Capitolo 2

Introduzione alla crittografia

In questo capitolo sono introdotti i concetti di base della crittografia e presentate informazioni preliminari necessarie per proseguire nella lettura del libro.

Sommario

2.1 Cifratura

2.2 Autenticazione

2.3 Cifratura a chiave pubblica

2.4 Firme digitali

2.5 Infrastruttura a chiave pubblica: PKI

2.6 Gli attacchi

2.7 Entriamo nei dettagli

2.8 Livelli di sicurezza

2.9 Prestazioni

2.10 Complessità

2.11 Esercizi

2.1 Cifratura

La cifratura è l'obiettivo di partenza della crittografia. Lo schema generale è mostrato nella Figura 2.1. Alice e Bob vogliono comunicare tra loro (l'uso di nomi di persona, in particolare Alice, Bob ed Eve, è una tradizione nel mondo della crittografia). Tuttavia, in generale i canali di comunicazione devono essere considerati non sicuri. Eve sta facendo attività di *eavesdropping*, intercettazione, sul canale. Ogni messaggio *m* che Alice invia a Bob viene ricevuto anche da Eve (lo stesso accade per i messaggi inviati da Bob verso Alice, ma si tratta dello stesso problema a parti inverse. Se possiamo proteggere i messaggi di Alice, la stessa soluzione funzionerà anche per i messaggi di Bob, quindi ci concentriamo sui messaggi di Alice). In che modo Alice e Bob possono comunicare senza che Eve intercetti tutto?

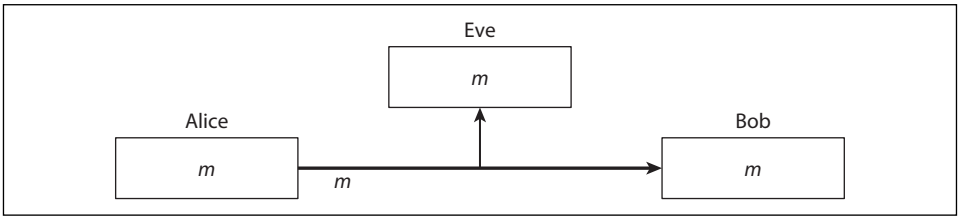

Figura 2.1 In che modo Alice e Bob possono comunicare in maniera sicura?

Per evitare che Eve possa venire a conoscenza del contenuto della conversazione tra Alice e Bob, questi ultimi utilizzano la crittografia come mostrato nella Figura 2.2. In primo luogo Alice e Bob si mettono d'accordo su una chiave segreta K_e. Questa operazione va svolta attraverso un canale di comunicazione che Eve non sia in grado di intercettare. Alice potrebbe inviare via email una copia della chiave a Bob, o qualcosa del genere. Torneremo a parlare di scambio delle chiavi più avanti.

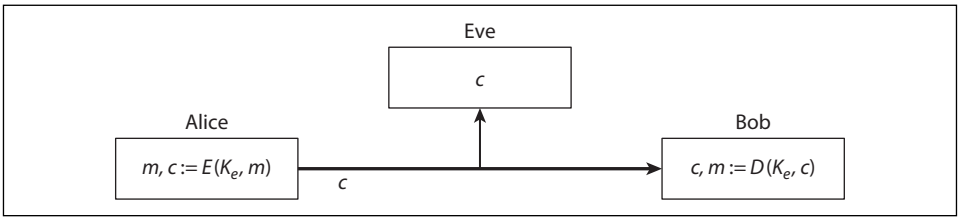

Figura 2.2 Configurazione generica per la cifratura.

Quando Alice vuole inviare un messaggio m, per prima cosa lo cifra con una funzione di crittografia. Scriviamo la funzione di cifratura come $E(K_e, m)$ e indichiamo il risultato come *testo cifrato* c (il messaggio originale è chiamato *testo in chiaro*). Invece di inviare m a Bob, Alice invia il testo cifrato $c := E(K_e, m)$. Quando Bob riceve c, può decifrarlo utilizzando la funzione di decifratura $D(K_e, c)$ per ottenere il testo in chiaro originale m che Alice voleva inviargli. Ma Eve non conosce la chiave K_e e quando riceve il testo cifrato c, non è in grado di decifrarlo. Una buona funzione di cifratura impedisce di ottenere il testo in chiaro m a partire dal testo cifrato c senza conoscere la chiave. O meglio, una buona funzione di cifratura dovrebbe fornire una riservatezza ancora maggiore: un attaccante non dovrebbe essere in grado di ottenere alcuna informazione sul messaggio m, a parte eventualmente la sua lunghezza e l'ora in cui è stato inviato.
Questo scenario ha applicazioni ovvie per la trasmissione di email, ma si applica anche all'attività di memorizzazione dei dati, che può essere pensata come trasmissione di messaggi nel tempo, anziché nello spazio. In quel caso, infatti, spesso Alice e Bob sono la stessa persona ma in momenti diversi.

2.1.1 Il principio di Kerckhoff

Per decifrare il testo cifrato, Bob deve conoscere l'algoritmo di decifratura D e la chiave K_e. Una regola importante è il principio di Kerckhoff: la sicurezza dello schema

di cifratura deve dipendere solo dalla segretezza della chiave K_e, non dalla segretezza dell'algoritmo.
Questa regola è fondata su molte e ottime motivazioni. Gli algoritmi sono difficili da cambiare. Sono integrati in software o hardware, difficili da aggiornare. In situazioni pratiche, lo stesso algoritmo viene utilizzato per molto tempo. Inoltre, se mantenere segreta una semplice chiave è difficile, mantenere segreto l'algoritmo lo è ancora di più difficile (e di conseguenza aumentano i costi). Nessuno costruisce un sistema crittografico per due soli utenti. Ogni partecipante al sistema (e potrebbero essere milioni) utilizza lo stesso algoritmo; a Eve basterebbe ottenere l'algoritmo da uno di essi, e uno facile da attaccare generalmente si trova. Eve potrebbe semplicemente rubare un portatile su cui sia disponibile l'algoritmo. Inoltre, ricordate il nostro modello basato sulla paranoia? Eve potrebbe anche essere uno degli utenti del sistema, persino uno dei suoi progettisti.
Esistono buone ragioni per rendere pubblici i gli algoritmi. Per esperienza sappiamo che è molto facile commettere un piccolo errore e creare un algoritmo di crittografia debole. Se l'algoritmo non è pubblico, nessuno troverà la falla finché un attaccante non provi ad attaccarlo, ma in questo caso l'attaccante potrebbe utilizzare la falla per entrare nel sistema. Abbiamo analizzato una certa quantità di algoritmi di crittografia segreti e *tutti* avevano uno o più punti deboli. Ecco perché c'è una salutare sfiducia verso gli argomenti proprietari, riservati o comunque segreti. Non lasciatevi ingannare dalle vecchie abitudini: "Se teniamo segreto anche l'algoritmo, aumentiamo la sicurezza". È sbagliato. Il potenziale aumento di sicurezza è ridotto, mentre la potenziale diminuzione di sicurezza è enorme. La lezione è semplice: non fidatevi degli algoritmi segreti.

