Capitolo 10

Protocolli per la sicurezza

Fino a questo punto, abbiamo visto una serie di strumenti crittografici, dagli algoritmi di cifratura alle funzioni hash e alle firme digitali. Resta da capire come usare questi strumenti per rendere sicuri i computer e le comunicazioni.
Si potrebbe pensare che i metodi a chiave pubblica siano la panacea di tutti i problemi: permettono di scambiare messaggi in modo sicuro a due soggetti che non si sono mai incontrati, inoltre forniscono un modo facile per autenticare l'origine di un messaggio e, se combinati con le funzioni hash, possono farlo in modo efficiente.
Sfortunatamente, gli algoritmi crittografici non sono in grado, da soli, di risolvere tutti i problemi. Per esempio, non abbiamo mai discusso della distribuzione delle chiavi pubbliche. Abbiamo menzionato che Alice annuncia la chiave pubblica di cui si servirà Bob. Tuttavia Bob non può essere così ingenuo da fidarsi di un annuncio pubblico. Come può sapere che l'annuncio provenga effettivamente da Alice? Forse proviene da una nemica di Alice, Mallory, che si finge Alice, ma, in realtà, annuncia la propria chiave. Allo stesso modo, quando si accede a un sito web per effettuare un acquisto, come si può essere sicuri che la transazione avvenga con il vero negozio elettronico e non con un finto negozio predisposto in modo truffaldino? La sfida di questi problemi è l'autenticazione: prima di inviare informazioni importanti, Bob deve sempre avere la conferma che il suo interlocutore sia Alice.
Fornire sicurezza combinando opportunamente diversi strumenti crittografici è più difficile che applicare un algoritmo preconfezionato. Per questo motivo i protocolli che regolano lo scambio di messaggi tra entità devono essere definiti con attenzione in modo da anticipare ed evitare attacchi ingegnosi. Il resto del capitolo tratta i protocolli per la sicurezza.

10.1 Intrusi e impostori

Se ricevete una e-mail che vi chiede di collegarvi a un sito web e aggiornare le vostre credenziali di accesso, come potete essere sicuri che il sito web sia legittimo? Un impostore può creare facilmente una pagina web identica all'originale, ma che registri i dati sensibili e li inoltri a Eva. Questo è un problema importante di autenticazione che deve essere affrontato dalle implementazioni reali dei protocolli crittografici. Una soluzione standard, discussa nel Paragrafo 10.7, usa i certificati e le autorità fidate.
In generale, l'autenticazione gioca un ruolo importante in molti protocolli descritti in questo capitolo e, insieme al problema dell'intruso, è il motivo della complessità dei protocolli per la sicurezza.

10.1.1 Attacco dell'intruso

Eva, che ha recentemente scoperto la differenza tra il cavallo e la torre, afferma di poter affrontare simultaneamente due grandi maestri di scacchi e di poter, in alternativa, vincere una partita oppure pareggiarle entrambe. La strategia è semplice: Eva aspetta la mossa del primo maestro e fa la stessa mossa contro il secondo maestro; quando il secondo maestro risponde, Eva gioca la stessa mossa contro il primo maestro. Continuando in questo modo Eva non può perdere entrambe le partite (se trascuriamo la possibilità che Eva superi il tempo massimo disponibile a causa del ritardo impegato nel trasferire le mosse).
Una strategia simile, chiamata **attacco dell'intruso** (*intruder-in-the-middle*), può essere usata per attaccare molti protocolli crittografici.
Questo attacco può essere usato, per esempio, contro il protocollo di scambio della chiave di Diffie-Hellman. Il protocollo, già presentato nel Paragrafo 7.4, permette ad Alice e Bob di scambiare una chiave di sessione e opera come segue.

1. Alice (o Bob) sceglie un numero primo p per il quale sia difficile il calcolo del logaritmo discreto e una radice primitiva $\alpha \pmod p$. I numeri p e α sono resi pubblici.
2. Alice sceglie un numero segreto casuale x con $1 \leq x \leq p-2$ e Bob sceglie un numero segreto casuale y con $1 \leq y \leq p-2$.
3. Alice invia $\alpha^x \pmod p$ a Bob e Bob invia $\alpha^y \pmod p$ ad Alice.
4. Usando ciascuno il messaggio ricevuto, Alice e Bob calcolano la chiave di sessione K. Alice calcola K come $K \equiv (\alpha^y)^x \pmod p$ e Bob calcola K come $K \equiv (\alpha^x)^y \pmod p$.

L'attacco dell'intruso può essere eseguito in questo modo.

1. Eva sceglie un esponente z.
2. Eva intercetta α^x e α^y.
3. Eva invia α^z ad Alice e a Bob. (Alice crede di ricevere α^y e Bob crede di ricevere α^x.)

4. Eva calcola $K_{AE} \equiv (\alpha^x)^z \pmod p$ e $K_{EB} \equiv (\alpha^y)^z \pmod p$. Anche Alice, non sapendo che Eva è in mezzo, calcola K_{AE}, mentre Bob calcola K_{EB}.
5. Quando Alice manda a Bob un messaggio cifrato con K_{AE}, Eva lo intercetta, lo decifra e lo cifra nuovamente usando K_{EB}, dopodiché lo manda a Bob. Bob decifra con K_{EB} e ottiene il messaggio in chiaro. Bob non ha motivo di credere che la comunicazione non sia sicura. Nel frattempo Eva può leggere tutte le informazioni interessanti che ha ottenuto.

Per evitare l'attacco dell'intruso, è desiderabile avere una procedura di mutua autenticazione delle identità di Alice e Bob durante la formazione della chiave. Un protocollo con questa funzionalità è chiamato protocollo di **pattuizione della chiave con autenticazione** (*authenticated key agreement*).
Un modo standard per bloccare l'attacco dell'intruso, è il protocollo **station-to-station (STS)**, che usa le firme digitali. Ogni utente, U, sceglie una funzione di firma digitale sig_U e il relativo algoritmo di verifica ver_U. Per esempio, sig_U può essere una firma RSA oppure di ElGamal. Gli algoritmi di verifica sono complilati e resi pubblici da Trent, un'autorità fidata. L'autorità fidata certifica che ver_U sia proprio l'algoritmo di verifica di U e non, per esempio, di Eva.
Per stabilire una chiave di sessione K da usare con l'algoritmo di cifratura E_K, Alice e Bob procedono come nello scambio di Diffie-Hellman, estendendolo in modo da sfruttare le loro firme digitali.

1. Scelgono un numero primo grande p e una radice primitiva α.
2. Alice sceglie un numero casuale x e Bob sceglie un numero casuale y.
3. Alice calcola $\alpha^x \pmod p$ e Bob calcola $\alpha^y \pmod p$.
4. Alice invia α^x a Bob.
5. Bob calcola $K \equiv (\alpha^x)^y \pmod p$.
6. Bob manda α^y e $E_K(sig_B(\alpha^y, \alpha^x))$ ad Alice.
7. Alice calcola $K \equiv (\alpha^y)^x \pmod p$.
8. Alice decifra $E_K(sig_B(\alpha^y, \alpha^x))$ e ottiene $sig_B(\alpha^y, \alpha^x)$.
9. Alice chiede a Trent se ver_B è l'algoritmo di verifica di Bob.
10. Alice usa ver_B per verificare la firma di Bob.
11. Alice invia $E_K(sig_A(\alpha^x, \alpha^y))$ a Bob.
12. Bob decifra, chiede a Trent se ver_A è l'algoritmo di verifica di Alice e usa ver_A per verificare la firma di Alice.

Questo protocollo è dovuto a Diffie, van Oorschot e Wiener. Alice e Bob sono sicuri che stanno usando la stessa chiave K perché è improbabile che, decifrando con una chiave non corretta, si ottenga una firma valida.
Si noti il ruolo che gioca nel protocollo l'autorità fidata. Alice e Bob devono fidarsi della risposta di Trent, se vogliono comunicare in modo sicuro. Nel resto del capitolo vedremo che l'autorità fidata gioca un ruolo importante in molti protocolli.

10.2 Distribuzione della chiave

Abbiamo fin qui discusso vari concetti crittografici e, in particolare, gli algoritmi per le comunicazioni sicure. Però, un algoritmo crittografico è tanto robusto quanto la sicurezza delle sua chiavi. Se Alice dovesse trasmettere pubblicamente la sua chiave prima di una sessione DES, tutti potrebbero intercettare la comunicazione. Nonostante la sua assurdità, questo scenario è un caso estremo di un problema concreto: se Alice e Bob non possono incontrarsi di persona per scambiarsi le chiavi, come possono accordarsi su una chiave senza compromettere le comunicazioni future?

In particolare, nella crittografia simmetrica il trasmettitore e il ricevitore usano la stessa chiave, quindi devono condividere un'informazione segreta. Per contro, nella crittografia a chiave pubblica, come RSA, il trasmettitore e il ricevitore usano chiavi diverse.

Nei protocolli di instaurazione di chiavi (*key establishment protocols*) Alice e Bob eseguono usa sequenza di passi che portano alla costituzione di quel segreto condiviso necessario per l'instaurazione della chiave. Il fatto che, nella crittografia a chiave pubblica, le chiavi siano memorizzate in basi di dati pubbliche potrebbe far pensare che la crittografia a chiave pubblica risolva facilmente il problema dell'instaurazione della chiave. Questo è vero solo in parte. Il principale inconveniente degli algoritmi crittografici a chiave pubblica è la loro lentezza computazionale in confronto agli algoritmi a chiave simmetrica. L'algoritmo RSA, per esempio, richiede un elevamento a potenza, che è più lento della sostituzione e permutazione di bit che avviene in DES. Quindi, tipicamente si usa RSA per trasmettere una chiave DES che sarà usata, a sua volta, per trasmettere i dati veri e propri. Ci sono tuttavia scenari in cui dobbiamo considerare altri metodi per lo scambio o l'instaurazione di chiavi simmetriche, per esempio lo scenario in cui un server centrale debba comunicare rapidamente con numerosi client e necessiti di metodi di instaurazione della chiave più veloci dei migliori algoritmi noti a chiave pubblica.

