Anna Bernasconi, Paolo Ferragina,
Fabrizio Luccio

Elementi di Crittografia

PISA
UNIVERSITY
PRESS

MANUALI

Bernasconi, Anna
Elementi di crittografia / Anna Bernasconi, Paolo Ferragina, Fabrizio Luccio. - Pisa : Pisa university press, 2015. - (Manuali)

005.82 (22.)
I. Ferragina, Paolo II. Luccio, Fabrizio 1. Crittografia <Informatica>

CIP a cura del Sistema bibliotecario dell'Università di Pisa

Società con socio unico Università di Pisa
Capitale Sociale € 20.000,00 i.v. – Partita IVA 02047370503
Sede legale: Lungarno Pacinotti 43/44 – 56126 Pisa
Tel. + 39 050 2212056 Fax + 39 050 2212945
press@unipi.it
www.pisauniversitypress.it

ISBN 978-88-6741-460-4

progetto grafico: Andrea Rosellini

I diritti d'autore di questo volume saranno devoluti all'associazione "Medici Senza Frontiere"

Prefazione

Nato nell'ambito del corso di Crittografia per la laurea in Informatica dell'Università di Pisa, questo testo si innesta su un classico precedente, scritto da Ferragina e Luccio per l'editore Bollati-Boringhieri e da tempo fuori commercio, di cui costituisce un sostanziale rifacimento, nonché un'estensione ai tanti aspetti del campo nati e sviluppatesi negli anni più recenti. Lavoro che ha visto la collaborazione essenziale di Bernasconi cui ora compete l'insegnamento della materia.

Il libro è tagliato su un corso universitario di primo livello che segue corsi matematici e algoritmici di base, quindi è principalmente diretto a chi tali basi possiede. Tuttavia alcuni concetti fondamentali di algebra e di computabilità vi sono ripetuti per quanto necessario alla comprensione del testo da parte di chiunque, dotato di un minimo di basi scientifiche, sia interessato ad approfondire una materia che sta acquisendo un'importanza fondamentale nella vita di oggi: dalle carte bancarie alle transazioni commerciali su Internet, dalle schede dei telefoni cellulari alla posta elettronica certificata, siamo infatti immersi in attività imbevute di crittografia.

Nel testo, oltre alle basi classiche della crittografia e alla sua storia passata, sono discusse la teoria e alcune delle principali applicazioni attuali nel campo, con la massima semplicità consentita da argomenti molto complicati ma senza mai abbandonare il rigore scientifico, come richiesto in un ambito matematico ove ormai si colloca la crittografia. Il testo si conclude con una proiezione nel futuro che potrebbe vedere la nascita di calcolatori quantistici e la conseguente riorganizzazione dei sistemi di sicurezza.

Pisa, Ottobre 2014

A.B, P.F, F.L.

Indice

Capitolo 1

La crittografia: una visione di insieme

Gli studiosi di creature viventi spiegano come la comunicazione in forma complessa sia una caratteristica esclusiva del genere umano. È un'affermazione spiacevole ma innegabile almeno nelle sue manifestazioni tecnologiche più avanzate, poiché con l'avvento dell'informatica la comunicazione ha raggiunto livelli definitivamente preclusi ai gatti e alle alghe, nonché agli esseri umani sprovvisti di una cultura adeguata. La comunicazione è la protagonista di questo volume, in un suo aspetto particolare: utilizzare mezzi che la rendano intellegibile solo ad alcuni indipendentemente dalla cultura degli altri. Esamineremo i meccanismi che consentono di rappresentare l'informazione basandoci sui principi primitivi di rappresentazione e di algoritmo e studieremo sofisticate tecniche matematiche per mascherare i messaggi o tentare di svelarli: entreremo così nel mondo affascinante e complesso della *crittografia*, etimologicamente "scrittura nascosta".

La costruzione di messaggi mascherati ha interessato gli uomini fin dalla nascita della scrittura al punto che questa stessa, inintelligibile alle masse, era spesso patrimonio esclusivo dei saggi e di altri prepotenti. Col passare del tempo l'interesse per il campo si è rafforzato fino ad assumere oggi importanza grandissima con l'accesso sempre più diffuso alle reti di calcolatori ove la riservatezza delle comunicazioni è discutibile e sospetta. I fini ultimi della crittografia non sempre sono nobilissimi, e la parola stessa evoca l'esistenza di due mondi in contrapposizione tra loro: da una parte troviamo persone che vogliono scambiarsi privatamente delle informazioni, dall'altra un nugolo di impiccioni che desiderano ascoltare o intromettersi nelle conversazioni altrui per semplice curiosità, o per legittima investigazione, o peggio per scopi malvagi. Dire chi siano i "buoni" e chi i "cattivi" è però difficile: se una banca sta trasferendo fondi per via elettronica dai conti di clienti benefattori a un'associazione

di carità e un intruso cerca di alterare i messaggi per dirigere tali fondi sul proprio conto, è molto probabile che quest'ultimo sia il cattivo; se invece due narcotrafficanti stanno concordando via rete una consegna di eroina, l'agente dell'Interpol che tenta di interpretarne i messaggi è probabilmente il buono. In sostanza a seconda dei casi i buoni possono essere coloro che conversano segretamente o coloro che cercano di intercettarne la comunicazione.

La cosa interessante è che questi due mondi hanno bisogno l'uno dell'altro e che almeno uno degli attori, e possibilmente tutti, deve essere animato da scopi malvagi. Infatti che motivo ci sarebbe per conversare privatamente se non esistessero gli impiccioni? O perché si dovrebbe origliare sulle comunicazioni altrui se non ci fossero segreti da nascondere? Questa realtà sottintende una sfida continua tra due gruppi: da un lato persone che applicano *metodi di cifratura* alle loro conversazioni per renderle inintelligibili a chiunque desideri intercettarle senza autorizzazione; dall'altro persone che sviluppano *metodi di crittoanalisi* per riportare alla luce le informazioni contenute in quelle conversazioni. Ora se esistesse un metodo di cifratura inattaccabile e facile da utilizzare tutti potrebbero impiegarlo per scambiarsi informazioni segrete e gli intercettatori non avrebbero più ragione d'esistere; invece, per quanto i cifrari inattaccabili esistano, essi richiedono operazioni così complesse da risultare utilizzabili solo in condizioni estreme. In conclusione i cifrari più diffusi nella pratica non sono perfetti ma possono essere dichiarati sicuri in quanto sono rimasti inviolati agli attacchi degli esperti, oppure perché, per violarli, è necessario risolvere alcuni problemi matematici difficilissimi. Questo secondo criterio ci lascia naturalmente più tranquilli circa la segretezza della comunicazione, ma è anche più difficile da applicare e implica solo una impossibilità pratica di forzare il cifrario: difficoltà di risoluzione significa in sostanza che il prezzo da pagare per forzare il cifrario è troppo alto perché valga la pena di sostenerlo, in quanto l'operazione richiederebbe di impiegare giganteschi calcolatori per tempi incredibilmente lunghi.

Si potrebbe a questo punto obiettare che se la soluzione di un problema richiede milioni di anni il problema stesso può in pratica essere considerato irrisolubile. Chi può infatti disporre di sufficiente autorità evocativa da convincere migliaia di generazioni di discendenti ad aspettare la risposta? E se ciò fosse anche possibile, che utilità avrebbe a quel punto la decifrazione del messaggio? Se questo è palesemente vero, dobbiamo d'altro canto rilevare che i problemi cui stiamo facendo riferimento hanno uno status computazionale non ancora caratterizzato completamente, per cui non è noto se la loro risoluzione richieda necessariamente tempi enormi o se tale necessità sia imputabile alla nostra incapacità attuale di individuare metodi più efficienti di risoluzione. Gli intercettatori possono ancora sperare! Questo sarà il motivo conduttore di questo testo, nonché il principio su cui si fonderanno affermazioni

quali "computazionalmente sicuro" o "computazionalmente difficile", contrapposte a "incondizionatamente sicuro" e "incondizionatamente difficile".[1]

1.1 Cifratura, decifrazione e attacchi

La crittografia con i suoi metodi di cifratura, e la crittoanalisi con i suoi metodi di interpretazione, sono indissolubilmente legate tra loro e costituiscono di fatto due aspetti di una stessa scienza che è stata correttamente chiamata *crittologia* (anche se il termine è poco diffuso). Il problema centrale di questa scienza è schematicamente il seguente: un mittente `Mitt` vuole comunicare con un destinatario `Dest` utilizzando un canale di trasmissione *insicuro*, cioè tale che altri possono intercettare i messaggi che vi transitano per conoscerli o alterarli. Per proteggere la comunicazione i due agenti devono adottare un metodo di cifratura che permetta a `Mitt` di spedire un *messaggio* m sotto forma di *crittogramma* c, incomprensibile a un ipotetico crittoanalista X in ascolto sul canale, ma di cui sia facile la decifrazione da parte di `Dest`. Se definiamo con MSG lo "spazio dei messaggi" e con CRT "lo spazio dei crittogrammi", le operazioni di cifratura e decifrazione si definiscono formalmente come segue:

Cifratura del messaggio. Operazione con cui si trasforma un generico messaggio in chiaro m in un crittogramma c applicando una funzione $\mathcal{C} : \text{MSG} \rightarrow \text{CRT}$.

Decifrazione del crittogramma. Operazione che permette di ricavare il messaggio in chiaro m a partire dal crittogramma c applicando una funzione $\mathcal{D} : \text{CRT} \rightarrow \text{MSG}$.

Lo schema di comunicazione è riportato nella figura 1.1. Matematicamente $\mathcal{D}(\mathcal{C}(m)) = m$ e le funzioni $\mathcal{C}$ e $\mathcal{D}$ sono una inversa dell'altra. Il termine "spazio" usato per definire MSG e CRT indica un insieme cui appartengono i messaggi o i crittogrammi, senza richiedere che la funzione $\mathcal{C}$ o $\mathcal{D}$ sia definita sull'intero insieme. Infatti i messaggi effettivamente scambiabili costituiscono in genere un sottoinsieme di MSG che

[1] Con le parole di Ron Rivest, uno dei massimi crittografi contemporanei: "Since we don't know many fundamental facts about computational difficulty (such as whether $P = NP$), we don't yet have the tools to prove unconditionally that cryptosystems are computationally secure. The best that can be done from a theoretical point of view is to prove that a cryptosystem is secure, based on the assumption that some well-defined problem (such as 'factoring large integers') is hard. Such assumptions may turn out to be false (indeed, it may be that $P = NP$), and so such proofs of security are really only reductions; the fundamental proofs are yet to be found." Inoltre egli afferma che quelle riduzioni sono: "...hard to come by, particularly for practical cryptosystems that must be efficient as well as secure. Thus, we often assess security by trial of fire. After a cryptosystem has been tested for a while without breaking it, we begin to place some trust in it. Sometimes we are still surprised."

varia con la particolare applicazione considerata, e i relativi crittogrammi costituiscono in genere un sottoinsieme di CRT. Le funzioni $\mathcal{C}$ e $\mathcal{D}$ sono quindi *parziali* e la $\mathcal{C}$ deve essere *iniettiva*, cioè a messaggi diversi devono corrispondere crittogrammi diversi, altrimenti `Dest` non potrebbe ricostruire univocamente un messaggio da ogni crittogramma.

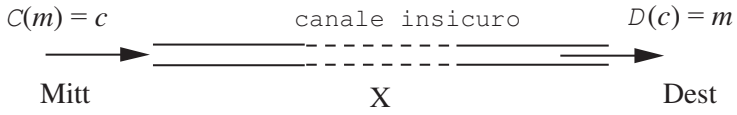


Figura 1.1: Schema di comunicazione crittografica. `Mitt` spedisce a `Dest` un messaggio m sotto forma di crittogramma c, su un canale insicuro su cui è in ascolto un crittoanalista X.

Il crittoanalista X che si inserisce nella comunicazione può essere animato da diversi propositi: scoprire il contenuto della comunicazione; disturbare la comunicazione modificando c in modo che `Dest` non sia in grado di risalire a m; modificare il contenuto del messaggio agendo su c in modo che `Dest` riceva un'informazione diversa da m. In genere si distingue tra comportamento *passivo*, se X si limita ad ascoltare la comunicazione, o *attivo* se X agisce sul canale disturbando la comunicazione o modificando i messaggi, intrusione in genere più pericolosa e difficile da contrastare.

L'attacco a un sistema crittografico ha l'obiettivo di *forzare* il sistema, ma il metodo scelto e il suo livello di pericolosità dipendono dalle informazioni in possesso del crittoanalista. Nel caso semplice di *Cipher Text Attack* il crittoanalista ha rilevato sul canale una serie di crittogrammi $c_1, \ldots, c_r$. Nel *Known Plain-Text Attack*, più pericoloso del precedente, il crittoanalista è venuto a conoscenza di una serie di coppie $(m_1, c_1), \ldots, (m_r, c_r)$ contenenti messaggi in chiaro e loro corrispondenti crittogrammi. Un attacco ancora più grave, e comunque plausibile, è il *Chosen Plain-Text Attack* in cui il crittoanalista si è procurato una serie di coppie $(m_1, c_1), \ldots, (m_r, c_r)$ relative a messaggi in chiaro che lui ha opportunamente scelto. Un modo per compromettere l'intero sistema è infine noto come *Man in-the-middle*, in cui il crittoanalista riesce a installarsi sul il canale di comunicazione interrompendo le comunicazioni dirette tra due utenti e sostituendole con messaggi propri, convincendo ciascun utente che tali messaggi provengano legittimamente dall'altro.

L'attacco al sistema può essere portato a termine con pieno successo se si scopre la funzione $\mathcal{D}$, o con successo più limitato, che può come limite consistere solo nella scoperta di qualche informazione su un particolare messaggio. Un'informazione parziale può comunque essere sufficiente per comprendere il significato del messaggio, in

particolare se si impiega un linguaggio naturale dove la struttura di parole e frasi è obbligata da regole precise che aiutano il crittoanalista e ostacolano la segretezza. Per questo a volte si eseguono alcune operazioni di "maquillage editoriale" sui messaggi in linguaggio naturale come l'eliminazione di spazi, accenti e punteggiatura, o vi si introducono alcune irregolarità inaspettate, per alterare in parte la loro struttura.

Naturalmente questi accorgimenti, pur complicando il lavoro del crittoanalista, sono troppo elementari per fondarvi la sicurezza di un sistema. E d'altra parte se un crittogramma rappresenta una struttura logica diversa da un testo in linguaggio naturale, la decifrazione esatta può risultare essenziale per la sua interpretazione. Se per esempio il messaggio in chiaro contiene un numero, un errore di decifrazione nella cifra più significativa ne altera irrimediabilmente il valore. Come esempio meno ovvio supponiamo che il presidente della commissione dell'esame di crittografia debba comunicare ai commissari l'elenco di esercizi prescelti per la prova, estratti da un libro di testo. L'elenco è rappresentato da una sequenza (o vettore) S di n bit, ove n è pari al numero totale degli esercizi del libro, e si è posto $S[i] = 1$ se l'i-esimo esercizio del libro è stato inserito nella prova, $S[i] = 0$ altrimenti. Il presidente cifra la sequenza S nel crittogramma S' applicando un metodo concordato con i commissari, quindi affida S' a un bidello per la consegna. Uno studente potrebbe probabilmente procurarsi il crittogramma sborsando una piccola mancia, ma dovrebbe poi decifrarlo esattamente per conoscere quali esercizi sono stati scelti (e qui sorge il sospetto che il presidente abbia agito in questo modo per indurre gli esaminandi a tentare la decifrazione, poiché proprio in questo consisteva l'esame). Si noti tra l'altro che se la regola che associa l'elenco degli esercizi d'esame alla sequenza S è nota, questa associazione costituisce semplicemente un processo di *codifica* dell'informazione e la cifratura interviene solo nella trasformazione di S in S'. Se invece la regola non è nota anch'essa fa parte del meccanismo di cifratura.

Si fa risalire al XIII secolo, e in particolare a Ruggero Bacone, la formulazione di tre principi che per secoli hanno caratterizzato la "robustezza" di un cifrario, e che in termini attuali possono esprimersi come:

1. le funzioni $\mathcal{C}$ e $\mathcal{D}$ devono essere *facili* da calcolare;
2. è *impossibile* ricavare la $\mathcal{D}$ se la $\mathcal{C}$ non è nota;
3. il crittogramma $c = \mathcal{C}(m)$ deve apparire *innocente*.

I termini "facile", "impossibile" e "innocente" sono legati all'epoca in cui Bacone li introdusse. In quel tempo la crittografia era basata su metodi completamente manuali, per cui facile significava poter eseguire cifratura e decifrazione con carta e penna in un tempo ragionevole; impossibile voleva dire che il messaggio non era *umanamente*

interpretabile senza conoscere la funzione di decifrazione; innocente significava che il crittogramma non doveva lasciare intendere che si trattasse di un messaggio cifrato, ma doveva esser scambiato per un messaggio in chiaro per non destare sospetti. Al giorno d'oggi, questi principi debbono essere opportunamente reinterpretati. I messaggi cifrati viaggiano su reti informatiche sotto forma di sequenze di bit, dunque tutti a prima vista sono innocenti. Differente è il discorso sui termini facile e impossibile poiché le operazioni crittografiche sono eseguite da un calcolatore. Ciò da un lato ha permesso di progettare metodi di cifratura estremamente sofisticati, impensabili fino ad alcuni decenni or sono; dall'altro ha giocato a favore dei crittoanalisti che possono eseguire velocemente moltissime operazioni sul crittogramma. Ne consegue che oggi anziché facile si dice *computazionalmente facile*, e anziché impossibile si dice *computazionalmente difficile*, in relazione ai tempi richiesti dalle diverse operazioni. Nel capitolo 3 definiremo matematicamente questi termini e vedremo che descrivono casi lontanissimi tra loro.

1.2 Livelli di segretezza

Una possibile classificazione dei metodi crittografici, detti anche brevemente *cifrari*, è legata al grado di segretezza del metodo, che a sua volta dipende dall'uso cui è destinato il cifrario. Vi sono *cifrari per uso ristretto* impiegati principalmente per comunicazioni diplomatiche o militari, in cui le funzioni di cifratura $\mathcal{C}$ e di decifrazione $\mathcal{D}$ sono tenute segrete in ogni loro aspetto. A prima vista questa richiesta potrebbe apparire ovvia per qualsiasi cifrario, ma qualche semplice riflessione mostra come sia impossibile pretendere di osservarla in una crittografia "di massa". Le operazioni eseguite dalle due funzioni sono in genere complesse e non è prudente fidarsi che tutti gli utenti le proteggano con pari scrupolo, o che addirittura qualcuno non le riveli di proposito per scopi illegittimi; dunque in un cifrario utilizzato da molti utenti la parte segreta del metodo deve essere limitata a qualche suo aspetto noto solo alla coppia di utenti che stanno comunicando: è infatti contraddittorio che un'informazione sia allo stesso tempo segreta e completamente nota a un gran numero di persone.

Nascono così i *cifrari per uso generale*, fondati non sulla segretezza delle funzioni $\mathcal{C}$ e $\mathcal{D}$, le quali sono addirittura pubblicamente note, ma sull'uso di una *chiave segreta* k diversa per ogni coppia di utenti. La chiave è inserita come parametro nelle funzioni $\mathcal{C}$ e $\mathcal{D}$, e queste sono progettate in modo che la conoscenza esplicita delle operazioni da esse eseguite, e di un crittogramma c carpito sul canale insicuro, non consenta a un intruso che non conosce la chiave k di estrarre utili informazioni sul messaggio originale m. Utilizzeremo la notazione $\mathcal{C}(m, k)$ per la cifratura e $\mathcal{D}(c, k)$ per la decifrazione.

Naturalmente occorre porre molta attenzione nella scelta della chiave e nell'ope-

razione di scambio della chiave stessa tra `Mitt` e `Dest`, poiché una intercettazione in questa fase pregiudicherebbe irrimediabilmente il sistema. Parrebbe comunque che il problema dello scambio segreto dei messaggi si ripresenti immutato nello scambio segreto della chiave, ma questo è vero solo in parte perché in genere la chiave è molto più corta dei messaggi e può essere scambiata su un canale sicuro anche se costoso (al limite in un incontro diretto tra `Mitt` e `Dest`). Inoltre, una volta fissata, la chiave può essere utilizzata per cifrare numerosi messaggi cosicché il costo del suo scambio viene *ammortizzato* nel corso del suo impiego.

In sostanza tenere segreta la chiave è più semplice che tenere segreto l'intero processo di cifratura e decifrazione, e tutti possono impiegare le funzioni pubbliche $\mathcal{C}$ e $\mathcal{D}$ a patto che scelgano chiavi diverse. Inoltre se un crittoanalista entra in possesso di una chiave occorre soltanto generarne una nuova, il che consente di realizzare e mantenere efficientemente in software o in hardware le funzioni $\mathcal{C}$ e $\mathcal{D}$ che rimangono pertanto inalterate. Per contro se si scoprono le caratteristiche di un cifrario per uso ristretto l'impiego di questo risulta totalmente compromesso. Molti dei cifrari in uso oggi come lo standard *DES* e il nuovo standard *AES* sono a chiave segreta.

Ma se la segretezza dipende unicamente dalla chiave, lo spazio KEY entro cui questa è scelta deve essere molto ampio. Infatti tale spazio è, o potrebbe essere, noto al crittoanalista che per ricostruire il messaggio dovrebbe solo verificare la significatività delle sequenze di caratteri prodotte con il calcolo di $\mathcal{D}(c, k)$, per ogni possibile chiave k. Questo attacco è detto *esauriente*, anche se più spesso è usato il brutto anglicismo *esaustivo*. Per esempio se fosse $|KEY| = 10^{20}$ (cioè se ogni chiave fosse composta da una sequenza arbitraria di venti cifre decimali), e un potente calcolatore impiegasse un milionesimo di secondo per calcolare $\mathcal{D}(c, k)$ e verificarne la significatività, occorrerebbe in media più di un milione di anni per scoprire il messaggio provando tutte le chiavi possibili. È bene tuttavia tener presente che uno spazio delle chiavi molto grande non sempre garantisce la segretezza perché questa, come vedremo, può essere violata con tecniche diverse di crittoanalisi: esistono infatti cifrari più sicuri di altri, pur basandosi su uno spazio di chiavi molto più piccolo.

Tutto questo sarà esaminato nel seguito in termini strettamente matematici, senza dimenticare due regole di folklore che codificano una saggezza antica e sono note a tutti i cultori di crittografia: un cifrario complicato non è necessariamente un cifrario sicuro; e: mai sottovalutare la bravura del crittoanalista.

1.3 La chiave pubblica

L'anno 1976 ha segnato una svolta nella storia della crittografia. Diffie e Hellman, e indipendentemente Merkle, introdussero la *crittografia a chiave pubblica*, con l'obietti-

vo di permettere a tutti di inviare messaggi cifrati, ma di abilitare solo il destinatario a decifrarli. Si trattava di un concetto totalmente nuovo. Nella crittografia a chiave segreta due utenti sono in grado di cifrare qualsiasi messaggio con la chiave segreta k della coppia, e decifrarlo con la stessa chiave. Nella crittografia a chiave pubblica le operazioni di cifratura e decifrazione utilizzano due chiavi diverse $k[pub]$ per cifrare, e $k[prv]$ per decifrare: la prima chiave è *pubblica*, cioè nota a tutti, la seconda è *privata*, cioè nota soltanto al destinatario del messaggio. Come in precedenza le funzioni di cifratura $\mathcal{C}$ e di decifrazione $\mathcal{D}$ sono di pubblico dominio, identiche per tutti gli utenti, e si calcolano inserendovi una chiave come parametro; quindi la cifratura è accessibile a tutti, perché a tutti è nota la relativa chiave, mentre la decifrazione è accessibile solo a chi possiede la chiave privata. Per tale motivo i sistemi a chiave pubblica sono detti *asimmetrici*, mentre quelli a chiave segreta sono detti *simmetrici* per la pari funzionalità attribuita a mittente e destinatario.

Affinché sia possibile costruire un sistema a chiave pubblica la funzione $\mathcal{C}$ deve possedere una proprietà forte detta *one-way trap-door*. Con ciò si intende che il calcolo $c = \mathcal{C}(m, k[pub])$ per la cifratura è facile, ma il calcolo inverso $m = \mathcal{D}(c, k[prv])$ per la decifrazione è difficile (funzione one-way) a meno che non si conosca un meccanismo segreto (trap-door) che in questo caso è rappresentato da $k[prv]$. Ancora una volta i termini facile e difficile si riferiscono alla complessità della computazione. L'organizzazione complessiva è quella di una comunicazione *molti a uno* illustrata schematicamente nella figura 1.2, ove tutti gli utenti possono inviare in modo sicuro messaggi a uno stesso destinatario cifrandoli con la funzione $\mathcal{C}$ e la chiave $k[pub]$ che sono pubbliche, ma solo il destinatario può decifrare tali messaggi.

A questo punto è lecito chiedersi se le funzioni one-way trap-door esistano realmente. In effetti sono note diverse funzioni facili da calcolare, ma la cui inversione è legata alla risoluzione di problemi ritenuti difficilissimi a meno che non si conosca il valore di un parametro che ha il ruolo di $k[prv]$. La difficoltà, come vedremo nel seguito, è però in genere condizionata a congetture matematiche universalmente accettate ma non dimostrate esplicitamente, per cui tali funzioni saranno utilizzabili solo fino al momento, obiettivamente poco probabile, in cui se ne dimostri una semplice invertibilità. Tra i sistemi crittografici a chiave pubblica un posto di primo piano spetta al cifrario *RSA* che avremo modo di discutere diffusamente nel seguito. Si va inoltre sempre più affermando la crittografia su curve ellittiche che esamineremo attentamente e che potrebbe costituire lo standard di un prossimo futuro.

L'introduzione dei cifrari asimmetrici ha avuto alcune importanti conseguenze. In un cifrario simmetrico con n utenti sono necessarie $n(n-1)/2$ chiavi segrete, una per ogni coppia di utenti; questo numero diventa enorme nelle reti ponendo seri problemi per la memorizzazione e soprattutto lo scambio segreto delle chiavi. In un cifrario

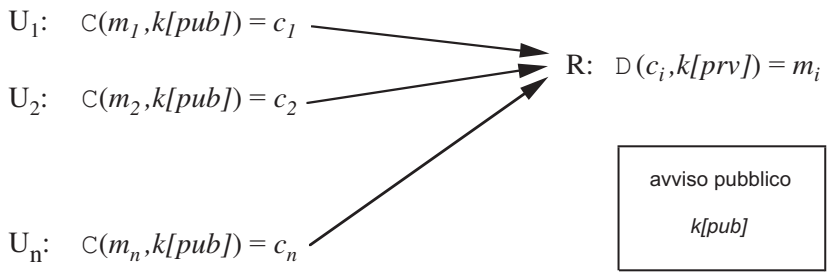


Figura 1.2: Crittografia a chiave pubblica. $k[prv]$ e $k[pub]$ sono le chiavi privata e pubblica dell'utente R. Gli altri utenti $U_1, U_2, \ldots, U_n$ inviano a R i messaggi m_i cifrati come $c_i = \mathcal{C}(m_i, k[pub])$, $1 \leq i \leq n$. R decifra i crittogrammi calcolando $\mathcal{D}(c_i, k[prv])$, mediante la chiave privata $k[prv]$ che solo lui conosce. Un crittoanalista in ascolto sul canale insicuro non può ricavare alcuna informazione sui messaggi m_i pur conoscendo $\mathcal{C}$, $\mathcal{D}$ e $k[pub]$.

asimmetrico sono invece sufficienti n coppie di chiavi pubblica/privata, e scompare il loro scambio segreto perché le chiavi di cifratura sono pubbliche, e ciascuna chiave di decifrazione è in possesso di un solo utente che la mantiene segreta. La crittografia a chiave pubblica è però molto più lenta e non ha soppiantato i cifrari simmetrici, che rimangono a tutt'oggi la scelta *de facto* nelle comunicazioni di massa. In effetti si adotta solitamente un approccio *ibrido* che utilizza i cifrari simmetrici per la cifratura e decifrazione dei messaggi mediante chiavi segrete, e i cifrari asimmetrici per lo scambio di tali chiavi.

1.4 L'approccio algoritmico

Il lettore che ci ha seguito fin qui si sarà forse costruito un'immagine della crittografia come scienza curiosa e piacevole. È bene allora che sappia che gli studi crittografici, elementari quando ci si rivolge al passato, presentano oggi aspetti algoritmici e matematici non semplici. Il fatto che questi studi siano comunque piacevoli dipende dai punti di vista; noi li presenteremo nel modo più semplice di cui saremo capaci, ma senza compromessi sul rigore matematico.

Anzitutto studieremo la rappresentazione matematica di oggetti, che nel caso della crittografia potranno essere messaggi o crittogrammi, funzioni o algoritmi per calcolarle. Poi ci rivolgeremo agli algoritmi ricordandone la struttura di base e alcune

caratteristiche avanzate, e indagando sulle problematiche connesse alla complessità di calcolo. Non è difficile comprendere come questo approccio sia inevitabile. Ogni sistema crittografico è crucialmente fondato sulla rappresentazione di messaggi, e deve basarsi su funzioni di cifratura e decifrazione facili da calcolare e su segreti difficili da svelare: le prime dovranno ammettere algoritmi di calcolo ragionevolmente rapidi, i segreti dovranno essere celati in problemi risolubili solo con algoritmi lentissimi. Anche se la sicurezza assoluta non sarà mai raggiunta, avremo facilitato le comunicazioni legittime e resa difficilissima l'opera del crittoanalista. In questo contesto discuteremo dell'influenza che gli eventi casuali esercitano sullo svolgimento del calcolo, e di come questi eventi possano essere ragionevolmente generati e diretti a nostro (crittografico) vantaggio.

Tutto questo richiede un'introduzione alla scienza algoritmica, oggetto dei prossimi tre capitoli. Chi si sentisse già ferrato su questi argomenti potrà rivolgersi direttamente alla parte propriamente crittografica del libro; o potrà comunque dirigersi subito su tale parte, tornando su concetti e risultati algoritmici quando sarà necessario: abbiamo infatti introdotto riferimenti ai primi tre capitoli ovunque la comprensione del testo lo richieda.

1.5 Applicazioni

Dopo l'immersione nelle tecniche di crittografia, di costruzione dei cifrari e di sicurezza delle comunicazioni, volgiamo ora l'attenzione al mondo delle applicazioni.

La prima osservazione è che la segretezza delle comunicazioni è certamente fondamentale ma non è l'unica caratteristica richiesta ai sistemi crittografici attuali. Vi sono in particolare tre requisiti importantissimi nelle applicazioni su rete che è bene mettere in evidenza. L'*identificazione* dell'utente, attraverso la quale un sistema è messo in grado di accertare l'identità di chi richiede di accedere ai suoi servizi. L'*autenticazione* di un messaggio, attraverso la quale un destinatario `Dest` accerta che il messaggio stesso è stato effettivamente spedito da `Mitt` (deve essere cioè difficile a un intruso spacciarsi per un altro utente). In questa fase `Dest` deve anche poter stabilire che il messaggio non è stato modificato o sostituito nella trasmissione. Infine la *firma digitale*, apposta la quale `Mitt` non può ricusare la paternità di un messaggio spedito a `Dest`, e questi può dimostrare a terzi che proprio ciò è avvenuto.

Stabilite quali sono le funzionalità che ci aspettiamo da un protocollo crittografico, osserveremo che siamo oggi letteralmente circondati da sistemi che utilizzano questi protocolli in modo più o meno completo e esplicito. Studieremo le applicazioni che riguardano i terminali per le carte bancarie e la moneta elettronica, e rivolgeremo grande attenzione alla trasmissione protetta di dati sulla rete Internet, in particolare

al protocollo SSL, perché in questo campo i sistemi crittografici esplicano le loro piene potenzialità.. Questi esempi dimostreranno come i concetti e le tecniche che abbiamo discusso in precedenza entrino in azione in applicazioni reali, come guida alla comprensione di ogni altro protocollo crittografico.

Concluderemo il testo spiegando come fenomeni di meccanica quantistica entrino già oggi in applicazioni crtttografiche, e con una proiezione nel futuro legata alla possibile nascita di calcolatori quantistici nel senso compiuto del termine e la conseguenze che ciò avrebbe sui sistemi di sicurezza.

Capitolo 2

Rappresentazione e calcolabilità

La rappresentazione di un oggetto può assumere forme diverse come un disegno o l'atteggiamento di un mimo, un gruppo di simboli matematici o l'insieme dei suoni che costituiscono una parola. Se l'oggetto è complesso la rappresentazione si sviluppa nel tempo: nel nostro studio essa diviene una sequenza di "segni" distinti da esaminare uno dopo l'altro, secondo una visione che è andata consolidandosi in cento anni di sviluppo della logica matematica prima e dell'informatica poi. È una visione obbligata se si vuole studiare la crittografia come scienza matematica e non come diletto enigmistico (che potrebbe essere assai più divertente e difficile); ed è comunque legata alla miriade di applicazioni pratiche che oggi richiedono la protezione crittografica dei messaggi. Studieremo dunque gli insiemi di segni necessari per la rappresentazione di oggetti e vedremo come questa rappresentazione possa essere costruita, interpretata, elaborata.

2.1 Alfabeti e sequenze

I segni scelti per la rappresentazione matematica di oggetti sono detti *caratteri* o *simboli*, e nel loro complesso costituiscono un *alfabeto*. Assumeremo sempre che l'alfabeto sia prefissato e finito. Un oggetto sarà rappresentato da una sequenza ordinata di caratteri in modo che a oggetti diversi corrispondano sequenze diverse: nella sua accezione più semplice rappresentare equivale ad "assegnare nomi" (le sequenze) ai singoli oggetti. Impiegando sequenze arbitrarie il numero di oggetti che si possono rappresentare non ha limite: ciò non significa che ci proponiamo di trattare infiniti oggetti il che, a parte ogni astrazione matematica, è un compito privo di senso; ma piuttosto scelto un intero arbitrariamente grande potremo sempre costruire un nu-

mero di sequenze maggiore di esso se le sequenze stesse sono sufficientemente lunghe. Questo è il primo punto che dobbiamo considerare.

Poniamo che l'alfabeto Γ contenga n caratteri, in simboli $|\Gamma| = n$, e indichiamo con N il numero di oggetti da rappresentare: n e N sono i parametri del problema. Sia $d(n, N)$ la lunghezza (cioè il numero di caratteri) della più lunga sequenza che rappresenta un oggetto dell'insieme, in funzione dei parametri. Il valore di $d(n, N)$ dipende dal metodo di rappresentazione scelto; indichiamo con $d_{min}(n, N)$ il valore minimo di $d(n, N)$ tra tutte le rappresentazioni possibili e studiamo tale grandezza: un metodo di rappresentazione è tanto migliore quanto più $d(n, N)$ si avvicina a $d_{min}(n, N)$.

Se $n = 1$, cioè Γ contiene un solo carattere che indicheremo convenzionalmente con la cifra 0, le sequenze di rappresentazione non possono che essere ripetizioni dello 0. Al meglio potremo assegnare ai vari oggetti sequenze composte da uno, due,...., N zeri ottenendo $d_{min}(1, N) = N$: questa rappresentazione è detta *unaria* ed è estremamente sfavorevole come ora vedremo.

Poniamo ora che sia $n = 2$ e, convenzionalmente, $\Gamma = \{0, 1\}$: la rappresentazione si dice *binaria*. Per ogni intero $k \geq 1$ le possibili sequenze di lunghezza k costituiscono le disposizioni con ripetizione dei due elementi 0 e 1 presi in gruppi di k, e sono in totale 2^k. Ricordando la relazione $\sum_{i=0}^{k} 2^i = 2^{k+1} - 1$ possiamo concludere che il numero totale di sequenze binarie lunghe da 1 a k è $2^{k+1} - 2$ (è stata esclusa dalla somma la sequenza di lunghezza zero cioè vuota). Per rappresentare N oggetti con queste sequenze deve risultare $2^{k+1} - 2 \geq N$, ovvero $k \geq \log_2(N + 2) - 1$. Poiché $d_{min}(2, N)$ è pari al minimo intero k che soddisfa tale relazione abbiamo: $d_{min}(2, N) = \lceil \log_2(N + 2) - 1 \rceil$, da cui si ottiene immediatamente:

$$\lceil \log_2 N \rceil - 1 \leq d_{min}(2, N) \leq \lceil \log_2 N \rceil. \tag{2.1}$$

Dunque $\lceil \log_2 N \rceil$ caratteri binari sono sufficienti per costruire N sequenze differenti, e in particolare si possono costruire N sequenze differenti *tutte* di $\lceil \log_2 N \rceil$ caratteri. Per esempio si possono costruire le $N = 7$ sequenze: 0, 1, 00, 01, 10, 11, 000 impiegando al massimo $\lceil \log_2 7 \rceil = 3$ caratteri, o le 7 sequenze: 000, 001, 010, 011, 100, 101, 110 impiegando esattamente 3 caratteri. Questa proprietà, come vedremo, è molto importante.

La relazione 2.1 si estende immediatamente a rappresentazioni *n-arie*, per ogni alfabeto $\Gamma = \{0, 1, ..., n-1\}$ con $n > 2$. Analogamente al caso binario, ma con calcoli un po' più complicati, si ottiene la limitazione:

$$\lceil \log_n N \rceil - 1 \leq d_{min}(n, N) \leq \lceil \log_n N \rceil. \tag{2.2}$$

Dunque $\lceil \log_n N \rceil$ caratteri n-ari sono sufficienti per costruire N sequenze differenti, e in particolare si possono costruire N sequenze differenti *tutte* di $\lceil \log_n N \rceil$ caratteri. Per esempio usando l'alfabeto decenario $\Gamma = \{0, 1, ..., 9\}$ si possono costruire $N = 1000$ sequenze di $\lceil \log_{10} 1000 \rceil = 3$ caratteri, dalla 000 alla 999.

Le rappresentazioni che impiegano un numero massimo di caratteri di ordine *logaritmico* nella cardinalità N dell'insieme da rappresentare si dicono *efficienti* e si pongono in alternativa, per esempio, alla non efficiente rappresentazione unaria. A tale scopo però l'alfabeto deve contenere almeno due caratteri. Se si impiegano solo sequenze della stessa lunghezza $d = \lceil \log_n N \rceil$ si ha in ogni caso $d \leq d_{min}(n, N) + 1$ (vedi relazione 2.2): è una piccola penalità in confronto al vantaggio che si ottiene se si devono rappresentare diversi oggetti in successione. Infatti per concatenare sequenze di lunghezza diversa è necessario inserire una marca di separazione tra l'una e l'altra o garantire particolari proprietà sulla loro struttura, onde poter decidere univocamente il punto di separazione tra una sequenza e la successiva. Questi accorgimenti non sono ovviamente necessari se la lunghezza delle sequenze è unica e nota.

Come abbiamo già detto la rappresentazione fin qui descritta consiste nell'assegnare nomi diversi ai diversi oggetti di un insieme, e in linea di principio non reca con sé alcun carattere peculiare degli oggetti stessi: è come assegnare dei titoli astratti ai libri di una biblioteca senza nulla dire del loro contenuto. L'operazione è però importante perché consente di determinare il numero minimo di caratteri necessari a individuare un oggetto tra N, e in alcuni casi conserva tutta l'informazione in essi contenuta. Ciò avviene in particolare se gli oggetti da rappresentare sono numeri: per esempio le mille sequenze di tre cifre decimali ordinate da 000 a 999 rappresentano completamente i mille numeri interi tra 0 e 999. Ciò non avviene invece se le stesse sequenze sono impiegate per rappresentare convenzionalmente mille parole della lingua italiana.

Dalle relazioni 2.1 e 2.2 segue una proprietà che è alla base di molti risultati crittografici. La consueta notazione posizionale per rappresentare i numeri interi è una rappresentazione efficiente indipendentemente dalla base n scelta (in pratica $n = 10$, o $n = 2$ all'interno dei calcolatori), perché un intero N è rappresentato con un numero di cifre d compreso tra $\lceil \log_n N \rceil$ e $\lceil \log_n N \rceil + 1$ (per esempio i numeri 99 e 100 sono rappresentati rispettivamente con $\lceil \log_{10} 99 \rceil = 2$ e $\lceil \log_{10} 100 \rceil + 1 = 3$ cifre decimali). Vi è quindi una riduzione logaritmica tra il *valore* N di un numero e la *lunghezza* d della sua rappresentazione standard.

Nel seguito, studiando la crittografia, ci occuperemo in sostanza di rappresentare oggetti con regole "segrete" in modo che solo chi conosce tali regole sia in grado di ricostruire la corrispondenza tra sequenze trasmesse e oggetti rappresentati da esse. In linea di principio qualunque metodo di rappresentazione potrebbe essere un

metodo crittografico, come lo fu di fatto la scrittura tra i popoli che ne riservavano la conoscenza ad alcune caste. Poiché però la prima fase di rappresentazione consiste in genere nel trascrivere un messaggio secondo un codice standard pubblicamente disponibile (per esempio la rappresentazione binaria ASCII dei caratteri tipografici nelle trasmissioni digitali), la trasformazione crittografica consiste in una successiva elaborazione delle sequenze secondo leggi speciali.

2.2 L'ordinamento canonico

Le proprietà esposte nel paragrafo precedente generano importanti conseguenze. Consideriamo l'insieme di tutte le sequenze generate con un alfabeto arbitrario e cerchiamo di elencarle in un ordine ragionevole. Poiché tali sequenze non sono in numero finito non potremo completare l'elenco, ma ci prefiggiamo lo scopo di raggiungere qualsiasi sequenza σ arbitrariamente scelta in un numero finito di passi: cioè σ dovrà trovarsi a distanza finita dall'inizio dell'elenco.

Il problema è semplice ma non è banale. Consideriamo per esempio le sequenze costruite sull'alfabeto $\Gamma = \{A, B, ..., Z\}$. Se, come in un dizionario, si ponessero all'inizio tutte le sequenze che cominciano per A non si raggiungerebbero mai quelle che iniziano per B poiché le prime sono infinite. Si ricorre quindi a un *ordinamento canonico* costruito come segue:

1. si definisce un ordine tra i caratteri dell'alfabeto (nel nostro esempio si userà l'ordinamento standard tra lettere);

2. si ordinano le sequenze in ordine di lunghezza crescente e, a pari lunghezza, in "ordine alfabetico".

Nel nostro esempio avremo l'elenco:

A, B,. . . , Z,
AA, AB, . . . , AZ, BA, BB, . . . , BZ, ... , ZA, ZB, . . . , ZZ,
AAA, AAB, . . . , AAZ, ... , ZZZ, . . .

Una sequenza arbitraria σ si troverà tra quelle di $|\sigma|$ caratteri,[1] e in posizione alfabetica tra queste.

L'ordinamento canonico è usualmente arricchito della sequenza vuota posta in posizione 0. Si mostra così che le sequenze costruite su un alfabeto arbitrario possono essere poste in corrispondenza biunivoca con i numeri naturali, poiché a una sequenza arbitraria σ corrisponde il numero che indica la posizione di σ nell'elenco, e a un nu-

[1] Le sequenze di lunghezza $|\sigma|$ si trovano nelle posizioni da $\frac{26^{|\sigma|}-1}{25}$ a $\frac{26^{|\sigma|+1}-1}{25} - 1$ dell'elenco.

mero naturale n corrisponde la sequenza in posizione n. In matematica ciò si esprime dicendo che le sequenze sono *infinite* e *numerabili*. Nel prossimo paragrafo studieremo alcune importanti conseguenze matematiche di questo fatto: per ora approfondiamo il problema da un punto di vista meno usuale.

Se ammettiamo che tutto lo scibile possa essere comunicato attraverso un insieme di libri, cioè di sequenze costruite su un opportuno alfabeto come per esempio quello cui si accede dalla tastiera di un calcolatore, dobbiamo concludere che le cognizioni che l'uomo ha accumulato e potrà accumulare nell'eternità sono possibilmente infinite ma numerabili. Queste cognizioni, come annotò Borges in un memorabile saggio, sono contenute in libri già scritti, o in infiniti libri mai scritti in infinite lingue sconosciute ma di cui potenzialmente esistono dizionari e sintassi; nelle critiche a questi libri; nelle critiche alle critiche; in edizioni che differiscono solo per un carattere; o per due caratteri; in sostanza in ogni sequenza concepibile. Né le cose cambiano se i libri contengono delle figure poiché queste possono essere digitalizzate con una finezza maggiore di quanto l'occhio possa percepire e tramutate a loro volta in sequenze. Il ragionamento umano sembra in sostanza confinato in un universo numerabile. E in verità sono numerabili anche i procedimenti di calcolo, cioè gli *algoritmi*, di cui ora cominceremo a interessarci.

2.3 Funzioni, problemi e algoritmi

La numerazione delle sequenze studiata nel paragrafo precedente è resa possibile dal fatto che esse sono di lunghezza finita anche se illimitata (cioè per qualunque intero d scelto a priori esistono sequenze di lunghezza maggiore di d). La teoria degli insiemi insegna però che per sequenze di lunghezza infinita la numerazione non è in genere possibile: i reali, per esempio, sono infinitamente "più numerosi" degli interi, ovvero l'insieme dei reali non è numerabile. Questo fatto riguarda da vicino anche la teoria delle funzioni e degli algoritmi.

Come noto una *funzione* è una corrispondenza tra gli elementi di due insiemi detti *dominio* e *codominio*. Nel mondo reale gli elementi dei due insiemi sono rappresentati da sequenze finite poiché sia i calcolatori che gli esseri umani rappresentano la loro conoscenza mediante caratteri e non possono lavorare su sequenze infinite. Sarà quindi legittimo rappresentare gli elementi dei due insiemi con i numeri naturali corrispondenti al loro ordinamento canonico e, d'ora in avanti, trattare le funzioni come corrispondenze tra numeri naturali.[2] Scopo di questa interpretazione è solo

[2]Questo punto merita una riflessione. La matematica può occuparsi per esempio del numero reale π solo perché può darne una definizione esprimibile in modo finito. La sequenza di definizione può essere considerata "il nome" di π. Ma, come abbiamo visto, esistono più numeri reali che nomi

poter dimostrare con semplicità alcune proprietà generali del calcolo, tornando alla rappresentazione originale quando il calcolo deve essere eseguito.

Seguendo questo schema una funzione può essere assegnata come la sequenza illimitata di elementi del codominio ordinatamente corrispondenti agli elementi $0, 1, 2, \ldots$ del dominio. Per esempio:

$$f : \ 37 \ \ 261 \ \ 2 \ \ 58 \ \ 2 \ \ \ldots \tag{2.3}$$

rappresenta una funzione f tale che $f(0) = 37$, $f(1) = 261$, $f(2) = 2$, $f(3) = 58$, $f(4) = 2$ e così via. Ovviamente qualunque sequenza di questo tipo corrisponde a una funzione, e poiché queste sequenze hanno lunghezza infinita dobbiamo concludere che *le funzioni non sono numerabili*.

Questo famoso risultato può d'altra parte essere dimostrato in modo molto semplice. Se esistesse una numerazione per le funzioni esse potrebbero essere indicate con nomi progressivi f_0, f_1, f_2,..., e vi sarebbe sempre almeno una funzione non inclusa nell'elenco. Infatti detto $f_i(j)$ il valore della funzione f_i calcolato nel punto j (cioè j è l'elemento del dominio, $f_i(j)$ il corrispondente elemento del codominio) potremmo legittimamente definire una nuova funzione g tale che $g(i) = f_i(i) + 1$ per ogni valore di i. Ma se la g appartenesse all'elenco delle f_i dovrebbe coincidere con una f_k per un opportuno valore di k, e per tale valore si avrebbe l'assurdo $f_k(k) = f_k(k) + 1$. Come vedremo tra poco questo fatto ha importanti conseguenze.

Dalla nozione di funzione si passa a quella di problema. Nell'ambito del calcolo un *problema* consiste in una domanda posta su alcuni *dati*, e la sua soluzione consiste in una risposta, o *risultato*, avente opportune proprietà. La precisa formulazione di un problema dipende da come sono rappresentati i dati e da cosa deve specificare il risultato. Per esempio il problema $\mathcal{P}_{ric}(k, I)$ chiede di stabilire se un elemento k è presente in un insieme I: i valori di k e I sono assegnati di volta in volta, cioè (come si dice in gergo) per ogni *istanza* del problema; il risultato è un'attestazione di appartenenza o meno di k a I. Qualunque sia l'alfabeto su cui si opera, dati e risultati sono rappresentati da sequenze di caratteri: se si ricerca il nome di una persona in un insieme di nomi, una sequenza di caratteri alfabetici specificherà i valori di k e degli elementi di I opportunamente concatenati, e un singolo carattere binario potrà rappresentare il risultato (1 se $k \in I$, 0 se $k \notin I$). Quindi un problema specifica in generale un'associazione tra sequenze, o tra interi che le rappresentano, e la sua risoluzione corrisponde al calcolo di una funzione. Non vogliamo con questo affermare che per risolvere un problema si debbano necessariamente stabilire tali corrispondenze

poiché i nomi sono numerabili e i reali non lo sono. In sostanza solo un insignificante sottoinsieme dei numeri reali è definibile, e la loro introduzione non aggiunge alcuna potenza al calcolo sulle sequenze finite.

formali, ma il precedente ragionamento ci consente di concludere che esistono in linea di principio tanti problemi quante funzioni; e poiché queste non sono numerabili così non lo sono i problemi. In sostanza *esistono più problemi che sequenze finite*.

Siamo così giunti al protagonista del calcolo, cioè l'*algoritmo*. Informalmente un algoritmo è la descrizione finita di una successione di passi da eseguire per risolvere un problema (calcolare una funzione) impiegando le regole elementari di elaborazione di cui si dispone. La teoria degli algoritmi ha però raggiunto un alto livello di maturità, e sono stati messi in luce alcuni paradigmi generali che consentono di descrivere a grandi linee una procedura di calcolo indipendentemente dalle azioni elementari da compiere; azioni che vengono poi specificate quando l'algoritmo è assegnato a un particolare attuatore ovvero, nella pratica, quando è trasformato in un *programma* destinato a un calcolatore. Vedremo in seguito come si organizza un algoritmo e se ne valuta la qualità; per il momento vogliamo esaminarne alcune caratteristiche generali.

Anzitutto notiamo che l'esecuzione di un calcolo su alcuni dati potrebbe non terminare perché il procedimento è stato mal concepito, o perché il problema stesso comporta che ciò avvenga (per esempio si tratta di un videogioco sempre attivo in attesa di clienti). Noi accetteremo solo algoritmi che terminano in un numero finito di passi per tutti i dati d'ingresso. Per di più se due algoritmi diversi risolvono lo stesso problema considereremo migliore quello che termina prima.

La formulazione di un algoritmo dipende ovviamente dal "modello di calcolo" utilizzato, che può per esempio essere un meccanismo matematico astratto come una Macchina di Turing o un programma in linguaggio C per un *personal computer*. Una tesi fondamentale della logica matematica, formulata negli anni tra il 1930 e il 1940 e mai smentita, afferma che tutti i modelli di calcolo "ragionevoli" hanno la stessa potenza, nel senso che sono in grado di calcolare le stesse funzioni. Qualsiasi modello si scelga gli algoritmi devono esservi descritti, ovvero rappresentati da sequenze finite di caratteri, quindi gli algoritmi sono possibilmente infiniti ma numerabili. La prima conseguenza è che non tutte le funzioni possono essere calcolate, ovvero non tutti i problemi possono essere risolti, perché esistono (infinitamente) più funzioni che algoritmi di calcolo. Mentre questo fatto è immediato, più difficile fu individuare funzioni algoritmicamente non calcolabili ovvero problemi algoritmicamente irrisolubili. I contributi fondamentali si devono ad alcuni logici tra cui Turing, che costruirono esempi di funzioni il cui calcolo avrebbe condotto a situazioni autocontraddittorie.

Pur senza pretendere di approfondire qui l'argomento, vale la pena di ricordarne qualche aspetto che potrà riguardarci. Anzitutto possiamo limitare lo studio alle funzioni che hanno come codominio $\{0, 1\}$, corrispondenti a problemi che richiedono una risposta binaria interpretata come *decisione* (per esempio il problema $\mathcal{P}_{ric}(k, I)$ visto sopra). Infatti anche queste funzioni possono essere rappresentate con una sequenza

infinita di risultati come la 2.3, e sono quindi non numerabili. La dimostrabile assenza di un algoritmo di calcolo qualifica la funzione *non calcolabile* e il corrispondente problema *indecidibile*. La costruzione di funzioni non calcolabili è sostanzialmente basata sullo stesso meccanismo che ci ha permesso sopra di dimostrare la non esistenza della funzione g, poiché è possibile descrivere una funzione in termini del comportamento di altre funzioni e costruire in tal modo dei paradossi. La profonda lezione di Turing è che non possono esistere algoritmi che decidono il comportamento di altri algoritmi esaminandoli "dall'esterno", cioè senza passare attraverso la loro simulazione. E in effetti il primo risultato fu la dimostrazione di indecidibilità del *problema della terminazione*: non può esistere un algoritmo T che decida in tempo finito se un algoritmo arbitrario A termina la sua computazione su dati arbitrari D.[3]

Dallo studio di casi tanto astratti è discesa nel tempo una lunga serie di risultati concreti che si riconducono l'uno all'altro. Per citare solo un esempio è indecidibile il problema $\mathcal{P}_{diof}(E)$ delle *equazioni diofantee*: data un'equazione algebrica E a coefficienti interi, di grado arbitrario e in un numero arbitrario di variabili, si deve decidere se la E ammette una soluzione in cui tutte le variabili hanno valore intero. Come avremo modo di vedere esiste un algoritmo di calcolo per particolari forme della E, ma non ne esiste alcuno per una forma generale: con una catena di derivazioni si può dimostrare che decidere algoritmicamente se un'equazione diofantea arbitraria ha soluzioni intere equivarrebbe a stabilire se un algoritmo arbitrario termina o no i suoi calcoli.[4]

Possiamo in conclusione osservare che il concetto di funzione è primitivo e valido in astratto, quindi non dobbiamo stupirci della nostra impossibilità di rivolgerci all'intero mondo delle funzioni. Gli algoritmi come li intendiamo sono invece creazione dell'uomo e per essere definiti devono soddisfare le proprietà di base di un qualunque sistema di rappresentazione. Sono due mondi completamente diversi, ma dalla loro correlazione scaturisce un risultato sorprendente: le funzioni calcolabili sono infinitamente meno numerose di quelle non calcolabili, eppure le prime si presentano spontaneamente alla nostra attenzione mentre le seconde sembrano costruite artificialmente per convalidare la teoria. Il lettore non si preoccupi: da questo momento considereremo solo funzioni calcolabili!

[3]Naturalmente ciò non significa che non si possa decidere in tempo finito la terminazione di algoritmi particolari: il problema è indecidibile per una coppia $\langle A, D\rangle$ scelta arbitrariamente. Si noti anche che T potrebbe simulare il funzionamento di A su D, ma se questa computazione non termina anche quella di T non termina.

[4]Un algoritmo esauriente che provi tutte le possibili combinazioni di interi come soluzione dell'equazione non è accettabile perché non termina in tempo finito se la soluzione non esiste.

Capitolo 3

Algoritmi e complessità

Una volta stabilita la relazione tra il mondo dei problemi, o delle funzioni associate a essi, e il mondo degli algoritmi, vediamo lungo quali direttrici principali si sviluppano questi ultimi, come si descrivono in pratica e come se ne determina la qualità.

Nel capitolo precedente abbiamo appreso che dobbiamo rivolgerci a un insieme ristretto di problemi: potranno essere problemi di decisione, quindi a risultato binario, o problemi più generali, ma in ogni caso dovranno ammettere un algoritmo di soluzione. E tra questi problemi scopriremo un'altra linea che li divide in due famiglie drasticamente separate: problemi *trattabili* che ammettono algoritmi di soluzione praticamente eseguibili, o *intrattabili* se tutti gli algoritmi possibili, o quanto meno noti, sono così inefficienti da impedirne una pratica esecuzione su qualsiasi calcolatore.

L'avvento dei calcolatori è stato accompagnato dallo sviluppo di *linguaggi di programmazione* in cui ogni dettaglio deve essere perfettamente specificato perché non è previsto che la macchina prenda decisioni autonome. In conseguenza, benché gli algoritmi possano essere descritti utilizzando un qualunque formalismo univocamente interpretabile, è consuetudine descriverli in un linguaggio di tipo progammativo anche se eventualmente non coincidente con alcuno di essi. Seguiremo questa via fidando che gli elementi di base della programmazione siano già noti al lettore o siano direttamente comprensibili dal contesto; e inseriremo comunque qualche spiegazione nel testo per accertarci della corretta interpretazione dei costrutti più complessi. Questo modo di procedere ci consentirà di descrivere tutte le operazioni in modo preciso e non ambiguo, bene adattandosi alla natura eminentemente matematica dei problemi che dovremo trattare: e per iniziare descriveremo due algoritmi antichissimi, ma ancora molto utili oggi, che nei documenti originali furono redatti rispettivamente in caratteri ieratici egizi e greci, assai diversi dai nostri.

3.1 Paradigmi algoritmici

Verso il 1650 a.C. lo scriba egiziano Ahmes redasse un papiro di argomento matematico tratto, per sua dichiarazione, da un esemplare del Regno Medio. Questo straordinario documento, conservato quasi per intero nel British Museum, contiene un gran numero di definizioni, problemi e regole di calcolo tra cui vogliamo ricordare un algoritmo per eseguire la moltiplicazione tra interi positivi. Detti a e b i due fattori, e p il loro prodotto, l'algoritmo egizio si può descrivere nella forma seguente:

`Function Molt`(a, b):
  $p \leftarrow 0$;
  while $a \geq 1$ **do**
    if a è dispari **then** $p \leftarrow p + b$;
    $a \leftarrow \lfloor a/2 \rfloor$;
    $b \leftarrow b \times 2$;
  return p.

Perché non vi siano dubbi sul formalismo adottato ricordiamo che:

- Con la parola `Function` si intende un'unità di programma che assume un valore calcolato al suo interno (p nell'esempio) e rilasciato attraverso il comando **return**, il quale ordina anche l'arresto del calcolo nell'unità stessa. Alla parola `Function` seguono il *nome* dell'unità di programma e, tra parentesi, i *parametri* su cui essa è chiamata a operare: a ogni impiego verranno comunicati all'unità i valori di tali parametri.

- La freccia comanda l'assegnazione, alla *variabile* che appare dalla parte della punta, del *valore* di quanto appare dalla parte della cocca ove può essere eseguito un calcolo il cui risultato sarà oggetto dell'assegnazione. Nell'esempio la frase $a \leftarrow \lfloor a/2 \rfloor$ ordina di dividere per due il valore corrente di a, prendere la parte intera, assegnare il risultato ad a che assume così un nuovo valore: si noti che la stessa variabile a avrà valore diverso a seconda della parte della freccia in cui si trova.

- Il costrutto **while** `C` **do** `S` ordina la ripetizione della sequenza `S` di passi (individuati dalla stessa indentatura) fintanto che è verificata la condizione `C`; ciò implica che nella sequenza `S` venga cambiato qualche elemento che appare in `C` (nell'esempio il valore di a), altrimenti il ciclo sarebbe ripetuto in eterno. Il costrutto **if** `C` **then** `S` dovrebbe avere significato ovvio.

- Il paradigma algoritmico che prevede l'indicazione esplicita di tutte le iterazioni del calcolo è detto *iterativo*. L'algoritmo dell'esempio è iterativo: la ripetizione dei passi computazionali è indicata esplicitamente nel costrutto **while**.

- Il nuovo programma:

  $r \leftarrow a +$ `Molt`(b, c)

  contiene una *chiamata* della funzione `Molt` il cui risultato viene sommato al valore di a per ottenere r. Eseguito per i valori $a = 5$, $b = 7$, $c = 3$ il programma dà come risultato $r = 5 + 7 \times 3 = 26$.

La correttezza dell'algoritmo `Molt` si dimostra immediatamente notando che $a \times b = a/2 \times 2b$ se a è pari, e $a \times b = (a-1) \times b + b = \lfloor a/2 \rfloor \times 2b + b$ se a è dispari. Seguiamone l'esecuzione per i valori $a = 37$ e $b = 23$, quindi $p = 37 \times 23 = 851$, prima di commentarne la struttura:

$p + b$	p	a	b
	0	37	23
0 + 23	23	18	46
	23	9	92
23 + 92	115	4	184
	115	2	368
	115	1	736
115 + 736	851	0	

È chiaro che questo algoritmo è completamente diverso da quello impiegato usualmente da noi. Il calcolo procede per addizioni, nonché per dimezzamenti e raddoppiamenti dei fattori, e questo induce a pensare che la sua ideazione si debba all'uso di abaci che permettevano di eseguire facilmente queste tre operazioni.[1] Rispetto al nostro, l'algoritmo si impone per la totale indipendenza dal metodo di rappresentazione dei numeri: gli antichi egiziani, per esempio, impiegavano una notazione decimale ma non conoscevano lo zero. Il confronto di efficienza tra i due algoritmi è rimandato al prossimo paragrafo; per il momento notiamo che se l'algoritmo egizio è applicato a

[1] Fino a tempi molto recenti l'algoritmo era in uso in Russia e nei paesi asiatici confinanti ove i calcoli di ogni giorno si eseguivano correntemente col pallottoliere, e invero è l'unico metodo pratico per eseguire la moltiplicazione con tale strumento. Per questo motivo il metodo è spesso indicato nei libri come *moltiplicazione del contadino russo*.

due numeri binari, i test e i calcoli aritmetici sono gli stessi eseguiti dai calcolatori nella loro realizzazione più semplice della moltiplicazione.[2]

Il secondo algoritmo che consideriamo ha "solo" duemilatrecento anni, poiché apparve negli *Elementi* di Euclide. Dopo aver definito un numero primo (*protos arithmós*) come un numero "misurato" (cioè diviso) solo dall'unità, e aver definito in conseguenza i numeri primi tra loro, Euclide enuncia la seguente proposizione (Libro VII, Prop. II):

Dati due numeri non primi tra loro trovare la loro più grande misura comune.

È il problema della determinazione del *massimo comun divisore*, che studieremo ora per la sua grande importanza nell'aritmetica in genere e nella crittografia in particolare. All'enunciato Euclide fa seguire la descrizione del metodo di calcolo, tanto efficiente e geniale da essere impiegato ancor oggi adattandolo al nostro formalismo.[3] Dati due interi $a > 0$, $b \geq 0$, con $a \geq b$, l'algoritmo di calcolo del massimo comun divisore *mcd(a,b)* è il seguente:

`Function Euclid(`a, b`):`
  if $b = 0$
    then return a
    else return `Euclid(`$b, a \bmod b$`)`.

Il formalismo contiene importanti novità:

- Il costrutto **if** `C` **then** `S` **else** `T` ordina l'esecuzione del passo (o sequenza di passi) `S` oppure `T` a seconda che la condizione `C` sia verificata o meno.

- La funzione contiene una *chiamata ricorsiva*, cioè a sé stessa, nella quale i parametri hanno i valori di b e di $a \bmod b$ (ricordiamo che $a \bmod b$ indica il resto della divisione intera tra a e b). Il calcolo della funzione chiamante invoca cioè l'esecuzione di un calcolo uguale su valori diversi dei parametri. La difficoltà pare solo rimandata, e in effetti una struttura di calcolo così concepita è accettabile solo se corredata di una clausola di chiusura che stabilisca come eseguire il cal-

[2] La frase: **if** *a è dispari* **then** $p \leftarrow p + b$ corrisponde all'operazione in hardware: se il bit meno significativo del moltiplicatore a è 1, addiziona il moltiplicando b al prodotto parziale p. Le frasi: $a \leftarrow \lfloor a/2 \rfloor$ e $b \leftarrow b \times 2$ corrispondono agli *shift* destro e sinistro dei due fattori.

[3] In particolare è chiaro che Euclide, pur senza farne esplicito riferimento, trattava i numeri come le corrispondenti misure di segmenti: un segmento ne "misurava" un'altro se era in esso contenuto un numero esatto di volte. La sua formulazione dell'algoritmo è basata su questa impostazione: apparentemente Euclide non si curò di esaminarne la straordinaria efficienza poiché considerava l'aritmetica una scienza puramente speculativa.

colo per valori limite dei parametri: nell'esempio il verificarsi della condizione $b = 0$ e corrispondente esecuzione della parte **then** del programma.

- Nell'esempio la chiamata ricorsiva costituisce l'ultima frase della funzione chiamante; se ciò non si verifica, il calcolo di questa deve essere sospeso in attesa che sia completato il calcolo della funzione chiamata, e successivamente portato a termine: ciò genera un gran numero di esecuzioni, tutte vive e interrotte una dentro l'altra in attesa di essere completate a partire dalla più "interna".

- Il paradigma algoritmico che prevede l'indicazione di alcune iterazioni del calcolo attraverso chiamate ricorsive è detto a sua volta *ricorsivo*.

Per dimostrare la correttezza dell'algoritmo `Euclid` è sufficiente dimostrare che $mcd(a,b) = mcd(b, a \bmod b)$. A tale scopo, ponendo $m = mcd(a,b)$, avremo $a = \alpha m$, $b = \beta m$, con α e β interi e primi tra loro. Possiamo anche porre: $a = q \cdot b + a \bmod b$ ove q è un quoziente intero; da cui: $a \bmod b = (\alpha - q\beta)m = \gamma m$ con γ intero. Dunque m è un divisore anche di $a \bmod b$. Ora è facile dimostrare che, poiché α e β sono primi tra loro, lo sono anche β e γ, e quindi il massimo comun divisore tra b e $a \bmod b$ è proprio m.

Seguiamo infine l'esecuzione dell'algoritmo per i valori $a = 306$ e $b = 135$. Troveremo in pochi passi $mcd(306, 135) = 9$.

a	b	$a \bmod b$	*risultato finale*
306	135	36	
135	36	27	
36	27	9	
27	9	0	
9	0		9

I due problemi studiati, e i loro algoritmi di soluzione, sono in genere detti *numerici* poiché i dati sono numeri, in contrapposizione ai problemi e algoritmi *combinatori* in cui più genericamente si richiede di operare su sequenze. Per quanto abbiamo discusso nel capitolo 2 questa distinzione è molto superficiale e al limite illegittima, ma a volte è utile. Vediamo un importante esempio di problema combinatorio.

Dobbiamo ricercare un elemento k in un insieme I: è il problema decisionale $\mathcal{P}_{ric}(k, I)$ già discusso nel paragrafo 2.3, che ora affrontiamo in una forma più generale chiedendo, nel caso che k sia un elemento di I, di indicare anche come "accedere"

a tale elemento. Per far questo occorrono alcune precisazioni. I dati, cioè k e gli elementi di I, sono sequenze arbitrarie ma nella pratica hanno spesso una lunghezza massima prefissata: per esempio rappresentano nomi di persone, per ciascuno dei quali è assegnato un numero massimo di caratteri alfabetici. Possiamo allora rappresentare gli elementi di I in un *vettore* A le cui celle sono numerate da 1 a $|A|$; con $A[j]$ si indica l'elemento in posizione j, e se k appartiene all'insieme si deve fornire come risultato il valore di j per cui $k = A[j]$. Consideriamo il confronto tra due elementi come operazione elementare senza entrare nel merito di come venga eseguita. Il risultato del confronto è binario nella sua forma più semplice, e indica se i due elementi confrontati sono uguali o diversi. Più ambizioso, ma legittimo, è immaginare che tra i dati esista una relazione d'ordine totale e il risultato del confronto sia ternario e indichi se due elementi sono uguali, o se uno precede o segue l'altro secondo la relazione d'ordine. Per indicare questo risultato impiegheremo i simboli aritmetici: $=$, $<$, $>$.[4] Possiamo ora presentare un primo algoritmo di ricerca:

`Function Ricerca_Sequenziale`(k, A):
  for $j \leftarrow 1$ **to** $|A|$ **do**
    if $k = A[j]$ **then return** j;
  return 0.

Il formalismo contiene una novità:

- Il costrutto **for** `C` **do** `S` ordina l'esecuzione del passo (o sequenza di passi) `S` per ogni valore intero di un *indice* che varia entro i limiti specificati nel comando `C`. Nell'esempio si comanda all'indice j di assumere in successione tutti i valori interi da 1 a $|A|$: l'algoritmo è quindi iterativo.

Il risultato della funzione `Ricerca_Sequenziale` è un intero: 0 se $k \notin I$, j compreso tra 1 e $|A|$ se $k \in I$. Potrebbe apparire inutile, nel secondo caso, indicare la posizione di k in A poiché in fondo il valore dell'elemento trovato è già noto; ma in realtà l'algoritmo si impiega in genere per la ricerca di *chiavi*, cioè nomi cui sono associate altre informazioni. Esempi immediati sono quelli della ricerca di un nome (la chiave) k nell'elenco del telefono, o di una parola in un dizionario: A rappresenta l'elenco o il dizionario, la risposta $j > 0$ indica la posizione nella quale, assieme a k, si trova il numero telefonico o la definizione cercata. La funzione `Ricerca_Sequenziale`

[4] Nella pratica le sequenze sono binarie e nel confronto vengono interpretate come numeri, sicché la relazione d'ordine è la normale relazione di maggioranza tra interi. Impiegando il codice internazionale ASCII per rappresentare i caratteri alfabetici, due parole ordinate alfabeticamente lo sono anche se interpretate come numeri.

scandisce tutti gli elementi di A fino a incontrare quello cercato: il metodo usualmente svolto a mano è molto diverso, ma per essere eseguito richiede che gli elementi del vettore siano disposti in ordine alfabetico. Vediamo dunque come si possa formalizzare l'algoritmo su un vettore ordinato.

Le operazioni eseguite dall'uomo fanno implicitamente uso di alcune proprietà statistiche delle chiavi elencate: se per esempio l'ordine è alfabetico, una chiave che comincia per d sarà cercata verso l'inizio dell'elenco, salvo poi saltare indietro di un gruppo di pagine se si è raggiunta la lettera f; il meccanismo è via via applicato a salti sempre più brevi, finché si leggono alcune chiavi in sequenza attorno a quella cercata. Questo metodo non è facile da formalizzare per un calcolatore, che in compenso può calcolare facilmente la posizione centrale nell'elenco se questo è memorizzato in un vettore. Il nuovo algoritmo, detto di *ricerca binaria*, impiega tale calcolo per la ricerca di k in una sezione arbitraria di A compresa tra le posizioni i e j, con $1 \leq i \leq j \leq |A|$ (se si raggiunge la condizione $i > j$ l'algoritmo termina dichiarando che k non è presente in A).

`Function Ricerca_Binaria`(k, A, i, j):

if $i > j$ **then return** 0;
$m \leftarrow \lfloor \frac{i+j}{2} \rfloor$;
if $k = A(m)$ **then return** m;
if $k < A(m)$ **then return** `Ricerca_Binaria`$(k, A, i, m-1)$;
if $k > A(m)$ **then return** `Ricerca_Binaria`$(k, A, m+1, j)$.

Sull'impiego di questa funzione è necessario qualche commento:

- La funzione è definita tra estremi arbitrari i, j per poter essere richiamata ricorsivamente al suo interno su sezioni diverse del vettore A. Per cercare k nell'intero vettore si impiegherà la chiamata:

  `Ricerca_Binaria`$(k, A, 1, |A|)$

  che innesca il meccanismo di calcolo sui sottovettori.

- Nella sezione di A compresa tra i e j, il valore $m = \lfloor \frac{i+j}{2} \rfloor$ corrisponde alla posizione scelta come "centrale", che è strettamente al centro solo se il numero di celle della sezione è dispari.

La correttezza dell'algoritmo `Ricerca_Binaria` può essere facilmente dimostrata per induzione. Seguiamone il funzionamento sulla ricerca dell'elemento $k = 48$ nel seguente vettore ordinato A che contiene quattordici interi:

posizione:	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A:	12	17	18	21	23	25	**30**	35	48	50	51	58	65	70
	12	17	18	21	23	25	30	35	48	50	**51**	58	65	70
	12	17	18	21	23	25	30	35	**48**	50	51	58	65	70

Le posizioni esaminate sono 7, 11, 9 (elementi 30, 51, 48) e qui l'algoritmo si arresta trasmettendo il valore 9 dell'indice ove si trova l'elemento cercato. La ricerca di $k = 24$ scandisce le posizioni 7, 3, 5, 6: a questo punto l'algoritmo esegue la chiamata ricorsiva `Ricerca_Binaria`$(16, A, 6, 5)$ (con $i > j$) e qui si arresta trasmettendo il valore 0 che indica l'assenza dell'elemento 24. In ogni caso l'algoritmo esamina pochi elementi del vettore e appare sotto questo aspetto migliore di `Ricerca_Sequenziale`: lo studio di questo miglioramento, che come vedremo cresce grandemente con la lunghezza del vettore, è argomento del prossimo paragrafo.