2.2 Autenticazione

Alice e Bob hanno un altro problema nella Figura 2.1: non solo Eve può intercettare il messaggio, può anche modificarlo. Questo richiede che Eve abbia un controllo appena maggiore sul canale di comunicazione, ma non è affatto impossibile che accada. Per esempio, nella Figura 2.3 Alice tenta di inviare il messaggio *m*, ma Eve interferisce con il canale di comunicazione. Invece di ricevere *m*, Bob riceve un differente messaggio *m'*. Supponiamo che Eve venga a conoscenza del contenuto del messaggio *m* che Alice ha tentato di inviare; allora potrebbe fare anche altre cose, come cancellare un messaggio in modo che Bob non lo riceva mai, inserire un nuovo messaggio inventato da lei, registrare un messaggio e inviarlo a Bob in un momento successivo, o modificare l'ordine dei messaggi.

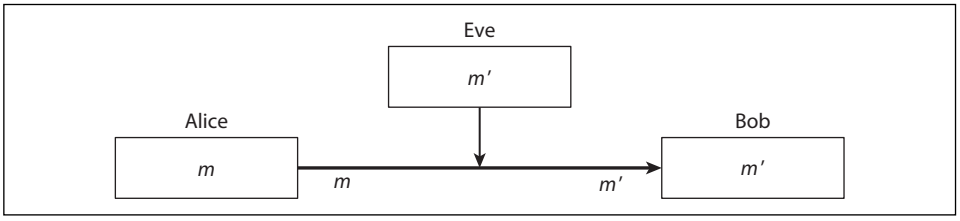

Figura 2.3 Come fa Bob a sapere chi ha inviato il messaggio?

Considerate il momento in cui Bob ha appena ricevuto un messaggio. Perché dovrebbe credere che il messaggio provenga da Alice? Non ha ragione di credere che sia così. E se non sa chi ha inviato il messaggio, allora quest'ultimo è pressoché inutile.
Per risolvere il problema, introduciamo il concetto di autenticazione. Come la crittografia, l'autenticazione utilizza una chiave segreta nota sia a Bob sia ad Alice. Chiameremo la chiave di autenticazione K_a per distinguerla dalla chiave di crittografia K_e. La Figura 2.4 mostra il processo di autenticazione di un messaggio m. Quando Alice invia il messaggio m, calcola un *codice di autenticazione del messaggio*, MAC (*Message Authentication Code*). Alice calcola il MAC a come $a := h(K_a, m)$, dove h è la funzione MAC e K_a è la chiave di autenticazione. Ora Alice invia m e a a Bob. Quando Bob riceve m e a, calcola a sua volta il valore che a dovrebbe avere, utilizzando la chiave K_a, e lo confronta con il valore di a ricevuto.

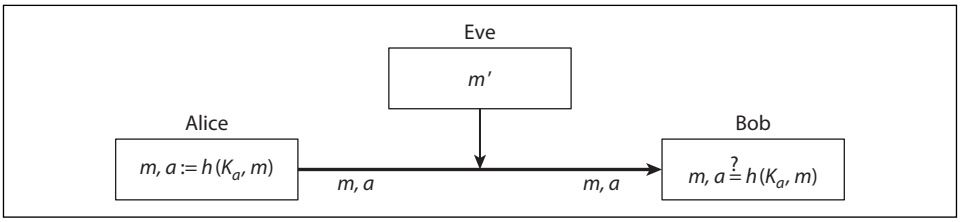

Figura 2.4 Configurazione generica per l'autenticazione.

Ora Eve vuole modificare il messaggio m in un diverso messaggio m'. Se si limitasse a sostituire m con m', Bob calcolerebbe $h(K_a, m')$ e confronterebbe il risultato a. Ma una buona funzione MAC non fornisce lo stesso risultato per messaggi diversi, quindi Bob si accorgerebbe che il messaggio non è corretto. Dato che il messaggio è sbagliato, in un modo o nell'altro, Bob lo scarterà.
Se supponiamo che Eve non conosca la chiave di autenticazione K_a, il solo modo in cui potrebbe ottenere un messaggio e un MAC valido è quello di intercettare Alice mentre invia il messaggio a Bob. Ciò permette ancora a Eve di provare a commettere qualche malefatta. Eve può registrare i messaggi e i loro MAC, e quindi riproporli inviandoli di nuovo a Bob in qualsiasi momento successivo.
L'autenticazione pura offre solo una soluzione parziale. Eve può ancora cancellare i messaggi inviati da Alice, può anche riproporre vecchi messaggi o modificarne l'ordine. Di conseguenza, l'autenticazione è quasi sempre combinata con uno schema di numerazione per numerare i messaggi in sequenza. Se m contiene tale numero sequenziale, Bob non si lascia ingannare dai vecchi messaggi riproposti da Eve: vedrà semplicemente che il messaggio ha un MAC corretto, ma un numero sequenziale relativo a un messaggio vecchio, quindi lo scarterà.
L'autenticazione in combinazione con la numerazione dei messaggi risolve la maggior parte dei problemi. Eve può ancora impedire la comunicazione tra Alice e Bob, o ritardare i messaggi eliminandoli e inviandoli a Bob in un momento successivo. Se i messaggi non sono cifrati, Eve può cancellarli o ritardarli selettivamente in base al loro contenuto, ma non può fare altro.
Il miglior modo di affrontare il problema consiste nel considerare il caso in cui Alice invia una sequenza di messaggi $m_1, m_2, m_3, \ldots$ e Bob accetta solo i messaggi con un MAC

valido e per i quali il numero di sequenza è strettamente maggiore del numero di sequenza dell'ultimo messaggio accettato (strettamente maggiore significa "maggiore e non uguale a"). Quindi Bob riceve una sequenza di messaggi che è una *sottosequenza* della sequenza inviata da Alice. Una sottosequenza è semplicemente la sequenza di partenza da cui mancano zero o più messaggi.
Questo è il massimo aiuto che la crittografia può fornire in questa situazione. Bob riceverà una sottosequenza dei messaggi inviati da Alice, ma Eve può soltanto eliminare alcuni messaggi o fermare tutte le comunicazioni, non può manipolare il contenuto dei messaggi. Per evitare la perdita di informazioni, Alice e Bob utilizzeranno spesso uno schema che prevede il rinvio dei messaggi andati persi, ma questo è un aspetto più specifico dell'applicazione e non rientra nella crittografia.
Naturalmente, in molte situazioni Alice e Bob dovranno utilizzare sia la crittografia, sia l'autenticazione. Discuteremo in dettaglio questa combinazione più avanti. Non confondete mai i due concetti: la cifratura di un messaggio non garantisce contro la manipolazione del suo contenuto, e l'autenticazione di un messaggio non ne mantiene la segretezza. Uno degli errori classici nel campo della crittografia è quello di pensare che cifrare un messaggio garantisca anche che un attaccante non lo possa modificare. Non è così.