Ci sono due tecniche fondamentali di instaurazione della chiave. Nei protocolli di **pattuizione della chiave** (*key agreement*) la chiave non è nota in anticipo ai partecipanti, ma è il risultato delle operazioni compiute durante l'esecuzione del protocollo. Nei protocolli di **distribuzione della chiave** (*key distribution*), un partecipante decide la chiave e la trasmette agli altri.

Lo scambio chiavi di Diffie-Hellman (vedi i Paragrafi 7.4 e 10.1) è un esempio di protocollo di pattuizione della chiave, mentre l'uso di RSA per trasmettere una chiave DES è un esempio di protocollo di distribuzione della chiave.

In tutti i protocolli di instaurazione della chiave, l'autenticazione e gli attacchi di intruso nel mezzo sono i principali problemi di sicurezza da affrontare. Vedremo prima la soluzione più semplice, cioè la predistribuzione. Una seconda soluzione è l'uso di un server che gestisca la distribuzione sicura delle chiavi alle due entità che vogliono comunicare. Accenneremo anche ad alcuni semplici protocolli che coinvolgono un terzo partecipante. Verso la fine del capitolo tratteremo alcune soluzioni di uso pratico per le comunicazioni in Internet.

10.2.1 Predistribuzione della chiave

Nella versione più semplice di questo protocollo, Alice e Bob decidono in anticipo le chiavi o lo *schedule* delle chiavi (ovvero una lista di chiavi con l'indicazione di quando debbano essere usate) e, in qualche modo, si scambiano queste informazioni in modo sicuro. Per esempio, questo metodo era usato dalla Marina tedesca durante la Seconda Guerra Mondiale. Tuttavia, gli Inglesi, avendo a disposizione i libri dei codici conservati nelle navi catturate, riuscirono a trovare le chiavi del giorno e decifrare i messaggi.

Ci sono alcune ovvie limitazioni e controindicazioni alla predistribuzione. Primo, richiede che i due partecipanti, Alice e Bob, si siano incontrati in precedenza, oppure abbiano già instaurato un canale sicuro. Secondo, in caso di compromissione della chiave, Alice e Bob non possono più comunicare e devono necessariamente incontrarsi ancora per cambiare la chiave. Più una chiave è stata usata, maggiore è la mole di dati disponibile per la crittoanalisi o per attacchi statistici e maggiore è il rischio di compromissione. Essendo le chiavi predeterminate, non c'è un modo semplice per cambiare una chiave che sia stata molto usata.

Uno scenario leggermente modificato richiede l'uso di un'autorità fidata, che chiameremo Trent. Per ogni coppia di utenti, (A,B), Trent genera una chiave casuale K_{AB} da usare in un algoritmo a chiave simmetrica (quindi $K_{AB} = K_{BA}$). Supponiamo che Trent abbia a disposizione una grande capacità di calcolo e anche un canale sicuro verso ogni utente. Se Trent è responsabile di n utenti, dovrà inviare $n(n-1)/2$ chiavi e ogni utente riceverà $n-1$ chiavi. Se n è grande, sia la trasmissione sia l'occupazione di memoria possono diventare un problema.

Una tecnica per ridurre la quantità di informazioni che deve inviare l'autorità fidata è lo **schema di predistribuzione di Blom**. Supponiamo di avere una rete di n utenti, e sia p un numero primo grande noto a tutti e tale che $p \geq n$. I passi del protocollo sono i seguenti.

1. A ogni utente U della rete è assegnato un numero pubblico distinto $r_U \pmod p$.
2. Trent sceglie tre numeri segreti $a, b, c \pmod p$.
3. Per ogni utente U, Trent calcola i numeri
$$a_U \equiv a + br_U \pmod p$$
$$b_U \equiv b + cr_U \pmod p$$
e li invia a U tramite il canale sicuro.
4. Ogni utente U costruisce il polinomio lineare
$$g_U(x) = a_U + b_U x.$$
5. Se Alice (A) vuole comunicare con Bob (B), calcola $K_{AB} = g_A(r_B)$, mentre Bob calcola $K_{BA} = g_B(r_A)$.
6. Si può dimostrare che $K_{AB} = K_{BA}$ (Esercizio 2). Alice e Bob comunicano usando un algoritmo a chiave simmetrica usando K_{AB} oppure una chiave derivata da K_{AB}.

Esempio. Si consideri una rete di tre utenti Alice, Bob e Carlo. Sia $p = 23$ e siano

$$r_A = 11, \quad r_B = 3, \quad r_C = 2.$$

Si supponga che Trent scelga i numeri $a = 8$, $b = 3$, $c = 1$, cui corrispondono i seguenti polinomi lineari:

$$g_A(x) = 18 + 14x, \quad g_B(x) = 17 + 6x, \quad g_C(x) = 14 + 5x.$$

È quindi possibile calcolare le chiavi usate in questo schema:

$$K_{AB} = g_A(r_B) = 14, \quad K_{AC} = g_A(r_C) = 0, \quad K_{BC} = g_B(r_C) = 6.$$

È facile verificare che $K_{AB} = K_{BA}$, ecc. ■

Se i due utenti Eva e Oscar collaborano, possono determinare a, b e c e quindi trovare i numeri a_A, b_A per tutti gli utenti. Eva e Oscar conoscono i numeri a_E, b_E, a_O, b_O. Le equazioni che definiscono gli ultimi tre numeri possono essere scritte in forma matriciale:

$$\begin{pmatrix} 0 & 1 & r_E \\ 1 & r_O & 0 \\ 0 & 1 & r_O \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} \equiv \begin{pmatrix} b_E \\ a_O \\ b_O \end{pmatrix} \pmod{p}.$$

Il determinante della matrice è $r_E - r_O$. Poiché i numeri r_A sono distinti mod p, il determinante è diverso da zero mod p e il sistema ha una sola soluzione per a, b, c. Senza l'aiuto di Eva, Oscar avrebbe a disposizione solo una matrice 2×3 e non potrebbe trovare a, b, c. Supponiamo che voglia calcolare la chiave K_{AB} usata da Alice e Bob. Poiché $K_{AB} \equiv a + b(r_A + r_B) + c(r_A r_B)$ (vedi Esercizio 2), potrebbe scrivere la matrice

$$\begin{pmatrix} 1 & r_A + r_B & r_A r_B \\ 1 & r_O & 0 \\ 0 & 1 & r_O \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} \equiv \begin{pmatrix} K_{AB} \\ a_O \\ b_O \end{pmatrix} \pmod{p}.$$

La matrice ha determinante $(r_O - r_A)(r_O - r_B) \not\equiv 0 \pmod{p}$. Pertanto, c'è una soluzione a, b, c per ogni possibile valore di K_{AB}. In altre parole, Oscar non ottiene nessuna informazione sul valore di K_{AB}.
Per ogni valore di $k \geq 1$ esistono schemi di Blom che sono sicuri contro coalizioni di al più k utenti, ma che soccombono contro un complotto di $k + 1$ utenti. Vedi [Blom].

10.2.2 Distribuzione della chiave con autenticazione

Gli schemi di predistribuzione della chiave sono poco pratici perché richiedono parecchie risorse per l'inizializzazione e non permettono di cambiare facilmente una chiave non più ritenuta sicura. Un modo per superare questi ostacoli è l'introduzione di un'**autorità fidata**, il cui compito è la distribuzione delle nuove chiavi ai partecipanti che ne hanno bisogno. Questo terzo partecipante può essere un server in una rete di computer oppure un'organizzazione cui sia Alice sia Bob si affidano per la distribuzione sicura delle chiavi.
L'autenticazione è un aspetto critico della distribuzione della chiave. Alice e Bob ottengono le chiavi da un'autorità fidata, Trent, e devono essere sicuri che le chiavi

non siano contraffazioni provenienti da un'entità fraudolenta che si spaccia per Trent. Inoltre, quando Alice e Bob scambiano messaggi tra di loro devono essere sicuri che il loro interlocutore sia esattamente la persona che essi pensano che sia.
Uno dei principali problemi che deve affrontatare uno schema di distribuzione è l'attacco di replica (*replay attack*). In questo attacco, un avversario registra un messaggio e lo ripete in un secondo momento nella speranza di fingersi un altro oppure di sollecitare una particolare risposta da una entità in modo da compromettere la sicurezza di una chiave. Per fornire l'autenticazione e la protezione contro gli attacchi di replica, dobbiamo essere sicuri che le informazioni vitali, quali le chiavi e i parametri di identificazione, rimangano segreti. Inoltre, dobbiamo garantire che tutti i messaggi siano attuali; cioé non siano ripetizioni di vecchi messaggi.
La segretezza delle informazioni può essere ottenuta facilmente usando le chiavi esistenti e già condivise tra i partecipanti. Queste chiavi sono usate per cifrare i messaggi usati nella distribuzione delle chiavi di sessione e, pertanto, sono chiamate chiavi di cifratura delle chiavi (*key encryption keys*). Sfortunatamente, comunque lo si guardi, si tratta del problema dell'uovo e della gallina: per distribuire le chiavi in modo sicuro, dobbiamo presupporre che le entità abbiano già una chiave sicura condivisa con l'autorità fidata.
Per affrontare il problema dell'attualità dei messaggi e impedire gli attacchi di replica occorre tipicamente introdurre dei campi aggiuntivi nei messaggi scambiati. Generalmente, si usano i seguenti tipi di campo.

- **Numero di sequenza:** ogni messaggio scambiato tra le due entità trasporta un numero di sequenza. Se un'entità riceve un messaggio con un numero di sequenza già usato, lo identifica subito come una replica. Il problema di questo approccio è che richiede alle entità di tenere traccia di tutti i numeri usati.
- ***Timestamp*:** ogni messaggio tra due entità trasporta un'indicazione del periodo di validità del messaggio stesso. Questo approccio richiede che le due entità abbiano orologi ragionevolmente sincronizzati.
- ***Nonce*:** un nonce è un messaggio casuale che cambia a ogni esecuzione del protocollo ed è usato all'interno di uno schema di tipo sfida e risposta (*challenge-response*). In questo schema, Alice invia a Bob un messaggio contenente il nonce e richiede a Bob di elaborarlo e inviare la risposta corretta.