3.2 Complessità computazionale

Per risolvere un problema siamo in genere interessati a scoprire l'algoritmo "più efficiente" tra tutti i possibili indipendentemente da altre proprietà. Per qualificare questa affermazione dobbiamo chiarire su quali parametri valutare l'efficienza, che normalmente coincide con l'ottimizzazione di impiego delle risorse disponibili. Impiegando un calcolatore le risorse principali sono la memoria e il tempo necessari a eseguire il calcolo. La richiesta di ridurre il tempo è in genere prevalente: efficiente dunque vuol dire veloce, e come vedremo vi sono argomenti molto importanti che avvalorano questa posizione.

È bene chiarire che non misureremo il tempo in secondi, il che non avrebbe alcun senso se non specificando ogni dettaglio del calcolatore e del corredo di software utilizzati, ma attraverso una funzione matematica che assume valori approssimativamente proporzionali al tempo reale: questa funzione misura la *complessità computazionale in tempo* (o semplicemente *complessità*) di un algoritmo. Ciò comporta una certa grossolanità di giudizio ma permette di confrontare l'efficienza di algoritmi diversi in modo molto semplice.

Dobbiamo anzitutto stabilire quali sono le variabili indipendenti su cui operare. In genere la compessità si valuta in funzione di un singolo parametro n detto *dimensione dell'ingresso* o *dimensione dei dati*, che indica la lunghezza della sequenza che specifica il valore dei dati per la particolare istanza del problema che l'algoritmo è chiamato a risolvere. Per esempio se l'algoritmo `Molt`(a, b) del paragrafo precedente è applicato all'istanza di moltiplicazione tra $a = 351$ e $b = 66$ possiamo porre $n = 5$ poiché tante sono le cifre (decimali) con cui si rappresentano i dati. E qui sorge apparentemente

un problema poiché il valore indicato dipende dalla *base di numerazione*, cioè dal metodo di rappresentazione scelto. Dobbiamo approfondire questo punto prima di procedere.

Cambiando il metodo di rappresentazione, ma rimanendo nell'ambito di rappresentazioni efficienti nel senso tecnico della parola (paragrafo2.1), le lunghezze delle diverse sequenze che rappresentano uno stesso dato sono *proporzionali tra loro* poiché espresse da funzioni logaritmiche in cui cambia solo la base, ovvero la cardinalità dell'alfabeto scelto. Infatti la funzione logaritmo gode della proprietà:

$$\log_r x = \log_r s \cdot \log_s x \tag{3.1}$$

che mostra come si possa passare tra due basi arbitrarie r, s moltiplicando la funzione per la quantità $\log_r s$, che è una *costante* per r e s costanti (in particolare indipendenti da x). Per esempio un intero N è rappresentato con $\lceil \log_{10} N \rceil$ cifre impiegando una numerazione in base 10. Passando da questa base alla base 2 il numero di cifre necessario viene moltiplicato per $\log_2 10 \simeq 3.32$. In conclusione affinché la complessità di un algoritmo sia indipendente dal metodo di rappresentazione dei dati dovremo limitarci a valutarla in *ordine di grandezza*, prendendo come dimensione n di un problema un valore proporzionale alla lunghezza della sequenza di ingresso.

Nello studio degli algoritmi i tre ordini di grandezza Θ, O e Ω hanno rilevanza particolare. Ne ricordiamo la definizione matematica limitata al caso in cui tutte le funzioni in gioco siano non negative (poiché rappresentano complessità di algoritmi) e siano definite per valori non negativi di una sola variabile n (la dimensione del problema). Gli ordini di grandezza sono insiemi infiniti che comprendono tutte le funzioni che hanno un dato comportamento rispetto a una funzione assegnata $g(n)$:

- $\Theta(g(n))$ è l'insieme di tutte le funzioni $f(n)$ per cui esistono tre costanti positive c_1, c_2, n_0 tali che $c_1\, g(n) \leq f(n) \leq c_2\, g(n)$ per ogni $n \geq n_0$. Per ogni funzione $f(n)$ di questo insieme si scrive, con notazione impropria perché non insiemistica: $f(n) = \Theta(g(n))$, e si legge: $f(n)$ è di ordine *teta* $g(n)$ (una simile terminologia si usa anche per gli altri ordini). Questo significa che, al crescere di n oltre n_0, la funzione $f(n)$ è compresa in una fascia delimitata da $c_1 g(n)$ e $c_2 g(n)$: ovvero che la $f(n)$ ha lo stesso comportamento della $g(n)$ a parte una costante moltiplicativa. Per esempio $3n^2 - 2n + 1 = \Theta(n^2)$ poiché, scegliendo $c_1 = 1$, $c_2 = 3$, $n_0 = 2$ si ha: $n^2 < 3n^2 - 2n + 1 < 3n^2$ per ogni $n \geq 2$.

- $\mathrm{O}(g(n))$ è l'insieme di tutte le funzioni $f(n)$ per cui esistono due costanti positive c, n_0 tali che $f(n) \leq c\, g(n)$ per ogni $n \geq n_0$. Questo significa che, al crescere di n oltre n_0, il comportamento della funzione $f(n)$ è limitato superiormente

da quello della $g(n)$ a parte una costante moltiplicativa. Per esempio si ha: $3n^2 - 2n + 1 = \mathrm{O}(n^2)$, ma anche $3n^2 - 2n + 1 = \mathrm{O}(n^3)$.

- $\Omega(g(n))$ è l'insieme di tutte le funzioni $f(n)$ per cui esistono due costanti positive c, n_0 tali che $c\,g(n) \leq f(n)$ per ogni $n \geq n_0$. Questo significa che, al crescere di n oltre n_0, il comportamento della funzione $f(n)$ è limitato inferiormente da quello della $g(n)$ a parte una costante moltiplicativa. Per esempio si ha: $3n^2 - 2n + 1 = \Omega(n^2)$, ma anche $3n^2 - 2n + 1 = \Omega(n^{1.5})$.

- Si noti che $f(n) = \Theta(g(n))$ se e solo se $f(n) = \mathrm{O}(g(n))$ e $f(n) = \Omega(g(n))$.

Per valutare l'ordine di grandezza di una funzione composta da una somma di più termini si deve considerare solo il termine che va all'infinito più rapidamente al crescere di n (o termine di grado massimo se la funzione è un polinomio), trascurando gli altri. Similmente si devono trascurare le costanti moltiplicative assorbite dalla definizione di ordine. Così nello studio della funzione $3n^2 - 2n + 1$ l'unico elemento che conta è n^2, sia che si consideri l'ordine Θ, o O, o Ω.

Una considerazione particolare merita la funzione logaritmo che apparirà di continuo nei nostri calcoli. Poiché i logaritmi di uguale argomento ma di diversa base sono tutti proporzionali tra loro (relazione 3.1), la base si ignora entro gli ordini di grandezza poiché ha solo l'effetto di una costante moltiplicativa. Si scrive per esempio: $3(\log_2 n)^3 = \Theta(\log^3 n)$. Ricordiamo inoltre che il logaritmo di n cresce più lentamente di n elevato a qualsiasi esponente positivo, cioè per qualunque $b > 1$ e $\epsilon > 0$ si ha: $\log_b n = \mathrm{O}(n^\epsilon)$, $n^\epsilon = \Omega(\log n)$ e $\log_b n \neq \Theta(n^\epsilon)$.

La funzione di complessità (in tempo) è in genere indicata con $T(n)$. Essa viene calcolata in ordine di grandezza non solo per affrancarla dal metodo di rappresentazione dei dati, ma anche per renderla indipendente dalle caratteristiche del sistema di calcolo. Si desidera cioè giungere a una valutazione dell'efficienza degli algoritmi che abbia per quanto possibile validità generale, con lo scopo di poter confrontare e scegliere tra algoritmi diversi, o valutare se un algoritmo è del tutto proponibile. Vediamo cosa si può dire degli algoritmi presentati nel paragrafo precedente.

Per l'algoritmo `Molt(`a, b`)` il calcolo non è semplicissimo. La dimensione dell'ingresso è $n = \Theta(\log a + \log b)$. Il prodotto finale $a \times b$ si rappresenta con $\Theta(\log ab) = \Theta(\log a + \log b)$ cifre, dunque ha la stessa dimensione dell'ingresso. A parte l'operazione iniziale che richiede tempo costante, l'algoritmo ripete un ciclo **while** in cui si eseguono tre operazioni aritmetiche di addizione e *shift* che richiedono complessivamente tempo $\Theta(\log a + \log b)$ perché eseguite su dati di tali dimensioni. Notiamo ora che a ogni ripetizione del ciclo il valore di a viene dimezzato fino a raggiungere

il valore 1, quindi il ciclo è ripetuto $\lfloor \log_2 a \rfloor = \Theta(\log a)$ volte.[5] La complessità di `Molt(`a, b`)` è quindi $\Theta(\log a \cdot (\log a + \log b)) = \Theta(\log^2 a + \log a \cdot \log b)$. Questa espressione assume valore massimo se a e b sono dello stesso ordine di grandezza, nel qual caso la complessità dell'algoritmo risulta:

$$T(n) = \Theta(n^2)$$

cioè quadratica nella *dimensione* (non nel *valore*) dei dati d'ingresso. Lasciamo al lettore il compito di verificare che l'usuale algoritmo di moltiplicazione ha la stessa complessità.

Il precedente calcolo si è concluso con l'esame del *caso pessimo* in riferimento ai possibili valori assunti dai dati d'ingresso. È un approccio pessimistico, ma garantisce che l'algoritmo termini *sempre* entro il tempo calcolato. Per tener conto che la situzione potrebbe essere più favorevole possiamo esprimere la complessità di `Molt(`a, b`)` come $T(n) = \mathrm{O}(n^2)$, che mostra come il valore n^2 non sia necessariamente sempre raggiunto. Questo spiega il senso dell'ordine O (o Ω) come limite superiore (o inferiore). Un'analisi alternativa molto più complessa si rivolge al *caso medio* e non sarà qui presa in considerazione perché non particolarmente rilevante ai nostri fini.

L'analisi di complessità dell'algoritmo `Euclid(`a, b`)` è piuttosto complicata e ci limitiamo a riportarne alcuni punti. La dimensione dell'ingresso è $n = \Theta(\log a + \log b) = \Theta(\log a)$ perché per ipotesi $a \geq b$. La complessità $T(n)$ è proporzionale al numero di chiamate ricorsive della funzione stessa, che dipende dal valore dei dati (se per esempio a è multiplo di b si ha $a \bmod b = 0$ e l'algoritmo termina alla seconda chiamata). Si può comunque dimostrare che in ogni chiamata risulta $a \bmod b < a/2$, quindi il parametro a si riduce almeno della metà ogni due chiamate: ne segue che il numero di chiamate è limitato da $\mathrm{O}(\log a) = \mathrm{O}(n)$. Ogni chiamata richiede di calcolare un modulo, e questa operazione può essere eseguita in tempo $\Theta(\log a \cdot \log b) = \mathrm{O}(n^2)$ con l'usuale algoritmo di divisione (ma si può fare di meglio). In conclusione la complessità dell'algoritmo risulta:

$$T(n) = O(n^3)$$

cioè al più cubica nella dimensione dell'ingresso.

L'analisi degli algoritmi `Ricerca_Sequenziale` e `Ricerca_Binaria` è molto più semplice. Nella nostra ipotesi gli elementi dell'insieme su cui si esegue la ricerca sono memorizzati in un vettore A le cui celle hanno dimensione prefissata e indi-

[5] Se si parte da 1 e si raddoppia il valore per n volte si ottiene 2^n. Poiché $\log_2 n$ è la funzione inversa di 2^n, se si dimezza un numero a per $\lfloor \log_2 a \rfloor$ volte si ottiene 1 (l'approssimazione inferiore è necessaria se a non è una potenza di 2).

pendente dalla lunghezza del vettore stesso. Ai nostri fini la dimensione delle celle è quindi una costante, e la dimensione complessiva dell'ingresso è $n = \Theta(|A|)$: trascurando la costante moltiplicativa porremo direttamente $n = |A|$. L'algoritmo `Ricerca_Sequenziale` ripete al massimo $|A|$ volte un ciclo all'interno del quale esegue un confronto tra dati di lunghezza costante; quindi ogni iterazione richiede tempo costante e in complesso si ha:

$$T(n) = O(n).$$

Il limite superiore contenuto nella notazione O è giustificato dal fatto che l'algoritmo potrebbe arrestarsi in pochi passi se l'elemento cercato è in una delle prime posizioni del vettore, ma è necessaria l'intera scansione se tale elemento è l'ultimo o addirittura non è presente.

L'algoritmo `Ricerca_Binaria` è più interessante. Notiamo che esso contiene due chiamate ricorsive su $\lfloor n/2 \rfloor$ elementi in conseguenza del verificarsi di condizioni opposte: quindi in ogni caso si esegue solo una di tali chiamate. Ammettendo che il test tra k e $A[m]$ abbia risultato ternario e quindi si esegua una volta sola per stabilire la relazione tra i due elementi, il numero totale $C(n)$ di confronti eseguiti è espresso dalla *equazione di ricorrenza*:

$$C(n) \begin{cases} = 1 & \text{se } n = 1, \\ \leq 1 + C(\lfloor n/2 \rfloor) & \text{se } n > 1. \end{cases}$$

Il valore limite $C(1) = 1$ indica che si esegue un solo confronto quando si raggiunge un sottoinsieme di un elemento, e il segno $\leq$ nell'equazione indica che non si esegue una successiva chiamata ricorsiva se si trova k. L'equazione si risolve facilmente sviluppando la serie:

$$C(n) \leq C(n/2) + 1 \leq C(n/4) + 1 + 1 \leq \cdots \leq C(1) + \cdots + 1 + 1 \leq 1 + \log_2 n$$

ove si è posto $C(1) = 1$, e il numero di uni nella somma è al più $\log_2 n$ come già osservato nell'analisi di `Molt`. In conclusione, ponendo che le singole operazioni contenute nell'algoritmo richiedano tempo costante, e che quindi la sua complessità $T(n)$ sia proporzionale al numero di confronti tra elementi, avremo:

$$T(n) = O(\log n).$$

Le notazioni Θ e O sono impiegate per esprimere la complessità di un algoritmo. La notazione Ω è invece necessaria per definire il *limite inferiore* alla *complessità di un problema*, cioè il tempo minimo necessario per *qualsiasi* algoritmo di soluzione.

Per esempio la moltiplicazione tra due interi a, b si può eseguire in tempo $O(n^2)$ con l'algoritmo `Molt`: questo rappresenta un *limite superiore* alla complessità del problema nel senso che ci permette di escludere l'impiego di algoritmi di complessità maggiore. Poiché però per ottenere il prodotto si deve esaminare l'intera sequenza che rappresenta a e b, che ha lunghezza proporzionale a n, un limite inferiore per il problema è dato da $\Omega(n)$. In sostanza i limiti inferiori si riferiscono ai problemi, i limiti superiori agli algoritmi di soluzione.

Lo studio dei limiti inferiori è in genere difficile se si cercano risultati meno ingenui di quello appena citato. Ci limitiamo a un'osservazione semplice ma importante. Per un dato problema $\mathcal{P}$, poniamo che L_s sia un limite superiore (relativo a un algoritmo A) e L_i sia un limite inferiore (relativo al problema stesso). Se $L_s = L_i$ l'algoritmo A si dice *ottimo*, ovviamente in ordine di grandezza: in linea di principio non è necessario cercare ulteriori algoritmi per $\mathcal{P}$ perché non sarebbero migliori di A. Per eseguire la moltiplicazione esistono algoritmi migliori di `Molt`, ma non esiste dimostrazione che alcuno di essi sia ottimo: il problema in sostanza è ancora aperto.

3.3 Algoritmi polinomiali e esponenziali

Nel paragrafo precedente abbiamo esaminato due algoritmi per la ricerca di un elemento in un insieme ottenendo, per il caso pessimo, tempi di $O(n)$ e $O(\log n)$. Tra queste funzioni vi è un divario che cresce grandemente all'aumentare di n. Ponendole a confronto, e assegnando per esempio la base 2 al logaritmo, abbiamo:

n:	1	2	4	8	16	...	1.024	...	1.048.576	...	1.073.741.824
$\log_2 n$:	0	1	2	3	4	...	10	...	20	...	30

In sostanza se il valore di n raddoppia il valore di $\log_2 n$ aumenta solo di uno. Questo dimostra che la ricerca binaria è estremamente più efficiente della ricerca sequenziale a meno che i valori di n siano molto piccoli, e spiega la comune esperienza che cercare un nome in un elenco telefonico di 100.000 o 200.000 utenti richiede praticamente lo stesso tempo.

Se si inverte la funzione logaritmo, cioè si passa da n a 2^n, si ottiene esattamente lo stesso effetto: la funzione esponenziale 2^n cresce incomparabilmente più in fretta della funzione lineare n, e crea problemi enormi se riferita alla complessità di un algoritmo. Nella terminologia corrente un algoritmo è detto *polinomiale* se la sua complessità $T(n)$ è di ordine $O(p(n))$, ove $p(n)$ è un polinomio arbitrario nella dimensione n dell'ingresso: tutti gli algoritmi esaminati fin qui sono polinomiali. Se un limite

superiore polinomiale non esiste l'algoritmo è detto *esponenziale*, anche se la famiglia viene così a comprendere alcune funzioni "intermedie", come $n^{\log n}$, che in genere non si presentano nello studio della complessità. Tipiche funzioni esponenziali in cui potremo imbatterci sono $T(n) = \Theta(c^n)$ con $c > 1$ costante, $T(n) = \Theta(n^n)$, $T(n) = \Theta(n!)$; a proposito della terza ricordiamo l'esponenzialità della funzione fattoriale attraverso l'*approssimazione di Stirling*:

$$n! \simeq \sqrt{2\,\pi\,n}\,(n/e)^n \tag{3.2}$$

ove e è la base dei logaritmi naturali.

La distinzione, sopra ogni altra, tra le famiglie di algoritmi polinomiali e esponenziali è giustificata dalla pratica inconfrontabilità tra le leggi di crescita delle loro complessità. Per fare un grossolano esempio quantitativo poniamo che tre algoritmi A_1, A_2 e A_3 risolvano lo stesso problema su n dati con un numero di operazioni rispettivamente di n, n^2 e 2^n, e che il calcolatore esegua un miliardo di operazioni al secondo. Per $n = 80$ i calcoli sono portati a termine da A_1 in $8 \cdot 10^{-8}$ secondi, da A_2 in $6.4 \cdot 10^{-6}$ secondi, mentre A_3 richiede $\sim 10^{15}$ secondi ovvero trenta milioni di anni. L'algoritmo esponenziale è perciò utilizzabile solo per piccolissimi valori di n.

La differenza tra algoritmi polinomiali e esponenziali appare altrettanto drastica se si studia la dimensione dei dati praticamente trattabili in funzione dell'incremento di velocità dei calcolatori. Si potrebbe infatti pensare che quello che non è fattibile oggi possa esserlo in futuro, ma un semplice calcolo induce a credere che ciò non sia vero.[6] Poniamo che un nuovo calcolatore esegua ogni operazione in un tempo pari a $1/k$ dell'attuale e che l'utente abbia a disposizione sempre lo stesso tempo di calcolo $\bar{t}$, e vediamo come varia la dimensione dei dati trattabili, che passerà da n_1 a n_2. Possiamo grossolanamente assumere che impiegare il nuovo calcolatore per un tempo $\bar{t}$ equivalga a impiegare il vecchio per un tempo $k\,\bar{t}$. Per un algoritmo polinomiale che risolve il problema in $c\,n^s$ secondi (con c ed s costanti) si ha:

$$c\,n_1^s = \bar{t},\ c\,n_2^s = k\,\bar{t}\ \ \Rightarrow\ n_2 = \sqrt[s]{k}\,n_1$$

con vantaggio tanto maggiore quanto più è basso il valore di s, cioè quanto migliore è l'algoritmo. Per esempio impiegando l'algoritmo quadratico A_2 discusso prima, un calcolatore mille volte più veloce consente di moltiplicare per $\sqrt{1000} \simeq 30$ il numero di dati trattabili a pari tempo di calcolo. Se però l'algoritmo scelto è esponenziale e risolve il problema in $c\,2^n$ secondi, si ha:

[6] Questa affermazione potrebbe essere smentita dall'entrata in funzione di *computer quantistici* che per ora non sono disponibili né probabilmente lo saranno in un vicino orizzonte di tempo. Torneremo su questo argomento nel capitolo conclusivo perché l'eventuale disponibilità di queste macchine richiederebbe una sostanziale rivisitazione dei codici crittografici oggi in uso.

$$c\,2^{n_1} = \bar{t},\ c\,2^{n_2} = k\,\bar{t} \ \Rightarrow\ 2^{n_2} = k\,2^{n_1} \ \Rightarrow\ n_2 = n_1 + \log_2 k.$$

In questo caso il beneficio derivante dall'incremento di velocità è semplicemente *additivo* e cresce inoltre con estrema lentezza poiché ridotto dalla funzione logaritmo. Per l'algoritmo esponenziale A_3 discusso prima, un calcolatore un milione di volte più veloce consente solo di sommare $\log_2 1.000.000 \simeq 20$ al numero di dati trattabili a pari tempo di calcolo. In sostanza un algoritmo esponenziale in pratica *non consente* di trattare dati di dimensioni anche modeste, e similmente si dimostra che nessun aiuto può venire dal calcolo parallelo o distribuito. Dunque dovremmo evitare di costruire algoritmi esponenziali: ma questo, come vedremo, non sempre è possibile.

Nella maggioranza dei problemi che si affrontano con i calcolatori l'esponenzialità è legata all'impiego, a volte mascherato, di due strutture combinatorie di base: l'insieme delle 2^n configurazioni di n bit, e l'insieme delle $n!$ permutazioni di n elementi. Vediamo come tali strutture possano essere costruite algoritmicamente e quindi utilizzate. Dato un vettore A di lunghezza n, il seguente algoritmo `Configurazioni` costruisce in A le 2^n possibili configurazioni binarie (disposizioni con ripetizione dei due elementi 0 e 1 in gruppi di n). Per ogni configurazione l'algoritmo stabilisce se essa possiede determinate proprietà mediante una procedura `Controllo` che specificheremo nel seguito trattando di problemi concreti. Il calcolo è avviato con la chiamata iniziale `Configurazioni(`$A, 1$`)` che innesca la costruzione di tutte le configurazioni binarie dalla $00\ldots0$ alla $11\ldots1$, facendo variare le cifre a partire dall'ultima.

```
Procedure Configurazioni(A, k):
    for i ← 0 to 1 do
        A[k] ← i;
        if k = n then Controllo(A)
                 else Configurazioni(A, k + 1).
```

Se si vuole, la procedura `Configurazioni` genera tutti i numeri binari di n cifre e li "controlla" uno a uno per verificarne determinate proprietà. Questo suggerisce un immediato impiego dell'algoritmo in campo numerico. Consideriamo per esempio il *problema della primalità* $\mathcal{P}_{primo}(N)$ che come vedremo è essenziale in molte applicazioni crittografiche: dato un intero positivo N stabilire se è primo. Rappresentiamo N con n cifre binarie. Se N non è primo uno dei suoi divisori deve essere $\leq \sqrt{N}$, e si può quindi risolvere il problema provando se N è divisibile per tutti gli interi compresi tra 2 e $\sqrt{N}$, cioè rappresentati da sequenze binarie di lunghezza al massimo $n/2$. Si può allora impiegare l'algoritmo `Configurazioni` combinato con una procedura `Controllo` che, per ogni numero generato R, verifica se N è divisibile per R

e in caso positivo arresta il calcolo con successo. Questo algoritmo ha complessità $O(2^{n/2} \cdot D(n))$, ove $D(n)$ è il costo (polinomiale) della divisione, ed è quindi esponenziale nella lunghezza della rappresentazione di N. Poiché la crittografia opera su numeri binari di migliaia di cifre dovremo ricorrere ad altri metodi per stabilire la primalità.

Vediamo ora come la costruzione delle configurazioni binarie intervenga in problemi combinatori. Consideriamo il seguente *problema dello zaino* $\mathcal{P}_{zaino}(I, b, c)$, paradigma di riferimento per molti problemi di allocazione di risorse: dato un insieme I di n interi positivi (non necessariamente diversi tra loro), e altri due interi positivi b, c, stabilire se esiste un sottoinsieme di I la cui somma degli elementi sia compresa tra b e c. Intuitivamente c è il massimo peso che può sopportare lo zaino, I un insieme di oggetti di pesi vari, b il peso complessivo minimo che si vuole caricare.[7] Poniamo che gli elementi di I siano memorizzati in un vettore E e introduciamo un vettore binario A *di appartenenza*, anch'esso di n elementi, per descrivere arbitrari sottoinsiemi $T \subseteq I$: per $1 \leq i \leq n$, $A[i] = 1$ indica che $E[i] \in T$, $A[i] = 0$ indica che $E[i] \notin T$. Poniamo per esempio:

E:	5	8	12	6	6	7	25	4	2	9
A:	0	1	0	1	0	1	0	0	1	0

Il vettore E corrisponde a un'istanza del problema dello zaino per $n = 10$ elementi, e il vettore A corrisponde al sottoinsieme di elementi $T = \{8, 6, 7, 2\}$. Ponendo $b = 20$ e $c = 24$ la configurazione binaria contenuta in A specifica una soluzione, poiché la somma degli elementi di T è pari a 23.

Poiché le configurazioni di A corrispondono ai sottoinsiemi di I (che sono infatti 2^n), esse possono essere costruite con l'algoritmo `Configurazioni`. Per risolvere il problema $\mathcal{P}_{zaino}(I, b, c)$ è allora sufficiente specificare la procedura di controllo come:

`Procedure Controllo(`A`)`:

if $b \leq \sum_{i=1}^{n} A[i]E[i] \leq c$ **then success**.

ove il comando **success** arresta il calcolo decretando che esiste una soluzione (se il comando non viene incontrato, sono state provate senza successo tutte le configurazioni e la soluzione non esiste). La complessità dell'algoritmo così costruito è $T(n) = O(2^n \cdot S(n))$, ove $S(n)$ è il costo (polinomiale) delle operazioni richieste

[7] Questa versione del problema è un po' semplificata rispetto a quella classica, ma è perfettamente adatta ai nostri scopi.

nella sommatoria, ed è quindi esponenziale nel numero di elementi di I, cioè nella dimensione dell'ingresso.

Il problema $\mathcal{P}_{zaino}(I, b, c)$ è stato posto in forma decisionale perché, come vedremo, questa forma è sufficiente a decretarne il grado di difficoltà. In pratica si può richiedere di comunicare all'esterno la soluzione trovata, o la soluzione *ottima* (cioè il sottoinsieme di somma massima che può essere contenuto nello zaino). Si tratta di versioni standard del problema, che si incontrano per esempio nell'allocazione di file in un disco. Lo schema di calcolo è sempre lo stesso e la complessità è ovviamente sempre esponenziale; per esempio la soluzione ottima si determina ponendo inizialmente $max \leftarrow 0$, quindi chiamando la procedura `Configurazioni` e specificando la procedura di controllo come:

`Procedure Controllo(`A`)`:

if $(b \leq \sum_{i=1}^{n} A[i]E[i] \leq c)$ **and** $(max < \sum_{i=1}^{n} A[i]E[i])$
then $max \leftarrow \sum_{i=1}^{n} A[i]E[i]$;
for $i \leftarrow 1$ **to** n **do** $\bar{A}[i] \leftarrow A[i]$.

Il connettivo **and** tra due condizioni indica che entrambe devono essere verificate per eseguire la parte **then**. Il valore finale della variabile max è pari alla somma degli elementi nella soluzione ottima, e questa è registrata nel vettore di servizio $\bar{A}$. Se $max = 0$ non esiste alcuna soluzione.

La soluzione proposta per il problema dello zaino può apparire irragionevole in quanto non sfrutta alcuna proprietà del problema ma procede per enumerazione completa dei casi. Ed è proprio da questa enumerazione che discende l'esponenzialità dell'algoritmo. Alcune considerazioni locali possono far diminuire il numero di casi da esaminare: per esempio nell'istanza riportata sopra l'elemento 25 potrebbe essere eliminato prima di procedere perché è maggiore di $c = 24$. Ma nella grande maggioranza delle situazioni, e certamente in quella più sfavorevole, il numero di configurazioni possibili rimane esponenziale in n e non si conosce strategia migliore che esaminarle tutte. In sostanza, come vedremo nel prossimo paragrafo, il problema stesso sembra avere complessità esponenziale anche se non è mai stato dimostrato un tale limite inferiore.

Un altro problema combinatorio di base è quello della costruzione di permutazioni. Dato un insieme di n elementi contenuti in un vettore P, l'algoritmo seguente costruisce in P tutte le $n!$ permutazioni di tali elementi. Come nel caso precedente, per ogni permutazione l'algoritmo stabilisce se essa possiede determinate proprietà mediante una procedura `Controllo`.

```
Procedure Permutazioni(P, k):
    if k = n then Controllo(P)
            else for i ← k to n do
                scambia P[k] ↔ P[i];
                Permutazioni(P, k + 1);
                scambia P[k] ↔ P[i].
```

Il calcolo è avviato con la chiamata iniziale `Permutazioni(`$P, 1$`)`. L'algoritmo è basato sull'osservazione che le permutazioni di n elementi possono essere divise in gruppi, ponendo in ciascun gruppo quelle che iniziano con il primo, il secondo,..., l'n-esimo elemento ciascuno seguito dalle permutazioni degli altri $n-1$. Nel primo gruppo troviamo $P[1]$ seguito da tutte le permutazioni di $P[2], \ldots, P[n]$; nel secondo troviamo $P[2]$ seguito da tutte le permutazioni di $P[1], P[3], \ldots, P[n]$ e così via. Questa definizione è induttiva e la correttezza dell'algoritmo si può quindi dimostrare per induzione: occorre solo notare che la seconda operazione di scambio tra $P[k]$ e $P[i]$ è necessaria per ripristinare l'ordine degli elementi dopo la costruzione ricorsiva delle permutazioni degli elementi che seguono il primo. Più difficile è capire il funzionamento dell'algoritmo e individuare l'ordine in cui vengono costruite le permutazioni. Invitiamo il lettore a rifletterci un momento simulando a mano il comportamento dell'algoritmo sull'insieme $\{1, 2, 3\}$: otterrà nell'ordine le permutazioni: 1,2,3 - 1,3,2 - 2,1,3 - 2,3,1 - 3,2,1 - 3,1,2, e con l'ultima operazione di scambio ripristinerà l'ordine iniziale 1,2,3 degli elementi.

La costruzione di permutazioni interviene per esempio nel *problema del ciclo hamiltoniano* $\mathcal{P}_{ham}(G)$, paradigma di riferimento per molti problemi di percorsi. Ricordiamo anzitutto che un grafo $G = (V, E)$ è composto da un insieme V di vertici, e un insieme E di spigoli che connettono coppie di vertici. La struttura può essere convenientemente descritta rappresentando i vertici con gli interi $1, \ldots, n$, con $n = |V|$, e rappresentando E con una matrice binaria A *di adiacenza*, di dimensioni $n \times n$, ove si pone $A[i, j] = 1$ se esiste uno spigolo che connette il vertice i al vertice j, $A[i, j] = 0$ se tale spigolo non esiste. La dimensione della descrizione di G è dunque di ordine $\Theta(n \log n)$ per rappresentare V più $\Theta(n^2)$ per rappresentare E: quindi in totale è di ordine $\Theta(n^2)$. Un *percorso* in un grafo è una sequenza di vertici $v_1, v_2, \ldots, v_k$ tale che esiste lo spigolo da v_i a v_{i+1} per ogni coppia di vertici consecutivi nel percorso. Se v_k coincide con v_1 il percorso è detto *ciclo*. Dato un grafo arbitrario G il problema $\mathcal{P}_{ham}(G)$ chiede di stabilire se esiste un ciclo in G che contiene tutti i vertici esattamente una volta.

Se associamo al grafo G un vettore P contenente gli interi tra 1 e n, una permutazione di P rappresenta una sequenza che contiene tutti i vertici: la permutazione rappresenta quindi una soluzione di $\mathcal{P}_{ham}(G)$ (cioè un ciclo hamiltoniano) se nella

matrice di adiacenza di A si ha $A[P[i], P[i+1]] = 1$ per ogni $1 \leq i \leq n-1$, e $A[P[n], P[1]] = 1$. Si possono allora costruire tutte le permutazioni in P con l'algoritmo `Permutazioni` e risolvere $\mathcal{P}_{ham}(G)$ specificando la procedura di controllo come:

`Procedure Controllo(`P`)`:
if $(A[P[i], P[i+1]] = 1, 1 \leq i \leq n-1)$ **and** $(A[P[n], P[1]] = 1)$
then success.

Come visto in precedenza il comando **success** arresta il calcolo decretando che esiste una soluzione. La complessità dell'algoritmo così costruito è $T(n) = \mathrm{O}(n \cdot n!)$ ove il fattore n nel prodotto rappresenta il tempo per il controllo nella matrice A, ed è quindi esponenziale nella dimensione dell'ingresso.

Il problema $\mathcal{P}_{ham}(G)$ è stato posto in forma decisionale ma in pratica si può richiedere di comunicare all'esterno la soluzione trovata, o una soluzione con determinate proprietà. In una sua famosa versione che prende il nome di *problema del commesso viaggiatore* $\mathcal{P}_{cv}(G)$, si assegna una *lunghezza* a ogni spigolo del grafo e si chiede di trovare il ciclo hamiltoniano (se ne esiste uno) di minima lunghezza complessiva: si tratta per esempio di stabilire il percorso più breve per la testa di un trapano a controllo numerico con cui si devono fare molti fori in un pezzo meccanico. Lo schema di calcolo è sempre lo stesso e la complessità è ovviamente sempre esponenziale: anche $\mathcal{P}_{ham}(G)$ non sembra risolubile in tempo polinomiale, ma non si conosce dimostrazione in proposito.

3.4 Le classi P, NP, co-NP e NPC

Dopo aver visto come risolvere alcuni problemi in tempo polinomiale e aver proposto per altri solo algoritmi esponenziali, dobbiamo fare un po' di ordine nella teoria. Questo ci aiuterà a impostare organicamente i problemi anche se lascerà aperte alcune questioni fondamentali per cui mancano risultati definitivi.

Anzitutto ogni ente di cui trattiamo dovrà essere rappresentato in modo efficiente nel senso tecnico del termine (capitolo 2); cioè i dati, gli algoritmi, e ogni struttura che questi possano costruire durante il calcolo avranno dimensione logaritmica nel numero di elementi della classe cui appartengono. In particolare un numero N sarà rappresentato con $\Theta(\log N)$ caratteri, o precisamente con $\lceil \log_2 N \rceil$ cifre binarie. Un insieme di n elementi sarà rappresentato con $\Theta(n)$ caratteri nell'ipotesi che ogni elemento abbia dimensione costante d. Notiamo però che in questo modo si possono rappresentare al più a^d elementi distinti, ove a è la cardinalità dell'alfabeto. Poiché

il valore di a^d per quanto grande è costante, all'aumentare di n l'insieme conterrà necessariamente elementi ripetuti: se si vuole evitare questa situazione ogni elemento dovrà essere rappresentato con $\Theta(\log n)$ caratteri portando a $\Theta(n \log n)$ la dimensione complessiva dell'insieme. L'ipotesi di rappresentazione efficiente degli enti in gioco è implicita e non sarà più ripetuta.

Abbiamo affermato che l'inerente difficoltà di un problema è in genere già presente nella sua forma decisionale che chiede di rispondere a un quesito binario sull'istanza di volta in volta considerata. Concretamente anziché *determinare* una soluzione relativa all'istanza si chiede di stabilire se esiste una soluzione che goda di una data proprietà: in caso affermativo diremo che l'istanza è *accettabile*. La teoria si rivolge inizialmente ai problemi decisionali dividendoli in due classi, dette **P** e **NP**, in relazione alla complessità degli algoritmi di soluzione. La prima definizione è semplice:

Definizione. **P** è la classe dei problemi decisionali risolubili con un algoritmo polinomiale.

Per esempio $\mathcal{P}_{ric}(k, I) \in \mathbf{P}$. Parimenti è in **P** il problema $\mathcal{P}_{coprimi}(N, M)$ che chiede di stabilire se due interi N, M sono primi tra loro, perché si può calcolare *mcd(N,M)* con l'algoritmo polinomiale `Euclid(`N, M`)`. Non sappiamo invece ancora se il problema $\mathcal{P}_{primo}(N)$ (stabilire se un intero N è primo) appartiene a **P** perché abbiamo indicato solo un algoritmo esponenziale per risolverlo.

La definizione della seconda classe è molto più sottile. Un problema $\mathcal{P}$ per cui non si conosce un algoritmo polinomiale è praticamente irrisolubile al crescere della lunghezza della sequenza x che rappresenta i suoi dati d'ingresso. È però possibile che un'informazione addizionale y, detta *certificato* di x, consenta di risolvere più efficientemente $\mathcal{P}$ attraverso un *algoritmo di verifica* $\mathcal{A}(x, y)$. Perché il discorso non appaia troppo astratto consideriamo come esempio l'istanza del problema $\mathcal{P}_{zaino}(I, b, c)$ presentata nel paragrafo precedente: la sequenza d'ingresso x rappresenta l'insieme $I = \{5, 8, 12, 6, 6, 7, 25, 4, 2, 9\}$ e i limiti $b = 20$, $c = 24$. Abbiamo visto che il sottoinsieme $T = \{8, 6, 7, 2\}$ è una soluzione che soddisfa la condizione dello zaino, quindi l'esistenza di T dimostra che l'istanza x è accettabile. Orbene in questo caso un certificato y è la stessa sequenza che rappresenta T, poiché esiste un algoritmo di verifica che scandisce $y = T$, calcola la somma dei suoi elementi, confronta tale somma con i valori di b e c in $x = I$, e stabilisce così che per x esiste una soluzione.

Affinché questo approccio abbia interesse dobbiamo chiarire alcuni punti importanti. Anzitutto abbiamo affermato che l'esistenza di un certificato y e di un algoritmo $\mathcal{A}$ di verifica permette di attestare che l'istanza x è accettabile: in questo caso porremo (convenzionalmente) $\mathcal{A}(x, y) = 1$; ma il risultato opposto $\mathcal{A}(x, y) = 0$ *non* indica che una soluzione per x non esiste, ma solo che y non ne è un certificato. Definire-

mo perciò il certificato solo per le istanze accettabili: in sostanza un certificato è un attestato di esistenza della soluzione, ma in genere non è facile costruire attestati di non esistenza. Inoltre, per un dato problema $\mathcal{P}$, è necessario che:

1. ogni istanza accettabile x di lunghezza n ammetta un certificato y di lunghezza polinomiale in n;

2. esista un algoritmo di verifica $\mathcal{A}$ *polinomiale* in n e applicabile a ogni coppia $\langle x, y\rangle$.

Se queste due condizioni sono valide si dice che $\mathcal{A}$ *verifica* $\mathcal{P}$ *in tempo polinomiale*. Possiamo allora porre la definizione:

Definizione. NP è la classe dei problemi decisionali verificabili in tempo polinomiale.[8]

Per quanto abbiamo già visto $\mathcal{P}_{zaino}(I, b, c) \in \mathbf{NP}$. Parimenti è in **NP** il problema $\mathcal{P}_{ham}(G)$; infatti, per ogni grafo G (instanza x) che ammette un ciclo hamiltoniano, esiste un certificato y che specifica per esempio la permutazione di vertici associata al ciclo.

Poiché in pratica un certificato contiene un'informazione molto prossima alla soluzione del problema, possiamo chiederci quale sia l'interesse di aver introdotto questo concetto. In fondo stiamo solo cercando di eludere la difficoltà: se la soluzione è (esponenzialmente) difficile da raggiungere ammettiamo di averla già! Il dubbio è assolutamente legittimo: come ora vedremo la teoria della verifica è utilissima per far luce sulle gerarchie di complessità dei problemi ma non aggiunge nulla alla possibilità di risolverli efficientemente.

Notiamo anzitutto che ogni problema $\mathcal{P} \in \mathbf{P}$ è banalmente verificabile in tempo polinomiale. Infatti si può costruire un certificato vuoto y per ogni sequenza d'ingresso x e definire un algoritmo $\mathcal{A}(x, y)$ che ignora y e verifica l'accettabilità di x risolvendo direttamente $\mathcal{P}$ in tempo polinomiale. Ne segue:

$$\mathbf{P} \subseteq \mathbf{NP}$$

Non si sa se $\mathbf{P} \neq \mathbf{NP}$ anche se vi sono forti motivi per ritenere che ciò sia vero. Dal punto di vista teorico questo è il principale problema aperto nello studio della complessità. In pratica si ammette che sia $\mathbf{P} \neq \mathbf{NP}$ fino a una (improbabile) prova contraria. Vi sono moltissimi problemi in **NP**, tra cui per esempio $\mathcal{P}_{zaino}$ e $\mathcal{P}_{ham}$,

[8] L'acronimo **NP** sta per "nondeterministico polinomiale", ed è nato nell'ambito del calcolo non deterministico ove questa teoria ha avuto origine.

per cui non è noto alcun algoritmo polinomiale di soluzione e si ritiene che questa situazione non sia destinata a cambiare. In gergo i problemi in **P** sono detti *trattabili* e i problemi in **NP - P** sono detti *intrattabili*, anche se questi ultimi, come vedremo, presentano presumibilmente diversi gradi di "difficoltà".

Un altro punto importante è la profonda differenza tra certificare l'esistenza (compito possibilmente semplice) o la non esistenza (compito spesso difficile) di una soluzione, come si può ben comprendere su un esempio. Il problema $\mathcal{P}_{ham}(G)$ può essere risolto con un algoritmo enumerativo esponenziale e, come abbiano visto, si può facilmente indicare un certificato polinomiale per esso. Il problema $\mathcal{P}_{non-ham}(G)$, che chiede di stabilire se G *non* possiede alcun ciclo hamiltoniano può ovviamente essere risolto con lo stesso algoritmo estendendo l'enumerazione a tutte le possibili permutazioni di vertici e dichiarando quindi la non esistenza del ciclo, ma non è semplice individuare per esso un certificato che indichi direttamente questa proprietà.

La questione generale si pone nei seguenti termini. Per ogni problema decisionale $\mathcal{P}$ si definisce *problema complementare* $\bar{\mathcal{P}}$ quello per cui sono accettabili le istanze non accettabili per $\mathcal{P}$, e viceversa. Per esempio $\mathcal{P}_{ham}(G)$ e $\mathcal{P}_{non-ham}(G)$ sono l'uno complementare dell'altro. Si pone quindi:

Definizione. co-NP è la classe dei problemi decisionali $\mathcal{P}$ per cui $\bar{\mathcal{P}} \in$ **NP**.

Poiché $\mathcal{P}_{ham}(G) \in$ **NP** si ha $\mathcal{P}_{non-ham}(G) \in$ **co-NP**. Con ragionamento analogo ai precedenti si vede anche immediatamente che **P** $\subseteq$ **co-NP**, quindi **P** $\subseteq$ **NP** $\cap$ **co-NP**; ma non si sa in che relazione **NP** e **co-NP** stiano tra loro, anche se si ammette che sia **NP** $\neq$ **co-NP**.[9]

Veniamo ora a un concetto che ci permetterà di aggiungere un tassello importante alla teoria. Si dice che un problema $\mathcal{P}_1$ *si riduce in tempo polinomiale* a un problema $\mathcal{P}_2$, e si scrive $\mathcal{P}_1 \leq_p \mathcal{P}_2$, se esiste un algoritmo polinomiale $\mathcal{R}$ che trasforma ogni istanza x_1 di $\mathcal{P}_1$ in un'istanza x_2 di $\mathcal{P}_2$ in modo che x_1 è accettabile in $\mathcal{P}_1$ se e solo se x_2 è accettabile in $\mathcal{P}_2$. L'esistenza di un algoritmo di riduzione $\mathcal{R}$ permette di risolvere $\mathcal{P}_1$ in tempo polinomiale se si sa risolvere $\mathcal{P}_2$ in tempo polinomiale: infatti per ogni istanza x_1 si può calcolare la corrispondente istanza $x_2 = \mathcal{R}(x_1)$ e poi risolvere $\mathcal{P}_2$ su questa. Possiamo così definire una fondamentale sottoclasse di **NP**:

Definizione. NPC è la classe dei problemi decisionali $\mathcal{P}$ tali che

$\mathcal{P} \in$ **NP**;

per ogni $\mathcal{P}' \in$ **NP** si ha $\mathcal{P}' \leq_p \mathcal{P}$.

[9] Si può dimostrare che la congettura **NP** $\neq$ **co-NP** implica la **P** $\neq$ **NP** ma non viceversa. Potrebbero perciò valere contemporaneamente **P** $\neq$ **NP** e **NP** = **co-NP**.

I problemi in **NPC** sono detti *NP-completi* a indicare che possiedono tutte le (peggiori) caratteristiche della classe **NP**. Essi sono infatti i "più difficili" problemi di quella classe poiché se si scoprisse un algoritmo polinomiale per risolverne uno, tutti i problemi in **NP** potrebbero essere risolti in tempo polinomiale attraverso l'algoritmo di riduzione. D'altra parte se si scoprisse che per risolvere anche uno solo dei problemi in **NP** è indispensabile un tempo esponenziale, tutti i problemi in **NPC** richiederebbero tale tempo e si sarebbe dimostrata la congettura $\mathbf{P} \neq \mathbf{NP}$.

Un caposaldo nella teoria della complessità fu posto con l'individuazione del primo problema *NP*-completo: successivamente attraverso il processo di riduzione sono stati riconosciuti come *NP*-completi moltissimi altri problemi, tra cui per esempio $\mathcal{P}_{zaino}(I, b, c)$ e $\mathcal{P}_{ham}(G)$. Altri problemi sono in una situazione intermedia perché o non ne è stata ancora dimostrata l'appartenenza a **NPC**, o la loro appartenenza a tale classe implicherebbe che $\mathbf{P} = \mathbf{NP}$ ed è quindi ritenuta estremamente improbabile: in effetti essi potrebbero essere "più facili" dei problemi *NP*-completi anche se non si conosce ancora un algoritmo polinomiale per risolverli. Tra questi ultimi è stato a lungo compreso il problema $\mathcal{P}_{primo}(N)$ particolarmente rilevante in crittografia: nel 2002 è stato individuato un algoritmo polinomiale per risolverlo senza creare alcun turbamento alla gerarchia delle classi di complessità.

Complessivamente le classi di complessità di cui abbiamo trattato si trovano nella situazione riportata nella figura 3.1 sotto le ipotesi $\mathbf{P} \neq \mathbf{NP}$, $\mathbf{NP} \neq$ **co-NP** e $\mathbf{P} \neq \mathbf{NP} \cap$ **co-NP**.

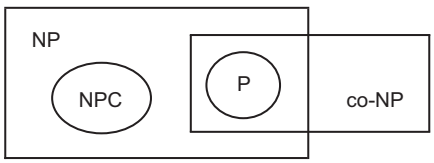


Figura 3.1: Relazione tra classi di complessità nell'ipotesi $\mathbf{P} \neq \mathbf{NP}$, $\mathbf{NP} \neq$ **co-NP**, $\mathbf{P} \neq \mathbf{NP} \cap$ **co-NP**. Esempi di appartenenza: $\mathcal{P}_{ric} \in \mathbf{P}$; $\mathcal{P}_{coprimi} \in \mathbf{P}$; $\mathcal{P}_{zaino} \in \mathbf{NPC}$; $\mathcal{P}_{ham} \in \mathbf{NPC}$.

Questo studio deve essere approfondito nei testi citati in bibliografia: poiché però abbiamo definito le classi di complessità solo per problemi decisionali, vediamo come si possono collocare in questa teoria tutti i problemi. Poniamo dunque un'ultima definizione:

Definizione. **NPH** è la classe dei problemi $\mathcal{P}$ tali che per ogni $\mathcal{P}' \in \mathbf{NP}$ si ha $\mathcal{P}' \leq_p \mathcal{P}$.

I problemi in **NPH** sono detti *NP-hard*, o *NP-ardui* in italiano: per essi non è richiesta l'appartenenza a **NP**, quindi non sono necessariamente problemi decisionali. Confrontando le definizioni di **NPC** e **NPH** si vede subito che per stabilire se un problema $\mathcal{P}$ è *NP*-arduo è sufficiente controllare che un qualunque problema *NP*-completo si riduca a $\mathcal{P}$. Per esempio si considerino il problema *NP*-completo $\mathcal{P}_{zaino}(I, b, c)$ e la sua versione $\mathcal{P}_{zaino.max}(I, c)$ che chiede di determinare il sottoinsieme di I di somma s massima con l'unica condizione che sia $s \leq c$. Si tratta per esempio di riempire al meglio un disco di capacità c con file tratti da un insieme I. Dalla soluzione di $\mathcal{P}_{zaino.max}(I, c)$ si ricava immediatamente (e polinomialmente) una soluzione per $\mathcal{P}_{zaino}(I, b, c)$ poiché l'istanza I, b, c di questo è accettabile se e solo se $s \geq b$. Dunque il problema $\mathcal{P}_{zaino.max}(I, c)$ è *NP*-arduo. Similmente si dimostra che il problema del commesso viaggiatore $\mathcal{P}_{cv}(G)$ è *NP*-arduo.

I problemi che si incontrano nella pratica richiedono di *determinare* una soluzione con particolari proprietà e non semplicemente di stabilirne l'esistenza. Un problema *NP*-arduo $\mathcal{P}$ è in genere "più difficile" del corrispondente problema decisionale $\mathcal{P}_D$ nel senso che un eventuale algoritmo polinomiale per risolvere $\mathcal{P}$ permetterebbe di risolvere anche $\mathcal{P}_D$ in tempo polinomiale attraverso la trasformazione delle istanze da $\mathcal{P}_D$ a $\mathcal{P}$ e dei risultati da $\mathcal{P}$ a $\mathcal{P}_D$. Quindi *se $\mathcal{P}_D$ è intrattabile anche $\mathcal{P}$ è intrattabile.* È comunque possibile che $\mathcal{P}$ sia "non più difficile" di $\mathcal{P}_D$, nel senso che ciascun problema si possa trasformare nell'altro in tempo polinomiale. In conclusione tutti i problemi in **NPH** sono "non più facili" di quelli in **NP**, ma possono essere "molto" più difficili: per esempio possono non essere verificabili in tempo polinomiale o addirittura essere indecidibili. Consideriamo un caso limite con cui concluderemo questa carrellata sulle classi di complessità.

Prendiamo nuovamente in esame il problema $\mathcal{P}_{diof}(E)$ delle equazioni diofantee che chiede di decidere se l'equazione algebrica E, a coefficienti interi, di grado arbitrario e in un numero arbitrario di variabili, ammette una soluzione in cui tutte le variabili hanno valori interi. Nel capitolo 2 abbiamo visto che $\mathcal{P}_{diof}(E)$ è algoritmicamente indecidibile, ma questo non implica che siano indecidibili alcune sue forme particolari. Immaginiamo che le variabili in E siano due, indicate con x e y, e che l'equazione abbia la forma lineare $ax + by + c = 0$, o la forma quadratica $ax^2 + by + c = 0$ (cioè rappresenti rispettivamente una retta o una parabola nel piano x, y). Indicheremo con $\mathcal{P}_{diof.lin}(E)$ e $\mathcal{P}_{diof.quad}(E)$ le forme assunte rispettivamente dal problema. I dati sono costituiti nei due casi dai coefficienti $a, b, c,$ e ponendo che questi siano dello stesso ordine di grandezza avremo una dimensione n dell'ingresso logaritmica nella descrizione di ciascuno di essi.

Lasciamo al lettore il compito di dimostrare che l'equazione $ax + by + c = 0$ ammette una soluzione intera (cioè la retta corrispondente passa per un punto del

piano a coordinate intere) se e solo se c è divisibile per *mcd(a,b)*.[10] Per esempio l'equazione $2x - 6y + 10 = 0$ ammette la soluzione intera $x = -2$, $y = 1$; ma la $2x - 6y + 9 = 0$ non ammette soluzioni intere. Poiché il calcolo di *mcd(a,b)* e di $c/$*mcd(a,b)* si può eseguire in tempo polinomiale in n, abbiamo $\mathcal{P}_{diof.lin}(E) \in \mathbf{P}$.

Il problema $\mathcal{P}_{diof.quad}(E)$ è più complesso. Il lettore potrà facilmente dimostrare che, in questo caso, se E ammette una soluzione intera con $x = \bar{x}$, allora ne ammette anche una con $x = \bar{x} + b$. Dunque si può risolvere il problema sostituendo in E tutti i valori interi di x tra 0 e $b - 1$ e controllando se, per uno di essi, y assume valore intero: E ammetterà una soluzione intera se e solo se ciò si verifica. Questo calcolo richiede tempo proporzionale al valore di b, quindi esponenziale in n: in effetti è possibile dimostrare che $\mathcal{P}_{diof.quad}(E)$ è un problema NP-completo e quindi non si può far meglio. Tuttavia è chiaro che un eventuale algoritmo $\mathcal{A}$ per risolvere $\mathcal{P}_{diof}(E)$ nella sua forma generale risolverebbe immediatamente anche $\mathcal{P}_{diof.quad}(E)$: dunque $\mathcal{P}_{diof}(E)$, benché indecidibile, è *NP*-arduo (e MOLTO più difficile dei problemi in **NP**).

[10] A tele scopo si utilizzi l'*identità di Bézout* che afferma che, per ogni coppia di interi a, b, esistono due interi $\bar{x}$, $\bar{y}$ tali che $a\bar{x} + b\bar{y} =$ *mcd(a,b)*.

Capitolo 4

Il ruolo del caso

Nello studio della complessità di calcolo abbiamo implicitamente ammesso che un algoritmo, pur richiedendo un tempo elevato di esecuzione, generi sempre un risultato corretto. Questa affermazione può tuttavia essere presa alla lettera solo come astrazione matematica: in pratica non si può escludere che un programma contenga un errore che eventualmente si manifesta solo per particolari dati d'ingresso e rimane quindi inosservato per molto tempo; o che il calcolatore stesso sia soggetto a sporadico malfunzionamento; o che il risultato sia interpretato scorrettamente. Dunque anche un algoritmo "sbagliato" può essere tollerato, o addirittura preferito ad algoritmi corretti ma molto più lenti, se esso genera risultati errati con probabilità inferiore a quella che si verifichino errori per altre cause.

Su questa considerazione si innesta lo sviluppo e l'analisi degli *algoritmi probabilistici* o *randomizzati*, il cui possibile comportamento erroneo è generato ad arte e misurato in senso probabilistico. Essi hanno un importante motivo di esistere se permettono di risolvere in tempo polinomiale problemi altrimenti esponenziali, con probabilità di errore arbitrariamente piccola e matematicamente misurabile. Alcuni algoritmi di questa famiglia sono impiegati in varie fasi dei processi crittografici, nei quali d'altra parte la sicurezza non può essere mai garantita in senso assoluto. Per costruirli è necessario far intervenire un elemento di aleatorietà nei loro passi computazionali e essere in grado di misurarne gli effetti: si alimenta l'algoritmo con una sequenza di valori casuali utilizzati come parametri del calcolo, e si analizza la correttezza dei risultati e la complessità computazionale in base alle proprietà della sequenza.

L'impiego di sequenze casuali interviene anche in un'altra importantissima fase dei processi crittografici. Come avremo modo di sottolineare più volte l'inviolabilità di un protocollo dipende fortemente da un'accurata scelta delle chiavi segrete impiegate negli algoritmi di cifratura e decifrazione. Il primo requisito di una chiave è quello di

non essere facilmente prevedibile da parte di un crittoanalista: per questo le chiavi devono essere generate con un procedimento casuale. È dunque alla possibilità di disporre di sequenze casuali, e al significato stesso del caso, che dobbiamo anzitutto rivolgerci.

4.1 Il significato algoritmico della casualità

In un famoso saggio filosofico sulla probabilità Laplace scriveva nel 1819:

"Nel gioco di testa o croce l'apparizione di testa per cento volte di seguito è considerata un evento straordinario perché le innumerevoli combinazioni che possono avverarsi in cento lanci successivi o formano sequenze regolari, nelle quali si può osservare una regola facile da comprendere, o formano sequenze irregolari che sono incomparabilmente più numerose."

Questa affermazione, affidata da Laplace all'intuizione dei matematici, ha trovato una caratterizzazione precisa solo dopo il 1960 nell'ambito della teoria algoritmica dell'informazione; teoria attribuita in genere al grande matematico russo Andrej Kolmogorov anche se altri studi indipendenti e coevi stavano conducendo alle stesse conclusioni.

L'esperimento di Laplace, rappresentato in notazione binaria, mette a confronto una sequenza composta solo di uni con un'altra in cui non si vede una regola precisa di generazione. Consideriamo le due sequenze di venti caratteri:

X: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Y: 1 1 0 1 0 0 1 0 1 1 1 0 0 1 0 1 0 1 0 0

Per ipotesi X e Y sono state generate con i lanci di una moneta e hanno la stessa probabilità di apparire (è una probabilità molto piccola, pari a 2^{-20}). Tuttavia esaminando le due sequenze siamo indotti a credere che la X non sia casuale perché una "regola" insita in essa ci permette di descriverla in modo semplice come "sequenza composta di 20 uni". Apparentemente nulla di simile si può invece dire sulla Y che per la sua irregolarità ci appare casuale. La definizione di sequenza casuale data da Kolmogorov si basa su una generalizzazione di queste osservazioni.

Scelto un alfabeto di c caratteri, numeriamo tutte le sequenze costruibili $X_0, X_1, X_2, ...$ secondo l'ordine canonico indicato nel capitolo 2. Per semplicità rappresentiamo gli indici in numerazione binaria. Sappiamo dal capitolo citato che la sequenza X_h ha

lunghezza $\simeq \log_c h$, e la sequenza binaria che rappresenta h ha lunghezza $\simeq \log_2 h$; quindi, per la relazione 3.1 che regola il cambio di base tra logaritmi, la lunghezza di entrambe le sequenze è di ordine $\Theta(\log h)$. Poiché i ragionamenti che seguono saranno basati unicamente sulla lunghezza delle sequenze, d'ora in avanti identificheremo ogni X_h con il suo indice h senza più farne menzione.

Poniamo che la rappresentazione di h consista di n uni. In un qualsiasi sistema di calcolo, per esempio uno specifico calcolatore con un suo linguaggio di programmazione, possiamo definire un semplicissimo algoritmo A che realizzi l'operazione: "genera n uni". La sequenza binaria che rappresenta A avrà lunghezza $m = \Theta(\log n)$ poiché l'unica parte variabile in A è il riferimento a n, a sua volta descritto con $\Theta(\log n)$ cifre binarie. La sequenza h ha invece lunghezza n, quindi molto maggiore della prima, e la differenza tra le due cresce enormemente al crescere di n. La "regola" indicata da Laplace è tradotta in un algoritmo di generazione di cui si può misurare la lunghezza; e siamo indotti a concludere, informalmente per ora, che *una sequenza binaria h è casuale se non ammette alcun algoritmo di generazione A la cui rappresentazione binaria sia più corta di h.* In sostanza il modo più economico per definire una sequenza casuale è assegnare la sequenza stessa. Notiamo che una siffatta definizione di casualità *richiede che esista* una regola semplice di descrizione per le sequenze non casuali e solo per queste, *ma non impone* che tali sequenze siano generate impiegando la regola: nell'esperimento di Laplace la sorgente non segue alcuna regola e la sequenza è frutto del caso.

Lo sviluppo rigoroso di questa teoria non è elementare e richiede anzitutto che si dimostri come la definizione di casualità sia indipendente dal sistema di calcolo in cui si assegnano gli algoritmi per la generazione delle sequenze. Ciò è regolato da un fondamentale teorema di invarianza di cui cercheremo di spiegare il significato senza addentrarci nei complessi dettagli tecnici. La teoria di riferimento è quella della calcolabilità mediante sistemi "ragionevoli" (macchine astratte o calcolatori reali, tutti computazionalmente equivalenti) introdotta nel capitolo 2. Poiché dobbiamo essere in grado di descrivere questi sistemi, essi sono numerabili e li indichiamo con $S_0, S_1, S_2, ...$ Un algoritmo per generare una sequenza binaria h in un sistema S_i è un "programma" per S_i, ed è anch'esso rappresentato da una sequenza binaria p: indichiamo questo fatto con: $S_i(p) = h$. Si definisce allora la *complessità* (secondo Kolmogorov) *della sequenza h rispetto al sistema di calcolo S_i* come:

$$\mathcal{K}_{S_i}(h) = min\{|p| : S_i(p) = h\} \tag{4.1}$$

ove $|p|$ indica la lunghezza della sequenza p. La complessità di una sequenza h all'interno di un sistema di calcolo è cioè la lunghezza del più corto programma che genera h. Si noti che se h non obbedisce ad alcuna legge semplice, il più corto programma

che la genera conterrà la sequenza stessa al suo interno e la genererà trasferendola nel risultato. Avremo in questo caso: $\mathcal{K}_{S_i}(h) = |h| + c$, dove c è una costante positiva che dipende dal sistema di calcolo e rappresenta la lunghezza della parte di programma necessaria per trasferire h nel risultato.

Ora tra i sistemi di calcolo ve n'è (almeno) uno S_u, detto *universale*, che può simulare il funzionamento di ogni altro se gli si fornisce la coppia $\langle i, p\rangle$, ove i è l'indice del sistema S_i da simulare e p è il programma di questo. La simulazione genera la stessa sequenza h, ovvero $h = S_i(p) = S_u(\langle i, p\rangle)$. Il tempo di calcolo non è rilevante ai nostri fini e può essere diversissimo nei due sistemi. Il programma $q = \langle i, p\rangle$ per S_u ha lunghezza $|q| = |p| + \lceil \log_2 i \rceil$ ove il termine $\lceil \log_2 i \rceil$ è indipendente da h. Quindi per qualsiasi h abbiamo:

$$\mathcal{K}_{S_u}(h) \leq \mathcal{K}_{S_i}(h) + c_i \tag{4.2}$$

con c_i costante positiva che dipende solo dal sistema S_i di riferimento. Si noti che la relazione 4.2 vale con il segno di uguaglianza per quelle sequenze h generate da S_u per simulazione di S_i non essendovi per S_u algoritmo più efficiente, e vale con il segno di minore stretto per le sequenze che possono essere generate con un programma più corto (per esempio per simulazione di un diverso sistema S_j). In conclusione per qualunque sistema di calcolo S_i la complessità di una sequenza h si può identificare con $\mathcal{K}_{S_u}(h)$ a meno di una costante, liberando il ragionamento dall'influenza del particolare sistema scelto. Trascureremo allora l'indice in $\mathcal{K}_{S_u}(h)$ e denoteremo con $\mathcal{K}(h)$ la *complessità* (secondo Kolmogorov) *della sequenza h.*

Potremmo a questo punto definire una sequenza h come casuale se $\mathcal{K}(h) \geq |h|$. Questa relazione è comunque considerata troppo restrittiva e si pone in genere la definizione: *una sequenza è casuale se* $\mathcal{K}(h) \geq |h| - \lceil \log_2 |h| \rceil$. Avvalora questa impostazione il fatto che le sequenze casuali cosí definite hanno le stesse proprietà matematiche delle sequenze generate da una sorgente casuale, ove ogni cifra può valere 0 o 1 con probabilità 1/2 indipendentemente dalle altre cifre della sequenza.

Chiediamoci ora quante siano le sequenze casuali tra tutte quelle possibili: vedremo ancora una volta che l'intuizione di Laplace: ... le sequenze irregolari sono incomparabilmente più numerose (di quelle regolari) ha un esatto riscontro algoritmico. Per ogni intero n vi sono $S = 2^n$ sequenze binarie lunghe n: poniamo che tra queste ve ne siano T non casuali e studiamo la relazione tra T e S. Complessivamente vi sono $N = 2^{n-\lceil \log_2 n \rceil} - 1$ sequenze più corte di $n - \lceil \log_2 n \rceil$, e tra queste vi sono tutti i programmi per generare le T sequenze non casuali lunghe n. Si ha dunque $T \leq N < S$, quindi esistono certamente sequenze casuali per ogni valore di n. Si ha inoltre: $T/S < 2^{-\lceil \log_2 n \rceil}$, funzione che tende a zero al crescere di n: quindi le sequenze casuali esistono e sono enormemente più numerose di quelle non casuali.

Ecco dunque una buona notizia per i crittografi che hanno continuo bisogno di sequenze casuali. Essi potrebbero infatti acciuffare sequenze dove capita e controllare se sono casuali, perché la probabilità che ciò si verifichi è molto alta. Ma come spesso accade, a una buona notizia se ne aggiunge subito una cattiva: il problema $\mathcal{P}_{rand}(h)$ che chiede di stabilire se una sequenza arbitraria h è casuale è un problema indecidibile nel senso illustrato nel capitolo 2; ovvero non può esistere un algoritmo per risolverlo.

Anche se non indispensabile nel seguito, riportiamo per la sua eleganza lo schema di dimostrazione della indecidibilità di $\mathcal{P}_{rand}(h)$ (il lettore non interessato può trascurare il resto del presente paragrafo). A tale scopo costruiremo una situazione autocontraddittoria nota come *paradosso di Berry* sugli interi. Poniamo che esista un algoritmo `Random`(h) per decidere se una sequenza h è casuale (`Random`(h)=1) oppure non lo è (`Random`(h)=0). Possiamo allora costruire un altro algoritmo `Paradosso` che genera tutte le sequenze binarie in ordine di lunghezza crescente, per esempio sotto forma di numeri binari progressivi; ne controlla la casualità mediante `Random`; e restituisce come risultato la prima sequenza casuale h tale che $|h| - \lceil \log_2 |h| \rceil > |P|$, ove P è la sequenza binaria che rappresenta il programma complessivo (`Paradosso` più `Random`).[1] Più precisamente poniamo:

`Function Paradosso`:

for binary $h \leftarrow 1$ **to** ∞ **do**

if $(|h| - \lceil \log_2 |h| \rceil > |P|)$ **and** (`Random`(h)=1)

then return h.

Poiché esistono sequenze casuali di ogni lunghezza, si incontrerà certamente una sequenza h per cui sono soddisfatte le due clausole $|h| - \lceil \log_2 |h| \rceil = |P| + 1$, e `Random`$(h)$=1; quindi il programma si arresta su questa h. Tuttavia le due clausole sono in contraddizione tra loro: infatti la prima indica che il programma rappresentato da P è "breve", e poiché questo programma genera h la sequenza non è casuale; la seconda indica invece che h è casuale poiché `Random`(h)=1. Dobbiamo concludere che l'algoritmo `Random` non può esistere, cioè $\mathcal{P}_{rand}(h)$ è indecidibile.

Riassumendo abbiamo visto che esistono moltissime sequenze casuali di ogni lunghezza, ma decidere se una sequenza arbitraria h è casuale è in genere algoritmicamente impossibile. Se si scopre un programma breve per generare h si conclude che la sequenza non è casuale; in caso contrario, e se la sorgente della sequenza non mostra particolari regolarità, si può ragionevolmente ritenere che h sia casuale e utilizzarla

[1] È importante osservare che il valore di $|P|$ è una costante; in particolare $|P|$ è indipendente da h che appare nel corpo del programma come *nome* di una variabile, mentre il *valore* di h è registrato nella memoria al di fuori del programma.

come tale. Ciò ci indurrà a applicare i teoremi del calcolo delle probabilità ottenendo risultati non corretti nel caso che h non sia casuale.

4.2 Generatori di numeri pseudo-casuali

Chiarito il significato di casualità rivolgiamo la nostra attenzione al problema delle sorgenti di sequenze. Nel caso binario una sorgente casuale genera una sequenza di *bit*, termine che propriamente indica la quantità di informazione associata allo stato di un oggetto che può assumere uno tra due stati equiprobabili. Poiché però il termine è entrato nell'uso anche come sinonimo di cifra binaria, diremo che la sorgente genera *bit casuali*, tali cioè che ognuno di essi può assumere il valore 0 con probabilità $\mathcal{P}r(0) = 1/2$ o il valore 1 con probabilità $\mathcal{P}r(1) = 1/2$, e la generazione di un bit è completamente indipendente da quella degli altri. Ciò implica immediatamente che il valore di un bit non può essere previsto osservando i precedenti.[2]

Benché le sequenze casuali esistono e, come abbiamo visto, sono in larga maggioranza, non è semplice individuare sorgenti casuali, al punto che la loro stessa esistenza è al centro di un dibattito filosofico. In particolare è difficile garantire che la generazione di un bit avvenga in modo indipendente dalla generazione degli altri, poiché nella realtà ogni esperimento modifica l'ambiente circostante e può influenzare l'esperimento successivo. Nella pratica la perfetta casualità di una sorgente non potrà essere garantita e dovremo accettare alcuni compromessi.

Per la generazione di sequenze *brevi* esistono due approcci che presentano sufficienti garanzie di casualità. Uno sfrutta la presunta aleatorietà di alcuni fenomeni fisici come i valori campionati del rumore proveniente da un microfono. L'altro impiega alcuni processi software rilevando per esempio come valori casuali lo stato dell'orologio interno di un calcolatore o la posizione della testina su un disco rigido. I due approcci producono risultati ragionevolmente imprevedibili se non si ha accesso fisico ai dispositivi utilizzati, i quali altrimenti potrebbero essere persino manomessi falsando ad arte il processo di generazione. Il loro impiego è però problematico per difficoltà di realizzazione, e perché le sequenze generate si considerano tanto più casuali quanto più sono brevi. D'altra parte poiché il numero di bit casuali richiesti nelle applicazioni

[2]Non è cruciale che le probabilità di apparizione di 0 e di 1 siano uguali tra loro, purché siano maggiori di zero e immutabili durante il processo. Infatti se una sorgente predilige la generazione di un bit (per esempio lo 0, quindi $\mathcal{P}r(0) > 1/2 > \mathcal{P}r(1)$), è possibile trasformarla da "sbilanciata" in "bilanciata" raggruppando i bit a coppie, scartando quelle del tipo 00 e 11, e trasformando le coppie 01 in 0 e le coppie 10 in 1: è facile dimostrare che nella nuova sequenza i valori 0 e 1 si presentano in modo casuale con pari probabilità.

crittografiche può essere molto elevato, si ricorre in genere a un diverso meccanismo algoritmico che ricerca la casualità all'interno di alcuni processi matematici.

Un *generatore di numeri pseudo-casuali* è un algoritmo che parte da un piccolo valore iniziale detto *seme*, solitamente fornito come dato di ingresso, e genera una sequenza arbitrariamente lunga di numeri; questa a sua volta contiene una sottosequenza detta *periodo* che si ripete indefinitamente. In linea di principio un generatore è tanto migliore quanto più lungo è il periodo, perché è questo che dovrebbe essere successivamente utilizzato come sequenza casuale. Per motivi di efficienza i generatori sono in genere realizzati con programmi molto brevi: quindi, sulla base della teoria sviluppata nel paragrafo precedente, le sequenze generate sono ben lungi da essere casuali. Questi generatori possono però essere considerati *amplificatori di casualità* perché se innescati da un seme casuale di lunghezza m, fornito dall'utente o prodotto con uno dei metodi visti sopra, generano una sequenza "apparentemente" casuale di lunghezza $n \gg m$. Una inerente limitazione è che il numero di sequenze diverse che possono essere così generate è al massimo pari al numero di semi possibili, cioè 2^m nel caso binario, enormemente minore del numero complessivo 2^n delle sequenze lunghe n. Tali caratteristiche motivano per questi generatori l'appellativo di pseudo-casuali.

Un generatore si valuta in base alla "similarità" tra la sequenza S da esso prodotta e una sequenza perfettamente casuale, e può essere o meno accettato a seconda dell'impiego previsto per la S. Se essa è destinata alla simulazione di un processo arbitrario, o alla scelta dei passi aleatori all'interno di un algoritmo randomizzato (si veda il paragrafo prossimo), è sufficiente che la sequenza superi una serie di test statistici standard sulla frequenza e la distribuzione dei suoi elementi. Se invece la S è destinata ad applicazioni crittografiche, in particolare alla generazione di chiavi segrete, essa deve superare anche un severo test addizionale che assicuri che sia impossibile fare previsioni sui suoi elementi prima che siano generati.[3]

I test standard di valutazione verificano dunque se una sequenza pseudo-casuale S rispetta alcune proprietà delle sequenze casuali senza riferimento alla predittibilità degli elementi. I principali sono: 1) il *test di frequenza*, che verifica se i diversi elementi appaiono in S approssimativamente lo stesso numero di volte; 2) il *poker test*, che verifica la equidistribuzione di sottosequenze di lunghezza arbitraria ma prefissata; 3) il *test di autocorrelazione*, che verifica il numero di elementi ripetuti a distanza prefissata; e 4) il *run test*, che verifica se le sottosequenze massimali contenenti elementi tutti uguali hanno una distribuzione esponenziale negativa.[4]

[3] I generatori pseudo-casuali e i relativi test statistici sono nati nel campo della simulazione. Il loro massiccio impiego in crittografia è più recente e sofisticato.

