \cdot, \cdot \rangle$ indica l'operazione di *prodotto interno modulo* 2, definita come segue.

Definizione 5.5. Siano $x = (x_1, x_2, \ldots, x_n)$ ed $y = (y_1, y_2, \ldots, y_n)$ due sequenze in $\{0, 1\}^n$. Il *prodotto interno modulo* 2 tra x e y, denotato con $\langle x, y \rangle$, è definito come segue:

$$\langle x, y \rangle \stackrel{\text{def}}{=} \sum_{i=1}^{n} x_i y_i \mod 2$$

∎

In pratica si tratta, come dice il nome, del calcolo del prodotto scalare tra i vettori x ed y, effettuato in $\mathbb{Z}_2$. Il teorema di Goldreich–Levin allora dice che, *qualunque sia la permutazione one–way f di partenza*, posso costruire una nuova permutazione one–way g su $\{0,1\}^{2n}$, concatenando semplicemente il secondo argomento (y) all'output di f sul primo argomento, e posso anche costruire un predicato hard–core per g, che è semplicemente il prodotto interno modulo 2 tra gli argomenti di g. Per quanto abbiamo visto, questo significa che possiamo costruirci anche un generatore pseudocasuale, come segue:

$$G(x,y) = g(x,y) \circ \mathbb{B}(x,y) = f(x) \circ y \circ \langle x,y \rangle$$

Il tutto, a partire dalla sola f. Osserviamo che il generatore G è del tipo $\{0,1\}^{2n} \to \{0,1\}^{2n+1}$.

6 Crittosistemi a chiave pubblica

Tutti i metodi di cifratura e decifratura a chiave segreta che abbiamo visto hanno un problema: sia Alice che Bob devono conoscere la chiave k, e quindi prima di iniziare a scambiarsi messaggi cifrati devono in qualche modo mettersi d'accordo su quale chiave utilizzare. È chiaro che questo è un punto debole dell'intero protocollo di comunicazione: per fare in modo che Eve non sia in grado di decifrare i messaggi, Alice e Bob devono usare un canale sicuro per scambiarsi la chiave. Ma non sempre è possibile stabilire una comunicazione usando un canale sicuro; inoltre, se quest'ultimo esistesse sempre, lo si potrebbe utilizzare per scambiarsi i messaggi, senza prendersi il disturbo di cifrarli e decifrarli!

L'uso delle tecniche di crittografia a chiave pubblica consente di risolvere il problema alla radice: la chiave usata per cifrare un messaggio è in generale diversa da quella usata per decifrarlo (motivo per cui i crittosistemi a chiave pubblica sono noti anche come *crittosistemi asimmetrici*); inoltre, la chiave usata per cifrare può (e, come vedremo presto, deve) essere resa pubblica, mentre la chiave usata per decifrare deve restare segreta, e deve essere nota solamente al destinatario legittimo del messaggio.

Il meccanismo può essere facilmente spiegato attraverso un esempio. Supponiamo che Alice voglia inviare un messaggio a Bob; scrive il messaggio su un foglio di carta, e quindi chiude il foglio dentro a una cassetta. Nel tipico approccio a chiave segreta, Alice potrebbe usare un lucchetto per chiudere la cassetta, e poi spedire a Bob la cassetta e la chiave in tempi e modi diversi. Il problema, come abbiamo visto, è che se Eve intercetta entrambe diventa in grado di aprire la cassetta e di leggere il messaggio. Nell'approccio a chiave pubblica, invece, Bob manda ad Alice una cassetta dotata di un lucchetto *aperto* di cui solo lui ha la chiave; Alice mette il foglio dentro alla cassetta, e chiude il lucchetto. A questo punto, solo Bob è in grado di aprire la cassetta, grazie alla chiave che solo lui possiede; pertanto, Alice può spedire tranquillamente la cassetta a Bob. Affinché l'esempio sia significativo, occorre pensare che il lucchetto e

la cassetta siano *molto* difficili da scassinare; tanto per fare un esempio, se le apparecchiature necessarie per aprire la cassetta senza avere la chiave costassero un milione di dollari, Eve dovrebbe essere *ben determinata* per procurarsele, e chiaramente lo farebbe solo se ne avesse un ritorno vantaggioso. Se invece le apparecchiature dovessero costare come l'intero Prodotto Interno Lordo (PIL) del pianeta, potremmo essere ragionevolmente sicuri che le cassette sono inviolabili.

Un metodo alternativo per spedire un messaggio utilizzando le cassette, senza dover trasferire la chiave è il seguente: Alice scrive il messaggio che vuole mandare a Bob, lo mette in una cassetta dotata di due chiusure ad anello, e ne chiude una con un lucchetto di cui solo lei ha la chiave. Spedisce la cassetta a Bob; questi non può aprirla, dato che non possiede la chiave del lucchetto, e quindi chiude la seconda chiusura ad anello con un proprio lucchetto, di cui solo lui ha la chiave, e rispedisce la cassetta ad Alice. Ora neanche Alice è più in grado di aprire la cassetta, dato che non possiede la chiave del lucchetto di Bob; l'unica cosa che può fare è togliere il proprio lucchetto, e rispedire la cassetta a Bob. Questi, finalmente, può aprire la cassetta e leggere il messaggio.

Vediamo ora una possibile formalizzazione di queste due tecniche. Per il momento restiamo su un livello astratto, e non mostriamo alcuna implementazione particolare. Per quanto riguarda la prima tecnica, si procede in questo modo: Bob sceglie due algoritmi (in realtà vedremo poi che è sufficiente che Bob selezioni due chiavi k_p e k_s, che individuano gli algoritmi richiesti all'interno di una opportuna famiglia di algoritmi), uno di cifratura — che indichiamo con E_B — e uno di decifratura — che indichiamo con D_B — tali che:

- D_B è in grado di decifrare i messaggi cifrati con E_B, ovvero:

$$\forall\, m \in PT \qquad D_B\big(E_B(m)\big) = m$$

- è estremamente difficile (cioè costituisce un problema intrattabile) ricavare D_B a partire dalla conoscenza di E_B.

A questo punto, Bob rende pubblico l'algoritmo di cifratura E_B, e custodisce segretamente l'algoritmo di decifratura D_B.

Ora, se Alice vuole mandare un messaggio m a Bob, lo cifra utilizzando l'algoritmo E_B e manda il risultato $E_B(m)$ attraverso il canale insicuro. Quando Bob riceve il messaggio cifrato da Alice, lo decifra utilizzando l'algoritmo D_B (che solo lui conosce) e recupera il testo in chiaro m. Se Eve dovesse intercettare $E_B(m)$, per risalire ad m dovrebbe in pratica scoprire qual è l'algoritmo D_B, ovvero dovrebbe risolvere un problema che abbiamo supposto essere intrattabile.

Osserviamo che, per mandare un messaggio a Bob, Alice deve conoscere l'algoritmo E_B generato da Bob; nessun altro algoritmo andrà bene, dato che altrimenti Bob non sarebbe in grado di decifrare correttamente il messaggio ricevuto. Nel caso in cui Bob voglia mandare un messaggio ad Alice, questa dovrà a sua volta generare due algoritmi E_A e D_A, e così via. Si capisce immediatamente che se Alice fa parte di una grossa rete, e vuole mandare

messaggi a molte persone, dovrà gestire un gran numero di algoritmi di cifratura, i quali cambiano periodicamente. Ciò è indubbiamente molto scomodo, e può essere evitato adottando il secondo schema, analogo all'uso di una cassetta con due chiusure.

Nel secondo schema, sia Alice che Bob generano un algoritmo di cifratura (che indichiamo rispettivamente con E_A ed E_B) e uno di decifratura (che indichiamo rispettivamente con D_A e D_B). Come abbiamo fatto precedentemente, richiediamo che valgano le seguenti proprietà:

- $\forall\, m \in M \qquad E_B\big(E_A(m)\big) = E_A\big(E_B(m)\big)$
- $\forall\, m \in M \qquad D_B\big(D_A(m)\big) = D_A\big(D_B(m)\big)$

Più in generale, possiamo richiedere che l'ordine di applicazione di E_A, E_B, ... e di D_A, D_B, ... sia *immateriale*.

A questo punto, gli algoritmi E_A ed E_B possono anche essere resi pubblici (ma per l'applicazione del secondo schema ciò non è necessario); è importante, invece, che l'unica a conoscere D_A sia Alice, e che l'unico a conoscere D_B sia Bob. Il secondo schema può allora essere formalizzato come segue: se Alice vuole mandare un messaggio m a Bob, lo cifra con E_A (applica il proprio lucchetto), e spedisce il risultato a Bob. Questi cifra il messaggio $E_A(m)$ ricevuto con E_B (applica il proprio lucchetto) e spedisce il risultato ad Alice. Questa rimuove il proprio lucchetto, cioè applica D_A al messaggio $E_B\big(E_A(m)\big)$ ricevuto, e ottiene:

$$D_A\big(E_B\big(E_A(m)\big)\big) = D_A\big(E_A\big(E_B(m)\big)\big) = E_B(m)$$

Infine, Alice spedisce $E_B(m)$ a Bob, il quale recupera m applicando D_B.

Osserviamo che in questo caso Alice applica solamente gli algoritmi E_A e D_A, e non ha bisogno di conoscere neppure E_B; quando un utente della rete teme che il proprio algoritmo di decifratura possa essere stato scoperto, lo può cambiare tranquillamente (modificando quindi anche l'algoritmo di cifratura corrispondente) senza causare alcun problema agli altri utenti.

Come dicevamo, la situazione di gran lunga più comune è quella in cui vi è una famiglia $\{E_k : k \in K\}$ di funzioni di cifratura, e una famiglia $\{D_k : k \in K\}$ di funzioni di decifratura. Come abbiamo visto, nei crittosistemi simmetrici la stessa chiave $k \in K$ viene usata sia per cifrare i messaggi che per decifrarli; questo non è più vero per i crittosistemi a chiave pubblica: in tali crittosistemi ogni utente sceglie in maniera opportuna due chiavi k_s e k_p, entrambi appartenenti a K, tali che:

- $\forall\, m \in PT \qquad D_{k_s}\big(E_{k_p}(m)\big) = m$
- è estremamente difficile ricavare il valore di k_s a partire da quello di k_p

e quindi procede come indicato sopra. Se si adotta il primo protocollo (equivalente all'uso di una cassetta con un solo lucchetto), sarà *necessario*

rendere pubblica la chiave k_p, mentre k_s dovrà restare segreta; se si adotta il secondo protocollo (equivalente all'uso di una cassetta con due lucchetti), si potrà addirittura mantenere segreto il valore di k_p (anche se non è strettamente necessario).

Vediamo ora un esempio di crittosistema a chiave pubblica, nel quale la difficoltà di ricavare k_s da k_p e la difficoltà di ricavare m da $E_{k_p}(m)$ sono una immediata conseguenza della (congetturata) intrattabilità del problema del *logaritmo discreto*. Il crittosistema è stato introdotto da El Gamal nel 1984. Ogni utente, per generare la propria coppia (k_s, k_p) di chiavi, svolge le seguenti operazioni:

- sceglie un numero primo q;
- considera il campo di Galois $\mathbb{Z}_q$ delle classi di resto modulo q, e il suo gruppo moltiplicativo $\mathbb{Z}_q^*$;
- sceglie un generatore g di $\mathbb{Z}_q^*$ (quindi, $\mathbb{Z}_q^* = \{g^0, g^1, g^2, \ldots, g^{q-2}\}$);
- sceglie a caso un intero a tale che $0 < a < q-1$, e calcola g^a.