2.3 Cifratura a chiave pubblica

Per utilizzare la crittografia come discusso nel Paragrafo 2.1, Alice e Bob devono condividere la chiave K_e. E come farlo? Alice non può semplicemente inviare la chiave a Bob utilizzando il canale di comunicazione, poiché anche Eve potrebbe leggerla. Il problema della distribuzione e della gestione delle chiavi è uno dei più difficili da affrontare, e sono disponibili solo soluzioni parziali.
Alice e Bob potrebbero aver scambiato la chiave il mese scorso quando si sono incontrati per un aperitivo. Ma se Alice e Bob fanno parte di un gruppo di 20 amici a cui piace comunicare l'uno con l'altro, ogni membro del gruppo dovrà scambiare un totale di 19 chiavi. In totale, il gruppo deve scambiare 190 chiavi. La situazione è già molto complessa, e peggiora all'aumentare del numero di persone con cui Alice comunica.
Lo scambio di chiavi crittografiche è un problema antico, e un contributo importante alla soluzione è portato dalla crittografia a chiave pubblica. Discuteremo per prima cosa la cifratura a chiave pubblica, rappresentata nella Figura 2.5. Abbiamo lasciato Eve fuori dal diagramma; da adesso in avanti, supponiamo che tutte le comunicazioni siano sempre accessibili a un nemico come Eve. A parte l'assenza di Eve, la figura è molto simile alla Figura 2.2; la differenza sostanziale è che Alice e Bob non usano più la stessa chiave, ma chiavi diverse. Questa è l'idea che sta alla base della crittografia a chiave pubblica: la chiave usata per cifrare un messaggio è diversa da quella usata per decifrarlo.

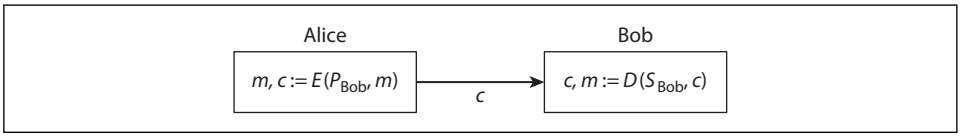

Figura 2.5 Configurazione generica per la cifratura a chiave pubblica.

Per prima cosa Bob genera una coppia di chiavi (S_{Bob}, P_{Bob}) utilizzando uno speciale algoritmo. Le due chiavi sono la chiave segreta S_{Bob} e la chiave pubblica P_{Bob}. Poi Bob fa un cosa a prima vista sorprendente: rende nota P_{Bob} come sua chiave pubblica. Questa operazione rende la chiave pubblica di Bob, P_{Bob}, accessibile a chiunque, incluse Alice ed Eve (perché dovrebbe chiamarsi chiave pubblica, altrimenti?).
Quando Alice vuole inviare un messaggio a Bob, per prima cosa si procura la chiave pubblica di Bob. Potrebbe ottenerla da un elenco pubblico, o forse da qualcuno di cui si fida. Alice cifra il messaggio m con la chiave pubblica P_{Bob} per ottenere il testo cifrato c, che invia a Bob. Questi utilizza la propria chiave segreta S_{Bob} e l'algoritmo crittografico per decifrare il messaggio c e ottenere il messaggio m.
Affinché tutto funzioni, l'algoritmo di generazione della coppia di chiavi, l'algoritmo di cifratura e di decifratura devono essere realizzati in modo tale di assicurare che la decifratura restituisca il messaggio originale. In altre parole, la relazione $D(S_{Bob}, E(P_{Bob}, m)) = m$ deve essere valida per ogni possibile messaggio m. Esamineremo nel dettaglio questo aspetto più avanti.
Non solo le due chiavi che Alice e Bob usano sono diverse, ma anche gli algoritmi di cifratura e decifratura possono essere molto differenti. Tutti gli schemi di crittografia a chiave pubblica si affidano pesantemente alla matematica. Un requisito ovvio è che non deve essere possibile ottenere la chiave segreta dalla corrispondente chiave pubblica, ma ci sono anche molti altri requisiti.
Questo tipo di cifratura è chiamato *cifratura a chiave asimmetrica*, o *a chiave pubblica*, in contrasto con la cifratura a chiave simmetrica o a chiave segreta di cui abbiamo parlato in precedenza.
La crittografia a chiave pubblica semplifica notevolmente il problema della distribuzione delle chiavi. Per utilizzare questo sistema, Bob deve semplicemente distribuire una singola chiave pubblica che chiunque può utilizzare. Alice pubblica la propria chiave pubblica nello stesso modo; ora Alice e Bob possono comunicare in maniera sicura. Anche nel caso di gruppi ampi, ogni membro del gruppo deve solo pubblicare una singola chiave pubblica, cosa che risulta gestibile con relativa facilità.
E allora perché mai utilizzare la cifratura a chiave segreta, se la cifratura a chiave pubblica è molto più facile? Perché la cifratura a chiave pubblica è molto meno efficiente, di diversi ordini di grandezza. Utilizzarla in ogni situazione è semplicemente troppo costoso. In sistemi reali che utilizzano la crittografia a chiave pubblica, quasi sempre si può notare che si tratta di una miscela di algoritmi a chiave pubblica e a chiave segreta. Gli algoritmi a chiave pubblica sono utilizzati per designare una chiave segreta, che viene poi utilizzata per cifrare i dati effettivi. Questo approccio combina la flessibilità della crittografia a chiave pubblica con l'efficienza della crittografia a chiave segreta.

2.4 Firme digitali

Le firme digitali sono equivalenti alle chiavi pubbliche dei codici di autenticazione dei messaggi. La situazione generica è mostrata nella Figura 2.6. Questa volta è Alice che usa un algoritmo per generare una coppia di chiavi (S_{Alice}, P_{Alice}) e pubblica la chiave pubblica P_{Alice}. Quando Alice vuole inviare un messaggio firmato m a Bob, elabora una firma $s := \sigma(S_{Alice}, m)$ e invia m e s a Bob. Questi utilizza un algoritmo di verifica $v(P_{Alice}, m, s)$ che sfrutta la chiave pubblica di Alice per verificare la firma. La firma funziona proprio

come un MAC, a parte il fatto che Bob può verificarla con la chiave pubblica, mentre la chiave segreta è necessaria per creare una nuova firma.