[4] A proposito di questo test si noti che se la lunghezza di una sequenza casuale tende all'infinito, essa include sottosequenze di qualsiasi lunghezza contenenti elementi tutti uguali, ma la frequenza

Un generatore pseudo-casuale molto semplice che supera con successo i quattro test citati è il *generatore lineare*, che produce una sequenza di interi positivi $x_1, x_2, \ldots, x_n$ a partire da un seme casuale x_0 secondo la relazione:

$$x_i = (a\, x_{i-1} + b) \ \texttt{mod}\ m$$

dove a, b, m sono interi positivi. Affinché il generatore abbia periodo lungo m, e quindi induca una permutazione degli interi $0, 1, \ldots, m-1$, i suoi parametri devono essere scelti in modo tale che $mcd(b, m) = 1$, $(a-1)$ sia divisibile per ogni fattore primo di m, e $(a-1)$ sia un multiplo di 4 se anche m è un multiplo di 4 (valori consigliati sono per esempio $a = 3141592653$, $b = 2718281829$, $m = 2^{32}$ e seme 0: si noti che se si fissa il seme la sequenza è completamente prevedibile). Un'ovvia generalizzazione è il generatore polinomiale che impiega una legge del tipo:

$$x_i = \left(a_1\, x_{i-1}^t + a_2\, x_{i-1}^{t-1} + \cdots + a_t\, x_{i-1} + a_{t+1}\right) \ \texttt{mod}\ m$$

e che supera anch'esso i test statistici. Questi generatori possono essere facilmente trasformati per produrre sequenze binarie pseudo-casuali. Si calcola il valore $r = x_i/m$: se la prima cifra decimale di r è dispari il bit generato è uno, altrimenti è zero.

I generatori lineari e polinomiali sono particolarmente efficienti ma non impediscono di fare previsioni sugli elementi generati, neanche quando il seme impiegato è strettamente casuale. Esistono infatti algoritmi che permettono di scoprire in tempo polinomiale i parametri del generatore partendo dall'osservazione di alcune sequenze prodotte, e questo ne svela completamente il funzionamento. Cosí se per esempio il generatore fosse impiegato per la creazione di una sequenza di chiavi, un crittoanalista potrebbe prevedere le chiavi future di un utente dalla conoscenza di alcune sue chiavi passate. Affinché questa condizione non sia verificata occorre rafforzare definizioni e test.

Il progetto dei nuovi generatori si fonda sull'esistenza di funzioni f, dette *one-way*, che sono computazionalmente facili da calcolare e difficili da invertire: cioè si conosce un algoritmo polinomiale per il calcolo di $y = f(x)$, ma si conoscono solo algoritmi esponenziali per il calcolo di $x = f^{-1}(y)$. Notiamo che si opera su numeri, quindi la complessità degli algoritmi deve essere riferita alla lunghezza della rappresentazione di x: un algoritmo che richiede un numero di operazioni proporzionale al valore di x sarà dunque esponenziale. Torneremo diverse volte su queste funzioni nel corso del nostro studio. Per il momento consideriamo la sequenza $S = x\ f(x)\ f(f(x))\ \ldots$. ottenuta iterando l'applicazione della f un numero arbitrario di volte. Ogni elemento della S si può calcolare efficientemente dal precedente, ma non dai successivi perché

di queste sottosequenze diminuisce esponenzialmente con la loro lunghezza.

f è one-way. Se dunque si calcola la S per un certo numero di passi senza svelare il risultato, e si comunicano poi gli elementi uno dopo l'altro in ordine inverso, ciascun elemento non è prevedibile in tempo polinomiale pur conoscendo quelli comunicati prima di esso.

Nelle applicazioni crittografiche si impiegano generatori binari e a questi ci rivolgiamo ora in modo formale. Un generatore binario supera il *test di prossimo bit* se non esiste un algoritmo polinomiale in grado di predire l'$(i+1)$-esimo bit della sequenza pseudo-casuale a partire dalla conoscenza degli i bit precedenti, con probabilità significativamente maggiore di $1/2$. Quindi per un crittoanalista che dispone di risorse computazionali ragionevoli, cioè polinomiali, la sequenza prodotta dal generatore risulta indistinguibile da una sequenza casuale. I generatori che superano il test di prossimo bit sono detti *crittograficamente sicuri*, e si può dimostrare che superano anche i quattro test standard discussi in precedenza. Essi vengono costruiti impiegando particolari *predicati* delle funzioni one-way, cioè proprietà che possono essere vere o false per ogni valore di x.

Un predicato $b(x)$ è detto *hard-core* per una funzione one-way $f(x)$ se $b(x)$ è facile da calcolare conoscendo il valore di x, ma è difficile da calcolare (o anche solo da prevedere con probabilità di successo maggiore di $1/2$) conoscendo solo il valore di $f(x)$. In sostanza la proprietà "hard-core", letteralmente "nucleo duro", concentra in un bit la difficoltà computazionale della f. Come vedremo (paragrafo 8.2) un esempio di funzione one-way è la $f(x) = x^2 \texttt{ mod } n$ se n non è primo. Posto per esempio $n = 77$ e $x = 10$, è (polinomialmente) facile calcolare il valore $y = 10^2 \texttt{ mod } 77 = 23$ ma è (esponenzialmente) difficile risalire al valore di $x = 10$ dato $y = 23$: questa operazione infatti si sa eseguire essenzialmente solo attraverso un numero esponenziale di tentativi. Ora il predicato: $b(x) =$ *"x è dispari"* è hard-core per la funzione suddetta; infatti il valore di x permette di calcolare immediatamente $b(x) = 1$ se x è dispari, $b(x) = 0$ se x è pari, ma il problema di calcolare $b(x)$ conoscendo solo il valore $y = x^2 \texttt{ mod } n$ è esponenzialmente difficile. Si può ovviamente risolvere il problema calcolando la radice quadrata di y modulo n, ma abbiamo osservato che questo è difficile; d'altra parte si può dimostrare che si riducono a esso altri problemi difficili della classe **NP** (paragrafo 3.4), confermando l'ipotesi che anche il calcolo di $b(x)$ dato $f(x)$ richieda tempo esponenziale.

Su questi risultati si fonda il generatore *BBS* introdotto da Blum, Blum e Shub. Sia $n = p \cdot q$ il prodotto di due numeri primi grandi tali che $p \texttt{ mod } 4 = 3$ e $q \texttt{ mod } 4 = 3$; $2 \cdot \lfloor p/4 \rfloor + 1$ e $2 \cdot \lfloor q/4 \rfloor + 1$ siano primi; e sia $x_0 = y^2 \texttt{ mod } n$ per un qualche y. Il generatore *BBS* impiega x_0 come seme, calcola una successione x_i di $m \leq n$ interi e genera in corrispondenza una sequenza binaria b_i secondo la legge:

$$x_i \leftarrow (x_{i-1})^2 \texttt{ mod } n, \quad b_i = 1 \Leftrightarrow x_{m-i} \text{ è dispari.}$$

Il generatore parte da x_0 cui corrisponde b_m, e procede al calcolo degli interi $x_1, \ldots, x_m$ memorizzando i corrispondenti bit $b_{m-1}, \ldots, b_0$ che comunica poi all'esterno in ordine inverso.

Dobbiamo dimostrare che il generatore *BBS* supera il test di prossimo bit, ma questo è in effetti un caso particolare di un risultato generale. Poniamo che $f(x)$ sia una arbitraria funzione one-way e $b(x)$ sia un suo predicato hard-core. Indichiamo con $f^{(i)}(x)$ l'applicazione iterata della funzione per i volte consecutive, e consideriamo la sequenza binaria che si ottiene partendo da un valore arbitrario di x, calcolando $b(x)$ e $f(x)$, iterando il calcolo della f un numero arbitrario di volte, e ordinando in senso inverso i valori del predicato così ottenuti:

$$b(f^{(i)}(x)) \quad b(f^{(i-1)}(x)) \quad \ldots \quad b(f^{(2)}(x)) \quad b(f(x)) \quad b(x) \tag{4.3}$$

Per quanto osservato in precedenza sulle funzioni one-way, e ricordando che il predicato scelto è hard-core, segue che ciascun elemento della sequenza supera il test di prossimo bit. In particolare la 4.3 è la sequenza generata dal *BBS* se si sceglie $f(x) = x^2 \bmod n$, e $b(x)$ è il corrispondente predicato di disparità.

Nonostante alcuni accorgimenti di calcolo per incrementarne l'efficienza, il generatore *BBS* è piuttosto lento poiché il calcolo di ogni bit richiede l'esecuzione di un elevamento al quadrato di un numero di grandi dimensioni nell'algebra modulare.[5] La lentezza è comunque una prerogativa di tutti i generatori progettati a partire da problemi computazionalmente difficili. Essi sono pertanto impiegati nella creazione di chiavi segrete corte, ma non quando si richiedono lunghe sequenze casuali: poiché questo caso si presenta in molte applicazioni crittografiche, si preferiscono generatori teoricamente meno sicuri ma efficientissimi e comunque a tutt'oggi inviolati.

È interessante notare che questi ultimi generatori possono essere costruiti utilizzando le funzioni di cifratura $\mathcal{C}(m, k)$ sviluppate per i cifrari simmetrici (paragrafo 1.2), utilizzando la chiave segreta k del cifrario e sostituendo il messaggio m con un opportuno valore relativo al generatore. In tal modo l'imprevedibilità dei risultati è garantita dalla struttura stessa dei cifrari, e per questi esistono realizzazioni estremamente efficienti. Poiché i cifrari generano parole (i crittogrammi) composte da molti bit, ogni parola prodotta potrè essere interpretata come numero pseudo-casuale o come sequenza di bit pseudo-casuali. Vediamo un generatore di questo tipo, approvato come *Federal Information Processing Standard* (*FIPS*) negli Stati Uniti: il cifrario

[5] Il calcolo del predicato può essere eseguito in modo più rapido sfruttando alcune proprietà dell'algebra modulare, poiché si conosce la fattorizzazione di n. Un miglioramento ulteriore si ottiene selezionando da ogni intero x_i un gruppo di bit di uscita anziché uno soltanto. Inoltre con una semplice variazione il generatore può produrre i bit di uscita nello stesso ordine degli interi x_i anziché in ordine opposto.

impiegato al suo interno è in genere una versione del *DES* di amplissima diffusione. Detto r il numero di bit delle parole prodotte (nel *DES* si ha $r = 64$), s un seme casuale di r bit, m il numero di parole che si desidera produrre, e ricordando che k è la chiave del cifrario, abbiamo:

```
Function Generatore(s, m):
    d ← rappresentazione su r bit di data e ora attuale;
    y ←C(d, k);
    z ← s;
    for i ← 1 to m do
        x_i ←C(y xor z, k);
        z  ←C(y xor x_i, k);
        comunica all'esterno x_i.
```

ove l'operatore **xor** indica l'OR esclusivo, bit a bit tra due sequenze binarie.

4.3 Algoritmi randomizzati

Se un risultato è difficile da conseguire si può pensare di affidarsi al caso. Naturalmente la probabilità di avere successo non aumenta per aver osato tanto, a meno che il giorno sia particolarmente fausto o qualche evento esterno venga in nostro soccorso. Non volendo contare sulla prima situazione andiamo alla ricerca della seconda: ci occuperemo così di un nuovo paradigma algoritmico che prese originariamente il nome di "probabilistico", perché nell'analisi della complessità intervengono alcuni concetti di calcolo delle probabilità, ma che oggi è detto in gergo "randomizzato" a indicare come la struttura stessa dell'algoritmo si basi sull'intervento di eventi casuali.

Una prima utilizzazione delle sorgenti casuali all'interno di un algoritmo è volta ad aggirare alcune situazioni sfavorevoli in cui si possono presentare i dati, combinando questi con una sequenza casuale. È una situazione simile a quella in cui opera un crittografo, perché si agisce con lo scopo di confondere un avversario che in questo caso non è il crittoanalista ma il fato avverso. L'esempio più noto è `Quicksort`, algoritmo per l'ordinamento di dati riportato in tutti i testi. Anziché operare sui dati d'ingresso nell'ordine in cui appaiono, l'algoritmo ne costruisce una permutazione casuale e opera su questa: lo scopo è quello di garantire che la complessità sia, nel caso medio, quella prevista probabilisticamente su dati arbitrari. Gli algoritmi di questa famiglia sono familiarmente noti come *Las Vegas*: essi generano un risultato *sicuramente* corretto in un tempo *probabilmente* breve.

Più recente e interessante per noi è l'impiego della casualità a fini in qualche modo complementari ai precedenti. Gli algoritmi cui ci riferiamo, noti come *Monte Carlo*,

generano un risultato *probabilmente* corretto in un tempo *sicuramente* breve.[6] Illustreremo il nuovo paradigma in relazione al problema $\mathcal{P}_{primo}(N)$ che chiede di stabilire se un dato numero N è primo. Il motivo della scelta è duplice. Anzitutto la soluzione di $P_{primo}(N)$ è importantissima in molti protocolli crittografici; come abbiamo visto nel paragrafo 3.4 il primo algoritmo polinomiale per la sua soluzione è stato proposto nel 2002, ma la complessità che lo caratterizzava era $O(n^{12})$, successivamente ridotta fino a $O(n^6)$, valore ancora estremamente alto in relazione alla lunghezza delle sequenze necessarie in crittografia ove n è tipicamente dell'ordine di migliaia di bit. Dobbiamo quindi cercare un algoritmo efficiente per risolvere il problema pur correndo il rischio di dare una risposta errata, purché la probabilità che ciò avvenga sia trascurabile. Il secondo motivo è storico, poiché proprio a questo problema sono stati dedicati i primi algoritmi randomizzati tipo Monte Carlo.

I diversi algoritmi randomizzati per risolvere $\mathcal{P}_{primo}(N)$ differiscono per le proprietà algebriche utilizzate, mentre la struttura generale è la stessa. Riportiamo per la sua semplicità ed efficienza l'*algoritmo di Miller e Rabin*, che prende il nome dai suoi inventori. Ammettiamo che N sia dispari, altrimenti è certamente composto, e poniamo $N-1 = 2^w z$ con z dispari (quindi $w \geq 1$ è il piú grande esponente che si può dare a 2). Sia y un intero arbitrario tale che $2 \leq y \leq N-1$. Se N è primo devono essere veri i due predicati:

P_1. $mcd(N, y) = 1$

P_2. ($y^z \bmod N = 1$) **or** (esiste un valore $i, 0 \leq i \leq w-1$, tale che $y^{2^i z} \bmod N = -1$)

che derivano dalla definizione di numero primo (P_1), e da note proprietà dell'algebra modulare (P_2, vedi paragrafo 8.1). Si ha poi:

Lemma (Miller e Rabin). Se N è composto, il numero di interi y compresi tra 2 e $N-1$ che soddisfano entrambi i predicati P_1 e P_2 è minore di $N/4$.

Dunque se per un intero y scelto a caso tra 2 e $N-1$ almeno uno dei predicati P_1, P_2 è falso, N è certamente un numero composto; se entrambi i predicati sono veri possiamo solo concludere che N è composto con probabilità minore di 1/4 (quindi è primo con probabilità maggiore di 3/4). Diciamo che y è un *certificato probabilistico*, o *testimone*, del fatto che N è composto. La validità del certificato può essere control-

[6] Il termine "Monte Carlo" è impiegato in analisi numerica per indicare una famiglia di algoritmi che forniscono soluzioni approssimate a problemi numerici. Il significato che noi diamo al termine è molto diverso: come vedremo la soluzione fornita dall'algoritmo potrà essere perfettamente giusta o completamente sbagliata, anche se il secondo caso si verificherà con bassissima probabilità.

lata attraverso la funzione `Verifica`(N, y): se la funzione assume valore 1 allora N è certamente composto, se assume valore 0 allora N "probabilmente" non è composto:

`Function Verifica`(N, y):

if (P_1 = **false**) **or** (P_2 = **false**)

then return 1

else return 0.

Un certificato è interessante se la sua verifica può essere eseguita in tempo polinomiale. Anzitutto partendo dall'intero $N-1$ e dimezzandolo per $O(\log N)$ volte consecutive si calcolano i valori w e z tali che $N-1 = 2^w z$, con z dispari. Nella funzione `Verifica`(N, y) il calcolo di P_1 si esegue in tempo polinomiale con l'algoritmo di Euclide, ma è critico il calcolo delle successive potenze y^z, y^{2z}, ..., $y^{2^{w-1}z} = y^{(N-1)/2}$ in P_2. Infatti, se eseguite in modo diretto mediante $(N-1)/2$ successive moltiplicazioni di y per sé stesso, queste operazioni richiedono tempo esponenziale nella dimensione di N. Tuttavia esiste un meccanismo polinomiale per calcolare l'elevamento a potenza che è bene studiare subito perché sarà implicitamente e spesso utilizzato nel seguito senza più farne menzione.

Il calcolo della potenza $x = y^z$ `mod` s, con y, z, s interi dello stesso ordine di grandezza, viene eseguito in tempo polinomiale se si procede per successive esponenziazioni. Si decompone anzitutto z in una somma di potenze del due: $z = 2^{w_1} + 2^{w_2} + \cdots + 2^{w_t}$ con $w_1 < \ldots < w_t$, operazione immediata se si utilizza la rappresentazione binaria di z. Si eseguono ora tutte le operazioni `mod` s. Si calcolano in successione le potenze y^{2^j} per $j = 1, 2, \ldots, w_t$ ottenendo ciascuna di esse come quadrato della precedente. Si calcola quindi x come prodotto tra gli y^{2^j} con $j = w_1, w_2, \ldots, w_t$. Ad esempio per $z = 45 = 1 + 4 + 8 + 32$ si eseguono le esponenzianioni: $y^2 = y \times y$, $y^4 = y^2 \times y^2$, $y^8 = y^4 \times y^4$, $y^{16} = y^8 \times y^8$, $y^{32} = y^{16} \times y^{16}$, e si calcola infine y^{45} come $y \times y^4 \times y^8 \times y^{32}$. In questo modo si eseguono $\Theta(\log_2 z)$ moltiplicazioni, cioè un numero lineare nella lunghezza della rappresentazione di z, il che conduce a un algoritmo complessivamente cubico.

Nell'algoritmo `Verifica`(N, y) si calcola y^z (in modulo) per successive esponenziazioni, poi si calcolano in successione i valori y^{2z}, ..., $y^{2^{w-1}z}$ elevando al quadrato y^z per $w-1$ volte consecutive. Il tempo complessivo di `Verifica`(N, y) è quindi polinomiale nella dimensione di N. Possiamo ora costruire l'algoritmo randomizzato `Test_MR`(N) per la verifica di primalità di N con il metodo di Miller e Rabin. La variabile k che appare nell'algoritmo ha un valore scelto dall'utente, che può essere fissato una volta per tutte o passato come parametro.

```
Function Test_MR(N):
    for i ← 1 to k do
        scegli y a caso tra 2 e N − 1;
        if Verifica(N, y)= 1 then return 0;
    return 1.
```

Il risultato $\texttt{Test\_MR}(N) = 1$ indica che N è primo e il risultato $\texttt{Test\_MR}(N) = 0$ indica che N è composto, ma le cose non stanno esattamente così. L'algoritmo può infatti arrestarsi durante il ciclo di **for** se incontra una condizione che attesta che N è composto, altrimenti esaurisce il ciclo e dichiara che N è primo. Nel primo caso il risultato è certamente corretto; nel secondo caso il risultato può essere errato (N in verità è composto), ma ciò accade solo se l'algoritmo ha impiegato k valori di y per cui il certificato fallisce. Se le k scelte di y sono state casuali e indipendenti, la probabilità di errore complessiva è data dal prodotto delle probabilità che il certificato abbia fallito ogni volta: e poiché queste sono tutte minori di $1/4$, la probabilità complessiva di errore è minore di $(1/4)^k$. Se per esempio si sceglie $k = 30$ l'algoritmo può dichiarare erroneamente primo un numero composto con probabilità al più $(1/4)^{30} \simeq 10^{-18}$, certamente inferiore a qualunque altra probabilità di errore per cause che possano ragionevolmente presentarsi. [7] Dunque possiamo legittimamente accettare il risultato come corretto dopo poche ripetizioni del ciclo, sempre che le scelte di y siano state eseguite con un buon generatore pseudo-casuale.

L'analisi di complessità dell'algoritmo è elementare. Poiché si ripete al massimo k volte un ciclo contenente operazioni di complessità polinomiale $p(n)$ nella dimensione n dei dati, la complessità è di ordine $\mathrm{O}(k{\cdot}p(n))$, polinomiale se k è costante o polinomiale.

Il contributo degli algoritmi randomizzati alla crittografia è determinante, quanto meno nella *generazione* di numeri binari primi grandissimi che come vedremo è alla base di molti cifrari. Potremo ripetere la generazione casuale di un intero alternata a un test probabilistico di primalità finché si incontra un numero dichiarato primo: la probabilità di errore di questo meccanismo è pari a quella del test, e valgono per essa i ragionamenti già fatti. Dobbiamo però sincerarci che il numero di interi da esaminare, fino a trovarne uno primo, sia ragionevolmente basso. Ora i numeri primi godono di una proprietà forte circa la loro densità sull'asse degli interi: il numero di interi primi e minori di N tende a $N/\log_e N$ per $N \to \infty$. Ciò significa che se N è

[7] In realtà si può dimostrare che per una grandissima maggioranza degli interi N composti, il numero di testimoni è molto maggiore del valore $3N/4$ derivante dal lemma di Miller e Rabin. Ne segue che scegliendo per esempio $k = 30$ la probabilità di errore è estremamente più piccola di $(1/4)^{30}$. A causa di tale rapidissima convergenza il test di Miller e Rabin è diventato uno standard.

grande, in un suo intorno di lunghezza $\log_e N$ cade mediamente un numero primo. E poiché $\log_e N$ è proporzionale alla dimensione (ovvero al numero di bit) n di N, il numero di prove attese è polinomiale in n.

Si può allora definire il seguente algoritmo `Primo(`n`)` per la generazione di un numero primo binario di almeno n bit, ove n è un valore fissato dall'utente. Il calcolo è innescato da un seme N di n bit, in cui i due bit estremi sono uguali a 1 e gli altri sono generati a caso.

`Function Primo(`n`)`:

$N \leftarrow 1S1$, ove S è una sequenza di $n-2$ bit prodotti da un generatore binario pseudo-casuale;

while `Test_MR(`N`)` $= 0$ **do** $N \leftarrow N + 2$;

return N.

Per quanto detto precedentemente l'algoritmo `Primo(`n`)` controlla in media un numero di interi proporzionale a n prima di trovarne uno primo, e poiché ogni iterazione richiede tempo polinomiale in n l'intero algoritmo richiede anch'esso tempo polinomiale. Notiamo che il numero N generato conterrà al massimo $n+1$ bit perché passando da n bit a $n+1$ il valore di N raddoppia, e tra N e $2N$ cadono certamente molti numeri primi.

Concludiamo questa breve discussione sugli algoritmi randomizzati con alcune considerazioni sul loro ruolo nella gerarchia delle classi di complessità. Abbiamo visto nel paragrafo 3.4 che l'*esistenza* di un certificato polinomiale determina l'appartenenza di un problema alla classe **NP**, ma che solo per la sottoclasse **P** sono noti algoritmi polinomiali per *determinare* un certificato, ovvero per risolvere il problema. L'introduzione dei certificati probabilistici apre un modo nuovo di considerare i problemi. Senza addentrarci in questo argomento molto complesso diciamo solo che, dato un problema decisionale $\mathcal{P}$ e una sua arbitraria istanza x di lunghezza n, un certificato probabilistico y di x è un'informazione di lunghezza polinomiale in n estratta perfettamente a caso da un insieme associato a x. Il certificato è utile se esiste un algoritmo di verifica $\mathcal{A}$ polinomiale in n e applicabile a ogni coppia $\langle x, y\rangle$ che attesta che x possiede la proprietà richiesta dal problema con probabilità maggiore di $1/2$, o attesta con certezza che non la possiede. Diciamo allora che $\mathcal{A}$ verifica $\mathcal{P}$ in tempo polinomiale randomizzato e poniamo:

Definizione RP è la classe dei problemi decisionali verificabili in tempo polinomiale randomizzato.[8]

[8] L'acronimo **RP** sta per "random polinomiale".

L'interesse per la classe **RP** deriva dalla possibilità di iterare l'applicazione dell'algoritmo di verifica fino a raggiungere una probabilità arbitrariamente bassa di errore nella risposta. La relazione tra le principali classi diviene:

$$\mathbf{P} \subseteq \mathbf{RP} \subseteq \mathbf{NP}$$

e si ritiene che non valga mai l'uguaglianza. Passiamo così dai problemi che ammettono soluzione in tempo polinomiale (la classe **P**), a quelli per cui ciò può essere garantito solo in senso probabilistico (la classe **RP**), a quelli per cui neanche questo sembra possibile (la classe **NP**).

Capitolo 5

Cifrari storici

Iniziamo la nostra discussione sui metodi di cifratura partendo dalle origini, cioè da alcuni *cifrari storici* proposti tanto tempo fa per consentire comunicazioni "sicure" tra poche persone.

La parola sicuro è riferita al passato poiché tutti questi cifrari sono stati già forzati grazie a tecniche in verità molto semplici. Tuttavia li descriveremo per vari motivi: da un lato è interessante comprendere l'evoluzione di questi studi; da un altro è istruttivo esaminare semplici linee di progetto che potrebbero comunque passarci per la mente, evidenziandone limiti e debolezze: scopriremo così che alcune antiche idee sono state riprese nei sofisticati metodi di oggi. Ultimo scopo (perché no?) è quello di mettere alla luce divertenti curiosità nascoste nella storia di questi cifrari.

Essenzialmente tratteremo di crittografia manuale, poiché fino a pochi decenni fa i procedimenti di cifratura e di decifrazione erano inevitabilmente realizzati con carta e penna. I messaggi da cifrare saranno composti da frasi di senso compiuto in linguaggio naturale, per le quali assumeremo che l'alfabeto sottostante sia composto dalle ventisei lettere $A, B, C, \ldots, X, Y, Z$, e i principi di Bacone espressi nel capitolo 1 saranno per quanto possibile rispettati. Il lettore non si spaventi se nella nostra carrellata partiremo dalla Grecia classica: il capitolo che dovrà leggere non sarà poi così lungo.

Il metodo più antico di cui si ha notiza fu inventato dagli spartani nel V secolo a. C.: si impiegava un'asta cilindrica, detta *scitale*, costruita in due esemplari identici posseduti dai due corrispondenti, su cui era avvolta a elica una fettuccia di cuoio. Il messaggio era scritto sulla fettuccia secondo le generatrici del cilindro; la fettuccia era quindi svolta e spedita al destinatario che poteva interpretarne il contenuto riavvolgendola sull'asta e ricostituendo così i giusti accostamenti tra le lettere. Naturalmente il principio di "innocenza" del crittogramma non era rispettato, e la fettuccia veniva celata dal messaggero che la indossava come cintura. Il metodo era piuttosto ingenuo,

ma anche i crittoanalisti dell'epoca dovevano esserlo; esisteva bensì una chiave segreta (il diametro della scitale) ma presumibilmente tutto il sistema era tenuto segreto.

Molti sistemi crittografici di questo genere sono descritti nella $\Pi o\lambda\iota o\rho\kappa\eta\tau\iota\kappa\acute{\alpha}$, opera di arte militare del IV secolo a.C. attribuita al greco Enea Tattico. Un intero capitolo è dedicato ai messaggi segreti e appare come la più antica trattazione organica a riguardo. Impressionano oggi la fantasia dispiegata e l'ingenuità delle proposte, tra cui quella, ripresa da Erodoto, di rasare la testa di uno schiavo, scrivere il messaggio sul cranio, aspettare che ricrescano i capelli, e spedire infine lo schiavo al destinatario del messaggio con l'invito a rasarlo di nuovo; e qui il criterio di "innocenza" è rispettato purché il messaggero non abbia un fare sospetto. Né mancano proposte di camuffare testi scritti, come quella di inviare un libro qualsiasi (innocenza) sottolineandovi un sottoinsieme di lettere che costituiscono il messaggio; o sostituire le vocali (chissà perché solo le vocali?) di un testo con altri simboli grafici corrispondenti. Sembra che tutto ciò fosse sufficiente a confondere gli avversari; e invero il primo cifrario impiegato secoli dopo dai romani, veri iniziatori di sistemi complessi, non era poi tanto più sicuro!

5.1 Il cifrario di Cesare

A parte i raffinati consigli che Ovidio elargì alle matrone desiderose di inviare messaggi segreti ai loro amanti (per i quali si rimanda all'originale nel terzo libro dell'*Ars Amatoria*), i romani furono grandi utenti di crittografia. Giulio Cesare in persona è l'autore di quello che, pur nella sua ingenuità, è probabilmente il più antico cifrario di concezione moderna. Se ne trova menzione nel *De bello gallico* e una descrizione precisa nell'opera di Svetonio secondo il quale Cesare soleva scrivere *per notas*, cioè in cifra, le lettere riservate.

L'idea di base è molto semplice: il crittogramma c è ottenuto dal messaggio in chiaro m sostituendo ogni lettera di m con la lettera che la segue tre posizioni più avanti nell'alfabeto. Affinché sia definita la terza lettera successiva anche per X, Y e Z si assume che l'alfabeto sia *circolare*, cioè che la Z sia seguita dalla A, ecc., e si parla di *rotazione* nell'alfabeto anziché di spostamento (figura 5.1). Per esempio la frase: "OGGI È UNA BELLA GIORNATA" viene trasformata in: "RJJL H XQD EHOOD JLRUQDWD". Abbiamo eliminato l'accento sulla lettera E, ma abbiamo mantenuto gli spazi tra le parole per consentire una facile lettura dell'esempio: si potrebbero però eliminare anche gli spazi per rendere il crittogramma meno facilmente interpretabile dall'esterno, mentre il destinatario ne comprenderebbe comunque il senso.

Il cifrario di Cesare era destinato all'uso ristretto di un gruppo di conoscenti e non richiede una chiave segreta, per cui scoprire il metodo di cifratura vuol dire compro-

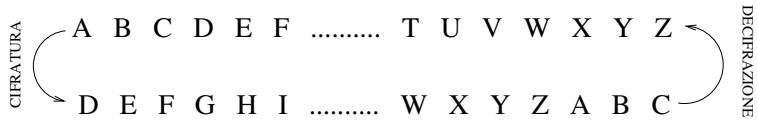


Figura 5.1: Il Cifrario di Cesare. Per la cifratura si sostituisce ogni lettera "in chiaro" della riga superiore con la lettera "cifrata" immediatamente sottostante. La decifrazione si esegue con il procedimento inverso.

metterne irrimediabilmente l'impiego: sembra tuttavia che l'autore confidasse tanto sulle proprie capacità di crittografo (nonché forse sull'ingenuità dei suoi antagonisti) da non temere alcuna forzatura. Il cifrario può però essere immediatamente trasformato per uso generale, aumentandone la sicurezza, mediante l'estensione dello spazio delle chiavi. Invece di rotare l'alfabeto di tre posizioni possiamo rotarlo di una quantità arbitraria k, con $1 \leq k \leq 25$ (il 26 è da evitare perché lascerebbe inalterato il messaggio). Il cifrario ha ora una chiave k che può assumere venticinque valori distinti: non molti, ma comunque meglio di quanto avesse fatto Cesare.

La versione generalizzata del cifrario di Cesare ammette una formulazione matematica molto semplice. Sia $pos(x)$ la posizione nell'alfabeto di una generica lettera x, partendo da $pos(A) = 0$ fino a $pos(Z) = 25$; e sia k la chiave, $1 \leq k \leq 25$. L'immagine cifrata di x è la lettera y tale che $pos(y) = (pos(x) + k)$ `mod` 26. D'altra parte la lettera in chiaro x è decifrata dalla y attraverso la relazione $pos(x) = (pos(y) - k)$ `mod` 26. Ad esempio se $k = 10$ per cifrare la lettera R, ove $pos(R) = 17$, si calcola $(17 + 10)$ `mod` $26 = 1 = pos(B)$; quindi alla R corrisponde la B. Queste operazioni si possono anche eseguire con due dischi concentrici su cui sono scritte le lettere dell'alfabeto in chiaro (disco interno) e le lettere cifrate (disco esterno) (figura 5.2). Rotando il disco interno in senso orario di k posizioni si pongono in corrispondenza le lettere in chiaro con quelle cifrate.

Il cifrario ha venticinque chiavi ed è quindi più flessibile di quello originale di Cesare, ma non è molto interessante perché il crittoanalista, che per ipotesi ne conosce la struttura, può applicare in breve tempo tutte le chiavi possibili a un crittogramma per decifrarlo e scoprire contemporaneamente la chiave segreta. Ma per quanto sostanzialmente inutilizzabile a fini crittografici, questo cifrario costituisce un buon punto di partenza per discutere alcuni aspetti propri dei cifrari moderni. In particolare esso gode della *proprietà commutativa*, cioè data una sequenza di chiavi $k_1, k_2, \ldots k_t$ e una sequenza di operazioni di cifratura o decifrazione che impiegano tali chiavi, l'ordine di

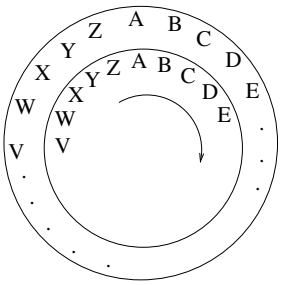


Figura 5.2: Realizzazione fisica del cifrario di Cesare generalizzato. Il disco interno, contenente l'alfabeto in chiaro, viene rotato di k posizioni portando le lettere in chiaro di fronte alle corrispondenti lettere cifrate che appaiono sul disco esterno.

tali operazioni può essere permutato arbitrariamente senza modificare il crittogramma finale. Inoltre è facile osservare che date due chiavi k_1, k_2 e una sequenza arbitraria s si ha $\mathcal{C}(\mathcal{C}(s, k_2), k_1) = \mathcal{C}(s, k_1 + k_2)$, e $\mathcal{D}(\mathcal{D}(s, k_2), k_1) = \mathcal{D}(s, k_1 + k_2)$, dove l'addizione tra le chiavi si esegue `mod` 26. Inoltre $\mathcal{C}(\mathcal{D}(s, k_2), k_1) = \mathcal{C}(s, k_1 - k_2)$ se $k_1 \geq k_2$, e $\mathcal{C}(\mathcal{D}(s, k_2), k_1) = \mathcal{D}(s, k_2 - k_1)$ se $k_2 > k_1$. Possiamo concludere che un sequenza di t operazioni di cifratura e decifrazione può essere ridotta a una sola operazione di cifratura o decifrazione, quindi *comporre più cifrari non necessariamente aumenta la sicurezza del sistema!* Ecco dunque un cardine fondamentale nel progetto di cifrari sicuri: più confusione non vuol dire necessariamente più sicurezza.

5.2 Una classificazione dei cifrari storici

Il cifrario di Cesare ci ha permesso di entrare nel mondo della crittografia: proseguiremo la discussione alla luce di alcune proprietà strutturali di interesse generale che caratterizzano i cifrari storici. Questi possono distinguersi anzitutto in:

- Cifrari a *sostituzione*, che sostituiscono ogni lettera del messaggio in chiaro con una o più lettere dell'alfabeto secondo una regola prefissata.
- Cifrari a *trasposizione*, che permutano le lettere del messaggio in chiaro secondo una regola prefissata.

A loro volta i cifrari del primo gruppo possono essere a sostituzione *monoalfabetica*, se alla stessa lettera del messaggio corrisponde sempre una stessa lettera nel

crittogramma; o a sostituzione *polialfabetica*, se alla stessa lettera del messaggio corrisponde una lettera scelta in un insieme di lettere possibili. Questa scelta varia durante il processo di cifratura/decifrazione seguendo criteri basati sulla posizione o sul contesto in cui appare la lettera nel messaggio.

Per esempio il cifrario di Cesare e la sua generalizzazione a più chiavi sono a sostituzione monoalfabetica, poiché la sostituzione di una lettera in chiaro dipende unicamente dalla lettera stessa e non dalla sua posizione o dal contesto in cui appare nel messaggio. Presenteremo ora degli esempi storici, alcuni anche abbastanza recenti, per ciascuna delle classi suddette e discuteremo le tecniche di crittoanalisi che hanno consentito di forzarli rendendoli così irrimediabilmente inutilizzabili.

5.3 Cifrari a sostituzione monoalfabetica

Se, come detto, il cifrario di Cesare generalizzato costituisce un semplice esempio di sostituzione monoalfabetica, la classe di questi cifrari risulta molto più ricca e variegata. Naturalmente cifrari più interessanti impiegheranno funzioni di cifratura e decifrazione più complesse dell'addizione e sottrazione in modulo; lo spazio delle chiavi sarà molto ampio; ma come vedremo la sicurezza sarà sempre molto modesta.

Come primo esempio scegliamo come funzione di cifratura $\mathcal{C}$ una *funzione affine*, secondo cui una lettera in chiaro x viene sostituita con la lettera cifrata y che occupa nell'alfabeto la posizione $pos(y) = (a \times pos(x) + b) \bmod 26$, dove a e b sono parametri del cifrario e ne costituiscono la chiave segreta $k = \langle a, b\rangle$. Per esempio con $k = \langle 3, 1\rangle$, il messaggio "IL NOSTRO TEMPO" diviene "ZI OREGAR GNBUR", poiché per la prima lettera sarà $pos(y) = (3 \times pos(I) + 1) \bmod 26 = (3 \times 8 + 1) \bmod 26 = 25 = pos(Z)$, quindi $x = \text{I}$ si cifra con $y = \text{Z}$, e così via.

La cifratura è un processo molto semplice realizzabile con carta e penna, ma la decifrazione è ora più complessa. Per eseguirla possiamo esplicitare la formula di cifratura rispetto a $pos(x)$ ottenendo la relazione $pos(x) = (a^{-1} \times (pos(y) - b)) \bmod 26$, ove a^{-1} indica l'inverso di a modulo 26 (ossia $a^{-1} \times a \equiv 1 \bmod 26$). Per esempio per $k = \langle 3, 1\rangle$ si ha $3^{-1} \bmod 26 = 9$, dunque $pos(x) = (9 \times (pos(Z) - 1) \bmod 26 = (9 \times 24) \bmod 26 = 8 = pos(I)$, quindi $y = \text{Z}$ si decifra come $x = \text{I}$.

Si noti però che per una nota proprietà dell'algebra modulare l'inverso di un intero n modulo m esiste ed è unico solo se n e m sono primi tra loro: cioè, nel nostro caso, se $mcd(a, 26) = 1$, il che impone un vincolo forte sulla definizione della chiave. Se a non fosse primo con 26 la funzione di cifratura non sarebbe più iniettiva, cioè più lettere in chiaro avrebbero come immagine la stessa lettera cifrata rendendo ambiguo il processo di decifrazione. Il parametro b può invece essere scelto liberamente tra 0 e 25. La chiave $k = \langle 3, 1\rangle$ scelta inizialmente soddisfa i requisiti suddetti poiché

$mcd(3, 26) = 1$ e 3^{-1} `mod` $26 = 9$; per cui $\mathcal{D}(y, k)$ è la (unica) lettera x dell'alfabeto che occupa la posizione $(9 \times (pos(y) - 1))$ `mod` 26. Se invece scegliessimo $k = \langle 13, 0\rangle$, sarebbe $mcd(13, 26) = 13$, e tredici lettere in chiaro verrebbero trasformate nella stessa lettera cifrata: in particolare tutte le lettere in posizione pari nell'alfabeto verrebbero trasformate nella lettera cifrata A, tutte quelle in posizione dispari verrebbero trasformate in N, e la decifrazione sarebbe quindi impossibile.

Ma allora quante sono le chiavi segrete possibili per il cifrario affine? Poiché il parametro a deve essere primo con 26 e i fattori primi di 26 sono 2 e 13, a può assumere qualsiasi valore dispari tra 1 e 25 con esclusione di 13, cioè uno tra dodici valori possibili. Poiché il parametro b può assumere ventisei valori e le chiavi legittime sono date da tutte le possibili coppie di a e b, vi sono in totale $12 \times 26 = 312$ chiavi, o meglio 311 perché è bene scartare la chiave $\langle 1, 0\rangle$ che lascia inalterato il messaggio.

In sostanza il cifrario affine permette di definire trecentoundici permutazioni dell' alfabeto da mettere in corrispondenza con le lettere in chiaro. Per quanto questo numero sia più grande del numero di chiavi del cifrario di Cesare generalizzato, e potrebbe rendere oneroso scongiurare un attacco esauriente condotto con carta e penna, esso risulta comunque irrisorio se si impiega un calcolatore per la crittoanalisi. Dobbiamo dunque aumentare il numero di permutazioni realizzabili con la chiave segreta del cifrario.

Il metodo più semplice per raggiungere questo obiettivo, e portarlo alle sue estreme conseguenze, è rappresentato dal *cifrario completo*, ove si assume che la chiave segreta k sia un'arbitraria permutazione delle lettere dell'alfabeto. In questo modo la lettera in chiaro che occupa la posizione i nell'alfabeto è trasformata nella lettera che occupa la posizione i nella permutazione scelta. La chiave è più lunga rispetto a quella del cifrario affine (ventisei lettere contro due numeri interi), ma lo spazio è esteso a $26! - 1$ chiavi possibili, escludendo la permutazione identica. Poiché la funzione fattoriale cresce come n^n (vedi paragrafo 3.3), lo spazio delle chiavi è incredibilmente vasto e risulta inesplorabile con metodi esaurienti. E se ricordare a memoria una chiave tanto lunga può risultare per alcuni improponibile, si può adottare qualsiasi metodo "personale" per generare univocamente una sequenza di ventisei lettere diverse tra loro, per esempio espandendo una chiave più corta con le lettere mancanti poste in ordine prefissato. Il metodo non è altrettanto sicuro ma, come vedremo, il cifrario stesso non è comunque sicuro per motivi diversi.

La semplicità della realizzazione e l'enorme spazio delle chiavi del cifrario completo possono tentarci, ma in crittografia è bene essere sempre poco ottimisti perché non è detto che un crittoanalista si serva dell'approccio contro cui vogliamo difenderci. E infatti, come vedremo nel seguito, alcune osservazioni sulla struttura logica dei messaggi in chiaro e sull'occorrenza statistica delle lettere permettono di forzare il

cifrario in modo quasi banale senza ricorrere a un attacco esauriente.

5.4 Cifrari a sostituzione polialfabetica

I cifrari a sostituzione sono fortemente potenziati se si adotta lo schema polialfabetico, nel quale una stessa lettera che si incontra in punti diversi del messaggio in chiaro ammette un insieme di lettere sostitutive possibili scelte con una regola opportuna.

L'esempio più antico risale nuovamente ai tempi di Roma imperiale, ed è molto interessante sia per la genialità del metodo che per la storia che condusse alla rottura del cifrario, ricostruite da Robert Graves in un libro famoso.[1] Durante il suo governo Augusto tenne un archivio speciale cifrato di cui solo lui e la moglie Livia conoscevano la struttura e la chiave. Rapporti sui senatori e sulle persone più in vista, corrispondenza diplomatica, messaggi intercettati dai "servizi segreti", tutto vi era custodito in cifra. Alla morte di Augusto, Tiberio entrò in possesso dell'archivio e chiese alla madre di svelargliene la chiave, ma Livia sostenne che solo Augusto era in grado di interpretare quei documenti. Tiberio studiò l'archivio senza successo, e il problema fu dimenticato fino a che l'imperatore Claudio, che era un acutissimo studioso, giunse alla soluzione in modo in parte accidentale.

I documenti dell'archivio erano scritti in numeri anziché in lettere. Augusto li redigeva in greco, poi metteva in corrispondenza la sequenza di lettere del documento con la sequenza di lettere del primo libro dell'Iliade, e sostituiva ogni lettera del documento con un numero che indicava la distanza, nell'alfabeto greco, di tale lettera da quella in pari posizione nell'Iliade. Per esempio se la lettera in posizione i nel documento era α e la lettera in posizione i nell'Iliade era ϵ, il carattere in posizione i nel crittogramma era il numero 4 perché questa è la distanza tra α e ϵ nell'alfabeto greco. Il cifrario risultante è polialfabetico perché la stessa lettera del messaggio è in genere cifrata con lettere diverse, in relazione al carattere dell'Iliade con cui si trova in corrispondenza. Claudio scoprì il sistema perché rinvenne casualmente un'edizione dell'Iliade che era stata di Augusto e riportava annotazioni appena percettibili nel primo libro, che egli immaginò potesse essere posto in relazione al cifrario.

Il metodo era completamente riservato, ma sarebbe rimasto praticamente inviolabile anche se fosse stato reso pubblico mantenendo segreta solo la chiave, cioè il libro scelto per rappresentarla. Un siffatto cifrario è infatti molto difficile da forzare se la chiave è lunghissima, ed è interessante osservare che fu utilizzato nella seconda guerra mondiale prendendo come chiave una pagina prefissata di un libro e cambiandola di giorno in giorno. Lo svantaggio è quello di dover registrare per iscritto la chiave per

[1] Si tratta di *I Claudius*, pubblicato da Barker Publ. nel 1934 e tradotto poi in molte lingue.

poterla ricordare, e forse per questo il cifrario di Augusto fu dimenticato. L'idea di ricorrere a sostituzioni polialfabetiche fu però riproposta nel Cinquecento da Leon Battista Alberti, per prender corpo nello stesso secolo in un metodo pratico proposto dal diplomatico francese Blaise de Vigenère, e ritenuto inattaccabile fino alla metà dell'Ottocento.

Il cifrario di Alberti era molto ingegnoso e si prestava a vari modi di impiego; era però basato su un "disco cifrante che mittente e destinatario dovevano possedere uguale (e diverso da quello di altre coppie di utenti), il che ne limitava un uso generale. Questo attrezzo, di cui si conservano diverse immagini, era in realtà costituito da due dischi rotanti uno nell'altro che recavano due alfabeti affacciati, similmente a quanto indicato nella figura 5.2 per il cifrario di Cesare: la corrispondenza tra i caratteri dei due alfabeti poteva quindi essere continuamente variata mediante rotazioni di uno o dell'altro.[2]

I due alfabeti erano diversi tra loro. Quello del disco esterno era costituto da un insieme di caratteri un po' limitato disposti in ordine alfabetico, seguito da alcune cifre numeriche: con esso si formulava il messaggio e le cifre erano utilizzate per variare liberamente la chiave durante la cifratura. Il disco interno conteneva un alfabeto più ricco disposto in un ordine arbitrario proprio di ogni coppia di dischi e non conteneva cifre: con questo si costruiva il crittogramma. Un nostro esempio è riportato nella figura 5.3 ove, per semplicità, gli alfabeti esterno e interno sono rappresentati su due righe parallele e una rotazione relativa dei dischi è rappresentata con uno *shift* ciclico della seconda riga. Per i due dischi abbiamo rispettivamente utilizzato i 21 caratteri caratteri dell'alfabeto italiano più le cifre da 1 a 5, e una permutazione arbitraria delle 26 lettere dell'alfabeto inglese. Per illustrare il funzionamento del cifrario vediamone un esempio che impiega il così detto "indice mobile, uno dei metodi applicabili col disco.

Consideriamo il messaggio IL DELFINO, che come d'uso sarà cifrato senza spazio tra le parole per non fornire un'informazione rilevante per un crittoanalista e sostanzialmente inutile per il destinatario. Si parte da una chiave concordata tra i due utenti che indica l'allineamento iniziale dei due dischi, specificando il corrispondente del carattere A nel secondo disco: nel nostro esempio la chiave è A-E (figura 5.3.a). Il messaggio m, e il conseguente testo cifrato c, saranno:

m:	I	L	D	*2*	E	L	*P*	F	I	N	O
c:	P	D	C	S	W	D	O	O	I	R	J

[2]Varie immagini del disco sono reperibili su Internet, ma le descrizioni del suo funzionamento sono spesso approssimative o addirittura sbagliate.

A B C D E F G H I L M N O P Q R S T U V Z 1 2 3 4 5
E Q H C W L M V P D N X A O G Y I B Z R J T S K U F

(a)

A B C D E F G H I L M N O P Q R S T U V Z 1 2 3 4 5
P D N X A O G Y I B Z R J T S K U F E Q H C W L M V

(b)

Figura 5.3: Il disco cifrante di Leon Battista Alberti sviluppato su due righe parallele. a) Allineamento degli alfabeti con chiave iniziale A-E. b) Nuovo allineamento dopo l'apparizione nel crittogramma di S (corrispondente a 2) e di O (corrispondente a P) dopo due caratteri, a indicare la nuova chiave A-P.

dove i caratteri aggiuntivi *2* e *P* in corsivo sono inseriti dal cifratore in modo arbitrario, per indicare un cambio di chiave. Infatti i primi tre caratteri vengono cifrati con gli alfabeti della figura 5.3.a. Il successivo carattere S di c viene decifrato con il numero 2: ciò indica che questo carattere non è parte di m, ma che dopo due successivi caratteri, codificati sempre con la stessa chiave, questa sarà cambiata. Così l'arrivo di O nel crittogramma, decifrato con il carattere P, indica la nuova chiave A-P che richiede una rotazione mutua dei dischi per portare P in coincidenza con A (figura 5.3.b): e con questa chiave si completa la cifratura.

Il cifrario di Vigenère è meno sofisticato di quello di Alberti, ma è assai più pratico perché basato su una semplice tabella $\mathcal{T}$ di pubblico dominio. Nella $\mathcal{T}$, di dimensioni 26×26, ogni riga i, $1 \leq i \leq 26$, contiene l'alfabeto di ventisei lettere rotato verso sinistra di $i - 1$ posizioni (figura 5.4: si noti che anche le colonne contengono alfabeti ugualmente rotati). Questa tabella è facile da costruire e nota a tutti. La chiave del cifrario è costituita da una parola segreta k, il cui senso (compiuto oppure no) e la cui lunghezza dipendono dall'abilità mnemonica degli utenti. Dato un messaggio in chiaro m, si produce il crittogramma disponendo m e k su due righe adiacenti e allineando le loro lettere in verticale. Se k è più corta di m, tale chiave viene ricopiata più volte consecutivamente per tutta la lunghezza di m. In questo modo ogni lettera x del messaggio in chiaro risulta allineata verticalmente con una lettera y della chiave, e la x viene sostituita nel crittogramma con la lettera che si trova nella cella di $\mathcal{T}$ all'incrocio tra la riga che inizia con x e la colonna che inizia con y.

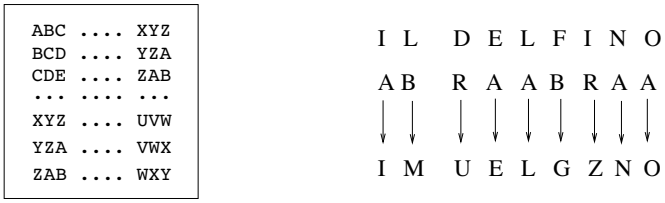


Figura 5.4: La tabella di Vigenère, e l'applicazione del cifrario al messaggio IL DELFINO con chiave ABRA.

Nella figura 5.4 si osserva che le lettere del messaggio "IL DELFINO" allineate con la lettera A della chiave "ABRA" non subiscono alcuna modifica; quelle allineate con la lettera B della chiave vengono traslate di una posizione in avanti (la L diventa M, la F diventa G); quelle allineate con R vengono traslate di diciassette posizioni in avanti (la D diviene U, la I diviene Z): ciò avviene perché le lettere B e R, capostipiti di colonne in $\mathcal{T}$, sono distanti rispettivamente di una e diciassette posizioni dall'inizio dell'alfabeto. Notiamo inoltre che una stessa lettera in chiaro, per esempio la I, viene cifrata in due modi diversi a seconda della lettera della chiave allineata con essa, cioè il cifrario è polialfabetico. Per decifrare il crittogramma si esegue l'ovvio procedimento inverso. Poiché le righe di $\mathcal{T}$ contengono tutte le diverse rotazioni cicliche dell'alfabeto, i processi di cifratura e decifrazione sono univoci.

Fissata un messaggio m, teoricamente le chiavi utilizzabili in un metodo a sostituzione polialfabetica sono tutte le possibili sequenze lunghe come m; ma in pratica la scelta è quasi sempre limitata a parole di senso compiuto non molto lunghe. Ed è proprio la lunghezza della chiave che influenza la sicurezza del metodo. Infatti se la chiave contiene h caratteri, le apparizioni di una stessa lettera distanti un multiplo di h nel messaggio in chiaro m saranno trasformate nella stessa lettera cifrata, poiché si sovrapporranno alla stessa lettera della chiave. Quindi per ogni intero positivo $i \leq h$ possiamo costruire il sottomessaggio $m[i]$ formato dalle lettere di m che occupano in esso le posizioni $i, i+h, i+2h, \ldots$, e in ciascuno di tali sottomessaggi tutte le lettere sono allineate con la stessa lettera della chiave. In sostanza il messaggio è decomposto in h sottomessaggi, ciascuno dei quali viene cifrato di fatto con un metodo monoalfabetico. Ciò fa immediatamente intuire come i cifrari polialfabetici non siano poi tanto più potenti dei monoalfabetici se le chiavi non sono molto lunghe.[3]

[3] A questo punto non dovrebbe sfuggire al lettore l'analogia tra questi cifrari e quello di Augusto. La vera differenza è nella chiave, che nel cifrario di Augusto era lunghissima (il che è un vantaggio per la sicurezza), ma era sempre la stessa per tutti i messaggi: scoperta la chiave il cifrario era

5.5 Cifrari a trasposizione

L'idea di base dei cifrari di questa famiglia è eliminare qualsiasi struttura linguistica presente nel crittogramma permutando le lettere del messaggio in chiaro e inserendovene eventualmente altre che vengono ignorate nella decifrazione.

Il più semplice cifrario a trasposizione è quello a *permutazione semplice*. Fissato un intero h e una permutazione π degli interi $\{1, 2, 3, \ldots h\}$ che costituisce la chiave, il processo di cifratura consiste nel suddividere il messaggio m in blocchi di h lettere ciascuno, e nel permutare le lettere di ciascun blocco in accordo alla permutazione π (figura 5.5). Se la lunghezza del messaggio non è divisibile per h, si aggiungono alla fine di esso delle lettere qualsiasi (in gergo *padding*) che partecipano alla trasposizione, ma sono ignorate dal destinatario perché la decifrazione le riporta in fondo al messaggio.

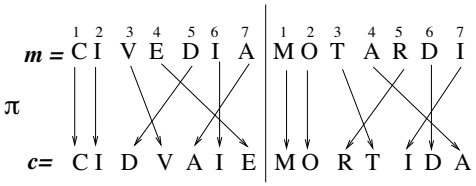


Figura 5.5: Cifrario a permutazione semplice, con $h = 7$ e $\pi = \{1253764\}$. Il messaggio in chiaro è CI VEDIAMO TARDI, il crittogramma corrispondente è CIDVAIEMORTIDA.

Questo cifrario ha tante chiavi possibili quante sono le permutazioni di h elementi, meno 1 (escludendo la permutazione identica), cioè $h! - 1$. Poiché h non è fissato a priori, tanto maggiore è h tanto più difficile è impostare un attacco esauriente, nonché scoprire la periodicità indotta dalla ripetizione della chiave. D'altra parte al crescere di h cresce come sempre la difficoltà di ricordare la chiave π, che sarà quindi scelta con un giusto compromesso.

Una interessante generalizzazione del cifrario precedente è il cifrario a *permutazione di colonne*, la cui chiave segreta $k = \langle c, r, \pi \rangle$ consiste di due interi c e r, che denotano rispettivamente il numero di colonne e di righe di una tabella di lavoro $\mathcal{T}$, e di una permutazione π degli interi $\{1, 2, \ldots, c\}$. Il messaggio in chiaro m viene decomposto in blocchi $m_1, m_2, \ldots$ di $c \times r$ caratteri ciascuno. Nella cifratura di un generico blocco m_i i caratteri sono distribuiti tra le celle di $\mathcal{T}$ in modo regolare, scrivendoli

definitivamente compromesso.

per righe dall'alto verso il basso. Poi le colonne di $\mathcal{T}$ vengono *permutate* secondo π e *lette* dall'alto verso il basso e da sinistra verso destra: la sequenza di caratteri così ottenuta costituisce il crittogramma di m_i (figura 5.6). Per cifrare il blocco successivo m_{i+1} la tabella $\mathcal{T}$ viene azzerata e il procedimento si ripete. Il crittogramma ha un periodo pari a $c \times r$. Il numero di possibili chiavi è teoricamente esponenziale nella lunghezza del messaggio non essendoci vincoli sulla scelta di c e r, sempre che, come al solito, sia possibile ricordare la permutazione π.

N	O	N	S	O	N
O	I	O	C	O	L
P	E	V	O	L	E

Tabella $\mathcal{T}$

O	N	O	N	S	N
I	O	O	O	C	L
E	P	L	V	O	E

Tabella $\mathcal{T}$ permutata

c = O I E N O P O O L N O V S C O N L E

Figura 5.6: Cifrario a permutazione di colonne, con $c = 6$, $r = 3$ e $\pi = \{215346\}$. Il messaggio in chiaro è NON SONO IO COLPEVOLE, il crittogramma corrispondente è OIENOPOOLNOVSCONLE.

Tra i cifrari a trasposizione meritano un posto speciale i cifrari *a griglia*, non tanto per l'interesse pratico che è oggi limitato ai giochi matematici, quanto per l'idea particolare su cui sono basati. Il loro progenitore è il cifrario di Richelieu, concepito dal cardinale per soddisfare il principio di "innocenza" di Bacone. Il crittogramma può infatti essere celato in un libro qualsiasi, e la chiave è costituita da una scheda perforata e dall'indicazione di una pagina di quel libro. Il processo di decifrazione consiste nel sovrapporre la scheda alla pagina: le lettere visibili attraverso la perforazione costituiscono il messaggio in chiaro.

Questa idea può essere estesa, rinunciando all'innocenza, in modo che il crittogramma invada tutta la pagina che in questo caso è costruita ad hoc. La chiave segreta è una *griglia* quadrata di dimensioni $q \times q$, con q pari, in cui $s = q^2/4$ celle (cioè un quarto del totale) sono trasparenti e le altre opache. La griglia viene posta su una pagina P di pari dimensioni, ove si scrivono in sequenza i primi s caratteri del messaggio nelle posizioni corrispondenti alle celle trasparenti della griglia. Questa viene poi rotata tre volte di 90^o in senso orario, e si ripete per ogni rotazione l'operazione di scrittura di tre successivi gruppi di s caratteri. Naturalmente la griglia deve essere scelta in modo che le posizioni corrispondenti alle celle trasparenti non si

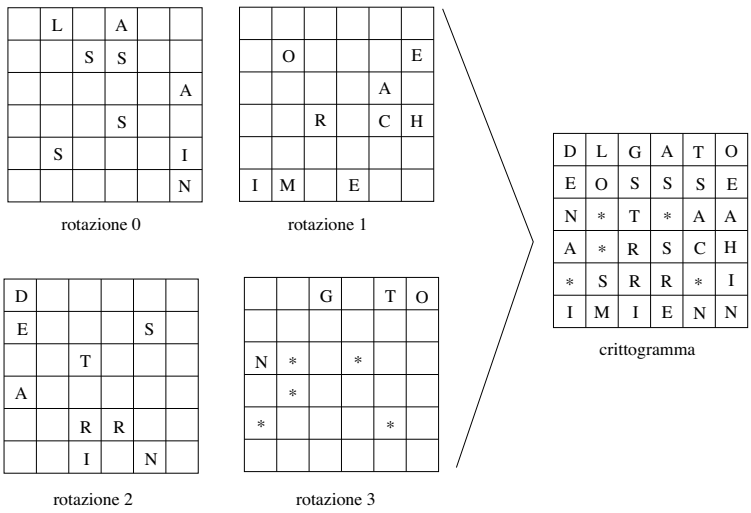


Figura 5.7: Uso di una griglia 6×6 con nove celle trasparenti per cifrare il messaggio L'ASSASSINO È ARCHIMEDES TARRINGTON. Le celle con * sono riempite con caratteri a caso.

sovrappongano mai nelle quattro rotazioni. Il processo è illustrato nella figura 5.7 su una griglia con $q = 6$ e $s = 9$. Se la lunghezza del messaggio è minore di $4s$, le posizioni della pagina P rimaste vuote vengono riempite con caratteri scelti a caso. Se tale lunghezza è maggiore di $4s$, il messaggio viene decomposto in blocchi consecutivi di $4s$ caratteri e ciascun blocco è cifrato indipendentemente dagli altri. La decifrazione di P è ovviamente eseguita sovrapponendovi quattro volte la griglia.

Vediamo ora come e in quanti modi si può costruire la griglia. Fissata sulla pagina una cella a_0, indichiamo con a_1, a_2, a_3 le tre celle che si raggiungono da a_0 con tre rotazioni successive: una e una sola di queste quattro celle sarà trasparente nella griglia. Fissiamo ora una seconda cella b_0 diversa dalle precedenti e individuiamo le tre celle b_1, b_2, b_3 raggiunte per rotazione. Ripetendo il procedimento $s = q^2/4$ volte tutte le celle della pagina saranno state raggiunte esattamente una volta e risulteranno divise in s gruppi di quattro celle ciascuno. La griglia si costruisce rendendo trasparente una cella arbitraria in ogni gruppo: vi sono perciò $G = 4^s = 2^{q^2/2}$ griglie (cioè chiavi segrete) possibili. Per esempio per $q = 6$ si ha $G = 2^{18} \approx 260.000$, valore molto modesto ma destinato a crescere rapidamente con q; per $q = 12$

si ha $G = 2^{72}$, sufficiente a porre il cifrario al riparo da un attacco esauriente.

5.6 Crittoanalisi statistica

La sicurezza di un cifrario è legata alla dimensione dello spazio delle chiavi che deve essere sufficientemente ampio da impedire attacchi esaurienti, cioè il tentativo di decifrazione con tutte le chiavi possibili finché il crittogramma si trasforma in un messaggio significativo. Ciò non esclude però la possibilità che il cifrario sia attaccato con altri metodi anche banali: tutti i cifrari storici sono stati infatti violati con un attacco *statistico* di tipo *cipher text*, in cui il crittoanalista ha a disposizione solo il crittogramma e qualche informazione sull'ambiente in cui esso è stato generato.[4] L'impiego del metodo si fa risalire in Europa alla metà del diciannovesimo secolo, quando si scoprì come violare il cifrario di Vigenère che era considerato assolutamente sicuro da trecento anni; ma un antico trattato rinvenuto recentemente a Istànbul mostra come i concetti di base fossero già noti agli arabi sin dal nono secolo. Discutiamo ora la crittoanalisi statistica iniziando dai cifrari a sostituzione monoalfabetica e applicandola poi agli altri cifrari storici, il che tra l'altro ci permetterà di apprezzare a fondo le differenze strutturali tra quelle famiglie di cifrari.

Tra le informazioni note al crittoanalista si ammette che questi conosca il metodo impiegato per la cifratura/decifrazione, ipotesi motivata dal sospetto che un segreto non possa essere mantenuto troppo a lungo (mentre la chiave può essere spesso cambiata), ma più realistica di quanto ci si aspetti anche in ambienti ove i segreti sono custoditi in maniera ferrea. Infatti il crittoanalista può ripetere i suoi attacchi in sequenza assumendo di volta in volta che sia stato utilizzato un diverso metodo di cifratura, fino a scoprire indirettamente quello giusto quando riesce a decifrare il crittogramma; o può usare altri indicatori che lo conducano al cifrario usato, come vedremo sotto. Una seconda assunzione è che il messaggio sia scritto in un linguaggio naturale noto al crittoanalista, o in uno tra vari linguaggi anch'essi noti: negli esempi ci riferiremo all'italiano. Infine si ammette che il messaggio sia sufficientemente lungo per poter rilevare alcuni dati statistici sui caratteri che compongono il crittogramma.

L'attacco si basa sull'impiego di un insieme di tavole note. È infatti ben studiata in ogni lingua la frequenza con cui appaiono in media le varie lettere dell'alfabeto, anche se i dati disponibili variano marginalmente a seconda dell'insieme di testi su cui è stata eseguita la statistica. Nella figura 5.8 è riportato un istogramma di frequenze

[4] Anche se non se ne parla esplicitamente negli studi crittografici, una parziale conoscenza degli "usi e costumi" degli utenti di un sistema è essenziale per tentare di comprenderne i messaggi. Siamo certi, pur senza poterne fornire una dimostrazione matematica, che la decifrazione di una conversazione tra marziani sia impossibile senza conoscere qualche abitudine di quei gentili esseri.

relativo all'italiano. Dati simili sono noti per le frequenze dei digrammi (gruppi di due lettere consecutive), dei trigrammi (gruppi di tre lettere consecutive), e così via. Tali gruppi sono collettivamente detti *q-grammi*: poiché il numero di essi cresce come 26^q, spesso ci si limita a considerare i più piccoli.

Ora se un crittogramma è stato generato per sostituzione monoalfabetica la frequenza con cui vi appare una lettera y, corrispondente alla x del messaggio, sarà approssimativamente uguale alla frequenza della x in italiano. Confrontando le frequenze delle lettere come appaiono nel crittogramma con quelle dell'italiano, e provando alcune permutazioni tra lettere di frequenze assai prossime, si ottengono diverse ipotesi di prima decifrazione in genere non lontane dalla realtà. La decifrazione nel cifrario di Cesare generalizzato è banalissima poiché è sufficiente scoprire la corrispondenza di una sola coppia $\langle y, x\rangle$ per svelare l'intero messaggio. La decifrazione in un cifrario affine è appena più complessa, perché è sufficiente individuare due coppie di lettere corrispondenti con cui impostare un sistema di due congruenze nelle due incognite a e b che formano la chiave segreta. La risoluzione del sistema è sempre possibile perché deve esistere l'inverso di a nell'aritmetica modulo 26, quindi a e b si possono calcolare prontamente. Ma anche la decifrazione in un cifrario completo, ove la chiave è una permutazione arbitraria dell'alfabeto, non presenta difficoltà se si associano le lettere in base alle frequenze. Se per esempio le lettere E, R e U appaiono nel crittogramma con frequenze comprese tra 6/100 e 7/100, esse rappresentano con ogni probabilità le lettere L, N e R del messaggio: si provano allora le sei associazioni possibili (E-L, R-N, U-R), (E-L, R-R, U-N), ..., (E-R, R-N, U-L), facendosi guidare dalla morfologia delle parole risultanti. Tali associazioni si possono raffinare controllando i digrammi più frequenti, e in genere a questo punto il messaggio è completamente svelato; altrimenti si passa ai trigrammi, e così via.

Assai più difficile è la decifrazione nei cifrari a sostituzione polialfabetica perché ciascuna lettera y del crittogramma dipende da una coppia di lettere $\langle x, z\rangle$ provenienti dal messaggio e dalla chiave: questo altera le frequenze delle lettere del crittogramma rendendole assai meno differenziate che nella lingua originale, e non consente una decifrazione diretta. Notiamo però che questi cifrari hanno un punto di debolezza se la chiave è unica e ripetuta più volte per coprire l'intero messaggio, come avviene nel cifrario di Vigenère che fu attaccato proprio su queste basi. Infatti, come già abbiamo osservato, se la chiave contiene h caratteri il crittogramma può essere decomposto in h sottosequenze ciascuna delle quali è stata ottenuta per sostituzione monoalfabetica: il problema è dunque quello di scoprire il valore di h per scomporre il crittogramma e continuare poi la decifrazione con il metodo monoalfabetico.

A questo punto possiamo fare un'osservazione chiave: il messaggio contiene quasi sicuramente gruppi di lettere adiacenti ripetuti più volte, come i trigrammi più fre-

Figura 5.8: La frequenza in percentuale delle lettere nell'italiano.

quenti nella lingua se non intere parole cui il testo crucialmente si riferisce. Ora se due apparizioni di una stessa sottosequenza sono allineate nel messaggio con la stessa porzione della chiave esse vengono trasformate nel crittogramma in due sottosequenze identiche. Naturalmente sottosequenze identiche potrebbero essere state generate casualmente da porzioni diverse del messaggio combinate con porzioni diverse della chiave, ma ciò è assai poco probabile, e comunque tanto meno probabile quanto più sono lunghe quelle sottosequenze. Si tratta quindi di cercare coppie di posizioni p_1, p_2 nel crittogramma in cui iniziano sottosequenze identiche, perché la distanza $d = p_2 - p_1$ è con ogni probabilità uguale alla lunghezza h della chiave o a un suo multiplo. Se vi sono distanze $d', d'', \ldots$ diverse per coppie diverse di sottosequenze, h deve essere uguale a un loro divisore comune. Si determina così il valore di h e si procede quindi alla crittoanalisi monoalfabetica delle h porzioni del crittogramma. La probabilità di successo dell'attacco cresce con la lunghezza del messaggio e decresce con la lunghezza della chiave. Infatti entrambe le fasi del metodo di crittoanalisi traggono vantaggio da tali caratteristiche, che determinano l'abbondanza di sottosequenze identiche nel crittogramma per calcolare h, e la scomposizione del crittogramma in h porzioni di lunghezza sufficiente per poter eseguire su di esse un'analisi statistica.[5]

Accenniamo infine alla decifrazione dei cifrari a trasposizione. Poiché le lettere

[5]Benché sia interessante solo per motivi storici, notiamo che il cifrario di Alberti è del tutto immune da questi attacchi se la chiave viene cambiata spesso evitando pattern ripetitivi. Mantenere a lungo una chiave mette a rischio il cifrario perché, in quel tratto, la sostituzione è monoalfabetica.

che appaiono nel crittogramma sono le stesse del messaggio in chiaro non ha senso condurre un attacco statistico sulle loro frequenze che saranno automaticamente corrispondenti a quelle proprie del linguaggio utilizzato. Questa constatazione sembrerebbe mettere fuori gioco la crittoanalisi statistica; ma un aiuto ci viene dallo studio dei q-grammi. Consideriamo un cifrario a permutazione semplice. Se si conosce la lunghezza h della chiave, il crittogramma si divide in porzioni di pari lunghezza e in ciascuna di esse si cercano i gruppi di q lettere (in genere non adiacenti) che costituiscono i q-grammi più comuni del linguaggio: se uno di questi gruppi deriva effettivamente da un q-gramma del messaggio, viene scoperta la parte di permutazione (cioè della chiave) che ha permutato l'ordine delle lettere di quel q-gramma. Se non si conosce il valore di h si ripete il procedimento per vari valori plausibili, facendosi guidare da eventuali apparizioni di lettere che sono probabilmente adiacenti nel messaggio: per esempio in italiano la Q è sempre seguita da U, perciò queste due lettere devono comparire nella stessa porzione del messaggio (e quindi del crittogramma) o agli estremi di due porzioni adiacenti. Considerazioni simili valgono per gli altri cifrari a trasposizione.

Un'ultima considerazione riguarda la possibilità di scoprire il tipo di cifrario usato nell'ipotesi che questo non sia noto. Tralasciando tecniche più raffinate, notiamo che la rilevazione delle frequenze delle singole lettere del crittogramma fornisce da sola un potente indizio per discernere tra i vari tipi di cifrari: infatti nei cifrari a trasposizione l'istogramma di frequenze del crittogramma coincide approssimativamente con quello proprio del linguaggio; nei cifrari a sostituzione monoalfabetica i due istogrammi coincidono a meno di una permutazione delle lettere; nei cifrari a sostituzione polialfabetica l'istogramma del crittogramma è assai più "appiattito" di quello del linguaggio, cioè le frequenze delle diverse lettere variano tra loro assai meno.

5.7 Una storia recente: la macchina Enigma

I numerosissimi sistemi crittografici usati dall'uomo hanno subito un epocale rinnovamento con l'apparizione dei calcolatori elettronici che hanno spedito tra i ricordi tutti i metodi inventati in precedenza. Quelli che abbiamo brevemente discusso sono i progenitori di intere famiglie di cifrari alcuni dei quali, come quello di Vigenère, sono stati impiegati a lungo nella diplomazia e nelle guerre, mentre oggi una grande attenzione è volta al mondo degli affari. Tra i tanti cifrari storici, però, uno degli ultimi in ordine di tempo deve essere ricordato esplicitamente sia perché rappresenta la prima evoluzione verso sistemi automatizzati, sia per il ruolo fondamentale che ha ricoperto nella storia recente e per la grande messe di studi che furono dedicati a comprometterne la sicurezza: i quali studi sono tra i fondamenti della nascita dei

calcolatori di oggi. Si tratta del sistema *Enigma*, sviluppato in Germania a partire dal 1918 per applicazioni commerciali come esplicita estensione elettromeccanica del cifrario di Alberti, e che ebbe poi un ruolo determinante nella seconda guerra mondiale.