La chiave segreta è $k_s = a$, e la chiave pubblica è $k_p = (q, g, g^a)$. Osserviamo che, volendo calcolare k_s a partire da k_p, sarebbe necessario calcolare il valore di a a partire dai valori noti q, g e g^a; in altre parole, sarebbe necessario calcolare il logaritmo (discreto) $\log_g g^a$ in $\mathbb{Z}_q^*$, che è un compito ritenuto molto difficile.

Supponiamo che Bob abbia effettuato le operazioni di cui sopra, e che disponga quindi di una chiave segreta $k_{s,B}$ e di una chiave pubblica $k_{p,B}$; se Alice vuole mandare un messaggio a Bob, supponendo di adottare il protocollo equivalente all'uso di cassette dotate di un singolo lucchetto, deve:

- prelevare la chiave pubblica di Bob: $k_{p,B} = (q, g, g^a)$;
- rappresentare il messaggio da spedire come un elemento $m \in \mathbb{Z}_q^*$; questo è sempre possibile, rappresentando il messaggio in binario e suddividendolo in blocchi in modo tale che ogni blocco rappresenti un numero intero compreso tra 1 e $q-1$;
- scegliere a caso un intero l, con $0 < l < q-1$;
- calcolare $\gamma = g^l \mod q$ e $\delta = m \cdot (g^a)^l \mod q$ (che risultano essere entrambi elementi di $\mathbb{Z}_q^*$);
- inviare a Bob il testo cifrato $c = (\gamma, \delta)$.

Quando Bob riceve il testo cifrato c, utilizza la propria chiave privata a per calcolare m come segue:

$$m = g^{-al} \cdot \delta = \gamma^{-a} \cdot \delta = \gamma^{q-1-a} \cdot \delta \tag{2}$$

Come si può notare, il calcolo di m comporta l'utilizzo — oltre che dei valori γ e δ ricavati da c — solamente dei valori q (pubblico) e a (privato); pertanto, se Eve dovesse intercettare il messaggio cifrato c, l'unica informazione che le mancherebbe per ricavare m sarebbe a. Osservando la formula (2) si vede come — non conoscendo il valore di a — l'unica possibilità che ha Eve è quella di provare a calcolare, per ogni valore e tale che $0 < e < q - 1$, l'espressione $\gamma^{q-1-e} \cdot \delta$, fino a che non trova un risultato che possa essere interpretato come un testo in chiaro dotato di senso compiuto. Osserviamo poi che il metodo di cifratura e di decifratura sopra esposto si riferisce ad un unico blocco m di testo in chiaro, rappresentabile come un intero compreso tra 1 e $q - 1$; nel caso in cui un testo in chiaro sia troppo lungo per poterlo rappresentare in tale intervallo, lo si dovrà suddividere in blocchi di opportuna lunghezza, e operare separatamente per ciascun blocco.

Abbiamo già visto che se fossimo in grado di risolvere il problema del logaritmo discreto allora potremmo facilmente ricavare il valore di $k_s (= a)$ a partire quello di k_p (in particolare, da g^a). Tuttavia, questa è una condizione *sufficiente* ma *non necessaria* per rompere il sistema: un altro metodo potrebbe essere quello di usare la conoscenza di g^l (ovvero γ) e di g^a per calcolare g^{al}; g^{-al} potrebbe essere ricavato facilmente dall'equazione $x \cdot g^{al} \equiv 1 \mod q$ (che si risolve tranquillamente in tempo polinomiale rispetto al numero di bit necessari per rappresentare gli elementi di $\mathbb{Z}_q^*$), e potremmo così calcolare $m = g^{-al} \cdot \delta$. Si congettura però che non esista alcun metodo che riesca a calcolare g^{al} a partire da g^l e da g^a senza essenzialmente risolvere il problema del logaritmo discreto.

7 Crittosistemi ibridi e scambio di chiavi

Come abbiamo visto, l'uso di crittosistemi a chiave pubblica ci consente di comunicare utilizzando un canale insicuro, senza doversi preventivamente accordare su una chiave segreta, e senza dover supporre l'esistenza di un canale sicuro lungo il quale scambiarla.

Il fatto che le tecniche a chiave pubblica che abbiamo visto siano nate solamente nella seconda metà degli anni '70, mentre la crittografia più in generale esiste fin dai tempi antichi, potrebbe in effetti stupire: le idee di base non sono complicate, e allora come mai c'è voluto tanto tempo per formularle? Il motivo è molto semplice: la crittografia a chiave pubblica si basa sull'esistenza di funzioni one–way, le quali a loro volta si basano su assunzioni riguardanti la difficoltà di risoluzione di certi problemi che vengono studiati nella Teoria della Complessità. Quest'ultima è una disciplina molto giovane, che ha conosciuto un forte sviluppo proprio a partire dagli anni '70 (ricordiamo che il famoso teorema di Cook–Levin sulla NP–completezza di SAT è proprio del 1970). A partire da quegli anni la crittografia è stata rifondata sulla Teoria della Complessità, di cui è diventata un ramo importante; questo nuovo approccio viene indicato solitamente in letteratura con il termine di "Crittografia moderna".

Sembrerebbe che l'uso dei crittosistemi a chiave pubblica risolva ogni problema, e che quindi non ci sia più spazio per l'utilizzo dei crittosistemi simmetrici. Tuttavia, tutti gli algoritmi a chiave pubblica noti sono notevolmente più lenti dei migliori (e più diffusi) algoritmi simmetrici; si pensi a tutte le situazioni in cui dispositivi di calcolo *poco potenti* (telefoni cellulari, smart card, dispositivi embedded) devono comunicare con altri dispositivi in maniera cifrata. È evidentemente inaccettabile dover attendere 10 minuti ogni volta che si vuol fare una telefonata, oppure quando si vuole fare un pagamento elettronico; vorremmo quindi trovare per questi dispositivi degli algoritmi che siano veloci come quelli simmetrici e sicuri come quelli a chiave pubblica. La soluzione, come sarebbe logico aspettarsi, consiste nell'usare un metodo ibrido: Alice sceglie una chiave k (che, per il fatto che può anche essere di volta in volta diversa, viene spesso chiamata *chiave di sessione*) e la comunica a Bob utilizzando un crittosistema a chiave pubblica; una volta che entrambi conoscono k, possono usarla come chiave di un crittosistema simmetrico per scambiarsi messaggi. Utilizzando questo approccio, il lento crittosistema a chiave pubblica viene usato solamente all'inizio della sessione di comunicazione, per scambiarsi la chiave; essendo questa generalmente molto più corta dei successivi messaggi, il tempo impiegato per cifrarla e decifrarla diventa solitamente trascurabile rispetto all'intera durata della sessione.

Un'alternativa all'uso di un crittosistema a chiave pubblica per scambiarsi la chiave k è quella di utilizzare il metodo di Diffie–Hellman, inventato per l'appunto da Whitfield Diffie e Martin Hellman nel 1976, e indipendentemente da Ralph Merkle. Alice e Bob scelgono un numero primo q (che può essere tranquillamente reso pubblico), e considerano il gruppo moltiplicativo $\mathbb{Z}_q^*$ del campo di Galois $\mathbb{Z}_q$. Inoltre, si accordano su un generatore g di $\mathbb{Z}_q^*$ (che, ricordiamo, è un gruppo ciclico). A questo punto, Alice sceglie a caso un intero x_A, con $0 < x_A < q-1$, calcola il valore g^{x_A} e lo invia a Bob; dal canto suo, Bob sceglie a caso un intero x_B, con $0 < x_B < q-1$, calcola il valore g^{x_B} e lo invia ad Alice. Quando Bob riceve il valore g^{x_A}, calcola $(g^{x_A})^{x_B} = g^{x_A x_B}$; analogamente, Alice riceve il valore g^{x_B} e calcola $(g^{x_B})^{x_A} = g^{x_A x_B}$; la chiave comune è allora $k = g^{x_A x_B}$. La sicurezza del protocollo è basata sulla difficoltà (presunta) di ricavare $g^{x_A x_B}$ a partire dai valori noti di q, g, g^{x_A} e g^{x_B}, che sono gli unici valori che transitano sul canale. In alternativa, Eve dovrebbe essere in grado di ricavare, ad esempio, x_A a partire da g^{x_A} (cioè dovrebbe essere in grado di risolvere il problema del logaritmo discreto), così poi potrebbe calcolare $g^{x_A x_B}$ come $(g^{x_B})^{x_A}$. Pertanto, osserviamo che i due problemi ritenuti difficili sui quali si basa la sicurezza dello scambio di chiavi di Diffie–Hellman sono gli stessi su cui si basa la sicurezza del crittosistema a chiave pubblica di El Gamal.

Esempio 7.1. Supponiamo che Alice e Bob si accordino per usare $q = 25307$ e $g = 2$ (che è un generatore di $\mathbb{Z}_{25307}^*$). Se Alice sceglie il numero $x_A = 3578$, e Bob sceglie il numero $x_B = 19956$, allora abbiamo che:

- Alice calcola $g^{x_A} = 2^{3578} = 6113 \pmod{25307}$, e lo spedisce a Bob; Bob

allora calcola:

$$k = (g^{x_A})^{x_B} = 6113^{19956} = 3694 \pmod{25307}$$

- Bob calcola $g^{x_B} = 2^{19956} = 7984 \pmod{25307}$ e lo spedisce ad Alice; Alice allora calcola:

$$k = (g^{x_B})^{x_A} = 7984^{3578} = 3694 \pmod{25307}$$

La chiave $k = 3694$ può ora essere utilizzata per scambiarsi messaggi utilizzando un crittosistema simmetrico. ∎

8 RSA

Il crittosistema a chiave pubblica sicuramente più famoso, più utilizzato e più studiato è l'RSA, che prende il suo nome dalle iniziali degli autori: Rivest, Shamir e Adleman. Tale crittosistema utilizza in maniera molto intelligente il fatto che il problema della fattorizzazione, consistente nel trovare i fattori primi p e q di un dato numero n $(= p \cdot q)$ viene considerato — allo stato attuale delle conoscenze — intrattabile. Più precisamente, si congettura che il problema consistente nel ricavare il testo in chiaro a partire dal testo cifrato e dalla chiave pubblica sia equivalente al problema della fattorizzazione.

La descrizione del funzionamento di RSA è estremamente semplice. Si scelgano a caso due numeri primi p e q, distinti (cioè $p \neq q$) e approssimativamente della stessa lunghezza (ovvero rappresentabili tramite lo stesso numero di bit). Affinché RSA possa essere usato per scopi crittografici con una certa sicurezza, si richiede che p e q non possano essere rappresentati con meno di 100 cifre in base 10. Poniamo $n = p \cdot q$; osserviamo che, per il problema della fattorizzazione, è estremamente difficile risalire a p e a q, dato n: il massimo della difficoltà lo si ottiene proprio quando p e q sono approssimativamente della stessa lunghezza. Quindi, si calcola il valore $\phi(n) = (p-1)(q-1)$ della cosiddetta *funzione di Eulero* $\phi(\cdot)$ per n; $\phi(x)$ è definita come il numero di interi $i \leq x$ che sono *coprimi* con x, ovvero tali che $\mathrm{MCD}(i, x) = 1$.[6] In pratica, detto $\mathbb{Z}_x$ l'anello (che non è necessariamente un campo, dato che l'intero x può anche non essere un primo) delle classi dei resti modulo x, si dimostra che l'insieme $\{i | (i, x) = 1\}$ è l'insieme di rappresentanti delle classi di resto che formano il più grosso gruppo moltiplicativo contenuto in $\mathbb{Z}_x$; detto $\mathbb{Z}_x^*$ tale gruppo, abbiamo che $\phi(x)$ è la cardinalità di $\mathbb{Z}_x^*$. Si dimostra inoltre che vale la seguente proprietà: se x e y sono numeri interi positivi tali che $\mathrm{MCD}(x, y) = 1$, allora vale:

$$\phi(x \cdot y) = \phi(x) \cdot \phi(y)$$

[6] In ambito crittografico, al posto di $\mathrm{MCD}(i, x)$ si usa scrivere (i, x). Adotteremo pertanto questo simbolo anche nelle presenti note, anche se effettivamente non dà alcun vantaggio significativo, e porta a confusione quando compare insieme a coppie (x, y) di elementi di un insieme.