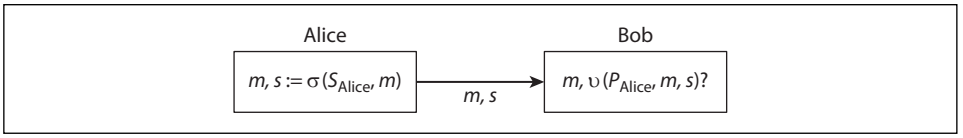

Figura 2.6 Configurazione generica per la firma digitale.

Bob necessita soltanto della chiave pubblica di Alice per verificare che il messaggio provenga da lei. È interessante notare che chiunque può conoscere la chiave pubblica di Alice e verificare che il messaggio provenga da lei. Ecco perché chiamiamo *s* una *firma digitale*. In un certo senso, Alice firma il messaggio. Qualora vi fosse un contenzioso, Bob potrebbe presentare il messaggio *m* e la firma *s* a un giudice e provare che Alice ha firmato il messaggio.
Tutto ciò è molto bello in teoria, e funziona... in teoria. Nel mondo reale le firme digitali hanno molte limitazioni di cui è importante rendersi conto. Il problema principale è che Alice non elabora da sé una firma, è il suo computer a calcolarla. La firma digitale, di conseguenza, non è una prova che Alice abbia approvato il messaggio, o che lo abbia almeno visto sullo schermo del suo computer. Data la facilità con cui i virus prendono il controllo dei computer, la firma digitale in realtà prova davvero poco in questo scenario. Nondimeno, quando vengono utilizzate in maniera appropriata, le firme digitali sono estremamente utili.

2.5 Infrastruttura a chiave pubblica: PKI

La crittografia a chiave pubblica semplifica la gestione delle chiavi, tuttavia Alice deve ancora trovare la chiave pubblica di Bob. Come può essere sicura che sia davvero la chiave di Bob e non quella di qualcun altro? Forse Eve ha creato una coppia di chiavi e ne ha pubblica una facendo apparire che sia stato Bob a farlo.
La soluzione generale è quella di utilizzare una *infrastruttura a chiave pubblica* o PKI (*Public Key Infrastructure*).
Alla base vi è un ente centrale chiamato *autorità di certificazione*, o CA (*Certification Authority*). Ogni utente presenta la propria chiave pubblica alla CA e si identifica con essa. La CA quindi firma la chiave pubblica dell'utente utilizzando una firma digitale. Il messaggio firmato, o *certificato*, recita: "Io, la CA, ho verificato che la chiave pubblica P_{Bob} appartiene a Bob". Spesso il certificato include una data di scadenza e altre informazioni utili.
Utilizzando i certificati, è molto più semplice per Alice trovare la chiave di Bob. Assumeremo che Alice abbia la chiave pubblica della CA e abbia verificato che sia la chiave corretta. Alice può quindi recuperare la chiave di Bob da un database, oppure Bob può inviare la sua chiave ad Alice. Alice può verificare il certificato in riferimento alla chiave, utilizzando la chiave della CA, già in suo possesso. Questo certificato le assicura di avere la chiave pubblica corretta per comunicare con Bob. Analogamente, Bob può cercare la chiave pubblica di Alice e assicurarsi di comunicare con la persona giusta.

In una PKI, ogni partecipante deve soltanto fare in modo che la CA certifichi la propria chiave pubblica, e conoscere la chiave pubblica della CA per poter verificare i certificati degli altri partecipanti. È uno scenario decisamente più agevole rispetto a quello in cui si devono scambiare chiavi con tutte le persone con cui vuole comunicare. Questo è il grande vantaggio di una PKI: ci si registra una sola volta e si utilizza ovunque.
Per ragioni pratiche, una PKI è spesso impostata con diversi livelli di CA. C'è una CA di primo livello, chiamata *root*, che emette i certificati sulle chiavi delle CA di livello inferiore, che invece certificano le chiavi degli utenti. Il sistema continua a comportarsi nello stesso modo, ma ora Alice deve controllare due certificati per verificare la chiave di Bob.
Una PKI non è la soluzione definitiva: ci sono ancora molti problemi. In primo luogo, la CA deve avere la fiducia di tutti. In alcune situazioni, non è un problema. All'interno di una società, il reparto del personale conosce tutti i dipendenti e può ricoprire il ruolo di CA. Ma nel mondo non esiste un ente che sia considerato affidabile da tutti. L'idea che una singola PKI possa gestire tutti gli utenti del mondo non sembra praticabile.
Il secondo problema è quello della responsabilità. Che cosa succede se la CA fornisce un certificato falso, o se la chiave privata della CA viene rubata? Alice si fiderebbe di un certificato falso, e potrebbe perdere molto denaro a causa di ciò. Chi paga? La CA può farsi carico di questo genere di assicurazione? Ciò richiede una relazione d'affari molto più estesa tra Alice e la CA.
Ci sono attualmente molte società che stanno cercando di diventare la CA leader nel mondo. VeriSign è probabilmente la più conosciuta. Tuttavia, VeriSign limita esplicitamente la propria responsabilità nel caso in cui non possa essere in grado di assolvere alla propria funzione. Nella maggior parte dei casi la responsabilità è limitata a 100 dollari statunitensi, meno di quanto molti utenti pagano per una normale transazione di commercio elettronico: transazioni che sono rese sicure utilizzando certificati firmati da VeriSign. In questi casi non ci sono problemi perché un pagamento con carta di credito è sicuro per il consumatore, tuttavia, non compreremo la nostra auto nuova utilizzando un certificato che VeriSign garantisce solo per 100 dollari.

2.6 Gli attacchi

Dopo aver descritto le più importanti funzioni utilizzate in crittografia, parleremo ora di alcuni attacchi, concentrandoci su quelli indirizzati contro gli schemi di cifratura. Esistono molti tipi di attacchi, ognuno dei quali ha la sua specifica gravità.

2.6.1 Il modello con solo testo cifrato

L'*attacco con solo testo cifrato* (*ciphertext-only*) è quello che molti intendono quando si parla di entrare in un sistema di cifratura. Questa è la situazione in cui Alice e Bob eseguono la cifratura dei loro dati e tutto ciò che un attaccante può vedere è il testo cifrato. Il tentativo di decifrare un messaggio quando si conosce solo il testo cifrato è chiamato attacco con solo testo cifrato. È il tipo di attacco più difficile, poiché l'attaccante ha a disposizione la quantità di informazione minima.