Molte descrizioni e fotografie di Enigma e dei suoi componenti sono reperibili in articoli e libri specializzati e ovviamente su Internet. Ci limiteremo qui a una descrizione sommaria sufficiente agli scopi di questo testo. Il sistema si presentava all'esterno come una cassetta dotata di una tastiera simile a quella delle macchine da scrivere del tempo, con ventisei tasti corrispondenti alle lettere dell'alfabeto tedesco; e di ventisei lampadine corrispondenti anch'esse alle lettere. Sulla tastiera si batteva il messaggio e le lampade indicavano il crittogramma che andava creandosi carattere per carattere; questo era tipicamente copiato con carta e penna e poi spedito al destinatario attraverso altri mezzi (in genere radio o telegrafo). Poiché il sistema aveva una struttura perfettamente reversibile, il ricevente batteva sulla tastiera il crittogramma per ricostruire il messaggio sulle lampadine.

Il cuore di Enigma era costituito da tre dischi di gomma rigida (i *rotori*) montati sullo stesso asse ma liberi di rotare in modo indipendente, e da un disco fisso (il *riflettore*). Ciascun disco era connesso al successivo attraverso dei contatti elettrici. La tastiera era collegata al primo rotore, ma durante lo sviluppo del sistema tra questi due elementi fu inserito un pannello di controllo (*plugboard*) che ne incrementava grandemente la sicurezza, come sarà spiegato nel seguito. Le due facce di ogni rotore, dette rispettivamente di *pad* e di *pin*, erano dotate di 26 contatti elettrici disposti in cerchio corrispondenti ai caratteri dell'alfabeto, e il rotore conteneva al suo interno un cablaggio fisso che collegava a coppie i contatti di una faccia con quelli dell'altra: il rotore conteneva quindi, tra pad e pin, una permutazione fissa delle lettere. Ogni tasto della tastiera era collegato elettricamente (tramite plugboard) a un contatto pad del primo rotore; ogni contatto pin di un rotore toccava il contatto pad del successivo rotore disposto nelle stessa posizione rispetto all'asse; i contatti pin del terzo rotore toccavano i contatti pad del riflettore, che aveva solo contatti di questo tipo cablati a coppie tra di loro.

La figura 5.9 riporta un esempio di assetto dei rotori con indicate solo le connessioni tra alcune coppie di contatti, mostrando così la corrispondenza tra un carattere del messaggio e il corrispondente carattere cifrato. Una caratteristica fondamentale era che i rotori non mantenevano la stessa posizione reciproca durante la cifratura, ma per ogni lettera battuta dall'operatore il primo rotore avanzava di un passo; dopo 26 passi il rotore era tornato nella posizione iniziale e avanzava di un passo il secondo rotore; quando anche questo aveva fatto una rotazione completa avanzava di un passo il terzo rotore. A causa dello spostamento dei rotori la corrispondenza tra caratteri

cambiava a ogni passo.

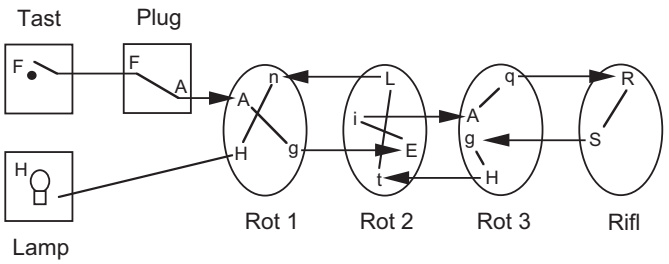


Figura 5.9: Assetto di Enigma per la cifratura di F in H. I contatti pad e pin sono indicati rispettivamente con lettere maiuscole e minuscole e sono riportati solo i cablaggi interessati alla cifratura. L'accostamento fisico dei contatti pin di un rotore con i contatti pad del successivo (frecce orizzontali) indicano la direzione del segnale.

Come possiamo valutare la sicurezza di Enigma? Ogni rotore realizza una permutazione fissa dell'alfabeto. Se i rotori fossero fermi le tre permutazioni si comporrebbero tra loro in una permutazione risultante, quindi saremmo in presenza di una sola chiave monoalfabetica la cui sicurezza, come abbiamo visto, è praticamente nulla. Poiché però i rotori si muovono uno rispetto all'altro la chiave cambia a ogni passo: 26 permutazioni si ottengono con le rotazioni del primo rotore rispetto al secondo; 26 tra il secondo e il terzo; 26 tra il terzo e il riflettore; per un totale di $26 \times 26 \times 26 = 17576$ chiavi diverse in un assetto polialfabetico. Le chiavi quindi sono molto numerose, ma in crittografia si deve sempre esercitare una grande cautela prima di tirare delle conclusioni: infatti poiché i rotori sono oggetti fisici immutabili quelle chiavi sarebbero sempre le stesse, applicate sempre nello stesso ordine: un evidente segno di debolezza rispetto a una variazione imprevedibile delle chiavi. E, assai peggio, sarebbero note a tutti i proprietari di una macchina Enigma.[6] Il problema fu risolto con vari meccanismi che resero la rottura del cifrario veramente molto ardua.

Anzitutto la macchina fu costruita in modo che i tre rotori potessero essere permutati tra loro: poiché vi sono sei permutazioni di tre elementi il numero di chiavi diviene $17576 \times 6 > 10^5$, e la serie di chiavi dipende dalla permutazione scelta. Inoltre, e questo è un fatto cruciale, dopo l'entrata in funzione dei primi modelli fu aggiunto tra la tastiera e il primo rotore un "plugboard (cioè un pannello di commutazione) che consentiva di scambiare tra loro i caratteri di sei coppie scelte arbitrariamente in

[6] A questo proposito si ricordi che Alberti aveva previsto che ogni coppia di utenti possedesse una propria coppia di dischi, diversa da tutte le altre.

ogni trasmissione mediante fili muniti di spinotti, simili a quelli usati a quei tempi dagli operatori delle centrali telefoniche. In tal modo, considerando che ogni cablaggio può essere descritto come una sequenza di 12 caratteri interpretati due a due come le coppie da scambiare, si potevano realizzare $\binom{26}{12} > 10^{10}$ combinazioni imprevedibili poiché tante sono le combinazioni di 26 elementi in gruppi di 12.

Per calcolare il numero di chiavi possibili dobbiamo considerare che ognuno di quei gruppi può presentarsi in 12! permutazioni differenti, ma non tutte producono diversi effetti. Per esempio le due permutazioni:

A B C D E F G H I J K L (A si scambia con B, C si scambia con D, ecc.), e

C D A B E F G H I J K L

producono lo stesso effetto, e con esse tutte le 6! permutazioni in cui le sei coppie si scambiano tra loro. In complesso l'introduzione del plugboard moltiplica il numero di chiavi dei rotori per $\frac{12!}{6!}\binom{26}{12} > 10^{15}$, per un totale di più di 10^{20} chiavi.

Inoltre durante la seconda guerra mondiale i tedeschi dotarono le macchine Enigma di otto rotori diversi di cui ne venivano montati di volta in volta tre; stabilirono che le posizioni iniziali mutue dei rotori fossero scelte arbitrariamente; e aumentarono a dieci le coppie di caratteri scambiabili nel plugboard.

Come conseguenza di tutti questi modi di funzionamento ogni trasmissione iniziava con una parte, a sua volta cifrata, che descriveva l'assetto complessivo dei rotori e del plugboard. Più precisamente i reparti militari possedevano un segretissimo elenco di *chiavi giornaliere* ciascuna delle quali stabiliva l'assetto iniziale della macchina per quel giorno. Con questo assetto si trasmetteva una nuova *chiave di messaggio* che indicava il nuovo assetto da usare in quella particolare trasmissione.

Come accennato la storia di Enigma è piena di capitoli interessanti che escono dai propositi di questo testo: ne ricordiamo brevemente uno per la sua connessione con la nascita del calcolo elettronico. Alcuni eccellenti matematici, prima polacchi e successivamente inglesi, studiarono come "rompere il cifrario Enigma che era divenuto lo standard militare tedesco durante la seconda guerra mondiale. In particolare in Inghilterra fu costituito il centro di Bletchley Park per la decrittazione delle comunicazioni radio tedesche. Benché la struttura di Enigma fosse nota il problema era difficilissimo perchè i tedeschi praticavano continue variazioni del sistema tenute ovviamente segrete, ed era d'altra parte necessario decifrare i messaggi in tempo brevissimo (non serve a gran che sapere che sta arrivando una bomba il giorno dopo il fatto). L'approccio seguito nel progetto dei polacchi, trasferito poi agli inglesi, fu quello di costruire un simulatore di Enigma e studiarne il comportamento sotto possibili variazioni. La versione conclusiva della macchina, nota come *Colossus*, fu costruita nel 1944 con la fondamentale consulenza di un gruppo di matematici tra cui

Alan Turing e presentava caratteri embrionali dei successivi calcolatori elettronici. Il successo di questi studi fu decisivo, e nella parte finale della guerra gli inglesi commisero volutamente alcuni errori militari secondari perché i tedeschi non si rendessero conto che le loro comunicazioni venivano intercettate e decifrate. A parziale ridimensionamento di tanta sbandierata gloria bisogna aggiungere che una parte del successo fu dovuto alla cattura di un sottomarino tedesco che gli inglesi fecero immediatamente saltare in aria dopo avervi asportato la macchina Enigma, inducendo i tedeschi a credere che il sottomarino fosse affondato in uno scontro diretto e la macchina fosse andata perduta.

Capitolo 6

Cifrari perfetti

Nel capitolo precedente abbiamo presentato alcuni cifrari molto semplici che possono essere utilizzati servendosi di carta e penna, e abbiamo indicato le tecniche per violarli. Anche il cifrario Enigma, che richiese un colossale lavoro di crittoanalisi, non era intrinsecamente più avanzato degli altri ma solo estremamente più complicato. Procediamo ora dall'estremo opposto chiedendoci se esistono cifrari inviolabili.

Può sembrare paradossale ma non vi è risposta soddisfacente a questa domanda. Esistono infatti cifrari in grado di nascondere l'informazione con certezza assoluta ma a un costo così alto da mettere in dubbio la loro esistenza pratica: il loro impiego, o meglio l'impiego di una buona approssimazione di essi, è limitato a comunicazioni sporadiche in casi di estrema segretezza. Ed esistono cifrari non altrettanto sofisticati, ma sufficientemente sicuri ed economici, che vengono utilizzati ogni giorno per comunicazioni di massa. Appartiene al primo tipo il cifrario *One-Time Pad*, protagonista del presente capitolo; ma prima di procedere dobbiamo definire con precisione i termini del discorso

In un famoso articolo del 1949 Claude Shannon formalizzò il processo crittografico mediante un modello matematico: in realtà egli aveva ottenuto quei risultati alcuni anni prima, ma la loro pubblicazione fu rimandata al periodo postbellico per motivi di segretezza militare. Fino a quel momento un sistema era informalmente considerato "perfetto" se la sua sicurezza era garantita qualunque fosse l'informazione carpita sul canale di trasmissione: in un tale sistema il crittoanalista, esaminando un crittogramma c, non doveva poter acquisire sul messaggio m alcuna conoscenza di cui non disponesse già da prima. Il contributo cruciale di Shannon fu la formalizzazione matematica del concetto di *conoscenza* legata ad una comunicazione segreta, e la conseguente definizione formale di cifrario perfetto. Seguiamo dunque la sua impostazione.

La comunicazione tra un mittente `Mitt` e un destinatario `Dest` è studiata come

un processo stocastico in cui il comportamento del mittente è descritto da una variabile aleatoria M che assume valori nello spazio `Msg` dei messaggi, e le comunicazioni sul canale sono descritte da una variabile aleatoria C che assume valori nello spazio `Critto` dei crittogrammi. La distribuzione di probabilità della M dipende dalle caratteristiche della sorgente, cioè dalla propensione di `Mitt` a spedire diversi messaggi. Per ogni $m \in$ `Msg` e per ogni $c \in$ `Critto` definiamo:

- $\mathcal{P}r(M = m)$ come la probabilità che `Mitt` voglia spedire il messaggio m a `Dest`;
- $\mathcal{P}r(M = m \,|\, C = c)$ come la probabilità a posteriori che il messaggio inviato sia effettivamente m dato che c è il crittogramma in transito.

Per accertare l'assoluta sicurezza del sistema crittografico poniamoci nella situazione di massimo pessimismo in cui il crittoanalista che ha intercettato c sia in possesso di tutta l'informazione possibile sul sistema tranne la chiave segreta. In particolare egli conosce la distribuzione di probabilità con cui `Mitt` genera i messaggi, il cifrario utilizzato e lo spazio `Key` delle chiavi. Possiamo ora riportare la definizione di Shannon:

Definizione. *Un cifrario è* perfetto *se per ogni* $m \in$ `Msg` *e per ogni* $c \in$ `Critto` *vale la relazione* $\mathcal{P}r(M{=}m \,|\, C{=}c) = \mathcal{P}r(M{=}m)$.

A parole un cifrario è perfetto se la probabilità che `Mitt` invii un messaggio m è uguale alla probabilità che il messaggio inviato sia effettivamente m se il crittogramma c transita sul canale, e questa proprietà vale per qualunque m e c. Quindi fissato un particolare messaggio m, la probabilità che esso sia stato inviato è la stessa qualunque sia il crittogramma c in transito, poiché $\mathcal{P}r(M{=}m \,|\, C{=}c)$ dipende ora solo da m. Una immediata conseguenza della definizione è che *impiegando un cifrario perfetto la conoscenza complessiva del crittoanalista non cambia dopo che egli ha osservato un crittogramma arbitrario c transitare sul canale.* Infatti dalla precedente conoscenza del cifrario e dello spazio delle chiavi egli può ricostruire il valore di $\mathcal{P}r(M{=}m \,|\, C{=}c)$ per ogni c e m. Poiché egli conosce anche $\mathcal{P}r(M{=}m)$ non potrà che confermare l'uguaglianza delle due probabilità senza inferire nulla di nuovo su m. Per comprendere meglio questo punto immaginiamo che sia $\mathcal{P}r(M{=}m) = p$, $0 < p < 1$, che il cifrario non sia perfetto e che si verifichi una delle condizioni limite $\mathcal{P}r(M{=}m \,|\, C{=}c) = 0$, o $\mathcal{P}r(M{=}m \,|\, C{=}c) = 1$ (quindi $\mathcal{P}r(M{=}m \,|\, C{=}c) \neq \mathcal{P}r(M{=}m)$). Nel primo caso, osservando c e sapendo che $\mathcal{P}r(M{=}m \,|\, C{=}c) = 0$, il crittoanalista deduce che il messaggio spedito non è m, mentre prima sapeva che sarebbe potuto transitare m con probabilità p. Nel secondo caso deduce addirittura che il messaggio spedito è proprio m. In tutti i casi intermedi egli raffina la sua conoscenza sul possibile messaggio spedito: tale conoscenza rimane inalterata solo se $\mathcal{P}r(M{=}m \,|\, C{=}c) = \mathcal{P}r(M{=}m)$.

Attraverso la definizione di Shannon si formalizza il concetto di perfetta segretezza. Dalla definizione discende però un importante teorema, anch'esso dovuto a Shannon, che sottolinea la non praticità dei cifrari perfetti:

Teorema 6.1. *In un cifrario perfetto il numero delle chiavi deve essere maggiore o uguale al numero dei messaggi possibili.*

La dimostrazione è molto semplice. Sia N_m il numero di messaggi possibili, cioè tali che $\mathcal{P}r(M{=}m) > 0$, e sia N_k il numero di chiavi. Poniamo per assurdo che sia $N_m > N_k$. A un crittogramma c, con $\mathcal{P}r(C{=}c) > 0$, corrispondono $s \leq N_k$ messaggi (non necessariamente distinti) ottenuti decrittando c con tutte le chiavi (figura 6.1). Poiché $N_m > N_k \geq s$, esiste almeno un messaggio m con $\mathcal{P}r(M{=}m) > 0$ non ottenibile da c. Questo implica $\mathcal{P}r(M{=}m \,|\, C{=}c) = 0 \neq \mathcal{P}r(M{=}m)$, ovvero per $N_m > N_k$ il cifrario non è perfetto.

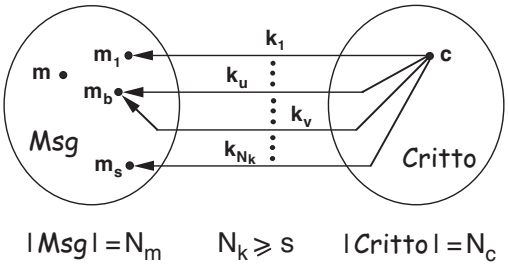


Figura 6.1: Decifrazione del crittogramma c con N_k chiavi diverse, ottenendo $s \leq N_k$ messaggi: è possibile che due chiavi k_u, k_v generino da c lo stesso messaggio m_b. Il messaggio m non è raggiungibile da c.

Questo teorema dimostra che l'uso di cifrari perfetti è necessariamente molto costoso poiché richiede chiavi lunghissime per descrivere l'intero spazio `Key`, più numeroso di quello di tutti i messaggi possibili, con conseguenti difficoltà nella gestione e nello scambio segreto delle chiavi stesse. I cifrari perfetti sono dunque inadatti a una crittografia di massa, ma risultano estremamente attraenti per chi richieda una sicurezza assoluta e sia disposto a pagarne i costi. Discuteremo ora la struttura, i pregi e i difetti del più importante di questi cifrari.

6.1 Il cifrario One-Time Pad

One-Time Pad è un cifrario perfetto molto semplice e veloce inventato da Mauborgne e Vernam nel 1917. Oltre che per l'efficienza della sua realizzazione, l'interesse per questo cifrario è giustificato da motivi storici poiché fu utilizzato, a quanto si sa, per le comunicazioni diplomatiche tra Washington e Mosca durante la guerra fredda: un mezzo di comunicazione vocale codificata in binario che ha preso il famoso nome di "telefono rosso". Il suo alto costo è legato all'impiego di una sequenza casuale di bit di lunghezza non limitata a priori: tale sequenza, che costituisce la chiave segreta, è una risorsa computazionale preziosissima di cui è persino dubbia la possibilità di generazione (vedi capitolo 4). Il nome del cifrario si riferisce familiarmente alla sequenza della chiave come se fosse scritta su un blocco di appunti (*pad*), e indica come tale sequenza venga progressivamente utilizzata per la cifratura e non sia riutilizzabile (*one-time*).

Per utilizzare il cifrario dobbiamo anzitutto assumere che i messaggi, le chiavi e i crittogrammi siano *sequenze binarie* arbitrariamente lunghe, il che è comunque ovvio se l'informazione deve essere spedita sulla rete ed è memorizzata ed elaborata da calcolatori. Si noti che qualsiasi meccanismo di trasformazione del messaggio da codice originale (per esempio linguaggio naturale) a sequenza binaria (per esempio in codice ASCII) è pubblico, per cui tale trasformazione non ha nulla a che vedere con la segretezza della comunicazione.

Le funzioni di cifratura e di decifrazione eseguiranno trasformazioni tra sequenze di bit preservando la loro lunghezza: dato un messaggio m di n bit e una chiave k, la cifratura produrrà un crittogramma c anch'esso di n bit. Le trasformazioni indotte dalle funzioni di cifratura $\mathcal{C}(m,k)$ e di decifrazione $\mathcal{D}(c,k)$ sono basate su un'operazione binaria che prende il nome di `OR ESCLUSIVO` o `XOR` (simbolo $\oplus$) definita come segue:

$$0 \oplus 0 = 1 \oplus 1 = 0$$
$$0 \oplus 1 = 1 \oplus 0 = 1$$

ovvero il risultato è uguale a 1 se e solo se i due bit su cui si opera sono diversi. L'operazione è commutativa, quindi indicheremo catene di `XOR` senza introdurvi parentesi, e gode della proprietà:

- date due variabili binarie x e y si ha: $x \oplus y \oplus y = x$.

Abbiamo così tutti gli ingredienti per definire il processo di cifratura e decifrazione di One-Time pad, che risulta molto veloce e facilmente realizzabile sia in hardware che in software.

Generazione della chiave segreta: Si costruisce una sequenza $k = k_1k_2\ldots$ di bit, nota a `Mitt` e a `Dest`, avente lunghezza maggiore o uguale a quella del messaggio da scambiare. Ogni bit k_i di k è scelto *perfettamente a caso* tra i valori 0 e 1.

Cifratura: Se il messaggio da trasmettere è la sequenza di n bit $m = m_1m_2\ldots m_n$, il crittogramma $c = c_1c_2\ldots c_n$ si genera bit a bit ponendo $c_i = m_i \oplus k_i$, per $i = 1, 2, \ldots, n$.

Decifrazione: Presi gli n bit iniziali $k_1k_2\ldots k_n$ della chiave segreta e il crittogramma c, il messaggio m è ricostruito bit a bit ponendo $m_i = c_i \oplus k_i$, per $i = 1, 2, \ldots, n$. (Infatti si ha $c_i \oplus k_i = m_i \oplus k_i \oplus k_i = m_i$ per la proprietà indicata sopra.)

Per esempio da $m = 01010101$ e $k = 01001110\ldots$ si ottiene il crittogramma $c = 00011011$. Si noti che il messaggio è periodico di periodo 2 ma il crittogramma non lo è: infatti uno stesso bit viene in genere cifrato in modo diverso a seconda del bit corrispondente nella chiave, rompendo così eventuali *pattern* esistenti nel messaggio in chiaro.

Un problema molto serio è quello della disponibilità di una chiave arbitrariamente lunga e perfettamente casuale. Approfondiremo questo punto nel prossimo paragrafo notando per ora che se una tale chiave potesse essere generata in modo economico ci troveremmo di fronte al miglior cifrario che si possa desiderare perché, oltre a essere semplicissimo, è perfetto e minimale nel senso che ora dimostreremo. Facciamo anzitutto due ipotesi:

Ipotesi 1. *Tutti i messaggi hanno la stessa lunghezza n.*

Ipotesi 2. *Tutte le sequenze di n bit sono messaggi possibili.*

In realtà non si tratta di una grave forzatura. Se i messaggi avessero lunghezze diverse tra loro, la lunghezza di uno di essi darebbe informazione al crittoanalista che escluderebbe automaticamente i messaggi più lunghi: è quindi sempre opportuno adottare uno schema in cui i messaggi più corti di n vengono completati con bit inutili, quelli più lunghi vengono divisi in blocchi. L'ipotesi che tutte le sequenze binarie siano messaggi reali è invece assai più discutibile: torneremo nel seguito su questo punto, immaginando per ora di assegnare probabilità bassissime alle sequenze non significative e di trasmetterle raramente previo accordo tra gli utenti. Possiamo così porre:

Teorema 6.2. *Sotto le ipotesi 1 e 2, e utilizzando una chiave scelta perfettamente a caso per ogni messaggio, il cifrario One-Time Pad è perfetto e impiega un numero minimo di chiavi.*

La dimostrazione di perfezione del cifrario richiede l'applicazione di alcune semplici proprietà di calcolo delle probabilità. Il lettore fiducioso può saltare direttamente al prossimo paragrafo notando però che, una volta provata la perfezione, la minimalità del numero di chiavi discende immediatamente dal Teorema 6.1, perché le chiavi stesse sono sequenze arbitrarie di n bit e dunque si ha $|\texttt{Key}| = |\texttt{Msg}| = 2^n$.

Dimostrare che il cifrario è perfetto significa provare la validità della relazione $\mathcal{P}r(M{=}m \,|\, C{=}c) = \mathcal{P}r(M{=}m)$. Applicando la definizione di probabilità condizionale possiamo riscrivere il termine sinistro della formula come:

$$\Pr(M{=}m \,|\, C{=}c) = \Pr(M{=}m, C{=}c) / \Pr(C{=}c). \qquad [6.1]$$

Osserviamo ora che l'evento $\{M{=}m, C{=}c\}$ descrive la situazione in cui `Mitt` ha generato il messaggio m e l'ha cifrato come crittogramma c. Per la definizione di `XOR`, fissato il messaggio, chiavi diverse danno origine a crittogrammi diversi, e ogni chiave può essere generata con probabilità $(1/2)^n$. Dunque, fissato comunque m, risulta $\mathcal{P}r(C{=}c) = (1/2)^n$ per ogni c, cioè la probabilità dell'evento $\{C{=}c\}$ è costante e quindi indipendente da m, ovvero gli eventi $\{M{=}m\}$ e $\{C{=}c\}$ sono indipendenti tra loro e si ha:

$$\mathcal{P}r(M{=}m, C{=}c) = \mathcal{P}r(M{=}m) \times \mathcal{P}r(C{=}c). \qquad [6.2]$$

Combinando le equazioni [6.1] e [6.2] otteniamo infine $\mathcal{P}r(M{=}m \,|\, C{=}c) = \mathcal{P}r(M{=}m)$, cioè il cifrario One-Time Pad è perfetto.

In sostanza, impiegando One-Time Pad, messaggio in chiaro e crittogramma risultano del tutto scorrelati tra loro se sono osservati dall'esterno, a sottolineare come in un cifrario perfetto *nessuna informazione* può filtrare dal crittogramma se non si conosce la chiave.

6.2 Generazione della chiave

La prima osservazione sul cifrario One-Time Pad è che non ha senso condurre un attacco esauriente sulle chiavi poiché, a parte l'improponibilità computazionale, ogni chiave farebbe ricostruire un messaggio legittimo senza aggiungere su questo alcuna informazione che non fosse già nota. Il problema della generazione della chiave merita invece un attento esame.

Se `Mitt` e `Dest` dovessero scambiarsi la chiave ogni volta che vogliono comunicare, il cifrario risulterebbe del tutto inutile: infatti invece di scambiarsi segretamente la chiave potrebbero scambiarsi direttamente e segretamente il messaggio, dato che chiave e messaggio hanno la stessa lunghezza. Dunque `Mitt` e `Dest` devono accordarsi

preventivamente su una sequenza k^* di bit casuali molto lunga, da "consumare" man mano che la comunicazione procede. Ogni volta che deve essere spedito un messaggio di lunghezza n, `Mitt` esegue la cifratura con i primi n bit di k^* che non sono stati ancora utilizzati, e `Dest` esegue la decifrazione impiegando gli stessi bit sulla sua copia di k^*.

La sequenza k^* deve essere via via scartata dopo il suo impiego, cioè gli stessi bit non devono essere utilizzati per cifrature successive, altrimenti il cifrario sarebbe esposto a un semplice attacco di tipo *cipher text* (capitolo 1). Poniamo infatti che due messaggi o porzioni di messaggi m', m'' di n bit siano cifrati con la stessa porzione k della chiave, generando i due crittogrammi c', c''. Abbiamo: $c'_i = m'_i \oplus k_i$, $c''_i = m''_i \oplus k_i$ e quindi $c'_i \oplus c''_i = m'_i \oplus m''_i$, per $1 \leq i \leq n$. Questa relazione fornisce importanti informazioni a un crittoanalista in possesso dei due crittogrammi: per esempio una sequenza di zeri consecutivi nello `XOR` tra c' e c'' corrisponde a tratti identici di m' e m''.

In conclusione affinché il cifrario sia perfetto la chiave deve essere scelta a caso (Teorema 6.2), e affinché il cifrario non sia facilmente attaccabile bisogna usare una sola volta i bit della chiave. Per soddisfare entrambe le richieste è quindi necessario generare *molti bit casuali*, problema per niente banale. Come abbiamo visto nel capitolo 4, una presumibile garanzia di casualità potrebbe essere fornita da sofisticati generatori basati su sorgenti fisiche "random"; ma poiché queste non sono per definizione disponibili in due copie identiche, `Mitt` e `Dest` dovrebbero generare e scambiarsi una lunghissima chiave prima di iniziare i loro colloqui. Bisogna quindi ricorrere a generatori pseudo-casuali disponibili in software in molte librerie standard, consci che le proprietà del cifrario saranno valide solo "con una certa approssimazione" che non è semplice definire o quantificare. E anche in questo caso si deve operare con grande cautela.

Ricordiamo dal capitolo 4 che un generatore di bit pseudo-casuali è un algoritmo che, partendo da un valore numerico iniziale scelto arbitrariamente (il *seme*), genera una sequenza di bit "apparentemente" casuali (il *periodo*) che si ripete indefinitamente. La lunghezza del periodo dipende dalle caratteristiche del generatore che in sostanza è impiegato come "amplificatore di casualità apparente". All'interno del cifrario è indispensabile che `Mitt` e `Dest` utilizzino due generatori crittograficamente sicuri e identici, e si sincronizzino sullo stesso punto iniziale della sequenza. Vi sono sostanzialmente due modi per inserire il generatore nel sistema, ed entrambi presentano punti di possibile debolezza:

1. Il generatore è pubblicamente noto e il suo seme costituisce l'informazione segreta del cifrario. Questo schema è consistente con le ipotesi più volte poste sui sistemi crittografici, ma espone il cifrario a un banale attacco esauriente

sull'insieme dei semi possibili se il numero di questi non è altissimo: il seme deve essere quindi espresso con molte cifre, il che non si verifica con i generatori più comuni. Inoltre, poiché è noto il generatore e quindi la lunghezza del suo periodo, il cifrario è esposto a un semplice attacco dovuto all'impiego ripetuto della chiave come abbiamo spiegato all'inizio di questo paragrafo. Questo attacco si può sventare impiegando generatori con periodi molto grandi oppure un nuovo seme per ogni nuovo messaggio trasmesso. Ciò però costringe `Mitt` e `Dest` ad accordarsi su un gran numero di semi che devono essere lunghi per evitare l'attacco esauriente su di essi, messaggio per messaggio.

2. Il generatore e il suo seme costituiscono la chiave segreta. Questa soluzione è in aperto contrasto col principio che gli algoritmi impiegati in un cifrario (in questo caso il generatore) siano pubblici; ma non è poi così assurda se si considera che i generatori pseudo-casuali sono in genere realizzati con programmi brevissimi, tipicamente poche decine di istruzioni in linguaggio macchina, per cui non ne è difficile lo scambio segreto. Ciò sventa gli attacchi a cui è soggetto lo schema del punto precedente, ma richiede che `Mitt` e `Dest` definiscano un loro generatore privato per evitare un attacco esauriente sui generatori noti e i loro semi.[1]

Un compromesso tra i due metodi, utilizzabile con ottimi risultati a livello dilettantistico, è quello di prendere come chiave un file binario arbitrario, lungo e disponibile a tutti (per esempio sulla rete), su cui `Mitt` e `Dest` si possono accordare scambiandosi solo pochi byte (nome del file e punto da dove iniziare a scorrerlo). Il cifrario è difficilissimo da attaccare se il file chiave è ragionevolmente casuale, ed è stato acquisito da `Mitt` e `Dest` in un periodo non sospetto, cioè quando le loro trasmissioni non erano controllate. Ottimi candidati a questo scopo sono i file che rappresentano lunghi tratti di DNA codificante acquisibili via rete dalle banche di dati genomiche.

Comunque si guardino le cose, ogni realizzazione del cifrario *One-Time Pad* che sia davvero buona è inevitabilmente molto costosa. Proviamo allora ad affrontare il problema da un altro punto di vista: poniamo che la comunicazione si svolga in un linguaggio naturale prestabilito, e rilasciamo l'ipotesi 2, su cui si basa il Teorema 6.2, che tutti i 2^n possibili messaggi di n bit siano legittimi. E in effetti si può dimostrare che la validità del teorema si mantiene senza questa ipotesi per quanto riguarda la perfezione del cifrario, ma il numero di chiavi può non essere minimo. L'ipotesi 1 che tutti i messaggi abbiano la stessa lunghezza n deve essere invece mantenuta per

[1] Non è noto come sia stata generata o scambiata la chiave per il telefono rosso tra Washington e Mosca. Nei prossimi capitoli studieremo diversi metodi per lo scambio di chiavi usati oggi e a quei tempi ufficialmente non noti. Non è escluso che qualcuno di questi metodi fosse stato già scoperto e impiegato dai servizi segreti dei due paesi senza svelarlo.

i motivi a suo tempo esposti. Studiamo allora la composizione dei messaggi da un punto di vista statistico ponendo alcune restrizioni sulle caratteristiche della sorgente aleatoria che rappresenta `Mitt`.

Si parte in genere dall'ipotesi che la sorgente generi sequenze di caratteri in accordo alla distribuzione di probabilità data dalla frequenza empirica con cui le lettere dell'alfabeto appaiono nella lingua considerata, e si raffina poi il modello aggiungendo memoria alla sorgente per imporre che la probabilità di generare un carattere sia funzione dei $q-1$ caratteri che lo precedono, in accordo alla frequenza dei q-grammi del linguaggio. Si ottiene una "fedeltà" tanto migliore quanto più grande è q, e in particolare non si generano i messaggi che contengono sequenze inesistenti nel linguaggio. Si tratta sempre di una grossolana approssimazione perché ogni frase reale è costruita secondo leggi morfologico/semantiche del tutto diverse, ma il modello risultante è in genere sufficiente per uno studio statistico ragionevolmente significativo. D'altro canto si stima che nei linguaggi naturali il numero di messaggi di n bit semanticamente validi sia approssimativamente espresso dalla funzione α^n, ove $\alpha < 2$ è una costante propria del linguaggio (per esempio $\alpha \approx 1.1$ nell'inglese); e si può notare che anche la generazione di messaggi basata sulla frequenza dei q-grammi segue in prima approssimazione un andamento simile.[2] Ma se il numero di messaggi è α^n anziché 2^n anche il numero complessivo di chiavi potrebbe essere ridotto, pur mantenendo per esse una lunghezza n pari a quella del messaggio per utilizzare One-Time Pad. Chiediamoci dunque quale sia un limite inferiore t al numero di bit casuali di cui si deve necessariamente disporre per mantenere la sicurezza del cifrario, estendendo poi ogni chiave da t a n bit in modo deterministico. Riferiamoci come esempio alla lingua inglese. Applicando il Teorema 6.1 abbiamo ora: $2^t \geq \alpha^n$, ovvero $t \geq n \log_2 \alpha \approx 0.12\, n$: il numero di bit della chiave passa così da n a $0.12\, n$, ma un ragionamento più sottile consiglia di innalzare questo limite.

Se il numero di messaggi α^n è molto minore delle 2^n possibili sequenze binarie che costituiscono i possibili crittogrammi, è poco probabile che due messaggi diversi cifrati con due chiavi diverse diano luogo allo stesso crittogramma (figura 6.2). Questo invece certamente accade se i messaggi possibili sono 2^n, poiché in questo caso ciascun crittogramma è generato da tutti i messaggi applicando ad essi chiavi diverse.

Se nella decifrazione una sola chiave trasforma un crittogramma c in un messaggio significativo m, un crittoanalista potrebbe decifrare m con un attacco esauriente sulle chiavi. Se invece, utilizzando chiavi diverse su c, si ricostruiscono messaggi significativi diversi, il crittoanalista non potrebbe stabilire quale sia quello effettivamente spedito: ed è proprio in questo che risiede la grande forza di One-Time Pad anche

[2] Si noti che il numero dei messaggi semanticamente validi cresce con la loro lunghezza n perché tra questi si trovano anche tutti i messaggi più brevi di n che sono estesi a n con un *padding* finale.

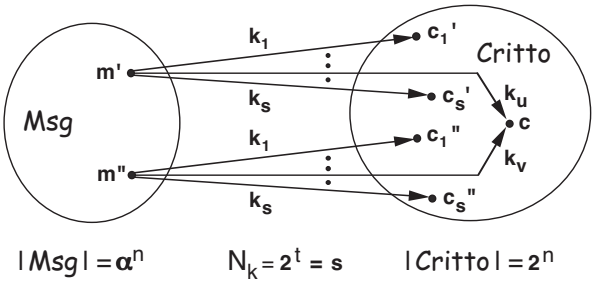


Figura 6.2: Applicando $s = 2^t$ chiavi diverse ad α^n nessaggi si ottengono $2^t \cdot \alpha^n$ crittogrammi. Per proteggere il cifrario è necessario che da più messaggi (m' e m'') si generi lo stesso crittogramma (c) impiegando chiavi diverse (k_u e k_v).

verso un ipotetico attacco enumerativo su tutte le chiavi. Ci si protegge dunque facendo sì che da molti messaggi diversi, cifrati con chiavi diverse, si generi lo stesso crittogramma (figura 6.2). A questo scopo i crittogrammi generati da tutti i messaggi con tutte le chiavi devono avere molte ripetizioni, ovvero $\alpha^n \cdot 2^t$ deve essere molto maggiore del numero 2^n dei crittogrammi possibili. Dalla relazione: $\alpha^n \cdot 2^t \gg 2^n$ si ricava immediatamente: $t + n \log_2 \alpha \gg n$, ovvero $t \gg 0.88n$. Dunque la lunghezza della chiave rimane necessariamente assai prossima a n anche sotto queste ipotesi restrittive.

Capitolo 7

Il cifrario simmetrico standard

Nel capitolo 5 abbiamo discusso alcune tecniche di crittoanalisi statistica per forzare cifrari "semplici", basate sulla frequenza delle singole lettere o dei gruppi di lettere consecutive (q-grammi) nei linguaggi naturali. Questi studi sono interessanti anche per il crittografo cui forniscono indicazioni su come manipolare il testo in chiaro perché il cifrario possa resistere a quel tipo di attacchi. A tale proposito Shannon propose di osservare due criteri che sono tuttora alla base dei cifrari a chiave segreta o simmetrici: *diffusione* e *confusione*.

Il proposito della diffusione è quello di alterare la struttura del testo in chiaro "spargendone" i caratteri su tutto il testo cifrato. Ciò si può ottenere permutando i caratteri (o i bit) del messaggio come avviene nei cifrari a trasposizione: in tal modo la frequenza delle singole lettere rimane immutata ma si perde l'informazione sulla frequenza dei q-grammi. Oppure si combinano tra loro i caratteri del messaggio in modo che ciascun carattere del crittogramma venga a dipendere da molti di essi: così facendo si perde l'informazione sulla frequenza delle singole lettere oltre che dei q-grammi. Il proposito della confusione è invece quello di combinare in modo complesso il messaggio e la chiave per non permettere al crittoanalista di separare queste due sequenze mediante un'analisi del crittogramma. Ciò si ottiene nei cifrari a sostituzione polialfabetica in misura tanto maggiore quanto più lunga è la chiave, e in modo virtualmente perfetto nel cifrario *One-Time Pad*.

La soluzione adottata nei cifrari attuali, meno perfetti ma più economici del *One-Time Pad*, è quella di eseguire trasformazioni che dipendono in modo *non lineare* dalle sequenze di ingresso, cioè da messaggio e chiave. L'esempio classico di cifrario simmetrico con diffusione e confusione è il *Data Encryption Standard*, brevemente *DES*, introdotto dalla *IBM* nel 1977 e divenuto per ben oltre vent'anni lo standard per le comunicazioni commerciali riservate ma "non classificate". Nel *DES* i criteri di Shannon sono soddisfatti attraverso una serie di permutazioni ed espansioni dei

bit del messaggio (diffusione), e la combinazione e successiva compressione dei bit del messaggio e della chiave (confusione). Il nucleo di quest'ultima operazione è costituito da un insieme di funzioni raggruppate in un blocco detto S-box da cui dipende crucialmente la sicurezza del cifrario. È importante notare che tutte le funzioni impiegate sono immutabili e pubbliche, e infatti le descriveremo in dettaglio in un prossimo paragrafo.

7.1 Un po' di storia

Nel 1972 il *National Bureau of Standards* (*NBS*), oggi *National Institute for Security and Technology* (*NIST*), ufficio americano per la definizione degli standard, iniziò un programma per proteggere le comunicazioni non classificate, cioè in genere le comunicazioni commerciali o personali fra privati. Il progetto doveva dirigersi verso un sistema di cifratura che presentasse facilità di secretazione della chiave, sicurezza certificata ed economicità nella sostituzione della chiave in presenza di forzature.

La necessità di sviluppare un tale sistema era in quel tempo fortemente sentita poiché molte compagnie comunicavano in modo "sicuro" impiegando prodotti non certificati ed esponendosi così a possibili attacchi, in particolare da parte dei realizzatori del software stesso. Inoltre i prodotti disponibili per la sicurezza erano diversi tra loro rendendo incompatibili le comunicazioni tra sistemi differenti. Il programma della *NBS* si proponeva di definire un prodotto a chiave segreta noto a tutti (*non oscuro*), che fosse unico (*compatibile*), e che potesse essere studiato a fondo per dimostrarne la sicurezza (*ufficialmente certificato*). Nel 1973 l'*NBS* pubblicò un bando che fra i vari punti richiedeva:

- che la sicurezza dell'algoritmo risedesse nella segretezza della chiave e non nel processo di cifratura e decifrazione;
- che l'algoritmo potesse essere realizzato efficientemente in hardware.

Sorprendentemente nessuno avanzò proposte significative. Un bando successivo venne accolto dalla *IBM* che propose un sistema, il *DES*, derivato da un software già noto chiamato *Lucifer*. La *National Security Agency* (*NSA*), ente che a quel tempo possedeva tutto lo scibile sulle comunicazioni segrete, certificò il *DES*, fece commenti sulla sua struttura e propose delle variazioni tra cui la riduzione della lunghezza della chiave da 128 a 56 bit e la modifica delle funzioni contenute nella S-box. Tra le motivazioni della *NSA* vi era la necessità di dissipare ogni possibile dubbio degli utenti sulla presenza di una "porta segreta" nel cifrario, definita dai proponenti per introdursi nelle comunicazioni degli altri; ma diversi utenti replicarono

allarmati temendo un'ingerenza proprio della *NSA* nella costruzione del cifrario: le profonde conoscenze dell'agenzia erano ben note e si temeva che proprio questa potesse introdurre una propria porta segreta nelle variazioni proposte. L'*IBM* accettò le modifiche solo dopo una severa serie di test i cui criteri sono rimasti segreti. Sembra oggi di poter affermare che il comportamento della *NSA* era stato corretto: molti dei suoi commenti sono risultati condivisibili da tutti solo quindici anni più tardi.

Il *DES* fu accettato e reso pubblicamente disponibile nel 1977. Con esso finalmente esisteva un cifrario certificato ufficialmente come sicuro, noto a tutti e che tutti potevano studiare con attenzione. In realtà la *NSA* pensava che il cifrario sarebbe rimasto sconosciuto ai più essendo realizzato solo in hardware, ma l'*NBS* lo rese totalmente pubblico catalizzando l'attenzione della comunità scientifica sulla crittografia.

Il *DES* doveva essere certificato ogni cinque anni e così avvenne regolarmente fino al 1987, quando ci si chiese se la sua sicurezza fosse ormai messa in pericolo dalle nuove tecniche crittoanalitiche sviluppate nei dieci anni precedenti, nonché dall'aumento di potenza dei calcolatori che rendeva più probabili futuri attacchi esaurienti sull'insieme delle chiavi. L'*NSA* propose nuovi algoritmi realizzati in hardware per sostituire il *DES*, che non sarebbero dovuti essere pubblici per evitare l'attacco suddetto. La proposta non fu ben accettata dalla comunità anche perché poneva seri problemi di riorganizzazione dei sistemi basati sul *DES*, ormai molto diffusi. Così il cifrario fu certificato di nuovo a più riprese fino al 1999, anno in cui fu dichiarato accettabile solo per scopi limitati ammettendone un uso generale nella versione estesa *3DES* (paragrafo 7.3). Nel 2005 anche questa versione fu tolta dagli standard, benché sia tuttora (2014) largamente utilizzata.

Nel 1993 il *NIST* decise di valutare nuove proposte, e nel novembre 2001 ha scelto il successore denominato *AES* per *Advanced Encryption Standard* (paragrafo 7.4). Comunque il *DES* è un pilastro della storia della crittografia e la sua realizzazione è interessante e aiuta anche a comprendere il nuovo cifrario *AES*. Per tale motivo inizieremo il nostro studio proprio dal *DES*.

7.2 Il cifrario *DES*

Il *DES* è un cifrario simmetrico di uso generale che presenta la seguente struttura:

- Il messaggio è suddiviso in *blocchi*, ciascuno dei quali viene cifrato e decifrato indipendentemente dagli altri. Nel *DES* ogni blocco contiene 64 bit.

- Cifratura e decifrazione procedono attraverso r *fasi* successive (o *round*) in cui si ripetono le stesse operazioni. Nel *DES* si ha $r = 16$.

- La chiave segreta k è composta di b byte. In ciascun byte i primi sette bit sono scelti arbitrariamente e l'ottavo è aggiunto per il controllo di parità. Nel *DES* si ha $b = 8$, cioè la chiave consta di 64 bit di cui 56 scelti arbitrariamente e 8 di parità.

- Dalla chiave k vengono create r *sottochiavi* $k[0], k[1], \ldots, k[r-1]$ impiegate una per fase.

- Il messaggio viene diviso in due metà S e D (sinistra e destra). In ciascuna fase si eseguono le due operazioni: $S \leftarrow D$ e $D \leftarrow f(k[i-1], D, S)$, dove f è una opportuna funzione *non lineare* e $i = 1, \ldots, r$. Alla fine delle r fasi le due metà vengono nuovamente scambiate e poi concatenate per produrre il crittogramma finale.

- La decifrazione consiste nel ripetere il processo invertendo l'ordine delle chiavi, caratteristica per nulla ovvia vista la struttura del cifrario.

Lo schema appena descritto si realizza usando permutazioni di bit, espansioni e compressioni di sequenze binarie e alcune semplici funzioni combinatorie tra messaggio e chiave. L'insieme di queste operazioni garantisce che alla fine del processo ogni bit del crittogramma dipenda da tutti i bit della chiave e da tutti i bit del messaggio in chiaro, rispondendo così ai criteri di confusione e diffusione proposti da Shannon. L'estensione della chiave con i bit di parità garantisce la corretta acquisizione e memorizzazione di tutti i bit della chiave stessa, condizioni cruciali per la decifrazione dell'intero crittogramma. La stessa protezione non è richiesta per il messaggio poiché un errore locale in esso si propaga nel crittogramma, ma è poi riprodotto identico all'originale nel messaggio decifrato con danno in genere irrilevante per la trasmissione. In prima approssimazione le operazioni del *DES* sono schematizzate nella figura 7.1. Esse realizzano:

Permutazioni iniziali. PI permuta i 64 bit del messaggio in chiaro. T depura la chiave dai bit di parità e permuta gli altri 56 bit per generare la prima sottochiave $k[0]$. Le operazioni sono specificate completamente nelle tabelle di figura 7.2.

Trasformazione centrale. La sequenza permutata da PI viene decomposta in un blocco sinistro $S[0]$ e un blocco destro $D[0]$ di 32 bit ciascuno. Su questi due blocchi si eseguono, nelle successive fasi del *DES*, sedici trasformazioni strutturalmente uguali, ciascuna con una diversa sottochiave $k[i]$ di 56 bit derivata dalla chiave iniziale k. La i-esima fase riceve in ingresso tre blocchi $S[i-1], D[i-1], k[i-1]$ rispettivamente di 32, 32 e 56 bit, e produce in uscita tre nuovi blocchi $S[i], D[i], k[i]$ che costituiscono l'ingresso alla fase successiva. Dopo l'ultima fase i due blocchi $S[16]$ e $D[16]$ vengono

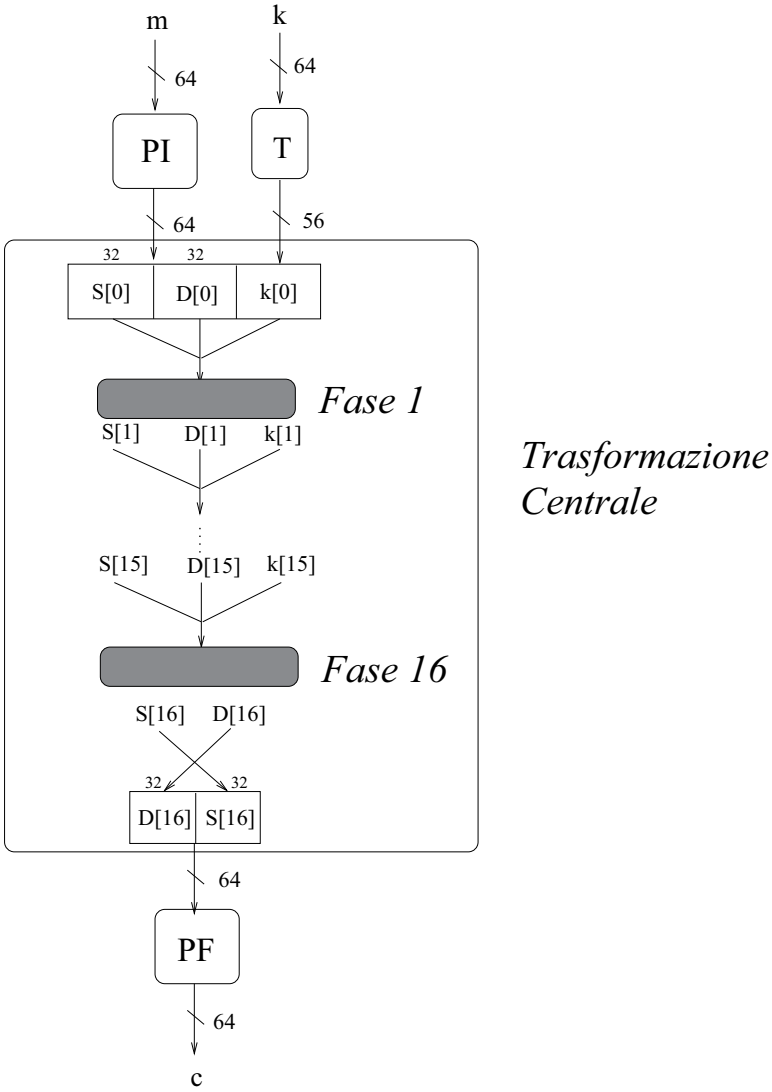


Figura 7.1: Struttura di massima del *DES*. m è un blocco del messaggio, c è il corrispondente blocco del crittogramma, k è la chiave segreta compresi i suoi bit di parità. Il numero di bit che transitano in parallelo su una linea è indicato con un trattino a fianco di essa.

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

Permutazione PI

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

Permutazione PF

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	52	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

Trasposizione T

Figura 7.2: Le permutazioni PI, PF e T. Le tabelle vanno lette per righe: per esempio PI riordina i bit del messaggio $m = m_1 m_2 ... m_{64}$ come $m_{58} m_{50} ... m_7$. PF è la permutazione inversa di PI, cioè riporta in posizione 58 il bit in posizione 1 e così via. T provvede anche a scartare dalla chiave $k = k_1 k_2 ... k_{64}$ i bit per il controllo di parità $k_8, k_{16}, ..., k_{64}$, generando una sequenza di 56 bit che costituisce la prima sottochiave $k[0]$ (si noti che questa tabella ha dimensioni 8×7).

scambiati e concatenati tra loro in un unico blocco di 64 bit da cui si costruirà il crittogramma finale.

Permutazione finale. PF genera la permutazione inversa di PI (figura 7.2) rimescolando nuovamente i bit del crittogramma.[1]

Possiamo ora studiare in dettaglio le varie fasi del *DES*. Nella figura 7.3 sono indicati il processo di cifratura nella generica fase i-esima e il meccanismo di costruzione e impiego della corrispondente sottochiave. La fase contiene le seguenti componenti:

Shift ciclico $SC[i]$. La sottochiave $k[i-1]$ di 56 bit, ricevuta dalla fase precedente, viene suddivisa in due metà di 28 bit ciascuna. Su ognuna di queste la funzione $SC[i]$ esegue uno shift ciclico verso sinistra di un numero di posizioni definito come segue: $SC[i] = 1$ per $i = 1, 2, 9, 16$, $SC[i] = 2$ altrimenti. Le due parti così traslate vengono concatenate in un unico blocco di 56 bit che costituisce la sottochiave $k[i]$ per la fase successiva.

Compressione e trasposizione CT. Il blocco di 56 bit prodotto dall'operazione precedente viene ulteriormente elaborato per produrre un nuovo blocco di 48

[1] Numerose realizzazioni hardware non comprendono le due permutazioni PI e PF inverse tra loro.

bit utilizzato nel processo di cifratura dei blocchi $S[i-1]$ e $D[i-1]$. La funzione CT esegue una permutazione del blocco e una selezione di 48 bit da esso come indicato nella figura 7.4. La combinazione delle funzioni $SC[i]$ e CT garantisce che in ogni fase venga estratto dalla sottochiave un diverso sottoinsieme di bit per la cifratura. Si calcola che nella cifratura ogni bit della chiave originale k partecipi in media a quattordici fasi.

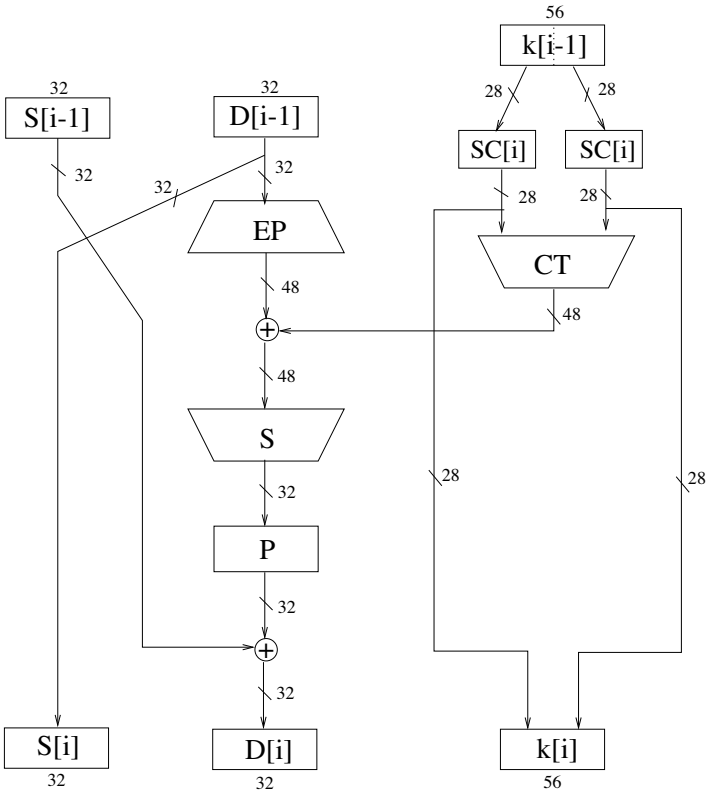


Figura 7.3: La fase i-esima del *DES*.

14	17	11	24	01	05
03	28	15	06	21	10
23	19	12	04	26	08
16	07	27	20	13	02
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

Figura 7.4: La funzione CT. La tabella va interpretata come le precedenti: otto bit dell'ingresso (per esempio il bit 09) non sono presenti in uscita.

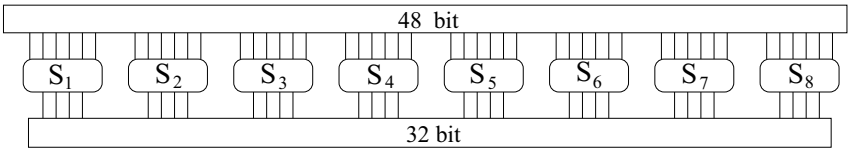
Espansione e permutazione EP. Il blocco $D[i-1]$ di 32 bit generato nella fase precedente viene espanso a 48 bit duplicando sedici bit in ingresso e spostandone altri (figura 7.5), per ottenere un blocco della stessa dimensione di quello estratto dalla sottochiave e poter eseguire lo `XOR` tra i due. Si incrementa così la dipendenza tra ingresso e uscita della fase, poiché i bit duplicati influenzano due delle sostituzioni operate dalla funzione S del blocco seguente.

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

Figura 7.5: La funzione EP. Sedici bit di ingresso vengono duplicati: per esempio il bit 32 è copiato nelle posizioni 1 e 47 dell'uscita.

Sostituzione S (o S-box). Questa funzione, progettata con molta cura dalla IBM e modificata con altrettanta cura dalla NSA, costituisce la parte cruciale e "magica" su cui si basa la sicurezza del cifrario, come spiegheremo nel seguito. Essa consta di otto sottofunzioni combinatorie $S_1, S_2, \ldots, S_8$ (figura 7.6). L'ingresso

di 48 bit viene decomposto in otto blocchi $B_1, B_2, \ldots, B_8$ di 6 bit ciascuno che costituiscono l'ingresso alle sottofunzioni di pari indice. Sia $B_j = b_1b_2b_3b_4b_5b_6$. Questi bit vengono divisi in due gruppi b_1b_6 e $b_2b_3b_4b_5$ che definiscono due numeri x, y, con $0 \leq x \leq 3$ e $0 \leq y \leq 15$, utilizzati per accedere alla cella di riga x e colonna y in una tabella che definisce la sottofunzione S_j. Il numero ivi contenuto è compreso tra 0 e 15, ed è quindi rappresentato con 4 bit che costituiscono l'uscita di S_j realizzando una compressione da 6 a 4 bit. Complessivamente gli otto blocchi generano una sequenza di 32 bit. I blocchi EP e S sono studiati in modo che tutti i bit di $D[i-1]$ influenzino l'uscita di S, senza di che non sarebbe poi possibile decifrare il messaggio.



S_1

x \ y	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	04	13	01	02	15	11	08	03	10	06	12	05	09	00	07
1	00	15	07	04	14	02	13	01	10	06	12	11	09	05	03	08
2	04	01	14	08	13	06	02	11	15	12	09	07	03	10	05	00
3	15	12	08	02	04	09	01	07	05	11	03	14	10	00	06	13

Figura 7.6: La struttura generale della S-box (in alto), e la tabella 4×16 che definisce la sottofunzione S_1. Le sottofunzioni $S_2, S_3, \ldots, S_8$ sono definite in modo simile: per esse rimandiamo ai testi specialistici.

16	*07*	*20*	*21*	*29*	*12*	*28*	*17*
01	*15*	*23*	*26*	*05*	*18*	*31*	*10*
02	*08*	*24*	*14*	*32*	*27*	*03*	*09*
19	*13*	*30*	*06*	*22*	*11*	*04*	*25*

Figura 7.7: La permutazione P.

Permutazione P**.** È una permutazione di 32 bit che genera il blocco finale $D[i]$ (figura 7.7).

Tutte le funzioni applicate dal cifrario, a eccezione della S-box, sono *lineari* se riferite alla operazione di `XOR`, cioè vale $f(x) \oplus f(y) = f(x \oplus y)$ ove f è una funzione permutazione, espansione o compressione di un vettore binario e x, y sono vettori binari arbirari. Ciò invece non si verifica se f è una delle otto funzioni della S-box che sono dunque *non lineari*: si stima che questo contribuisca in modo determinante alla sicurezza del cifrario.

Prima di addentrarci in uno studio dei possibili attacchi e delle alternative concrete che si pongono al *DES* è opportuno fare alcune considerazioni sulla sua struttura, anche perché esse si applicano a tutti i cifrari di questo tipo. I tre aspetti che sono stati più diffusamente studiati riguardano la lunghezza della chiave, il progetto della funzione S-box e il numero di fasi.

Abbiamo già detto che nel progetto iniziale la chiave era di 128 bit, ridotti poi a 56 dalla *NSA*. Discuteremo nel prossimo paragrafo l'opportunità di tale scelta, e se una chiave tanto "corta" sia adeguata a proteggere il cifrario dagli attacchi possibili oggi. Alla funzione S-box è legata una storia di sospetti reciproci tra *IBM*, *NSA* e utenti (paragrafo 7.1). Questa situazione indusse l'*NBS* a eseguire un'indagine che accertò la correttezza dell'operato di tutti i partecipanti: così il *DES*, nella forma che oggi conosciamo, venne accettato come standard e messo a disposizione del pubblico. La funzione S-box adottata si è dimostrata all'altezza del suo compito fino all'avvento di mezzi di calcolo per la crittoanalisi di potenza improponibile quando nacque il cifrario.

Anche il numero di fasi sembrava imposto arbitrariamente, ma oggi possiamo affermare che la scelta di 16 fasi era stata oculata perché, alla fine del procedimento, il crittogramma ottenuto può essere praticamente considerato una sequenza casuale. Per quanto riguarda le componenti *lineari* del *DES* (permutazioni, espansioni, ecc.) dobbiamo notare che, pur nella loro grande semplicità, hanno un ruolo importante ai fini della diffusione e confusione richieste da Shannon. Infatti tali componenti fanno sì che ogni bit subisca diverse trasformazioni in ogni fase, e rendono difficile scoprire il tracciato dei bit attraverso il sistema.

Oltre alla loro efficacia nel processo di cifratura tutti i pezzi del *DES* paiono ineliminabili per garantirne la sicurezza, come è stato messo in evidenza dai numerosissimi studi eseguiti sul cifrario: il che depone a favore della serietà del progetto originale.[2] E mentre la sicurezza del *DES* è stata convalidata dalle tecniche di crittoanalisi sviluppate successivamente, tutte le operazioni eseguite nelle varie fasi del cifrario risultano

[2]Secondo l'*IBM* il progetto era il risultato di uno studio molto approfondito e aveva richiesto un lavoro quantificabile in diciassette anni/uomo.

molto semplici dal punto di vista algoritmico, e hanno così permesso realizzazioni efficientissime. L'uniformità dei processi di cifratura e decifrazione consente inoltre di utilizzare lo stesso dispositivo per le due operazioni applicando le sottochiavi in ordine inverso, il che determina economicità e facilità di impiego del sistema.

7.3 Attacchi, variazioni e alternative al *DES*

Le prime considerazioni sulla vulnerabilità del *DES* riguardano gli attacchi esaurienti. Poiché la chiave è formata da 56 bit si dovrebbero provare in linea di principio tutte le 2^{56} chiavi possibili. Alcune osservazioni sulla struttura del cifrario permettono però di ridurre leggermente lo spazio da esplorare.

Denotiamo con $\mathcal{C}_{DES}$ e $\mathcal{D}_{DES}$ le funzioni complessive di cifratura e decifrazione ricordando che entrambe sono realizzate con la struttura di calcolo mostrata in figura 7.3. Si possono anzitutto escludere sessantaquattro chiavi *deboli*, non utilizzabili perché compromettono da sole la sicurezza del cifrario. Tra queste sono banalmente comprese la chiave composta unicamente da zeri, o unicamente da uni, o da ventotto zeri seguiti da ventotto uni, o da ventotto uni seguiti da ventotto zeri, che generano la stessa sottochiave in ogni fase del cifrario (vedi figura 7.3).

Una riduzione più interessante è legata al fatto che la relazione $\mathcal{C}_{DES}(m,k) = c$ implica $\mathcal{C}_{DES}(\bar{m},\bar{k}) = \bar{c}$, ove la barra indica la complementazione bit a bit di una sequenza binaria. Infatti le uniche funzioni del cifrario influenzate dalla complementazione dei bit d'ingresso sono `XOR` e S-box. La prima ripropone uguale uscita se entrambi gli ingressi vengono complementati, quindi l'ingresso alla S-box è identico per le coppie (m,k) e $(\bar{m},\bar{k})$, e l'uscita da questo blocco è anche identica. Il successivo blocco `XOR` incontrato nella catena di trasformazioni ha così un ingresso complementato $S[i-1]$ e uno diretto proveniente dal blocco P, e genera dunque un'uscita complementata. Basandosi su questa proprietà si può condurre un attacco di tipo *chosen plain-text* nel modo seguente. È sufficiente che il crittoanalista si procuri alcune coppie messaggio/crittogramma del tipo $\langle m, c_1\rangle$ e $\langle \bar{m}, c_2\rangle$. Si parte da una di tali coppie e, scelta una chiave k, si calcola $\mathcal{C}_{DES}(m,k)$. Se $\mathcal{C}_{DES}(m,k) = c_1$ probabilmente k è la chiave segreta (non vi è ancora certezza di questo fatto perché più chiavi potrebbero trasformare m in c_1). Si prova allora k su altre coppie messaggio/crittogramma: se il successo si ripete per alcune volte consecutive si ha la pratica certezza di aver trovato la chiave. Se invece $\mathcal{C}_{DES}(m,k) = \bar{c}_2$, probabilmente $\bar{k}$ è la chiave segreta perché genererebbe correttamente la coppia $\langle \bar{m}, c_2\rangle$: come nel caso precedente si ripete la prova su altre coppie. Infine se i due casi precedenti falliscono, quindi né k né $\bar{k}$ è la chiave segreta, si procede provando un'altra chiave. In conclu-

sione provata una chiave non è necessario ripetere l'operazione sul suo complemento e lo spazio si riduce a 2^{55} chiavi, la metà di tutte quelle possibili.

Le considerazioni fatte finora consentono di ridurre solo marginalmente lo spazio delle chiavi. Un attacco crittoanalitico completamente diverso, detto di *crittoanalisi differenziale* (1990), è di tipo *chosen plain-text* e impiega un insieme di 2^{47} messaggi in chiaro che il crittoanalista ha scelto e di cui è riuscito a procurarsi i relativi crittogrammi. Si noti che il numero di coppie $<m, c>$ note deve essere talmente alto da rendere l'attacco applicabile solo se il crittoanalista è in grado di "costringere" un sistema automatizzato a generare crittogrammi su una lunghissima sequenza di messaggi segretamente introdotti, ipotesi più accademica che reale. Scegliendo coppie di messaggi con particolari differenze, la differenza tra i relativi crittogrammi permette di assegnare probabilità diverse a chiavi diverse. Procedendo nella scelta di coppie di messaggi e nell'analisi dei crittogrammi corrispondenti la chiave cercata emerge sulle altre come quella avente probabilità massima. L'uso della crittoanalisi differenziale ha tra l'altro permesso di dimostrare che sedici fasi sono indispensabili se si vuole garantire al *DES* ragionevole sicurezza, perché questa tecnica permette di attaccare facilmente il cifrario se il numero di fasi è inferiore a sedici.[3]

Un'altra tecnica di attacco di tipo *chosen plain-text*, detta *crittoanalisi lineare* (1993), permette di inferire alcuni bit della chiave segreta mediante un'approssimazione lineare della funzione di cifratura, mentre i restanti bit sono determinati attraverso un attacco esauriente. Il numero di coppie messaggio-crittogramma da esaminare si riduce a 2^{43}, numero ancora tanto ragguardevole da rendere difficile oggi un attacco con metodi economici. Tutti questi risultati hanno però rafforzato i dubbi sulla sicurezza del cifrario, anche in relazione all'aumento di potenza dei mezzi di calcolo utilizzabili per la crittoanalisi. Il colpo finale al *DES* è stato sferrato nel 2008 con la macchina *RIVYERA*; un calcolatore, peraltro molto costoso, costruito appositamente per rompere il cifrario in meno di ventiquattr'ore. Sono stati quindi proposti diversi metodi per incrementare la sicurezza del *DES*, e diversi cifrari in alternativa.

Le variazioni del *DES* hanno sostanzialmente lo scopo di preservare la semplicità del cifrario espandendone allo stesso tempo lo spazio delle chiavi in modo da rendere improponibili gli attacchi precedenti. I metodi più interessanti sono la *scelta indipendente delle sottochiavi* e la *cifratura multipla*. Nel primo caso le sedici sottochiavi di 48 bit utilizzate nelle successive fasi vengono scelte indipendentemente l'una dall'altra, per un totale di 768 bit anziché 56, il che rende impraticabile un attacco esauriente sulle chiavi stesse. Si può dimostrare che un attacco con l'analisi differenziale richiede in questo caso l'esame 2^{61} coppie messaggio-crittogramma, ma il metodo è comunque

[3] In effetti dopo il 1990 l'*IBM* rese pubblici i criteri su cui era basato il progetto delle fasi e della *S*-box, sottolineando come la nuova tecnica di crittoanalisi fosse nota ai suoi studiosi sin dall'inizio.

condizionato al grande numero di bit che un utente deve generare per costruire la chiave.

Più rilevante in pratica è la cifratura multipla, che prevede la concatenazione di più copie del *DES* che utilizzano chiavi diverse. Si costruisce così un *cifrario composto* il che, come abbiamo già avuto modo di osservare, non garantisce in linea di principio un aumento della sicurezza del cifrario complessivo rispetto ai componenti. Nel caso del *DES* però, la concatenazione di più copie del cifrario non è equivalente a un'applicazione singola del cifrario stesso. Matematicamente ciò si esprime con l'affermazione che, date due arbitrarie chiavi k_1, k_2, si ha $\mathcal{C}_{DES}(\mathcal{C}_{DES}(m, k_1), k_2) \neq \mathcal{C}_{DES}(m, k_3)$ per qualsiasi messaggio m e qualsiasi chiave k_3.

L'aumento di sicurezza indotto dal metodo è stato confermato in pratica e invita, come naturale conseguenza, a concatenare diversi cifrari *DES*. Poiché però ciascuno di questi richiede una sua chiave segreta, occorre trovare un buon compromesso tra sicurezza e semplicità d'impiego. La proposta più importante è nota come *Triple DES* con due o tre chiavi, rispettivamente indicato con $2TDEA$ e $3TDEA$ (l'acronimo sta per Triple Data Encryption Algorithm). La cifratura con $2TDEA$ è realizzata come:

$$c = \mathcal{C}_{DES}(\mathcal{D}_{DES}(\mathcal{C}_{DES}(m, k_1), k_2), k_1),$$

dove k_1 e k_2 sono chiavi indipendenti di 56 bit segreti concordate dai due utenti, sicché la chiave complessiva risulta di 128 bit di cui 112 segreti e 16 di parità. La decifrazione segue immediatamente come:

$$m = \mathcal{D}_{DES}(\mathcal{C}_{DES}(\mathcal{D}_{DES}(c, k_1), k_2), k_1).$$

Questo schema preserva la compatibilità con le realizzazioni convenzionali a una sola chiave, poiché scegliendo $k_1 = k_2$ il cifrario diviene equivalente a un *DES* singolo. $3TDEA$ incrementa la sicurezza del metodo utilizzando tre chiavi distinte anziché due, una per ogni modulo *DES* della composizione. Queste variazioni del *DES* sono tuttora largamente usate in sistemi delicati come per esempio quelli della moneta elettronica, benché il cifrario non sia più dichiarato standard.

A parte il nuovo standard *AES* di cui parleremo nel prossimo paragrafo, sono stati proposti nel tempo molti altri sistemi a chiave segreta che hanno adottato la struttura del *DES* migliorandone alcune parti. Accenniamo brevemente ai cifrari *RC5* e *IDEA* che sono particolarmente sicuri e impiegati con una certa frequenza.

RC5 è l'acronimo di *Ron's Code 5* dal nome del suo ideatore Ron Rivest, ed è il successore di altri quattro cifrari di cui *RC2* e *RC4* sono stati ampiamente utilizzati. *RC5* segue le linee di base del *DES* ma lascia maggiore libertà agli utenti. È un cifrario a blocchi di 64 bit, con chiave di $c \times 32$ bit, organizzato in r fasi: i parametri c ed r possono essere scelti a piacere (un valore consigliato è comunque $r = 16$ come

nel *DES*). Dalla chiave vengono generate $2r+2$ sottochiavi $k[0], ..., k[2r+1]$ di 32 bit ciascuna: $k[0]$ e $k[1]$ sono impiegate in una elaborazione iniziale, $k[2i]$ e $k[2i+1]$ nella fase i-esima. Ogni blocco del messaggio viene decomposto in due sottoblocchi X e Y di 32 bit. Si inizia il processo ponendo $X \leftarrow (X+k[0])$ `mod` 2^{32} e $Y \leftarrow (Y+k[1])$ `mod` 2^{32} e i sottoblocchi subiscono poi una serie di trasformazioni nel corso delle r fasi. *RC5* associa semplicità di realizzazione a grande sicurezza. Le operazioni usate nell'intero cifrario sono lo shift ciclico, lo `XOR` e l'addizione modulo 2^{32}: a quest'ultima si deve l'interdipendenza tra bit di ingresso e uscita a causa della propagazione del riporto. In complesso il cifrario è molto veloce e resiste con successo a tutti gli attacchi standard purché i valori dei parametri c ed r siano scelti opportunamente.

IDEA è l'acronimo di *International Data Encryption Algorithm*. Introdotto nel 1992, questo cifrario come i precedenti divide il messaggio in blocchi di 64 bit. La chiave è di 128 bit, e da essa vengono estratte cinquantadue sottochiavi di 16 bit mediante un serie di shift ciclici di 25 posizioni, che garantiscono l'estrazione dalla chiave di bit sempre diversi (25 e 128 sono primi tra loro). La cifratura si sviluppa in otto fasi uguali tra loro. Le operazioni usate nell'intero cifrario sono lo shift ciclico, lo `XOR`, l'addizione `mod` 2^{16} e la moltiplicazione `mod` $(2^{16}+1)$ (che è un numero primo): queste ultime due in particolare garantiscono l'interdipendenza tra bit di ingresso e uscita. La maggiore complicazione delle operazioni di *IDEA* rispetto a *RC5* è compensata dal limitato numero di fasi. La sicurezza poggia su forti basi teoriche e in effetti il cifrario è rimasto sostanzialmente inviolato.

7.4 *AES*: il nuovo standard

Nel corso degli anni novanta dello scorso secolo il *DES* appariva ormai inadeguato alle necessità di un futuro imminente. Nel gennaio 1997 il *NIST* annunciò di voler prendere in considerazione un nuovo cifrario simmetrico per le comunicazioni non classificate, atto a svolgere la funzione di cifrario internazionale standard per il ventunesimo secolo. Si sarebbe chiamato *AES* per *Advanced Encryption Standard*. Questo obiettivo venne formalizzato in un bando pubblicato in quell'anno sul Registro Federale degli Stati Uniti, che definiva i criteri per la selezione tra i cifrari candidati e fissava al giugno del 1998 la scadenza per la presentazione delle proposte.