Pertanto, abbiamo:

$$\phi(n) = \phi(p) \cdot \phi(q)$$

e, poiché $\mathbb{Z}_p^*$ e $\mathbb{Z}_q^*$ sono i gruppi moltiplicativi contenuti nei campi $\mathbb{Z}_p$ e $\mathbb{Z}_q$ rispettivamente, abbiamo:

$$\phi(p) = p - 1$$
$$\phi(q) = q - 1$$

e quindi $\phi(n) = (p-1)(q-1)$.

A questo punto, si sceglie a caso un numero d (possibilmente grosso) tale che $1 < d < \phi(n)$ e $\big(d, \phi(n)\big) = 1$, e si calcola — utilizzando l'algoritmo esteso di Euclide[7] — l'unico intero e tale che $1 < e < \phi(n)$ e:

$$ed \equiv 1 \mod \phi(n)$$

Ovvero $e \equiv d^{-1} \mod \phi(n)$, e diciamo che e è l'inverso di d (modulo $\phi(n)$). La chiave pubblica del crittosistema è $k_p = (n, e)$, mentre la chiave privata k_s — da usare per la decifratura — è costituita da d. I valori p, q e $\phi(n)$, nonostante non siano strettamente necessari per la decifratura, vanno anch'essi mantenuti segreti. La sicurezza del crittosistema RSA è basata sul fatto che la funzione:

$$\mathrm{RSA}(n, e, x) \stackrel{\text{def}}{=} x^e \mod n$$

è congetturata essere una funzione one–way; inoltre, $\mathrm{lsb}(x)$, il bit meno significativo di x, è congetturato essere un hard–core bit per la funzione RSA.

Se Alice vuole inviare un messaggio a Bob, preleva la chiave pubblica $k_{p,B} = (n_B, e_B)$ di Bob, rappresenta il messaggio come un intero m compreso nell'intervallo $[0 \ldots n_B - 1]$, calcola:

$$c = m^{e_B} \mod n_B$$

e invia c a Bob. Una volta che Bob ha ricevuto c, per recuperare il messaggio originale m calcola semplicemente:

$$c^{d_B} \mod n_B$$

utilizzando la propria chiave privata d_B. Bob ritrova esattamente m poiché:

$$\begin{aligned} c^{d_B} \mod n_B &\equiv (m^{e_B})^{d_B} \mod n_B \\ &\equiv m^{e_B d_B} \mod n_B \\ &\equiv m^{k\phi(n_B)+1} \mod n_B \qquad (3) \\ &\equiv m \cdot (m^{\phi(n_B)})^k \mod n_B \qquad (4) \\ &\equiv m \mod n_B \qquad (5) \end{aligned}$$

[7] Ricordiamo che l'algoritmo di Euclide consente di calcolare il MCD tra due numeri interi qualsiasi x ed y. Detto $d = \mathrm{MCD}(x, y)$, l'*identità di Bezout* afferma che esistono due interi a e b tali che $d = ax + by$; l'algoritmo esteso di Euclide, risalendo le divisioni successive svolte per calcolare d, consente di determinare il valore di a e di b.

Infatti, poiché $e_B d_B \equiv 1 \mod \phi(n_B)$, allora $e_B d_B - 1$ è un multiplo di $\phi(n_B)$, ovvero esiste un intero k tale che $e_B d_B - 1 = k\phi(n_B)$. Tale intero è stato utilizzato per scrivere l'equazione (3); per passare da (4) a (5), si utilizza il seguente teorema di Eulero.

Teorema 8.1 (Eulero). *Siano a ed m due interi tali che $(a, m) = 1$. Allora:*

$$a^{\phi(m)} \equiv 1 \mod m$$

■

Pertanto, in (4) abbiamo che $m^{\phi(n_B)} \equiv 1 \mod n_B$, $(m^{\phi(n_B)})^k \equiv 1 \mod n_B$, e quindi $m \cdot (m^{\phi(n_B)})^k \equiv m \mod n_B$. Nei casi in cui $(m, n_B) \neq 1$, cioè nei casi in cui p, q o entrambi dividono m, non è possibile applicare il teorema di Eulero; tuttavia, anche in questi casi la decifratura di RSA avviene correttamente, come dimostrato nel Lemma 4.1 a pag. 126 del libro di testo [1].

A titolo d'esempio (i numeri sono stati scelti piccoli per semplicità), supponiamo che Bob scelga $p = 101$ e $q = 113$. Allora $n_B = p \cdot q = 11413$ e $\phi(n_B) = (p-1)(q-1) = 11200$. Ora Bob deve scegliere a caso un numero d_B, compreso tra 2 e $\phi(n_B) - 1$, tale che $(d_B, \phi(n_B)) = 1$. In pratica, quello che fa Bob è prendere a caso un numero d_B compreso tra 2 e 11199 e utilizzare l'algoritmo di Euclide per verificare che sia $(d_B, 11200) = 1$; se quest'ultima condizione non è verificata, si prova con un altro valore di d_B, e così via. Dato che stiamo lavorando con numeri piccoli, possiamo osservare che $11200 = 2^6 \cdot 5^2 \cdot 7$, e quindi Bob deve scegliere un d_B che non sia divisibile per 2, 5 e 7. Supponiamo che scelga $d_B = 6597$; tramite l'algoritmo di Euclide esteso calcola $e_B \equiv d_B^{-1} \mod \phi(n_B) \equiv 3533 \mod 11200$. La chiave pubblica di Bob è allora $k_{p,B} = (n_B, e_B) = (11413, 3533)$, mentre la chiave segreta è $k_{s,B} = d_B = 6597$. Se Alice vuole inviare a Bob il messaggio $m = 9726$, calcola $c = 9726^{3533} \mod 11413 = 5761$ e lo invia a Bob; questi, ricevuto $c = 5761$, recupera m calcolando $5761^{6597} \mod 11413 = 9726$.

Affinché l'implementazione di RSA risulti efficiente, osserviamo che è possibile implementare l'*esponenziazione modulare* sfruttando il più possibile l'operazione di *elevamento al quadrato*, riducendo così il numero di moltiplicazioni in $\mathbb{Z}_n$. Infatti, se dobbiamo calcolare

$$a^r \mod n$$

a partire da a e da r, utilizzando la definizione di esponenziazione dovremmo calcolare:

$$\begin{aligned} a^2 &= a \cdot a \\ a^3 &= a^2 \cdot a \\ &\vdots \\ a^r &= a^{r-1} \cdot a \end{aligned}$$

effettuando quindi $r-1$ moltiplicazioni. Se invece scriviamo r come:

$$r = \sum_{j=0}^{k} x_j 2^j$$

con $x_j \in \{0,1\}$ per ogni $j = 0, 1, 2, \ldots, k$ e $k = \lfloor \log_2 r \rfloor + 1$ (in pratica, la sequenza $x_k \ldots x_1 x_0$ è la rappresentazione binaria di r), possiamo anzitutto calcolare:

$$\begin{aligned} &a^0 \mod n = 1 \\ &a^2 \mod n \\ &a^4 \mod n \\ &\vdots \\ &a^{2^k} \mod n \end{aligned}$$

ovvero $a^{2^j} \mod n$ per ogni $j = 0, 1, \ldots, k$. Dati i valori di $a^{2^j} \mod n$, possiamo quindi calcolare $a^r \mod n$ facendo al più $k-1$ moltiplicazioni.

Una questione molto più importante è la seguente: scelti due numeri p e q molto grandi, come facciamo a sapere se sono primi? Un possibile metodo sarebbe quello di fattorizzarli: se non hanno fattori propri allora sono primi. Ma abbiamo visto che il problema della fattorizzazione viene ritenuto difficile da risolvere, e non sono noti algoritmi efficienti per fattorizzare un numero intero; quindi, come possiamo fare?

In realtà, per sapere se un certo numero intero è primo oppure no, non è necessario fattorizzarlo; mentre è evidente che se sappiamo che un certo numero è primo allora ne conosciamo pure la fattorizzazione (banale), potremmo benissimo sapere che un certo numero è composto senza conoscerne la fattorizzazione. Esistono molti criteri che ci consentono di decidere se un certo numero è composto; tali criteri sono della forma:

$$\text{se } C(x) \text{ è vero, allora } x \text{ è composto}$$

dove $C(\cdot)$ è un predicato unario che esprime una qualche proprietà dei numeri interi. C'è però un problema: non si conosce alcun criterio *efficiente* (che possa cioè essere verificato in tempo polinomiale da una MdT deterministica) che sia verificato se e soltanto se il numero x è composto. In altre parole, ad ogni criterio efficiente è associato un predicato $C(\cdot)$ tale che:

- se $C(x)$ è vero, possiamo concludere con certezza che x è composto;
- se $C(x)$ è falso, x potrebbe essere primo, ma potrebbe anche essere composto.

Formalmente, l'antiimmagine di "vero" per il predicato $C(\cdot)$ è un sottoinsieme proprio dei numeri composti. Inoltre, non si conosce nemmeno un insieme di

criteri efficienti le cui antiimmagini di "vero" dei predicati associati formino una *copertura* dell'insieme dei numeri composti[8]. Pertanto, il meglio che possiamo fare è di scegliere un insieme di criteri e di applicarli sul numero x; se tutti i criteri rispondono "falso" dovremmo avere una buona probabilità che x è primo (e la probabilità è tanto più alta quanto più numerosi sono i criteri adottati). Uno di tali criteri è basato sul teorema di Eulero che abbiamo visto; si supponga che m sia un intero dispari, e si consideri un intero w tale che $(w, m) = 1$.[9] Se m è primo, allora per il teorema di Eulero (che in questo caso viene anche chiamato Piccolo Teorema di Fermat) vale:

$$w^{m-1} \equiv 1 \mod m \tag{6}$$

Se invece m è composto, è *possibile ma poco probabile* che valga l'equazione (6); nel caso in cui ciò accada, si dice che m è uno *pseudoprimo rispetto alla base* w. Queste affermazioni portano a formulare il seguente test di composizione (ovvero, di non primalità): m passa il test $C(m)$ se e solo se:

$$w^{m-1} \not\equiv 1 \mod m$$

per qualche w tale che $(w, m) = 1$. Se m fallisce il test $C(m)$ per w, ovvero (6) vale, m potrebbe nondimeno essere composto. Un intero w tale che $(w, m) = 1$ e per il quale vale l'equazione (6) viene detto un *testimone* per la primalità di m; come abbiamo detto, esistono anche "falsi testimoni", rispetto ai quali m è solamente uno pseudoprimo. Un metodo che consente di mostrare con alta probabilità che m è primo consiste nel raccogliere molti testimoni per la primalità di m. Vale infatti il seguente:

Lemma 8.2 (pag. 138 del libro di testo). *Tutti oppure al più metà degli interi w tali che $1 \leq w < m$ e $(w, m) = 1$ sono testimoni per la primalità di m.* ∎

Possiamo allora procedere come segue: dato m, scegliamo a caso un intero w tale che $1 \leq w < m$. Quindi, calcoliamo il massimo comun divisore (w, m) tramite l'algoritmo di Euclide; se $(w, m) > 1$, w ed m hanno un fattore comune, e quindi m è composto. Altrimenti, se $(w, m) = 1$, calcoliamo $u = w^{m-1} \mod m$ utilizzando il metodo dei quadrati successivi. Se $u \neq 1$, per il criterio di Eulero possiamo concludere che m è composto; se invece $u = 1$, w è un testimone per la primalità di m, e abbiamo quindi raccolto una prova del fatto che m potrebbe essere primo. Come abbiamo detto, più testimoni troviamo e più è alta la probabilità che m sia primo. Per il lemma di cui sopra, quando abbiamo trovato k testimoni abbiamo che la probabilità che m sia composto è al più 2^{-k}, tranne che per il caso sfortunato in cui *tutti* i numeri w tali che $1 \leq w < m$ e $(w, m) = 1$ sono testimoni.