2.6.2 Il modello con testo in chiaro noto

Nell'*attacco con testo in chiaro noto* (*known-plaintext*) l'attaccante conosce sia il testo cifrato, sia il testo in chiaro. L'obiettivo più ovvio è quello di trovare la chiave di decifratura. Apparentemente sembra uno scenario poco plausibile: come si potrebbe conoscere il testo in chiaro? In realtà esistono molte situazioni nelle quali un attaccante può venire a conoscenza del testo in chiaro di una comunicazione. A volte ci sono messaggi che sono facilmente prevedibili; per esempio quando Alice è in vacanza e ha un risponditore email automatico che invia una risposta: "Sono in vacanza" a ogni email in arrivo. È possibile ottenere una esatta copia di questo messaggio inviando una email ad Alice e leggendo la risposta. Quando Bob invia un messaggio email ad Alice, il risponditore automatico risponde, questa volta cifrando il messaggio. Ora è disponibile sia il messaggio in chiaro, sia il messaggio cifrato. Se l'attaccante riesce a trovare la chiave, potrà decifrare tutti gli altri messaggi che Alice e Bob scambiano con la stessa chiave. L'ultima parte è importante e vale la pena ripeterla: si utilizza la conoscenza di alcune coppie di testo in chiaro e testo cifrato per ottenere la chiave, e successivamente si sfrutta la conoscenza della chiave per decifrare altri testi cifrati.

Un'altra situazione tipica è quella in cui Alice invia lo stesso messaggio a molte persone tra cui l'attaccante. Questo rende disponibile all'attaccante sia il testo in chiaro, sia i testi cifrati della copia che Alice ha inviato a tutti gli altri.

Supponiamo che Alice e Bob si scambino le bozze di un articolo di stampa. Una volta che l'articolo viene pubblicato, l'attaccante può conoscere il testo in chiaro e il testo cifrato.

Anche se non conosce l'intero testo cifrato, spesso l'attaccante ne conosce una parte. Le email hanno un inizio prevedibile, o una firma fissa alla fine. L'intestazione di un pacchetto IP è altamente prevedibile. Tali dati prevedibili portano a avere un testo in chiaro parzialmente noto. Un attacco con testo in chiaro noto è molto più potente di un attacco con solo testo cifrato, poiché l'attaccante dispone di più informazioni rispetto al caso del solo testo cifrato.

2.6.3 Il modello con testo in chiaro scelto

Il livello successivo è quello in cui l'attaccante può scegliere il testo in chiaro. Questo è un tipo di attacco più potente di un attacco con testo in chiaro noto. Ora l'attaccante può scegliere testi in chiaro preparati appositamente, scelti per facilitare l'attacco di un sistema. L'attaccante può scegliere un numero qualsiasi di testi in chiaro e ottenere i testi cifrati corrispondenti. Di nuovo, non è un caso irrealistico; esistono parecchie situazioni in cui un attaccante può scegliere i dati da cifrare. Spesso Alice otterrà delle informazioni da qualche sorgente esterna (che potrebbe essere influenzata dall'attaccante) e quindi le inoltrerà a Bob in forma cifrata. Per esempio, un attaccante potrebbe inviare a Alice una email sapendo che Alice la inoltrerà a Bob.

Un buon algoritmo di cifratura non ha problemi a opporsi a un attacco con testo in chiaro scelto, ma questo tipo di attacco non va affatto considerato impossibile. Siate molto scettici se qualcuno dovesse tentare di convincervi che un attacco di questo tipo non può riguardare il suo sistema.

Esistono due varianti di questo attacco. Nel caso dell'attacco offline, l'attaccante prepara una lista di tutti i testi in chiaro che vuole far cifrare per ottenere i testi cifrati. Nell'attacco online, l'attaccante sceglie nuovi testi in chiaro in base ai testi cifrati già ricevuti. Nella maggior parte dei casi questa distinzione può essere ignorata. Normalmente si parlerà della versione online dell'attacco, la più potente.

2.6.4 Il modello con testo cifrato scelto

Parlare di *testo cifrato scelto* in realtà è fuorviante. Questo tipo di attacco si dovrebbe chiamare testo cifrato e testo in chiaro scelti, ma sarebbe un nome troppo lungo. In un attacco di questo tipo l'attaccante può scegliere sia i testi in chiaro, sia i testi cifrati. Per ogni testo in chiaro scelto, ottiene il testo cifrato corrispondente, e per ogni testo cifrato scelto, ottiene il corrispondente testo in chiaro.
Ovviamente l'attacco con testo cifrato scelto è più potente di un attacco con testo in chiaro scelto, dal momento che l'attaccante ha maggiore libertà. L'obiettivo è ancora il recupero della chiave, con cui è possibile decifrare i testi cifrati. Di nuovo, qualsiasi schema di crittografia ben fatto non ha problemi a contrastare un attacco di questo tipo.

2.6.5 L'obiettivo dell'attacco discriminante

Gli attacchi descritti in precedenza recuperano il testo in chiaro o la chiave di decifratura, mentre esistono altri attacchi che non recuperano la chiave, ma consentono di decifrare un altro specifico messaggio. Ci sono anche attacchi che non recuperano un messaggio, ma rivelano alcune informazioni parziali su di esso. Per esempio, dati dieci testi in chiaro scelti e un undicesimo testo cifrato, potrebbe essere possibile capire se il bit meno significativo dell'undicesimo testo cifrato è 1 o 0, anche se non è possibile ricavare la chiave di decifratura corrispondente. Questo tipo di informazione può essere molto preziosa per un attaccante.
Esistono troppe forme di attacco, e nuove forme vengono elaborate continuamente. Quindi, da cosa ci dovremmo difendere?
È auspicabile difendersi da un *attacco discriminante*, ovvero da qualsiasi metodo non banale che rilevi una differenza tra lo schema di cifratura ideale e quello in uso. Questo include tutti gli attacchi discussi finora, così come tutti gli attacchi ancora da scoprire. Naturalmente occorre definire quale sia lo schema ideale, cosa che non abbiamo ancora fatto. Nel prossimo capitolo cominceremo a chiarire meglio il tema.
Forse tutto ciò vi sembrerà piuttosto improbabile, ma in realtà non è così. La nostra esperienza mostra che è necessario che i componenti di base siano perfetti. Alcune funzioni di crittografia hanno imperfezioni che le rendono vulnerabili agli attacchi discriminanti, ma a parte questo sono perfettamente soddisfacenti. Ogni volta che si utilizzano tali funzioni, ci si deve assicurare che queste imperfezioni non portino a qualche problema. In un sistema con diversi componenti di base, si deve anche controllare se ogni combinazione di imperfezioni possa causare problemi. Tutto ciò diventa presto insostenibile, e nella pratica abbiamo trovato sistemi reali che soffrono di debolezze dovute a imperfezioni conosciute nei loro componenti di base.