Il progetto dell'*AES* doveva concentrarsi su tre aspetti: sicurezza, costo di realizzazione e caratteristiche algoritmiche. La sicurezza rappresentava il fattore più importante: nel valutarla si sarebbe posta particolare attenzione alla resistenza agli attacchi crittoanalitici noti, alla correttezza del processo di cifratura, alla casualità dell'uscita prodotta, alla robustezza rispetto agli altri cifrari proposti.

La valutazione del costo si sarebbe basata principalmente sull'economicità della

realizzazione hardware, sulla velocità di cifratura e decifrazione su varie piattaforme software, sull'occupazione di memoria. Poiché l'*AES* doveva essere di pubblico dominio e di uso gratuito, era essenziale che non vi fossero vincoli derivanti dalla proprietà intellettuale di parti del cifrario. Il *NIST* confidava di portare alla luce l'esistenza di eventuali brevetti che avrebbero inficiato la gratuità del cifrario, utilizzando i commenti provenienti dalla comunità scientifica e industriale.

Il terzo requisito per la selezione consisteva nelle buone caratteristiche algoritmiche del cifrario. Esso doveva essere flessibile, quindi utilizzabile anche per altri "servizi" correlati come la generazione di funzioni hash, facilmente portabile su varie piattaforme e realizzabile in hardware, applicabile a chiavi di 128, 192 o 256 bit e alla cifratura di blocchi di almeno 128 bit.

Nell'agosto del 1998 il *NIST* annunciò una prima selezione di quindici cifrari proposti da ricercatori di università e industrie di dodici paesi, sollecitando commenti su di essi. Dopo una lunga analisi di tutti i commenti ricevuti condotta dal comitato selezionatore, il *NIST* annunciò nell'aprile del 1999 di aver scelto cinque finalisti: *MARS*, proposto dalla *IBM*; *RC6*, proposto da *RSA Laboratories* come variazione di *RC5*; *Rijndael*, proposto dalla *Proton World International* assieme alla Università Cattolica di Leuven; *Serpent*, proposto dalle università di Cambridge, Technion e San Diego; *Twofish*, di vari proponenti tra cui le università di Berkeley e Princeton. L'analisi di queste cinque proposte si è svolta tra il 1999 e il 2000, con un procedimento molto approfondito per cui il *NIST* ha continuamente sollecitato contributi, proposte e commenti da parte delle comunità accademica e industriale. La discussione è avvenuta in gran parte attraverso un forum elettronico che ha svolto la funzione di strumento per la disseminazione immediata dei risultati ottenuti sui cifrari candidati da ricercatori che li hanno esaminati in tutto il mondo. Nell'aprile 2000 è stata indetta una conferenza sull'argomento; un comitato di esperti crittografi si è poi riunito per analizzare in profondità tutti i commenti ricevuti, finché nell'ottobre 2000 il *NIST* ha dato l'annuncio della selezione dell'algoritmo candidato a costituire l'*AES*: si tratta di *Rijndael*, un prodotto europeo. Il *NIST* ha giustificato la scelta sostenendo che *Rijndael* è il cifrario che offre la migliore combinazione di sicurezza, efficienza, realizzabilità hardware e software, e flessibilità.

Un'ultima fase di commenti pubblici e di analisi dettagliate da parte del NIST si è svolta nel corso del 2001 senza che siano state riscontrate particolari deficienze di *Rijndael*; quindi il cifrario, con alcune precisazioni, è stato certificato come *AES* destinandolo a sostituire ufficialmente il *DES* e le sue variazioni correntemente in uso. Dopo una inevitabile fase transitoria *AES* avrebbe dovuto soppiantare tutti gli altri cifrari simmetrici, anche se alcuni di essi sopravvivono nelle applicazioni. Descriviamo ora a grandi linee la struttura del nuovo cifrario, notando che ha recepito molti dei

principi su cui sono basati i cifrari precedenti.

Rijndael è un cifrario a blocchi progettato dai ricercatori belgi Joan Daemen e Vincent Rijmen, e il suo nome è una contrazione dei cognomi dei progettisti. L'algoritmo, che può essere efficientemente realizzato su una vasta gamma di *processori* e su *smartcard*, prevede che i blocchi in cui è diviso il messaggio e la chiave segreta siano formati da 128,192 o 256 bit, comunque estensibili per multipli di 32 bit. Lo standard *AES* però prevede solo blocchi di 128 bit e chiavi di 128, 192 o 256 bit, e a questo ci riferiremo ora.

Come nel *DES* il processo di cifratura e decifrazione di *AES* opera per fasi, in numero di dieci, dodici o quattordici in corrispondenza alle tre lunghezze della chiave.[4] Ogni fase impiega una propria *chiave locale* creata a partire dalla chiave segreta del cifrario mediante un opportuno processo di espansione e selezione. Un'ulteriore *chiave iniziale*, ricavata con lo stesso procedimento, è impiegata all'inizio delle operazioni. Vediamo ora come funziona il cifrario nella versione con chiave di 128 bit.

$b_{0,0}$	$b_{0,1}$	$b_{0,2}$	$b_{0,3}$
$b_{1,0}$	$b_{1,1}$	$b_{1,2}$	$b_{1,3}$
$b_{2,0}$	$b_{2,1}$	$b_{2,2}$	$b_{2,3}$
$b_{3,0}$	$b_{3,1}$	$b_{3,2}$	$b_{3,3}$

B

Figura 7.8: La matrice B del cifrario *AES*. I 16 byte $b_{i,j}$ che costituiscono gli elementi della matrice B sono trasformati nel corso di ogni fase.

Ogni fase opera su un blocco di 128 bit, logicamente organizzato come matrice bidimensionale B di 16 byte (figura 7.8): il blocco della prima fase è estratto dal messaggio e il blocco finale ne costituisce il crittogramma, entrambi disposti per righe nella matrice. Anche le chiavi locali sono divise in 16 byte posti in corrispondenza ai byte di B. A grandi linee l'organizzazione è la seguente.

Trasformazione iniziale. Il messaggio m è caricato nella matrice B. Ogni byte di B è posto in `XOR` (bit a bit) con il corrispondente byte della chiave iniziale. Segue la fase sotto riportata, ripetuta per dieci volte.

[4] Il basso numero di fasi fu oggetto di numerose critiche durante il processo di selezione, ma i progettisti dimostrarono che il cifrario non era esposto ad alcun attacco crittoanalitico noto.

Organizzazione di una fase. Consiste nelle quattro operazioni:

Op1. ogni byte della matrice B è trasformato mediante una S-box;

Op2. la matrice così ottenuta è permutata mediante *shift ciclici* sulle righe;

Op3. le colonne risultanti vengono trasformate con un'operazione algebrica;

Op4. ogni byte della matrice risultante è posto in `XOR` con un byte della chiave locale per quella fase.

Più precisamente le operazioni da F1 a F4 sono eseguite come segue:

Op1. La S-box differisce da quella del *DES* per i criteri algebrici molto raffinati secondo cui è costruita la trasformazione, sui quali sarebbe fuori luogo addentrarci in questa sede: criteri che assicurano la non linearità della trasformazione e la sua sicurezza. La struttura è però la stessa del *DES* anche se, riferendosi alla figura 7.6 che indicava 8 blocchi combinatori $S_1, \ldots, S_8$ a 6 ingressi e 4 uscite, siamo ora in presenza di 16 blocchi $S_{0,0}, \ldots, S_{3,3}$ a 8 ingressi e 8 uscite, ciascuno dei quali realizza la trasformazione $b_{i,j} \leftarrow S_{i,j}(b_{i,j})$ di un byte della matrice B.

Op2. La prima riga (riga 0) della matrice B resta inalterata mentre i byte contenuti nelle righe 1, 2, 3 sono soggetti a uno *shift ciclico* verso sinistra rispettivamente di 1, 2 o 3 posizioni. Così per esempio il byte $b_{2,2}$ sarà spostato in posizione $b_{2,0}$; il byte $b_{3,1}$ sarà spostato in posizione $b_{3,2}$.

Op3. Ogni colonna della matrice, trattata come vettore di 4 elementi, viene moltiplicata per una matrice prefissata M di 4×4 byte, ove la moltiplicazione tra byte è eseguita modulo 2^8 e l'addizione è eseguita come `XOR`. Anche in questo caso non entriamo nel dettaglio, notando solo che la M è scelta in modo che ogni byte di una colonna di B influenzi, nel prodotto con la M, tutti i byte della colonna stessa.

Op4. Ogni byte $b_{i,j}$ della matrice B è trasformato come $b_{i,j} \leftarrow b_{i,j} \oplus k_{i,j}$, ove $k_{i,j}$ è il corrispondente byte della chiave locale.

Nonostante i numerosissimi studi di attacco si può affermare che a oggi (2014) nessuno di essi è stato in grado di compromettere *AES* anche nella sua versione più semplice con chiave di 128 bit. Alcuni attacchi hanno mostrato che il cifrario presenta una possibile debolezza se le fasi sono meno di dieci: ciò potrebbe indurre a credere che dieci fasi possano rivelarsi insufficienti in futuro, ma è bene osservare che, come proposto nella versione originale di *Rijndael*, il loro numero può essere aumentato senza alcuna difficoltà.[5]

[5]Deve essere riconosciuta grande correttezza a Bruce Schneier, un eminente crittografo tra i proponenti del cifrario *Twofish* che entrò nella competizione finale per la scelta di *AES*. Una sua

Come per ogni cifrario, tuttavia, è bene non abbassare la guardia poiché attacchi diversi possono venire da direzioni inaspettate con cui si superano le barriere algebriche e la complicazione della struttura (e in questo senso *AES* è complicatissimo). Infatti per *AES*, come praticamente per tutti i cifrari a chiavi simmetriche, si conoscono pericolosissimi attacchi *side-channel* che non sfruttano le caratteristiche del cifrario ma le possibili debolezze della piattaforma su cui esso è implementato.

7.5 Cifrari a composizione di blocchi

I cifrari simmetrici impiegati in pratica funzionano a blocchi, in particolare di 64 bit (8 caratteri) nel *DES* e 128 bit nell'*AES*, sicché la trasmissione risulta in un certo senso *sincopata*. Ciò può creare problemi tecnici nella trasmissione su reti ad alta velocità, ma soprattutto espone la comunicazione ad attacchi fin qui non considerati: infatti blocchi uguali nel messaggio producono blocchi cifrati uguali inducendo una periodicità nel crittogramma che fornisce utili informazioni per la crittoanalisi. Naturalmente ciò avviene se si trasmettono messaggi composti da molti blocchi, ma questo accade comunemente poiché i blocchi stessi sono piccoli. I metodi proposti per ovviare al problema intervengono componendo i blocchi tra loro: presenteremo ora il più diffuso, detto *Cipher Block Chaining* o brevemente *CBC*.

Il modo *CBC* segue il principio della diffusione di Shannon inducendo una *dipendenza di posizione* tra il blocco in elaborazione e quelli precedenti. Il cifrario risultante è ancora strutturato a blocchi, ma blocchi uguali nel messaggio vengono (con pratica certezza) cifrati in modo diverso eliminandone così la periodicità. Il metodo si impiega in cifratura e in decifrazione utilizzando le funzioni $\mathcal{C}$ e $\mathcal{D}$ del cifrario d'origine ed è basato su un'idea molto semplice. Supponiamo che il cifrario d'origine operi su blocchi di b bit con chiave k. Il messaggio m è diviso in blocchi, $m = m_1 m_2 \dots m_s$: se m_s contiene $r < b$ bit lo si completa aggiungendovi la sequenza binaria $10000\dots$ di lunghezza $b - r$, altrimenti si aggiunge al messaggio un intero nuovo blocco $m_{s+1} = 10000\dots$ di lunghezza b. In entrambi i casi la nuova sequenza funge da terminatore del messaggio. Sia ora c_0 una sequenza di b bit scelta per esempio in modo casuale e diversa per ogni messaggio (nulla osta a che c_0 sia pubblica). Il metodo è illustrato nella figura 7.9.

---

affermazione su *Rijndael* dopo che questo ebbe il sopravvento suona circa così: "Come sempre emergeranno in futuro attacchi efficaci a *Rijndael* di interesse puramente accademico, ma nella pratica non mi aspetto che si riuscirà mai a compromettere questo cifrario".

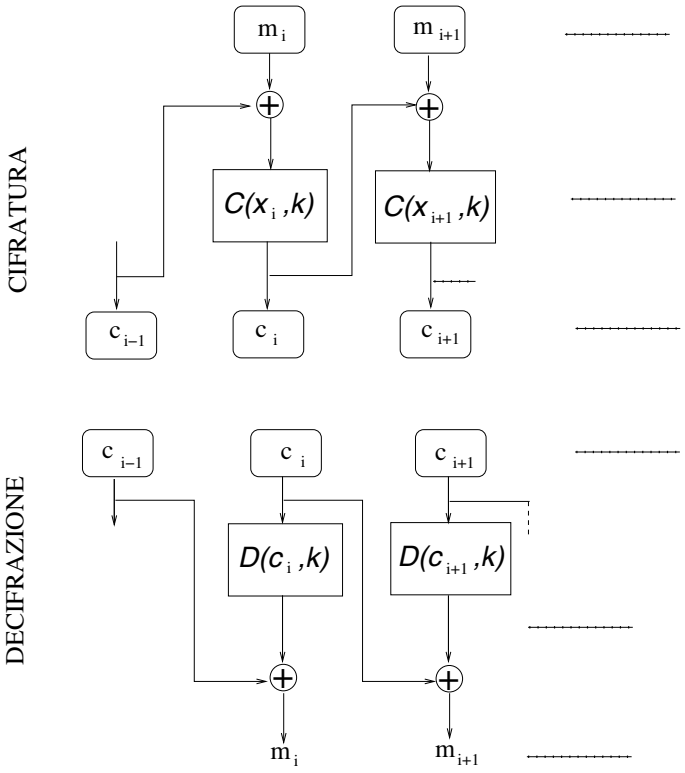


Figura 7.9: Il modo *Cipher Block Chaining* (CBC). La notazione x_i è utilizzata per indicare il bit $m_i \oplus c_{i-1}$.

Ogni blocco m_i viene cifrato come: $c_i = \mathcal{C}(m_i \oplus c_{i-1},\, k)$. La decifrazione del blocco c_i è eseguita come: $m_i = c_{i-1} \oplus \mathcal{D}(c_i, k)$. Si osservi come il processo di cifratura sia strettamente sequenziale in quanto il calcolo di ogni c_i impiega il risultato c_{i-1} del passo precedente, mentre il processo di decifrazione può essere eseguito in parallelo se tutti i blocchi cifrati sono disponibili.

L'introduzione della sequenza c_0 e la sua modifica per ogni nuovo messaggio sono necessarie per evitare che messaggi uguali generino crittogrammi uguali, o che prefissi di messaggi uguali generino prefissi di crittogrammi uguali. Essa può essere costruita

in modo pseudocasuale, per esempio mediante una *marca temporale*, cioè il segnale orario proveniente dall'orologio del calcolatore, e viene solitamente spedita in chiaro con il crittogramma e scartata all'atto della decifrazione. Tutto ciò non pregiudica in alcun modo la robustezza del sistema crittografico.

I vantaggi offerti dal metodo *CBC* dovrebbero essere evidenti. I processi di cifratura e decifrazione non sono apprezzabilmente più lenti di quelli del cifrario originale poiché le operazioni di composizione impiegano esclusivamente l'operazione di `XOR`. Più messaggi possono essere cifrati con la stessa chiave. Il testo risulta difficile da alterare da parte di un crittoanalista che potrebbe unicamente aggiungere nuovi blocchi all'estremità. Tra le proprietà più interessanti vi è quella di *resistenza agli errori* nel crittogramma, dovuti per esempio alla trasmissione o a manomissioni controllate da un crittoanalista, poiché la sostituzione di un bit nel blocco c_i induce un errore nella decifrazione dell'intero blocco m_i (attraverso la funzione $\mathcal{D}(c_i, k)$) e del corrispondente bit nel blocco m_{i+1} (attraverso lo `XOR`), ma qui la propagazione dell'errore si arresta. Il metodo è però completamente esposto a errori di inserimento o cancellazione di bit nel crittogramma e, per essere protetto da questi, deve essere combinato con altri meccanismi.

Nonostante i vantaggi offerti il *CBC* è comunque strutturato a blocchi e conserva un funzionamento sincopato. Sono stati proposti vari metodi per superare questo punto, che permettono di costruire un cifrario *a flusso* (tipo *One-Time Pad*) a partire da un cifrario *a blocchi* (tipo *DES*). I più noti sono il *Cipher Feedback* (CFB) e l'*Output Feedback* (OFB), nei quali il messaggio in chiaro è posto in `XOR` con l'uscita di un generatore di bit pseudo-casuali (paragrafo 4.2). Rimandiamo alla letteratura specialistica per l'approfondimento di questi temi.

Capitolo 8

Crittografia a chiave pubblica

Il 1976 fu un anno importante nella storia della crittografia non solo perché vide nascere il *DES* nella sua configurazione standard, ma perché in quell'anno Diffie e Hellman, e indipendentemente Merkle, introdussero un nuovo concetto che avrebbe rivoluzionato il modo di concepire le comunicazioni segrete: i *cifrari a chiave pubblica*. Inizialmente la costruzione di protocolli basati su tale principio non suscitò l'attenzione che avrebbe meritato perché l'ambiente era poco incline alle innovazioni a causa di una cultura crittografia ancora non molto diffusa, e forse anche per la resistenza di molte aziende per le quali il *DES* e la produzione dei relativi circuiti era motivo di consistenti affari. Ma l'idea si fece poi strada per le sue possibilità radicalmente nuove: vediamo di comprendere i motivi del cambiamento.

Accettato il criterio generale che le funzioni di cifratura e decifrazione siano pubbliche, uguali per tutti gli utenti e parametriche nelle chiavi, la novità riguarda la gestione delle chiavi stesse. Nei cifrari *simmetrici* visti sinora la chiave di cifratura è uguale a quella di decifrazione (o comunque ciascuna può essere facilmente calcolata dall'altra), ed è nota solo ai due partner che la scelgono di comune accordo e la mantengono segreta. Nei cifrari a chiave pubblica, o *asimmetrici*, le chiavi di cifratura e di decifrazione sono completamente diverse tra loro. Esse sono scelte dal destinatario che rende pubblica la chiave di cifratura $k[pub]$, che è quindi *nota a tutti*, e mantiene segreta la chiave di decifrazione $k[prv]$ che è quindi *nota soltanto a lui*.

In sostanza esiste una coppia $\langle k[pub], k[prv] \rangle$ per ogni utente del sistema, scelta da questi nella sua veste di possibile destinatario `Dest`. La cifratura di un messaggio m da inviare a `Dest` è eseguita da qualunque mittente come $c = \mathcal{C}(m, k[pub])$, ove sia la chiave $k[pub]$ che la funzione di cifratura $\mathcal{C}$ sono note a tutti. La decifrazione è eseguita da `Dest` come $m = \mathcal{D}(c, k[prv])$, ove $\mathcal{D}$ è la funzione di decifrazione anch'essa nota a tutti, ma $k[prv]$ non è disponibile agli altri che non possono quindi ricostruire m. L'appellativo di asimmetrici assegnato a questi cifrari sottolinea i ruoli completamente

diversi svolti da `Mitt` e `Dest`, in contrapposizione ai ruoli intercambiabili che essi hanno nei cifrari simmetrici ove condividono la stessa informazione (cioè la chiave) segreta. Per funzionare correttamente, il processo di cifratura e decifrazione deve soddisfare alcune proprietà:

1. Per ogni possibile messaggio m si ha: $\mathcal{D}(\mathcal{C}(m, k[pub]), k[prv]) = m$. Ossia `Dest` deve avere la possibilità di interpretare qualunque messaggio che gli altri utenti decidano di spedirgli.

2. La sicurezza e l'efficienza del sistema dipendono dalle funzioni $\mathcal{C}$ e $\mathcal{D}$, e dalla relazione che esiste tra le chiavi $k[prv]$ e $k[pub]$ di ogni coppia. Più esattamente:

   (a) la coppia $\langle k[prv], k[pub]\rangle$ è *facile* da generare, e deve risultare praticamente impossibile che due utenti scelgano la stessa chiave;

   (b) dati m e $k[pub]$, è *facile* per il mittente calcolare il crittogramma $c = \mathcal{C}(m, k[pub])$;

   (c) dati c e $k[prv]$, è *facile* per il destinatario calcolare il messaggio originale $m = \mathcal{D}(c, k[prv])$;

   (d) pur conoscendo il crittogramma c, la chiave pubblica $k[pub]$, e le funzioni $\mathcal{C}$ e $\mathcal{D}$, è *difficile* per un crittoanalista risalire al messaggio m.

Come sempre i termini "facile" e "difficile" devono intendersi in senso computazionale. La proprietà 1 garantisce la correttezza del processo complessivo di cifratura e decifrazione. La proprietà 2a esclude che due utenti possano avere le stesse chiavi imponendo in pratica che la generazione delle chiavi sia casuale. Le proprietà 2b e 2c assicurano che il processo possa essere di fatto adottato. La proprietà 2d garantisce la sicurezza del cifrario.

Da tutto questo deriva che $\mathcal{C}$ deve essere una funzione *one-way* (paragrafo 4.2), cioè facile da calcolare e difficile da invertire, ma deve contenere un meccanismo segreto detto *trap-door* che ne consenta la facile invertibilità solo a chi conosca tale meccanismo. La conoscenza di $k[pub]$ non fornisce alcuna indicazione sul meccanismo segreto, che è svelato da $k[prv]$ quando questa chiave è inserita nella funzione $\mathcal{D}$. Naturalmente le funzioni one-way trap-door potrebbero non esistere del tutto, e senza di esse non esisterebbero i cifrari a chiave pubblica. Vedremo che non è così, ma per approfondire l'argomento dobbiamo prima ripassare alcuni concetti elementari di algebra modulare su cui si fondano tutti i meccanismi crittografici che tratteremo da qui in poi. Chi conoscesse già questi argomenti potrà limitarsi a scorrere velocemente il prossimo paragrafo per controllare la notazione usata: in ogni caso citeremo sempre

nel seguito i risultati algebrici necessari inserendo riferimenti puntuali, sicché il lettore potrà anche saltare direttamente al paragrafo 8.2 salvo ritornare sul precedente quando necessario.

8.1 Alcuni richiami di algebra modulare

Molti algoritmi crittografici interessanti utilizzano l'algebra modulare essenzialmente per due motivi. Il primo è ridurre lo spazio dei numeri su cui gli algoritmi sono chiamati a operare e quindi aumentare la velocità di calcolo; il secondo, anche più importante, è rendere "difficili" alcuni problemi computazionali che sono semplici (o addirittura banali) nell'algebra non modulare, con lo scopo di costruire difese contro la crittoanalisi. Ricordiamo dunque alcuni concetti algebrici di base.

Preso un numero intero positivo n indichiamo con $\mathcal{Z}_n = \{0, 1, \ldots, n-1\}$ l'insieme di tutti gli interi non negativi minori di n, e con $\mathcal{Z}_n^*$ l'insieme di tutti gli elementi di $\mathcal{Z}_n$ relativamente primi con n (quindi 0 è escluso e 1 è incluso). Il problema di determinare $\mathcal{Z}_n^*$ dato n è in genere computazionalmente difficile poiché richiede tempo proporzionale al *valore* di n (per esempio per confrontare con n tutti gli elementi di $\mathcal{Z}_n$), quindi esponenziale nella sua *dimensione*, cioè nel numero di bit con cui si rappresenta n. Se però n è un numero primo si ha direttamente $\mathcal{Z}_n^* = \{1, \ldots, n-1\}$.

Dati due interi $a \geq 0$ e $n > 0$, ricordiamo che $a \texttt{ mod } n$ indica il resto della divisione intera tra a e n: per esempio $5 \texttt{ mod } 3 = 2$. Dati tre interi $a, b \geq 0$ e $n > 0$, si ha $a \equiv b \texttt{ mod } n$ (e si legge "*a congruo* a *b modulo n*") se esiste un intero k per cui $a = b + kn$, o equivalentemente se $a \texttt{ mod } n = b \texttt{ mod } n$: per esempio $5 \equiv 8 \texttt{ mod } 3$. Si noti che nelle relazioni di congruenza la notazione $\texttt{mod } n$ si riferisce all'intera relazione, mentre nelle relazioni di uguaglianza la stessa notazione si riferisce solo al membro ove appare (non si può quindi scrivere $5 = 8 \texttt{ mod } 3$, ma $2 = 8 \texttt{ mod } 3$).

Le seguenti proprietà sono immediate e saranno impiegate nel seguito senza esplicito riferimento:

- $(a + b) \texttt{ mod } n = ((a \texttt{ mod } n) + (b \texttt{ mod } n)) \texttt{ mod } n$.
- $(a - b) \texttt{ mod } n = ((a \texttt{ mod } n) - (b \texttt{ mod } n)) \texttt{ mod } n$.
- $(a \times b) \texttt{ mod } n = ((a \texttt{ mod } n) \times (b \texttt{ mod } n)) \texttt{ mod } n$.
- $a^{r \times s} \texttt{ mod } n = ((a^r \texttt{ mod } n)^s) \texttt{ mod } n$, con r e s interi positivi qualunque.

Per un intero $n > 1$ si definisce la *funzione di Eulero* $\Phi(n)$ come il numero di interi minori di n e relativamente primi con esso. Quindi $\Phi(n) = |\mathcal{Z}_n^*|$, e in particolare

$\Phi(n) = n - 1$ se n è primo. È noto un procedimento di calcolo per $\Phi(n)$ se si conosce la scomposizione di n in fattori primi. In particolare vale:

Teorema 8.1. *Se* n *è il prodotto di due numeri primi* p,q *si ha* $\Phi(n) = (p-1)(q-1)$.

Vale inoltre:

Teorema 8.2 (Eulero). *Per* n > 1 *e per ogni intero* a *primo con* n *si ha* $a^{\Phi(n)} \equiv 1 \texttt{ mod } n$.

Per n primo si ha dunque $a^{n-1} \equiv 1 \texttt{ mod } n$ per ogni $a \in \mathcal{Z}_n^*$ (questo risultato è noto come *piccolo teorema di Fermat*). Inoltre per qualunque a primo con n si ha $a \times a^{\Phi(n)-1} \equiv 1 \texttt{ mod } n$ per il teorema di Eulero, e $a \times a^{-1} \equiv 1 \texttt{ mod } n$ per la definizione di inverso, quindi $a^{\Phi(n)-1} \texttt{ mod } n$ coincide con l'inverso a^{-1} di a modulo n: ciò implica che, per a primo con n, tale inverso si può calcolare per esponenziazione di a se si conosce $\Phi(n)$.

Possiamo ora studiare l'equazione $ax \equiv b \texttt{ mod } n$ con coefficienti a, b, n interi. Nell'algebra non modulare l'equazione si risolve semplicemente come $x = a^{-1} \times b$, calcolando l'inverso di a e moltiplicandolo per b. Nell'algebra modulare l'esistenza dell'inverso non è garantita perché a^{-1} deve essere intero, e l'equazione può ammettere una o più soluzioni, o può non ammetterne affatto. Tutto ciò è regolato dal seguente teorema:

Teorema 8.3. *L'equazione* $a\,x \equiv b \texttt{ mod } n$ *ammette soluzione se e solo se il massimo comun divisore* $mcd(a, n)$ *divide* b. *In questo caso si hanno esattamente* $mcd(a, n)$ *soluzioni distinte.*

Dunque l'equazione considerata ammette esattamente una soluzione se e solo se $mcd(a, n) = 1$, cioè a e n sono primi tra loro, caso in cui è garantita l'esistenza di a^{-1}. In particolare ponendo $b = 1$ nell'equazione questa diviene $a\,x \equiv 1 \texttt{ mod } n$ e ammette esattamente una soluzione (l'inverso di a) se e solo se a e n sono primi tra loro. Abbiamo visto che tale inverso può essere calcolato come $a^{\Phi(n)-1} \texttt{ mod } n$, ma ciò richiede di conoscere $\Phi(n)$ cioè di fattorizzare n, e questo come vedremo è un problema computazionalmente difficile.

Esiste un altro procedimento per il calcolo dell'inverso, basato su un'estensione dell'algoritmo di Euclide per il calcolo del massimo comun divisore (paragrafo 3.1). Tale algoritmo può essere infatti generalizzato per risolvere l'equazione in due incognite $ax + by = mcd(a, b)$ secondo lo schema seguente (si veda anche il problema delle equazioni diofantee lineari, paragrafo3.4):

```
Function Extended_Euclid(a, b):
if b = 0
    then return ⟨a, 1, 0⟩
    else ⟨d', x', y'⟩ ← Extended_Euclid(b, a mod b);
         ⟨d, x, y⟩ ← ⟨d', y', x' − ⌊a/b⌋y'⟩;
         return ⟨d, x, y⟩.
```

Si può facilmente dimostrare che questa funzione restituisce una delle triple di valori $\langle mcd(a,b), x, y\rangle$, con x, y tali che $ax + by = mcd(a,b)$. Come per l'algoritmo di Euclide, la complessità in tempo di `Extended_Euclid`(a,b) è logaritmica nel *valore* dei parametri d'ingresso a, b, quindi polinomiale nella *dimensione* dei parametri stessi. Questo, come ora vedremo, consente di calcolare direttamente e efficientemente l'inverso $x = a^{-1}$ `mod` b di un intero arbitrario a primo con b. Infatti la relazione $ax \equiv 1$ `mod` b è equivalente alla $ax = bz + 1$ per un opportuno valore di z; da cui, ponendo $y = -z$ e ricordando che $mcd(a,b) = 1$ si ottiene proprio l'equazione $ax + by = mcd(a,b)$, e dalla soluzione di questa il valore dell'inverso x.

Esistono particolari interi n, e in corrispondenza a essi particolari interi $a \in \mathcal{Z}_n^*$, per cui la funzione a^k `mod` n, con $1 \leq k \leq \Phi(n)$, genera tutti e soli gli elementi di $\mathcal{Z}_n^*$. In questo caso a è detto *generatore* di $\mathcal{Z}_n^*$. Si ricordi che per il teorema di Eulero $a^{\Phi(n)}$ `mod` $n = 1$, quindi l'elemento 1 (sempre presente in $\mathcal{Z}_n^*$) viene generato per $k = \Phi(n)$ e deve risultare per ogni generatore: $a^k \not\equiv 1$ `mod` n per ogni $1 \leq k < \Phi(n)$. Per esempio 3 è un generatore di $\mathcal{Z}_7^*$ poiché si ha $\Phi(7) = 6$ e per $k = 1, ..., 6$ vengono generati nell'ordine i valori $3^k \, mod \, 7 = 3, 2, 6, 4, 5, 1$. Invece 2 non è un generatore di $\mathcal{Z}_7^*$ poiché si ha 2^3 `mod` $7 = 2^6$ `mod` $7 = 1$.

L'esistenza delle coppie $\langle n, a\rangle$ è regolata da opportuni teoremi: noi ci occuperemo in particolare del caso, importante in crittografia, in cui n è un numero primo, quindi $\mathcal{Z}_n^* = \{1, \ldots, n-1\}$ e $\Phi(n) = n - 1$.

Teorema 8.4. *Per ogni* n *primo,* $\mathcal{Z}_n^*$ *ammette (almeno) un generatore.*

Si noti che per n primo non tutti gli elementi di $\mathcal{Z}_n^*$ sono suoi generatori; l'elemento 1 non è mai un generatore perché le sue potenze sono tutte uguali a 1, e altri elementi possono non esserlo. In effetti per n primo si può dimostrare che i generatori di $\mathcal{Z}_n^*$ sono in totale $\Phi(n-1)$. Ad esempio, $\mathcal{Z}_7^*$ ha $\Phi(6) = 2$ generatori che sono 3 e 5.

Due problemi computazionali legati ai generatori sono particolarmente rilevanti in crittografia. Il primo è quello della *determinazione di un generatore* a di $\mathcal{Z}_n^*$ per un primo n dato. Si possono provare per a tutti gli interi compresi tra 2 e $n-1$ fino a trovarne uno che, per $1 \leq k \leq n-1$, genera tutti gli elementi di $\mathcal{Z}_n^*$, ma ciò richiede tempo esponenziale nella dimensione dell'ingresso n. Il problema è ritenuto

computazionalmente difficile e viene in pratica risolto, con alta probabilità di successo, impiegando algoritmi randomizzati. Torneremo su questo problema a proposito dello scambio di chiavi segrete (paragrafo 8.6).

Il secondo problema, noto come *calcolo del logaritmo discreto*, riguarda la risoluzione nell'incognita x dell'equazione $a^x \equiv b \bmod n$, per n primo. Dal teorema 8.4 sappiamo che questa equazione ammette una soluzione per ogni valore di b se e solo se a è un generatore di $\mathcal{Z}_n^*$. Tuttavia, anche se questo è il caso, non è a priori noto in che ordine la successione $a^x \bmod n$, con $x = 1, 2, ..., n-1$, genera gli elementi di $\mathcal{Z}_n^*$ (salvo per $a^{n-1} \bmod n = 1$), quindi non è noto per quale valore di x si genera $b \bmod n$. Anche in questo caso un esame diretto della successione richiede tempo esponenziale nella dimensione di n, e non è noto un algoritmo polinomiale di soluzione. Torneremo su questo problema nel prossimo paragrafo.

Infine la sequenza $a^k \bmod n$ prodotta da un generatore a, per $k = 1, \ldots, \Phi(n)$, si ripete identica per $k = \Phi(n) + 1, \Phi(n) + 2, \ldots$. Per esempio per $n = 7$ il generatore 3 produce la sequenza 3, 2, 6, 4, 5, 1 ripetuta all'infinito.

8.2 Le funzioni one-way trap-door

Potrà apparire sorprendente ma i meccanismi di tipo one-way trap-door di cui abbiamo parlato all'inizio del capitolo si incontrano con frequenza in ogni campo. Il più ovvio può essere messo in relazione alla protezione "domestica" delle comunicazioni: i messaggi sono comuni lettere; la chiave pubblica è l'indirizzo del destinatario; il metodo pubblico di cifratura consiste nell'imbucare una lettera nella cassetta postale del destinatario, accessibile a tutti; il metodo privato di decifrazione consiste nell'apertura della cassetta mediante la chiave (metallica) posseduta solo dal destinatario. Il meccanismo è one-way trap-door in quanto è facile depositare le lettere nella cassetta ma è difficile estrarle se non si possiede la chiave.

Un esempio tanto elementare lascia supporre che le funzioni one-way trap-door esistano anche in matematica. La cosa non è così semplice ma, con lo sviluppo degli studi crittografici, sono state individuate alcune funzioni che possiedono i requisiti richiesti utilizzando proprietà della teoria dei numeri e dell'algebra modulare. Il calcolo diretto di queste funzioni è infatti incondizionatamente semplice e la loro inversione è semplice solo se si dispone di una informazione aggiuntiva sui dati, cioè di una chiave privata. Senza questa informazione l'inversione richiede la soluzione di un problema *NP*-arduo (capitolo 3), o comunque di un problema per cui non si conosce un algoritmo polinomiale: tale inversione rimarrà quindi computazionalmente difficile fino all'improbabile momento in cui si trovi una soluzione efficiente a questi problemi. Consideriamo ora tre di queste funzioni particolarmente rilevanti in crittografia.

1. *Fattorizzazione.* Calcolare il prodotto n di due interi p e q è polinomialmente facile poiché richiede tempo quadratico nella lunghezza della loro rappresentazione. Invertire la funzione significa ricostruire p e q a partire da n, il che è univocamente possibile solo se p e q sono primi. Tale inversione richiede tempo esponenziale per quanto è noto fino a oggi, anche se non vi è dimostrazione che il problema sia *NP*-arduo (quindi l'esistenza di un algoritmo polinomiale di soluzione è assai improbabile ma non assolutamente da escludere). Se però si conosce uno dei fattori (la chiave segreta) ricostruire l'altro è ovviamente facile.

2. *Calcolo della radice in modulo.* Come abbiamo visto nel paragrafo 4.3, calcolare la potenza $y = x^z \bmod s$, con x, z, s interi, richiede tempo polinomiale se si procede per successive esponenziazioni. Il metodo richiede infatti di eseguire $\Theta(\log_2 z)$ moltiplicazioni, cioè un numero lineare nella lunghezza della rappresentazione di z, il che conduce a un algoritmo complessivamente cubico.

   Se s non è primo e se ne ignora la fattorizzazione, invertire la funzione, cioè calcolare $x = \sqrt[z]{y} \bmod s$ se sono noti y, z, s, richiede tempo esponenziale per quanto noto fino a oggi (lo status computazionale di questo problema è simile a quello della fattorizzazione citato al punto precedente). Se però x è primo con s e si conosce l'inverso v di z modulo $\Phi(s)$, cioè $z\,v \equiv 1 \bmod \Phi(s)$, si ha: $y^v \bmod s = x^{zv} \bmod s = x^{1+k\Phi(s)} \bmod s = x \bmod s$, in cui l'ultima eguaglianza è basata sul teorema di Eulero. Si ricostruisce quindi in tempo polinomiale x calcolando $y^v \bmod s$: in questo caso v è la chiave segreta per invertire la funzione.

3. *Calcolo del logaritmo discreto.* La funzione potenza del punto precedente si può invertire anche rispetto a z: dati cioè x, y, s interi si richiede di trovare il valore di z tale che $y = x^z \bmod s$. Nell'algebra non modulare il problema, di semplice soluzione, è quello del calcolo del logaritmo di y in base x; ma operando in modulo il problema è computazionalmente difficile e non sempre ha soluzione. Da quanto abbiamo visto nel paragrafo precedente (teorema 8.4 e sue conseguenze), se s è primo esiste una soluzione per ogni y se e solo se x è un generatore di $\mathcal{Z}_s^*$: nel seguito ci limiteremo a considerare questo caso. A parte alcuni particolari primi s per cui il calcolo si semplifica (per esempio se tutti i fattori primi di $s-1$ sono piccoli), gli algoritmi noti fino a oggi hanno la stessa complessità della fattorizzazione anche se non è stato dimostrato che i due problemi siano computazionalmente equivalenti. Ciò dimostra che la funzione logaritmo discreto è one-way; è possibile dimostrare con un artificio piuttosto complesso come introdurre una trap-door in essa.

A questo punto le principali idee su cui si basano i cifrari a chiave pubblica

dovrebbero essere sufficientemente chiare. È però opportuno fare alcune ulteriori considerazioni prima di procedere alla descrizione del più importante di tali cifrari.

8.3 Pregi e difetti del nuovo metodo

Rispetto all'impiego di cifrari simmetrici, la crittografia a chiave pubblica presenta due vantaggi immediati e strettamente connessi: se gli utenti di un sistema sono n, il numero complessivo di chiavi (pubbliche e private) è $2n$ anziché $n(n-1)/2$; inoltre non è richiesto alcuno scambio segreto di chiavi tra gli utenti. Tali vantaggi sono molto rilevanti perché la globalizzazione delle comunicazioni, e quindi le grandi dimensioni dei sistemi in uso, rende praticamente impossibile gestire lo scambio segreto di un numero quadratico di chiavi. Ma accanto a questi vantaggi i cifrari a chiave pubblica comportano alcuni svantaggi altrettanto immediati:

- Il sistema è esposto ad attacchi del tipo *chosen plain-text* in modo del tutto ovvio. Un crittoanalista può scegliere un numero qualsiasi di messaggi in chiaro $m_1, m_2, \ldots, m_h$ e cifrarli utilizzando la funzione pubblica $\mathcal{C}$ e la chiave pubblica $k[pub]$ di un destinatario `Dest`, ottenendo così i crittogrammi $c_1, c_2, \ldots, c_h$. A questo punto, spiando sul canale di comunicazione, egli può confrontare qualsiasi messaggio cifrato c^* in viaggio verso `Dest` con i crittogrammi di cui è in possesso: se c^* coincide con uno di essi il messaggio è automaticamente decifrato; se invece $c^* \neq c_i$, per ogni i, il crittoanalista ha comunque acquisito una informazione importante, cioè che il messaggio è diverso da quelli che lui ha scelto. L'attacco è particolarmente pericoloso se il crittoanalista sospetta che `Dest` debba ricevere un messaggio particolare ed è in attesa di vedere quando questo accada; oppure se c^* rappresenta un messaggio breve e di struttura prevedibile, come per esempio un indirizzo Internet o una *password* scelta ingenuamente.

- Questi sistemi sono molto più lenti di quelli basati su cifrari simmetrici: stime (non molto aggiornate) indicano che il rapporto tra le loro velocità sia da due a tre ordini di grandezza e che, in questo senso, le realizzazioni hardware siano meno competitive di quelle software. Si potrebbe obiettare che questo è un problema secondario a causa della continua crescita di velocità dei calcolatori, ma l'effetto è invece sensibile per la richiesta sempre crescente di comunicazioni sicure.

Poiché i cifrari simmetrici e asimmetrici presentano pregi e difetti complementari, nei protocolli crittografici esistenti si tende ad adottare un approccio ibrido che combina i pregi ed esclude i difetti delle due famiglie. Come vedremo in seguito si usa un

cifrario a chiave segreta come il *DES Triplo* o l'*AES* per le comunicazioni di massa, e un cifrario a chiave pubblica per scambiare le chiavi segrete relative al primo senza un incontro fisico tra gli utenti. In questo modo la trasmissione dei messaggi lunghi avviene ad alta velocità mentre è lento lo scambio delle chiavi segrete che però sono assai più brevi, il che tra l'altro permette agli utenti di cambiarle frequentemente per incrementare la sicurezza delle comunicazioni. Inoltre l'attacco *chosen plain-text* di cui soffrono i cifrari asimmetrici è ovviato se l'informazione cifrata con la chiave pubblica (cioè la chiave segreta del *DES* o dell'*AES*) è scelta in modo da risultare imprevedibile al crittonalista.

Alla definizione del nuovo protocollo a chiave pubblica seguirono in breve tempo due realizzazioni. Il primo cifrario, proposto da Merkle, basava la difficoltà di inversione della funzione $\mathcal{C}$ sulla risoluzione del *problema dello zaino* (paragrafo 3.3). Benché tale problema sia *NP*-arduo il cifrario è stato violato per altra via, mostrando ancora una volta con quanta cautela vada affrontato il problema della sicurezza (successivi cifrari basati sullo stesso problema sono invece rimasti inviolati).

Il secondo cifrario, proposto da Rivest, Shamir e Adleman e noto come *RSA*, fonda la sua sicurezza sulla difficoltà di fattorizzare grandi numeri interi (vedi paragrafo precedente). Benché tale problema non sia dimostratamente *NP*-arduo, e quindi potrebbe essere "più semplice" del problema dello zaino, *RSA* è sostanzialmente inviolabile per chiavi sufficientemente lunghe, ed è il cifrario asimmetrico di più largo impiego. Studieremo quindi con attenzione questo cifrario rimandando a un prossimo paragrafo la discussione su una "nuova frontiera" nel campo della chiave pubblica, varcata in modo ormai massiccio dalla famiglia dei *cifrari su curve ellittiche*.

8.4 Il cifrario *RSA*

Nel 1978 fu presentato il cifrario a chiave pubblica *RSA*, così detto dalle iniziali degli autori. La sua fortuna è stata enorme: accanto a una grande semplicità strutturale che ne ha permesso l'impiego in numerosissimi sistemi hardware e software, il cifrario si è dimostrato fino a oggi sostanzialmente inviolabile. Vediamone anzitutto l'organizzazione generale, basata su alcuni risultati di algebra modulare presentati nel paragrafo 8.1.

Creazione della chiave. Come possibile destinatario, ogni utente `Dest` esegue le seguenti operazioni:

- sceglie due numeri primi p, q molto grandi (paragrafo 4.3);
- calcola $n = p \times q$, e la funzione di Eulero $\Phi(n) = (p-1)(q-1)$;

- sceglie un intero e minore di $\Phi(n)$ e primo con esso;
- calcola l'intero d inverso di e modulo $\Phi(n)$ (l'esistenza e unicità di d è assicurata dal fatto che e e $\Phi(n)$ sono primi tra loro, e il calcolo si esegue efficientemente mediante la funzione `Extended_Euclid`: vedi paragrafo 8.1);
- rende pubblica la chiave $k[pub] = \langle e, n \rangle$, e mantiene segreta la chiave $k[prv] = \langle d \rangle$.

Messaggio. Ogni messaggio è codificato come sequenza binaria, che viene trattata come un numero intero m. Per impiegare il cifrario deve risultare $m < n$, il che è sempre possibile dividendo il messaggio in blocchi di al più $\lfloor \log_2 n \rfloor$ bit. Si noti che la dimensione massima del blocco dipende dalla chiave pubblica del destinatario: si stabilisce in pratica un limite comune per impiegare blocchi delle stesse dimensioni per tutti i destinatari.

Cifratura. Per inviare a `Dest` un messaggio m, un utente arbitrario genera il crittogramma c calcolando la funzione $c = \mathcal{C}(m, k[pub]) = m^e \texttt{ mod } n$, ove i valori e, n sono contenuti nella chiave pubblica $k[pub]$ di `Dest`. Ovviamente risulta $c < n$.

Decifrazione. `Dest` riceve il crittogramma c e lo decifra calcolando $m = \mathcal{D}(c, k[prv]) = c^d \texttt{ mod } n$, ove il valore di d è contenuto nella sua chiave privata $k[prv]$.

Illustriamo queste operazioni mediante un esempio che richiede calcoli limitatissimi, prima ancora di dimostrarne la legittimità. Scegliamo i numeri primi $p = 5$ e $q = 11$ (evidentemente non sono *molto grandi*); calcoliamo $n = 55$ e $\Phi(n) = 40$; scegliamo $e = 7$, primo con 40; determiniamo l'inverso $d = 23$, ove $23 \times 7 \equiv 1 \texttt{ mod } 40$; pubblichiamo $k[pub] = \langle 7, 55 \rangle$ e manteniamo segreta $k[prv] = \langle 23 \rangle$. Chiunque voglia spedirci un messaggio $m < 55$ dovrà calcolare $c = m^7 \texttt{ mod } 55$ e recapitare c. Per decifrare c noi dovremo calcolare $m = c^{23} \texttt{ mod } 55$.

Per convincerci della correttezza del cifrario dovremmo anzitutto dimostrare che la scelta di un qualsiasi intero e relativamente primo con $\Phi(n)$ implica che la funzione $x^e \texttt{ mod } n$ sia una permutazione di $\mathcal{Z}_n$. Ciò garantisce l'invertibilità del processo di cifratura, poiché per qualsiasi coppia di messaggi $m_1 \neq m_2$ risulta $\mathcal{C}(m_1, k[pub]) \neq \mathcal{C}(m_2, k[pub])$, ma non é sufficiente al nostro scopo perché non è ovvio che decifrando un crittogramma si ricostruisca proprio il messaggio che è stato spedito, cioè che $\mathcal{D}(\mathcal{C}(m, k[pub]), k[prv]) = m$, ovvero $(m^e \texttt{ mod } n)^d \texttt{ mod } n = m$ (infatti d è l'inverso di e modulo $\Phi(n)$ mentre nel calcolo indicato intervengono operazioni modulo n). Dimostriamo dunque che questa relazione è valida, cioè che complessivamente il cifrario è corretto.

Teorema 8.5. *Per qualunque intero* m<n *si ha:* $(m^e \texttt{ mod } n)^d \texttt{ mod } n = m$*, ove* n, e, d *sono i parametri del cifrario* RSA.

Per una proprietà del calcolo modulare (paragrafo 8.1) la relazione da dimostrare può essere riscritta come $m^{ed} \texttt{ mod } n = m$. Per la dimostrazione distinguiamo due casi, relativi ai valori di p e q scelti dal destinatario e al valore di m scelto dal mittente:

p e q non dividono m. Abbiamo $mcd(m, n) = 1$, quindi per il teorema di Eulero (paragrafo 8.1) risulta $m^{\Phi(n)} \equiv 1 \texttt{ mod } n$. Poiché d è l'inverso di e in $\mathcal{Z}^*_{\Phi(n)}$, abbiamo $e \times d \equiv 1 \texttt{ mod } \Phi(n)$, ovvero $e \times d = 1 + r\Phi(n)$, con r intero positivo opportuno. Otteniamo quindi: $m^{ed} \texttt{ mod } n = m^{1+r\Phi(n)} \texttt{ mod } n = m \times (m^{\Phi(n)})^r \texttt{ mod } n = m \times 1^r \texttt{ mod } n = m$.

p (oppure q) divide m, ma q (oppure p) non divide m. Poiché p divide m abbiamo $m \equiv m^r \equiv 0 \texttt{ mod } p$, ovvero $(m^r - m) \equiv 0 \texttt{ mod } p$, per qualunque intero positivo r. Con un procedimento analogo a quello del punto precedente abbiamo anche: $m^{ed} \texttt{ mod } q = m^{1+r\Phi(n)} \texttt{ mod } q = m \times m^{r(p-1)(q-1)} \texttt{ mod } q = m \times (m^{(q-1)})^{r(p-1)} \texttt{ mod } q = m \texttt{ mod } q$, ove l'ultima uguaglianza è dovuta alla relazione $m^{(q-1)} \equiv 1 \texttt{ mod } q$ per il teorema di Eulero. Dunque $m^{ed} \equiv m \texttt{ mod } q$, e quindi $(m^{ed} - m) \equiv 0 \texttt{ mod } q$. Ne consegue che $m^{ed} - m$ è divisibile sia per p che per q, quindi è divisibile per il loro prodotto $p \times q = n$; ovvero $(m^{ed} - m) \equiv 0 \texttt{ mod } n$ da cui deriva immediatamente la tesi.

Si noti che p e q non possono dividere entrambi m perché si avrebbe $m \geq n$ contro l'ipotesi sulla dimensione dei blocchi. La correttezza del cifrario è così dimostrata. Vi sono però diverse considerazioni da fare sulla complessità delle operazioni che esso richiede.

Esaminiamo anzitutto la generazione della chiave. I numeri primi p e q devono essere molto grandi: con i calcolatori di oggi è necessario che contengano almeno un migliaio di cifre binarie per garantire che la fattorizzazione di n sia praticamente inattuabile, e che la probabilità che due utenti scelgano la stessa chiave segreta sia praticamente trascurabile. La generazione di p e q può avvenire servendosi dell'algoritmo di Miller-Rabin, e dunque richiede tempo polinomiale nella dimensione di questi numeri (paragrafo 4.3). I valori di n e $\Phi(n)$ si calcolano come prodotti, quindi in tempo polinomiale nelle loro dimensioni. L'intero d, inverso di e modulo $\Phi(n)$, può essere anch'esso generato in tempo polinomiale utilizzando come già detto la funzione `Extended_Euclid`. In conclusione le chiavi possono essere generate in modo efficiente.

I processi di cifratura e decifrazione sono altrettanto efficienti se si calcolano le potenze in modulo per successive esponenziazioni (paragrafo 4.3). Tale schema di calcolo suggerisce anche quali valori di e sarebbe auspicabile scegliere per facilitare

il processo di cifratura, sempre che essi siano primi con $\Phi(n)$: si noti che non vi è alcuna controindicazione al fatto che due utenti scelgano lo stesso valore di e purché ovviamente scelgano diversi valori di n quindi complessivamente diverse chiavi pubbliche. Sulla scelta del valore di e ritorneremo comunque nel seguito trattando dei possibili attacchi a RSA.

Facciamo infine qualche considerazione sulla decomposizione del messaggio in blocchi. Abbiamo richiesto che ogni blocco m, interpretato come numero intero, sia minore di n: se infatti fosse $m \geq n$, i due messaggi m e (m `mod` n) genererebbero lo stesso crittogramma. Poiché però il valore di n è relativo a un singolo destinatario, si deve stabilire uno standard per la decomposizione in blocchi nell'intero sistema. La via più semplice è quella, già indicata, di adottare una dimensione dei blocchi uguale per tutti, stabilendo implicitamente un limite inferiore al valore di n che gli utenti possono scegliere. In alternativa si può decomporre il messaggio in blocchi di $\ell = \lfloor \log_2 n \rfloor$ bit, ove il valore di n e quindi di ℓ è relativo al destinatario del messaggio. Notando che n non è una potenza di 2, questo meccanismo garantisce che sia $m \leq 2^\ell < n$: dopo aver decifrato il crittogramma come un numero intero m, il destinatario deve infatti rappresentare questo numero utilizzando esattamente ℓ bit.

8.5 Attacchi all'*RSA*

Come già osservato tutti i cifrari a chiave pubblica sono soggetti a immediati attacchi di tipo *chosen plain-text*, sostanzialmente inoffensivi solo se il crittoanalista non è in grado di prevedere il contenuto dei messaggi che vengono scambiati (paragrafo 8.3). Torneremo nel seguito sull'importanza di questa ipotesi esaminando ora la difficoltà computazionale di attacchi legati alle proprietà aritmetiche del cifrario *RSA* e i conseguenti accorgimenti di difesa nella scelta dei parametri.

Anzitutto la sicurezza del cifrario è strettamente legata alla difficoltà di fattorizzare un numero intero arbitrario molto grande (paragrafo 8.2). Infatti la ricostruzione degli interi p, q, il cui prodotto costituisce il valore n dichiarato in una chiave pubblica, permette di calcolare immediatamente $\Phi(n)$ e quindi la chiave privata di chi ha pubblicato n. In sostanza la facoltà di fattorizzare efficientemente un intero implica quella di forzare efficientemente il cifrario ma non è nota l'implicazione inversa, cioè non si sa se forzare efficientemente il cifrario implichi fattorizzare efficientemente un intero. Infatti nonostante numerosi studi non è ancora noto se per forzare l'*RSA* si debba crucialmente passare attraverso la fattorizzazione di n anche se ciò è ritenuto molto plausibile. Esistono bensì alcune varianti dell'*RSA* la cui sicurezza è dimostratamente equivalente al problema della fattorizzazione, ma tali varianti sono più complesse del sistema originale e non hanno avuto lo stesso successo di questo.

In effetti per decifrare un crittogramma c sarebbe sufficiente calcolare la sua radice e-esima modulo n, perché tutti conoscono $k[pub] = \langle e, n\rangle$ e sanno che $c = m^e \bmod n$. Sappiamo però (paragrafo 8.2) che il calcolo della radice e-esima nell'algebra modulare può essere eseguito efficientemente se il modulo n è primo, ma è ritenuto difficile quanto la fattorizzazione se n è composto. Ne segue che il calcolo di m a partire da e, n, c non è più facile del calcolo dei fattori primi di n: tanto vale quindi calcolare questi ultimi, anche perché ciò implica la forzatura totale del sistema e non la sola decifrazione di un particolare crittogramma.

Un metodo alternativo di forzatura potrebbe basarsi sulla scoperta di $\Phi(n)$ senza passare attraverso i valori p, q, poiché conoscendo $\Phi(n)$ si calcola immediatamente la chiave segreta. Ma sorprendentemente ciò non cambia le cose. Possiamo infatti scrivere: $\Phi(n) = (p-1)(q-1) = n - (p+q) + 1$ e, conoscendo $\Phi(n)$, possiamo quindi calcolare $x_1 = (p+q)$. Osservando poi che vale l'uguaglianza: $(p-q)^2 = (p+q)^2 - 4n$, possiamo calcolare $x_2 = (p-q)$, da cui infine si ha: $p = (x_1 + x_2)/2$, $q = (x_1 - x_2)/2$. Ciò vuol dire che il calcolo di $\Phi(n)$ e la fattorizzazione di n sono problemi computazionalmente equivalenti poiché ciascuno si trasforma nell'altro in tempo polinomiale.

Infine si potrebbe condurre un attacco esauriente sulla chiave segreta d, cioè verificare, per ogni possibile valore di d, se la decifrazione dà origine a un messaggio significativo. Il processo potrebbe però essere molto più costoso della fattorizzazione di n poiché, in dipendenza dalla scelta di e, il valore di d può essere molto più grande di quelli di p e di q (vedi oltre). Non è noto se sia possibile calcolare efficientemente d utilizzando solo la chiave pubblica $\langle e, n\rangle$ ed eventualmente il crittogramma c.

In sostanza tutti i tentativi compiuti hanno confermato la congettura che forzare il cifrario *RSA* implichi fattorizzare n. Tuttavia sono necessari alcuni accorgimenti nella scelta dei parametri p, q ed e per evitare forzature che sfruttino le proprietà di questi numeri.

Abbiamo già detto che p e q devono essere molto grandi: tipicamente la loro rappresentazione binaria deve contenere un migliaio di bit per resistere agli attacchi esaurienti dei calcolatori di oggi. Meno ovvio è che anche la differenza tra p e q deve essere grande. Infatti se il numero $|p-q|$ fosse piccolo, $(p+q)/2$ sarebbe prossimo a $\sqrt{n}$, e questa informazione consentirebbe di determinare velocemente p e q con i seguenti semplici calcoli. Anzitutto vale l'uguaglianza già ricordata $\frac{(p+q)^2}{4} - n = \frac{(p-q)^2}{4}$ che mostra come il membro sinistro sia un quadrato perfetto; e si può immediatamente dimostrare che $(p+q)/2$, cioè il valor medio tra p e q, è maggiore di $\sqrt{n}$. Dunque si possono scandire gli interi maggiori di $\sqrt{n}$ fino a trovare un intero z tale che $z^2 - n$ sia un quadrato perfetto, diciamo $z^2 - n = w^2$; supporre quindi che sia $z = (p+q)/2$, $w = (p-q)/2$, da cui inferire i due valori $p = z + w$, $q = z - w$; infine controllare

la significatività di tali valori su c. La condizione imposta sulla vicinanza tra p e q assicura che non dobbiamo allontanarci troppo da $\sqrt{n}$ prima di trovare i valori dei due parametri. In pratica questo procedimento di attacco può essere molto efficiente. Un ragionamento più complesso consente di stabilire che il cifrario può essere attaccato anche se è grande il massimo comun divisore tra $p-1$ e $q-1$, il che avviene se questi due numeri hanno fattori grandi in comune. Poiché $p-1$ e $q-1$ sono entrambi pari, la scelta migliore è quella per cui $(p-1)/2$ e $(q-1)/2$ siano primi tra loro.

Anche la scelta di e deve avvenire oculatamente. Anzitutto se k è un divisore di $\Phi(n)$ (in particolare $\Phi(n) = (p-1)(q-1)$ è divisibile per 2 e per 4), e se m, n sono primi tra loro, per $e = 1+\Phi(n)/k$ si avrebbe: $c = m^e \bmod n = m \times (m^{\Phi(n)})^{1/k} \bmod n = m$ (Teorema 8.2); quindi tale valore di e non indurrebbe alcuna trasformazione del messaggio. Ma esistono controindicazioni più sofisticate. Si può scegliere per e un valore piccolo per rendere più rapido il processo di cifratura, ma tale valore non può scendere sotto certi limiti per evitare un attacco particolare. Poniamo infatti che diversi utenti abbiano scelto lo stesso valore di e, e che almeno e fra essi ricevano lo stesso messaggio m attraverso i crittogrammi $c_1 = m^e \bmod n_1$, $c_2 = m^e \bmod n_2$, ..., $c_e = m^e \bmod n_e$, ove $m < n_i$, $1 \leq i \leq e$, per la corretta generazione di $c_1, \ldots, c_e$ mediante l'*RSA*. [1] Possiamo assumere che $n_1, n_2, \ldots, n_e$ siano primi tra loro poiché la probabilità che ciò non avvenga è trascurabile, e in caso contrario un crittoanalista potrebbe fattorizzarli calcolandone a coppie il massimo comun divisore. Per un risultato di algebra noto come *teorema cinese del resto* esiste (e si può facilmente calcolare) un unico $m' < n = n_1 \times n_2 \times \ldots \times n_e$ che soddisfa l'equazione $m' \equiv m^e \bmod n$. Poiché $m < n_i$ per ogni i, abbiamo $m^e < n$ e la congruenza può essere riscritta semplicemente come $m' = m^e$: dunque, calcolato il valore di m' attraverso il teorema suddetto, si può da questo ricavare m mediante l'estrazione della radice e-esima che in questo caso si esegue facilmente perché non interviene l'operazione in modulo. Poiché però valori piccoli di e rendono rapida la cifratura, in molte applicazioni si consiglia di scegliere $e = 3$, o comunque uguale a un numero primo molto piccolo, e si impone di aggiungere una sequenza casuale di bit diversa per ogni destinatario alla fine di ogni messaggio (*padding*) per impedire l'attacco appena descritto.

Infine è necessario che gli utenti scelgano valori di n diversi fra loro, anche se poi le chiavi potrebbero essere differenziate mediante la scelta di e (è da evitare, per esempio, che all'interno della stessa organizzazione sia assegnato a più utenti lo stesso valore di n). In caso contrario un crittoanalista potrebbe selezionare due utenti che hanno chiavi pubbliche $\langle e_1, n\rangle$ e $\langle e_2, n\rangle$ tale che $mcd(e_1, e_2) = 1$, il che è molto

[1] Questa ipotesi sembra bizzarra, ma se e è piccolo potrebbe per esempio verificarsi tra e utenti di una stessa organizzazione che ricevono messaggi uguali da una centrale e hanno scelto tutti lo stesso piccolo valore di e consigliato (vedi oltre).

probabile se e_1 e e_2 sono scelti in modo casuale. In questo caso esistono due interi r, s calcolabili direttamente con l'algoritmo `Extended_Euclid` (paragrafo 8.1) per cui $e_1r+e_2s = 1$. Poniamo che sia $r < 0$ (il caso $s < 0$ è simmetrico) e che il crittoanalista riesca a intercettare due crittogrammi c_1 e c_2 relativi allo stesso messaggio m inviato ai due utenti. Abbiamo allora $m = m^{e_1r+e_2s} = (c_1^r \times c_2^s)$ `mod` $n = ((c_1^{-1})^{-r} \times c_2^s)$ `mod` n. Poiché in pratica c_1 e n sono primi tra loro (in caso contrario risulterebbe c_1 multiplo di p o di q, cosa estremamente improbabile), il crittoanalista può calcolare c_1^{-1} `mod` n mediante l'algoritmo `Extended_Euclid`; quindi $(c_1^{-1})^{-r}$ per esponenziazione poiché $-r$ è positivo; infine anche c_2^s per esponenziazione; e può ora ricostruire m attraverso l'espressione sopra riportata.

Per quanto riguarda la possibile periodicità dei crittogrammi notiamo che l'*RSA* soffre degli stessi problemi che si presentano nei cifrari simmetrici a blocchi, discussi nel paragrafo 7.5. Anche il modo di affrontare questi problemi è lo stesso: in particolare il modo *CBC* per la composizione dei blocchi si applica indistintamente ai cifrari simmetrici e asimmetrici, quindi all'*RSA*.

8.6 Cifrari ibridi e scambio di chiavi

Poiché i cifrari asimmetrici sono esposti ad attacchi *chosen plain-text* e la loro realizzazione richiede tempi di cifratura e decifrazione molto maggiori di quelli del cifrari simmetrici, i due tipi di cifrari vengono in genere utilizzato in combinazione dando origine a *cifrari ibridi* in cui un cifrario a chiave pubblica è utilizzato per lo scambio della chiave segreta che viene impiegata nelle successive comunicazioni simmetriche. Vediamo per esempio la classica combinazione tra *RSA* e *AES*.

La chiave segreta trasmessa con l'*RSA*, denominata *chiave di sessione* e indicata con $k[session]$, può essere modificata in ogni nuovo interscambio di messaggi senza che i due partner impegnati nella comunicazione debbano incontrarsi di persona. I due partner potranno così utilizzare la potenza dei sistemi asimmetrici per lo scambio a distanza di chiavi segrete, che resisteranno ad attacchi *chosen plain-text* se "prive di semantica". Inoltre la loro comunicazione non soffrirà della lentezza propria dell'*RSA* in quanto questo cifrario sarà utilizzato molto di rado e le comunicazioni saranno brevi perché lo sono le chiavi. In sintesi questo protocollo scambia i messaggi segreti nella forma $\langle \mathcal{C}_{RSA}(k[session], k[pub]), \mathcal{C}_{AES}(m, k[session]) \rangle$. La stessa chiave $k[session]$ potrebbe essere utilizzata in più comunicazioni successive per ammortizzare il costo di utilizzazione del cifrario asimmetrico, ma questa prassi è in genere sconsigliata per rendere più sicuro il protocollo nel suo complesso.

Nel sistema ibrido *RSA/AES* il mittente ha il compito di generare la chiave di sessione e inviarla al destinatario prima del messaggio vero e proprio. Perché ciò sia

possibile il mittente deve disporre di risorse computazionali sufficienti a garantire la creazione di una chiave sicura, il che non è affatto banale. In ambienti distribuiti a larga diffusione, come per esempio quello di Internet, si preferisce attribuire pari responsabilità al mittente e al destinatario che tra l'altro nella maggioranza dei casi non si conoscono e potrebbero non fidarsi l'uno dell'altro. In questo ambito si preferisce parlare di due partner piuttosto che di mittente e destinatario: con un linguaggio ormai consolidato i due partner sono indicati come `Alice` e `Bob` (forse perché le loro iniziali sono A e B) e si forniscono loro strumenti pubblici per la generazione e lo scambio delle chiavi che possono sostituire l'*RSA*. Uno di questi è l'elegante algoritmo *DH* proposto da Diffie e Hellman: per la sua semplicità e resistenza agli attacchi passivi, cioè basati sull'intercettazione dei messaggi senza alterarli, *DH* è un algoritmo molto diffuso nei protocolli crittografici usati su Internet come per esempio il *Secure Socket Layer* (capitolo 11).

Protocollo *DH* per lo scambio pubblico delle chiavi. L'algoritmo sfrutta la proprietà di funzione one-way del logaritmo discreto (paragrafo 8.2). I due partner `Alice` e `Bob` generano la chiave $k[session]$ in modo *incrementale* inviandosi in chiaro alcuni pezzi di essa; questi pezzi sono sufficienti a ricostruire la chiave stessa solo se vengono combinati con le informazioni segretamente in possesso di ciascun partner e diverse per entrambi. In particolare:

- `Alice` e `Bob` si accordano pubblicamente su un primo p molto grande, tipicamente di un migliaio di cifre binarie (paragrafo 4.3), e su un generatore g di $\mathcal{Z}_p^*$. Si noti che g esiste sicuramente perché p è primo (vedi teorema 8.4 e sue conseguenze nel paragrafo 8.1), e inoltre si ha $\mathcal{Z}_p^* = \{1, 2, ..., p-1\}$. Se `Alice` e `Bob` non sono in grado di generare una coppia $\langle p, g \rangle$ possono utilizzare una coppia pubblica messa a disposizione nel sistema, che può essere la stessa per tutti gli utenti senza inficiare la sicurezza del protocollo.

- `Alice` estrae un intero positivo casuale $x < p$, calcola $X = g^x \texttt{ mod } p$ e spedisce in chiaro questo valore a `Bob`. Per definizione di generatore, valori diversi di x generano valori diversi di X che può così assumere qualsiasi valore tra 1 e $p-1$.

- `Bob` estrae un intero positivo casuale $y < p$, calcola $Y = g^y \texttt{ mod } p$ e spedisce in chiaro questo valore a `Alice`. Valori diversi di y generano valori diversi di Y.

- `Alice` riceve Y e calcola $k[session] = Y^x \texttt{ mod } p = g^{yx} \texttt{ mod } p$ sfruttando la sua conoscenza privata di x.

- `Bob` riceve X e calcola $k[session] = X^y \texttt{ mod } p = g^{xy} \texttt{ mod } p$ sfruttando la sua conoscenza privata di y (ovviamente $g^{yx} \texttt{ mod } p = g^{xy} \texttt{ mod } p$).

Alla fine del protocollo entrambi i partner hanno generato la stessa chiave di sessione che viene così utilizzata per le cifrature simmetriche successive. Un crittoanalista passivo può aver intercettato i valori p, g, X, Y scambiati in chiaro tra i due partner, ma per calcolare la chiave di sessione deve risolvere l'equazione $X = g^x \bmod p$ rispetto a x, oppure $Y = g^y \bmod p$ rispetto a y, ovvero calcolare il logaritmo discreto di X, o di Y, che è un problema computazionalmente difficile (paragrafo 8.2) e quindi improponibile per valori di p molto grandi.

Le insidie non sono però scongiurate perché un *crittoanalista attivo* può condurre attacchi distruttivi sul protocollo se è in grado di modificare la comunicazione tra `Alice` e `Bob`. Questa situazione, nota come *man in-the-middle*, permette di compromettere il protocollo *DH* con grande facilità. Il crittoanalista prende il nome di `Eve` (non vi è alcun riferimento alla prima donna che sarebbe trattata da impicciona: il nome è contrazione fonetica di "eavesdropper" che in inglese indica chi ascolta in segreto le conversazioni altrui). Lo schema dell'attacco è indicato nella figura 8.1.

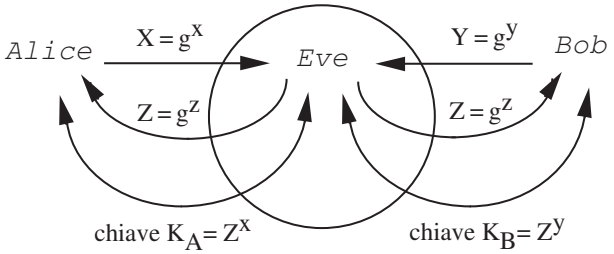


Figura 8.1: Attacco di tipo *man in-the-middle* al protocollo *DH*. I valori relativi alle chiavi sono calcolati $\bmod p$. Le frecce a due punte indicano la successiva comunicazione con chiavi false, gestita da `Eve`.

`Eve` sceglie un intero qualsiasi z, calcola il valore $Z = g^z \bmod p$ e si frappone sul canale bloccando le comunicazioni tra `Alice` e `Bob` per sostituirle con le proprie. `Eve` cattura i messaggi X e Y di `Alice` e `Bob` e risponde a entrambi con Z. `Alice` e `Bob` interpretano Z come proveniente dall'altro partner e costruiscono le chiavi (diverse) $K_A = Z^x \bmod p = g^{xz} \bmod p$ e $K_B = Z^y \bmod p = g^{yz} \bmod p$ con cui proseguiranno la comunicazione con `Eve` che colloquia con `Alice` usando K_A e con `Bob` usando K_B.

Nel prossimo capitolo esamineremo questo tipo di attacchi presentando la contromossa più comune nelle comunicazioni commerciali, cioè l'istituzione delle *Certification Authority* cui è demandata la protezione e certificazione delle chiavi. Si ricordi però che ogni contromossa crittografica può essere esposta a nuovi e imprevedibili

tipi di attacco, per cui lo scambio protetto delle chiavi rimane un passo cruciale per la sicurezza dei sistemi crittografici: i malfidenti potranno tranquillizzarsi solo attraverso un incontro personale con i loro partner per accordarsi sulla chiave di sessione, sempre che non credano all'esistenza di sosia.

8.7 Crittografia su curve ellittiche

Come abbiamo già osservato, le realizzazioni del cifrario *RSA* richiedono tempi di cifratura e decifrazione notevolmente maggiori di quelli dei cifrari simmetrici; inoltre il carico computazionale si è ulteriormente aggravato col progressivo aumento della lunghezza della chiave richiesta per garantire un utilizzo sicuro del cifrario: oggi si usano abitualmente chiavi di 1024 bit, ma il *NIST*[2] raccomanda di usare chiavi di almeno 2048 bit se si vogliono proteggere dati per periodi di tempo indicativamente fino al 2030, o addirittura chiavi di 3072 bit se la sicurezza è richiesta per periodi di tempo maggiori.

Negli ultimi vent'anni si è reso disponibile un sistema alternativo di crittografia a chiave pubblica, la *crittografia su curve ellittiche* (in breve *ECC*, dall'inglese *Elliptic Curve Cryptography*), in grado di offrire prestazioni migliori e maggiore sicurezza rispetto ai sistemi a chiave pubblica di "prima generazione", quali appunto l'*RSA* e il protocollo *DH* per lo scambio pubblico delle chiavi. In pratica, a parità di sicurezza la crittografia su curve ellittiche richiede chiavi di dimensioni di gran lunga inferiori, riducendo così il carico computazionale (vedi prossimo paragrafo 8.7.5). Si tratta di un vantaggio non indifferente che avremo modo di discutere nel seguito, ma che già da ora giustifica lo studio di alcuni degli aspetti principali della teoria delle curve ellittiche, la cui descrizione matematica è certamente meno familiare di quella della fattorizzazione degli interi[3].