[8] Ovvero, la cui unione dia esattamente l'insieme dei numeri composti.

[9] Se fosse $(w, m) > 1$, potremmo concludere immediatamente che sia m che w sono composti.

Osserviamo che se m è primo allora tutti i numeri w compresi tra 1 ed $m-1$ sono testimoni, e le prove raccolte avvallano la conclusione corretta. Tuttavia, può accadere che tutti i numeri w compresi tra 1 ed $m-1$ (e tali che $(w,m)=1$) siano testimoni *senza che* m sia primo. I numeri interi m per cui ciò accade vengono detti *numeri di Carmichael*, e per fortuna sono molto rari.

Oltre al criterio di primalità che abbiamo appena esposto ne esistono molti altri; ad esempio, sul libro di testo [1] si può trovare una descrizione dei test di Solovay–Strassen e di Miller–Rabin.

9 Attacchi ad RSA

Accenniamo ad alcuni possibili attacchi che possono essere sferrati ad RSA, riguardanti la scelta dei parametri p e q, e indichiamo le misure da adottare per limitare l'efficacia di tali attacchi.

Anzitutto, occorre dire che è solo *congetturato* che il problema della fattorizzazione sia intrattabile; inoltre, è *congetturato* che il problema consistente nel rompere RSA sia equivalente a quello di fattorizzare n. Se anche una sola di tali congetture dovesse risultare falsa, dovremmo abbandonare l'uso di RSA e cercare un crittosistema alternativo. In particolare, osserviamo che se la seconda congettura dovesse risultare falsa allora esisterebbe un metodo che consente di ricavare la chiave segreta d a partire dalla chiave pubblica (n,e) *senza* far uso della fattorizzazione di n in p e q. D'altra parte, visto che non si sa se tale metodo esiste, tutti gli attacchi rivolti verso RSA proposti finora fanno uso di particolari caratteristiche di p e q che consentono di ricavarli a partire da n, oppure che consentono di ricavare il valore di $\phi(n)$ (che, ricordiamo, deve restare segreto).

Infatti, noti p e q, Eve potrebbe calcolare:

$$\phi(n) = (p-1)(q-1)$$

e quindi potrebbe ricavare la chiave segreta d a partire da e (che è pubblico) risolvendo la seguente congruenza lineare:

$$e \cdot x \equiv 1 \mod \phi(n)$$

Se invece Eve conoscesse $\phi(n)$, potrebbe ricavare p e q risolvendo il seguente sistema nelle incognite p e q:

$$\begin{cases} n = pq \\ \phi(n) = (p-1)(q-1) \end{cases}$$

Possiamo infatti riscrivere $\phi(n)$ come:

$$\phi(n) = pq - (p+q) + 1$$

da cui:

$$p + q = n - \phi(n) + 1$$

Dato che conosciamo sia n che $\phi(n)$, conosciamo anche quanto vale la somma $(p+q)$ e il prodotto (pq) delle nostre incognite. Risolvendo l'equazione di secondo grado:

$$x^2 - (p+q)x + pq = 0$$

otteniamo i valori di p e q cercati. Pertanto, se Eve conoscesse il valore di $\phi(n)$ potrebbe rompere il crittosistema. Tuttavia, si congettura che il calcolo di $\phi(n)$ non sia più semplice della fattorizzazione di n.

Anche la scelta dei parametri p e q dovrebbe essere fatta con cura. Ad esempio, non dovrebbero essere troppo vicini: senza perdere in generalità, supponiamo che sia $p > q$; se p e q fossero vicini, allora il valore $(p-q)/2$ sarebbe piccolo, e il valore $(p+q)/2$ sarebbe solo leggermente più grande di $\sqrt{n}$. Inoltre, vale sempre l'uguaglianza:

$$(p+q)^2/4 - n = (p-q)^2/4$$

da cui si deduce che $(p+q)^2/4 - n$ è un quadrato perfetto. Per fattorizzare n è allora sufficiente testare gli interi $x > \sqrt{n}$ finché non se ne trova uno per cui il valore $x^2 - n$ è un quadrato perfetto. Detto y^2 tale quadrato perfetto, abbiamo:

$$x^2 - n = y^2$$

da cui:

$$n = x^2 - y^2 = (x+y)(x-y)$$

e quindi $p = x + y$ e $q = x - y$.

Anche il valore di $\phi(n)$ dovrebbe essere tenuto in considerazione nella scelta di p e di q. Si supponga infatti che $(p-1, q-1)$ sia grande e, di conseguenza, che il minimo comune multiplo u di $p-1$ e $q-1$[10] sia piccolo rispetto a $\phi(n)$. Allora, un qualsiasi inverso di e modulo u potrà essere usato come esponente di decifratura (al posto di d; questo fatto deriva dal Teorema di Eulero che abbiamo visto); dato che u è relativamente piccolo, un tale inverso di e potrebbe essere trovato per tentativi. Pertanto, è meglio che $p-1$ e $q-1$ non abbiano alcun fattore comune grande.

Si potrebbe pensare di adottare RSA scegliendo un valore di n comune per tutti gli utenti di un certo gruppo, mentre i valori di e e di d sarebbero diversi per ciascuno. Purtroppo, questa scelta rende RSA debole: infatti, supponiamo che un utente debba mandare lo stesso testo in chiaro m a due (o più) utenti diversi. Dette e_1 ed e_2 le chiavi pubbliche di tali utenti, il mittente del messaggio trasmette i valori:

$$c_1 = m^{e_1} \mod n$$
$$c_2 = m^{e_2} \mod n$$

[10] Che, ricordiamo, si ottiene come:

$$\text{mcm}(p-1, q-1) = \frac{(p-1)\cdot(q-1)}{\text{MCD}(p-1, q-1)} = \frac{\phi(n)}{(p-1, q-1)}$$

Se e_1 ed e_2 sono coprimi (ovvero MCD$(e_1, e_2) = 1$) allora chiunque può ricostruire m a partire da c_1 e c_2: utilizzando l'algoritmo esteso di Euclide, è possibile determinare due interi r ed s tali che:

$$re_1 + se_2 = 1$$

Nell'ipotesi che r sia negativo (r e s sono sempre uno positivo e l'altro negativo) è possibile calcolare c_1^{-1}, ovvero la soluzione della seguente equazione Diofantea:

$$c_1 x \equiv 1 \mod n$$

Allora, è facile verificare che valgono le seguenti uguaglianze:

$$(c_1^{-1})^{-r} c_2^s = c_1^r c_2^s = m^{re_1} m^{se_2} = m^{re_1 + se_2} = m \mod n$$

Anche la scelta della stessa chiave pubblica e di moduli diversi rende debole RSA. Supponiamo infatti che tre utenti A, B e C scelgano lo stesso valore di e, ad esempio $e = 3$, e tre valori distinti n_A, n_B ed n_C per i moduli. Inoltre, supponiamo che n_A, n_B ed n_C siano a due a due coprimi. Se un quarto utente vuole mandare lo stesso messaggio m ad A, B, C, dovrà inviare:

$$\begin{aligned} c_A &= m^3 \mod n_A \\ c_B &= m^3 \mod n_B \\ c_C &= m^3 \mod n_C \end{aligned}$$

Chiunque intercetti i 3 messaggi inviati è in grado di risalire ad m: usando il *Teorema Cinese del Resto* è infatti in grado di calcolare una soluzione al seguente sistema di equazioni Diofantee:

$$\begin{cases} x \equiv c_A \mod n_A \\ x \equiv c_B \mod n_B \\ x \equiv c_C \mod n_C \end{cases}$$

Come è noto, la soluzione di tale sistema è unica a meno di multipli di $N = n_A n_B n_C$; pertanto, esiste un'unica soluzione $\overline{x}$ compresa tra 0 ed $N - 1$. Poiché m^3 è minore di N (è minore, contemporaneamente, di n_A, di n_B e di n_C), e inoltre soddisfa le tre equazioni del sistema, vale $\overline{x} = m^3$. Pertanto, è sufficiente calcolare la radice cubica intera di $\overline{x}$ per ricavare m.

Concludiamo il discorso su RSA enunciando il Teorema 4.1 di pag. 143 del libro di testo [1].

Teorema 9.1. *Un algoritmo per computare d può essere convertito in un algoritmo probabilistico per fattorizzare n.* ∎

Questo teorema — per la cui discussione e dimostrazione si rimanda al libro di testo — dà un'indicazione della difficoltà presunta di ricavare la chiave segreta d a partire da quella pubblica (n, e): se fossimo in grado di risolvere questo problema, allora saremmo anche in grado di esibire un algoritmo efficiente per la fattorizzazione. Il precedente teorema, insieme al fatto che il problema della fattorizzazione viene ritenuto intrattabile, giustifica il fatto che RSA sia in generale ritenuto un crittosistema sicuro. Gli attacchi che gli sono stati sferrati fin dalla sua nascita sono molti, ma tutti funzionano solo in particolari condizioni (come quelli che abbiamo visto) e sono quindi facilmente evitabili scegliendo i parametri (p, q, e, d, n) in maniera opportuna.

9.1 RSA randomizzato

Con RSA ci sono due messaggi molto importanti che sono sempre facili da decodificare. Supponiamo che Alice e Bob comunichino utilizzando RSA, e che molto spesso Alice debba inviare a Bob un singolo bit confidenziale: $b \in \{0, 1\}$. Alice dovrebbe cifrare questo bit come se fosse un messaggio normale, cioè calcolando $b^e \bmod n$? Ovviamente no: dato che $b^e = b$ per $b \in \{0, 1\}$, il testo cifrato sarebbe identico al testo in chiaro! Vediamo allora cosa potrebbe fare Alice per superare questo inconveniente. Potrebbe generare un intero x a caso tale che $x < \frac{n}{2}$, e trasmettere a Bob $y = (2x+b)^e \bmod n$. Ricevuto y, Bob userebbe la propria chiave privata per recuperare $2x + b$; a questo punto, b sarebbe il bit meno significativo di tale numero intero.