2.6.6 Altri tipi di attacchi

Finora abbiamo parlato principalmente di attacchi a funzioni di cifratura. Si possono anche definire attacchi verso altre funzioni crittografiche, come l'autenticazione, le firme digitali e così via. Discuteremo queste forme quando le incontreremo.
Anche per le funzioni di cifratura ci siamo limitati a discutere i modelli di attacco di base, in cui un attaccante conosce o sceglie i testi in chiaro o i testi cifrati. A volte l'attaccante sa anche quando i testi cifrati sono stati generati, oppure conosce la velocità delle operazioni di cifratura e decifratura. Le informazioni relative al tempo e alla lunghezza del testo cifrato possono rivelare dati riservati sui testi crittografati. Gli attacchi che fanno uso di queste informazioni aggiuntive sono chiamati *information leakage (fuga di informazioni)* o *side-channel (canale laterale)*.

2.7 Entriamo nei dettagli

Ora esaminiamo più in dettaglio due generiche tecniche di attacco.

2.7.1 Attacchi del compleanno

Gli *attacchi del compleanno* sono chiamati così dal cosiddetto *paradosso del compleanno*. Se ci sono 23 persone in una stanza, la possibilità che due di esse compiano gli anni nello stesso giorno è superiore al 50%. Questa è una probabilità sorprendentemente alta, dato che ci sono 365 possibili giorni di compleanno.
L'attacco del compleanno si basa sul fatto che valori duplicati, chiamati anche *collisioni*, si presentano molto più rapidamente di quanto ci si aspetterebbe. Supponete che un sistema per transazioni finanziarie sicure utilizzi una nuova chiave di autenticazione a 64 bit per ogni transazione (per semplicità, supponiamo che non sia utilizzata alcuna cifratura). Ci sono 2^{64} ($=18 \cdot 10^{18}$, diciotto miliardi di miliardi) di possibili valori della chiave, quindi dovrebbe essere piuttosto difficile da violare, giusto? Sbagliato! Dopo aver visto circa 2^{32} ($=4 \cdot 10^{9}$, quattro miliardi) di transazioni, un attaccante può aspettarsi che due transazioni abbiano la stessa chiave. Supponete che il primo messaggio autenticato sia sempre lo stesso: "Sei pronto a ricevere una transazione?". Se due transazioni hanno la stessa chiave di autenticazione, allora anche i valori MAC nei loro primi messaggi saranno gli stessi, cosa semplice da scoprire per un attaccante. Sapendo che le due chiavi sono le stesse, un attaccante può ora inserire i messaggi della transazione più vecchia nella nuova transazione, mentre questa è in corso. Dato che la chiave di autenticazione è corretta, questi falsi messaggi vengono accettati, cosa che costituisce una chiara violazione del sistema di transazioni finanziarie.
In generale, se un elemento può avere N diversi valori, ci si può aspettare la prima collisione dopo aver scelto circa $\sqrt{N}$ elementi casuali. Tralasciamo i dettagli esatti, ma $\sqrt{N}$ è abbastanza vicino al valore reale. Per il paradosso del compleanno abbiamo $N = 365$ e $\sqrt{N} \approx 19$. Il numero di persone necessarie prima che la probabilità di un giorno di compleanno duplicato superi il 50% è in effetti 23, ma $\sqrt{N}$ è abbastanza vicino per i nostri scopi e in crittografia si utilizza spesso questa approssimazione. Un modo di vedere le cose è il seguente: se si scelgono k elementi, allora ci sono $k(k-1)/2$ coppie di elementi, ognuna delle quali ha probabilità pari a $1/N$ di essere una coppia di valori uguali. Perciò

la probabilità di trovare una collisione è vicina a $k(k-1)/2N$. Quando $k \approx \sqrt{N}$, questa probabilità è vicina al 50% (si tratta solo di approssimazioni, ma sono sufficientemente buone per i nostri scopi).
Nella maggior parte dei casi parliamo di valori a n bit. Dato che ci sono 2^n possibili valori, servono quasi $\sqrt{2^n} = 2^{n/2}$ elementi nell'insieme prima di potersi attendere una collisione. Parleremo spesso di questo limite $2^{n/2}$, chiamato *limite del compleanno*.

2.7.2 Attacchi meet-in-the-middle

Gli *attacchi meet-in-the-middle* sono parenti degli attacchi del compleanno (nel loro insieme li chiamiamo *attacchi di collisione*). Sono, però, più comuni e più efficaci. Invece di attendere che una chiave si ripeta, l'attaccante può costruire una tabella di chiavi scelta da lui stesso.
Torniamo al nostro esempio precedente sul sistema di transazioni finanziarie che utilizza una nuova chiave a 64 bit per autenticare ogni transazione. Utilizzando un attacco meet-in-the-midde, l'attaccante può violare il sistema anche più a fondo. Ecco come: sceglie 2^{32} diverse chiavi casuali a 64 bit. Per ognuna di queste chiavi, calcola il MAC per il messaggio:"Sei pronto per ricevere una transazione?" e memorizza sia il MAC che la chiave in una tabella. Poi intercetta ogni transazione e verifica se il MAC del primo messaggio appare nella tabella; se appare, allora c'è una buona probabilità che la chiave di autenticazione per quella transazione sia la stessa chiave che l'attaccante ha utilizzato per calcolare l'elemento della tabella, e quel valore della chiave è memorizzato proprio a fianco del valore MAC nella tabella. Ora che l'attaccante conosce la chiave di autenticazione, può inserire messaggi arbitrari di sua scelta nella transazione (l'attacco del compleanno gli permetteva solo di inserire messaggi da una vecchia transazione).
Quante transazioni deve osservare l'attaccante? Ha precalcolato il MAC su una tra 2^{32} possibili chiavi, quindi ogni volta che il sistema sceglie una chiave, c'è una probabilità su 2^{32} di scegliere una di quelle che l'attaccante può riconoscere. Dopo circa 2^{32} transazioni l'attaccante si può aspettare una transazione che utilizza una chiave per cui ha già precalcolato un MAC. Il carico di lavoro totale per l'attaccante è pari a circa 2^{32} passaggi di precalcolo oltre all'osservazione di 2^{32} transazioni, una quantità di lavoro molto inferiore rispetto a quella necessaria per provare tutte le 2^{64} possibili chiavi.
La differenza tra l'attacco del compleanno e l'attacco meet-in-the-middle è che in un attacco del compleanno si aspetta che un singolo valore si ripeta due volte in uno stesso insieme di elementi, mentre in un attacco meet-in-the-middle si hanno due insiemi e si aspetta una sovrapposizione tra di essi. In entrambi i casi ci si può aspettare di trovare il primo risultato dopo lo stesso numero di elementi.
Un attacco meet-in-the-middle è più flessibile di un attacco del compleanno. Esaminiamolo da un punto di vista più astratto. Supponiamo di avere N possibili valori. Il primo insieme ha P elementi, il secondo ha Q elementi. Ci sono coppie PQ di elementi, e ogni coppia ha una probabilità di corrispondenza pari a $1/N$. Ci si aspetta una collisione appena PQ/N è vicino a 1. La scelta più efficiente è $P \approx Q \approx \sqrt{N}$. Questo è esattamente, di nuovo, il limite del compleanno. L'attacco meet-in-the-middle fornisce una maggiore flessibilità. A volte è più semplice ottenere elementi per uno degli insiemi piuttosto che per l'altro. Il solo requisito è che PQ sia vicino a N. Si potrebbero scegliere $P \approx N^{1/3}$ e $Q \approx N^{2/33}$. Nell'esempio precedente, l'attaccante potrebbe preparare una lista di 2^{40}