Le curve ellittiche sono curve algebriche descritte da equazioni cubiche simili a quelle utilizzate per il calcolo della lunghezza degli archi delle ellissi. L'interesse per questa famiglia di curve risale alla metà del XIX secolo, ma la loro applicazione nei sistemi crittografici è molto recente. I primi a proporre algoritmi di cifratura basati su curve ellittiche furono infatti Victor S. Miller (IBM) e Neil Koblitz (University of Washington) nel 1985: il loro fondamentale contributo sta nell'aver suggerito, in modo del tutto indipendente, di sostituire negli algoritmi già esistenti le operazioni basate sull'algebra modulare con operazioni definite sui punti di una curva ellittica.

[2] Agenzia governativa degli Stati Uniti d'America per gli standard, vedi paragrafo 7.1.

[3] Per un'introduzione intuitiva e di gradevole lettura alla crittografia su curve ellittiche consigliamo l'ottimo articolo *"A (Relatively Easy To Understand) Primer on Elliptic Curve Cryptography"* di Nick Sullivan, pubblicato in rete da Ars Technica (`http://arstechnica.com`).

Ricordiamo anzitutto che un *campo* $\mathcal{K}$ è un insieme non vuoto dotato di due operazioni binarie interne, chiamate addizione e moltiplicazione, che soddisfano le proprietà associativa, commutativa, di esistenza dell'elemento neutro e di esistenza dell'inverso di ciascun elemento (ad eccezione dello zero per la moltiplicazione). Nella sua forma più generale, una curva ellittica E su un campo $\mathcal{K}$ è definita come l'insieme dei punti $(x, y) \in \mathcal{K}^2$ che soddisfano l'equazione algebrica

$$y^2 + axy + by = x^3 + cx^2 + dx + e\,,$$

dove $a, b, c, d, e \in \mathcal{K}$. La *caratteristica di un campo* è definita come il più piccolo numero naturale k tale che sommando k volte l'elemento neutro moltiplicativo del campo $\mathcal{K}$ (indicato con 1), si ottiene l'elemento neutro additivo di $\mathcal{K}$ (indicato con 0). Se un tale k non esiste, la caratteristica è 0 per definizione. Se la caratteristica del campo $\mathcal{K}$ è diversa da 2 e da 3, l'equazione che definisce una curva ellittica si può ridurre all'equazione cubica in *forma normale di Weierstrass*

$$y^2 = x^3 + ax + b\,. \tag{8.1}$$

La peculiarità che rende le curve ellittiche utilizzabili nelle applicazioni crittografiche è la possibilità di attribuire all'insieme dei punti di una curva la struttura algebrica di un *gruppo abeliano additivo*, ovvero di definire una legge di composizione interna che permette di associare ad ogni coppia di punti sulla curva, un terzo punto sempre sulla curva. La legge di composizione, chiamata *legge di addizione* o più semplicemente *somma sulle curve ellittiche*, è associativa, commutativa, ammette un elemento neutro (tale cioè che la somma di questo elemento con un punto della curva è uguale al punto stesso) ed è tale che sia definito l'inverso di ogni punto, come richiesto dalla definizione di gruppo abeliano. Una curva ellittica su $\mathcal{K}$ è quindi l'insieme dei punti $(x, y) \in \mathcal{K}^2$ che soddisfano l'equazione (8.1), oltre al punto neutro.

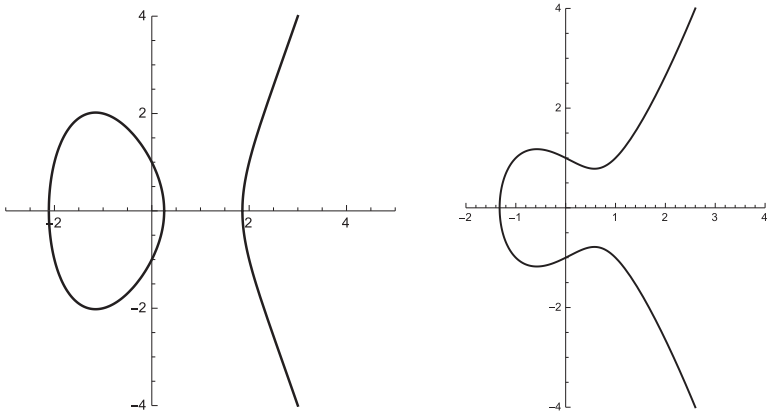
8.7.1 Curve ellittiche sui numeri reali

Prima di discutere le curve ellittiche sui campi finiti, di interesse per le applicazioni crittografiche, assumiamo per il momento che la curva sia definita nel campo $\mathcal{R}$ dei numeri reali e che sia dunque rappresentabile nel piano cartesiano. L'elemento neutro O sarà il punto all'infinito sull'asse delle ordinate. La curva sarà costituita dall'insieme $E(a, b)$ dei punti $(x, y) \in \mathcal{R}^2$ che soddisfano l'equazione (8.1), notando che essa contiene il punto all'infinito O.[4] Ovvero:

$$E(a, b) = \{(x, y) \in \mathcal{R}^2 \mid y^2 = x^3 + ax + b\}$$

[4] O è il punto all'infinito in direzione dell'asse y (ricordiamo in proposito che rette parallele si incontrano all'infinito nello stesso punto, che è unico indipendentemente dalla direzione in cui le

ove anche $a, b \in \mathcal{R}$. Assumiamo inoltre che sia $4a^3 + 27b^2 \neq 0$: questa condizione assicura che il polinomio cubico $x^3 + ax + b$ non abbia radici multiple e che di conseguenza la curva sia priva di *punti singolari* come "cuspidi" o "nodi" dove non sarebbe definita in modo univoco la tangente. Lavorare sui reali ci permetterà di fornire una rappresentazione grafica dell'operazione di addizione di due punti su una curva ellittica che, come vedremo, non corrisponde affatto alla somma delle loro coordinate.



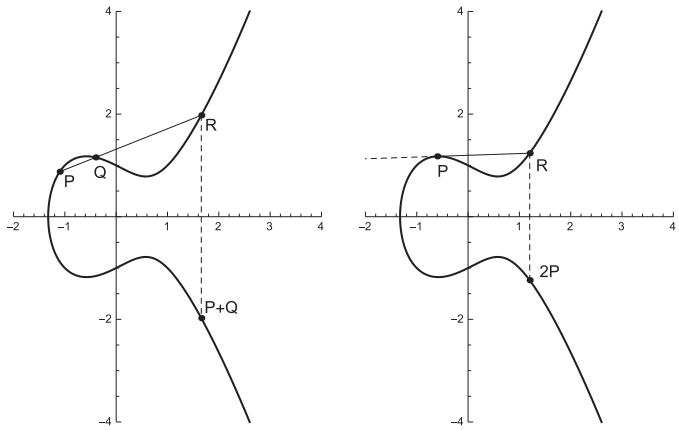
(a) Curva $y^2 = x^3 - 4x + 1$ (b) Curva $y^2 = x^3 - x + 1$

Figura 8.2: Esempi di curve ellittiche sul piano cartesiano.

Il grafico delle curve ellittiche sui reali può assumere una tra due possibili forme: una forma a due componenti (si veda la figura 8.2(a)) che si presenta quando il polinomio in x ha tre radici reali, e una forma con una sola componente (figura 8.2(b)) che si presenta quando il polinomio ha una sola radice reale. In entrambi i casi, le curve ellittiche presentano una simmetria orizzontale: ogni punto $P = (x, y)$ sulla curva si riflette rispetto all'asse delle ascisse nel punto $(x, -y)$ anch'esso sulla curva, che indicheremo con $-P$ (l'immagine speculare rispetto all'asse x del punto all'infinito O è lo stesso O).

Interessante per i nostri scopi è la proprietà secondo cui ogni retta interseca una curva ellittica in al più tre punti. Infatti intersecando una curva di terzo grado con una di primo grado (cioè con una retta) e sostituendo nella prima l'espressione di y

rette sono percorse). Per $x \to \infty$, l'equazione (8.1) tende alla $y^2 = x^3$, ovvero $y = \pm x^{3/2}$, con derivata $y' = \pm \frac{3}{2} x^{1/2}$: quindi per $x \to \infty$ si ha $y' \to \pm\infty$, ovvero la curva ha un asintoto verticale e contiene quindi il punto O.



(a) Somma di due punti P e Q. (b) Raddoppio di un punto P

Figura 8.3: Somma e raddoppio di punti su una curva ellittica.

della seconda, si ottiene un'equazione di terzo grado in x che ha tre soluzioni, reali o complesse, corrispondenti alle ascisse dei punti di intersezione tra la curva ellittica e la retta. Possiamo quindi avere una soluzione reale e due soluzioni complesse e coniugate, quindi un punto (reale) di intersezione, come avviene per esempio intersecando le curve di figura 8.2 con una retta orizzontale di ordinata sufficientemente elevata o con una retta verticale che incontri la curva solo nel punto all'infinito. Oppure si hanno tre soluzioni reali, quindi tre punti di intersezione[5]. Quindi se una retta interseca la curva $E(a, b)$ in due punti P e Q, coincidenti se la retta è una tangente, allora la retta interseca $E(a, b)$ anche in un terzo punto R. In particolare se la retta è verticale, essa interseca la curva $E(a, b)$ nei due punti P e Q (in questo caso $Q = -P$) e nel punto all'infinito O.

La definizione dell'operazione di *addizione su una curva ellittica* è basata sulle proprietà appena enunciate. Dati tre punti P, Q, R di una curva ellittica $E(a, b)$, se P, Q e R sono disposti su una retta, allora la loro somma si pone uguale al punto all'infinito:

$$P + Q + R = O.$$

Da questa definizione, è possibile ricavare la regola per sommare due punti P e Q

[5]Non possono esservi zero o due punti di intersezione perché le soluzioni complesse dell'equazione si presentano in coppie coniugate.

(figura 8.3):

- si considera la retta passante per P e Q, oppure l'unica tangente alla curva in P nel caso P e Q coincidano;
- si determina il punto di intersezione R tra la curva e la retta per P e Q, oppure tra la curva e la tangente in P per $P = Q$;
- si definisce somma di P e Q il punto simmetrico a R rispetto all'asse delle ascisse, ovvero si pone $P + Q = -R$.

La somma è ben definita in quanto, come abbiamo già osservato, anche $-R$ è un punto della curva. Nel caso particolare in cui P e Q abbiano la stessa coordinata x, ovvero se $Q = -P$, i due punti sono allineati lungo una verticale che interseca la curva nel punto all'infinito e si pone $P+Q = P+(-P) = O$, in quanto il simmetrico di O rispetto all'asse delle ascisse è sempre O. Questa legge di addizione attribuisce all'insieme dei punti di una curva ellittica $E(a, b)$ la struttura algebrica di un gruppo abeliano additivo, che ha per elemento neutro il punto all'infinito O; l'operazione soddisfa infatti le seguenti proprietà:

Chiusura: $\forall\ P, Q \in E(a, b)$, $P + Q \in E(a, b)$;

Elemento neutro: $\forall\ P \in E(a, b)$, $P + O = O + P = P$ (infatti le rette passanti per O sono verticali, dunque la retta per P e O interseca la curva in $-P$, il cui simmetrico è P);

Inverso: $\forall\ P \in E(a, b)$, esiste un unico $Q = -P \in E(a, b)$ tale che $P + Q = Q + P = O$;

Assiociatività: $\forall\ P, Q, R \in E(a, b)$, $P + (Q + R) = (P + Q) + R$;

Commutatività: $\forall\ P, Q \in E(a, b)$, $P + Q = Q + P$.

La formulazione algebrica dell'operazione di addizione permette di calcolare le coordinate del punto $P + Q$ a partire dalle coordinate di P e Q. Siano $P = (x_P, y_P)$ e $Q = (x_Q, y_Q)$ due punti distinti della curva, con $P \neq -Q$ altrimenti la loro somma sarebbe O. La somma $P + Q$ è data dal punto S (ovvero $-R$) di coordinate

$$\begin{aligned} x_S &= \lambda^2 - x_P - x_Q\,, \\ y_S &= -y_P + \lambda(x_P - x_S)\,, \end{aligned} \tag{8.2}$$

dove $\lambda = (y_Q - y_P)/(x_Q - x_P)$ è il coefficiente angolare della retta passante per P e Q. La derivazione di questi risultati è immediata: occorre trovare il terzo punto

di intersezione tra la retta per P e Q e la curva ellittica, risolvendo il sistema tra le equazioni della curva e della retta e sfruttando il fatto di conoscere già i due punti di intersezione P e Q; una volta individuate le coordinate del terzo punto di intersezione, è sufficiente cambiare il segno alla coordinata y per ottenere il suo inverso, definito come sappiamo dal punto simmetrico rispetto all'asse x.

Se $P = Q$, sommare P e Q corrisponde a raddoppiare P, e in questo caso, la retta per P e Q è sostituita dalla tangente alla curva in P, univocamente definita poiché per ipotesi la curva è priva di punti singolari. In particolare, se $y_P \neq 0$, le coordinate del punto $S = P + P = 2P$ sono ancora date dalle formule (8.2), dove ora λ rappresenta il coefficiente angolare della tangente, ovvero $\lambda = (3x_P^2 + a)/2y_P$. Se invece $y_P = 0$, la tangente alla curva in P è una retta verticale che interseca la curva nel punto all'infinito. Si pone dunque $2P = O$.

8.7.2 Curve ellittiche su campi finiti

Passiamo ora ad esaminare le curve ellittiche definite sui campi finiti, le uniche di interesse per la crittografia. Gli algoritmi crittografici hanno infatti bisogno di un'aritmetica veloce e precisa e pertanto non possono utilizzare le curve ellittiche sui reali che richiedono elaborazioni lente e inaccurate a causa degli errori di arrotondamento. Inoltre potremo lavorare in aritmetica modulare ove, come abbiamo già visto, alcuni problemi divengono computazionalmente difficili.

L'esempio più semplice di campo finito è l'insieme $\mathcal{Z}_p$ degli interi modulo un numero primo p. Nel campo $\mathcal{Z}_p$ tutte le operazioni si intendono in algebra modulare, e dunque coinvolgono interi compresi fra 0 e $p-1$. La caratteristica del campo è p, per cui ci limiteremo a considerare campi $\mathcal{Z}_p$ con $p > 3$ in modo da poter ridurre l'equazione generale di una curva ellittica alla forma normale di Weierstrass (8.1).

Le curve ellittiche con variabili e coefficienti ristretti agli elementi del campo $\mathcal{Z}_p$ sono chiamate *curve ellittiche prime*. Presi dunque $a, b \in \mathcal{Z}_p$, la curva ellittica prima $E_p(a, b)$ è definita come l'insieme dei punti che soddisfano l'equazione (8.1) modulo p insieme al punto all'infinito (elemento neutro):

$$E_p(a,b) = \{(x,y) \in \mathcal{Z}_p^2 \mid y^2 \bmod p = (x^3 + ax + b) \bmod p\} \cup \{O\}\,.$$

Evidentemente una curva ellittica prima contiene un numero finito di punti, e non è più rappresentata da una curva nel piano bensì da una nuvola di punti come mostrato nella figura 8.4 per $p = 67$, $a = -1$, $b = 1$. Questi punti vengono riportati nel sottospazio $0 \leq x \leq p-1$, $0 \leq y \leq p-1$ mentre, per quanto visto, i valori di y dovrebbero essere simmetrici a coppie rispetto all'asse x. Notando però che $-y \bmod p = p - y$ tutti i punti sono rappresentati in $0 \leq y \leq p-1$ traslando verso

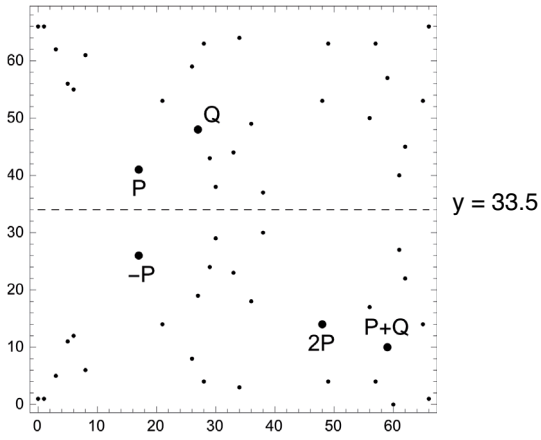


Figura 8.4: La curva ellittica prima $E_{67}(-1,1)$.

l'alto i valori negativi, e la curva prima $E_p(a,b)$ risulta simmetrica rispetto alla retta $y = p/2$. Così nella figura l'insieme dei punti della curva presenta una simmetria rispetto alla retta $y = 33.5$, con la sola eccezione del punto (60,0) simmetrico di sé stesso. Il punto $(x, -y \bmod p) = (x, p-y)$ definisce l'*inverso* di P e si indica con $-P$. Per esempio il punto $P = (17, 41)$ appartiene alla curva $E_{67}(-1,1)$, quindi anche il punto $-P = (17, 67-41) = (17, 26)$ le appartiene.[6]

Si può dimostrare che se il polinomio $(x^3 + ax + b) \bmod p$ non ha radici multiple, ovvero se $(4a^3+27b^2) \bmod p \neq 0$, i punti della curva prima $E_p(a,b)$ formano un gruppo abeliano finito rispetto all'operazione di addizione. Data la natura discreta delle curve prime non è più possibile descrivere questa operazione in termini geometrici, tuttavia le regole e la formulazione algebrica della somma definite per le curve ellittiche sui numeri reali si possono immediatamente adattare al caso finito, con l'unica accortezza di intendere ed eseguire tutte le operazioni in algebra modulare. Le coordinate del punto $P+Q$, si possono dunque determinare a partire dalle coordinate di P e di Q utilizzando le formule (8.2) ed eseguendo tutte le operazioni modulo p. Questo comporta tra l'altro che per calcolare il coefficiente angolare $\lambda = (y_Q - y_P)/(x_Q - x_P) \bmod p$ nel caso in cui $Q \neq \pm P$, oppure $\lambda = (3x_P^2 + a)/2y_P \bmod p$ nel caso dell'operazione di raddoppio del punto P, si deve calcolare l'inverso del denominatore in modulo, la

[6] La curva ha equazione $(y^2 = x^3 + x + 1) \bmod 67$. Per $x = 17$ si ha $y^2 = 6$, e per entrambi i valori $y = 41$ e $y = 26$ si ha $41^2 \bmod 67 = 26^2 \bmod 67 = 6$.

cui esistenza e unicità è garantita dal fatto che p è un numero primo (paragrafo 8.1). Eseguendo questi calcoli per i punti $P = (17, 41)$ e $Q = (27, 48)$ della curva di figura 8.4 si ottengono i punti $P + Q = (59, 10)$ e $2P = (48, 14)$.

Un parametro importante ai fini della sicurezza delle applicazioni crittografiche basate sulle curve ellittiche è l'*ordine* di una curva, ovvero il suo numero di punti. È evidente che una curva prima $E_p(a, b)$ può avere al più $2\,p+1$ punti: il punto all'infinito e le p coppie di punti (x, y) e $(x, -y)$ che soddisfano l'equazione $y^2 = x^3 + ax + b$ in modulo, al variare di x in $\mathcal{Z}_p$. Ma ci si può aspettare che la curva contenga $\Theta(p)$ punti considerando i *residui quadratici* di $\mathcal{Z}_p$, cioè gli elementi del campo che ammettono radice quadrata in $\mathcal{Z}_p$.

In generale si può dimostrare che esattamente $(p - 1)/2$ elementi di $\mathcal{Z}_p$ (oltre lo 0) sono *residui quadratici*, e che ciascuno di essi ha esattamente due radici quadrate in $\mathcal{Z}_p$ la cui somma è uguale a p.[7] Ricordando che l'equazione della curva è definita per y^2 ci si può aspettare che la curva stessa sia formata da circa p punti oltre al punto all'infinito: in genere non tutti i valori di x in $\mathcal{Z}_P$ danno luogo a un residuo quadratico y^2 nel campo, e quindi la curva non contiene punti con quelle ascisse; ma può avvenire che valori diversi di x generino lo stesso residuo quadratico e quindi esistano coppie di punti con x diverso e gli stessi valori di y. Si veda per esempio la figura 8.4 che contiene 54 punti, punto all'infinito incluso. Una stima più accurata è data dal seguente teorema che fornisce un limite superiore all'ordine delle curve, ma non ne stabilisce un valore minimo.

Teorema 8.6 (Hasse). *L'ordine N di una curva ellittica $E_p(a, b)$ verifica la disuguaglianza $|N - (p + 1)| \leq 2\sqrt{p}$.*

Oltre alle curve prime, la crittografia su curve ellittiche utilizza un'altra famiglia di curve ellittiche, chiamate *curve ellittiche binarie*. Si tratta delle curve i cui coefficienti e le cui variabili assumono valore nel campo $GF(2^m)$, costituito da 2^m elementi che si possono pensare come tutti gli interi binari di m cifre e su cui si può operare mediante l'aritmetica polinomiale modulare (per una descrizione del campo $GF(2^m)$ rimandiamo ai testi di algebra; le lettere GF stanno per *Galois Field*, o campo di Galois). Dato che la caratteristica del campo $GF(2^m)$ è uguale a 2, l'equazione che

[7] Presi due interi a, b in $\mathcal{Z}_p$, si considerino i residui quadratici $y_a = a^2 \bmod p$ e $y_b = b^2 \bmod p$, con $y_a \geq y_b$ (per $y_b \geq y_a$ il discorso è identico). Si ha $y_a - y_b = a^2 - b^2 \bmod p = (a + b)(a - b) \bmod p$, dunque $y_a = y_b$ se tale valore è uguale a zero, cioè se $(a + b)(a - b)$ è multiplo di p. Poiché si ha $0 \leq a+b \leq 2p-2$ e $0 \leq a-b < p-1$, risulta $y_a = y_b$ se e solo se $a = b = 0$ o $a+b = p$, verificate solo per le $(p-1)/2$ coppie di valori $\langle a = 1, b = p-1\rangle$, $\langle a = 2, b = p-2\rangle$, ... , $\langle a = (p-1)/2, b = (p+1)/2\rangle$, oltre alla coppia $\langle a = 0, b = 0\rangle$. Nell'esempio nella precedente nota 6, indipendentemente dalla curva il valore 6 è un residuo quadratico in $\mathcal{Z}_{67}$ e ammette le due radici quadrate 26 e 41.

definisce la curva ellittica non può essere espressa nella forma normale (8.1) e assume una forma leggermente differente, ovvero

$$y^2 + xy = x^3 + ax^2 + b \tag{8.3}$$

con $a, b \in GF(2^m)$. Una curva ellittica binaria $E_{2^m}(a, b)$ è dunque definita come l'insieme finito di tutti i punti $(x, y) \in GF(2^m)^2$ che soddisfano l'equazione (8.3), insieme a un punto all'infinito O. Se $b \neq 0$, si può definire un gruppo abeliano additivo su una curva ellittica binaria, con l'operazione di addizione descritta come segue:

- per ogni $P = (x_P, y_P) \in E_{2^m}(a, b)$: $P + O = P$ e $-P = (x_P, x_P + y_P)$;
- per ogni $P, Q \in E_{2^m}(a, b)$, con $P \neq \pm Q$, il punto $P + Q$ corrisponde al punto S di coordinate $x_S = \lambda^2 + \lambda + x_P + x_Q + a$ e $y_S = \lambda(x_P + x_S) + x_S + y_P$, dove $\lambda = (y_P + y_Q)/(x_P + x_Q)$;
- per ogni $P \in E_{2^m}(a, b)$, $2P = O$ se $x_P = 0$, altrimenti $2P$ è il punto S di coordinate $x_S = \lambda^2 + \lambda + a$ e $y_S = x_P^2 + (\lambda + 1)x_S$, dove $\lambda = x_P + y_P/x_P$.

Anche nel caso delle curve binarie, l'ordine N della curva si può stimare applicando il teorema di Hasse, che nel caso specifico fornisce la stima $|N - (2^m + 1)| \leq 2\sqrt{2^m}$.

Mentre le curve ellittiche prime sono particolarmente indicate per le applicazioni software, le curve ellittiche binarie sono per loro natura molto più adatte per le applicazioni hardware. Tutto quanto contenuto nei paragrafi seguenti si applica indistintamente a entrambe le famiglie di curve che non saranno più esplicitamente citate, con l'accortezza di impiegare le formule appropriate per il calcolo della somma di punti.

8.7.3 Il problema del logaritmo discreto per le curve ellittiche

Per definire un sistema crittografico a chiave pubblica usando le curve ellittiche occorre per prima cosa individuare una buona funzione one-way trap-door che ne garantisca la sicurezza. I due sistemi a chiave pubblica che abbiamo già esaminato, l'*RSA* e il protocollo di Diffie e Hellman per lo scambio delle chiavi sono basati rispettivamente sul problema della fattorizzazione e sul problema del logaritmo discreto nell'algebra modulare.

Per le curve ellittiche si può definire una funzione one-way trap-door analoga al logaritmo discreto nell'algebra modulare. In effetti, l'operazione di addizione di punti di una curva ellittica su un campo finito presenta delle analogie con l'operazione di prodotto modulare. In particolare, fissato un intero positivo k possiamo mettere

in relazione l'elevamento alla potenza k di un intero in modulo con la *moltiplicazione scalare* per k di un punto P di una curva ellittica, operazione che consiste nel sommare P con sé stesso k volte. Entrambe le operazioni si possono eseguire in tempo polinomiale: calcolare la potenza $y = x^k \bmod p$ richiede $\Theta(\log k)$ moltiplicazioni se si procede per quadrature successive, e allo stesso modo si può calcolare $Q = k\,P = P + P + \ldots + P$ (k volte) con $\Theta(\log k)$ raddoppi e $O(\log k)$ addizioni di punti grazie al metodo del raddoppio ripetuto:

- si scrive anzitutto k come somma di potenze del due: $k = \sum_{i=0}^{t} k_i 2^i$, dove $k_t \ldots k_1 k_0$ è la rappresentazione binaria dell'intero k, con $t = \lfloor \log_2 k \rfloor$;
- si calcolano in successione i punti $2\,P, 4\,P, \ldots, 2^t\,P$, ottenendo ciascuno dal raddoppio del precedente (complessivamente si eseguono t raddoppi);
- infine si calcola Q come somma dei punti $2^i\,P$, per $0 \leq i \leq t$ tale che $k_i = 1$.

Ad esempio il punto $Q = 13\,P$ si calcola con i tre raddoppi (indicati in grassetto) $\mathbf{2}\,P$, $4\,P = \mathbf{2}(2\,P)$, $8\,P = \mathbf{2}(4\,P)$, e le due addizioni $Q = P + 4\,P + 8\,P$.

L'esponenziazione modulare e la moltiplicazione scalare su una curva ellittica hanno dunque una struttura molto simile, pur con le diverse notazioni (moltiplicativa e additiva) utilizzate: la prima operazione si riferisce al processo di moltiplicazione di x per sé stesso k volte, mentre la seconda al processo di addizione di P con sé stesso k volte.

Consideriamo ora l'operazione inversa della moltiplicazione scalare su una curva ellittica, ovvero dati due punti P e Q trovare, se esiste, il più piccolo intero k tale che $Q = k\,P$. Questo problema è noto come il *problema del logaritmo discreto per le curve ellittiche*, in analogia al problema del logaritmo discreto su insiemi finiti, operazione inversa dell'esponenziazione modulare. Il numero intero k, quando definito, è chiamato *logaritmo in base P del punto Q*.

Come nell'algebra modulare, anche il problema del logaritmo discreto per le curve ellittiche risulta computazionalmente difficile perché tutti gli algoritmi noti per risolverlo hanno complessità esponenziale se i parametri della curva sono scelti opportunamente; in sostanza non è stato ancora trovato un algoritmo per risolvere questo problema che migliori il metodo a forza bruta basato sul calcolo di tutti i multipli di P fino a trovare Q, chiaramente improponibile se k è sufficientemente grande. Abbiamo così individuato una funzione computazionalmente trattabile in una direzione (calcolo di $Q = k\,P$ dati P e k), ma praticamente intrattabile nell'altra (calcolo di k dati P e $Q = k\,P$), vale a dire una funzione one-way trap door su cui basare la sicurezza della crittografia su curve ellittiche.

8.7.4 Applicazioni delle curve ellittiche in crittografia

Descriviamo ora due algoritmi crittografici che usano le curve ellittiche. Entrambi sono riproposizioni di algoritmi classici basati sul problema del logaritmo discreto ottenute sostituendo le operazioni dell'algebra modulare con le operazioni sui punti di una curva prima o binaria: più precisamente sostituendo il prodotto con la somma di punti e l'esponenziazione con la moltiplicazione scalare di un intero per un punto della curva. Come vedremo nel prossimo paragrafo, i due nuovi algoritmi garantiscono maggior sicurezza. Vi sono anche altri protocolli crittografici basati su curve ellittiche per i quali rimandiamo ai testi specialistici.

Protocollo *DH* per lo scambio di chiavi su curve ellittiche. Supponiamo che due partner `Alice` e `Bob` vogliano concordare una chiave segreta di sessione $K[session]$ da utilizzare per comunicazioni successive protette con un cifrario simmetrico. A tale scopo possono eseguire le operazioni seguenti:

- `Alice` e `Bob` si accordano pubblicamente su un campo finito e su una curva ellittica definita su questo campo. Quindi scelgono, nella curva, un punto B di *ordine* n molto grande, ove l'ordine n di un punto (da non confondere con l'ordine N della curva già definito) è il più piccolo intero positivo n tale che $n\,B = O$ (ovviamente $n \leq N - 1$ perché questo è il massimo numero di punti distinti che si possono raggiungere da B). `Alice` e `Bob` possono anche decidere di utilizzare una delle curve ellittiche raccomandate dal *NIST* (paragrafo 8.7.5). Come vedremo il punto B, anch'esso parametro pubblico del sistema, gioca un ruolo analogo a quello del generatore g nell'algoritmo *DH* sui campi finiti.

- `Alice` estrae un intero positivo casuale $n_A < n$ come propria chiave privata, genera una chiave pubblica $P_A = n_A B$ e la spedisce in chiaro a `Bob`. La chiave pubblica corrisponde dunque ad un punto della curva scelto casualmente.

- Analogamente, `Bob` estrae un intero positivo casuale $n_B < n$ come propria chiave privata, genera una chiave pubblica $P_B = n_B B$ e la spedisce in chiaro a `Alice`. Di nuovo, la chiave pubblica è un punto della curva scelto casualmente.

- `Alice` riceve P_B e calcola $n_A P_B = n_A n_B B = S$ usando la sua chiave privata n_A (si noti che S è un punto della curva).

- `Bob` riceve P_A e calcola $n_B P_A = n_B n_A B = S$ usando la sua chiave privata n_B.

- A questo punto `Alice` e `Bob` condividono lo stesso punto S determinato dalle scelte casuali di entrambi. Per trasformare S in una chiave segreta k per la

cifratura simmetrica convenzionale occorre convertirlo in un numero intero. Ad esempio si pone $k = x_S \bmod 2^{256}$ (ovvero k è costituito dagli ultimi 256 bit dell'ascissa di S).

Un crittoanalista che ha intercettato i messaggi P_A e P_B scambiati in chiaro tra i due partner e che conosce i parametri della curva ellittica impiegata e il punto B, non è in grado di violare lo schema in quanto per calcolare S dovrebbe calcolare n_A dati B e $P_A(= n_A B)$, oppure n_B dati B e $P_B(= n_B B)$, ovvero dovrebbe risolvere il problema del logaritmo discreto per le curve ellittiche, certamente intrattabile in questo caso date le dimensioni dei valori coinvolti[8]. Il protocollo è comunque vulnerabile agli attacchi attivi di tipo *man-in-the-middle*, esattamente come il protocollo *DH* sui campi finiti (paragrafo 8.6).

Scambio di messaggi cifrati su curve ellittiche. Presentiamo ora un algoritmo per lo scambio di messaggi cifrati, la cui versione classica è nota come cifrario a chiave pubblica di ElGamal. Per cifrare un messaggio m codificato come numero intero utilizzando le curve ellittiche occorre per prima cosa trasformare m in un punto di una curva prima o binaria, che sarà a sua volta trasformato in un nuovo punto da usare come testo cifrato. Non si tratta di un compito semplice e infatti non è noto alcun algoritmo deterministico polinomiale per effettuare una tale trasformazione. Esistono tuttavia degli algoritmi randomizzati molto efficienti che hanno una probabilità arbitrariamente bassa di fallire, cioè di non produrre un punto della curva. Tra questi vi è l'*algoritmo di Koblitz* che illustriamo brevemente.

Supponiamo di voler trasformare un numero intero positivo $m < p$ in un punto di una curva ellittica prima $E_p(a, b)$. Usando m come ascissa, la probabilità di trovare un punto della curva è pari alla probabilità che $m^3 + am + b \pmod p$ sia un residuo quadratico; che è circa $1/2$ come abbiamo visto discutendo il Teorema 8.6. Pertanto si fissa un intero positivo h tale che $(m + 1)\, h < p$, e si considerano come potenziali valori dell'ascissa del punto della curva gli interi $x = mh + i$, al variare di i da 0 a $h - 1$. Poiché $x^3 + ax + b$ deve uguagliare il quadrato di y, il tutto $\bmod p$, per ciascun x si prova ad estrarre la radice quadrata y di $x^3 + ax + b \pmod p$, operazione polinomiale se p è primo (paragrafo 8.2): se questa radice esiste si prende $P_m = (x, y)$ come punto sulla curva corrispondente a m, altrimenti si incrementa i e si riprova con il nuovo valore di x. La procedura si itera fino a quando si trova una radice, oppure i raggiunge il valore h, e si conclude che non è stato possibile trasformare il messaggio in un punto della curva. Benché la distribuzione delle ascisse dei punti

[8] Si ritiene, anche se non vi è una dimostrazione formale, che non sia possibile ricavare S da P_A e P_B con metodi che non richiedano il calcolo del logaritmo discreto.

della curva non segua una legge nota, ci si può aspettare che la probabilità che, per ogni valore di x considerato, $x^3 + ax + b$ non sia un quadrato modulo p sia circa $1/2$: quindi il metodo ha una probabilità complessiva di successo pari a circa $1 - 2^{-h}$.[9] Per risalire al messaggio dal punto $P_m = (x, y)$ individuato basta calcolare $m = \lfloor x/h \rfloor$: si noti in proposito che è stato scelto $(m + 1)\, h < p$ per evitare che il messaggio sia corrotto dalle operazioni in modulo. Il procedimento per trasformare un messaggio in un punto di una curva binaria è analogo.

Passiamo ora a descrivere l'algoritmo per lo scambio di messaggi cifrati. Anche in questo caso occorre accordarsi su una curva ellittica, su un punto B della curva di ordine n elevato, e su un intero h per la trasformazione dei messaggi in punti della curva. Ogni utente `U` genera la propria coppia chiave pubblica/chiave privata scegliendo un intero casuale $n_U < n$ come chiave privata e pubblicando la chiave $P_U = n_U B$. Supponiamo ora che l'utente `Mitt` voglia inviare un messaggio m a `Dest`.

Cifratura: `Mitt` trasforma il messaggio m nel punto P_m sulla curva; sceglie un intero casuale r e calcola i due punti $V = rB$, $W = P_m + rP_D$ dove P_D è la chiave pubblica di `Dest`; invia a `Dest` la coppia di punti $\langle V, W \rangle$.

Decifrazione: `Dest` riceve la coppia di punti $\langle V, W \rangle$ e ricostruisce il punto P_m con la sua chiave privata n_D, calcolando $W - n_D V = P_m + rP_D - n_D(rB) = P_m + r(n_D B) - n_D(rB) = P_m$, poiché $P_D = n_D B$. Quindi trasforma P_m nel messaggio m.

Per risalire al messaggio in chiaro, un crittoanalista che ha intercettato in crittogramma $\langle V = rB, W = P_m + rP_D \rangle$ dovrebbe calcolare r dai valori di B ed rB, cioè dovrebbe risolvere il problema del logaritmo discreto.

8.7.5 Sicurezza della crittografia su curve ellittiche

Come abbiamo visto, la sicurezza della crittografia su curve ellittiche è strettamente legata alla difficoltà di calcolare il logaritmo discreto di un punto, ovvero di calcolare l'intero k dati P e $Q = k\,P$, problema per cui ad oggi non è noto alcun algoritmo efficiente di risoluzione. Nonostante manchi una dimostrazione formale della sua intrattabilità, questo problema è considerato estremamente difficile, e in particolare molto più difficile dei tradizionali problemi della fattorizzazione degli interi e del logaritmo discreto nell'algebra modulare. Infatti per questi problemi esiste un algoritmo subesponenziale chiamato *index calculus* per calcolare il logaritmo discreto, che può essere utilizzato per attaccare sia il protocollo *DH* che il cifrario *RSA*. Per chiavi di

[9] Nell'improbabilissimo caso di insuccesso si può sostituire il messaggio con un altro equivalente.

b bit, infatti, questo algoritmo richiede in media $O(2^{\sqrt{b \log b}})$ operazioni ed è dunque più efficiente degli attacchi a forza bruta che ne richiedono $O(2^b)$. L'algoritmo *index calculus* - per la cui descrizione rimandiamo alla letteratura specialistica - sfrutta una struttura algebrica dei campi finiti che non è presente sulle curve ellittiche. Ad oggi (2014) nessuno è stato capace di progettare algoritmi di tipo *index calculus* per risolvere il problema del logaritmo discreto per le curve ellittiche: quindi i protocolli per tali curve sembrano invulnerabili a questo tipo di attacchi.

Al momento il migliore attacco noto al problema del logaritmo discreto per le curve ellittiche, detto *Pollard* ρ, richiede in media $O(2^{b/2})$ operazioni per chiavi di b bit ed è dunque pienamente esponenziale. Esistono attacchi particolarmente efficaci contro alcune famiglie di curve, ma non sono utili in generale poiché queste famiglie sono note e facilmente evitate.

In sostanza, nonostante anni di studi e ricerche di possibili attacchi di costo subesponenziale ai protocolli basati sulle curve ellittiche, ad oggi non si conoscono algoritmi per calcolare il logaritmo discreto che siano sostanzialmente più efficienti del metodo completamente enumerativo. Tutto ciò ha delle implicazioni evidenti sulla sicurezza: a parità di lunghezza delle chiavi, i protocolli basati sulle curve ellittiche (ECC) sono molto più sicuri del cifrario RSA e del protocollo DH, considerati equivalenti dal punto di vista della sicurezza in quanto entrambi soggetti agli attacchi basati sull'*index calculus*. Questo significa che per garantire lo stesso livello di sicurezza la crittografia su curve ellittiche richiede chiavi pubbliche di lunghezza minore. In proposito il *NIST* ha pubblicato la seguente tabella di equivalenza fra i livelli di sicurezza degli algoritmi simmetrici rispetto agli algoritmi a chiave pubblica. Se il confronto tra le due famiglie non è in realtà molto significativo a causa dell'impiego completamente diverso cui sono destinate, è molto interessante il confronto tra RSA ed ECC.

TDEA, AES (bit della chiave)	RSA e DH (bit del modulo)	ECC (bit dell'ordine)
80	1024	160
112	2048	224
128	3072	256
192	7680	384
256	15360	512

La tabella riporta la dimensione in bit delle chiavi che garantiscono livelli di sicurezza equivalenti nei tre diversi sistemi, dove due sistemi si considerano di sicurezza equivalente se è richiesto lo stesso costo computazionale per forzarli. Ad esempio, il

lavoro necessario per forzare un cifrario simmetrico come *AES* con chiave di 128 bit è pari a quello necessario per forzare un cifrario *RSA* a 3072 bit, mentre per garantire un'analoga sicurezza con un cifrario basato sull'*ECC* sono sufficienti 256 bit. La differenza tra i cifrari asimmetrici diventa ancora più evidente al crescere del livello di sicurezza richiesto: la dimensione delle chiavi per *RSA* e *DH* cresce molto più rapidamente della dimensione dei sistemi crittografici su curve ellittiche. Ad esempio, passando da chiavi simmetriche di 128 bit a chiavi di 256 bit, la dimensione in bit dei sistemi su curve ellittiche raddoppia, invece le chiavi *RSA* e *DH* diventano cinque volte più lunghe. Un'interpretazione alternativa di questi risultati è che all'aumentare della lunghezza della chiave, la sicurezza della crittografia a chiave pubblica su curve ellittiche cresce molto più della sicurezza dei sistemi a chiave pubblica tradizionali.

Oltre alla sicurezza, un'altra misura importante di cui tenere conto quando si confrontano sistemi crittografici diversi è la loro efficienza. Considerato che per chiavi di pari lunghezza il costo computazionale delle implementazioni dell'*RSA* e dei sistemi basati su *ECC* è confrontabile, il fatto di poter usare chiavi più corte a parità di sicurezza assicura vantaggi anche in termini di efficienza alla crittografia su curve ellittiche: elaborazioni più veloci, implementazioni hardware più compatte, consumi energetici contenuti e quindi minore dissipazione di calore, tutti vantaggi che rendono l'*ECC* particolarmente adatta all'utilizzo su dispositivi mobili.

Nonostante la crittografia a chiave pubblica di prima generazione sia ancora la norma nella maggior parte delle applicazioni, la crittografia su curve ellittiche inizia ad essere molto diffusa. Il *NIST* ha pubblicato un elenco di quindici curve ellittiche di cui raccomanda l'uso; si tratta di dieci curve binarie su $GF(2^m)$ per m compreso tra 163 e 571, e cinque curve prime su $\mathcal{Z}_p$ per primi p di 192, 224, 256, 384 e 521 bit. Queste curve garantiscono livelli di sicurezza equivalenti a quelli offerti dai cifrari simmetrici con chiavi di lunghezza variabile da 80 a oltre 256 bit ed è stato accertato che nessuna di esse appartiene alla famiglia delle curve suscettibili agli attacchi.

Oggi tutti i browser moderni supportano la crittografia su curve ellittiche; Stati Uniti, Regno Unito, Canada e alcuni altri paesi della NATO la utilizzano per proteggere le comunicazioni interne e tra i loro governi. Anche la moneta elettronica *Bitcoin* utilizza un protocollo di firma digitale basato su curve ellittiche: ma di firme digitali e moneta elettronica discuteremo nei prossimi capitoli.

Concludiamo questa discussione ricordando un interessante articolo del 2013 in cui viene definita una *misura universale di sicurezza*, che può aiutare a comprendere e visualizzare in modo intuitivo e anche divertente la sicurezza di un sistema crittografico. L'idea è di misurare la quantità di energia necessaria per forzare il sistema e confrontarla con la quantità di acqua che quell'energia potrebbe far bollire. Ad esempio, usando questa misura si può verificare che forzare un cifrario *RSA* a 228

bit richiede meno energia di quella necessaria per far bollire un cucchiaino di acqua: in confronto, l'energia necessaria per forzare un sistema basato su curve ellittiche a 228 bit potrebbe far bollire tutta l'acqua sulla Terra![10] A questo punto il lettore non dovrebbe avere alcun dubbio nella scelta del cifrario a chiave pubblica a cui affidare i suoi segreti.

[10] Arjen K. Lenstra, Thorsten Kleinjung e Emmanuel Thom: Universal Security - From Bits and Mips to Pools, Lakes - and Beyond. *Number Theory and Cryptography* 2013: 121-124. L'esempio sulle chiavi di 228 bit è tratto dall'articolo citato nella nota 3.

Capitolo 9

La firma digitale

In origine i metodi crittografici sono stati sviluppati per garantire la confidenzialità delle comunicazioni, in particolare tra coppie di persone in ambienti ristretti. Con la diffusione delle reti, e in particolare di Internet, il problema si è però enormemente esteso ed ha assunto connotazioni nuove. Sono così emerse tre funzionalità importanti che sono oggi richieste ai protocolli crittografici a seconda dell'uso e del livello di protezione desiderato. Esse sono:

Identificazione. Un sistema di elaborazione isolato o inserito in una rete deve essere in grado di accertare l'identità di un utente che richiede di accedere ai suoi servizi.

Autenticazione. Il destinatario di un messaggio deve essere in grado di accertare l'identità del mittente e l'**integrità** del crittogramma ricevuto, cioè che questo non sia stato modificato o sostituito nella trasmissione. Deve quindi essere difficile a un intruso spacciarsi per un altro utente o modificare i messaggi da questo inviati.

Firma digitale. È la funzionalità più complessa e risulta necessaria se il mittente e il destinatario non sono tenuti a fidarsi l'uno dell'altro. La firma digitale deve possedere tre requisiti: (1) il mittente non deve poter negare di aver inviato un messaggio m; (2) il destinatario deve poter accertare l'identità del mittente e l'integrità del crittogramma ricevuto (cioè deve poter autenticare il messaggio); (3) il destinatario non deve poter sostenere che un messaggio $m' \neq m$ è quello inviatogli dal mittente. Questi requisiti devono inoltre essere verificabili da una terza parte, cioè da un "giudice" possibilmente chiamato a certificare il corretto svolgimento della comunicazione.

Queste tre funzionalità non sono indipendenti ma ciascuna estende le precedenti, poiché l'autenticazione di un messaggio garantisce l'identificazione del mittente, e l'apposizione della firma garantisce l'autenticazione del messaggio. Ogni funzionalità è utilizzata per contrastare possibili *attacchi attivi* (in particolare di tipo *man in-the-middle*) che, come abbiamo più volte affermato, sono molto pericolosi.

Discuteremo attentamente ciascuna di queste funzionalità illustrandone alcune realizzazioni algoritmiche basate su cifrari asimmetrici.[1] E a questo scopo dobbiamo anzitutto introdurre una classe di funzioni di cui ancora non abbiamo parlato.

9.1 Funzioni hash one-way

Una *funzione hash* $f : X \to Y$ è definita per un dominio X e un codominio Y finiti e tali che $n = |X| \gg m = |Y|$. Nella costruzione di algoritmi queste funzioni sono impiegate per operare su elementi $x \in X$ attraverso la loro *immagine* (o *fingerprint*) $y = f(x) \in Y$, essenzialmente per due scopi. Anzitutto la rappresentazione di y richiede $\log_2 m$ bit ed è quindi più breve di quella di x che ne richiede $\log_2 n$; inoltre Y può avere un "struttura" assente in X: per esempio gli elementi y corrispondono alle posizioni di un vettore di m celle in cui memorizzare informazioni associate a m diversi elementi di X.

La differenza di cardinalità tra i due insiemi X e Y implica che esiste una partizione di X in sottoinsiemi disgiunti $X_1, ..., X_m$ tali che, per ogni valore dell'indice i, tutti gli elementi in X_i hanno come immagine uno stesso elemento di Y. Una buona funzione hash deve assicurare che: 1) i sottoinsiemi $X_1, ..., X_m$ abbiano circa la stessa cardinalità, in modo che due elementi x_h, x_k estratti a caso da X abbiano probabilità circa pari a $1/m$ di avere la stessa immagine in Y; 2) elementi di X "molto simili" tra loro appartengano a due sottoinsiemi diversi (per esempio se X è un insieme di interi, due elementi con valori prossimi devono avere immagini diverse). Naturalmente se si lavora su molti elementi di X alcuni avranno inevitabilmente la stessa immagine (si dice che vi è una *collisione*) e l'algoritmo che impiega la funzione hash f dovrà prevedere come affrontare questa situazione. Quando le funzioni hash sono applicate in crittografia le proprietà più interessanti sono però un po' diverse. In particolare sono importanti le *funzioni hash one-way*, tali che:

1. Per ogni elemento $x \in X$ è computazionalmente facile calcolare $f(x)$.

[1] Per realizzare queste funzionalità è anche possibile impiegare i cifrari simmetrici, ma ciò comporta notevoli complicazioni e alti costi computazionali, per cui non ne discutiamo in questo testo.

2. Per la *maggior parte* degli elementi $y \in Y$ è computazionalmente difficile determinare un x tale che $f(x) = y$.

3. È computazionalmente difficile determinare una coppia di elementi x', x'' in X tali che $f(x') = f(x'')$.

Ovviamente tutte le funzioni hash dovrebbero soddisfare la prima proprietà. La seconda proprietà (detta *one-way*) e la terza proprietà (detta *claw-free*), necessarie per impieghi crittografici, dovrebbero essere garantite attraverso una dimostrazione formale di intrattabilità; poiché però le funzioni che consentono una tale dimostrazione sono difficili da utilizzare ci si accontenta di funzioni semplici per cui siano falliti in pratica tutti i tentativi di attacco. In questa condizione si trovano le funzioni hash one-way più usate in crittografia tra cui emergono oggi quelle della famiglia *SHA*. Vi accenneremo brevemente perché la loro descrizione dettagliata è complessa e poco interessante: ricordando la struttura del *DES* vi si ritroveranno idee e tecniche simili.

Anzitutto un'importante funzione hash one-way chiamata *MD5* (per *Message Digest* versione 5), che costruisce un'immagine di 128 bit per qualunque messaggio, fu proposta da Rivest nel 1992 ed è stata usata per molto tempo in applicazioni crittografiche. Dieci anni più tardi sono stati però individuati diversi attacchi, in particolare legati alla imperfetta proprietà di claw-free della funzione, che ne sconsigliano l'impiego in molte applicazioni crittografiche. Data la sua semplicità ed efficienza *MD5* rimane però largamente usata in particolare fuori dalla crittografia, per esempio per la ricerca e il confronto di file su Internet. Una successiva funzione in competizione con essa è *RIPEMD-160* (per *Race Integrity Primitive Evaluation Message Digest*, anche se il nome suggerisce una versione "matura" delle funzioni *MD*), nata nel 1995 nell'ambito di un progetto dell'Unione Europea, che produce immagini di 160 bit ed è esente dai difetti di *MD5*. La struttura di queste funzioni non è dissimile da quelle della famiglia *SHA* che descriveremo ora, e che costituiscono uno standard.

La prima funzione *SHA* (per *Secure Hash Algorithm*), nota in seguito come *SHA0*, fu proposta dal *NIST* nel 1993 in competizione a *MD5* e fu presto ritirata a causa di un punto di debolezza scoperto al suo interno. La *National Security Agency* (*NSA*) la sostituì con la nuova versione *SHA1* che come la precedente opera su sequenze lunghe fino a $2^{64} - 1$ bit e produce immagini di 160 bit. *SHA1* è largamente usata in molti protocolli crittografici anche se, a causa di possibili attacchi di importanza forse più teorica che pratica, non è più certificata come standard dal 2010. Seguirono così diverse funzioni, anch'esse progettate da *NSA*, che operano su blocchi di dimensioni variabili a seconda della funzione scelta nella famiglia: largamente impiegata in applicazioni di alta sicurezza è *SHA-256* che genera un'immagine di 256 bit. Descriveremo a grandi linee *SHA1* che rimane la funzione più usata. Notiamo che tutte le altre,

compresa *MD5* che ne è in qualche modo capostipite, hanno una struttura simile e differiscono tra loro essenzialmente per la dimensione dei blocchi, il numero di fasi, e per alcune operazioni usate al loro interno.

SHA1 opera al suo interno su blocchi di 160 bit contenuti in un "buffer" di 5 "registri" A, B, C, D, E di 32 bit ciascuno, in cui sono caricati inizialmente valori fissi e pubblici. Il messaggio m di cui si vuole calcolare l'immagine è concatenato con una sequenza di *padding* che ne rende la lunghezza multipla di 512 bit. Il contenuto dei registri varia nel corso di cicli successivi in cui questi valori si combinano tra loro e con blocchi di 32 bit provenienti da m, nonché con alcuni parametri relativi al ciclo. Alla fine del procedimento, quando è stato letto l'intero messaggio, i registri contengono l'hash *SHA1(m)*. La struttura di massima per il calcolo della funzione è indicata nella figura 9.1.

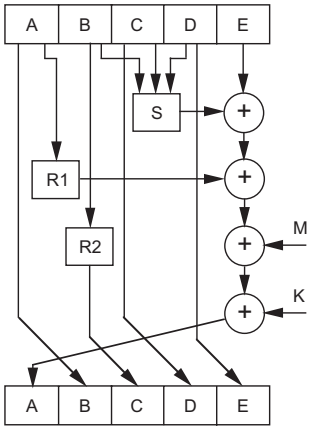


Figura 9.1: Struttura di massima del ciclo i-esimo di *SHA1*. La S-box realizza una funzione non lineare a 96 bit in ingresso e 32 in uscita. $R1, R2$ e K indicano due rotazioni cicliche e una quantità costante che dipendono da i. M è un blocco del messaggio. Il segno + indica l'addizione modulo 2^{32}.

Il lettore potrà sbizzarrirsi a cercare su Internet dei generatori di *SHA1* che producono l'immagine costruita dalla funzione per qualunque sequenza introdotta dalla tastiera: sarà particolarmente istruttivo constatare che sequenze similissime, per esempio che differiscono solo per un carattere, generano immagini hash completamente diverse.

Notiamo infine che per qualsiasi funzione hash one-way che accetti sequenze di ingresso T molto lunghe, e per le funzioni della famiglia *SHA* in particolare, è possibile aumentare la lunghezza delle immagini in modo molto semplice. Sia h la funzione hash e ℓ il numero di bit delle immagini generate. Si pone $D_0 \leftarrow T$ e si calcolano in successione i valori $D_i \leftarrow h(D_{i-1})D_{i-1}$ ottenuti concatenando la sequenza $h(D_{i-1})$ di lunghezza ℓ con la sequenza D_{i-1} di lunghezza crescente con i. Un'immagine complessiva di T, composta di $\ell \times k$ bit, si può ottenere concatenando le sequenze $h(D_0), \ldots, h(D_{k-1})$. In molte applicazioni un allungamento della sequenza immagine consente di aumentare la sicurezza.

9.2 Identificazione

Per introdurre il problema dell'identificazione consideriamo l'accesso di un utente alla propria casella di posta elettronica, o ai file personali memorizzati su un calcolatore ad accesso riservato ai membri della sua organizzazione. L'utente solitamente inizia il collegamento inserendo un codice di *login* (per esempio nome o cognome) e successivamente una *password*. Gli utenti sprovveduti si servono di password banali per paura di dimenticarle, mentre i più attenti scelgono password molto strane per rendere difficile una loro individuazione: non ci preoccuperemo dei primi perché alla loro disattenzione è comunque difficile porre rimedio, ma ci interesseremo degli utenti esperti che desiderano essere protetti efficacemente da attacchi esterni o interni al sistema a cui si connettono.

Se il canale attraverso il quale avviene il processo di identificazione è protetto in lettura e scrittura, un attacco può essere sferrato soltanto da un utente locale, per esempio dal suo *amministratore* (o *super-user*) che ha accesso a tutti i file memorizzati e in particolare a quello contenente le password degli utenti; o da un hacker che è riuscito a intrufolarsi nel sistema. Ne segue che il meccanismo di identificazione deve prevedere una opportuna cifratura delle password, che può essere realizzata in modo semplice ed efficace impiegando funzioni hash one-way. Il metodo che ora descriviamo è utilizzato per esempio nei sistemi UNIX.

Quando un utente `U` fornisce per la prima volta la password, il sistema associa a `U` due sequenze binarie S, Q che memorizza nel file delle password in luogo di questa. La S (*seme*) è prodotta da un generatore pseudo-casuale; la Q è l'immagine, secondo una funzione hash one-way, della sequenza ottenuta concatenando S alla password di `U`. A ogni successiva connessione dell'utente il sistema recupera il seme S dal file delle password, lo concatena con la password fornita da `U` e calcola l'immagine della nuova sequenza: se questa coincide con Q l'identificazione ha successo (figura 9.2).

. . .	. . .	. . .
bernasconi	S_b	Q_b
ferragina	S_f	Q_f
luccio	S_l	Q_l
. . .	. . .	. . .

Figura 9.2: Identificazione di un utente `U` su un canale protetto. Se la password trasmessa da bernasconi è P_b, il sistema controlla che sia $SHA1(P_bS_b) = Q_b$.

In questo modo un eventuale accesso illecito al file delle password non fornisce alcuna informazione interessante, poiché risulta computazionalmente difficile ricavare la password originale dalla sua immagine one-way. Inoltre la presenza di un seme casuale diverso per ogni utente impedisce al crittoanalista di scoprire se due utenti hanno scelto due password uguali guardando solo le loro immagini, e rende inoltre impossibile un attacco simultaneo alle chiavi su tutto il file.

L'assunzione sulla perfetta protezione del canale risulta ovviamente molto forte, e comunque inaccettabile in un contesto distribuito come quello di Internet. Se il canale è insicuro la password può essere intercettata durante la sua trasmissione *in chiaro*: è quindi opportuno che il sistema non maneggi mai direttamente la password ma una sua immagine inattaccabile. Descriviamo qui di seguito un semplice protocollo di identificazione basato sul cifrario *RSA* (capitolo 8), anche se il paradigma può essere facilmente adattato a un qualunque cifrario a chiave pubblica.

Siano $\langle e, n\rangle$ e $\langle d\rangle$ le chiavi pubblica e privata di un utente `U` che vuole accedere ai servizi offerti da un sistema `S`.

1. In corrispondenza a una richiesta di accesso proveniente da `U`, il sistema `S` genera un numero casuale $r < n$ e lo invia in chiaro a `U`.

2. `U` "decifra" il messaggio r calcolando $f = r^d \texttt{ mod } n$ con la sua chiave privata $\langle d\rangle$, e spedisce a `S` il valore di f. Questo valore prende il nome di *firma* di `U` su r.

3. `S` verifica la correttezza del valore ricevuto "cifrandolo" con la chiave pubblica $\langle e, n\rangle$ di `U` come $f^e \texttt{ mod } n$, e controllando che il risultato coincida con r.

Si noti l'inversione d'ordine delle operazioni di cifratura e decifrazione rispetto all'impiego standard del cifrario *RSA*: ciò è possibile perché le due funzioni sono commutative, ovvero $(x^e \texttt{ mod } n)^d \texttt{ mod } n = (x^d \texttt{ mod } n)^e \texttt{ mod } n$. Il valore f può essere generato solo da `U` che possiede la chiave privata $\langle d\rangle$: quindi se il passo 3 va a buon fine il

sistema ha la garanzia che l'utente che ha richiesto l'identificazione sia effettivamente `U`, anche se il canale è insicuro.

È importante osservare che l'identificazione con la chiave pubblica si presta a un attacco molto subdolo. Riferendosi allo scambio di messaggi descritto sopra si noti che `S` chiede a `U` di applicare la sua chiave segreta a una sequenza r che `S` stesso ha generato. Un sistema disonesto potrebbe inviare una r non-random, scelta di proposito per ricavare qualche informazione sulla chiave privata di `U` attraverso l'esame della trasformazione f di r costruita con tale chiave, soprattutto nel caso che `U` debba identificarsi molte volte con `S` e questo possa quindi impiegare una serie mirata di scelte di r. Si può dimostrare che questo tipo di attacco può permettere a `U` di ridurre notevolmente il campo di variabilità in cui, a sua conoscenza, può trovarsi la chiave privata di `U`. Nel paragrafo 9.7 presenteremo un metodo matematico estremamente interessante per impedire che da una comunicazione si possa estrarre più di quanto sia nelle intenzioni del comunicatore.

9.3 Autenticazione

Il processo di identificazione di cui abbiamo fin qui parlato è volto a stabilire l'identità di un utente ma non si occupa dei suoi messaggi. Consideriamo allora di nuovo la situazione fondamentale in cui due partner `Mitt` e `Dest` devono scambiarsi un messaggio m in maniera confidenziale, ma `Dest` deve anche *autenticare* il messaggio accertando l'identità di `Mitt` e l'integrità di m. Vediamo ora come ciò possa realizzarsi impiegando una chiave segreta k concordata a questo scopo tra `Mitt` e `Dest`.

In generale il meccanismo di autenticazione può essere descritto attraverso una funzione $\mathcal{A}(m, k)$ che genera un'informazione (in gergo *MAC*, per *Message Authentication Code*) utile a garantire la provenienza e l'integrità di m. Il *MAC* è allegato al messaggio: se non è richiesta confidenzialità `Mitt` spedisce la coppia $\langle m, \mathcal{A}(m, k)\rangle$ contenente il messaggio in chiaro, altrimenti spedisce la coppia $\langle \mathcal{C}(m, k'), \mathcal{A}(m, k)\rangle$, dove $\mathcal{C}$ è la funzione di cifratura e k' la chiave del cifrario scelto (segreta in caso simmetrico, pubblica in caso asimmetrico). In entrambi i casi `Dest` entra in possesso del messaggio m (dopo averlo eventualmente decifrato), ed essendo a conoscenza di $\mathcal{A}$ e k può ricalcolare $\mathcal{A}(m, k)$, confrontare il risultato con il valore inviatogli da `Mitt` e verificare che il *MAC* ricevuto sia proprio quello corrispondente al messaggio a cui risulta allegato. Se la verifica si conclude con successo il messaggio è autenticato, altrimenti `Dest` conclude che si è verificata un'alterazione casuale o maliziosa e scarta il messaggio.

Per economicità della trasmissione è necessario che la lunghezza del *MAC* sia molto minore di quella del messaggio (se questo viaggia in chiaro) o del corrispon-

dente crittogramma. Ricordando che lo spazio dei crittogrammi è almeno grande quanto quello dei messaggi, altrimenti più messaggi corrisponderebbero allo stesso crittogramma rendendo questo non univocamente decifrabile, segue che lo spazio dei *MAC* deve essere molto più piccolo dello spazio dei messaggi, quindi la funzione $\mathcal{A}$ non è iniettiva e come tale non invertibile. Quest'ultima proprietà deve però essere garantita in senso assai più forte, come vedremo qui di seguito.

Il lettore potrà trovare qualche similitudine e alcune essenziali differenze tra la funzione $\mathcal{A}$ e altre funzioni già note. Nei cifrari simmetrici, per esempio, la funzione di cifratura si serve di una chiave segreta k nota solo ai due partner, ma a differenza di questi cifrari $\mathcal{A}$ non è invertibile. Un altro confronto spontaneo si ha con i *codici ridondanti* che impiegano un bit di parità o più bit di controllo e correzione. Come in questi codici il *MAC* consente il controllo di integrità dei dati in presenza di errori, ma a differenza di essi permette di individuare anche errori *maliziosi*, cioè indotti ad arte.[2]

Quindi il *MAC* è uno strumento molto potente. Ne sono state proposte diverse realizzazioni basate su cifrari simmetrici e asimmetrici, nonché su funzioni hash one-way. Quest'ultimo approccio conduce a protocolli molto economici che sono attualmente i più usati in pratica: esaminiamone lo schema di base che ritroveremo nel protocollo *SSL* (paragrafo 9.6). Si impiega una funzione hash-one way h tipo *SHA*, e si definisce il *MAC* sulla concatenazione delle sequenze m e k come $\mathcal{A}(m,k) = h(m\,k)$. La prima conseguenza è che risulta computazionalmente difficile per un crittoanalista scoprire la chiave segreta k. Infatti, sebbene la funzione h sia nota a tutti e m possa viaggiare in chiaro o essere scoperto per altra via, k viaggia all'interno del *MAC*: per recuperare k si dovrebbe quindi invertire la h, il che è difficile per ipotesi. Da ciò consegue che il crittoanalista non può sostituire facilmente il messaggio m con un altro messaggio m', perché dovrebbe allegare alla comunicazione di m' l'informazione $\mathcal{A}(m',k)$ che può produrre solo conoscendo la chiave segreta k. In alternativa il crittoanalista dovrebbe registrare lo scambio di molti messaggi e relativi *MAC*, e tentare di estrapolare da questi informazioni utili sulla chiave k: è un attacco più pericoloso ma comunque difficile.

In sostanza il *MAC* è un'immagine breve del messaggio che può essere stata generata solo da un mittente conosciuto dal destinatario previ opportuni accordi. Ne esistono versioni *incondizionatamente sicure* che estendono la nozione di cifrario perfetto (capitolo 6) a quella di *autenticazione perfetta*, incorrendo però in alti costi computazionali: un esempio di tali protocolli prende non a caso il nome di *one-time MAC*.

[2]Per esempio alterando di proposito due bit di m la parità rimane assicurata ma $\mathcal{A}(m,k)$ svela l'alterazione, a meno che l'attacco non sia avvenuto conoscendo k.

Si può infine notare che l'impiego di un qualsiasi cifrario in modalità *CBC* (vedi paragrafo 7.5) consente di generare un *MAC*: basta prendere il blocco finale del crittogramma che è funzione dell'intero messaggio. La sicurezza di questi protocolli di autenticazione è determinata dal cifrario impiegato.

9.4 Firma digitale

Per discutere di firma digitale è opportuno esaminare anzitutto i motivi che rendono necessaria la firma di un documento di qualsiasi genere e le proprietà che essa deve in conseguenza possedere. La *firma manuale*, utilizzata comunemente per provare l'autenticità o la paternità di un documento, o per siglare un accordo formulato in esso, soddisfa alcuni requisiti fondamentali:

1. La firma è *autentica* e *non* è *falsificabile*, dunque costituisce prova che chi l'ha prodotta è veramente colui che ha sottoscritto il documento.

2. La firma *non* è *riutilizzabile*, e quindi risulta legata strettamente al documento su cui è stata apposta.

3. Il documento firmato *non* è *alterabile*, e quindi chi ha prodotto la firma è sicuro che questa si riferirà solo al documento sottoscritto nella sua forma originale.

4. La firma *non* può essere *ripudiata* da chi l'ha apposta, e costituisce dunque prova legale di un accordo o di una dichiarazione contenuta nel documento.

Anche se a volte questi requisiti sono annullati da impostori in grado di falsificare perfettamente le firme manuali, essi sono validi nella maggior parte dei casi e la firma rimane il mezzo standard per l'autenticazione di un documento. Una *versione digitale* della firma è quindi utilissima operando sulle reti dove si scambiano quotidianamente innumerevoli transazioni, ma la forma che essa deve possedere è del tutto nuova. La firma digitale infatti non può semplicemente consistere di una *digitalizzazione* (per esempio attraverso uno scanner) del documento originale firmato manualmente, poiché un crittoanalista potrebbe condurre su di essa attacchi molto semplici, il più rischioso ed elementare dei quali è "tagliare" dal documento digitale la parte contenente la firma e "copiarla" su di un altro documento. Quindi, a differenza dalla firma manuale, la firma digitale deve avere una forma che dipende dal documento su cui viene apposta, per essere inscindibile da questo.

Per progettare firme digitali si possono impiegare sia i cifrari simmetrici che quelli asimmetrici, anche se i primi danno luogo in genere a realizzazioni più complicate e

computazionalmente costose. In questo paragrafo esamineremo il classico *protocollo di firma di Diffie e Hellman* basato sui cifrari asimmetrici, in cui la chiave privata del mittente è utilizzata da questi per produrre la firma, e la sua chiave pubblica è utilizzata dal destinatario per verificarne l'autenticità.[3]

Sia U un generico utente, $k_U[prv]$ e $k_U[pub]$ le sue chiavi, $\mathcal{C}$ e $\mathcal{D}$ le funzioni di cifratura e decifrazione del cifrario asimmetrico. Un primo protocollo è il seguente:

Protocollo 1. Messaggio m in chiaro e firmato.

Firma. L'utente U genera la firma $f = \mathcal{D}(m, k_U[prv])$ per m. Il messaggio firmato può essere spedito a un qualunque altro utente V come tripla $\langle \mathtt{U}, m, f \rangle$.

Verifica. L'utente V riceve la tripla $\langle \mathtt{U}, m, f \rangle$ e verifica l'autenticità della firma f calcolando $\mathcal{C}(f, k_U[pub])$ e controllando che questo valore sia uguale a m. L'indicazione esplicita del mittente (ossia l'etichetta U nella tripla) consente al destinatario di selezionare l'appropriata chiave pubblica $k_U[pub]$ da utilizzare nel calcolo.

Come nel protocollo di identificazione basato su *RSA* (paragrafo 9.2), i processi di firma e verifica impiegano le funzioni $\mathcal{C}$ e $\mathcal{D}$ del cifrario in ordine inverso a quello standard: questa particolarità impone che le due funzioni siano commutative, ossia $\mathcal{C}(\mathcal{D}(m)) = \mathcal{D}(\mathcal{C}(m)) = m$.

Il Protocollo 1 soddisfa i requisiti propri della firma manuale. Infatti si ha:

- La firma f è autentica e non falsificabile (requisito 1), poiché la chiave $k_U[prv]$ utilizzata per produrla è nota solo a U e la funzione $\mathcal{D}$ è one-way.
- Per il motivo precedente il documento firmato $\langle \mathtt{U}, m, f \rangle$ non può essere alterato (se non da U), pena la perdita di consistenza tra m e f; quindi U può tranquillamente rilasciarlo ad altri (requisito 3). Ma poiché solo U può aver prodotto f, egli non può a sua volta ripudiare tale firma (requisito 4).[4]
- La firma f non è riutilizzabile su un altro documento $m' \neq m$ (requisito 2) poiché essa è immagine di m.[5]

[3] Non confondere questo protocollo con il protocollo *DE* per lo scambio di chiavi proposto dagli stessi autori, vedi paragrafo 8.6.

[4] Se si deve evitare che un messaggio inviato da U a V possa essere inviato da V a un terzo utente facendogli credere che gli sia stato inviato direttamente da U, occorre che U comunichi la quadrupla $\langle \mathtt{U}, \mathtt{V}, m, f' \rangle$, ove f' è la firma di U sulla coppia $\langle \mathtt{V}, m \rangle$.

[5] Il protocollo non può però impedire che siano generate più copie firmate dello stesso messaggio m: se è necessario impedire che ciò avvenga si deve introdurre qualche ulteriore complicazione.

Si noti inoltre che il protocollo è definito per un particolare utente U ma non per un particolare destinatario. Chiunque può convincersi dell'autenticità della firma facendo uso soltanto della chiave pubblica di U; non vi è quindi necessità di stabilire uno specifico controllo centralizzato, ma qualunque "giudice" potrà essere chiamato a verificare la firma in caso di controversia.

Per quanto sicuro e affidabile, il Protocollo 1 deve essere visto solo come schema di principio. Infatti esso comporta lo scambio di un messaggio di lunghezza doppia dell'originale poiché la dimensione della firma f è paragonabile a quella di m, e questo a sua volta potrebbe essere lunghissimo. Inoltre il messaggio non può essere mascherato poiché è comunque ricavabile pubblicamente dalla firma attraverso la verifica di questa. A tali inconvenienti si può ovviare trasformando il protocollo in vari modi: illustreremo ora il più importante di essi e ne discuteremo le possibilità di attacco, per giungere infine alla soluzione più comunemente adottata che combina cifrari asimmetrici e funzioni hash one-way. Ovviamente anche queste variazioni del protocollo soddisfano i requisiti della firma manuale.

La nuova soluzione consiste nell'*incapsulare* la firma entro il documento cifrato, simulando nel mondo digitale la spedizione di un documento in una busta sigillata. Come nel Protocollo 1 la firma si ottiene "decifrando" il messaggio con la chiave privata del mittente, ma tale firma è ora utilizzata anche per ricostruire il messaggio, evitando così di raddoppiare la lunghezza del testo trasmesso. Per un mittente U e un destinatario V, ora esplicitamente definito, si pone:

Protocollo 2. Messaggio m cifrato e firmato.

Firma e cifratura. Il mittente U genera la firma $f = \mathcal{D}(m, k_U[prv])$ per m, calcola il *crittogramma firmato* $c = \mathcal{C}(f, k_V[pub])$ con la chiave pubblica del destinatario, e spedisce la coppia $\langle \mathtt{U}, c \rangle$ a V.

Decifrazione e verifica. Il destinatario V riceve la coppia $\langle \mathtt{U}, c \rangle$, decifra il crittogramma con la sua chiave privata, cioè calcola $\mathcal{D}(c, k_V[prv])$ ottenendo un valore pari a f, quindi cifra tale valore con la chiave pubblica di U ottenendo $\mathcal{C}(\mathcal{D}(m, k_U[prv]), k_U[pub]) = m$. In tal modo V ricostruisce il messaggio m, attestando a un tempo l'identità del mittente se m è significativo (infatti un utente diverso da U ha probabilità praticamente nulla di generare un crittogramma che assuma qualsivoglia significato accettabile se "cifrato" con la chiave pubblica di U).

Si noti che le chiavi pubblica e privata giocano un ruolo differente a seconda che vengano utilizzate dal meccanismo di firma o da quello di cifratura. Per poter descrivere un algoritmo effettivo che segua il Protocollo 2 abbiamo bisogno di specializzare

le formule precedenti servendoci delle funzioni di cifratura e decifrazione di un cifrario asimmetrico reale. In particolare impiegando il cifrario *RSA*, ove $\langle d_U \rangle$, $\langle e_U, n_U \rangle$ sono le chiavi di `U` e $\langle d_V \rangle$, $\langle e_V, n_V \rangle$ sono le chiavi di `V`, abbiamo:

- `U` genera la firma del messaggio m come $f = m^{d_U} \texttt{ mod } n_U$; esegue la cifratura di tale firma calcolando $c = f^{e_V} \texttt{ mod } n_V$; spedisce a `V` la coppia $\langle \texttt{U}, c \rangle$.

- Ricevuta la coppia $\langle \texttt{U}, c \rangle$, `V` calcola $c^{d_V} \texttt{ mod } n_V$ ottenendo la firma f; "decifra" f con la chiave pubblica di `U` calcolando il valore $f^{e_U} \texttt{ mod } n_U$ che coincide esattamente con il messaggio m; se il valore ottenuto è significativo, `V` conclude che il messaggio è autentico.