Naturalmente ci chiediamo se questo metodo di scambiarsi bit cifrati è sicuro. Chiaramente può essere al più sicuro quanto lo è RSA, dato che se un avversario potesse recuperare $2x + b$ a partire da y allora potrebbe immediatamente determinare il valore di b. Ma magari è possibile recuperare b a partire da y e da e, senza necessariamente riuscire a recuperare tutto il testo in chiaro $2x + b$. A questo proposito, si può dimostrare che il metodo esposto per cifrare un bit nel bit meno significativo di un intero è sicuro *esattamente* quanto RSA; in altre parole, ogni metodo che sia in grado di indovinare il valore di b da $(2x+b)^e \bmod n$ e da e con probabilità di successo significativamente maggiore di $\frac{1}{2}$ può essere utilizzato per rompere il crittosistema RSA. Questo fatto importante apre una possibilità interessante: dopo tutto, ogni messaggio non è altro che una sequenza di bit. Alice potrebbe allora inviare un messaggio qualsiasi a Bob suddividendolo in bit e trasmettendo un bit alla volta, scegliendo ogni volta un x a caso tra 0 e $\lfloor \frac{n}{2} \rfloor - 1$ in maniera indipendente. Il *crittosistema randomizzato* che ne risulta è chiaramente molto sicuro: l'analisi del testo cifrato risulta essere molto più difficile di quanto avviene con l'RSA tradizionale. D'altra parte, dobbiamo inviare un blocco cifrato per ogni bit del messaggio in chiaro: questo significa che se il messaggio da trasmettere è molto lungo sarà necessaria una velocità di trasmissione molto elevata oppure un intervallo di tempo molto lungo.

9.2 Attacco ai protocolli

Osserviamo infine che non è sufficiente avere un crittosistema sicuro (o ritenuto tale), in quanto esistono attacchi sferrati *contro il protocollo di comunicazione*. Di tali attacchi abbiamo parlato brevemente all'inizio di questo corso, quando abbiamo detto che se Eve ha la possibilità di leggere e anche di scrivere sul canale allora potrebbe ingannare una delle due entità comunicanti facendo finta di essere l'altra (attacco attivo). In realtà, Eve deve avere anche la capacità di *rimuovere* i messaggi in transito sul canale (operazione in genere più difficile della semplice lettura o scrittura sul canale, e che a volte non è possibile effettuare — si pensi ad esempio a messaggi inviati con segnali radio). Supponendo che Eve sia in grado di fare tutto ciò, vediamo un esempio di attacco di tipo attivo; senza entrare nei dettagli di un particolare crittosistema, è sufficiente ragionare in modo astratto sui due protocolli a chiave pubblica che abbiamo visto, utilizzando le cassette di sicurezza con i lucchetti. Supponiamo allora che Alice voglia mandare un messaggio a Bob; la prima cosa che fa è chiedere a Bob (attraverso il canale) qual è la sua chiave pubblica.[11] Bob invia ad Alice la propria chiave pubblica, ma questa viene intercettata da Eve e sostituita con la sua chiave pubblica. Alice riceve una chiave, e pensa che sia quella di Bob (mentre invece è quella di Eve); la usa per cifrare il messaggio, e spedisce il testo cifrato a Bob. Il testo cifrato viene intercettato da Eve; essendo l'unica in grado di decifrarlo, lo decifra, lo legge (memorizza, altera, ecc.), lo cifra usando la chiave pubblica di Bob e lo invia a Bob. Questi riceve il testo cifrato inviato da Eve, credendo che gli sia stato spedito da Alice, lo decifra e lo legge. Questo tipo di attacchi vengono solitamente chiamati *man-in-the-middle attacks*.

Attacchi più sofisticati si rendono necessari quando vengono adottati protocolli più complessi; in tali attacchi, solitamente è necessario che Eve faccia finta di essere Bob quando comunica con Alice, e che faccia finta di essere Alice quando comunica con Bob. Tale tipo di attacchi possono essere prevenuti in almeno tre modi:

- utilizzando le cosiddette *firme digitali*: ogni messaggio inviato deve essere firmato dal mittente in modo tale che il destinatario possa verificare l'identità di chi glielo ha spedito; esistono diversi schemi di firma digitale che assicurano la non duplicabilità della firma (che non è sempre la stessa, ma dipende dal messaggio firmato) e la non falsificabilità;

- utilizzando le cosiddette *dimostrazioni interattive zero–knowledge*: chi spedisce un messaggio *dimostra* a chi lo desideri che è stato lui/lei a spedirlo, oppure di conoscere il testo in chiaro, senza però rivelarlo (e senza rilasciare informazioni che consentano a Eve di ricostruire la dimostrazione). Senza entrare nei dettagli, osserviamo solamente che

[11] Supponiamo che Bob non abbia pubblicato la propria chiave pubblica in una pagina web, altrimenti le cose per Eve si complicano.

questo tipo di protocolli risulta essere molto spesso inefficiente rispetto all'uso del canale di comunicazione, in quanto richiede lo scambio di un elevato numero di messaggi tra Alice e Bob;

- utilizzando un'entità di cui si fidano tutti (Trusted Third Party) che custodisce tutte le coppie chiave-pubblica/identità degli utenti del sistema. È evidente che risulta molto difficile trovare un'entità di cui veramente tutti si fidano, soprattutto se tale entità è un computer collegato ad Internet (con tutti i bachi di sicurezza del software che è normale trovare in tutti i computer); sembrerebbe quindi la soluzione più improbabile, e invece viene presa molto spesso in considerazione. Per alleviare il carico di lavoro a cui dovrebbe sottostare tale entità, solitamente quest'ultima *delega* altre entità a fornire le chiavi pubbliche o a verificare le identità degli utenti. Queste entità possono a loro volta delegarne altre, e così via; la struttura ad albero che si ottiene viene chiamata *Public Key Infrastructure* (PKI), e si occupa anche di gestire le cosiddette *revocation lists*, cioè gli elenchi di chiavi pubbliche che non sono più valide (o perché si voleva che dopo un certo tempo scadessero, o magari perché le corrispondenti chiavi private sono state compromesse o rubate).

In pratica, gli attacchi nei quali Eve fa finta di essere qualcun altro possono essere evitati inserendo nel protocollo un meccanismo di *autenticazione*, che consenta di stabilire con ragionevole sicurezza l'identità di chi invia un messaggio.

Un altro tipo di attacco dinamico prevede che Eve *modifichi* i messaggi in transito sul canale, senza sostituirli. Supponiamo ad esempio di avere una smart card (ovvero una carta tipo Bancomat, dotata di un proprio microprocessore, memoria, sistema operativo, ecc.) utilizzabile come denaro elettronico; ogni tanto, vado ad uno sportello Bancomat a caricare 100 euro, e poi uso la carta nei negozi al posto del denaro contante. Supponiamo di aver costruito un dispositivo che si interfaccia sia allo sportello che alla carta, e che è quindi in grado di leggere e scrivere sul canale di comunicazione. Quando lo sportello invia il messaggio alla carta con le istruzioni di aggiungere 100 euro al credito residuo della carta, il dispositivo (Eve) cambia qualche bit del messaggio in modo tale che i 100 euro diventino, diciamo, 200 euro. Incredibilmente, questo attacco potrebbe funzionare anche se non sono in grado di decifrare il messaggio: ad esempio, so che modificando il 5^o bit i 100 euro diventano 200 euro, e questo mi basta. Anche contro questo tipo di attacchi sono stati (naturalmente!) studiati dei protocolli sicuri, che vengono detti *non malleabili*; in tali protocolli, la probabilità di ottenere un messaggio valido modificando parti del testo cifrato è veramente molto bassa (e quindi, la probabilità di essere scoperti a modificare i messaggi cifrati è molto alta).

10 Dimostrazioni Zero-Knowledge

Le dimostrazioni Zero-Knowledge (ZK) costituiscono un'interessante estensione del concetto di dimostrazione classica. Fin dai tempi di Aristotele, le dimostrazioni sono una sequenza finita di deduzioni logiche che consentono, a partire da alcune *ipotesi*, di mostrare la validità di una proposizione, detta *tesi*. Pertanto, se Alice vuole convincere Bob che un certo teorema è vero, può ad esempio scrivere su un foglio i passaggi logici che portano dalle ipotesi del teorema alla tesi dello stesso; se tali passaggi sono corretti, e se Bob ha un background culturale che gli consente di capirli, Bob non potrà far altro che riconoscere la validità del teorema. Resta implicito il fatto che Bob, una volta vista la dimostrazione, è in grado di ripeterla davanti ad una terza persona per convincerla della validità del teorema (e magari del fatto di essere stato lui il primo a dimostrarlo!).

Nelle dimostrazioni Zero-Knowledge, Alice è in grado di convincere Bob che conosce una certa informazione, *senza però rivelarla*. La dimostrazione assume la forma di un *protocollo* che viene eseguito da Alice e Bob; al termine dell'esecuzione del protocollo Bob è convinto che Alice conosce l'informazione che dice di conoscere, ma non è in grado di eseguire il protocollo con nessun altro per convincerlo della stessa cosa. A prima vista si direbbe che non è possibile convincere un'altra persona di sapere qualche cosa senza rivelarla; consideriamo pertanto il seguente esempio. Supponiamo che esista una grotta avente la forma illustrata in Figura 3. Nella grotta esiste una porta magica, che si apre solamente

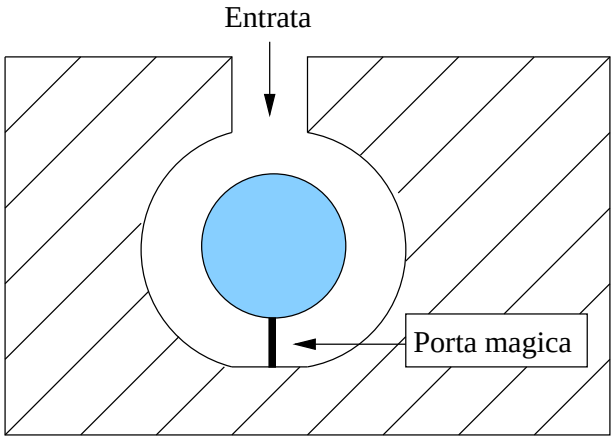


Figura 3: La grotta di Alì Babà

se si dice la parola giusta. Alì Babà vuole convincere i 40 ladroni del fatto che egli conosce la parola magica. Nel caso di una dimostrazione classica, Alì Babà e i 40 ladroni si metterebbero davanti alla porta, Alì Babà pronuncerebbe la parola magica e, aprendo la porta, convincerebbe i 40 ladroni. Lo svantaggio

di questo modo di procedere è che da quel momento in poi anche i 40 ladroni conoscerebbero la parola magica; è facile immaginare la situazione in cui la porta magica dà accesso ad una stanza contenente un tesoro: se i 40 ladroni conoscono la parola magica, non hanno più bisogno di Alì Babà per raggiungere il tesoro...

Nel caso di una dimostrazione Zero-Knowledge, Alì Babà non ha bisogno di rivelare la parola magica ai 40 ladroni per dimostrare loro di conoscerla. Può procedere come segue: entra nella grotta prendendo una delle due direzioni, mentre i 40 ladroni attendono all'ingresso della grotta; se riesce ad uscire dall'altra parte, vuol dire che conosce la parola magica. Questo metodo si estende facilmente al caso in cui la porta dia accesso alla stanza del tesoro: Alì Babà dimostra di conoscere la parola magica (senza rivelarla) se porta fuori qualcosa (ad esempio, qualche moneta) che non aveva quando è entrato.