I due esempi che seguono sono schemi di distribuzione della chiave con i relativi attacchi che possono essere usati per violare la sicurezza del protocollo. Gli esempi hanno lo scopo di mettere in evidenza la difficoltà della distribuzione sicura delle chiavi.
Il primo protocollo è noto come **protocollo *wide-mouthed frog*** (letteralmente: "rana dalla bocca larga") ed è uno dei più semplici protocolli di distribuzione della chiave basati su crittografia simmetrica che coinvolgono un'autorità fidata. In questo protocollo, Alice sceglie una chiave di sessione K_{AB} per comunicare con Bob e chiede a Trent di trasferirla a Bob in modo sicuro.

1. Alice $\rightarrow$ Trent: $E_{K_{AT}}[t_A \| ID_B \| K_{AB}]$.
2. Trent $\rightarrow$ Bob: $E_{K_{BT}}[t_T \| ID_A \| K_{AB}]$.

In questo schema, K_{AT} è la chiave condivisa tra Alice e Trent, mentre K_{BT} è la chiave condivisa tra Bob e Trent. Gli identificativi di Alice e Bob sono rispettivamente ID_A e ID_B. Il parametro t_A è il timestamp fornito da Alice, mentre t_T è il timestamp fornito da Trent. Ipotizzando che Alice, Trent e Bob abbiano orologi sincronizzati, Bob accetta K_{AB} come attuale solo se arriva entro una certa finestra temporale. La chiave K_{AB} è valida solo per un certo periodo di tempo a partire da t_T.
Lo scopo dei due timestamp è permettere a Bob di verificare l'attualità del messaggio. Nel primo messaggio, Alice invia il timestamp t_A. Se Trent riceve il messaggio entro un tempo non troppo lontano da t_A, accetta di inoltrare il messaggio a Bob.
Il problema del protocollo è nel secondo messaggio, in cui Trent aggiorna il timestamp al nuovo valore t_T. Sfortunatamente, questa semplice modifica permette a Mallory di compiere un attacco il cui scopo è indurre Trent a estendere la durata di una vecchia chiave.

1. Dopo avere visto una esecuzione del protocollo, Mallory impersona Bob e finge di voler condividere una chiave con Alice. Mallory invia a Trent una replica del vecchio messaggio $E_{K_{BT}}[t_T \| ID_A \| K_{AB}]$.
2. Trent invia $E_{K_{AT}}\left[t_T' \| ID_B \| K_{AB}\right]$ ad Alice, con un nuovo timestamp t_T'. Alice pensa che sia un messaggio valido, perché viene da Trent ed è cifrato con la chiave di Trent. Il periodo di validità della chiave K_{AB} comincia al tempo t_T'.
3. Mallory si finge Alice e ottiene $E_{K_{BT}}\left[t_T'' \| ID_A \| K_{AB}\right]$. Ora il periodo di validità della chiave K_{AB} comincia al tempo $t_T'' > t_T'$.
4. Mallory continua a ingannare Trent fingendosi ora Bob ora Alice.

Il risultato netto è che Mallory può manipolare Trent e usarlo per indurre Alice e Bob a usare la stessa chiave K_{AB} indefinitamente. Ovviamente Alice e Bob dovrebbero tenere traccia delle chiavi che hanno usato in precedenza e dovrebbero insospettirsi vedendo ripetutamente la stessa chiave. Tuttavia il protocollo non dice esplicitamente che Alice e Bob devono effettuare questo controllo, mentre i protocolli sicuri devono sempre essere molto espliciti quanto a presupposti. D'altra parte, il vero problema è che Trent aggiorna t_A con t_T. Se Trent avesse lasciato il timestamp t_A, il protocollo non avrebbe avuto questo problema.
Un altro esempio di protocollo di distribuzione della chiave con autenticazione è dovuto a Needham e Schroeder. Nel protocollo di Needham-Shroeder, Alice e Bob ottengono da Trent una chiave di sessione per comunicare tra di loro. Il protocollo è composto dai seguenti passi.

1. Alice → Trent: $ID_A \| ID_B \| r_1$.
2. Trent → Alice: $E_{K_{AT}}[K_S \| ID_B \| r_1 \| E_{K_{BT}}[K_S \| ID_A]]$.
3. Alice → Bob: $E_{K_{BT}}[K_S \| ID_A]$.
4. Bob → Alice: $E_{K_S}[r_2]$.
5. Alice → Bob: $E_{K_S}[r_2 - 1]$.

Esattamente come nel protocollo visto prima, K_{AT} è la chiave condivisa tra Alice e Trent, mentre K_{BT} è la chiave condivisa tra Bob e Trent. A differenza del protocollo wide-mouthed frog, il protocollo di Needham-Schroeder non usa i timestamp, ma i nonce, ovvero i numeri casuali r_1 e r_2. Nel primo passo, Alice invia a Trent la sua richiesta, che contiene la dichiarazione della propria identità, dell'identità del partecipante con cui vuole comunicare e il numero casuale r_1. Trent dà ad Alice la chiave di sessione K_S e la sequenza $E_{K_{BT}}[K_S \| ID_A]$, che Alice consegna a Bob nel passo successivo. Bob decifra il messaggio di Alice e recupera la chiave di sessione e l'indentità del suo interlocutore. Può quindi inviare ad Alice la sua sfida, ovvero invia il secondo nonce r_2. Nell'ultimo passo, Alice dimostra la propria identità a Bob rispondendo alla sfida. (In questo messaggio Alice invia $r_2 - 1$ invece di r_2 per impedire a Mallory di usare una replica del messaggio 4.)
Si noti che la vera e propria distribuzione della chiave è completa alla fine del terzo messaggio. Gli ultimi due messaggi, apparentemente superflui, servono a impedire attacchi di replica in cui Mallory invia a Bob una vecchia sequenza $E_{K_{BT}}[K_S \| ID_A]$. Se non ci fossero i messaggi 4 e 5, Bob concluderebbe che K_S è la chiave da usare. In questo modo Mallory indurrebbe Bob a inviare ad Alice altri messaggi usando K_S. I passi 4 e 5 permettono a Bob di chiedere ad Alice di dimostrare di conoscere veramente la chiave di sessione K_S. Solo Alice è in grado di calcolare $E_{K_S}[r_2 - 1]$.
Nonostante l'apparente sicurezza fornita dallo schema di sfida e risposta, c'è un potenziale problema di sicurezza nel caso in cui Mallory scopra la chiave di sessione K_S. Mallory esegue il seguente attacco.

1. Mallory → Bob: $E_{K_{BT}}[K_S \| ID_A]$
2. Bob → Alice: $E_{K_S}[r_3]$
3. Mallory → Bob: $E_{K_S}[r_3 - 1]$

In questo attacco, Mallory replica un vecchio messaggio numero 3 del protocollo Needham-Schroeder, comportandosi come se fosse Alice. Quando Bob riceve il messaggio, invia ad Alice la propria sfida, consistente nel nonce r_3. Mallory può intercettare la sfida e, conoscendo K_S, rispondere correttamente. Il risultato netto è che Mallory supera i controlli di autenticazione di Bob come se fosse Alice. Da questo momento in poi, Bob comunicherà usando K_S credendo che il suo interlocutore sia Alice. Mallory ha così rubato l'identità di Alice.
È difficile costruire un protocollo di distribuzione della chiave che sia resistente. La letteratura contiene numerosi esempi di protocolli fallimentari perché, alcuni anni dopo, si sono trovati degli attacchi. Potrebbe anche sembrare una causa persa, perché entrambi i protocolli esaminati hanno punti deboli. Tuttavia il resto del capitolo tratta protocolli che sono ritenuti, a oggi, sicuri. Il primo esempio è Kerberos, che nasce come un miglioramento di Needham-Shroeder. Fino a oggi, Kerberos ha resistito a esami approfonditi da parte della comunità scientifica ed è adottato in molte applicazioni.

10.3 Kerberos

Kerberos (che prende il nome dal cane con tre teste a guardia dell'ingresso dell'Ade) è un'implementazione reale di un protocollo basato sulla crittografia simmetrica il

cui scopo è fornire elevati livelli di autenticazione e sicurezza nella distribuzione della chiave tra utenti in una rete. Nel seguito useremo il termine *utenti* molto liberamente, per indicare sia individui sia programmi che vogliono comunicare con altri programmi. Kerberos nasce dal più ampio progetto Athena, sviluppato presso il MIT, che ha lo scopo di fornire agli studenti un'enorme rete di computer che permetta loro di accedere facilmente ai file personali da qualunque punto della rete. Come prevedibile, un progetto di questo tipo incontra rapidamente problemi di sicurezza. In particolare, la comunicazione attraverso una rete pubblica come Athena non è sicura. Osservare i dati in transito nella rete e cercare informazioni interessanti come password o altre informazioni private è facile. Kerberos è stato sviluppato con lo scopo di affrontare questi problemi. Nel seguito vedremo il modello fondamentale di Kerberos e descriveremo cos'è e cosa cerca di fare. Per una descrizione più approfondita, vedi [Schneier].
Kerberos è basato su un'architettura client/server. Un client può essere un utente oppure un programma che deve compiere delle operazioni, per esempio inviare e-mail, stampare documenti o montare dispositivi. I server sono entità la cui funzione è fornire servizi ai client. Per esempio, i domain name server (DNS) di Internet hanno il compito di fornire nomi e indirizzi ai client come i programmi di posta elettronica oppure i web browser.
Il modello fondamentale di Kerberos ha i seguenti partecipanti:

- Cliff: un client
- Serge: un server
- Trent: un'autorità fidata, detta anche "server di autenticazione"
- Grant: un ticket-granting server (letteralmente: un server fornitore di biglietti).

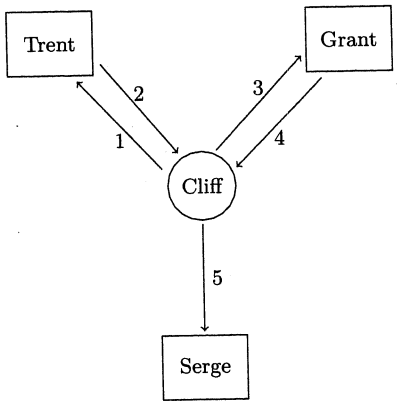

Figura 10.1 Kerberos.