La correttezza del procedimento è soggetta a una condizione che deriva dalle proprietà algebriche del cifrario *RSA*. Deve infatti valere la relazione $n_U \leq n_V$ perché risulti $f < n_V$ e la firma possa essere cifrata correttamente da `U` e spedita a `V`. Ma poiché ciò impedirebbe a `V` di inviare messaggi firmati e cifrati a `U`, ogni utente deve stabilire chiavi distinte per la firma e per la cifratura. Il sistema è allora organizzato come segue. Si fissa pubblicamente un parametro H molto grande, per esempio $H = 2^{1024}$, che determina la demarcazione tra chiavi di firma e chiavi di cifratura. Per istanziare il proprio accesso al sistema un utente `U` genera una coppia di chiavi privata/pubblica $\langle d_U \rangle$, $\langle e_U, n_U \rangle$, con $n_U > H$, da utilizzare nel processo di cifratura e decifrazione, e una seconda coppia $\langle d'_U \rangle$, $\langle e'_U, n'_U \rangle$, con $n'_U < H$, da utilizzare nel processo di firma e verifica. Si noti che il valore di H è molto grande e assicura così che tutte le chiavi possano essere scelte sufficientemente grandi e quindi inattaccabili in modo esauriente.

Anche se l'uso di chiavi diverse per la firma e la cifratura comporta un incremento della sicurezza totale del sistema, è bene prestare molta attenzione al meccanismo di firma che può essere attaccato subdolamente. Non entreremo in complessi aspetti specialistici e dettagli tecnici limitandoci a discutere alcuni pericoli cui è possibile far fronte con un po' di attenzione per giungere, in fondo al paragrafo, al protocollo di firma che resiste ai principali attacchi ed è il più usato nelle applicazioni reali.

Il principale attacco che consideriamo si basa sulla possibilità che un crittoanalista attivo riesca a procurarsi la firma di un utente su messaggi opportunamente scelti e possibilmente privi di senso; il che, come vedremo, potrebbe facilmente avvenire.

Attacco la Protocollo 2. Poniamo che un utente `U` invii una *risposta automatica* (*ack*) ogni volta che riceve un messaggio m cifrato e firmato, per informare il mittente che la comunicazione è giunta a destinazione; e poniamo, per rapidità di scambio, che il segnale di ack sia il messaggio ricevuto, firmato da `U` per

ricezione. Ciò consente a un crittoanalista attivo X installato sul canale di decifrare i messaggi inviati a U dagli altri utenti. Infatti:

1. X intercetta il crittogramma firmato $c = \mathcal{C}(f, k_U[pub])$ che è stato inviato a U da un altro utente V, lo rimuove dal canale e lo rispedisce a U facendogli quindi credere che è stato lui a inviare c. Ricordiamo che $f = \mathcal{D}(m, k_V[prv])$
2. U decifra c ottenendo $f = \mathcal{D}(c, k_U[prv])$ (firma corretta di V su m), ma supponendo che c provenga da X, verifica f utilizzando la chiave pubblica di X e ottenendo così un messaggio $m' = \mathcal{C}(f, k_X[pub]) \neq m$. Ovviamente m' è privo di senso, ma il sistema spedisce automaticamente a X un ack sotto forma di crittogramma firmato $c' = \mathcal{C}(f', k_X[pub])$, ove $f' = \mathcal{D}(m', k_U[prv])$ è la firma di U su m'.
3. Ora X può risalire da c' al messaggio originale m con le chiavi pubbliche di V e U attraverso la catena di calcoli: $\mathcal{D}(c', k_X[prv]) = f'$, $\mathcal{C}(f', k_U[pub]) = m'$, $\mathcal{D}(m', k_X[prv]) = f$, $\mathcal{C}(f, k_V[pub]) = m$.

L'invio di un messaggio di ack del tipo precedente implica che U firmi automaticamente ogni messaggio ricevuto. Poiché questo modo di agire può essere molto pericoloso, è consigliabile che U blocchi la risposta automatica e invii l'ack solo dopo aver esaminato il messaggio, scartando i messaggi privi di senso: il che però può implicare notevoli ritardi perché richiede l'esame preventivo di m.

Una mossa per contrastare l'attacco precedente e molti altri attacchi di tipo algebrico consiste nell'evitare la *firma diretta* del messaggio m, ma apporre la firma digitale su un'immagine di m ottenuta mediante una funzione h hash one-way pubblica tipo *MD5* o *SHA*. Il Protocollo 2 del paragrafo precedente diviene allora:

Protocollo 3. Messaggio m cifrato e firmato in hash.

Firma e cifratura. Il mittente U calcola $h(m)$ e genera la firma $f = \mathcal{D}(h(m), k_U[prv])$; calcola separatamente il crittogramma $c = \mathcal{C}(m, k_V[pub])$; spedisce a V la tripla $\langle \mathtt{U}, c, f \rangle$.

Decifrazione e verifica. Il destinatario V riceve la tripla $\langle \mathtt{U}, c, f \rangle$; decifra c come $\mathcal{D}(c, k_V[prv])$ ottenendo m; calcola separatamente $h(m)$ e $\mathcal{C}(f, k_U[pub])$: se questi due valori sono uguali conclude che il messaggio è autentico.

Il Protocollo 3 richiede lo scambio di una quantità maggiore di dati rispetto al Protocollo 2 a firma diretta, ma l'incremento è trascurabile, la firma è difficilmente assoggettata a manipolazioni algebriche e può essere calcolata più velocemente. Quindi il Protocollo 3 è frequentemente usato per ottenere firme sicure.

Notiamo infine che non è consigliabile invertire l'ordine delle operazioni di firma e cifratura. Nel mondo delle firme manuali questa inversione corrisponde a firmare la busta anziché il documento: un destinatario disonesto potrebbe aprire la busta e sostituire la lettera originale con un'altra, sostenendo poi che è quella inviatagli dal mittente. Nel mondo digitale si può simulare lo stesso attacco, ma non approfondiamo l'argomento perché non vi è alcun motivo che consigli di invertire l'ordine dei calcoli.

9.5 La *Certification Authority*

Se esaminiamo con attenzione i protocolli di firma descritti nel paragrafo precedente possiamo riscontrare una debolezza che deriva proprio da uno dei pregi dei cifrari asimmetrici: le chiavi di cifratura sono pubbliche e quindi non richiedono un incontro diretto tra gli utenti per il loro scambio. Una chiave pubblica può infatti essere recuperata dalla pagina Web del mittente oppure attraverso un e.mail: tutto ciò sembra ragionevole, ma non tiene conto del fatto che un crittoanalista attivo può intromettersi proprio in questa fase iniziale del protocollo, pregiudicando il suo corretto svolgimento.

Abbiamo già visto questo tipo di attacco *man in-the-middle* nel paragrafo 8.6 in relazione allo scambio di chiavi con il protocollo DH: il crittoanalista X si intromette nella comunicazione tra un mittente `U` e un destinatario `V` e si comporta a gli occhi di `U` come se fosse `V` e agli occhi di `V` come se fosse `U`, intercettando e bloccando le comunicazioni che provengono dai due. L'attacco sulle chiavi pubbliche è molto semplice:

1. `U` richiede a `V` la sua chiave pubblica, per esempio attraverso un e.mail.

2. `X` intercetta la risposta contenente $k_V[pub]$ e la sostituisce con la sua chiave pubblica $k_X[pub]$.

3. Da questo momento in poi `X` si pone costantemente in ascolto in attesa dei crittogrammi spediti da `U` a `V`, cifrati mediante $k_X[pub]$. Ciascuno di questi crittogrammi viene rimosso dal canale, decrittato da `X`, cifrato mediante $k_V[pub]$ e rispedito a `V`.

Poiché uno di questi crittogrammi potrebbe contenere la chiave di sessione di un cifrario ibrido, `X` potrebbe facilmente decrittare tutte le comunicazioni eseguite in quella sessione. `U` e `V` non si accorgono della presenza di `X` se il processo di intercettazione e rispedizione è sufficientemente veloce da rendere il relativo ritardo apparentemente attribuibile alla rete.

Per evitare attacchi come quello appena descritto sono nate le *(Key) Certification Authority* (brevemente `CA`), enti preposti alla certificazione di validità delle chiavi pubbliche. La `CA` autentica l'associazione ⟨ `utente, chiave pubblica` ⟩ emettendo un *certificato digitale*, così come l'anagrafe di un comune autentica l'associazione ⟨ `dati personali, fotografia` ⟩ rilasciando una carta di identità. Il certificato digitale consiste della chiave pubblica e di una lista di informazioni relative al suo proprietario, opportunamente *firmate* dalla `CA`. Per svolgere correttamente il suo ruolo la `CA` mantiene un archivio di chiavi pubbliche sicuro, accessibile a tutti e protetto da attacchi in scrittura non autorizzati. La chiave pubblica della `CA` è nota agli utenti che la mantengono protetta da attacchi esterni e la utilizzano per verificare la firma della `CA` stessa sui certificati. In dettaglio un certificato digitale contiene:

- Una indicazione del suo formato (numero di versione).
- Il nome della `CA` che l'ha rilasciato.
- Un numero seriale che individua univocamente questo certificato all'interno della `CA` emittente.
- La specifica dell'algoritmo utilizzato dalla `CA` per creare la firma elettronica, combinata con una descrizione del formato dei parametri di cui esso ha bisogno.
- Il periodo di validità del certificato (data di inizio e data di fine).
- Il nome dell'utente a cui questo certificato si riferisce e una serie di informazioni a esso legate.
- Un'indicazione del protocollo a chiave pubblica adottato da questo utente per la cifratura e la firma: nome dell'algoritmo, suoi parametri e *chiave pubblica dell'utente*.
- Firma della `CA` eseguita su tutte le informazioni precedenti.

Se un utente `U` vuole comunicare con un altro utente `V` può richiedere la chiave pubblica di quest'ultimo alla `CA`, che risponde inviando a `U` il certificato digitale di `V`. Poiché `U` conosce la chiave pubblica della `CA`, egli può controllare l'autenticità del certificato verificando il suo periodo di validità e la firma prodotta dalla `CA` su di esso. Se questi controlli vanno a buon fine, la chiave pubblica di `V` contenuta nel certificato è corretta e `U` la può utilizzare per avviare una comunicazione cifrata con `V`. Un crittoanalista potrebbe intromettersi soltanto falsificando quella certificazione, ma si assume che la `CA` sia *fidata* e il suo archivio delle chiavi sia inattaccabile. Attualmente

esistono in ogni stato diverse CA possibilmente organizzate *ad albero*: in questo caso la verifica di un certificato è leggermente più complicata e si svolge attraverso una catena di verifiche che vanno dalla CA di U alla CA di V.

Il protocollo suddetto richiede che ogni utente acceda alla CA per poter ricavare la chiave pubblica del suo partner, il che potrebbe costituire un collo di bottiglia nelle comunicazioni tra numerosi utenti. Il modo più semplice per ovviare a questo inconveniente è quello di imporre che ogni utente U mantenga localmente al suo sistema una copia del proprio certificato $\texttt{cert}_U$ (cioè quello contenente la propria chiave pubblica $k_U[pub]$), e una copia della chiave pubblica della CA. È anche desiderabile, come ora vedremo, che U esponga $\texttt{cert}_U$ sulla propria pagina Web, o che tale certificato sia reperibile pubblicamente in un sito contenente una lista di certificati prodotti da CA. Ciò consente all'utente di interagire con la CA una volta soltanto, e precisamente nel momento in cui egli desidera entrare nel "circuito delle comunicazioni sicure" fornendosi di un cifrario asimmetrico con propria chiave privata e pubblica: da questo momento la gestione delle chiavi pubbliche diventa completamente decentralizzata e può avvenire contestualmente allo scambio dei messaggi e delle loro firme. Vediamo come ciò sia possibile.

Per inviare a V un messaggio m cifrato e firmato, U si procura il certificato $\texttt{cert}_V$ di V chiedendolo a V stesso, o estraendolo da un sito ove esso è disponibile, e da tale certificato estrae la chiave pubblica $k_V[pub]$. Il metodo risultante è un'estensione del Protocollo 3 del paragrafo precedente, ove anziché la specificazione degli utenti appaiono i loro certificati.

Protocollo 4. Messaggio m cifrato, firmato in hash e certificato.

Firma, cifratura e certificazione. Il mittente U calcola $h(m)$ e genera la firma $f = \mathcal{D}(h(m), k_U[prv])$; calcola il crittogramma $c = \mathcal{C}(m, k_V[pub])$; spedisce a V la tripla $\langle \texttt{cert}_U, c, f \rangle$. (Si noti che $\texttt{cert}_U$ contiene la chiave $k_U[pub]$ e la specificazione della funzione h utilizzata).

Verifica del certificato. Il destinatario V riceve la tripla $\langle \texttt{cert}_U, c, f \rangle$ e verifica l'autenticità di $\texttt{cert}_U$ (e quindi della $k_U[pub]$ ivi contenuta) utilizzando la propria copia della chiave pubblica di CA.

Decifrazione e verifica della firma. V decifra il crittogramma c mediante la sua chiave privata ottenendo $m = \mathcal{D}(c, k_V[prv])$; verifica l'autenticità della firma apposta da U su m controllando che risulti $\mathcal{C}(f, k_U[pub]) = h(m)$.

Un punto debole di questo meccanismo è rappresentato dai certificati *revocati*, cioè non più validi in quanto emessi con (o relativi a) una chiave pubblica forzata.

Le CA mettono a disposizione degli utenti un archivio pubblico contenente i certificati che non sono più validi: la frequenza di controllo di questi certificati e le modalità della loro comunicazione agli utenti sono punti cruciali per la sicurezza e l'efficienza dei sistemi, e sono tutt'oggi oggetto di discussione.

Per concludere possiamo dire che l'impiego dei sistemi di cifratura asimmetrici combinato all'istituzione delle certification authority costituisce il metodo più semplice ed economico per garantire la sicurezza nello scambio di messaggi e di chiavi di sessione. Attacchi del tipo *man in-the-middle* sono sempre possibili a monte, ma comunque possono essere evitati se si stabilisce un incontro diretto tra utente e certification authority, da effettuarsi unicamente all'atto della istanziazione del sistema asimmetrico dell'utente. Se la presenza di una CA centralizzata è considerata troppo restrittiva ci si può rivolgere a meccanismi di *certificazione distribuita* (per esempio il *Pretty Good Privacy*, brevemente *PGP*), normalmente utilizzati nei sistemi di posta elettronica: anche questi meccanismi presentano pregi e difetti, per lo studio dei quali rimandiamo alla letteratura specializzata.

9.6 Il protocollo *SSL*

I procedimenti telematici quali *Internet banking*, *E-commerce* e molti altri sono ormai diffusissimi e richiedono in genere lo scambio via rete di importanti dati riservati, per esempio il numero della propria carta di credito da inserire in un modulo reperibile in una pagina Web. Poiché l'intercettazione di questi dati potrebbe esporre gli utenti a gravi danni, il sistema che offre il servizio via Internet deve garantire opportune contromisure per assicurare che la comunicazione avvenga in modo altamente confidenziale. D'altro canto molti utenti ignorano i meccanismi che sono alla base di questa comunicazione, assumendo un atteggiamento tra fatalista e fiducioso nella competenza e correttezza di chi fornisce il servizio. Cerchiamo quindi di fare un po' di chiarezza su come gli strumenti crittografici fin qui presentati possano essere combinati per definire protocolli sicuri, tra cui prendiamo a esempio il diffusissimo *Secure Socket Layer* (brevemente *SSL*) che garantisce la confidenzialità e l'affidabilità delle comunicazioni su Internet.[6]

Consideriamo un utente U che desidera accedere tramite Internet a un servizio offerto da un sistema S. Il protocollo *SSL* garantisce la **confidenzialità** poiché la trasmissione è cifrata mediante un sistema ibrido, in cui un cifrario asimmetrico è utilizzato per costruire e scambiare le chiavi di sessione, e un cifrario simmetrico

[6] L'esempio è "storico" in quanto, nel momento in cui è scritto questo testo, *SSL* è in uso da vent'anni. Ma la sua struttura ha valore sufficientemente generale per chiarire l'approccio da seguire in questi casi.

utilizza queste chiavi per criptare i dati nelle comunicazioni successive (paragrafo 8.6). Il protocollo garantisce inoltre la **autenticazione** dei messaggi accertando l'identità dei due partner, attraverso un cifrario asimmetrico o facendo uso di certificati digitali (paragrafo 9.5), e inserendo nei messaggi stessi un apposito *MAC* che utilizza una funzione hash one-way crittograficamente sicura (paragrafo 9.3). Per chi ha qualche dimestichezza con la struttura dei sistemi operativi ricordiamo che *SSL* si innesta tra un protocollo di trasporto affidabile, per esempio *TCP/IP*, e un protocollo di applicazione, per esempio *HTTP*. *SSL* è completamente indipendente dal protocollo di applicazione sovrastante che può utilizzarlo in maniera trasparente.[7]

SSL è organizzato su due livelli. Al livello più basso risiede il protocollo *SSL Record* che è connesso direttamente al protocollo di trasporto e ha l'obiettivo di incapsulare i dati spediti dai protocolli dei livelli superiori assicurando la confidenzialità e l'integrità della comunicazione. Al livello superiore risiede il protocollo *SSL Handshake* che è interfacciato all'applicazione sovrastante e permette all'utente e al sistema di autenticarsi, di negoziare gli algoritmi di cifratura e firma, e di stabilire le chiavi per i singoli algoritmi crittografici e per il *MAC*. In sostanza *SSL Handshake* crea un canale sicuro, affidabile e autenticato tra utente e sistema, entro il quale *SSL Record* fa viaggiare i messaggi incapsulandoli in blocchi opportunamente cifrati e autenticati. Ci concentreremo in particolare sul protocollo *SSL Handshake* che ha il compito di definire il canale attraverso una *suite crittografica* che contiene i meccanismi di cifratura e autenticazione, e le relative chiavi. Il protocollo *SSL Record* realizza poi *fisicamente* questo canale utilizzando la *suite* per cifrare e autenticare i blocchi dei dati prima di spedirli attraverso il protocollo di trasporto sottostante, ed ha interesse come meccanismo di rete piuttosto che crittografico.

Una sessione di comunicazione *SSL* è innescata da uno scambio di messaggi preliminari che prende il nome di *handshake* (stretta di mano) come il protocollo che lo governa. Attraverso questi messaggi il sistema `S` (in gergo *server*) e l'utente `U` (in gergo *client*) si identificano a vicenda e cooperano alla costruzione delle chiavi segrete da impiegare nelle comunicazioni simmetriche successive: costruzione che procede da un'unica informazione segreta detta *master secret* su cui l'utente e il sistema si accordano in modo incrementale.

I passi principali dell'*SSL Handshake* sono i seguenti (figura 9.3):

[7] La combinazione tra *SSL* e *HTTP* prende il nome di protocollo *HTTPS* ed è utilizzata da numerosi *Web-server* sicuri che adottano il prefisso `https://...` nel loro indirizzo. In questo caso un'icona rappresentante un "lucchetto chiuso" appare nella finestra del *browser*.

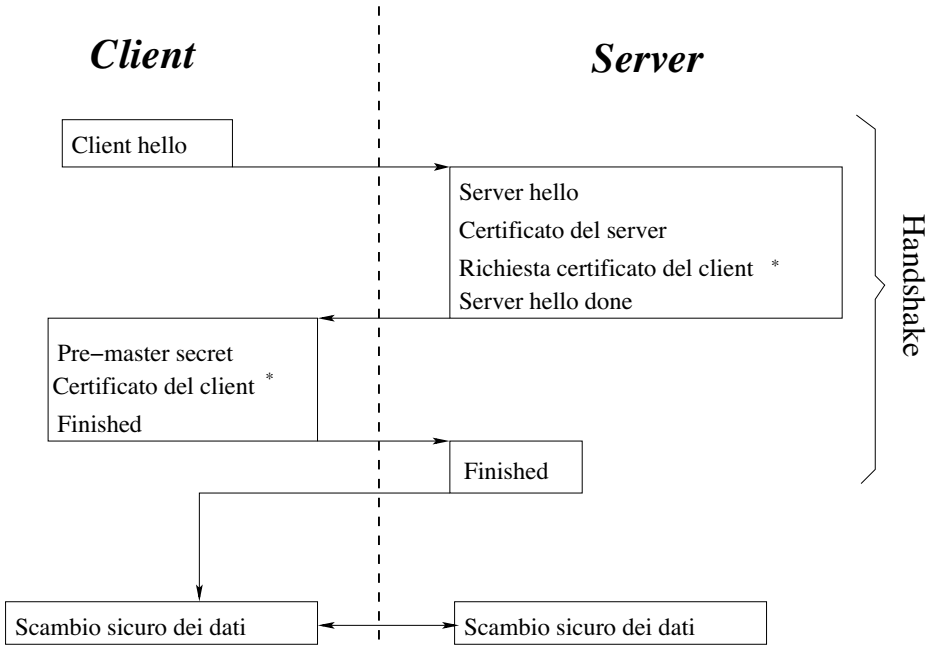


Figura 9.3: Lo scambio di messaggi in *SSL*. L'asterisco indica messaggi opzionali.

1. **Utente:** ***client hello.*** L'utente `U` manda al sistema `S` un messaggio, detto *client hello*, con cui richiede la creazione di una connessione *SSL*, specifica le "prestazioni" di sicurezza che desidera siano garantite durante la connessione, e invia una sequenza di byte casuali. In dettaglio il messaggio di *client hello* specifica la versione del protocollo *SSL* eseguito da `U`, un elenco di algoritmi di compressione, e una lista di *cipher suite*, cioè di cifrari e meccanismi di autenticazione che `U` può supportare ordinati secondo le sue preferenze. Per esempio la *cipher suite* indicata con la sigla `SSL_RSA_WITH_2TDEA_EDE_CBC_SHA1` prevede *RSA* per lo scambio delle chiavi di sessione; *2TDEA_EDE_CBC* per la cifratura simmetrica, ove *EDE* indica la sequenza *encryption-decryption-encryption* per il *Des Triplo* con due chiavi *2TDEA* e *CBC* indica l'impiego di un cifrario a composizione di blocchi (paragrafo 7.5); e prevede *SHA1* come funzione hash one-way per il calcolo dei *MAC*.

2. **Sistema: *server hello.*** Il sistema `S` riceve il messaggio di *client hello* e seleziona da questo un *cipher suite* e un algoritmo di compressione che anch'esso è in grado di supportare. Dopodiché invia all'utente un messaggio, detto *server hello*, che specifica la sua scelta e contiene una nuova sequenza di byte casuali. Se `U` non riceve il messaggio di *server hello*, interrompe la comunicazione.

3. **Sistema: invio del certificato.** Solitamente il sistema si autentica con l'utente inviandogli il proprio certificato digitale. Come abbiamo spiegato (paragrafo 9.5) esistono più Certification Authority (`CA`) possibilmente organizzate in una *struttura ad albero*, e non è detto che `U` e `S` si "affidino" alla stessa `CA` per l'emissione dei loro certificati. In questo caso il sistema invia all'utente una *sequenza* di certificati ordinati in accordo alla `CA` che li ha firmati, da quella che ha certificato `S` a quella che è alla radice dell'albero delle `CA`.

   Nel caso in cui i servizi offerti da `S` debbano essere protetti negli accessi, il sistema può richiedere all'utente di autenticarsi inviando il suo certificato digitale (vedi successivo passo 8).

4. **Sistema: *server hello done.*** Il sistema invia il messaggio *server hello done* con il quale sancisce la fine degli accordi sul *cipher suite* e sui parametri crittografici ad essa associati.

5. **Utente: autenticazione del sistema.** Per accertare l'autenticità del certificato ricevuto dal sistema, `U` controlla che la data corrente sia inclusa nel periodo di validità del certificato, che la `CA` che ha firmato il certificato sia tra quelle *fidate* e che la firma da essa apposta sia autentica (paragrafo 9.5). Nel caso che `S` invii una lista di certificati, `U` esegue la relativa catena di verifiche.

6. **Utente: invio del *pre-master secret* e costruzione del *master secret.*** L'utente `U` costruisce un *pre-master secret* costituito da una nuova sequenza di byte casuali, lo cifra con il cifrario a chiave pubblica del *cipher suite* su cui si è accordato con `S` e spedisce il relativo crittogramma a `S`. Con riferimento alla *cipher suite* presa in esempio al passo 1, `U` cifra il pre-master secret con l'algoritmo *RSA* utilizzando la chiave pubblica presente nel certificato di `S`.

   Il *pre-master secret* viene inoltre combinato da `U` con alcune stringhe note e con i byte casuali presenti sia nel messaggio di *client hello* che in quello di *server hello*. A tutte queste sequenze l'utente applica le funzioni hash one-way *SHA1* e *MD5* secondo una combinazione opportuna. Il risultato costituisce il *master secret*.

7. **Sistema: ricezione del *pre-master secret* e costruzione del *master secret*.** Il sistema decifra il crittogramma contenente il *pre-master secret* ricevuto dall'utente e calcola il *master secret* mediante le stesse operazioni eseguite dall'utente nel passo 6, in quanto dispone delle stesse informazioni.

8. **Utente: invio del certificato (opzionale).** Se all'utente viene richiesto un certificato (passo 3) ed egli non lo possiede, il sistema interrompe l'esecuzione del protocollo. Altrimenti l'utente invia il proprio certificato con allegate una serie di informazioni firmate con la sua chiave privata, che contengono tra l'altro il *master secret* che egli stesso ha generato e tutti i messaggi scambiati fino a questo momento (*SSL-history*). `S` controlla il certificato di `U` con operazioni simmetriche a quelle eseguite da `U` nel passo 5, e verifica l'autenticità e la correttezza della *SSL-history*. In presenza di anomalie la comunicazione con `U` viene interrotta.

9. **Utente/Sistema: messaggio *finished*.** Questo messaggio è il primo ad essere protetto mediante il *master secret* e il *cipher suite* su cui i due partner si sono accordati. Il messaggio viene prima costruito dall'utente e spedito al sistema, poi costruito dal sistema e spedito all'utente: nei due invii la struttura del messaggio è la stessa ma cambiano le informazioni in esso contenute. La sua costruzione consta di due passi. All'inizio si concatenano il *master secret*, tutti i messaggi di *handshake* scambiati fino a quel momento e l'identità del mittente (`U` oppure `S`). La stringa così ottenuta viene trasformata applicando le funzioni *SHA1* e *MD5*, dando origine a una coppia di valori che costituisce il messaggio *finished*. Questo messaggio è diverso nelle due comunicazioni poiché `S` aggiunge ai messaggi di *handshake* anche il messaggio *finished* ricevuto dall'utente. Il destinatario della coppia (`U` oppure `S`) non può invertire la computazione precedente in quanto generata da funzioni one-way, ma ricostruisce l'ingresso delle due funzioni *SHA1* e *MD5*, ricalcola queste funzioni e controlla che la coppia così generata coincida con quella ricevuta, come dimostrazione che la comunicazione è avvenuta correttamente.

Il *master secret* viene utilizzato da `U` e da `S` per costruire una propria *tripla* contenente la chiave segreta da adottare nel cifrario simmetrico (per esempio *3DES_EDE*), la chiave da adottare in una funzione hash one-way per la costruzione del *MAC* (per esempio *SHA1*), e la sequenza di inizializzazione per cifrare in modo aperiodico messaggi molto lunghi (usata per esempio nel *CBC*). Le triple dell'utente e del sistema sono diverse tra loro ma note a entrambi i partner: ciascuno usa la propria, il che aumenta la sicurezza delle comunicazioni.

Il canale sicuro approntato dal protocollo *SSL Handshake* viene realizzato dal protocollo *SSL Record*. I dati sono *frammentati* in blocchi di lunghezza opportuna; ciascun blocco viene numerato, compresso, autenticato attraverso l'aggiunta di un *MAC*, cifrato mediante il cifrario simmetrico su cui l'utente e il sistema si sono accordati, e finalmente trasmesso dall'*SSL Record* utilizzando il protocollo di trasporto sottostante. Il destinatario della comunicazione esegue un procedimento inverso sui blocchi ricevuti: decifra e verifica la loro integrità attraverso il *MAC*, decomprime e riassembla i blocchi in chiaro, infine li consegna all'applicazione sovrastante.

Se il sistema `S` non dispone di un certificato, il protocollo di *SSL Handshake* può ancora essere eseguito se i due partner si servono di un algoritmo asimmetrico per lo scambio segreto del *pre-master secret*, quale ad esempio quello del protocollo *DH* (paragrafo 8.6) che dovrà essere specificato nel *cypher suite*. In questo caso però, come abbiamo già osservato, non si ha protezione dall'intrusione di un crittoanalista attivo che esegua un attacco tipo *man-in-the-middle*. Se però il protocollo prevede il certificato di `S` ed eventualmente quello di `U`, la comunicazione che si stabilisce è confidenziale e affidabile, ed è protetta anche dagli attacchi di un crittoanalista in grado di modificare, catturare e sostituire i messaggi che transitano sul canale. Ciò può essere compreso esaminando i criteri che hanno orientato la progettazione del protocollo, in particolare:

***Client hello* e *server hello*.** Nei passi di *hello* i due partner creano e si inviano due sequenze casuali per la costruzione del *master secret*, che risulta così diverso in ogni sessione di *SSL*. Questo impedisce a un crittoanalista di riutilizzare i messaggi di *handshake* catturati sul canale in una sessione precedente, per sostituirsi a `S` in una successiva comunicazione con `U` (attacco di *reply*).

***MAC* dei blocchi di dati.** Il protocollo *SSL Record* numera in modo incrementale ogni blocco di dati e autentica il blocco attraverso un *MAC*. Per prevenire la modifica del blocco da parte di un crittoanalista attivo, il *MAC* viene calcolato come immagine hash one-way di una stringa costruita concatenando il contenuto del blocco, il numero del blocco nella sequenza, la chiave del *MAC*, e alcune stringhe note e fissate apriori. La specificazione del numero del blocco consente di prevenire l'impiego fraudolento e iterato del blocco stesso nell'ambito della stessa sessione (attacco di *reply*), perché si dovrebbe cambiare quel numero e il *MAC* corrispondente. (Una conseguenza negativa per gli utenti è però che, se un blocco viene perduto nella trasmissione, i blocchi successivi devono essere ricreati e spediti nuovamente). Poiché i *MAC* sono cifrati insieme al messaggio, un crittoanalista non può alterarli senza aver forzato prima la chiave simmetrica

di cifratura: quindi un attacco volto a modificare la comunicazione tra i due partner è difficile almeno quanto quello volto alla sua decrittazione.

Il sistema è autenticato. Il canale definito dal protocollo *SSL Handshake* è immune da attacchi del tipo *man in-the-middle* poiché il sistema viene autenticato con un certificato digitale. L'utente ha così la possibilità di comunicare il *pre-master secret* al sistema in modo sicuro attraverso la chiave pubblica presente nel certificato di `S`. Solo `S` può decifrare quel crittogramma e quindi costruire il corretto *master secret*, su cui si fonda la costruzione di tutte le chiavi segrete adottate nelle comunicazioni successive. Quindi solo il sistema `S`, ossia quello a cui si riferisce il certificato, potrà entrare nella comunicazione con l'utente `U`.

L'utente può essere autenticato. Il certificato dell'utente (se richiesto) e la sua firma apposta sui messaggi scambiati nel protocollo (*SSL-history*) consentono al sistema di verificare che l'utente è effettivamente quello che dichiara di essere e che i messaggi sono stati effettivamente spediti da esso. Se ciò non si verifica, il sistema deduce che il protocollo è stato alterato casualmente o maliziosamente e interrompe la comunicazione.

Sottolineiamo come l'*opzionalità* dell'autenticazione dell'utente sia uno dei fattori che hanno favorito la diffusione di *SSL* e di altri protocolli simili nelle transazioni commerciali via Internet. Infatti mentre i sistemi sono solitamente creati da società che possono facilmente ottenere certificati da parte di `CA` fidate, per gli utenti, spesso singoli privati, la necessità di certificazione può costituire un serio ostacolo pratico ed economico, o anche di semplice opportunità per la gestione di grandi masse di certificati da parte delle `CA`. Se l'autenticazione dell'utente è comunque necessaria per garantire la protezione di alcuni servizi, si preferisce procedere creando il canale sicuro attraverso *SSL* e poi autenticando l'utente con metodi tradizionali, per esempio mediante login e password, (paragrafo 9.2), o mediante il numero di carta e il codice di sicurezza per i pagamenti con carte di credito.

Il messaggio *finished*. Come visto questo messaggio viene costruito in funzione del *master secret* e contiene tutte le informazioni che i due partner si sono scambiati nel corso dell'*handshake*. Lo scopo è quello di consentire ai due partner di effettuare un ulteriore controllo sulle comunicazioni precedenti per garantire che queste siano avvenute correttamente, che essi dispongano dello stesso *master secret*, e che la loro comunicazione non sia stata oggetto di un attacco attivo.

Generazione delle sequenze casuali. Le tre sequenze casuali generate dall'utente e dal sistema e comunicate nei messaggi di *client hello*, *server hello* e *pre-*

master secret entrano crucialmente in gioco nella creazione del *master secret* e quindi nella generazione delle chiavi di sessione. In particolare la sequenza corrispondente al *pre-master secret* viene generata dall'utente e comunicata per via cifrata al sistema. La non predicibilità di questa sequenza è uno dei fulcri su cui si fonda l'intera sicurezza del canale *SSL*: una sua cattiva generazione renderebbe il protocollo molto debole, ribadendo l'importanza che rivestono i generatori pseudo-casuali nei processi crittografici (paragrafo 4.2).

Per concludere notiamo che il protocollo *SSL* è sicuro almeno quanto *il più debole cipher suite* da esso supportato. Poiché, dopo i tentativi di alcuni governi di limitare l'impiego della crittografia su Web per motivi di "sicurezza nazionale", dal Gennaio 2000 le norme internazionali non pongono alcuna limitazione sui cifrari impiegabili se non in alcuni paesi che si trovano in condizioni politiche particolari, è consigliabile disabilitare i propri sistemi dall'impiego di cifrari ormai notoriamente insicuri o di chiavi troppo corte. Similmente sono ovviamente da evitare connessioni con sistemi *anonimi*, cioè privi di certificato, che potrebbero risultare veri e propri "cavalli di troia" volti all'estorsione della password dell'interlocutore o di altri dati riservati.

9.7 Protocolli *Zero-Knowledge*

Nel 2012 il Turing Award, ovvero il più importante premio scientifico per l'informatica considerato equivalente al Premio Nobel per altre discipline, è stato assegnato a Shafi Goldwasser e Silvio Micali del Massachusetts Institute of Technology. Per la prima volta riceve il premio un italiano, Micali appunto, laureato in Matematica a Roma e ora anche cittadino americano. I loro studi hanno illustrato l'importanza degli eventi casuali nella crittografia dando poi luogo ai metodi di "dimostrazione a conoscenza zero" e costituiscono una pietra miliare nell'informatica teorica. In questo paragrafo cercheremo di far comprendere il significato dello schema matematico detto *zero-knowledge* nell'ambito dei protocolli di scambio d'informazione tra un *prover* P (dimostratore) e un *verifier* V (verificatore).[8] Compreso il senso del discorso ridiscuteremo poi come impostare un nuovo protocollo di identificazione e come l'introduzione di un meccanismo di zero-knowledge ne consenta la massima sicurezza.

La caratteristica principale di uno scambio di messaggi zero-knowledge è che il prover P riuscirà a dimostrare al verifier V di essere in possesso di una facoltà particolare o di una conoscenza specifica su un argomento senza comunicargli alcun'altra

[8] L'argomento è molto complesso e può essere approfondito nella letteratura specialistica affrontandolo con una solida conoscenza di computabilità e complessità di calcolo. Altre fonti di più semplice lettura sono in genere piuttosto deludenti: si salva, senza alcuna pretesa di profondità, la pagina Zero Knowledge di Wikipedia.

informazione salvo l'evidenza del suo possesso di tale facoltà o conoscenza. Introduciamo l'argomento come un semplice gioco: P afferma che, condotto su una qualsiasi spiaggia, è in grado con un'occhiata di contarne i granelli di sabbia (più seriamente potremmo dire che afferma di saper risolvere in tempo polinomiale un problema NP-hard che non sia *verificabile* in tempo polinomiale, vedi paragrafo 3.4). V controllerà che questa affermazione è vera con una probabilità arbitrariamente prossima a 1 (certezza) da lui stesso stabilita, senza però ricavare alcun'altra informazione sul metodo seguito da P per contare i granelli.

P e V adottano un protocollo iterativo di scambio, concordato in precedenza, consistente in un numero $k+1$ di domande e risposte ove il valore di k è scelto da V. Per semplificare ulteriormente le cose ammettiamo che V si accontenti che le risposte di P siano costituite da un bit che indica se il numero di granelli è pari o dispari. Alla iterazione $i = 0, 1, 2, ..., k$ la risposta deve essere $b_i = 0$ se il numero di granelli è pari, $b_i = 1$ se è dispari. Secondo il protocollo concordato V conduce P su una spiaggia a sua scelta e gli chiede di calcolare b_0 (come vedremo nulla cambia se P conosce già la spiaggia e il suo numero di granelli). Quindi V esegue le seguenti operazioni, iterandole per k volte se stabilirà che l'affermazione di P è vera, o arrestandosi prima se scoprirà che P è un impostore.

iterazione $i = 1, 2, ..., k$

V chiede a P di voltarsi per non osservare quello che farà;

V sceglie un bit random e lanciando una moneta;

if $(e = 0)$ V toglie un granello di sabbia e lo intasca **else** non fa nulla;

V chiede a P di guardare di nuovo la spiaggia e di calcolare il nuovo valore b_i;

if ($e = 0$ AND $b_i \neq b_{i-1}$) OR ($e = 1$ AND $b_i = b_{i-1}$) (cioè se le risposte di P ai passi i e $i-1$ sono consistenti tra loro) V passa all'iterazione $i+1$

else arresta il procedimento dichiarando che P è un impostore.

Se le k iterazioni vanno a buon fine V può stabilire che P ha la capacità asserita con probabilità $1 - 1/2^k$. Infatti se P è un impostore ha, a ogni iterazione i, una probabilità $1/2$ di indovinare il valore corretto di b_i pur senza saperlo calcolare, e tutti questi valori di probabilità si moltiplicano tra loro poiché sono indipendenti in quanto funzione della scelta casuale di e fatta da V e non nota a P. In questo caso la probabilità complessiva che P possa ingannare V è $1/2^k$. Se invece P possiede la capacità asserita risponderà sempre esattamente e tutte le operazioni andranno a buon fine per qualunque valore k scelto da V.

I principi generali su cui si basano i protocolli zero-knowledge sono i seguenti. Anzitutto i due interlocutori possono essere **onesti**, cioè sia P che V seguono esattamente

il protocollo concordato e P possiede effettivamente la conoscenza affermata; o **disonesti** se qualcuna di queste caratteristiche è violata. Per definizione ogni protocollo zero-knowledge deve possedere le tre proprietà:

Completezza: se l'affermazione di P è vera, V ne accetta sempre la dimostrazione.

Correttezza: se l'affermazione di P è falsa (P è disonesto), V può essere convinto della veridicità di tale affermazione solo con "bassa probabilità", cioè una probabilità $\leq 1/2^k$ per un valore arbitrario k scelto da lui stesso. Alternativamente la correttezza può essere riformulata dicendo che se P è onesto sarà in grado di convincerne V con probabilità $\geq 1 - 1/2^k$.

Conoscenza-Zero: Se l'affermazione di P è vera nessun verificatore V, anche se disonesto (nell'uso del protocollo), può acquisire alcuna informazione su questo fatto salvo la sua veridicità.

Per quanto riguarda l'esempio della spiaggia pensiamo che le tre proprietà appaiano intuitivamente verificate senza darne una dimostrazione formale; in genere la proprietà più difficile da dimostrare è la terza.

I primi articoli nei quali fu introdotto il concetto di zero-knowledge trattavano problemi della classe NP, in particolare su grafi. Vediamo però quale contributo sostanziale abbia dato in seguito questo approccio ai protocolli crittografici di identificazione. Un prover P - nella pratica, spesso un singolo utente - deve dimostrare la propria identità a un verifier V - di solito un'organizzazione; e idealmente non deve svelare altro che la sua identità né a V né a possibili intrusi. Ci limiteremo qui a una semplice introduzione presentando il primo protocollo di identificazione zero-knowledge proposto da Fiat e Shamir, che costituisce uno schema generale per tutti i protocolli successivi.

Ricordiamo anzitutto che l'identificazione può essere ottenuta mediante una password su un canale sicuro, o mediante un cifrario asimmetrico su un canale insicuro (paragrafo 9.2). Nel secondo caso, cui ci riferiamo ora, il metodo può essere messo in pericolo dal fatto che V chiede a P di applicare la sua chiave privata a una sequenza r che V stesso ha generato onde controllare l'identità di P con la chiave pubblica di questo. Come abbiamo visto un verifier disonesto potrebbe generare una r ad arte per carpire qualche informazione sulla chiave privata di P. Consideriamo dunque un protocollo zero-knowledge con cui P dimostra la sua identità senza svelare alcun'altra informazione: esso è basato sulla difficoltà di invertire la funzione potenza nell'algebra modulare, ovvero di calcolare la radice in modulo (paragrafo 8.2). Nel caso presente si tratta della radice quadrata: data l'equazione $t = s^2 \bmod n$, con n non primo, è polinomialmente facile calcolare t dati s ed n; ma è esponenzialmente difficile calcolare s (radice quadrata di t) dati t ed n se non si conosce la fattorizzazione di n.

Il **protocollo di identificazione di Fiat-Shamir** nella sua forma base è il seguente. P sceglie $n = pq$ con p, q primi, e un intero positivo $s < n$. Calcola $t = s^2 \texttt{ mod } n$. Rende nota la coppia $<n, t>$ come una sorta di chiave pubblica: il suo segreto è la terna $<p, q, s>$ e nessuno di questi valori può essere desunto in tempo polinomiale da $<n, t>$. Quindi può partire il protocollo i cui passi, iterati per k volte, consentono a V di stabilire l'identità di P con probabilità $1 - 1/2^k$ senza nulla poter inferire sul segreto di P.

1. V **chiede** a P di iniziare una iterazione;
2. P **genera** un intero random $r < n$;
   calcola $u = r^2 \texttt{ mod } n$;
   comunica u a V;
   || commento: si genera un nuovo valore di r e quindi di u a ogni iterazione: il valore di r può esere "bruciato" al passo 4
3. V **genera** un intero random e in $\{0, 1\}$;
   comunica e a P;
4. P **calcola** $z = rs^e \texttt{ mod } n$;
   comunica z a V;
   || commento: se $e = 0$ si ha $z = r$; se $e = 1$ si ha $z = rs \texttt{ mod } n$
5. V **calcola** $x = z^2 \texttt{ mod } n$;
   if $(x = ut^e \texttt{ mod } n)$ ripete dal passo 1
   else blocca il protocollo senza identificare P.

Le seguenti dimostrazioni di completezza e correttezza sono semplici; la dimostrazione di conoscenza-zero è molto più complicata ed è presentata solo in schema. Riflettendo sulla dimostrazione di correttezza ci si potrà rendere conto della necessità di utilizzare l'intero random e.

1. **Completezza**. Se P è onesto la condizione $x = ut^e \texttt{ mod } n$ del passo 5 è sempre verificata: infatti per $e = 0$ si ha $x = r^2 \texttt{ mod } n$ e $ut^e \texttt{ mod } n = r^2 \texttt{ mod } n$; per $e = 1$ si ha $x = r^2 s^2 \texttt{ mod } n$ e $ut^e \texttt{ mod } n = r^2 s^2 \texttt{ mod } n$. Dunque V accetta l'asserzione di P per qualunque valore di k.

2. **Correttezza**. Se P è disonesto (cioè intende farsi identificare come un altro utente) e conosce il valore di t che è pubblico ma non conosce il valore di s, può tentare di ingannare V prevedendo i valori di e che questi genererà al passo 3 del protocollo. Così, al passo 2, P invierà a V il valore $u = r^2 \texttt{ mod } n$ se prevede di ricevere $e = 0$, o il valore $u = r^2/t \texttt{ mod } n$ se prevede di ricevere $e = 1$. In entrambi i casi invierà poi a V lo stesso valore $z = r$ al passo 4. Se la previsione di P su e è corretta, al passo

5, per entrambi i valori di e, V scoprirà che $x = ut^e \texttt{ mod } n = r^2 \texttt{ mod } n$ e accetterà l'iterazione; ma se la previsione è sbagliata la relazione non sarà verificata e sarà scoperto l'inganno. La proprietà di correttezza segue dall'osservazione che, poiché e viene generato a caso, le previsioni di P su e sono corrette con probabilità 1/2.

3. **Conoscenza-Zero**. La dimostrazione che V non scopre nulla su P salvo la sua identità è molto complessa. Come per tutte le dimostrazioni di conoscenza-zero essa si basa sull'esistenza di un *simulatore (Macchina di Turing) probabilistico* disponibile a qualunque verifier onesto o meno, che, senza accesso al prover, può costruire una falsa sequenza di iterazioni tra P e V che con alta probabilità coincide con l'esecuzione di un protocollo corretto di accettazione. Il non accesso a P garantisce che V non possa acquisire alcuna informazione su di esso.

Notiamo infine un'altra caratteristica interessante dei protocolli zero-knowledge. Dopo che un onesto V ha terminato i suoi scambi con un onesto P, non può ripetere a un terzo attore T (per esempio a un giudice) la dimostrazione di veridicità dell'asserzione di P. In effetti V potrebbe eseguire una registrazione di tutti i passi del protocollo e mostrarla a T; ma questa registrazione potrebbe essere stata costruita ad arte da un disonesto V senza alcuna interazione con P, simulando il funzionamente di un protocollo truccato ove P conosce a priori le scelte (ora non) random di V e si comporta in conseguenza. Nell'esempio di identificazione sopra descritto V potrebbe costruire una falsa registrazione in cui P conosce a priori i valori di e.

Capitolo 10

Due importanti applicazioni

Quanto abbiamo illustrato fino a questo punto dovrebbe aver posto basi sufficienti a comprendere, almeno in linea di principio, il ruolo e l'utilizzazione della crittografia nella società di oggi. Per rendere più concreto questo salto fuori dalla teoria discuteremo due applicazioni che si basano crucialmente sugli algoritmi crittografici senza i quali non potrebbero esser nate. Le abbiamo scelte entrambe nel campo del denaro, cui tutti sembrano particolarmente affezionati, ma sono molto diverse una dall'altra: infatti la prima applicazione riguarda la struttura e l'uso delle carte elettroniche familiari a tutti, come per esempio le carte bancomat o le *SIM* (*Subscriber Identity Module*) dei telefoni cellulari; la seconda è relativa al mondo immateriale della moneta elettronica, cioè quella forma di denaro virtuale che esiste soltanto nella rete Internet sotto forma di segnali elettrici e ha ancora un uso limitato (*Bitcoin* ne è il tipo più diffuso). Ma è facile comprendere come la discussione che segue possa fare da schema a moltissime altre applicazioni in cui è richiesto un alto livello di protezione dei dati.

10.1 Le *smart card*

Il nome generico di *smart card* (cioè carta intelligente, ma la traduzione italiana non è usata) è riservato a quelle tessere di plastica dotate di microprocessore e memoria a stato solido che affollano le nostre giornate assieme ai relativi *PIN* (*Personal Identification Number*). Sono la versione evoluta delle carte dotate solo di banda magnetica per la memorizzazione di dati sensibili: mentre queste carte demandano ogni elaborazione agli apparecchi di lettura come i terminali *ATM* (*Automatic Teller Machine*, noti anche come *bancomat* dal nome del primo sistema italiano di carte bancarie), o i dispositivi *POS* (*Point of Sale*) degli esercizi commerciali, il chip contenuto nella smart card possiede ed esercita notevoli capacità di calcolo.

La storia delle smart card è lunga e confusa. Vi sono state all'inizio proposte di realizzazione e brevetti di valore locale, finché nei primi anni Ottanta dello scorso secolo le smart card entrarono in produzione in Francia come carte telefoniche prepagate, evolvendosi fino a riguardare oggi un grandissimo numero di applicazioni. Ciò che può anzitutto stupire è che queste carte, nate e diffuse in Europa, hanno poi trovato impiego praticamente in tutto il mondo tranne che negli Stati Uniti che conservano le vecchie carte a banda magnetica considerandole sufficientemente sicure, e cominciano solo oggi (2014) ad accettare un uso limitato delle smart card come carte bancarie e di credito. Ed è indicativo in proposito che le smart card si stiano diffondendo molto rapidamente in paesi in via di sviluppo nei quali la sicurezza è più a rischio.

Dal punto di vista fisico le smart card hanno raggiunto uno standard internazionale che sembra ormai immutabile. La tessera è di plastica rigida con dimensioni in millimetri di 86×54 (o 25×15 nelle *SIM*), e spessore 0.76. Il microprocessore e la memoria sono integrate in un'areola di circa 1 cm^2 in posizione fissa sulla carta, ricoperti da uno strato metallizzato di contatti per collegarsi al lettore (*contact smart card*, le più comuni); o con un'antenna interna per un accoppiamento a radiofrequenza col lettore (*contactless smart card*); o in alcuni casi con entrambe le funzionalità. In tutti i casi le carte sono *passive*, cioè l'energia elettrica per il funzionamento dei loro circuiti è fornita dal lettore. Le areole di collegamento elettrico delle carte a contatto hanno un bel colore dorato che non si altera nel tempo perché sono placcate con uno strato sottilissimo d'oro per evitare che si ossidino. Ma non è questo il motivo principale del costo per la loro produzione che può essere valutato in due o più Euro contro qualche centesimo per una carta a banda magnetica: questi valori dipendono da molti fattori che differenziano un carta dall'altra.

Prima di studiarne le potenzialità crittografiche ricordiamo infine che tutte le smart card conservano una banda magnetica che duplica alcuni dati della carta per poter essere usate anche su lettori non abilitati al collegamento con un microprocessore (così per esempio una carta di credito europea può in genere essere impiegata su lettori americani e viceversa). Alcune smart card prevedono inoltre altri campi, come un campo per la firma fisica o un ologramma per riconoscerne l'autenticità.[1]

[1]Un esempio un po' infelice riguarda la carta d'identità elettronica rilasciata da alcuni comuni italiani, primo in ordine di tempo il comune di Pisa. Partita da un progetto innovativo molto interessante la carta ha adottato una banda a lettura laser anziché magnetica che non è poi entrata negli standard, nonché un ologramma di riconoscimento. Tutto questo ha presumibilmente fatto levitare il costo di produzione delle carte e dei terminali dedicati. Un progetto ancora da confermare è quello di fondere in un documento unico la carta d'identità e la tessera sanitaria nazionale che attualmente rispetta uno standard europeo per essere accettata in tutti i paesi dell'Unione.

10.1.1 I protocolli d'impiego

Le smart card vivono in simbiosi con gli apparati che interagiscono con esse, in assenza dei quali non esisterebbero. E poiché vi sono carte e apparati di varia natura, ci limiteremo qui alle carte di credito e alle carte di debito (cioè le usuali carte bancomat) che sono le più largamente usate, e ai relativi terminali *ATM* o *POS*.[2]

La prima considerazione è che la maggior parte dei terminali sono inseriti in una rete telematica che connette le banche nazionali (in Italia la *Rete Interbancaria*), e queste sono a loro volta inserite nella rete internazionale *SWIFT* (*Society for Worldwide Financial Telecommunications*). Questa rete, fondata in Belgio nel lontano 1973 come rete indipendente, è traslocata su Internet dal 2001 ed è stata oggetto di accese polemiche nel 2013 per il controllo che la *National Security Agency* degli Stati Uniti evidentemente esercita su di essa. Utilizzando *SWIFT*, in linea di principio, una carta bancaria italiana può essere accettata da un terminale *ATM* in Giappone dopo una verifica via rete dei fondi disponibili. Ma come vedremo le cose non sono così semplici.

Vediamo com'è organizzato il trattamento delle carte bancarie: le carte di credito sono trattate in modo simile. Come sappiamo l'identificazione di chi presenta la carta avviene attraverso un *PIN* costituito da quattro cifre decimali: una sequenza molto corta, ma che non consente attacchi esaustivi dal terminale perché dopo tre (o cinque) tentativi di identificazione con un *PIN* scorretto la carta viene bloccata. La generazione del *PIN* è effettuata da un ente terzo rispetto alla banca che fornisce la carta; essa consiste, secondo uno degli standard in uso, nelle seguenti operazioni che impiegano il *Triple DES* per la cifratura con una chiave k_P prefissata:

Generazione e distribuzione del *PIN*. Quattro cifre decimali.

1. **generazione** di un numero casuale di quattro cifre, utilizzato come *PIN*;
2. **concatenazione** del *PIN* con alcuni dati relativi al conto bancario dell'utente per formare una sequenza complessiva S;
3. **generazione** della sequenza cifrata $T = \mathcal{C}(S, k_P)$, ove $\mathcal{C}$ è la funzione di cifratura;
4. **estrazione** della sottosequenza O_P formata dalle prime quattro cifre decimali di T, detta *Offset* del *PIN*;
5. **comunicazione postale** del *PIN* al cliente e comunicazione di O_P alla banca;
6. **rimozione** dalla memoria di ogni traccia del *PIN* originale.

[2] Naturalmente anche le carte *SIM* sono estremamente diffuse: il loro funzionamento interno non differisce in modo sostanziale da quello delle altre carte.

A questo punto solo il cliente conosce il suo *PIN*, che non può essere ricostruito dall'Offset O_P poiché il processo di generazione non è invertibile se non per enumerazione completa. Neanche la banca conosce il *PIN* per evitare al cliente un attacco dall'interno della banca stessa. Il processo di identificazione del cliente al terminale sarà effettuato con la ricostruzione e il confronto in hardware dell'Offset a partire dal *PIN* fornito dal cliente stesso, similmente a quanto visto nel paragrafo 9.2 per i sistemi Unix. E tutto questo avviene sia che la carta sia dotata solo di banda magnetica o comunque il terminale non possa interagire col chip della carta, sia che siano impiegate le piene potenzialità della smart card. Ma le operazioni eseguite sono diverse.

Nelle operazioni con la banda magnetica un ulteriore codice identificativo *CVV* (*Card Verification Value*) viene calcolato dalla banca in modo simile al *PIN*, estraendo alcuni caratteri dalla sequenza ottenuta cifrando il numero di conto del cliente con una chiave locale: il *CVV* viene memorizzato nella banda magnetica della carta ed è utilizzato nelle transazioni per verificare la validità della carta stessa (in particolare, che la carta non sia stata costruita illegalmente, perché questo richiederebbe di conoscere la chiave usata per costruire il *CVV*). Una transazione con la banda magnetica, per esempio il prelievo di contanti da un terminale *ATM*, può avvenire secondo il seguente protocollo qui descritto in modo semplificato:

Protocollo 1: prelievo da terminale *ATM*. Carta con banda magnetica presentata a un terminale della banca che ha emesso la carta.

Cliente: inserisce la carta nel terminale;

Terminale: legge e memorizza la banda magnetica; richiede il *PIN*;

Cliente: digita il *PIN*;

Terminale: carica il *PIN* in uno speciale modulo hardware; cifra il numero della carta, il *PIN* e il *CVV* con un cifrario simmetrico e spedisce il crittogramma `C` alla sua banca;

Banca: utilizzando `C` decifra il *PIN* e ne calcola l'Offset (questa operazione è eseguita in hardware per non svelare il *PIN*); confronta il valore calcolato con quello dell'Offset del cliente contenuto nella sua base di dati; calcola il *CVV* del cliente e lo confronta con il valore di *CVV* contenuto in `C` per verificare la validità della carta; se entrambi i test sono positivi risponde con l'OK alla transazione specificando la massima cifra erogabile;

Terminale: invita il cliente a proseguire la transazione;

Cliente: digita la cifra richiesta;

Terminale: se la cifra non supera il valore consentito eroga i contanti.

Questo protocollo è uno tra quelli adottati dalle diverse banche che hanno ampia libertà di scelta in merito. La connessione tra terminale e banca avviene attraverso un *server ATM* installato presso la banca stessa, che ha il compito di verificare il *PIN* e presiede alle comunicazioni tra terminale e calcolatore centrale. Più precisamente la verifica del *PIN* è eseguita in hardware da un *modulo PIN* che possiede tutti gli Offset delle carte emesse dalla banca.

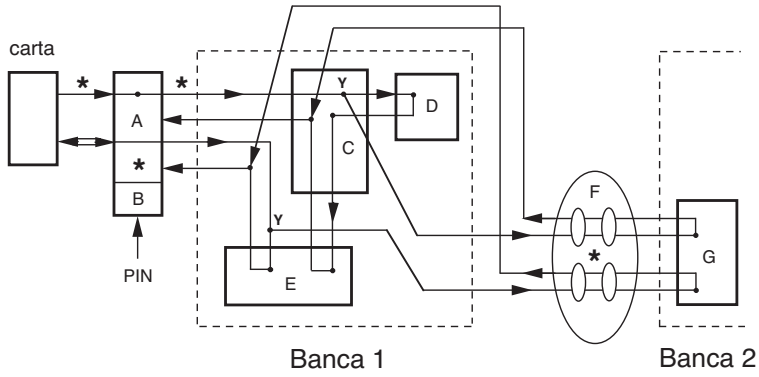


Figura 10.1: Flussi d'informazione nei Protocolli 1 e 2. Le lettere indicano: A terminale Banca 1 - B tastiera - C server *ATM* - D modulo *PIN* - E calcolatore centrale Banca 1 - F rete *SWIFT* - G calcolatore centrale Banca 2. Gli asterischi indicano i punti di maggiore vulnerabilità del sistema.

Il protocollo è più complicato se il terminale *ATM* non appartiene alla banca che ha emesso la carta: in questo caso il server *ATM* della banca proprietaria del terminale si connette via rete alla banca emittente con un cifrario simmetrico affinché questa autorizzi la transazione, seguendo poi il percorso inverso per la comunicazione della risposta. Poiché gli standard consentono a banche diverse di utilizzare chiavi e cifrari diversi (per esempio il *Triple DES* a tre chiavi), il crittogramma è soggetto a decrittazione e nuova cifratura nei nodi traversati nella rete: con evidente diminuzione della sicurezza, soprattutto se i sistemi di transito sono poco protetti. Complessivamente il flusso d'informazione è indicato nella figura 10.1 ove Y indica il punto di diramazione nel caso di accesso a un'altra banca (il percorso che inizia con una doppia freccia tra carta e terminale è relativo a uno schema per smart card che vedremo sotto). Una comune variazione del protocollo consente al terminale di accettare transazioni per cifre modeste senza il controllo preventivo della banca. Ciò avviene quando si usano

le carte bancarie su terminali remoti per pagare pedaggi autostradali e simili, e sono soggette al rischio che la carta sia falsa o il cliente non abbia i fondi necessari.

Stabiliti i concetti di base che regolano le carte a banda magnetica esaminiamo ora il funzionamento delle smart card, ricordando che anch'esse hanno la banda e possono quindi lavorare con i protocolli precedenti. Nella versione più generale il microprocessore inserito nella carta è progettato per eseguire operazioni crittografiche di tipo standard, in particolare relative a cifrari simmetrici (*Triple DES*) e asimmetrici (*RSA*), e calcolare funzioni hash one-way (*SHA1*). La grande versatilità che si ottiene è utilizzata per permettere alla carta di eseguire diversi protocolli standard, o meglio che seguono uno standard che lascia però molta libertà nelle realizzazioni, tanto che le operazioni di fatto eseguite spesso non sono state rivelate in dettaglio dalle banche. Su queste basi, nei primi anni 2000 è stato proposto lo standard globale *EMV* (*Europay MasterCard Visa*) per l'impiego compatibile di carte di credito e carte bancarie nei terminali *ATM* e *POS*. A questo standard hanno aderito tutte le banche europee e quelle di moltissimi altri paesi, con l'importante eccezione degli Stati Uniti ove, come già detto, solo nel 2014 le smart card abbinate ad *EMV* hanno cominciato lentamente a farsi strada.

La carta e il terminale possono operare con uno tra diversi protocolli di validazione e di scambio, scelto in un contatto preliminare tra i due. Consideriamo il più generale in cui i contatti tra carta e terminale avvengono mediante un cifrario a chiave pubblica disponibile a entrambi: i suoi passi possono ricordare quelli del protocollo *SSL* illustrato nel capitolo 9. Si noti che anche il terminale deve avere potenzialità ben maggiori di quelle per interagire con una carta a banda magnetica. Oltre al software crittografico la carta contiene in memoria la sua chiave privata $k_C[prv]$ e un certificato *CERT* firmato dalla banca che specifica la sua chiave pubblica $k_C[pub]$, l'Offset del *PIN* e un codice identificativo della carta. Indicando con $\mathcal{C}$ e $\mathcal{D}$ le funzioni di cifratura e decifrazione del cifrario asimmetrico abbiamo in estrema sintesi:

Protocollo 2: prelievo da terminale *ATM*. Smart card presentata a un terminale qualsiasi.

Cliente: inserisce la carta nel terminale;

Terminale: 1) legge il certificato *CERT* e ne verifica la firma con la chiave pubblica della banca; 2) controlla localmente la validità della carta attraverso il suo codice identificativo; 3) concorda con la carta le modalità di scambio dei dati per la transazione, in particolare se lo scambio deve avvenire online od offline (poniamo di essere nel secondo caso); 4) richiede il *PIN* al cliente;

Cliente: digita il *PIN*;

Terminale: 1) calcola in hardware l'Offset del *PIN* e lo confronta con quello contenuto in *CERT*; 2) spedisce il *PIN* alla carta che fa lo stesso controllo; 3) invita il cliente a proseguire la transazione;

Cliente: digita la cifra richiesta;

Terminale: costruisce una sequenza T con tutti i dati della transazione fin qui scambiati, inclusi ora e data, e l'invia alla carta;

Carta: firma la T con la sua chiave privata come $F = \mathcal{D}(h(T), k_C[prv])$ ove h è una funzione hash one-way concordata, e spedisce F al terminale;

Terminale: 1) verifica la firma controllando che sia $\mathcal{C}(F, k_C[pub]) = h(T)$;[3] 2) se la cifra richiesta non supera un ammontare prestabilito, eroga i contanti e comunica successivamente la transazione online alla banca, altrimenti chiede preventivamente l'autorizzazione alla banca come per le carte a banda magnetica.

La figura 10.1 indica il flusso d'informazione relativo a questo protocollo, iniziando con la doppia freccia tra carta e terminale ove si svolge l'intenso scambio di messaggi descritto sopra. Il Protocollo 2 realizza uno degli schemi più sicuri: in particolare non tutte le carte, e soprattutto non tutti i terminali, consentono di utilizzare cifrari asimmetrici. Per questo le carte possono anche scambiare informazioni col terminale tramite un cifrario simmetrico di cui possiedono la chiave al loro interno, assieme a una chiave per controllare il *MAC* della trasmissione (paragrafo 9.3). E naturalmente sono abilitate ad eseguire un protocollo adatto a questo funzionamento.

10.1.2 Problemi di sicurezza

Come in tutti i sistemi che impiegano la crittografia gli attacchi alle smart card sfruttano la debolezza delle linee di trasmissione e degli apparati ivi connessi più che tentare di "rompere" i cifrari che sono ormai consolidati. La peculiarità degli attacchi in questo campo deriva da situazioni di rischio che non si presentano in applicazioni finanziarie più importanti, dal movimento di ingenti quantità di fondi tra banche fino alle transazioni con carte di credito su Internet, tutte protette dai cifrari ibridi. E in effetti, benché ciascuno sia relativo a un piccolo importo di danaro, gli attacchi portati con successo alle transazioni su terminali *POS* e *ATM* sono molto frequenti e hanno determinato nel loro complesso una notevole turbativa all'attività delle banche e ancor più dei loro clienti.

[3] Questo è un controllo forte della validità della carta e della presenza al terminale del suo proprietario

Anzitutto la popolare espressine "carta clonata" non si riferisce quasi mai alla costruzione fisica della copia di una carta legittima ma al furto e successivo impiego dei suoi dati. In effetti la duplicazione fisica della carta è relativamente semplice per i sistemi a banda magnetica perché, rubato il contenuto della banda, si può costruire un'altra carta con la stessa banda utilizzando banali attrezzature elettroniche; ma costruire una smart card che duplichi il chip contenuto in un'altra è sostanzialmente impossibile con mezzi sufficientemente economici per giustificare il vantaggio che se ne potrebbe trarre. Questo comunque apre il campo, in cui finora non c'eravamo imbattuti, degli attacchi compiuti non solo con software pirata ma con la costruzione e l'impiego di circuiti dedicati.

I principali attacchi fisici diretti richiedono che la smart card sia rubata e se ne conosca lo schema elettrico interno. Un attacco è basato su un'analisi dei ritardi e della potenza elettrica assorbita dalla carta quando esegue diverse operazioni crittografiche, da cui in molti casi si possono dedurre per esempio la chiave privata di *RSA* e la chiave simmetrica del *MAC* contenute nella carta. Un attacco più drastico, detto *disassembly*, è realizzabile solo da esperti di laboratorio e consiste nell'abrasione dello strato di materiale che protegge il chip al punto da poter accedere al circuito e copiare il certificato e le chiavi contenute. Il primo attacco ha valore relativo perché richiede che il proprietario legittimo non si sia accorto in breve tempo di essere stato derubato bloccando la carta. Il secondo ha valore dimostrativo delle possibilità di attacco più che valore pratico, e inoltre la carta viene distrutta nel procedimento. In entrambi i casi vengono però compromesse le chiavi simmetriche comuni a tutte le carte di quella serie se tali chiavi non possono essere cambiate. Tuttavia, come ora vedremo, gli attacchi più comuni e pericolosi vengono svolti manomettendo i terminali e infiltrando le linee di trasmissione.

Uno dei principali punti di debolezza è che i terminali *ATM* non sono sorvegliati e anche i *POS* sono spesso utilizzati in luoghi dove sono facili le intrusioni. A parte la banale manomissione della tastiera o addirittura l'installazione di una telecamera nascosta nei pressi del terminale per leggere il *PIN* mentre viene inserito, di cui in genere ci si può rendere facilmente conto, gli attacchi fisici ai terminali sono stati condotti in modi piuttosto sofisticati. Senza entrare in una casistica senza fine ricordiamo che essi consistono sempre nel collegare al terminale un circuito spia che registra i dati in elaborazione. Particolarmente attaccata è la lettura della banda magnetica, ove utilizzata.[4]

[4]Uno standard in via di abbandono prevede di inserire l'Offset del *PIN* nella banda della carta per permettere al terminale di controllare direttamente la validità di questo senza ricorrere al modulo *PIN*. In questo caso rubare dal terminale la lettura della banda compromette irrimediabilmente la carta perché se ne può immediatamente ricostrire anche il *PIN*: se infatti inserire da tastiera un *PIN* per tentativi è impossibile perché dopo tre errori la carta viene bloccata, una volta conosciuto

Tra le tante manomissioni degli *ATM* ne ricordiamo una per illustrare "in che razza di mondo viviamo". Nel 2008 centinaia di terminali costruiti in Cina per conto di banche Inglesi e di diversi altri paesi del nord Europa risultarono dotati di un addizionale circuito spia al loro interno che registrava i dati delle carte bancarie e di credito e i relativi *PIN* e li spediva automaticamente a un'associazione di delinquenti in Pakistan mediante una connessione di telefonia cellulare. Tale circuito era stato inserito all'interno dei terminali prima dell'imballaggio e della spedizione: se non (sperabilmente) a opera della ditta costruttrice, con ogni evidenza a opera di suoi impiegati corrotti. Il danno prodotto da questa manomissione non è stato esattamente quantificato ma fu sicuramente enorme, anche perché l'origine della truffa fu scoperta quasi un anno dopo l'entrata in funzione dei terminali.

Le operazioni piratesche sono però affiancate da studi di università e di piccole ditte altamente specializzate che cercano di compromettere i terminali per dimostrarne i pericoli d'uso e le contromisure da prendere: tra tutti uno studio del 2010 dell'Università di Cambridge ha dimostrato che si può aggirare il *PIN* di una smart card rubata aggiungendo al terminale un circuito piuttosto semplice che gli consente di accettare come *PIN* qualsiasi sequenza di quattro cifre decimali. Per evitare tutto questo è probabile che la soluzione dei problemi che riguardano i *PIN* consisterà nel dotare smart card e terminali di elementi di riconoscimento "biologici" come ad esempio le impronte digitali già presenti in molti documenti d'identità: in particolare questo approccio è stato già adottato nei terminali che accettano moneta elettronica, di cui parleremo nel prossimo paragrafo.

Per quanto riguarda gli attacchi alle comunicazioni tra parti del sistema, possono essere a rischio quelle tra un terminale e la sua banca, ma tutte le operazioni interne a questa sono considerate sicure a meno che un impiegato della banca partecipi all'illecito: e anche in questo caso gli attacchi non sono semplici se il sistema informatico è ben costruito. Sono invece a rischio le comunicazioni tra diverse banche perché gli standard, incluso *EMV*, consentono alle banche stesse di usare codici crittografici propri, e questo implica la possibile decrittazione e successiva cifratura dei dati nei nodi della rete: i quali sono in realtà altre banche e possono avere standard di sicurezza non elevati e comunque non noti all'utente.[5] Questo problema ha apparentemente causato frequenti distrazione di fondi, tanto che le transazioni internazionali sono spesso limitate a cifre modeste per richieste provenienti da paesi ritenuti a rischio. A partire dal primo gennaio 2005 le banche dell'Unione Europea declinano ogni re-

l'Offset si può ricercare fuori linea un *PIN* corrispondente effettuando a programma un massimo di $10^4 = 10.000$ prove, tante quanti sono i *PIN* possibili.

[5] In effetti, come in ogni connessione attraverso Internet, l'utente non sa neanche quali nodi sono attraversati dai suoi messaggi.

sponsabilità sulla perdita di fondi se le ditte che consentono transazioni con smart card non seguono il protocollo *EMV*. Un discorso diverso riguarda i terminali *POS* che sono in genere installati in locali scarsamente protetti e sono soggetti ad attacchi sulle linee che li connettono al server a opera di estranei di passaggio, come addetti alla manutenzione o alle pulizie notturne (ovviamente dietro l'incarico di esperti). In questi casi il livello di rischio può non essere trascurabile.

Un'ultima considerazione riguarda le chiavi dei codici simmetrici utilizzati nel corso di una transazione. Come sappiamo la forza di tutti i cifrari consiste nella segretezza della chiave e nella possibilità di cambiarla se la circostanza lo richiede. Se la smart card e il terminale lo consentono, le chiavi simmetriche di scambio dati tra i due potrebbero essere ristabilite a ogni nuova utilizzazione della carta. Poiché però il costo di produzione di una smart card cresce notevolmente con la potenza del microprocessore ivi contenuto, non è detto che molte funzionalità crittografiche siano abilitate al suo interno (e ovviamente il cliente non ne è mai informato). La frequente sostituzione della chiave dovrebbe aver luogo anche nelle comunicazioni tra terminale e server e in quelle tra le diverse banche, cosa molto più semplice per la potenza di calcolo disponibile. Purtroppo la libertà concessa in proposito dagli standard consente di sapere solo approssimativamente cosa in realtà avviene perché molte scelte sono riservate.

10.2 La moneta elettronica

La moneta è cosa antichissima ed è apparsa indipendentemente in diverse parti del mondo. In termini moderni è un mezzo per custodire ricchezza di qualunque entità o per scambiare beni, e costituisce un'unità di riferimento di valore.[6] Già da secoli, comunque, la moneta ha perso a volte la sua identità fisica prendendo per esempio la forma delle lettere di credito dei banchieri senesi e fiorentini, o di assegni o cambiali, per giungere oggi all'immateriale scambio elettronico con le carte di credito. In aggiunta a questo, la diffusione di Internet potrebbe determinare un cambiamento epocale attraverso una nuova entità che va sotto i nomi di *moneta elettronica*, o *digitale*, o *virtuale*, o anche *crittomoneta*. Alla data di compilazione di questo testo (2014) è prematuro stabilirne il futuro, ma è opportuno discuterne qui per il legame strettissimo che questa forma di moneta ha con la crittografia.

Attualmente la moneta elettronica di gran lunga più importante nelle applicazioni è *bitcoin* e a essa ci riferiamo qui: senza entrare nel dibattito economico sulla natura di bitcoin come moneta o piuttosto come "protocollo di pagamento", ma spiegando

[6] Questa definizione è di certo molto rozza: non abbiamo necessità di addentrarci in dettagli lasciandone la disquisizione agli economisti.

nella sostanza tecnica di cosa si tratta. E poiché non ci sembra onesto tacere su alcuni aspetti "politici" dell'impiego di questo mezzo, ne tratteremo brevemente in fondo al capitolo rimandando gli approfondimenti ad altre fonti. Anche perché, trattandosi di creazione e di scambio di valore completamente avulso dal controllo dei governi, delle banche centrali e di altri istituti finanziari, tali aspetti possono avere una grande importanza.