Quelli esposti sono due esempi di *dimostrazione Zero-Knowledge non interattiva* (i 40 ladroni non partecipano attivamente al protocollo) *di conoscenza*. Facciamo ora qualche passo indietro, e vediamo queste cose da un punto di vista più formale.

10.1 Introduzione ai sistemi di dimostrazione interattivi

Come abbiamo già detto, consideriamo una dimostrazione come un protocollo eseguito da due partecipanti: un "dimostratore", che produce una dimostrazione della proposizione considerata, e un "verificatore" che verifica la validità della dimostrazione. Si possono ottenere diverse nozioni di dimostrazione variando il grado di interazione consentita tra il dimostratore e il verificatore. Nelle dimostrazioni classiche, come quelle che si trovano solitamente sui libri di matematica, l'interazione è costituita da un unico messaggio (la dimostrazione) che va dal dimostratore al verificatore. Come vedremo, consentendo al dimostratore e al verificatore di comunicare in maniera interattiva si possono ottenere concetti di dimostrazione "più potenti".

In particolare, in questi appunti saremo interessati alla *complessità computazionale* delle dimostrazioni; ovvero, siamo interessati a quantificare le risorse necessarie a "produrre" e a "verificare" una dimostrazione. Tipicamente, le proposizioni di cui considereremo le dimostrazioni avranno la seguente forma:

$$\textit{la stringa } x \textit{ appartiene al linguaggio } L$$

(cioè, considereremo le cosiddette *dimostrazioni di membership*), e saremo interessati a determinare la quantità necessaria di risorse per certi linguaggi al crescere della lunghezza di x, cioè del numero di bit necessari per rappresentare x.

Procederemo anzitutto presentando la ben nota classe NP di linguaggi riconosciuti in tempo polinomiale da macchine di Turing non deterministiche come la classe dei linguaggi che ammettono dimostrazioni monodirezionali *efficienti*. Presenteremo quindi le nozioni di *sistema di dimostrazione interattivo* e di *sistema di dimostrazione Zero-Knowledge*. Mostreremo la potenza delle

dimostrazioni interattive dando un sistema di dimostrazione interattivo per un linguaggio che non si sa neppure se appartiene o no alla classe NP, e daremo un esempio di dimostrazione interattiva Zero-Knowledge per il linguaggio costituito da coppie di grafi isomorfi.

10.2 NP come sistema di dimostrazione

Come abbiamo accennato, la classe di linguaggi NP cattura il concetto di sistema di dimostrazione efficiente. È possibile dare diverse definizioni equivalenti della classe NP: ad esempio, durante il corso di Informatica Teorica NP viene definita come la classe dei problemi di decisione che sono risolubili in tempo polinomiale da macchine di Turing non deterministiche; una definizione alternativa è la seguente: NP è la classe dei problemi di decisione le cui soluzioni (dette *testimoni*) possono essere verificate in tempo polinomiale da una macchina di Turing deterministica. Un'altra definizione equivalente e adatta ai nostri scopi è la seguente.

Definizione 10.1 (Classe NP). Un linguaggio $L \subseteq \{0,1\}^*$ appartiene alla classe NP se esiste una costante $c > 0$ e un algoritmo V eseguibile da una macchina di Turing deterministica in tempo polinomiale tale che per tutti gli $x \in \{0,1\}^*$ *sufficientemente lunghi* valgono le seguenti condizioni:

- **completezza**: se $x \in L$ allora esiste $w_x \in \{0,1\}^{|x|^c}$ tale che

$$V(x, w_x) = \text{ACCETTA}$$

- **correttezza**: se $x \notin L$, allora $\forall\, w \in \{0,1\}^{|x|^c}$ vale

$$V(x, w) = \text{RIFIUTA}$$

■

Intuitivamente, la condizione di completezza dice che se la proposizione da dimostrare è vera (cioè, se la stringa x appartiene effettivamente al linguaggio L) allora esiste una dimostrazione w_x di lunghezza al più polinomiale rispetto alla lunghezza di x che può essere verificata in maniera efficiente (cioè in tempo polinomiale, eseguendo un opportuno algoritmo V su una macchina di Turing deterministica). D'altra parte, se la proposizione che si vuole dimostrare è falsa (cioè, se la stringa x *non* appartiene al linguaggio L) allora non esiste *nessuna* dimostrazione w di lunghezza al più polinomiale rispetto alla lunghezza di x che possa "convincere" V.

Osserviamo che la nozione di NP richiede che il procedimento di verifica sia efficiente; in particolare, si richiede che la stringa w_x sia "corta" (di lunghezza al più polinomiale rispetto alla lunghezza di x) e che il processo di verifica possa essere eseguito da una macchina di Turing deterministica in maniera efficiente

(in tempo al più polinomiale rispetto alla lunghezza di x). Osserviamo anche che nulla viene detto su come la stringa w_x viene prodotta e su quante risorse computazionali (tempo e spazio) sono richieste per produrla; possiamo pensare che w_x venga prodotta da una macchina di Turing sulla quale non viene imposta alcuna restrizione. Possiamo allora riformulare la definizione di NP nel modo seguente.

Definizione 10.2 (Classe NP — una definizione equivalente). Un linguaggio $L \subseteq \{0,1\}^*$ appartiene alla classe NP se esistono una costante $c > 0$, una macchina di Turing P che su input di lunghezza n produce output di lunghezza n^c, e un algoritmo V eseguibile in tempo polinomiale su una macchina di Turing deterministica tali che, per tutte le stringhe $x \in \{0,1\}^*$ sufficientemente lunghe, valgono le seguenti proprietà:

- **completezza**: se $x \in L$ allora
$$V(x, P(x)) = \text{ACCETTA}$$
- **correttezza**: se $x \notin L$ allora per tutte le macchine di Turing P' vale
$$V(x, P'(x)) = \text{RIFIUTA}$$

■

10.3 Due semplici esempi

Ricordiamo che un *grafo* (orientato) è una coppia (U, E) di insiemi tali che $E \subseteq U \times U$. Per i nostri scopi consideriamo solamente grafi *finiti*, ovvero coppie (U, E) di insiemi finiti. U viene detto comunemente *insieme dei nodi* o *insieme dei vertici* del grafo, mentre E — che è costituito da coppie (u, v) di nodi — è comunemente detto *insieme degli archi* o *insieme dei lati* del grafo.

Due grafi $G_0 = (U_0, E_0)$ e $G_1 = (U_1, E_1)$ sono *isomorfi* se esiste una funzione biunivoca $f : U_0 \to U_1$ che *preserva i lati*, cioè tale che per tutti gli u e v appartenenti a U_0 vale:

$$(u, v) \in E_0 \iff \big(f(u), f(v)\big) \in E_1$$

Poiché U_0 e U_1 sono insiemi finiti, può esistere una funzione biunivoca $f : U_0 \to U_1$ solo se U_0 e U_1 hanno la stessa cardinalità. Allora, a meno di una ridenominazione dei nodi possiamo porre $U_0 = U_1 = U$, e dare la seguente definizione equivalente di grafi isomorfi.

Definizione 10.3 (Grafi isomorfi). Due grafi $G_0 = (U, E_0)$ e $G_1 = (U, E_1)$ definiti sullo stesso insieme U di nodi sono *isomorfi* se esiste una permutazione π di U (cioè una funzione biunivoca $\pi : U \to U$) tale che, per tutti gli u e v appartenenti a U vale:

$$(u, v) \in E_0 \iff \big(\pi(u), \pi(v)\big) \in E_1$$

■

Se G_0 e G_1 sono due grafi isomorfi e π è un isomorfismo tra G_0 e G_1 allora scriveremo semplicemente $G_1 = \pi(G_0)$.

Fissato un insieme U_n di n nodi, al variare di E in $\mathcal{P}(U_n \times U_n)$ (l'insieme delle parti di $U_n \times U_n$, ovvero l'insieme di tutti i possibili sottoinsiemi di $U_n \times U_n$, compresi l'insieme vuoto ϕ e l'insieme $U_n \times U_n$ stesso) otteniamo una famiglia $\mathcal{G}_n$ di grafi, tutti definiti sullo stesso insieme di nodi. Sia allora $\mathcal{G} = \bigcup_{n\in\mathbb{N}} \mathcal{G}_n$, e sia ISO $\subseteq \mathcal{G} \times \mathcal{G}$ l'insieme (il linguaggio[12]) costituito dalle coppie di grafi isomorfi. È facile vedere che ISO $\in$ NP; infatti, dati due grafi isomorfi G_0 e G_1, al dimostratore P basterà produrre una stringa che codifica l'isomorfismo π tale che $G_1 = \pi(G_0)$. Supponendo che G_0 e G_1 siano definiti su U_n, π risulta essere una permutazione degli elementi di U_n e quindi ammette una rappresentazione di lunghezza polinomiale rispetto ad n (è sufficiente elencare gli elementi di U_n nell'ordine desiderato). Data la codifica di π, è facile verificare che sia $G_1 = \pi(G_0)$: è sufficiente infatti considerare tutti i lati di G_0 e vedere se vengono mappati da π in lati di G_1. Poichè i lati di G_0 sono al più n^2, la verifica può essere fatta in tempo polinomiale da una macchina di Turing deterministica. D'altra parte, se G_0 e G_1 *non sono* isomorfi, risulta impossibile per P produrre una funzione biunivoca $\pi : U_n \to U_n$ tale che $G_1 = \pi(G_0)$, e quindi risulta impossibile indurre il verificatore ad accettare come isomorfi due grafi che non lo sono.

Se invece consideriamo l'insieme (ovvero il linguaggio) NONISO delle coppie di grafi che *non sono* isomorfi, le definizioni di cui sopra non consentono di dire se NONISO appartiene oppure no ad NP. Infatti, dati due grafi $G_0 = (U_n, E_0)$ e $G_1 = (U_n, E_1)$ definiti sullo stesso insieme di nodi tali che $(G_0, G_1) \in$ NONISO, il dimostratore dovrebbe far vedere che *nessuna* permutazione π di U_n è un isomorfismo tra G_0 e G_1. In altre parole, dovrebbe elencare tutte le permutazioni π di U_n; ma queste sono $n!$, e risulta impossibile codificarle in una stringa di lunghezza al più polinomiale rispetto ad n. Questo ovviamente non basta per concludere che NONISO $\notin$ NP: abbiamo solo fatto vedere che non è possibile concludere che NONISO appartiene ad NP utilizzando la definizione di NP in termini di sistema di dimostrazione. In effetti, a tutt'oggi nessuno è riuscito a determinare se NONISO appartiene ad NP oppure no.

10.4 Sistemi di dimostrazione interattivi

La nozione di *sistema di dimostrazione interattivo* viene ottenuta considerando due estensioni naturali della nozione di sistema di dimostrazione efficiente catturata dalla classe NP.

La prima estensione riguarda la natura dell'interazione tra il dimostratore e il verificatore. Nella definizione di NP come sistema di dimostrazione, l'interazione tra il dimostratore e il verificatore è molto ristretta: si ha un singolo messaggio

[12]ISO può essere considerato un linguaggio contenuto in $\{0,1\}^*$ codificando in maniera opportuna i nodi e i lati dei grafi tramite stringhe binarie.

w_x che va dal dimostratore al verificatore. Viene spontaneo chiedersi se il sistema di dimostrazione diventi più potente consentendo al dimostratore e al verificatore di interagire in maniera più elaborata.