All'inizio Cliff e Serge non condividono una chiave segreta e Kerberos ha lo scopo di fornirla loro in modo sicuro. Un risultato del protocollo Kerberos è che Serge avrà verificato l'identità di Cliff (in modo da non conversare con un falso Cliff) e sarà stabilita una chiave di sessione.
Il protocollo, mostrato in Figura 10.1, inizia con Cliff che chiede un ticket al servizio di ticket-granting di Trent. Trent, che è l'autorità fidata, ha il database di tutte le password di tutti i client (per questo motivo Trent viene anche chiamato server Kerberos). Trent restituisce un ticket cifrato con la password segreta del client. A questo punto Cliff vorrebbe usare il servizio fornito da Serge, ma prima deve essere autorizzato. Pertanto Cliff presenta il suo ticket a Grant, il ticket-granting server. Grant prende il ticket che contiene le informazioni necessarie per identificare Cliff e, se è tutto a posto, dà a Cliff un nuovo ticket che gli permetterà di accedere al servizio di Serge e solo a quello; il ticket non è valido per un altro server. Cliff invia a Serge il ticket per il servizio insieme alle credenziali di autenticazione. Serge confronta il ticket con le credenziali per essere sicuro che sia valido; se corrispondono, Serge fornisce il servizio a Cliff. Il protocollo Kerberos non è altro che una versione formale di protocolli usati nella vita quotidiana, in cui diverse entità sono coinvolte nell'autorizzazione di diversi passaggi di una procedura; per esempio prelevare banconote a un Bancomat e usarle per acquistare un biglietto per i mezzi pubblici.
Kerberos usa algoritmi a chiave simmetrica. Nella versione V, Kerberos usa DES in modalità CBC; tuttavia qualunque algoritmo a chiave simmetrica va bene.

1. Cliff a Trent: Cliff invia a Trent un messaggio che contiene il proprio nome e il nome del ticket-granting server che userà (in questo caso Grant).

2. Trent a Cliff: Trent cerca il nome di Cliff nel proprio database. Se lo trova, genera una chiave di sessione K_{CG} che sarà usata tra Cliff e Grant. Trent cifra la chiave di sessione con la chiave segreta K_C che usa per comunicare con Cliff:

$$T = e_{K_C}(K_{CG}).$$

   Inotre, Trent crea un Ticket Granting Ticket (TGT), che permetterà a Cliff di autenticarsi a Grant. Questo ticket è cifrato usando la chiave personale di Grant K_G, nota solo a Grant e Trent:

$$TGT = \text{nome di Grant} \| e_{K_G}(\text{nome di Cliff, indirizzo di Cliff, Timestamp1}, K_{CG}).$$

   Il simbolo $\|$ indica la concatenazione. Il ticket che Cliff riceve è la concatenazione dei due sottoticket

$$\text{Ticket} = T \| TGT.$$

3. Cliff a Grant: Cliff estrae K_{CG} usando la chiave K_C, che condivide con Trent. Usando K_{CG}, Cliff può comunicare con Grant in modo sicuro. A questo punto Cliff crea un autenticatore (*authenticator*), che consiste nel proprio nome, nel proprio indirizzo e in un timestamp e lo cifra usando K_{CG} per ottenere

$$\text{Auth}_{CG} = e_{K_{CG}}(\text{nome di Cliff, indirizzo di Cliff, Timestamp2}).$$

   Cliff invia Auth_{CG} e il TGT a Grant in modo che questi possa generare il service ticket.

4. Grant a Cliff: Grant ora ha Auth_{CG} e il TGT. Parte del TGT è cifrata con la chiave di Grant. Grant la estrate, la decifra e ottiene il nome di Cliff, l'indirizzo di Cliff, il Timestamp1 e K_{CG}. Ora Grant può usare K_{CG} per decifrare Auth_{CG} e ottenere una seconda copia del nome e dell'indirizzo di Cliff e un nuovo timestamp, Timestamp2. Se le due versioni dei dati di Cliff corrispondono e i timestamp Timestamp1 e Timestamp2 sono sufficientemente vicini, Grant ha verificato l'autenticità della richiesta e l'identità di Cliff. A questo punto, Grant genera una chiave di sessione K_{CS} per permettere a Cliff di comunicare con Serge e restituisce un service ticket cifrato usando la chiave segreta K_S che Grant condivide con Serge. Il service ticket è

$$\text{ServTicket} = e_{K_S}(\text{nome di Cliff, indirizzo di Cliff, Timestamp3, ExpirationTime}, K_{CS}).$$

Il valore di ExpirationTime indica la durata della validità del service ticket. La chiave di sessione è cifrata usando la chiave (anch'essa di sessione) tra Cliff e Grant:

$$e_{K_{CG}}(K_{CS}).$$

Grant invia ServTicket e $e_{K_{CG}}(K_{CS})$ a Cliff.

5. Cliff a Serge: Cliff è pronto per usare i servizi di Serge. Inizia decifrando $e_{K_{CG}}(K_{CS})$ per ottenere la chiave di sessione K_{CS} che userà per comunicare con Serge. Quindi crea un autenticatore da usare con Serge:

$$\text{Auth}_{CS} = e_{K_{CS}}(\text{identificativo di Cliff, indirizzo di Cliff, Timestamp4}).$$

Cliff invia a Serge Auth_{CS} e ServTicket. Serge decifra ServTicket ed estrae la chiave di sessione K_{CS} che userà con Cliff. Usando questa chiave, decifra Auth_{CS}, verifica l'identità di Cliff e verifica che Timestamp4 ricada nell'intervallo di validità del service ticket. Se il ticket di Cliff è scaduto, Serge rifiuta il servizio, altrimenti Cliff e Serge possono usare K_{CS} per comunicare.

10.4 Infrastrutture a chiave pubblica

La crittografia a chiave pubblica è uno strumento potente che permette l'autenticazione, la distribuzione della chiave e impedisce il ripudio. In queste applicazioni, la chiave è pubblica, ma che garanzie abbiamo che la chiave pubblica di Alice sia veramente di Alice? Eva potrebbe averla sostituita con la propria chiave. Se non c'è sicurezza sulla modalità di generazione della chiave e sulla sua autenticità e validità, i benefici della crittografia a chiave pubblica sono minimi.
Perché la crittografia a chiave pubblica sia utile nelle applicazioni commerciali, è necessario avere un'infrastruttura che garantisca la validità delle chiavi pubbliche e ne regoli l'uso. Un'infrastruttura a chiave pubblica (*Public Key Infrastructure, PKI*) è un insieme di direttive che definiscono le regole secondo le quali devono operare i sistemi crittografici e che definiscono le procedure per la generazione e pubblicazione di chiavi e certificati.

Tutte le PKI definiscono le operazioni di certificazione e validazione. La certificazione lega una chiave pubblica a un'entità, che può essere un utente oppure un'informazione. La validazione garantisce che i certificati siano validi.
Un **certificato** è un insieme di informazioni firmate dal soggetto che le ha pubblicate, generalmente chiamato **autorità di certificazione** (*Certification Authority, CA*). I certificati possono essere di vario tipo. I due più comuni sono i certificati di identità e i certificati credenziali. I certificati di indentità contengono informazioni sull'identità di un'entità, per esempio l'indirizzo di posta elettronica e una lista di chiavi pubbliche dell'entità. I certificati credenziali contengono informazioni relative ai diritti di accesso a una risorsa. In entrambi i casi i dati sono cifrati usando la chiave privata della CA.
Si supponga che, nel contesto di una PKI, la CA pubblichi i certificati di Alice e Bob. Se Alice conosce la chiave pubblica della CA, allora può recuperare il certificato d'identità di Bob, estrarre l'informazione d'identità di Bob e una lista di chiavi pubbliche da usare per comunicare con Bob in modo sicuro. La differenza tra questo scenario e lo scenario convenzionale della crittografia a chiave pubblica consiste nel fatto che non è Bob a pubblicare la chiave, ma la CA, quindi la relazione di fiducia è posta tra Alice e la CA. Alice potrebbe non fidarsi di Bob, ma fidarsi di una CA istituzionale come il governo oppure la compagnia telefonica. Il concetto di fiducia è critico per la PKI ed è forse la sua proprietà più importante.
È difficile che una singola entità possa tenere traccia e pubblicare tutte le chiavi pubbliche degli utenti Internet. Per questo, le PKI sono spesso composte da molteplici CA che si certificano tra di loro ed emettono certificati per gli utenti. Pertanto Bob potrebbe essere associato a una CA diversa da quella di Alice e, nel richiedere il certificato di identità di Bob, Alice potrebbe fidarsi solo se la sua CA si fida della CA di Bob. In grandi reti, come Internet, potrebbero esserci molte CA tra Alice e Bob ed è necessario che tutte le CA si fidino l'una dell'altra.
Inoltre, molte PKI prevedono diversi livelli di fiducia, che si traducono nel fatto che alcune CA si fidano di altre CA solo nel contesto di alcune operazioni. Per esempio, la CA di Alice potrebbe fidarsi della CA di Bob solo relativamente all'emissione di certificati di utenti e non relativamente alla certificazione di altre CA, mentre la CA di Alice potrebbe fidarsi della CA di Davide relativamente alla certificazione di altre CA. Le relazioni di fiducia possono diventare molto elaborate e, al crescere della complessità, diventa più difficile determinare il grado di fiducia che Alice può avere nei confronti del certificato che ha ricevuto.
Di seguito discuteremo due esempi di PKI usate in pratica.

10.5 Certificati X.509

Supponiamo di voler comprare un articolo su Internet. Apriamo il sito `gigafirm.com`, scegliamo i prodotti e procediamo alla cassa dove ci vengono chiesti il numero di carta di credito e altre informazioni. Il sito assicura di usare la crittografia a chiave pubblica per proteggere le comunicazioni, ma come possiamo essere sicuri che Eva non abbia sostituito la propria chiave pubblica a quella del negoziante? In altre parole, quando usiamo le chiavi pubbliche, come possiamo essere sicuri della loro correttezza? I certificati digitali risolvono questo problema.