Iniziamo con una brevissima storia di questa moneta. *Bitcoin* (in acronimo BTC, indicato in simbolo come una B attraversata da due barre verticali) indica una "unità monetaria" nata in un articolo pubblicato nel 2008 da Satoshi Nakamoto, un misterioso autore giapponese la cui identità è ignota tanto da indurre molti a credere che dietro al nome si celi un gruppo di ricercatori.[7] Il primo programma per la sua gestione, chiamato *Bitcoin-Qt*, fu distribuito dallo stesso autore nel 2009 come codice *open source*, per essere poi aggiornato e ormai sostituito dalla nuova versione *Bitcoin Core*. Come vedremo il software consente scambi finanziari sulla rete ma contribuisce anche a creare bitcoin in un sistema completamente decentralizzato che segue un classico approccio dei sistemi *Peer-to-Peer* (*P2P*) ove la correttezza delle transazioni può essere controllata da tutti gli utenti sia in tempo reale che in qualsiasi momento successivo. L'intero sistema è quindi soggetto solo al controllo degli utenti: basta questa caratteristica a far comprendere il grande impatto psicologico del metodo.

Le parole chiave relative al protocollo sono: *indirizzo* che consiste in un identificatore dei singoli utenti sulla rete (non è quindi un indirizzo in senso classico né un indirizzo IP); *portafogli* (*wallet*) che è l'insieme di credenziali digitali che attesta la proprietà in bitcoin di un utente; *transazione* ovvero lo scambio di bitcoin tra due utenti; *broadcast* che indica una comunicazione in rete diretta a tutti gli utenti; *libro contabile* (*ledger*), il cuore del sistema, che contiene la registrazione pubblica di tutte le transazioni eseguite nella storia di bitcoin; *blocco* e *block-chain*, cioè blocco di transazioni e catena di blocchi, ove quest'ultima costituisce il libro contabile; *mining* che indica le operazioni per il mantenimento della catena di blocchi. Spiegheremo nel seguito il significato di questi termini impiegando a volte la forma inglese che si incontra ormai in tutta la letteratura tecnico-scientifica.

Seguendo l'impostazione di Nakamoto, i passi che si eseguono sono i seguenti:

1. Le nuove transazioni sono diffuse sulla rete via broadcast.

2. Ogni nodo del sistema raccoglie le nuove transazioni in un blocco.

[7] L'articolo di Nakamoto, intitolato "Bitcoin: A Peer-to-Peer Electronic Cash System", costituisce una pietra miliare nella letteratura sui sistemi *P2P*. Il testo, facilmente reperibile su Internet, è intenso ma chiarissimo. L'impostazione riportata in questo paragrafo è basata su di esso, completata dai necessari dettagli tecnici.

3. Ogni nodo cerca di individuare una dimostrazione di correttezza per il suo blocco. Questo volutamente implica la soluzione di un "problema difficile".

4. Quando un nodo trova una dimostrazione di correttezza la diffonde per broadcast a tutti i nodi per inserire il blocco nel block-chain. Un premio in bitcoin viene accreditato al nodo autore della dimostrazione.

5. I nodi accettano il blocco solo se le transazioni in esso contente sono valide e non sono apparse in blocchi precedenti.

6. I nodi esprimono la loro accettazione del blocco iniziando a creare un nuovo blocco da inserire nel block-chain.

Vediamo ora come questi passi vengono realizzati.

10.2.1 Le transazioni bitcoin

Il software cliente di bitcoin, attualmente *Bitcoin Core*, è caricato sul PC o sullo smartphone di ogni utente A e genera anzitutto una coppia di chiavi privata-pubblica $k_A[prv], k_A[pub]$ per un cifrario asimmetrico su curve ellittiche (paragrafo 8.7).[8] La chiave privata $k_A[prv]$ è ovviamente nota solo ad A e, come vedremo in seguito, è utilizzata da esso per firmare le transazioni che genera e diffonde sulla rete. La chiave pubblica $k_A[pub]$ è utilizzata per controllare la firma di A (paragrafo 9.4) ed è anche impiegata come suo identificatore: a tale scopo viene trasformata attraverso applicazioni ripetute della funzione hash *SHA-256* (immagine a 256 bit), per essere poi compressa via *RIPEMD-160* in un'immagine di 160 bit (paragrafo 9.1) in testa alla quale è aggiunta una speciale sequenza che indica che la stringa complessiva è di fatto un indirizzo bitcoin. Come già osservato questo indirizzo non corrisponde a una "locazione" del cliente ma a un modo per identificarlo, sia per potergli dirigere una transazione che per controllare che possieda effettivamente i fondi che pretende di spendere. Oltre a creare le chiavi e gli indirizzi il software cliente permette di costruire e proporre le transazioni.

Una transazione in bitcoin ha la forma "il pagatore A vuole inviare X bitcoin al ricevente B", completata dalla firma digitale di A. Si noti che A e B sono indicati attraverso i loro indirizzi bitcoin e la transazione viene inviata per broadcast a tutti gli utenti. Diversamente da ogni altra forma di transazione economica, il ricevente B non ha la garanzia che la transazione sia valida finché, come vedremo, essa non è convalidata dalla rete. In effetti B sarebbe in grado di verificare sia la firma di A

[8] Dopo una discussione tra esperti è stata scelta una curva indicata con la sigla secp256k1.

che i fondi che A ha a disposizione, poiché la chiave pubblica di A è nota e tutte le transazioni eseguite sulla rete sono pubbliche, ma non ha il tempo di verificare che A non abbia utilizzato gli stessi fondi in istanti immediatamente precedenti per pagamenti diversi.[9] Questo sistema di verifica pubblica costituisce il vero cuore del sistema bitcoin. Si noti la differenza con il pagamento con carta di credito su Internet in cui la transazione è inviata alla banca che ha emesso la carta, che ha il compito di convalidarla e di bloccare diverse richieste in successione quando i fondi concessi al cliente si esauriscono: per tale meccanismo è necessario l'intervento di un ente terzo e al di sopra agli altri (la banca, appunto), cosa che il sistema bitcoin per principio vuole evitare: anche perché, come è ovvio, quell'ente offre i suoi servizi tutt'altro che gratis. Formalmente una transazione è innescata secondo il seguente schema:

Protocollo di transazione bitcoin. Messaggio in chiaro firmato in hash.

Messaggio. L'utente A genera un messaggio $m = adr_A - X - adr_B$, ove adr_A, adr_B sono gli indirizzi bitcoin di A e B, e X è la somma da trasferire da A a B.

Firma. L'utente A calcola l'hash $h = SHA\text{-}256(m)$ del messaggio e genera la firma $f = \mathcal{D}(h, k_A[prv])$ per m.

Broadcast. La coppia $<m, h>$ viene diffusa da A sulla rete.

La verifica di validità della transazione è eseguita dalla rete che aggiorna in conseguenza il libro contabile. Il meccanismo piuttosto complesso è spiegato nel prossimo paragrafo. L'utente B attende che la rete convalidi la transazione prima di accettarla: tipicamente se B deve spedire della merce ad A in seguito al pagamento di X bitcoin, e non ha motivo particolare per fidarsi di A, dovrà per prudenza attendere la convalida del messaggio prima della spedizione. Nel funzionamento attuale del sistema una convalida richiede una decina di minuti: questo obiettivamente limita l'uso di bitcoin all'interno dei negozi ed altri istituti commerciali ove invece sarebbe molto comodo utilizzarlo sul momento, per esempio con lo smartphone dell'acquirente.

Prima di esaminare i passi di verifica è bene sottolineare che la chiave privata è **l'unico documento valido** per dimostrare la proprietà in bitcoin di un utente A. La perdita della chiave comporta la perdita della proprietà a causa dell'impossibilità di firmare transazioni, e il furto della chiave da parte di un truffatore che la usi per firmare al posto di A comporta anch'esso la perdita di proprietà: in entrambi i casi

[9] L'articolo citato di Nakamoto era proprio indirizzato al problema, detto *double-spending*, di evitare che un utente possa spendere più volte gli stessi fondi in un sistema di moneta elettronica.

non vi è alcuna possibilità di recupero. La chiave privata deve quindi essere conservata con assoluta sicurezza, possibilmente su carta o su un PC non collegato in rete per evitare attacchi.

10.2.2 Validazione tramite *mining*

Secondo lo schema originale proposto da Nakamoto ogni nodo del sistema bitcoin può partecipare al processo di validazione delle transazioni lanciate sulla rete. Poiché però tale processo richiede la soluzione enumerativa di un problema esponenziale secondo un esponente t che viene incrementato nel tempo, solo nodi con grande potenza di calcolo hanno la possibilità pratica di partecipare al processo ricevendo un premio in bitcoin in caso di successo. La grande maggioranza dei nodi si limita a lanciare le transazioni e a controllarne la validazione effettuata da altri nodi strutturati a questo proposito. Infatti mentre la dimostrazione di validazione richiede in media 2^t operazioni, il controllo di correttezza della dimostrazione può essere eseguito da tutti i nodi mediante un unico semplice calcolo.

Come detto il libro contabile consiste in una catena di blocchi (*block-chain*) ciascuno dei quali contiene una sequenza di transazioni lanciate in rete in un certo intervallo di tempo. In ogni istante il blocco corrente, ancora non inserito nella catena, contiene le ultime transazioni la cui legittimazione deve essere approvata; tutti i blocchi precedenti sono invece già stati approvati e la catena fin lì è immutabile e contiene l'intera storia di bitcoin. Il procedimento di valutazione di un blocco, oltre a controllare la legittimità delle transazioni ivi contenute, ovvero che i pagatori dispongano dei fondi necessari per ogni transazione, è associato a un'operazione detta *mining* anche se il termine ha un senso diverso da quello usualmente adottato sulla rete: non si tratta infatti di scovare elementi memorizzati da qualche parte, ma numeri, o meglio sequenze binarie dette *nonce*, con particolari proprietà in relazione a tutta la sequenza di transazioni passate. Questo procedimento sembrerà bizzarro a chi non ha dimestichezza con i sistemi *P2P*.

Un blocco B_i contiene la sequenza di transazioni relativa al suo periodo temporale, un valore hash h_{i-1} proveniente dal blocco precedente, e un nonce n_i (vedi figura 10.2). Il processo di mining consiste nella ricerca di un valore di n_i che, concatenato alla sequenza di transizioni del blocco e ad h_{i-1}, generi un'immagine hash *SHA-256* che inizia almeno con t zeri, ove t è un valore prefissato dal sistema. Poiché la funzione *SHA-256* è crittograficamente forte non è prevedibile in tempo polinomiale in t l'immagine che genererà la presenza di un nonce arbitrario: l'unica strategia possibile è quindi enumerativa, tipicamente la prova in successione di nonce consecutivi a partire da un valore arbitrario finché se ne individua uno che genera nell'hash una sequenza

iniziale di t zeri. Ciò richiede in media l'applicazione di *SHA-256* per 2^t volte mentre, ottenuto un nonce giusto, la verifica della sua proprietà richiede solo un calcolo della funzione hash.

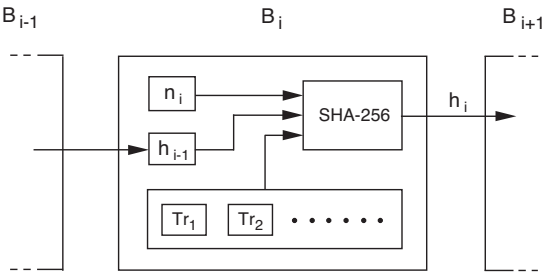


Figura 10.2: Struttura di block-chain. n_i, h_i, Tr_1, Tr_2, . . . , sono nonce, hash, e transazioni del blocco i-esimo.

Quando un nodo di mining N trova un valore giusto di nonce convalida il blocco trasmettendolo a tutti gli altri nodi; questi controllano a loro volta le transizioni ivi contenute e il nonce indicato da N e confermano sulla rete la loro accettazione del blocco che viene così inserito nel block-chain e non è più alterabile. In presenza di più blocchi convalidati contemporaneamente il sistema accetta quello che contiene il maggior numero di transazioni. Il sistema è dimensionato in modo che la validazione di un blocco avvenga circa ogni dieci minuti: poiché questo tempo è legato alla scoperta di un nonce, il sistema adegua in continuazione il valore di t in funzione del numero e della potenza di calcolo dei nodi di mining presumibilmente in azione. E poiché il numero di tali nodi e soprattutto la loro potenza di calcolo crescono costantemente nel tempo, il valore di t viene costantemente aumentato.

Questa corsa al mining dipende da una caratteristica fondamentale del sistema che premia il nodo che valida un blocco con l'accreditamento di alcuni bitcoin: il premio, di 25 bitcoin nel 2014, sarà dimezzato approssimativamente ogni quattro anni per essere definitivamente azzerato nel 2140 quando il numero complessivo di bitcoin esistenti dovrebbe raggiungere 21 milioni.[10] L'idea originale che ogni nodo possa concorrere alla validazione dei blocchi si è rivelata illusoria perché un utente con un singolo PC, anche molto potente, non ha alcuna possibilià di competere con centri dedicati che utilizzano ormai grandi sistemi paralleli composti di chip *ASIC*

[10]In tale data, per mantenere il sistema, dovrebbe entrare in uso un compenso oggi facoltativo corrisposto da chi lancia una transazione; anche se è obiettivamente difficile immaginare che nell'anno 2140 bitcoin continui ad esistere, quanto meno con i meccanismi attuali.

(*Application Specific Integrated Circuits*) per il calcolo super-rapido della funzione *SHA-256*.

Rimandando al paragrafo seguente la discussione sulla sicurezza del sistema, una considerazione finale riguarda lo spazio nella memoria dei nodi che la sempre crescente block-chain invade progressivamente. A tale proposito vengono messi in atto vari accorgimenti per strutturare la catena nella forma di albero compatto in cui sia semplice eseguire ricerche di passate transazioni che possano mantenere interesse tagliando rami molto antichi senza interrompere la catena di hash.

10.2.3 Sicurezza, anonimato e aspetti socioeconomici

Per quanto appreso in questo libro sulla sicurezza dei protocolli crittografici dovrebbe risultare chiaro che il sistema bitcoin è organizzato secondo criteri classici della crittografia. Alcune sue caratteristiche specifiche potrebbero però renderlo attaccabile. Discutiamo brevemente questi aspetti rimandando gli approfondimenti alla letteratura specializzata e ai numerosi documenti reperibili su Internet.

Consideriamo anzitutto il cifrario asimmetrico impiegato per la costruzione degli indirizzi e per la firma delle transazioni. Mentre la crittografia su curve ellittiche risulta oggi sostanzialmente inattaccabile, l'importanza di proteggere e trattare le chiavi è più che mai cruciale in bitcoin. Le chiavi pubbliche costituiscono di fatto gli indirizzi degli utenti; perciò in una transazione del tipo "A paga X bitcoin a B", l'utente A deve accertarsi dell'indirizzo di B per non dirigere il pagamento a qualcun altro senza possibilità di correzione successiva. Infatti, per motivi di principio, nel sistema bitcoin non esiste un ente tipo *Certification Authority* che garantisce l'autenticità delle chiavi. Né, per proteggere l'anonimato, esiste un registro pubblico che associa agli indirizzi bitcoin l'identità degli utenti. Questi potranno scambiarsi indirizzi bitcoin tra conoscenti come si fa per gli indirizzi e-mail, o più comunemente un esercizio commerciale potrà comunicare a un cliente il suo indirizzo per ricevere un pagamento, con tutti i problemi che possono presentarsi se ciò avviene in rete. Quanto alle chiavi private abbiamo già detto: la loro perdita comporta la perdita dei propri fondi, e il loro furto permette ad altri di utilizzarli senza possibilità di recupero. La firma delle transazioni non presenta invece problemi.

La mancanza di una certificazione esterna delle chiavi mostra anche che se il software di due utenti dovesse generare la stessa coppia di chiavi vi sarebbe inevitabilmente una commistione di fondi. Si può però dimostrare che questo evento ha una probabilità tanto piccola di verificarsi che gli utenti possono di fatto ignorare il problema. Inoltre questa remota eventualità può non essere troppo grave perché, per un motivo generale di protezione dei fondi, ogni utente è in genere invitato a crearsi

diverse coppie di chiavi e quindi diversi indirizzi. In effetti i problemi più seri sulle chiavi, come spesso avviene in crittografia, derivano da attacchi "laterali" che non sono diretti ai protocolli ma ai sistemi operativi dei calcolatori che li ospitano. L'installazione fraudolenta di *malware* in un *PC* può causare l'individuazione e il furto delle chiavi bitcoin al suo interno.

L'altra serie di attacchi, studiata sin dalla proposta iniziale di Nakamoto, consiste nell'intervenire sul mining per registrare transazioni false che accreditino fondi non dovuti, o cancellino pagamenti attraverso l'eliminazione di transizioni dai blocchi, o permettano di spendere più volte gli stessi fondi convalidando transazioni multiple. La difesa generale è insita nel criterio stesso di mining. Per avere una buona probabilità di convalidare falsamente un blocco un sistema deve avere una potenza di calcolo maggiore di quella del miner più potente: in questo caso potrebbe convenirgli di usare questa potenza per superare gli altri in un mining onesto guadagnando i bitcoin assegnati per questo lavoro.

Particolarmente serio sarebbe un intervento dell'attaccante che riguardi blocchi ormai convalidati e inseriti nella catena, per generare una catena diversa che si ponga in antitesi a quella corretta. Ma per modificare un blocco passato è necessario aggiornare anche tutti i successivi a causa della concatenazione di hash tra i blocchi della catena (vedi figura 10.2): ciò richiede di ricalcolare i nonce di tutti i blocchi interessati che è un lavoro computazionalmente enorme. L'unica possibilià concreta di attacco sembrerebbe la scoperta di una falla nella funzione SHA-256 che permetta di prevederne in parte l'immagine prodotta rendendo più veloce la generazione dei nonce. Tale falla dovrebbe però essere scoperta e tenuta segreta dall'attaccante altrimenti tutti i miner potrebbero sfruttarla riportando la competizione al punto di partenza.

Un altro aspetto di bitcoin riguarda l'anonimato degli utenti. Poiché gli indirizzi non riportano dati che permettano di risalire a individui o organizzazioni particolari l'anonimato dovrebbe essere automaticamente garantito. Molti governi sono però intervenuti obbligando i sistemi che gestiscono il servizio a rilasciare informazioni sugli utenti, in particolare su coloro che acquistano o vendono bitcoin contro valute standard: una parziale difesa dell'anonimato si realizza impiegando indirizzi multipli per ogni utente. Ma l'aspetto fondamentale di bitcoin come elemento di innovazione è legato alle sue implicazioni socioeconomiche. È questo un argomento importante e molto dibattuto su cui discuteremo solo brevemente perché non riguarda le caratteristiche tecniche del sistema.[11]

Dal 2009, anno della sua nascita, bitcoin è stato adottato da un numero sempre

[11]Tra i molti documenti disponibili consigliamo la lettura del bel rapporto di Denis Roio ("in arte", l'hacker Jaromil): Bitcoin, the end of the Taboo on Money, Dyne.org Digital Press, 2013.

crescente di utenti ed è divenuto un'icona dei movimenti politici contrari allo strapotere dei governi e degli istituti finanziari. Anche i conservatori più convinti non possono negare la ventata di aria pulita che questo comporta: e infatti la controversia tra i fautori di un controllo pubblico sulla moneta e chi sostiene il ruolo egemone delle banche centrali non si basa su questioni di principio ma si svolge per altre vie. Anzitutto la moneta elettronica, in qualunque sua forma, esiste solo come combinazione di segnali elettrici: indubbiamente un'innovazione straordinaria. In particolare i bitcoin sono creati e attribuiti come effetto della dimostrazione di aver eseguito un "duro lavoro" (il mining) che richiede grande dispendio di hardware e di energia elettrica. Quindi la creazione di bitcoin inquina l'ambiente; molto meno, però, dei sofisticati procedimenti per coniare monete e stampare banconote, e dei colossali sistemi di sicurezza per proteggerle e distribuirle. Come impatto sull'ambiente bitcoin batte le valute tradizionali senza possibilità di confronto. D'altra parte è innegabile che il mining, pur realizzazione di un'idea geniale, consiste in un lavoro estremamente banale mentre la produzione di moneta tradizionale è figlia di attività altamente qualificate, parzialmente artigianali e spesso decisamente artistiche.[12] Sotto l'aspetto della creatività la moneta tradizionale è imbattuta.

Un altro aspetto rilevante è che bitcoin ha subito nella sua breve storia incredibili fluttuazioni di valore. Partito da un cambio iniziale di pochi centesimi di dollaro USA, un bitcoin è arrivato a valere più di 1000 dollari alla fine del 2013 per valerne circa 600 durante il 2014: citare questi tassi ha però solo valore indicativo sulla *volatilità* di bitcoin perché è difficile scoprirvi una tendenza su come la variazione proceda. Il fenomeno ha comunque generato una corsa all'acquisto di bitcoin come bene di investimento ad alto rischio consegnando capitali considerevoli nelle mani di semplici speculatori che non li hanno guadagnati con il loro lavoro (di mining). Con comprensibile irritazione dei suoi onesti sostenitori, la moneta elettronica non si è dunque ancora attestata come moneta "democratica", tale da recare giustizia sociale nell'economia. Per comprendere meglio la portata del fenomeno è bene ricordare che il sistema prevede che i bitcoin possano essere scambiati liberamente con altre valute: il primo terminale *ATM* dove tali cambi possono aver luogo è stato installato in Canada nel 2013 e suoi simili si stanno lentamente diffondendo in altri paesi: primo tra tutti gli Stai Uniti dove bitcoin è stato sostanzialmente accettato nell'economia ufficiale (mentre in Cina è pesantemente osteggiato).

Un campo in cui però la moneta elettronica può intervenire per favorire l'equilibrio sociale è nelle rimesse di denaro che i migranti effettuano a vantaggio dei loro familiari in paesi lontani: rimesse al momento gestite da agenzie internazionali che si fanno

[12]Esemplari sono le diverse banconote in lire, ideate nel corso di tutto il XX secolo da famosi incisori per la Zecca italiana. Tutta un'altra classe rispetto alle banconote in euro.

pagare il servizio praticando in genere altissimi tassi di cambio.[13] Quando in Somalia o in Eritrea tutte le famiglie avranno accesso a un PC il trasferimento di danaro potrà avvenire al solo costo di una connessione Internet.

Veniamo infine all'aspetto più largamente discusso di bitcoin che riguarda l'uso che se ne può fare. Nato per lo scambio di merci e servizi, diretto poi agli investimenti, bitcoin è stato purtroppo impiegato in pesanti operazioni illegali, scoperte e largamente rese note su Web, il che lascia prevedere che altre simili siano in atto o lo saranno in futuro. Si tratta di commercio di droga e di materiale pedopornografico, di riciclaggio di denaro sporco e di altri sconci negozi obiettivamente difficili da scoprire in rete, tanto che la tendenza di tutti i governi è porre in opera misure di controllo che rischiano di limitare la libera circolazione della moneta elettronica. Al monito dei critici che affermano "di aver previsto questi effetti sin dall'inizio" si contrappone la risposta più ovvia: ogni innovazione tecnico-scientifica può essere soggetta a usi buoni e cattivi. Per fortuna, a parte certe piccolezze come la bomba atomica, sembrano prevalere i primi.

[13] In questo campo l'inizio (2014) non stato particolarmente elegante: un'agenzia finanziaria ha istituito un servizio di trasferimento di fondi dall'Europa al Kenya cambiando al suo interno gli Euro ricevuti in bitcoin, e poi questi in Scellini kenyoti da consegnare ai destinatari. Naturalmente è un'operazione a pagamento, ma il servizio è meno caro di quello offerto dalle agenzie tradizionali.

Capitolo 11

L'effetto della computazione quantistica

Il continuo studio di fenomeni fisici che permettano di eseguire calcoli in senso lato, e dei principi algoritmici su cui potrebbe basarsi la loro utilizzazione, ha diretto da decenni l'attenzione dei ricercatori verso sistemi regolati dalle leggi della meccanica quantistica: una branca della fisica molto difficile da comprendere per persone anche di alta ma non specifica cultura. Il progetto scientifico, per il quale è difficile prevedere quando e se sarà disponibile una realizzazione alla portata di tutti, è quello della costruzione di un *computer quantistico* (*quantum computer* nel linguaggio internazionale), termine che può riferirsi a macchine differenti che sfruttino i principi di quella branca della fisica. Lasciando da parte le fatue disquisizioni di moda che appaiono nei giornali o in televisione l'argomento merita la massima attenzione. In particolare per i suoi effetti in crittografia, di cui ci occupiamo qui.

Per comprendere questi effetti indicheremo in modo succinto, e senza alcuna pretesa di profondità scientifica, alcuni fenomeni della meccanica quantistica che possono intervenire nell'elaborazione dell'informazione. Nel mondo degli elementi più piccoli studiati dall'uomo, in particolare muovendosi tra atomi, particelle elementari e fotoni, la prima legge che ci interessa si chiama di *sovrapposizione* e si può riassumere come la proprietà di un sistema quantistico di trovarsi in diversi stati contemporaneamente; tuttavia, e questa è la seconda legge che ci interessa, il tentativo di osservare o misurare lo stato del sistema ne determina il collasso verso uno stato singolo, ovvero il sistema così disturbato perde la sovrapposizione di stati. Il fenomeno è detto *decoerenza*, e in effetti la manipolazione sperimentale di sistemi quantistici è estremamente difficile perchè ogni minimo disturbo può determinarla. In qualche modo legata a questo fenomeno è la *impossibilità di duplicare* un sistema conservando nella copia lo stato quantico dell'originale. Mentre tutto ciò è ragionevolmente accettabile da

chiunque abbia fiducia nell'opera dei fisici, un fenomeno davvero stupefacente detto *entanglement* riguarda la possibilità che due o più elementi si trovino in stati quantici completamente correlati tra loro in modo che, pur se trasportati a grande distanza uno dall'altro, mantengano la correlazione. Einstein, tra coloro che hanno attentamente studiato il fenomeno, lo definì una "spooky action at a distance" (potremmo tradurre: un'azione a distanza che fa rizzare i capelli).

Facciamo allora qualche esempio che ci riguarderà da vicino, per comprendere cosa ci aspetta. Mentre i sistemi di calcolo tradizionali sono basati su una rappresentazione binaria dell'informazione il cui elemento costituente è il bit, ovvero un'entità che può valere zero o uno ma che in ogni istante assume uno di tali valori, nei sistemi studiati per il calcolo quantistico l'unità è il *qbit* che si trova in sovrapposizione dei due stati. Dunque anche in questi sistemi la rappresentazione dell'informazione è binaria ma ogni qbit contiene doppia informazione rispetto a un bit. Uno degli elementi fondamentali su cui lavorano gli studiosi è il registro di n qbit ove questi possono essere realizzati con diversi componenti microscopici che abbiano caratteristiche quantistiche. In linea di principio un tale registro, invece di contenere n bit di informazione può contenerne fino a 2^n. In pratica ciò non vuol dire che si possono memorizzare in un solo registro tutte le parole di n bit contemporaneamente perché all'atto della lettura se ne otterrà comunque una sola, ma si può sperare di individuare un algoritmo che faccia convergere tutti i possibili 2^n stati del registro in uno che costituisca la soluzione di un problema in esame. Vedremo nel seguito cosa questo possa significare.

Un altro modo di usare fenomeni quantistici è legato alla polarizzazione dei fotoni nel piano. Tale polarizzazione, ridotta a due stati tra gli infiniti possibili, è impiegata per assegnare un'informazione binaria a ogni fotone che partecipa al gioco. Mentre la realizzazione di registri di qbit di dimensioni abbastanza grandi per essere utili appare ancora lontana nel tempo, l'impiego di sequenze temporali di fotoni polarizzati è molto più concreto: la prima applicazione crittografica basata su questo principio è stata infatti ideata e realizzata a partire dagli anni ottanta del secolo scorso per lo scambio sicuro di chiavi e può sostituire il protocollo *DH* o altri che hanno lo stesso scopo. Nel prossimo paragrafo ci occuperemo di questa applicazione e delle variazioni che sono seguite, che vanno spesso sotto il nome di *crittografia quantistica*. Nel paragrafo successivo discuteremo invece l'effetto che avrebbe sulla crittografia l'entrata in funzione di veri e propri computer quantistici, atti cioè a eseguire generali operazioni di calcolo e sostituire quindi i calcolatori di oggi, e delle misure da prendere: un insieme di studi che va sotto il nome di *crittografia post-quantistica*. In ogni caso è bene precisare sin da ora che, a differenza degli algoritmi deterministici classici che garantiscono soluzioni corrette, gli algoritmi quantistici producono soluzioni corrette "con alta probabilità" nel senso discusso nel capitolo 4.

11.1 Lo scambio quantistico di chiavi

Il meccanismo dello scambio di chiavi tra due utenti mediante l'invio di fotoni polarizzati è noto come *QKD*, acronimo di *Quantum Key Distribution*. Nel 1984 lo statunitense Charles Bennett e il canadese Gilles Brassard ne hanno definito il primo protocollo, noto come *BB84* dalle iniziali dei due proponenti. Numerosi studi di fattibilità del protocollo sono stati successivamente elaborati in vari laboratori di ricerca, per giungere alla prima importante realizzazione pratica nel 2001 presso il *CERN* di Ginevra ove la spedizione della chiave è stata effettuata lungo una fibra ottica standard che collega Ginevra a Losanna, per una lunghezza di 67 chilometri. Oggi le tecnologie *QKD* sono più mature e diverse compagnie offrono apparati di trasmissione e servizi in proposito, anche se molto lavoro rimane da fare. Poiché le idee principali che animano *BB84* costituiscono tuttora la base dei protocolli successivi, descriveremo anzitutto la struttura di questo protocollo spiegandone poi alcuni aspetti che ne garantiscono la sicurezza e le linee su cui si svolgono protocolli alternativi.

Per chiarire la struttura del protocollo dobbiamo anzitutto descrivere, in modo necessariamente molto approssimativo, alcuni fenomeni di meccanica quantistica e le apparecchiature che basano il loro funzionamento su tali fenomeni. Il costituente elementare della luce è il *fotone*, che può essere visto come una particella indivisibile priva di massa, che si propaga alla velocità della luce. Ogni fotone possiede una *polarizzazione* definita come il piano di oscillazione del suo campo elettrico: e in questo piano il fotone può trovarsi in diversi stati. In particolare possiamo considerare la polarizzazione verticale $\mathcal{V}$ o orizzontale $\mathcal{H}$, rappresentati graficamente con una freccia verticale o orizzontale. Sovrapponendo gli stati $\mathcal{V}$ e $\mathcal{H}$ si possono ottenere altri stati di polarizzazione tra cui $-45°$ o $+45°$, rappresentati con frecce inclinate a $\pm 45°$. Questi quattro stati saranno associati ai bit 0 e 1 per rappresentare l'informazione codificata in una sequenza di fotoni, secondo le corrispondenze:

- **bit 0**: associato a un fotone polarizzato $\mathcal{V}$ ($\uparrow$) o $+45°$ ($\nearrow$); (0)
- **bit 1**: associato a un fotone polarizzato $\mathcal{H}$ ($\rightarrow$) o $-45°$ ($\searrow$). (1)

Per realizzare il protocollo *BB84* sono necessarie tre apparecchiature di base, oltre a un mezzo di trasmissione per i fotoni che ne conservi la polarizzazione (tipicamente una fibra ottica). La prima apparecchiatura detta *one-photon gun* (*OPG*) consente di generare ed emettere un fotone alla volta con una polarizzazione prestabilita. Si tratta di un problema molto difficile perchè i laser, usuale sorgente di luce, emettono fasci luminosi con caratteristiche imposte, ma ogni fascio contiene molti fotoni. Diversi apparati one-photon gun sono stati ideati e realizzati con speciali tecnologie, ma un

loro ulteriore sviluppo sarebbe assai utile per aumentare l'affidabilità dei sistemi di distribuzione delle chiavi.

Il secondo apparecchio necessario è la *cella di Pockels* (PC) che consente di imporre la polarizzazione di un fotone agendo su un campo elettrico di controllo.

Il terzo apparecchio è il *beam splitter polarizzante* (PBS) che dirotta i fotoni in ingresso verso una tra due uscite A o R per un fenomeno ottico rispettivamente di attraversamento o riflessione. Il PBS ha un suo asse di polarizzazione $\mathcal{S}$. Un fotone in ingresso polarizzato in direzione $\mathcal{F}$ viene inviato all'uscita A con probabilità $cos^2(\theta)$, o all'uscita R con probabilità $sin^2(\theta)$, ove θ è l'angolo compreso tra $\mathcal{F}$ e $\mathcal{S}$. Nel primo caso il fotone esce da A con la polarizzazione $\mathcal{S}$ del PBS; nel secondo caso esce da R con polarizzazione ortogonale a $\mathcal{S}$. Nel caso che $\mathcal{F}$ coincida con $\mathcal{S}$ si ha $\theta = 0°$ quindi $cos^2(\theta) = 1$ e $sin^2(\theta) = 0$ e il fotone esce sicuramente da A. Nel caso che $\mathcal{F}$ e $\mathcal{S}$ siano ortogonali si ha $\theta = 90°$ quindi $cos^2(\theta) = 0$ e $sin^2(\theta) = 1$ e il fotone esce sicuramente da R. Nel caso $\theta = \pm 45°$ si ha $cos^2(\theta) = sin^2(\theta) = 1/2$: il fotone può uscire con pari probabilità da A o da R e la sua polarizzazione è conseguentemente cambiata. È questo il primo fenomeno quantistico che si utilizza nel protocollo: la lettura attraverso il PBS ha distrutto lo stato quantico precedente.

È bene notare fin da ora che inevitabili inesattezze di costruzione o allineamento degli apparati, o piccole variazioni di polarizzazione dei fotoni nella trasmissione, fanno sì che l'angolo θ non sia esattamente 0°, o 90°, o ±45°. Ciò implica che l'uscita dal PBS possa essere diversa da quella prevista dalla teoria, ma ciò avviene con probabilità molto bassa: poiché non è accettabile alcun errore nella determinazione di chiavi crittografiche dovremo prevedere un opportuno sistema di controllo e correzione. Per semplicità descriveremo il protocollo *BB84* nella sua struttura di base che non prevede errori, per spiegare nel seguito come si possa ovviare a questi.

Tra le molte proposte, una struttura circuitale per eseguire il protocollo tra i partner Alice e Bob è mostrata nella figura 11.1. Dette *basi di polarizzazione* le coppie di riferimento $\mathcal{V}$-$\mathcal{H}$ e ±45°, brevemente indicate con + e ×, Alice trasmette a Bob una sequenza S di bit sotto forma di fotoni, stabilendo a caso la base di polarizzazione per ciascuno di essi e utilizzando le associazioni (0) e (1) tra bit e polarizzazione indicate sopra: a fine procedimento la chiave sarà estratta dalla sequenza S. Alice stabilisce la polarizzazione dei fotoni controllando la cella PC posta all'ingresso della linea di trasmissione. Bob stabilisce a caso una base di polarizzazione controllando un deviatore che dirotta il fotone in arrivo su uno di due moduli PBS orientati + e ×, e *legge* il bit ricevuto a seconda che il fotone provenga dall'uscita A o R del PBS interessato. Le basi impiegate da Alice e Bob sono scorrelate e coincidono in media nella metà dei casi, stima significativa poiché la S è molto lunga: dunque solo metà dei bit di S mantengono la polarizzazione scelta da Alice dopo l'intervento di Bob e

sono letti correttamente da esso, mentre le letture relative alle basi diverse di Alice e Bob non sono significative. Alice e Bob devono ora decidere quali sono i bit corretti.

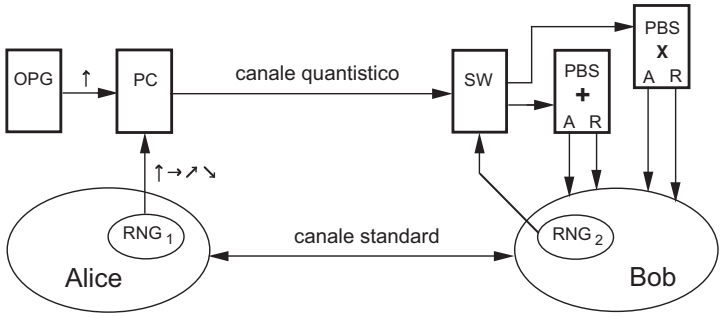


Figura 11.1: Circuito per l'esecuzione del protocollo *BB84*. Il generatore OPG emette una sequenza di fotoni polarizzati verticalmente. Agendo sulla cella PC, Alice varia la polarizzazione dei fotoni secondo le scelte di un generatore casuale RNG_1. Bob decide la base di lettura agendo su un deviatore SW, secondo le scelte di un generatore casuale RNG_2. Alice e Bob sono anche collegati con un canale autenticato standard.

Per eseguire la scelta dei bit Alice e Bob devono scambiarsi alcune informazioni addizionali per cui possono utilizzare un canale di trasmissione standard (figura 11.1), con l'unica garanzia che non sia presente un crittoanalista attivo che possa portare attacchi di tipo *man in-the-middle*, interrompendo le comunicazioni tra i due partner e comunicando con Alice fingendosi Bob, e con Bob fingendosi Alice. Per questo è sufficiente che il canale sia autenticato con un codice MAC protetto da una chiave segreta che i due partner hanno concordato in precedenza (vedi paragrafo 9.3). La trasmissione può avvenire in chiaro. Vediamo ora nello specifico la struttura del protocollo.

11.1.1 Il protocollo *BB84*: struttura di base

A grandi linee il protocollo *BB84* è organizzato nel modo seguente:

1. Alice (A) invia la sequenza codificata S_A sul canale quantistico.

2. Bob (B) interpreta la S_A con le basi che ha scelto ottenendo una sequenza S_B che coincide con S_A per circa metà degli elementi.

3. Usando il canale standard, Bob comunica ad Alice le basi che ha scelto e Alice gli indica quali basi sono comuni alle sue.

4. In assenza di interferenze sul canale quantistico da parte di un crittoanalista attivo Eve, Alice e Bob possiedono una sottosequenza identica $S'_A = S'_B$ relativa ai bit codificati e decodificati con le basi comuni che potrebbe costituire la chiave.

Si noti però che non vi è certezza dell'assenza di Eve, che dovremo scoprire.

Per comprendere questa prima parte del protocollo ricordiamo che l'esito della lettura di un fotone da parte di Bob dipende dalla base scelta, come è riassunto nella tabella di figura 11.2.

Figura 11.2: Esito della lettura di Bob su un fotone inviato da Alice. La presenza di due frecce in una stessa casella indica che la lettura è imprevedibile.

Il protocollo può ora essere seguito sull'esempio di figura 11.3. Si noti che le sequenze S'_A, S'_B coincidono, ma anche altri bit di S_A e S_B potrebbero essere uguali per l'indeterminatezza della lettura con basi diverse (nell'esempio $S_A(5) = S_B(5)$): tuttavia non si può fare alcun uso di questi bit.

Il crittoanalista Eve potrebbe intercettare la sequenza spedita da Alice, interpretarla e rispedirla a Bob: per fare questo dovrebbe usare una sequenza di basi per la lettura inevitabilmente scorrelata dalla sequenza di basi scelta da Alice che Eve non conosce, e questo distruggerebbe parte dell'informazione contenuta nella sequenza S_A. In conseguenza Bob, interpretando la sequenza di fotoni ricevuta da Eve otterrà una sequenza S_B che coincide con la S_A in meno elementi di quelli di S'_B, perchè questi saranno in parte corrotti quando vengono rielaborati da Eve con basi diverse. Dunque il protocollo prosegue effettuando il passo:

1	2	3	4	5	6	7	. . . .	successione temporale
1	0	1	1	1	0	0	. . . .	S_A
+	×	+	×	×	+	×	. . . .	basi di A
→	↗	→	↘	↘	↑	↗	. . . .	invio di A
+	×	+	+	+	×	×	. . . .	basi di B
→	↗	→	↑	→	↘	↗	. . . .	lettura di B
1	**0**	**1**	0	1	1	**0**	. . . .	S_B

Figura 11.3: Le prime fasi del protocollo *BB84* in assenza di un crittoanalista sul canale quantistico. La sottosequenza comune $S'_A = S'_B$ è indicata in grassetto.

5. Alice e Bob sacrificano una parte S''_A, S''_B delle sequenze S'_A, S'_B in posizioni prestabilite, comunicandole sul canale standard. In caso di errore, cioè se $S''_A \neq S''_B$, i due partner interrompono lo scambio di chiave perché vi è evidenza di intercettazione o comunque di malfunzionamento. In caso di assenza di errore, $S'_A - S''_A = S'_B - S''_B$ è adottata come chiave.

Le operazioni del protocollo, completato dall'intrusione di Eve (E), può essere seguito sulla figura 11.4 ove le basi scelte da Alice e Bob e la sequenza S_A sono le stesse della figura 11.3, e Eve si inserisce con una terza scelta di basi. L'interpretazione della figura è analoga a quella della figura precedente ed è lasciata al lettore. Si noti in particolare che $S'_B(7) = S'_A(7)$ nonostante che la base di Eve in quella posizione non coincida con la base comune di Alice e Bob: ciò dipende dall'incertezza della lettura di Bob che ha usato una base diversa da quella di Eve.

È bene chiarire come Eve potrebbe utilizzare il suo attacco. La sequenza S_E che Eve costruisce con le sue basi coincide con S'_A e S'_B nei punti in cui Eve ha utilizzato le stesse basi di Alice e Bob (istanti 1 e 3 nella figura 11.4): in media un bit di S_E su quattro è corretto, mentre gli altri non sono significativi. Poiché il canale standard con cui Alice e Bob si comunicano le basi scelte non è protetto, Eve può facilmente intercettare la comunicazione tra i due e stabilire in quali punti le sue chiavi coincidono con le loro, rubando così un quarto della chiave. Senza il passo 5 del protocollo per il controllo delle intrusioni, il danno per Alice e Bob sarebbe molto serio.

Per aggirare il controllo si potrebbe supporre che Eve, oltre a interpretare i fotoni in arrivo con le sue chiavi, li "duplichi" prima di leggerli per inviarli incorrotti a Bob: in tal modo Alice e Bob non rileverebbero alcuna differenza nelle porzioni di sequenze S''_A e S''_B e l'attacco potrebbe avere luogo come detto sopra senza essere scoperto.

1	2	3	4	5	6	7	. . . .	successione temporale
1	0	1	1	1	0	0	. . . .	S_A
+	×	+	×	×	+	×	. . . .	basi di A
→	↗	→	↘	↘	↑	↗	. . . .	invio di A
+	+	+	×	+	×	+	. . . .	basi di E
→	→	→	↘	↑	↗	→	. . . .	lettura e invio di E
1	1	1	1	0	0	1	. . . .	S_E
+	×	+	+	+	×	×	. . . .	basi di B
→	↗	→	↑	↑	↗	↗	. . . .	lettura di B
1	**1**	**1**	0	0	0	**0**	. . . .	S_B

Figura 11.4: Esecuzione del protocollo *BB84* in presenza di un crittoanalista Eve sul canale quantistico. La sottosequenza S'_B calcolata da Bob con le basi comuni ad Alice, indicata in grassetto, differisce dalla S'_A nell'elemento $S_B(2) \neq S_A(2)$. Posto che Alice e Bob concordino di sacrificare proprio i primi quattro bit di S'_B comunicandoli sul canale standard, l'intrusione viene scoperta.

Ma a questo si oppone uno dei principi della meccanica quantistica citati all'inizio del capitolo, che indica come non sia possibile duplicare un sistema quantistico senza causarne la decoerenza. In sostanza il protocollo apparirebbe assolutamente sicuro: ma, come ora vedremo, le cose non sono così semplici.

11.1.2 Problemi di realizzazione e alternative

La discussione sul protocollo *BB84* riportata qui sopra, e gli esempi esplicativi delle figure 11.3 e 11.4, hanno l'unico scopo di far comprendere la struttura di base del protocollo. Le sequenze effettivamente inviate sono costituite da migliaia di bit e i fenomeni di concordanza o meno delle sequenze si manifestano con certezza statistica; il punto però in cui questi esempi elementari si discostano completamente dalla realtà è legato agli inevitabili errori nella trasmissione e nella lettura, collettivamente indicati come *rumore*. Anzitutto i fotoni possono subire piccole variazioni di polarizzazione transitando nel canale quantistico e non giungere a Bob perfettamente allineati con i moduli PBS; e anche questi moduli possono avere piccole imperfezioni di allineamento nel montaggio. Dunque, pur se con piccola probabilità, le uscite dei moduli PBS possono essere errate anche se le basi scelte sono corrette: come ora vedremo

il fenomeno viene contrastato con l'impiego di opportuni codici di trasmissione, ma per Alice e Bob è indispensabile poter stabilire se gli errori sono dovuti a rumore o all'intrusione di Eve.

In base alle caratteristiche del canale e dei dispositivi impiegati si stabilisce una percentuale prevedibile di bit errati dovuti al rumore, detta *Quantum bit error rate* ($QBER$), che viene prudenzialmente tenuta alta. Se la percentuale di errori nella porzione di S'_B sacrificata per il controllo (passo 5 del protocollo riportato sopra) supera il $QBER$, Alice e Bob concludono che vi è stata una intrusione di Eve o comunque un serio malfunzionamento del sistema e la chiave viene scartata. Se la percentuale di errori è tollerabile, Alice e Bob devono comunque ricostruire una chiave corretta. A tale scopo la sequenza S_A viene rappresentata usando un *codice a correzione di errore* commisurato al $QBER$, per la cui descrizione rimandiamo ai testi di Teoria dei Codici: in tal modo Alice e Bob ricostruiscono la chiave corretta.

Vi è comunque un attacco subdolo che Eve può intraprendere aggirando la misura del $QBER$. Eve intercetta solo una piccola parte della chiave con lo scopo di ottenere un'informazione parziale su di essa: a tale fine individua i punti in cui la sua polarizzazione coincide con quelle di Alice e Bob, che dedurrà spiando la comunicazione tra questi sul canale standard, perché in tali punti la sua lettura coincide con quella dei due partner. Il numero di errori così introdotto è molto basso e il $QBER$ può non essere superato. Per tale motivo Alice e Bob, stabilita la sequenza corretta S_C derivante dall'informazione comune dopo la correzione data dal codice, eseguono una *amplificazione di privacy* dividendo la S_C in blocchi e calcolando per ciascuno di essi una funzione hash crittograficamente forte la cui uscita sarà utilizzata come chiave. Poiché Eve conosce solo alcuni bit di S_C non potrà in alcun modo risalire ai bit della chiave a causa delle note proprietà delle funzioni hash. Complessivamente il protocollo *BB84* normalmente implementato è il seguente:

Protocollo *BB84* per lo scambio quantistico di chiavi soggetto a interferenza di un crittoanalista attivo.

Sul canale quantistico: $S_A[1,n]$ è la sequenza iniziale di bit da cui estrarre la chiave, rappresentata con un codice a correzione di errore.

for $i = 1$ **to** n

Alice: sceglie una base a caso, codifica $S_A[i]$, invia il relativo fotone a Bob;

Eve (se presente): intercetta il fotone con una sua base, lo invia a Bob, calcola $S_E[i]$;
(alternativamente può intercettare solo alcuni fotoni spediti da Alice);

Bob: sceglie una base a caso, interpreta il fotone ricevuto, costruisce $S_B[i]$.

Sul canale standard: $QBER$ è il valore fissato come percentuale massima di bit errati dovuti a rumore; h è una funzione hash concordata e crittograficamente forte.

Bob: comunica ad Alice la sequenza di basi scelte;

Alice: comunica a Bob le basi comuni;

Alice e Bob: singolarmente e comunicando tra loro:

1. estraggono le sottosequenze S'_A e S'_B corrispondenti alle basi comuni, e due sottosequenze di esse S''_A e S''_B in posizioni concordate;
2. si scambiano S''_A e S''_B; se la percentuale di bit in cui queste sequenze differiscono è maggiore di $QBER$ **lo scambio è interrotto**, altrimenti:
   3. calcolano le sequenze $S'_A - S''_A$ e $S'_B - S''_B$, e le decodificano con il codice a correzione di errore ottenendo la sequenza comune S_C;
   4. calcolano $k = h(S_C)$; **assumono** k **come chiave**.

Dopo la proposta di Bennett e Brassard il protocollo *BB84* è stato realizzato con diversi dispositivi circuitali e ha comunque fornito lo schema di base anche per altri metodi di QKD. Infatti, pur sfruttando altri apparecchi ottici e diverse caratteristiche dei fotoni polarizzati, in particolare l'*entanglement* di cui abbiamo accennato all'inizio di questo capitolo, la sicurezza di tutti i protocolli si basa sull'alterazione dello stato quantico di un fotone se un crittoanalista lo intercetta e decodifica: questo permette agli utenti legittimi di individuare la manomissione.

Per quanto riguarda la trasmissione dei dati, sia nel protocollo iniziale che nelle sue realizzazioni successive, il mezzo principale è stato la fibra ottica anche su linee commerciali. Tuttavia la distanza a cui si può far giungere la trasmissione è a tutt'oggi limitata a un centinaio di chilometri oltre i quali la qualità del segnale degrada: in un sistema elettrico convenzionale si rimedia a questo inconveniente inserendo sulla linea apparecchi di ricezione e amplificazione del segnale, ma questo è ovviamente impossibile usando fotoni polarizzati il cui stato quantico sarebbe distrutto dalla lettura. Vi sono stati diversi esperimenti anche di successo per trasmettere i fotoni polarizzati via etere, i più avanzati con raggi laser attraverso un satellite, e questi studi sono oggi (2014) in piena evoluzione; ma per il momento la fibra ottica usata per distanze relativamente brevi rimane la via di tutte le applicazioni commerciali.

Il sistema di più grandi dimensioni in via di realizzazione consiste in un anello di postazioni lungo circa 600 km installato nell'area di Washington. La rete in fibra ottica collega un insieme di nodi QKD, ciascuno distante dal successivo meno di 100 km. La trasmissione tra due nodi non collegati direttamente avviene attraverso i nodi intermedi dell'anello. Così per comunicare una chiave tra i nodi A e C attraverso B, la trasmissione di A è interpretata da B e ritrasmessa a C mediante il protocollo *BB84*,

ove le operazioni e i dati all'interno di B sono protetti crittograficamente con One-Time Pad. Ciò implica che l'intero anello sia dedicato a una istituzione particolare e non sia utilizzabile da altri, con costi al momento piuttosto elevati: a parte la non condivisibilià del sistema, si stima che l'hardware installato abbia un costo circa il 50% superiore a quello di un corrispondente sistema convenzionale: ma ciò è ritenuto accettabile in relazione all'alto livello di sicurezza raggiunto.

Per concludere vediamo come si possa organizzare un sistema QKD sfruttando l'*entanglement* di fotoni polarizzati. Questa proprietà si riferisce a coppie di particelle generate contemporaneamente che mantengono uno stato quantico correlato tra loro in modo che la misura dello stato di una particella della coppia altera istantaneamente lo stato dell'altra, e il fenomeno si verifica indipendentemente dalla posizione nello spazio delle due particelle. Per esempio è possibile generare una coppia di fotoni entangled polarizzati in direzioni perpendicolari, che mantengono questo stato finché uno dei due viene "disturbato" da una misurazione perdendo il suo stato di particella quantica. Se la polarizzazione dell'uno è stata alterata dalla misurazione, la polarizzazione dell'altro subisce una trasformazione corrispondente per mantenere correlate le due polarizzazioni, indipendentemente dalla distanza tra i due fotoni della coppia.

Il protocollo di base che sfrutta questo fenomeno fu proposto dal polacco Artur Ekert nel 1991. Una sorgente indipendente di coppie di fotoni α, β entangled polarizzati secondo le basi + o $\times$, spedisce α ad Alice e β a Bob. I due partner leggono i fotoni con basi casuali: se le due basi coincidono la lettura è identica per entrambi (o meglio, correlata, perché indica due polarizzazioni ortogonali); se le basi non coincidono la lettura è imprevedibile e non significativa. Il protocollo può essere ora organizzato sulle stesse linee di *BB84* come il lettore potrà facilmente immaginare.

Anche il protocollo di Ekert è stato realizzato secondo diverse variazioni e potrebbe essere un metodo vincente in un futuro prossimo.

11.2 Il computer quantistico

Vediamo ora più da vicino su quali basi sono realizzati i computer quantistici e quali effetti potrebbero avere sulla crittografia. Come detto un elemento di base dell'hardware è il registro qR di n qbit, ciascuno dei quali può trovarsi nella sovrapposizione di due stati quantici che indicheremo con 0 e 1 anche se in questo caso la notazione non è standard. Tale sovrapposizione, tuttavia, deve essere intesa in senso probabilistico.

Confrontiamo qR con un registro R standard. Come sappiamo R può contenere in ogni istante una tra le 2^n configurazioni di n bit, da tutti 0 a tutti 1: ci riferiamo alla configurazione contenuta in R come allo stato del registro. Nella computazione deterministica, in cui si sviluppa la grande maggioranza delle attività dei calcolatori,

lo stato di R in ogni istante è esattamente determinato, dunque la corrispondente configurazione di bit è presente con probabilità 1 mentre tutte le altre hanno probabilità 0. In una computazione probabilistica, come per esempio quella degli algoritmi randomizzati (paragrafo 4.3) o dei protocolli zero-knowledge (paragrafo 9.7), R potrà contenere diverse tra le 2^n configurazioni di n bit, ciascuna con una certa probabilità $p_i, 1 \leq i \leq 2^n$ a seconda del calcolo eseguito; e la somma di tutte queste probabilità deve risultare uguale a 1. Per esempio nel protocollo di identificazione di Fiat-Shamir del paragrafo 9.7 il parametro $z = rs^e \bmod n$ calcolato dal *prover* P al passo 4 avrà valore $z = r$, o $z = rs \bmod n$, a seconda del valore $e = 0$, o $e = 1$, estratto perfettamente a caso dal *verifier* V al passo precedente: dunque le due corrispondenti configurazioni di n bit nel registro contenente z avranno ciascuna probabilità $1/2$, e tutte le altre configurazioni avranno probabilità zero.

Nel registro qR la sovrapposizione quantistica degli stati dà luogo a una situazione assai più complessa. Per studiarla, le 2^n probabilità p_i delle configurazioni sono costituite da numeri complessi anziché reali e la metrica di calcolo impone che la somma dei quadrati dei loro moduli sia uguale a 1, ovvero $\sum_{i=1}^{2^n} |p_i|^2 = 1$. Al di là del calcolo e della sua descrizione del mondo fisico, troppo complicati per essere discussi qui, ricordiamo che sono ora i valori di tali quadrati a rappresentare le probabilità delle diverse configurazioni sovrapposte.

In modo molto sommario il modello classico di calcolo quantistico è il seguente. Inizialmente una specifica configurazione binaria che rappresenta i dati del problema da risolvere viene caricata in un registro qR. Da questo momento i registri del sistema si comportano e interagiscono in modo quantistico, connessi tra loro attraverso circuiti logici elementari che operano su qbit rispettando la natura dei fenomeni in gioco. La sovrapposizione di stati nei registri dà origine a una sovrapposizione di calcoli ciascuno legato a una certa probabilità di convergere verso un risultato significativo. Alla fine delle operazioni il risultato, contenuto anch'esso in un registro, si presenta come una sovrapposizione di risultati e collassa su quello con più alta probabilità in seguito alla lettura del registro. Dunque l'interazione con l'esterno avviene attraverso classiche sequenze di bit: di fatto qbit che assumono in quell'istante un unico stato.

Il problema più serio che si presenta nello sviluppo di questi sistemi, e che forse più di ogni altro ne ha ritardato la realizzazione al di fuori dei laboratori specializzati, è quello già accennato della decoerenza, cioè la possibilità che il sistema perda la sovrapposizione di stati a causa di disturbi di qualsiasi genere, sia esterni che interni ai meccanismi impiegati per rappresentare i qbit. Poiché la decoerenza si presenta in genere in tempi brevissimi, è necessario che il calcolo possa concludersi in tempi ancora più brevi.

L'idea di calcolo quantistico è nata con lo scopo di simulare fenomeni di fisica

quantistica troppo difficili per poter essere affrontati con metodi di calcolo standard. Il modello ha però attratto anche gli informatici per l'aiuto che avrebbe potuto fornire nella soluzione di problemi di alta complessità computazione o addirittura esponenziali. Prima di discutere dei principali algoritmi proposti e della loro utilizzazione, cerchiamo di inquadrare i problemi che si possono affrontare nel modello quantistico nella gerarchia di complessità studiata nel capitolo 3. Anzitutto la computazione quantistica non viola la *tesi di Church-Turing* in base alla quale non esiste modello di calcolo sostanzialmente più potente della Macchina di Turing quanto ai problemi risolubili con essa. Dal punto di vista della complessità si pone:

Definizione. **BQP** è la classe dei problemi decisionali risolubili con un algoritmo quantistico in tempo polinomiale, tali che la probabilità di errore sia $< 1/2$.[1]

La definizione di **BQP**, intrinseca alla natura dei fenomeni utilizzati per il calcolo, è legata alla possibilità che un algoritmo quantistico fornisca un risultato garantendone la correttezza solo con una certa probabilià, purché questa sia maggiore di 1/2. Questo non deve preoccupare perché, con un procedimento simile a quello che già abbiamo discusso per gli algoritmi randomizzati, la ripetizione dell'algoritmo permette di aumentare la probabilità di successo fino a un valore desiderato se il risultato è ogni volta identico. Nella gerarchia di complessità, che come si ricorderà è comunque condizionata dal quesito irrisolto "$\mathbf{P} \neq \mathbf{NP?}$", si ritiene a oggi che $\mathbf{P} \subset \mathbf{BQP}$ e $\mathbf{BQP} \cap \mathbf{NPC} = \emptyset$ (figura 11.5). Si ritiene anche che esistano problemi in **BQP** non contenuti in **NP**. Queste relazioni smentiscono l'affermazione "popolare" che il calcolo quantistico porterà in campo polinomiale i problemi **NP**-completi: come vedremo questo fatto gioca un ruolo importante in crittografia.

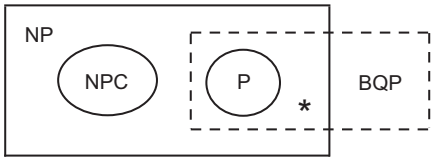


Figura 11.5: Presunta relazione tra le classi **P**, **NP**, **NPC** e **BQP**. L'asterisco indica la zona ove è contenuto il problema $\mathcal{P}_{2fat}(N)$.

Nel mondo più vasto dei problemi non esclusivamente decisionali, tra cui i problemi "ardui" della classe **NPH** (paragrafo 3.4), un problema fondamentale per noi

[1] L'acronimo **BQP** sta per "Bounded-error Quantum Polynomial", ovvero "quantistico polinomiale a errore limitato".

è quello di scomporre un intero N in fattori primi: è detto *problema della fattorizzazione* e lo indicheremo con $\mathcal{P}_{fat}(N)$. Una sua versione decisionale $\mathcal{P}_{2fat}(N)$ chiede di stabilire se N è scomponibile nel prodotto di due primi. Banalmente $\mathcal{P}_{2fat}(N) \in \mathbf{NP}$ perché ciascuno dei due fattori ne costituisce un certificato polinomiale; a oggi non si conosce un algoritmo deterministico polinomiale per risolverlo ma, come vedremo, tale algoritmo esiste per un computer quantistico: quindi $\mathcal{P}_{2fat}(N) \in \mathbf{NP} \cap \mathbf{BQP}$, ma $\mathcal{P}_{2fat}(N) \notin \mathbf{P}$ (figura 11.5).

Un algoritmo per risolvere $\mathcal{P}_{fat}(N)$ in tempo polinomiale su un computer quantistico è stato proposto nel 1994 da Peter Shor ed ha avuto una enorme risonanza nel campo della crittografia perché la disponibilità di una macchina su cui possa essere eseguito metterebbe fuori gioco tutti i cifrari a chiave pubblica oggi più comunemente usati; la più immediata conseguenza si avrebbe sul cifrario *RSA*. Ricordiamo infatti che in questo cifrario la chiave pubblica contiene un intero ottenuto come prodotto di due primi e la chiave privata si può calcolare solo conoscendo questi due fattori (paragrafo **??**). Nella notazione standard di *RSA* questi interi sono indicati come $n = p \cdot q$; li indichiamo qui con le maiuscole $N = P \cdot Q$ mentre $n = \lceil log_2 N \rceil$ indica il numero di bit con cui è rappresentato N, cioè la dimensione del problema.

L'algoritmo di Shor è troppo complicato per poterlo descrivere qui. Esso accoppia una parte standard di calcolo deterministico polinomiale a una trasformazione matematica risolubile in tempo $O(n^3)$ su un computer quantistico, cosa impossibile su un calcolatore standard.[2] Adattamenti dell'algoritmo permettono di risolvere in tempo polinomiale il problema del logaritmo discreto e quindi compromettere il protocollo *DH* per lo scambio di chiavi (paragrafo 8.6), e di attaccare i cifrari a chiave pubblica basati sulle curve ellittiche (paragrafo 8.7).

Sembrerebbe dunque che l'algoritmo di Shor sia destinato a giocare un ruolo fondamentale nel futuro della crittografia; ma un problema legato alla sua esecuzione, solo raramente sottolineato, indica che un suo uso corrente sarà improbabile al di là della disponibilità effettiva di computer quantistici. Infatti il tempo richiesto dall'algoritmo è cubico, ma anche la dimensione del circuito cresce con la stessa legge. In particolare i registri di ingresso e uscita devono contenere un numero di qbit compreso tra $2n$ e $4n$, ma il numero di porte logiche per elaborare tale informazione secondo leggi quantistiche è proporzionale a n^3: un conteggio più accurato indica per esempio che per attaccare una chiave *RSA* di quattromila bit occorrono più di quattromila Giga di porte logiche. Il problema non è solo l'abnorme dimensione del circuito che dovrebbe funzionare senza decoerenza, ma il fatto stesso che le dimensioni della macchina debbano essere stabilite in funzione della dimensione del problema

[2] L'algoritmo deterministico più efficiente che si conosca per la fattorizzazione richiede un tempo che cresce più di $2^{2n^{1/3}}$.

potrebbe confinare l'algoritmo nel mondo della teoria. Nella pratica, dopo il primo annuncio del MIT nel 2001 sull'esecuzione dell'algoritmo di Shor per fattorizzare il numero 15 su un computer quantistico con registri di 7 qbit (comunque fondamentale per dimostrare la correttezza del procedimento e la sua possibile realizzazione), altre esecuzioni sono seguite finora su numeri solo di pochi bit. La vera grande importanza dell'algoritmo di Shor risiede nell'aver mostrato che la crittografia attuale è attaccabile, stimolando lo studio di altri algoritmi quantistici e di cifrari *post-quantistici* che possano comunque resistergli. Vediamo dunque in quali direzioni si stanno movendo gli studi in questo campo.

Anzitutto, posto di disporre di chiavi random continuamente rinnovabili, il cifrario *One-Time Pad* è inattaccabile anche da un computer quantistico perché la perfezione del cifrario è basata su un argomento generale di teoria dell'informazione indipendente dal modello di calcolo impiegato. Dunque abbinare *One-Time Pad* con la distribuzione quantistica delle chiavi descritta nel paragrafo precedente costituisce una soluzione crittografica di assoluta sicurezza, almeno finché non saranno trovate nuove vie di attacco a tale distribuzione. Con la tecnologia attuale questa soluzione è però utilizzabile solo da un limitato numero di utenti "importanti" per la necessità di stabilire un collegamento diretto tra ogni coppia di essi per la distribuzione quantistica delle chiavi. Per un impiego generale si cercano quindi altre vie.

Poiché i computer quantistici non sembrano in grado di risolvere in tempo polinomiale problemi della classe **NPH**, l'idea di base dei cifrari post-quantistici è quella di poterne dimostrare la riduzione con uno di quei problemi: dimostrare cioè che compromettere il cifrario implica risolvere un problema tra i più difficili della classe **NP**. Le proposte che sono state avanzate finora si basano su diverse tecniche e non per tutte sono state dimostrate le riduzioni necessarie. In alcuni casi ci si accontenta della riduzione con problemi di cui non si conosce soluzione deterministica polinomiale senza dimostrazione che appartengano alla classe **NPH**, ma che comunque non sono risolubili con l'algoritmo di Shor o con le sue variazioni note.

Anche in questo caso sarebbe davvero troppo complicato entrare in una specifica descrizione di questi cifrari: citiamo solo che la famiglia che appare più promettente al momento è basata su insiemi ordinati di punti detti *reticoli* trattati con metodi algebrici, per cui in alcuni casi è stata dimostrata la riduzione con problemi in **NPH**. Un problema che riguarda tutti questi cifrari è comunque la eccessiva lunghezza delle chiavi che in casi estremi ne attesta solo un interesse teorico. In ogni modo si tratta di cifrari asimmetrici intesi a sostituire *RSA* e curve ellittiche. Esiste però un'applicazione del calcolo quantistico nell'attacco ai cifrari simmetrici che, pur di efficienza più limitata rispetto all'algoritmo di Shor, è comunque interessante: si basa su un algoritmo quantistico differente proposto da Lov Grover nel 1996.

L'algoritmo di Grover in realtà è stato proposto per la ricerca di un dato in un elenco non ordinato di n elementi e richiede tempo $O(n^{1/2})$ anziché $O(n)$ come nel calcolo standard; ma può essere immediatamente adattato alla ricerca esauriente di elementi con una proprietà assegnata come per esempio le chiavi di un cifrario simmetrico. In questo caso la riduzione di complessità non passa da esponenziale a polinomiale come nell'algoritmo di Shor, ma una riduzione da $O(n)$ a $O(\sqrt{n})$ può essere comunque importante per valori grandi di n. Nel caso di attacco per enumerazione delle possibili chiavi di un cifrario simmetrico o asimmetrico ciò significa che per mantenerne la sicurezza è necessario raddoppiare la lunghezza delle chiavi stesse: per esempio attaccare *AES* con chiavi di 512 bit mediante l'algoritmo di Grover con un calcolatore quantistico avrebbe la stessa efficacia che attaccare *AES* con chiavi di 256 bit per semplice enumerazione.

Concludiamo questa breve carrellata sull'effetto del calcolo quantistico notando che moltissime operazioni eseguite dai calcolatori tradizionali si realizzano in modo difficile e inefficiente nei computer quantistici, che invece hanno un assoluto sopravvento su quelle parti del calcolo che riguardano l'esame di un gran numero di configurazioni di dati, alla ricerca di quella, o quelle, con determinate proprietà. Tale vantaggio può essere enorme, ma si manifesta nel caso che la struttura del problema non consenta di prendere particolari scorciatoie che escludano la necessità di ricerche esaurienti, come avviene per la classe **P**; e che il problema stesso si sottometta alle leggi del calcolo quantistico come nel caso della fattorizzazione (ma non per i problemi **NP**-hard). Potremo dunque aspettarci che i computer quantistici, se entreranno effettivamente nell'uso, siano composti da un blocco di circuiti tradizionali per le operazioni correnti e da un blocco di circuiti quantistici per risolvere le enumerazioni non strutturate.

Bibliografia

ALCUNE BASI MATEMATICHE

- Ivan Niven e Herbert S. Zuckerman. *An introduction to the theory of numbers*. John Wiley & Sons, 1980. Uno dei testi standard di riferimento sulla teoria dei numeri.

- Giuseppe Longo. *Teoria dell'informazione*. Boringhieri, 1980.
  Importante testo in italiano, qui segnalato per la teoria della codifica e per un ricco compendio di concetti e risultati matematici relativi alla teoria dell'informazione.

- Ming Li e Paul M.B. Vitanyi. Kolmogorov complexity and its applications. In *Handbook of Theoretical Computer Science*, Chapter 4. Elsevier Publisher, 1990.
  Una fonte diretta e ragionevolmente accessibile per approfondire questo difficile tema.

ALGORITMICA

- Thomas H. Cormen, Charlie E. Leiserson, Ron L. Rivest e Clifford Stein. *Introduzione agli algoritmi e strutture dati*. McGraw-Hill, 2005.
  Testo standard internazionale sugli algoritmi in genere, qui citato nella traduzione italiana.

- Rajeev Motwani e Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.
  Testo standard internazionale sugli algoritmi randomizzati.

- Donald Knuth. *The Art of Computer Programming: Seminumerical Algorithms*. Addison-Wesley, 1998.
  Testo fondamentale, particolarmente utile in questo contesto per lo studio dei generatori di numeri pseudocasuali.

- Jonathan Knudsen. *Java Cryptography.* O'Reilly, 1998.
  Testo introduttivo all'uso del linguaggio Java per la programmazione di protocolli crittografici.

CRITTOGRAFIA

- Ronald L. Rivest. *Cryptography.* In *Handbook of Theoretical Computer Science*, Chapter 10. Elsevier Publisher, 1990.
  Classica rassegna scritta da uno dei massimi crittografi contemporanei. Consigliabile a tutti come riferimento iniziale.

- Andrea Sgarro. *Crittografia.* Franco Muzzio Editore, 1993.
  Un classico italiano del campo, accoppia la serietà di trattamento a un brillante stile di presentazione.

- Hans Delfs e Helmut Knebl. *Introduction to Cryptography: Principles and Applications.* Springer, 2002.
  Trattato molto serio a carattere spiccatamente matematico, si segnala per un esteso esame dei protocolli interattivi.

- Wenbo Mao. *Modern Cryptography: Theory and Practice.* Prentice Hall, 2004.
  Contiene una parte sostanziale di calcolo delle probabilità, computabilità, algebra e teoria dei numeri, propedeutica ai capitoli seguenti sulla crittografia. Un testo di livello-medio alto che si segnala in particolare per l'esteso trattamento dell'integrità e dell'autenticazione.

- William Stallings. *Crittografia e sicurezza delle reti.* II Edizione. McGraw-Hill Education, 2007.
  Un testo standard internazionale su crittografia e sicurezza, qui citato nella traduzione italiana.

- Joan Daemen e Vincent Rijmen. *The Design of Rijndael.* Springer, 2002.
  Testo fondamentale per la descrizione dell'*AES* scritto dagli autori del cifrario.

- Lawrence C. Washington. *Elliptic Curves Number Theory and Cryptography.* II Edition. Chapman & Hall, 2008.
  Un testo fondamentale sulle curve ellittiche e il loro impiego in crittografia.

- Simon Singh. *Codici e segreti.* Rizzoli, 1999.
  Apprezzabile testo di storia della crittografia scritto da un esperto di divulgazione scientifica.

- Bruce Schneier. *Secrets and Lies: Digital Security in a Networked World.* John Wiley & Sons Inc., 2000.
  Testo divertente e professionale sui limiti e sul futuro dei prodotti crittografici.

APPLICAZIONI E SICUREZZA DELLE RETI

- Ross Anderson. *Security Engineering.* II Edition. Wiley, 2008.
  Considerato in genere il miglior testo sulla sicurezza e sulla crittografia dal punto di vista ingegneristico, impressiona per la vastità dei temi trattati con altissima competenza. Dal 2013 l'intero testo può essere scaricato gratis accedendo al sito Web dell'autore.

- Mike Hendry. *Smart card Security and Applications.* II Edition. Artech House, 2001.
  Testo rivolto alle applicazioni delle smart card, ricco di dettagli pratici ma scientificamente non molto profondo.

- Larry L. Peterson e Bruce S. Davie. *Computer Networks. A Systems Approach..* Morgan Kaufmann, 2000.
  Testo molto chiaro, si segnala anche per la esplicita presentazione dei principali protocolli di rete in linguaggio C.

- William Stallings e Lawrie Brown. *Computer Security: Principles and Practice.* III Edition. Prentice Hall, 2014.
  Un trattamento molto esteso sulla sicurezza dei calcolatori e delle reti. Probabilmente il miglior testo sull'argomento in circolazione oggi.

SITI WEB

- Compilare un elenco soddisfacente di siti interessanti sulla crittologia è compito vano per l'immensa mole di riferimenti possibili e per la loro estrema variabilità nel tempo. Alcuni siti rilevanti per la materia trattata in singoli punti del testo sono stati indicati nelle note. Il lettore potrà comunque accedere alle *directory* relative al soggetto "cryptography" o simili messe a disposizione dai principali motori di ricerca. Gli interessati a seguire l'evoluzione degli standard crittografici, e i curiosi di "national security", potranno riferirsi ai siti di *NIST* e *NSA* (o di *WikiLeaks* secondo i gusti), rispettivamente:

  `http://csrc.ncsl.nist.gov/`

  `http://www.nsa.gov/`

  `http://www.wikileaks.org/`

Indice analitico

Finito di stampare nel mese di gennaio 2015
da Litogì srl - Milano
per conto di Pisa University Press