Una seconda estensione consiste nel supporre che V sia un algoritmo eseguibile in tempo polinomiale da una macchina di Turing *probabilistica* anziché deterministica. Questa è un'ipotesi molto naturale se consideriamo le computazioni effettuate dalle macchine di Turing probabilistiche una nozione soddisfacente di computazione efficiente. Come conseguenza dell'introduzione della casualità otteniamo un importante rilassamento della nozione di dimostrazione: consentiamo una probabilità d'errore nelle proprietà di completezza e di correttezza. Questo significa che c'è una probabilità maggiore di zero che il dimostratore non riesca a convincere il verificatore della validità di una proposizione vera, e inoltre c'è anche una probabilità maggiore di zero che il dimostratore riesca a far accettare come vera una proposizione falsa.

Formalmente, un *verificatore* V è un algoritmo eseguibile in tempo polinomiale da una macchina di Turing probabilistica dotata di uno speciale nastro di comunicazione. Un *dimostratore* P è un'arbitraria funzione f che ad ogni sequenza finita $x, q_1, q_2, \ldots$ di stringhe in $\{0,1\}^*$ associa una stringa in $\{0,1\}^*$. La computazione procede come segue. Entrambi P e V prendono in ingresso $x \in \{0,1\}^*$. V computa per un certo tempo e scrive $q_1 \in \{0,1\}^*$ sul proprio nastro di comunicazione. P risponde sostituendo q_1 con $f(x, q_1)$. V computa ancora per un po', sovrascrive $f(x, q_1)$ con q_2, e attende la risposta di P, che sarà $f(x, q_1, q_2)$. Questo processo continua fino a che V termina e accetta o rifiuta x. Possiamo allora dare le seguenti definizioni.

Definizione 10.4 (Sistema di dimostrazione interattivo). Una coppia (P, V), dove P è un dimostratore e V è un verificatore, è un *sistema di dimostrazione interattivo* per il linguaggio L se valgono entrambe le seguenti condizioni:

- **completezza**: se $x \in L$, allora

$$\Pr[V \text{ accetta } x \text{ interagendo con } P] \geq \frac{2}{3}$$

- **correttezza**: se $x \notin L$, allora per tutti i dimostratori P' vale

$$\Pr\left[V \text{ accetta } x \text{ interagendo con } P'\right] \leq \frac{1}{2}$$

■

Definizione 10.5 (Round). Un *round* è una domanda posta da V seguita da una risposta data da P. ■

Definizione 10.6 (IP). Un linguaggio L appartiene alla classe IP se e solo se esiste per esso un sistema di dimostrazione interattivo (P, V). ■

Commentiamo brevemente le definizioni date. Anzitutto osserviamo che V deve poter essere eseguito in tempo polinomiale; questo significa che sia il numero di round che la lunghezza dei messaggi scambiati devono essere al più polinomiali rispetto alla lunghezza dell'input x. Nella definizione di sistema di dimostrazione interattivo abbiamo richiesto che la probabilità di accettare proposizioni vere sia maggiore o uguale di $\frac{2}{3}$, e che la probabilità di accettare una proposizione falsa non superi $\frac{1}{2}$. Questi valori sono *immateriali*, dato che possono essere portati arbitrariamente vicino a 1 e a 0 rispettivamente, ripetendo il protocollo più volte, utilizzando ogni volta dei bit casuali indipendenti per le computazioni delle macchine di Turing probabilistiche. Per chi fosse interessato, il numero di volte che è necessario ripetere il protocollo è dato da un lemma tecnico della Teoria delle Probabilità, noto come *Chernoff's bound*. Inoltre, si può dimostrare che ogni linguaggio che ammette un sistema di dimostrazione interattivo ne ammette anche uno in cui il requisito di completezza è soddisfatto con probabilità 1; ovvero, in tale sistema di dimostrazione il verificatore accetta sempre le proposizioni vere. D'altra parte, si può dimostrare che se un linguaggio L ammette un sistema di dimostrazione interattivo in cui il verificatore non accetta mai le proposizioni false allora $L \in \text{NP}$. In altre parole, tollerare una piccola probabilità d'errore è il prezzo che ci tocca pagare per ottenere una nozione più potente di dimostrazione.

Osserviamo infine che nella definizione di sistema di dimostrazione interattivo non poniamo alcuna limitazione sulla capacità computazionale del dimostratore. Tuttavia, si può dimostrare che ogni linguaggio L che ammette un sistema di dimostrazione interattivo ne ammette anche uno in cui il dimostratore è una macchina di Turing deterministica che utilizza una quantità di spazio al più polinomiale rispetto alla lunghezza dell'input x. Questo significa che solo i linguaggi L appartenenti alla classe PSPACE ammettono sistemi di dimostrazione interattivi, e quindi vale l'inclusione $\text{IP} \subseteq \text{PSPACE}$. Inoltre, si può dimostrare che vale anche l'inclusione opposta, cioè $\text{PSPACE} \subseteq \text{IP}$; ciò significa che ogni linguaggio riconosciuto da una macchina di Turing deterministica utilizzando una quantità di spazio al più polinomiale rispetto alla lunghezza dell'input ammette un sistema di dimostrazione interattivo. Unendo le due inclusioni possiamo concludere che le classi di linguaggi IP e PSPACE coincidono.

10.5 Un sistema di dimostrazione interattivo per il non-isomorfismo di grafi

Diamo ora una prova della potenza dei sistemi di dimostrazione interattivi, descrivendone uno per il linguaggio NONISO. Ricordiamo che non è noto se tale linguaggio appartiene alla classe NP oppure no.

SISTEMA DI DIMOSTRAZIONE INTERATTIVO (P, V) PER NONISO

Input: una coppia di grafi (G_0, G_1)

V.1 Scegli a caso (con probabilità uniforme) un bit $b \in \{0,1\}$ e una permutazione π di U.

Costruisci $H = \pi(G_b)$ e invialo a P.

P.1 Ricevi H da V.

Calcola $a \in \{0,1\}$ tale che H è isomorfo a G_a.

Invia a a V.

V.2 Se $a = b$ allora ACCETTA, altrimenti RIFIUTA.

I passi V.1 e V.2 si riferiscono chiaramente alle istruzioni che vanno eseguite dal verificatore, mentre il passo P.1 contiene le istruzioni che vanno eseguite dal dimostratore.

Il protocollo è molto semplice: dati i due grafi G_0 e G_1, entrambi definiti sullo stesso insieme di vertici U, il verificatore ne sceglie uno a caso, lo trasforma applicando un'isomorfismo π scelto anch'esso a caso, e sfida il dimostratore a dire da quale grafo è partito per ottenere tale versione trasformata; se il dimostratore indovina allora ACCETTA, altrimenti RIFIUTA. Per verificare che il protocollo dato sia un sistema di dimostrazione interattivo per il linguaggio NONISO dobbiamo verificare tre cose:

1. che la parte di protocollo del verificatore sia eseguibile in tempo polinomiale da una macchina di Turing probabilistica;
2. che sia verificata la proprietà di completezza;
3. che sia verificata la proprietà di correttezza.

Il punto 1 è banalmente verificato: per scegliere a caso $b \in \{0,1\}$ basta prendere un bit casuale dalla sequenza messa a disposizione per la macchina di Turing probabilistica, mentre la scelta di π può essere effettuata codificando gli elementi di U come numeri binari e scegliendo a caso tra tali numeri, rimuovendo ogni volta il numero scelto dal pool delle scelte successive. La scelta casuale di un numero codificato in binario può essere fatta semplicemente scegliendo a caso un bit alla volta.

Per il punto 2, supponiamo che $(G_0, G_1) \in$ NONISO. Allora i due grafi *non* sono isomorfi, e H è isomorfo o a G_0 o a G_1 (mentre, chiaramente, non può essere isomorfo ad entrambi, altrimenti G_0 e G_1 sarebbero anch'essi isomorfi, contrariamente alle ipotesi fatte). Questo significa che P è in grado di ricavare il valore corretto di b scelto da V, e quindi nel passo V.2 il verificatore accetta sempre.

Per quanto riguarda il punto 3, supponiamo che $(G_0, G_1) \notin$ NONISO. Allora i due grafi sono isomorfi, e H è isomorfo ad entrambi (la composizione tra due

isomorfismi di grafi è, infatti, un isomorfismo tra grafi). Questo significa che P non può far altro che tirare a indovinare per cercare di azzeccare il valore di b scelto da V, e quindi riesce ad indovinare il valore corretto — e a far accettare a V la coppia (G_0, G_1) come appartenente a NONISO — al più la metà delle volte.

10.6 Sistemi di dimostrazione interattivi Zero–Knowledge

Un sistema di dimostrazione interattivo Zero–Knowledge (ZKIPS, dall'inglese *Zero–Knowledge Interactive Proof System*) è un sistema di dimostrazione nel quale il dimostratore convince il verificatore del fatto che una certa asserzione è valida senza però rilasciare altra "conoscenza". Questo concetto viene formalizzato dicendo che i ZKIPS sono quei sistemi di dimostrazione interattivi nei quali il verificatore è in grado di produrre da solo e in maniera efficiente qualcosa che è "uguale" a ciò che vedrebbe interagendo con il dimostratore. la nozione di *efficienza* viene resa precisa richiedendo che la computazione sia eseguita in tempo *medio* polinomiale (il che, notare bene, è diverso dal richiedere che *tutte* le computazioni possibili siano eseguibili in tempo polinomiale!). Inoltre, tre diverse nozioni di "uguaglianza" danno luogo a tre diverse versioni di ZKIPS: *perfetto*, *statistico* e *computazionale*. I ZKIPS sono abbastanza recenti: sono stati introdotti da Goldwasser, Micali e Rackoff nel 1989.

Più precisamente, consideriamo un sistema di dimostrazione interattivo (P, V) per un linguaggio L. Definiamo $View_V(x)$, ovvero ciò che V vede quando interagisce con P dato l'input x, come lo *spazio di probabilità* che assegna alle coppie $(R; log)$ la probabilità che R è la porzione del nastro casuale di V usata durante l'esecuzione del protocollo e che log è la trascrizione di una conversazione tra P e V per l'input x, dato che V utilizza come bit casuali quelli della stringa R.

Diciamo che (P, V) è un ZKIPS se per ogni verificatore V^* esiste un algoritmo probabilistico efficiente tale che la variabile casuale costituita dal suo output per l'input x è "uguale" a $View_V(x)$. Come abbiamo detto, possiamo assegnare diversi significati alla nozione di uguaglianza tra distribuzioni di probabilità. In particolare, abbiamo una *uguaglianza perfetta* se le due distribuzioni di probabilità assegnano esattamente la stessa probabilità ad ogni possibile stringa x; abbiamo una *uguaglianza statistica* se la distanza statistica tra le due distribuzioni decresce più velocemente dell'inverso di un qualsiasi polinomio al crescere della lunghezza delle stringhe in input; e infine, abbiamo una *uguaglianza computazionale* se ogni algoritmo eseguibile in tempo polinomiale su una macchina di Turing deterministica non riesce a distinguere le due distribuzioni di probabilità. Formalmente, possiamo dare le seguenti definizioni.