possibili valori MAC per il primo messaggio e aspettarsi di trovare la prima chiave di autenticazione dopo aver osservato solo 2^{24} transazioni.
Quando analizziamo la facilità con cui si possa attaccare un sistema, utilizziamo spesso una dimensione pari a $\sqrt{N}$ per entrambi gli insieme, perché questo generalmente minimizza il numero di passaggi che l'attaccante deve eseguire. Ciò richiede anche un'analisi più dettagliata per verificare se gli elementi di un insieme possano essere più difficili da ottenere rispetto a quelli di un altro insieme. Se voleste provare a eseguire realmente un attacco meet-in-the-middle, dovreste scegliere accuratamente le dimensioni degli insiemi per assicurarvi che $PQ \approx N$ al minor costo possibile.

2.8 Livelli di sicurezza

Lavorando a sufficienza, qualsiasi sistema crittografico reale può essere attaccato con successo. La vera domanda è quanto lavoro sia necessario per violare un sistema. Un modo semplice per quantificare il carico di lavoro necessario a un attaccante è quello di confrontarlo con una ricerca esaustiva. Un *attacco con ricerca esaustiva* prova tutti i possibili valori per un obiettivo, come la chiave. Se un attacco richiede 2^{235} passaggi, questo valore corrisponde a una ricerca esaustiva per un valore a 235 bit.
Parliamo spesso di un attaccante che ha bisogno di un certo numero di passaggi, ma non abbiamo ancora specificato che cosa sia un passaggio, in parte per pigrizia, ma anche perché così si semplifica l'analisi. Quando si attacca un sistema di cifratura, calcolare una singola cifratura di un dato messaggio con una data chiave può essere considerato come un singolo passaggio. A volte il passaggio è la semplice lettura di un dato in una tabella. Insomma, è un aspetto che può variare, ma in ogni caso un passaggio può essere eseguito da un computer in un tempo molto breve. A volte può essere eseguito in un ciclo di clock, a volte richiede un milione di cicli di clock, ma nei termini di carico di lavoro che un attacco crittografico richiede, un singolo fattore di un milione non è poi così importante. La facilità di usare un'analisi basata sui passaggi compensa ampiamente le imprecisioni intrinseche. Si può sempre eseguire un'analisi più dettagliata per capire quanto lavoro sia richiesto per un passaggio. Per una stima veloce, si assume sempre che un singolo passaggio richieda un singolo ciclo di clock.
Qualsiasi sistema progettato oggi ha bisogno almeno di un livello di sicurezza a 128 bit. Questo significa che ogni attacco avrà bisogno di almeno 2^{128} passaggi. Un nuovo sistema progettato oggi, se ha successo, ha una buona probabilità di essere operativo ancora tra 30 anni, e dovrebbe fornire almeno 20 anni di riservatezza per i dati dopo il momento in cui non sarà più utilizzato. Quindi si deve puntare a fornire sicurezza per i prossimi 50 anni. È un obiettivo elevato, ma si è lavorato per estrapolare la legge di Moore e applicarla alla crittografia. Un livello di sicurezza di 128 bit è sufficiente [85]. Qualcuno potrebbe anche sostenere che 100 bit siano sufficienti, o magari 110 bit, ma le primitive crittografiche sono spesso progettate attorno alle potenze di due, quindi utilizziamo 128 bit.
Questo concetto di livello di sicurezza è solo approssimato. Misuriamo solo il lavoro totale che un attaccante deve svolgere, e ignoriamo aspetti come la memoria o le interazioni con il sistema sul campo. Pensare solo al carico di lavoro di un attacco è già piuttosto difficile; complicare il modello renderebbe l'analisi di sicurezza ancora più complessa e aumenterebbe di molto la possibilità di tralasciare un punto vitale. Dato che il costo

di utilizzare un approccio semplice e conservativo è relativamente basso, utilizziamo il semplice concetto di livello di sicurezza, che comunque è funzione dell'accesso a disposizione dell'avversario: è limitato al modello con testo in chiaro noto o può operare seguendo il modello con testo in chiaro scelto? E quanti messaggi cifrati può vedere nel suo attacco?

2.9 Prestazioni

La sicurezza non si ottiene gratis. Mentre gli esperti di crittografia cercano di rendere gli algoritmi di crittografia più efficienti possibile, questi algoritmi sono a volte percepiti come troppo lenti. Creare una crittografia ad hoc per ottenere maggiore efficienza può essere molto rischioso. Nell'ambito della sicurezza, se si devia dal percorso segnato, si dovrà eseguire un enorme lavoro di analisi per assicurarsi di non creare accidentalmente un sistema debole. Tale analisi richiede crittografi esperti. Per la maggior parte dei sistemi, è molto più economico acquistare un computer più veloce piuttosto che andare incontro a problemi e spese nel tentativo di progettare e implementare un sistema di sicurezza più efficiente.

Inoltre, per la maggior parte dei sistemi le prestazioni della crittografia non rappresentano un problema. Le CPU moderne sono talmente veloci da poter sostenere qualsiasi flusso di dati che gestiscono. Per esempio, cifrare un collegamento a 100 Mb/s con l'algoritmo AES richiede solo il 20% dei cicli macchina di una CPU Pentium III a 1 GHz (in realtà meno, dato che non si trasferiscono mai 100 Mb/s su tale collegamento, a causa del carico aggiuntivo del protocollo di comunicazione).