Il tipo di certificato più comune segue lo standard X.509. In questo sistema, ogni utente ha un certificato la cui validità dipende da una catena di fiducia. In cima ci sono le **autorità di certificazione** (*certification authority*, CA), che sono generalmente imprese commerciali, quali Verisign, GTE, AT&T e altre. Si presuppone che la CA sia fidata. Ogni CA emette il proprio certificato e lo firma. Questo certificato viene generalmente pubblicato sul sito Internet dell'autorità stessa, inoltre, per favorirne la diffusione, molte CA si sono accordate in modo da far includere i loro certificati nei browser Internet. Dietro compenso, la CA emette anche i certificati per i suoi clienti. Il certificato del cliente Gigafirm contiene la chiave pubblica di Gigafirm ed è firmato dalla CA usando la chiave privata della CA. Per ragioni di efficienza, spesso la CA delega la firma dei certificati a una o più autorità di registrazione (*registration authority*, RA).[1] Ogni RA ha un proprio certificato, che è firmato dalla CA.
In alcune circostanze, chi ha un certificato può, a sua volta, firmare certificati di altri. In questo modo si ottiene una **gerarchia di certificazione** in cui la validità di ogni certificato è garantita da un altro utente, fino ad arrivare alla CA. La Figura 10.2 mostra un esempio di gerarchia di certificazione.

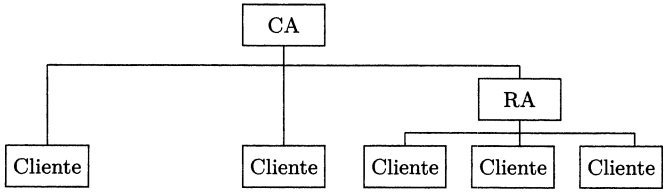

Figura 10.2 Una gerarchia di certificazione.

Se Alice vuole verificare la correttezza della chiave pubblica di Gigafirm, usa la copia del certificato della CA memorizzata sul proprio computer in modo da ottenere la chiave pubblica della CA, quindi verifica la firma sul certificato di Gigafirm. Se è valida, Alice si fida del certificato e ha una chiave pubblica fidata per Gigafirm. Ovviamente, Alice deve fidarsi della chiave pubblica della CA, quindi deve fidarsi della compagnia che le ha fatto avere il certificato della CA che è memorizzato sul proprio computer. Questa compagnia è incentivata economicamente a mantenere una buona reputazione, per cui è ragionevole fidarsi. Ma se Alice ha comprato un computer usato in cui Eva ha alterato i certificati, potrebbe esserci un problema (in altre parole, non comprate computer usati dai vostri avversari, se non per estrarne informazioni).
Le Figure 10.3, 10.4 e 10.5 sono esempi di certificati X.509.[2] Quelli in Figura 10.3 e 10.4 sono della CA, in particolare di VeriSign. La parte in Figura 10.3 dà informazioni generali sul certificato, tra le quali i possibili usi. La Figura 10.4 dà le informazioni di dettaglio. Il certificato in Figura 10.5 è una versione semplificata della parte di dettaglio del certificato della banca Wells Fargo.

[1] Un'autorità di registrazione svolge anche molte altre funzioni, per esempio l'identificazione degli utenti e la pubblicazione dei certificati. Quando la RA ha il compito di firmare certificati per conto della CA, viene anche indicata come "CA intermedia". (*N.d.Rev.*)

[2] Le figure mostrano il contenuto dei certificati secondo il formato adottato da Mozilla Firefox Versione 3.0. (*N.d.Rev.*)

Il certificato è stato verificato per i seguenti utilizzi:

Certificato firmatario e-mail
Certificato destinatario e-mail
Certificato di stato del risponditore

Rilasciato a:
Nome Comune (CN): <non incluso nel certificato>
Organizzazione (O): VeriSign, Inc.
Unità organizzativa (OU): Class 1 Public Primary Certification Authority - G2
Numero seriale: 39:CA:54:89:FE:50:22:32:FE:32:D9:DB:FB:1B:84:19

Rilasciato da:
Nome Comune (CN): <non incluso nel certificato>
Organizzazione (O): VeriSign, Inc.
Unità organizzativa (OU): Class 1 Public Primary Certification Authority - G2

Validità:
Rilasciato il: 17/05/1998
Scade il: 18/05/2018

Impronte digitali: [illegible] come ID del certificato (a l'ho?)
Impronta digitale SHA1: 04:98:11:05:6A:FE:9F:D0:F5:BE:01:68:5A:AC:E6:A5:D1:C4:45:4C
Impronta digitale MD5: F2:7D:E9:54:E4:A3:22:0D:76:9F:E7:0B:BB:B3:24:2B

Figura 10.3 Certificato della CA; informazioni generali.

Questi sono alcuni dei campi della Figura 10.4.

1. *Versione:* ci sono tre versioni, la prima (*Versione 1*) è del 1988, la più recente (*Versione 3*) è del 1997.

2. *Numero seriale:* ogni certificato emesso da una CA è contraddistinto da un numero seriale.

3. *Algoritmo firma certificato:* Si possono usare diversi algoritmi di firma. Questo certificato usa *RSA* per firmare l'output della funzione hash *SHA-1*.

4. *Emittente:* il nome della CA che ha generato e firmato il certificato. *OU* è l'unità entro l'organizzazione, *O* è l'organizzazione, *C* è il paese.

5. *Soggetto:* il nome del titolare di questo certificato.

6. *Chiave pubblica soggetto:* sono possibili diverse opzioni. Questo usa RSA con un modulo di 1024 bit. La chiave è espressa in notazione esadecimale: le lettere `a, b, c, d, e, f` rappresentano i numeri `10, 11, 12, 13, 14, 15`. Ogni coppia di simboli è un byte (8 bit). Per esempio, `b6` rappresenta `11, 6`, che è `10110110` in binario.

   Gli ultimi tre byte della chiave pubblica sono `01 00 01`, che sono $65537 = 2^{16}+1$. Si tratta di un esponente e di cifratura RSA molto comune, perché elevare una base a questa potenza effettuando quadrati successivi è molto veloce (vedi Paragrafo 3.5). I byte precedenti `02 03` e i byte `30 81 89 02 81 81 00` all'inizio della chiave sono simboli di controllo. I restanti 128 byte `aa d0 ba ··· 6b e7 75` sono il modulo RSA n di 1024 bit.

Gerarchia certificato

▷ Verisign Class 1 Public Primary Certification Authority - G2

Campi certificato

```
Verisign Class 1 Public Primary Certification Authority - G2
  Certificato
    Versione: Versione 1
    Numero seriale: 39:CA:54:89:FE:50:22:32:FE:32:D9:DB:FB:1B:84:19
    Algoritmo firma certificato: PKCS #1 SHA-1 con cifratura RSA
    Emittente: OU = VeriSign Trust Network
          OU = (c) 1998 VeriSign, Inc. - For authorized use only
          OU = Class 1 Public Primary Certification Authority - G2
          O = VeriSign, Inc.
          C = US
  Validità
    Non prima: 17/05/98 20:00:00 (05/18/98 00:00:00 GMT)
    Non dopo: 18/05/18 19:59:59 (05/18/18 23:59:59 GMT)
  Soggetto: OU = VeriSign Trust Network
          OU = (c) 1998 VeriSign, Inc. - For authorized use only
          OU = Class 1 Public Primary Certification Authority - G2
          O = VeriSign, Inc.
          C = US
    Informazioni chiave pubblica soggetto: PKCS #1 cifratura RSA
    Chiave pubblica del soggetto:
          30 81 89 02 81 81 00 aa d0 ba be 16 2d b8 83 d4
          ca d2 0f bc 76 31 ca 94 d8 1d 93 8c 56 02 bc d9
          6f 1a 6f 52 36 6e 75 56 0a 55 d3 df 43 87 21 11
          65 8a 7e 8f bd 21 de 6b 32 3f 1b 84 34 95 05 9d
          41 35 eb 92 eb 96 dd aa 59 3f 01 53 6d 99 4f ed
          e5 e2 2a 5a 90 c1 b9 c4 a6 15 cf c8 45 eb a6 5d
          8e 9c 3e f0 64 24 76 a5 cd ab 1a 6f b6 d8 7b 51
          61 6e a6 7f 87 c8 e2 b7 e5 34 dc 41 88 ea 09 40
          be 73 92 3d 6b e7 75 02 03 01 00 01
  Algoritmo firma certificato: PKCS #1 SHA-1 con cifratura RSA
  Valore firma certificato:
          8b f7 1a 10 ce 76 5c 07 ab 83 99 dc 17 80 6f 34
          39 5d 98 3e 6b 72 2c e1 c7 a2 7b 40 29 b9 78 88
          ba 4c c5 a3 6a 5e 9e 6e 7b e3 f2 02 41 0c 66 be
          ad fb ae a2 14 ce 92 f3 a2 34 8b b4 b2 b6 24 f2
          e5 d5 e0 c8 e5 62 6d 84 7b cb be bb 03 8b 7c 57
          ca f0 37 a9 90 af 8a ee 03 be 1d 28 9c d9 26 76
          a0 cd c4 9d 4e f0 ae 07 16 d5 be af 57 08 6a d0
          a0 42 42 42 1e f4 20 cc a5 78 82 95 26 38 8a 47
```

Figura 10.4 Certificato della CA; dettagli.