Definizione 10.7 (Zero–Knowledge perfetta). Un sistema di dimostrazione interattivo (P, V) per un linguaggio L è un *sistema di dimostrazione con Zero–Knowledge perfetta* (in breve, è un sistema di dimostrazione PZK) se per ogni V^* esiste un algoritmo S_{V^*} (chiamato il *simulatore*) eseguibile in tempo *medio*

polinomiale su una macchina di Turing probabilistica tale che per ogni $x \in L$ gli spazi di probabilità $View_{V^*}(x)$ e $S_{V^*}(x)$ *coincidono*. ∎

Definizione 10.8 (Zero–Knowledge statistica). Un sistema di dimostrazione interattivo (P, V) per un linguaggio L è un *sistema di dimostrazione con Zero–Knowledge statistica* (in breve, è un sistema di dimostrazione SZK) se per ogni V^* esiste un algoritmo S_{V^*} (chiamato il *simulatore*) eseguibile in tempo *medio* polinomiale su una macchina di Turing probabilistica tale che per tutte le costanti c e per tutti gli $x \in L$ sufficientemente lunghi vale:

$$\sum_{\alpha} \left| \mathrm{Prob}_{View_{V^*}(x)}[\alpha] - \mathrm{Prob}_{S_{V^*}(x)}[\alpha] \right| < |x|^{-c}$$

∎

Per definire i sistemi di dimostrazione Zero–Knowledge computazionali introduciamo la nozione di *indistinguibilità polinomiale*.

Definizione 10.9 (Indistinguibilità polinomiale). Sia $L \subseteq \{0,1\}^*$ un linguaggio e $\mathcal{U} = \{U(x) : x \in L\}$ e $\mathcal{V} = \{V(x) : x \in L\}$ due famiglie di variabili casuali che assumono valori in $\{0,1\}^*$. Diciamo che $\mathcal{U}$ e $\mathcal{V}$ sono *polinomialmente indistinguibili su* L se per tutti gli algoritmi D eseguibili in tempo polinomiale su macchine di Turing probabilistiche, per tutte le costanti $c > 0$ e tutti gli $x \in L$ sufficientemente lunghi vale:

$$\left| \Pr\left[a \leftarrow U(x) : D(a) = 1\right] - \Pr\left[a \leftarrow V(x) : D(a) = 1\right] \right| < |x|^{-c}$$

∎

Definizione 10.10 (Zero–Knowledge computazionale). Un sistema di dimostrazione interattivo (P, V) per un linguaggio L è un *sistema di dimostrazione con Zero–Knowledge computazionale* (in breve, è un sistema di dimostrazione CZK) se per ogni V^* esiste un algoritmo S_{V^*} (chiamato il *simulatore*) eseguibile in tempo *medio* polinomiale su una macchina di Turing probabilistica tale che le due famiglie di variabili casuali $\{S_{V^*}(x) : x \in L\}$ e $\{View_{V^*}(x) : x \in L\}$ sono polinomialmente indistinguibili. ∎

La nozione di indistinguibilità polinomiale può essere data anche per macchine di Turing *non uniformi*, dove cioè si suppone di avere una famiglia di macchine — una per ogni lunghezza possibile delle stringhe x di input — anziché una singola macchina per tutte le stringhe di ingresso possibili. Il modello non–uniforme risulta essere più potente di quello uniforme che viene solitamente adoperato, e quindi anche i sistemi di dimostrazione CZK corrispondenti risultano essere più potenti.

Osserviamo che il requisito che i sistemi di dimostrazione interattivi devono soddisfare per essere Zero–Knowledge è molto stringente. Infatti, si richiede che il dimostratore P non rilasci alcuna informazione aggiuntiva *non solo* al

verificatore V "legittimo", cioè a quello che esegue correttamente e onestamente il protocollo, ma anche a *qualunque altro* verificatore V^*. Si può ottenere una nozione più debole di ZK, detta *honest–verifier ZK*, richiedendo che il simulatore esista solo per il verificatore V legittimo.

10.7 Un sistema di dimostrazione interattivo ZK per l'isomorfismo di grafi

In precedenza abbiamo visto che il linguaggio ISO appartiene alla classe NP; in realtà, si può dimostrare che ISO è anche NP–completo. Questo significa che si suppone che il compito di decidere se due grafi G_0 e G_1 definiti sullo stesso insieme di nodi U sono isomorfi non può essere risolto in tempo al più polinomiale da una macchina di Turing deterministica. In caso contrario, infatti, sarebbe P = NP, il che è veramente poco probabile.

Mostriamo ora un semplice sistema di dimostrazione per ISO che è dotato di ZK perfetta (che è, fra quelle che abbiamo visto, la nozione più forte di ZK). Tutte le scelte casuali menzionate d'ora in poi si suppone che vengano fatte con *probabilità uniforme* (cioè, ogni elemento dell'insieme da cui si fa la scelta ha la stessa probabilità di essere selezionato).

SISTEMA DI DIMOSTRAZIONE INTERATTIVO ZK (P, V) PER ISO

Input: una coppia di grafi (G_0, G_1)

P.1 Scegli a caso una permutazione τ di U.
Calcola $H = \tau(G_0)$ e invia H a V.

V.1 Scegli a caso $b \in \{0, 1\}$.
Invia b a P.

P.2 Scegli a caso una permutazione σ di U tale che $H = \sigma(G_b)$.
Invia σ a V.

V.2 Se $H = \sigma(G_b)$ allora ACCETTA, altrimenti RIFIUTA.

Verifichiamo ora che il protocollo proposto soddisfi i requisiti di completezza e di correttezza, e che inoltre sia Zero–Knowledge.

Completezza. Supponiamo che sia $(G_0, G_1) \in$ ISO. Allora i due grafi di input sono isomorfi, e anche il grafo H costruito dal dimostratore al passso P.1 è isomorfo ad entrambi. Qualunque sia il valore di b scelto dal verificatore al passo V.1, il dimostratore riesce in ogni caso a costruire la permutazione σ: infatti, se

$b = 0$ basta porre $\sigma = \tau$; se invece $b = 1$, detta π la permutazione di U tale che $G_1 = \pi(G_0)$, basta porre $\sigma = \tau \circ \pi^{-1}$. Pertanto, al passo V.2, il verificatore accetta sempre.

Correttezza. Supponiamo che $(G_0, G_1) \notin$ ISO. Allora i due grafi di input *non* sono isomorfi, e il grafo H costruito dal dimostratore al passo P.1 è isomorfo a G_0 ma non a G_1. Se al passo V.1 il verificatore sceglie $b = 0$, il dimostratore può porre semplicemente $\sigma = \tau$ e far accettare il verificatore. Se invece il verificatore sceglie $b = 1$, il dimostratore non è in grado di trovare alcuna permutazione σ di U che convinca il verificatore che H e G_1 sono isomorfi (dato che tale permutazione non esiste). Pertanto, il verificatore accetta con probabilità uguale a $\frac{1}{2}$. Come è già stato detto, se si desidera che il verificatore accetti istanze false con probabilità più bassa è sufficiente ripetere più volte il protocollo, effettuando ogni volta le scelte casuali in maniera indipendente; in questo caso, ripetendo k volte il protocollo si ottiene che il verificatore accetta coppie (G_0, G_1) di grafi non isomorfi con probabilità 2^{-k}.

Zero–Knowledge perfetta. Consideriamo il seguente simulatore S_{V^*}:

IL SIMULATORE S_{V^*}

Input: una coppia di grafi $(G_0, G_1) \in$ ISO

1. Scegli a caso una stringa *Random* di lunghezza appropriata e scrivila sul nastro dei bit casuali del verificatore V^*.
2. Scegli a caso $a \in \{0, 1\}$ e una permutazione π di U, e costruisci $H = \pi(G_a)$. Invia H a V^*.
3. Ricevi b da V^*.
   Se $b \notin \{0, 1\}$ allora emetti in **output** $(Random;\ H, b)$ e termina.
   Se $a = b$
   allora emetti in **output** $(Random;\ H, b, \pi)$ e termina
   altrimenti GOTO 1.

Come abbiamo detto, S_{V^*} deve riuscire a produrre da solo e in maniera efficiente (cioè, in tempo al più polinomiale rispetto alla dimensione dell'input) lo spazio di probabilità $View_{V^*}(\cdot)$, ovvero l'insieme di coppie $(R;\ log)$ di variabili casuali, dove R è la porzione del nastro casuale (contenente cioè i bit casuali che vengono letti quando si simula il lancio di una moneta) di V^* usata durante l'esecuzione del protocollo e *log* è la trascrizione di una conversazione (cioè

l'insieme dei messaggi scambiati) tra P e V^* per l'input dato, con la condizione che V^* utilizza come bit casuali proprio quelli di R.

Osserviamo allora anzitutto che al punto 1 il simulatore va a scrivere sul nastro dei bit casuali di V^* la stessa stringa casuale *Random* che emette in output al punto 3. Per quanto riguarda la trascrizione dei messaggi che transitano da P a V^* e viceversa mentre si dimostra che due grafi G_0 e G_1 sono isomorfi, osserviamo anzitutto che l'input di S_{V^*} è sempre costituito da coppie di grafi isomorfi. Inoltre, per riuscire a simulare la conversazione tra P e V^*, il simulatore impersonifica P quando dialoga con V^*. Nel passo 2, S_{V^*} sceglie a caso uno dei due grafi dati in input, applica una permutazione casuale ai nodi e invia il risultato H al verificatore V^*. Quindi, al passo 3, attende che V^* gli comunichi il valore che ha scelto per b. Dato che il simulatore S_{V^*} deve poter interagire con qualsiasi verificatore V^*, anche con quelli che non seguono il protocollo in maniera onesta, la prima cosa che fa è controllare che il valore di b proposto da V^* sia 0 oppure 1. Se non è uno di questi valori, allora si suppone che il protocollo venga bloccato, dato che il dimostratore non saprebbe comunque come andare avanti; pertanto, il simulatore emette la sequenza di messaggi scambiati fino a questo momento (ovvero H seguito da b) e termina l'esecuzione. Se invece il valore di b inviato da V^* è 0 oppure 1 allora osserviamo che, poiché anche a viene scelto a caso nell'insieme $\{0,1\}$, la probabilità che sia $a = b$ è $\frac{1}{2}$. Pertanto, a causa dell'istruzione GOTO 1, il protocollo viene eseguito in media 2 volte; dato che ogni passo del protocollo è eseguibile in tempo polinomiale da una macchina di Turing probabilistica, l'intera esecuzione di S_{V^*} avviene in tempo *medio* polinomiale.

Resta da far vedere che i valori H, b e π prodotti da S_{V^*} e dall'esecuzione del protocollo hanno la stessa distribuzione di probabilità. Poiché siamo sotto l'ipotesi che $(G_0, G_1) \in \text{ISO}$, in entrambi i casi H è un grafo isomorfo sia a G_0 che a G_1, scelto a caso. Il valore di b viene scelto a caso nell'insieme $\{0,1\}$ in entrambi i casi, e poiché π viene emesso da S_{V^*} solo nel caso in cui sia $a = b$, π risulta essere in entrambi i casi un isomorfismo scelto a caso che trasforma il grafo G_b nel grafo H.

Riferimenti bibliografici

[1] Arto Salomaa. *Public-Key Cryptography.* Seconda edizione, Springer, 1996.

[2] A. Menezes, P. van Oorschot, S. Vanstone. *Handbook of Applied Cryptography.* CRC Press Series on Discrete Mathematics and its Applications, 1996.

[3] Bruce Schneier. *Applied Cryptography. Protocols, Algorithms and Source Code in C.* John Wiley & Sons, 1996.

[4] David Kahn. *The Codebreakers: The Comprehensive History of Secret Communication from Ancient Times to the Internet.* Scribner, 1996.

[5] Simon Singh. *Codici & Segreti.* Rizzoli, 1999.

[6] Oded Goldreich. *Modern Cryptography, Probabilistic Proofs and Pseudo–randomness.* Springer, 1999.