Esistono tuttavia alcune situazioni in cui la crittografia crea un collo di bottiglia rilevante. Un valido esempio sono i web server che utilizzano un gran numero di connessioni SSL. L'inizializzazione di una connessione SSL usa la crittografia a chiave pubblica e richiede una grande potenza di calcolo lato server. Invece di sviluppare un sostituto di SSL che sia più efficiente per il server, è molto più economico e sicuro acquistare componenti hardware in grado di gestire il protocollo SSL esistente in modo più rapido.

Recentemente abbiamo trovato una buona argomentazione per convincere le persone a scegliere la sicurezza piuttosto che le prestazioni. "Ci sono già abbastanza sistemi veloci e insicuri; non abbiamo bisogno di un altro". Questo è decisamente vero. Le mezze misure in sicurezza costano quasi quanto fare le cose al meglio, ma forniscono ben poca sicurezza pratica. Crediamo fermamente che, se si implementa qualche aspetto che abbia a che fare con la sicurezza, lo si debba fare bene.

2.10 Complessità

Più complesso è un sistema, più probabilmente avrà problemi di sicurezza. Ci piace dire che la complessità è il peggior nemico della sicurezza. È un'affermazione semplice, ma ci è servito un po' di tempo per capirla veramente.

Una parte del problema è il processo di sviluppo per test e correzioni utilizzato fin troppo frequentemente: si compila qualcosa, si eseguono test per individuare errori, si torna indietro a correggere gli errori, si cercano altri errori, e così via. Controlla, correggi, ripeti. E si prosegue così finché le finanze della società o altri fattori impongono che il

prodotto sia consegnato. Di sicuro, il risultato funzionerà ragionevolmente bene, se viene utilizzato solo per le funzioni per cui è stato verificato.
Questo metodo potrebbe essere sufficiente per quanto riguarda la funzionalità, ma è del tutto inadeguato per i sistemi di sicurezza.
Il problema del metodo per test e correzioni è che il test mostra solo la presenza di errori, e in realtà solo di quegli errori che chi esegue il test sta cercando. I sistemi di sicurezza devono lavorare anche sotto l'attacco di persone capaci e male intenzionate. Il sistema non può essere testato per tutte le situazioni a cui gli attaccanti potrebbero esporlo. La fase di test può soltanto verificare la funzionalità; la sicurezza è assenza di funzionalità. L'attaccante non dovrebbe essere in grado di ottenere una certa proprietà a prescindere da che cosa faccia, e il test non può mostrare l'assenza di funzionalità. Il sistema deve essere sicuro fin dall'inizio.
Considerate la seguente analogia. Supponete di scrivere un'applicazione di medie dimensioni in un linguaggio di programmazione diffuso. Gli errori vengono corretti fino a ottenere una prima compilazione. Quindi, senza ulteriori verifiche, impacchettate il programma e lo consegnate all'utente. Nessuno si aspetterebbe di ottenere un prodotto funzionale procedendo in questo modo.
Tuttavia, questo è esattamente ciò che si fa normalmente per i sistemi di sicurezza. Tali sistemi sono impossibili da verificare perché nessuno sa come testarli. Per definizione, un attaccante ha successo quando trova un aspetto che non è stato testato. E se c'è qualche bug, il prodotto è difettoso. Quindi, il solo modo per ottenere un sistema sicuro consiste nel costruire un sistema molto robusto fin dalle fondamenta. E ciò richiede un sistema semplice.
Il solo modo che conosciamo per costruire un sistema semplice è quello di costruirlo in maniera modulare. Lo sappiamo tutti dallo sviluppo del software. Ma in questo caso non possiamo permetterci nessun bug, quindi dobbiamo essere piuttosto spietati nella modularizzazione. Questo porta a un'altra regola: la correttezza deve essere una proprietà locale. In altre parole, una parte del sistema dovrebbe comportarsi correttamente indipendentemente dal fatto che il resto del sistema funzioni. No, non vogliamo sentire affermazioni del tipo: "Questo non sarà un problema perché quest'altra parte del sistema non permetterà mai che accada". L'altra parte potrebbe avere un bug, o potrebbe cambiare in una versione futura. Ogni parte del sistema è responsabile del proprio funzionamento.

2.11 Esercizi

Esercizio 2.1 Considerate il principio di Kerckhoff. Fornite almeno due argomenti a favore del principio di Kerckhoff e almeno due argomenti a sfavore. Quindi riportate e argomentate il vostro punto di vista sulla validità del principio in questione.

Esercizio 2.2 Supponete che Alice e Bob stiano scambiandosi email via Internet. Stanno inviando queste email dai loro computer portatili, che sono connessi a reti wireless gratuite fornite dai loro pub preferiti. Elencate tutti i soggetti che potrebbero essere in grado di attaccare il sistema e che cosa potrebbero essere in grado di ottenere. Descrivete nella maniera più specifica possibile come Alice e Bob possano difendersi da ciascun tipo di attacco identificato.

Esercizio 2.3 Considerate un gruppo di 30 persone che vogliono stabilire connessioni a coppie sicure utilizzando la crittografia a chiave simmetrica. Quante chiavi devono essere scambiate in totale?

Esercizio 2.4 Supponete che Bob riceva un messaggio firmato utilizzando uno schema di firma digitale con la chiave di firma segreta di Alice. Questo prova che Alice abbia visto il messaggio in questione e abbia scelto di firmarlo?

Esercizio 2.5 Supponete che le PKI, come descritte nel Paragrafo 2.5, non esistano. Supponete che Alice ottenga una chiave pubblica P che sembrerebbe appartenere a Bob. Come può Alice essere sicura che P appartenga realmente a Bob? Considerate la domanda in ognuno dei seguenti scenari:

- Alice può parlare al telefono con Bob.
- Alice può parlare al telefono con qualcun altro di cui si fida (chiamiamolo Charlie) e Charlie ha già verificato che P appartiene a Bob.
- Alice può comunicare con Charlie via email, e Charlie ha già verificato che P appartiene a Bob.

Esponete in modo esplicito ogni ipotesi aggiuntiva che occorre fare.

Esercizio 2.6 Supponete che un attacco di tipo con testo cifrato scelto non possa recuperare la chiave di decifratura segreta per uno schema di cifratura. Questo significa che lo schema di cifratura è sicuro?

Esercizio 2.7 Considerate un sistema crittografico a chiave simmetrica nel quale le chiavi crittografiche sono selezionate in maniera causale da un insieme di tutte le stringhe a n bit. Quanto dovrebbe essere grande n, approssimativamente, per fornire 128 bit di sicurezza contro un attacco del compleanno?