Gerarchia certificato

▷ Verisign Class 3 Public Primary CA
  ▷ www.verisign.com/CPS Incorp. by Ref. LIABILITY LTD.(c)97VeriSign
    ▷ online.wellsfargo.com

Campi certificato

```
Verisign Class 3 Public Primary Certification Authority
  Certificato
    Versione: Versione 3
    Numero seriale: 03:D7:98:CA:98:59:30:B1:B2:D3:BD:28:B8:E7:2B:8F
    Algoritmo firma certificato: md5RSA
    Emittente: OU = www.verisign.com/CPS Incorp. ...
          OU = VeriSign International Server CA - Class 3
          OU = VeriSign, Inc.
          O = VeriSign Trust Network
          C = US
  Validità
    Non prima: Domenica 21 Settembre 2003 19:00:00
    Non dopo: Mercoledì 21 Settembre 2005 18:59:59
  Soggetto: CN = online.wellsfargo.com
          OU = Terms of use at www.verisign.com.rpa (c)00
          OU = Class 1 Public Primary Certification Authority - G2
          OU = ISG
          O = Wells Fargo and Company
          L = San Francisco
          S = California
          C = US
    Informazioni chiave pubblica soggetto: PKCS #1 RSA Encryption
    Chiave pubblica del soggetto: 30 81 89 02 81 81 00 a9 ...
    Vincoli base del certificato: tipo di soggetto = entità finale,
                                  limite alla lunghezza del percorso = nessuno
    Uso chiave certificato: firma digitale, cifratura di chiavi (AO)
    Punti di distribuzione CRL: (1) punto di distribuzione CRL
                                    Nome del punto di distribuzione:
                                      Nome completo:
                                        URL=http://crl.verisign.com/
                                            class3InternationalServer.crl
  Algoritmo firma certificato: MD5 With RSA Encryption
  Valore firma certificato: ......
```

Figura 10.5 Il certificato di un client.

7. *Valore firma certificato:* si calcola l'hash delle informazioni di questo certificato usando l'algoritmo specificato (in questo caso *SHA-1*) e lo si firma usando l'esponente privato di decifrazione RSA della CA.

Il certificato in Figura 10.5 ha anche qualche altra riga di informazioni. Una di queste è la voce *Gerarchia certificato*. Il certificato della Wells Fargo è stato firmato dalla autorità di registrazione (RA) nella riga sopra, il cui certificato è stato firmato, a sua volta, dalla CA radice. Un'altra voce interessante contiene i punti di distribuzione della lista di revoca dei certificati (*certificate revocation list*, CRL), che contiene i numeri di serie dei certificati revocati. Nessuno dei due modi generalmente usati per distribuire queste liste agli utenti è perfetto.
Il primo modo consiste nel mandare un annuncio ogni volta che un certificato è revocato. Questo ha lo svantaggio di inviare parecchie informazioni irrilevanti per la maggior parte degli utenti (la maggior parte della gente non è interessata a sapere che la palestra Point Barrow ha perso il proprio certificato). Il secondo metodo consiste nel mantenere una lista (come quella alla URL inserita nel certificato) alla quale si può accedere in qualunque momento. Lo svantaggio di questo approccio è il ritardo causato dal controllo di ogni certificato. Inoltre, un tale sito web sarebbe facilmente soggetto a sovraccarico se molte persone cercassero di accedervi contemporaneamente. Per esempio, se tutti compiono operazioni di Borsa nella pausa pranzo e i computer controllano l'eventuale revoca di ogni certificato all'inizio di ogni transazione, il sito sarebbe facilmente sopraffatto.
Quando Alice (o, tipicamente, il suo computer) vuole verificare la validità del certificato in Figura 10.5, dalla gerarchica di certificati scopre che il certificato di Wells Fargo è firmato dalla RA di Verisign, il cui certificato è firmato dalla CA radice. Alice verifica la firma del certificato di Wells Fargo usando la chiave pubblica (cioè la coppia RSA (n,e)) ottenuta dal certificato della RA; specificamente eleva l'hash cifrato contenuto nel certificato alla e-esima potenza mod n. Se il risultato è uguale all'hash calcolato sul certificato (e Alice si fida del certificato della RA), allora si fida del certificato di Wells Fargo. Il certificato della RA può essere verificato allo stesso modo usando la chiave pubblica della CA radice. Alice si fida del certificato della CA radice perché lo ha ottenuto da una sorgente affidabile, per esempio era incluso nel suo Internet browser e la compagnia distributrice gode di buona reputazione. In questo modo, Alice ha stabilito la validità del certificato di Wells Fargo. Perciò può effettuare con sicurezza transazioni elettroniche con Wells Fargo.
Ci sono due livelli di certificati. I certificati ad alta sicurezza (*high assurance*) sono emessi dalla CA in seguito a controlli abbastanza severi. Questi certificati sono distribuiti alle aziende commerciali. I certificati a bassa sicurezza (*low assurance*) sono emessi con maggiore libertà e certificano che le comunicazioni provengono da una particolare sorgente. Pertanto, se Bob ottiene un certificato di questo secondo tipo per il proprio computer, il certificato garantisce che la comunicazione proviene dal computer di Bob, ma non ci dice se il computer è usato da Bob o Eva. I certificati in molti personal computer contengono la seguente riga, che indica che il certificato è a bassa sicurezza. In tal caso, non c'è alcuna garanzia riguardo l'identità dell'utente.

Soggetto: Verisign Class 1 CA Individual Subscriber - Persona Not Validated.

Se usate un computer con installato Internet Explorer, facendo clic su *Strumenti*, quindi su *Opzioni Internet* e infine su *Contenuti*, potrete trovare l'elenco delle CA i cui certificati sono distribuiti con il browser. Per alcuni certificati è possibile guardare il percorso di certificazione (*certification path*), che mostra il percorso dal certificato installato sul computer fino alla CA. Generalmente si tratta di un solo passaggio.

10.6 Pretty Good Privacy

Il software Pretty Good Privacy, generalmente noto come *PGP*, fu sviluppato da Phil Zimmerman tra la fine degli anni Ottanta e i primi anni Novanta.[3]
Diversamente dai certificati X.509, PGP è un sistema decentralizzato senza CA. Ogni utente ha un certificato, ma la fiducia in questo certificato è garantita, a diversi livelli, dagli altri utenti. Questo schema crea una rete di fiducia (*web of trust*).
Per esempio, se Alice conosce Bob e può verificare direttamente che il suo certificato è valido, allora lo firma con la propria chiave privata. Carlo si fida di Alice e conosce la sua chiave pubblica, quindi può controllare che la firma di Alice sul certificato di Bob sia valida. Quindi Carlo si fida del certificato di Bob. Tuttavia, ciò non significa che Carlo si fiderà dei certificati firmati da Bob, ma solo della sua chiave pubblica. Bob potrebbe essere un credulone che firma tutti i certificati che gli sono sottoposti. La sua firma è valida, ma non vuol dire che lo siano i certificati firmati da lui.
Ogni utente, per esempio Alice, mantiene un file portachiavi (*keyring*), che indica i livelli di fiducia di Alice nei confronti delle varie firme che conosce. I possibili livelli sono: nessuna informazione, nessuna fiducia, fiducia parziale, fiducia completa. Nel giudicare la validità di un certificato, PGP accetta certificati firmati da qualcuno di cui Alice si fida in modo completo oppure da almeno un certo numero di utenti di cui Alice si fida parzialmente. In caso negativo, chiede ad Alice di decidere se continuare.
La principale applicazione di PGP è l'autenticazione e la cifratura della posta elettronica. Si supponga che Alice riceva una e-mail che le chiede le sue coordinate bancarie in modo che Carlo le possa trasferire milioni di euro. Alice vuole essere sicura che l'e-mail provenga da Carlo e non da Eva, che vuole usare queste informazioni per una truffa. Nell'improbabile caso che l'e-mail venga da Carlo, Alice invia le informazini bancarie, ma le cifra in modo che Eva non possa intercettarle. Perciò, le prima e-mail necessita di autenticazione, ovvero la garanzia che provenga da Carlo, mentre la seconda e-mail richiede cifratura. PGP è in grado di gestire anche i casi in cui sono desiderabili sia l'autenticazione sia la cifratura. Nel seguito vedremo come PGP gestisce i vari casi. Supporremo che Alice stia inviando un messaggio a Bob.

Autenticazione

1. Alice calcola l'hash del messaggio usando un'opportuna funzione, generalmente *SHA-1*.

[3] Oggi, oltre a PGP esistono altri programmi che usano un approccio simile e che usano chiavi compatibili con PGP. Tra questi si segnala GPG, che è sostenuto dalla Free Software Foundation. Esiste anche uno standard Internet, denominato OpenPGP, che definisce il formato dei messaggi scambiati tra queste applicazioni. (*N.d.Rev.*)

2. Alice firma l'hash elevandolo alla potenza dell'esponente segreto di decifrazione $d \bmod n$ e lo manda a Bob insieme al messaggio.

3. Bob eleva l'hash firmato alla potenza dell'esponente RSA pubblico e. Il risultato è confrontato con l'hash del resto del messaggio.

4. Se i due valori concordano, e se Bob si fida della chiave pubblica di Alice, accetta il messaggio come proveniente da Alice.

   L'autenticazione avviene usando la firma RSA dal Paragrafo 9.1. Notare il ruolo dalla fiducia: se Bob non crede che la chiave pubblica di Alice sia veramente di Alice, non può essere sicuro che il messagio non venga, invece, da Eva e che la firma non sia di Eva.

Cifratura

1. Il computer di Alice genera un numero casuale da usare come chiave di sessione per un algoritmo di cifratura a chiave simmetrica. Generalmente la chiave è di 128 bit e viene usata con un cifrario a blocco quale *3DES*, *IDEA* o *CAST-128*.

2. Alice cifra il proprio messaggio usando il cifrario simmetrico e questa chiave di sessione.

3. Alice cifra la chiave di sessione usando la chiave pubblica di Bob.

4. La chiave cifrata e il messaggio cifrato sono inviati a Bob.

5. Bob usa la sua chiave privata RSA per decifrare la chiave di sessione. Usa la chiave di sessione per decifrare il messaggio di Alice.

   Si combinano un algoritmo a chiave pubblica e un algoritmo a chiave simmetrica perché la cifratura simmetrica è più veloce. Perciò l'algoritmo a chiave pubblica RSA è usato per cifrare la chiave di sessione, che è piccola, mentre l'algoritmo simmetrico viene usato per cifrare il messaggio, che è potenzialmente lungo.

   Si noti che la fiducia non è necessaria quando si desidera solo la cifratura.

Autenticazione e cifratura

1. Alice calcola l'hash del proprio messaggio e lo firma, come nel passo (2) della procedura di autenticazione. Questo hash è inserito all'inizio del messaggio.

2. Alice genera una chiave di sessione casuale di 128 bit che usa con un algoritmo simmetrico per cifrare l'hash concatenato al messaggio.

3. Alice usa la chiave pubblica di Bob per cifrare la chiave di sessione.

4. La chiave cifrata e la concatenazione cifrata di hash e messaggio sono inviati a Bob.

5. Bob usa la propria chiave privata per decifrare la chiave di sessione.

6. Bob usa la chiave di sessione per ottenere hash e messaggio.

7. Bob verifica la firma usando la chiave pubblica di Alice.

   Ovviamente, questa procedura richiede che Bob si fidi della chiave pubblica di Alice. Inoltre, il messaggio è firmato prima della cifratura, in modo che Bob possa scartare la chiave di sessione dopo la decifrazione e conservare solo il testo in chiaro e la sua firma.

Per generare un certificato PGP, il computer di Alice raccoglie un input casuale misurando le battute sulla tastiera, l'orologio di sistema, i movimenti del mouse e altri eventi. Questo input è usato per generare i numeri primi p e q, e quindi produrre il modulo RSA $n = pq$ e gli esponenti di cifratura e decifrazione e e d. I numeri n ed e diventano la chiave pubblica di Alice. Alice sceglie anche una parola d'ordine. La chiave segreta d è memorizzata nel computer in modo sicuro. Quando il computer deve accedere alla sua chiave privata, chiede la frase segreta per essere sicuro che Alice sia la persona giusta e impedire che Eva usi il computer di Alice e si finga Alice. Il vantaggio di usare la frase segreta è che Alice non deve memorizzare e inserire nel computer l'esponente di decifrazione ogni volta, visto che è lungo oltre un centinaio di cifre.
Gli esempi visti finora usano RSA per la firma e per la cifratura della chiave di sessione. Sono possibili altri algoritmi, per esempio Diffie-Hellman per instaurare la chiave di sessione e DSA per firmare i messaggi.

10.7 Protocolli SSL e TLS

Se avete fatto acquisti su Internet, con grande probabilità la sicurezza della vostra transazione è stata garantita dal protocollo SSL o dal suo parente prossimo TLS. Il protocollo SSL (Secure Socket Layer) fu sviluppato da Netscape per effettuare comunicazioni sicure con il protocollo HTTP. La prima versione fu rilasciata nel 1994, la Versione 3 nel 1995. Il protocollo TLS (Transport Layer Security) è una variante di SSL Versione 3 con piccole modifiche e fu rilasciato dalla Internet Engineering Task Force nel 1999. Entrambi i protocolli sono stati progettati in modo da permettere la comunicazione tra computer che non conoscono le reciproche funzionalità.
Nel seguito descriveremo SSL Versione 3; TLS differisce per piccoli dettagli, quali la generazione dei numeri pseudocasuali. SSL è composto da due componenti principali. Il primo componente è il *record protocol*, responsabile della compressione e cifratura dei dati trasmessi tra le entità. Il secondo componente è un insieme di protocolli di gestione, responsabili dell'instaurazione e del mantenimento dei parametri usati dal record protocol. Il principale e più complicato di questi protocolli è il protocollo di *handshake*, che è anche il primo che vedremo.
Supponiamo che Alice abbia acquistato qualcosa dalla Gigafirm e voglia pagare. Il protocollo di handshake svolge l'autenticazione tra il computer di Alice e il server di Gigafirm e permette ai due partecipanti di concordare su quali algoritmi crittografici usare. Il computer di Alice inizia la comunicazione inviando al server di Gigafirm un messaggio contenente le seguenti informazioni.

1. La più alta versione di SSL che il computer di Alice è in grado di usare.

2. Un numero casuale, composto da un timestamp di 4 byte e da 28 byte casuali.

3. Una suite crittografica che contiene, in ordine di preferenza, gli algoritmi che il computer di Alice vuole usare per la crittografia a chiave pubblica (per esempio, RSA, Diffie-Hellman, ...), per la crittografia simmetrica (3DES, DES, AES, ...), per il calcolo degli hash (SHA-1, MD5, ...) e per la compressione (DEFLATE, Lempel-Ziv-Stac, ...).

Il computer di Gigafirm risponde con un numero casuale di 32 byte, scelto in modo simile al client, e le sue scelte di algoritmi crittografici da usare; per esempio RSA, DES, SHA-1, DEFLATE.
Il computer di Gigafirm invia quindi il proprio certificato X.509 e gli altri certificati della catena di certificazione fino alla CA radice. Gigafirm potrebbe chiedere ad Alice il suo certificato, ma ciò avviene solo raramente, perché la maggior parte degli utenti non ha un certificato personale. Questo significa che Gigafirm dovrà assicurarsi dell'identità di Alice in un secondo tempo. Nel caso dell'acquisto elettronico, Alice inserirà il proprio numero di carta di credito più avanti nella transazione e questo servirà anche per autenticarla (sempre che la carta non sia rubata).
Supporremo nel seguito che si usi RSA come algoritmo a chiave pubblica. Le differenze del protocollo in caso di altri algoritmi sono minime.
Alice genera un numero segreto, chiamato *pre-master secret*, di 48 byte, lo cifra con la chiave pubblica di Gigafirm (che ha ottenuto dal certificato) e spedisce il risultato a Gigafirm, che lo decifra. Sia Alice che Gigafirm ora conoscono i seguenti numeri casuali.

1. Il numero casuale di 32 byte r_A che Alice ha mandato a Gigafirm.

2. Il numero casuale di 32 byte r_G che Gigafirm ha mandato ad Alice.

3. Il pre-master secret di 48 byte s_{pm}.

I due numeri di 32 byte sono stati inviati in chiaro, mentre il pre-master secret è sicuro. Avendo gli stessi numeri, sia Alice sia Gigafirm possono calcolare il *master secret*, che è la concatenazione di

$$\text{MD5}(s_{pm}||\text{SHA-1}(A||s_{pm}||r_A||r_G))$$

$$\text{MD5}(s_{pm}||\text{SHA-1}(BB||s_{pm}||r_A||r_G))$$

$$\text{MD5}(s_{pm}||\text{SHA-1}(CCC||s_{pm}||r_A||r_G)).$$

Le stringhe A, BB e CCC sono dei riempitivi. I timestamp inclusi in r_A in r_G impediscono a Eva di effettuare attacchi di replica nel caso provasse a usare informazioni intercettate da una sessione che effettua una transazione simile.
Poiché MD5 produce un output di 128 bit (=16 byte), il master secret ha 48 byte. Dal master secret si genera un *key block* con la stessa procedura, ovvero concatenando un numero di hash sufficientemente lungo da ottenere un blocco della lunghezza desiderata. Il key block è quindi diviso in sei chiavi segrete, tre per le comunicazioni da Alice a Gigafirm e tre per le comunicazioni da Gigafirm ad Alice. Per la direzione da Alice a Gigafirm, la prima chiave svolge il compito di chiave segreta nel cifrario a blocco

(3DES, AES, ...) scelto all'inizio della comunicazione. La seconda è la chiave per l'autenticazione del messaggio. La terza è il valore iniziale per il modo CBC del cifrario a blocco. Le altre tre chiavi sono per le funzioni corrispondenti da Gigafirm ad Alice. Terminato lo handshake, Alice e Gigafirm possono comunicare usando il protocollo di record. Quando Alice manda un messaggio a Gigafirm, effettua le seguenti operazioni.

1. Comprime il messaggio usando il metodo di compressione concordato.

2. Calcola il codice di autenticazione del messaggio (*message authentication code*, MAC) applicando la funzione hash legando insieme il messaggio compresso e la chiave di autenticazione del messaggio (la seconda chiave ottenuta dal key block).[4]

3. Usa il cifrario a blocco in modo CBC per cifrare il messaggio compresso concatenato con il MAC e invia il risultato a Gigafirm.

Per recuperare il messaggio di Alice, Gigafirm compie queste operazioni.

1. Usa il cifrario a blocco per decifrare il messaggio ricevuto; in questo modo ottiene il messaggio compresso e il suo MAC.

2. Usa il messaggio compresso e la chiave di autenticazione nella direzione Alice-Gigafirm per ricalcolare il MAC. Se è uguale al MAC recapitato con il messaggio, allora il messaggio è autenticato.

3. Decomprime il messaggio compresso e ottiene il messaggio di Alice.

Le comunicazioni da Gigafirm sono cifrate e decifrate allo stesso modo usando le altre tre chiavi ottenute dal key block.

10.8 Protocollo SET

Ogni volta che viene effettuata una transazione elettronica su Internet, si trasmettono grosse quantità di dati che devono essere protette da intercettazione per garantire la privacy degli acquirenti ed evitare truffe o furti di numeri di carta di credito.
Un buon sistema di commercio elettronico deve garantire almeno i seguenti requisiti anche su un canale di comunicazione pubblico come Internet.

1. **Autenticità:** le identità dei partecipanti alla transazione non possono essere contraffatte e le firme non possono essere falsificate.

2. **Integrità:** tutti i documenti (per esempio gli ordini di acquisto o le istruzioni di pagamento) non possono essere modificati.

3. **Riservatezza:** i dettagli della transazione devono essere segreti.

[4] La funzione hash viene applicata due volte. Prima si calcola l'hash della chiave concatenata a un numero di sequenza e al messaggio compresso. Successivamente si applica la funzione hash alla chiave concatenata al risultato della prima operazione. (*N.d.Rev.*)

4. **Sicurezza:** i dati sensibili, come i numeri di carta di credito, devono essere protetti.

Nel 1996 le compagnie emettitrici di carte di credito Mastercard e Visa cercarono di stabilire uno standard per il commercio elettronico. Il risultato, che coinvolse anche molte altre aziende, è il protocollo SET (Secure Electronic Transaction™) che permette di usare i sistemi di carta di credito esistenti in modo sicuro anche su un canale aperto. Il protocollo SET è abbastanza complesso e coinvolge, per esempio, il protocollo SSL per la certificazione che il detentore della carta di credito e il negoziante siamo legittimi e specifica anche come devono essere fatte le richieste di pagamento. Nel seguito ci concentreremo su un singolo aspetto del protocollo, cioè l'uso delle firme doppie (*dual signatures*).
Nella discussione del protocollo considereremo il caso più semplice, in cui si usa solo la crittografia a chiave pubblica. Ovviamente sono possibili molte varianti, per esempio, per aumentare la velocità, si può usare la crittografia simmetrica per cifrare il messaggio e la crittografia a chiave pubblica per trasmettere la chiave di sessione e un hash del messaggio.
Si supponga che Alice voglia comprare un libro dal titolo *Come usare i numeri di carta di credito di altre persone per frodare le banche*, di cui ha visto la pubblicità su Internet. Per ovvie ragioni, si trova a disagio a spedire al libraio il suo numero di carta di credito e certamente non vuole che la sua banca sappia cosa sta comprando. Una situazione analoga si verifica in molte transazioni: la banca non ha bisogno di sapere cosa sta comprando il cliente e, per ragioni di sicurezza, il negoziante non dovrebbe sapere il numero di carta di credito. Tuttavia queste due informazioni devono essere in qualche modo collegate tra loro, altrimenti il negoziante non potrebbe associare i pagamenti e gli ordini. Le **firme doppie** risolvono questo problema.
I tre partecipanti al protocollo sono il Cardholder (il titolare della carta e acquirente), il Merchant (il negoziante) e la Banca (che autorizza l'uso della carta).
Il Cardholder costruisce due insiemi di informazioni.

- GSO = *Goods and Services Order* (Ordine di beni e servizi), che contiene i nomi del titolare della carta e del negoziante, le quantità ordinate, i prezzi e così via.
- PI = *Payment Instructions* (Istruzioni di pagamento), che contiene il nome del negoziate, il numero di carta di credito, il prezzo totale e così via.

Il sistema usa una funzione hash pubblica, H, e un crittosistema a chiave pubblica come RSA. Ogni partecipante ha una chiave privata e una corrispondente chiave pubblica. Siano E_C, E_M e E_B le funzioni pubbliche di cifratura del Cardholder, del Merchant e della Banca e siano D_C, D_M e D_B le funzioni private di decifrazione.
Il Cardholder compie le seguenti operazioni.

1. Calcola $GSOMD = H(E_M(GSO))$, che è il *message digest*, o hash, del GSO cifrato.
2. Calcola $PIMD = H(E_B(PI))$, che è il *message digest* del PI cifrato.
3. Concatena $GSOMD$ e $PIMD$ per ottenere $PIMD||GSOMD$, calcola l'hash del risultato per ottenere il *payment-order message digest* $POMD = H(PIMD||GSOMD)$.

4. Firma $POMD$ calcolando $DS = D_C(POMD)$. Questa operazione prende il nome di firma doppia, perché è applicata alla concatenazione di due oggetti per due destinatari diversi.
5. Invia $E_M(GSO)$, DS, $PIMD$ e $E_B(PI)$ al Merchant.

Il Merchant compie le seguenti operazioni.

1. Calcola $H(E_M(GSO))$ (che dovrebbe essere uguale a $GSOMD$).
2. Calcola $H(PIMD||H(E_M(GSO)))$ e $E_C(DS)$. Se sono uguali, il Merchant ha verificato la firma del Cardholder ed è sicuro che l'ordine sia autentico.
3. Calcola $D_M(E_M(GSO))$ e ottiene GSO.
4. Invia $GSOMD$, $E_B(PI)$ e DS alla Banca.

La Banca compie le seguenti operazioni.

1. Calcola $H(E_B(PI))$ (che dovrebbe essere uguale a $PIMD$).
2. Concatena $H(E_B(PI))$ e $GSOMD$.
3. Calcola $H(H(E_B(PI))||GSOMD)$ e $E_C(DS)$. Se sono uguali, la Banca ha verificato la firma del Cardholder.
4. Calcola $D_B(E_B(PI))$ e ottiene le istruzioni di pagamento PI.
5. Restituisce un'autorizzazione firmata e cifrata (usando E_M) con cui garantisce il pagamento.

Il Merchant completa la procedura.

1. Restituisce al Cardholder una ricevuta firmata e cifrata (con E_C) indicando che la transazione è completa.

Il Merchant vede solo il modulo cifrato $E_B(PI)$ contenente le istruzioni di pagamento e quindi non vede il numero di carta di credito. Il Merchant e la Banca non riescono a modificare le informazioni sull'ordine perché DS è ottenuto usando una funzione hash. La Banca vede solo il *message digest* del GSO, che contiene l'ordine, e non ha idea di cosa sia stato ordinato.
Questa procedura garantisce i requisiti di integrità, privacy e sicurezza. Nelle implementazioni concrete, l'autenticità richiede molti altri passi; per esempio bisogna garantire che le chiavi pubbliche appartengano veramente ai partecipanti e non a impostori. Per questo scopo si usano i certificati emessi da autorità fidate.

10.9 Esercizi

1. In una rete di tre utenti (A, B e C) si vuole usare uno schema di Blom per instaurare le chiavi di sessione tra le coppie di utenti. Sia $p = 31$ e siano: $r_A = 11$, $r_B = 3$ e $r_C = 2$. Si supponga che Trent scelga i numeri: $a = 8$, $b = 3$ e $c = 1$. Calcolare le chiavi di sessione.

2. **(a)** Dimostrare che, nello schema di Blom,

$$K_{AB} \equiv a + b(r_A + r_B) + cr_Ar_B \pmod{p}.$$

(b) Dimostrare che $K_{AB} = K_{BA}$.

(c) Un modo alternativo per vedere lo schema di Blom è usando polinomi in due variabili. Si definisca il polinomio:

$$f(x,y) = a + b(x + y) + cxy \pmod{p}.$$

Esprimere la chiave K_{AB} in termini di f.

3. Tu (U) e io (I) siamo utenti disonesti in una rete che usa lo schema di Blom per l'instaurazione della chiave con $k = 1$. Abbiamo deciso di coalizzarci per scoprire le altre chiavi di sessione nella rete. In particolare, si supponga $p = 31$, $r_U = 9$ e $r_I = 2$. Abbiamo ricevuto da Trent, l'autorità fidata, $a_U = 18$, $b_U = 29$, $a_I = 24$, $b_I = 23$. Calcolare a, b e c.

4. Questa è una versione alternativa dell'attacco dell'intruso allo scambio di chiave di Diffie-Hellman del Paragrafo 10.1 e ha il "vantaggio" di non richiedere che Eva intercetti e ritrasmetta tutti i messaggi tra Bob e Alice. Si supponga che Eva scopra che $p = Mq + 1$, dove q è un intero e M è piccolo. Eva intercetta α^x e α^y, come prima. Invia a Bob $(\alpha^x)^q \pmod{p}$ e invia ad Alice $(\alpha^y)^q \pmod{p}$.

(a) Dimostrare che Alice e Bob calcolano la stessa chiave K.

(b) Dimostrare che ci sono solo M possibili valori di K, per cui Eva può trovare la chiave con una ricerca esaustiva.

5. Bob, Ted, Carla e Alice vogliono accordarsi su una chiave crittografica comune. Scelgono pubblicamente un numero primo grande p e una radice primitiva α, poi ciascuno sceglie privatamente un numero casuale, rispettivamente b, t, c, a. Descrivere un protocollo che permetta di calcolare $K \equiv \alpha^{btca} \pmod{p}$ in modo sicuro (ignorare i possibili attacchi dell'intruso).

6. Si supponga che Nelson provi a implementare ingenuamente un analogo del protocollo a tre vie del Paragrafo 3.6 per mandare una chiave K a Heidi. Sceglie come chiave una stringa monouso K_N, la somma XOR con K e invia $M_1 = K_N \oplus K$ a Heidi. Heidi somma XOR quello che riceve con la propria stringa monouso K_H, ottiene $M_2 = M_1 \oplus K_H$ e invia M_2 a Nelson, che calcola $M_3 = M_2 \oplus K_N$. Nelson invia M_3 a Heidi, che recupera K e $M_3 \oplus K_H$.

(a) Dimostrare che $K = M_3 \oplus K_H$.

(b) Supponendo che Eva intercetti M_1, M_2, M_3, come può recuperare K?

Capitolo 11

Moneta digitale

Con l'avanzata delle tecnologie di comunicazione come Internet e le reti senza fili, sono nate nuove possibilità per il commercio. Molte transazioni oggi avvengono elettronicamente, spesso usando carte di credito. Le carte di credito, tuttavia, non sono la stessa cosa del denaro contante. Nel concludere un acquisto usando monete e banconote, l'acquirente ha la garanzia che la sua identità non venga rivelata al venditore. In questo capitolo vedremo un modello di moneta digitale che emula il comportamento del denaro contante usando informazioni digitali. L'obiettivo è garantire l'anonimato anche in un sistema elettronico in cui si scambiano file, non monete, in cambio di prodotti e servizi. Poiché è facile copiare file elettronici, se garantiamo l'anonimato dobbiamo anche prendere delle misure per impedire la contraffazione.

11.1 Moneta digitale

Supponiamo che l'On. Approvoni voglia nascondere la provenienza delle ingenti donazioni del suo amico Comm. Ricconi e finga che il denaro provenga per lo più da persone come la Dott.ssa Degli Onesti. Oppure supponiamo che il Comm. Ricconi non voglia che Approvoni sappia chi è l'autore delle donazioni. Se Ricconi pagasse con un assegno, qualche dipendente della banca lo verrebbe a sapere e potrebbe rivelarlo alla stampa. Allo stesso modo, Approvoni non può accettare pagamenti con carta di credito, per cui l'unico schema di pagamento anonimo sembra il contante.
Supponiamo per un momento che Approvoni sia rimasto in carica molte legislature. Ci troviamo alla fine del XXI secolo e tutto il commercio avviene elettronicamente. Com'è possibile avere denaro contante elettronico? Anche se fotocopiare banconote è possibile, un osservatore attento potrebbe identificare le differenze tra l'originale e la copia. Invece, le copie di un'informazione elettronica sono indistinguibili dall'originale. Perciò, chi avesse una banconota elettronica potrebbe farne molte copie. Pertanto è necessario un sistema per impedire che la stessa moneta elettronica sia spesa due o più volte.