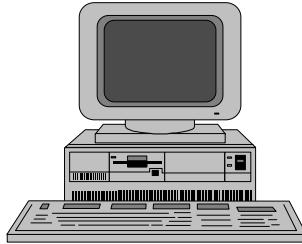


Insegnamento di **Sistemi ed Algoritmi per la Protezione dei Dati 1**

(Corso di Laurea Specialistica in Ingegneria Elettronica)



Introduzione alla crittologia **Protocolli**

Rodolfo Zunino

Edizione 2004



Scuola di Elettronica
DIBE – Università degli Studi di Genova

Indice

1 Gli elementi fondamentali

1.1 Introduzione ai protocolli

- 1.1.1 Lo scopo dei protocolli
- 1.1.2 I personaggi
- 1.1.3 Protocolli con arbitro
- 1.1.4 Protocolli con giudice
- 1.1.5 Protocolli self-enforcing
- 1.1.6 Attacchi contro i protocolli

1.2 One-way functions

1.3 One-way hash functions

- 1.3.1 Message Authentication Codes

1.4 Generazione di sequenze casuali e pseudo-casuali

- 1.4.1 Sequenze pseudo-casuali
- 1.4.2 Sequenze pseudo-casuali crittograficamente sicure

1.5 Protocolli che utilizzano la crittografia simmetrica

1.6 Protocolli che utilizzano la crittografia a chiave pubblica

1.7 Sistemi crittografici ibridi

1.8 Firme digitali

- 1.8.1 Algoritmi e terminologia

2 Protocolli

2.1 Scambio delle chiavi

- 2.1.1 Scambio delle chiavi mediante crittografia simmetrica
- 2.1.2 Scambio delle chiavi mediante crittografia a chiave pubblica
- 2.1.3 Man-in-the-middle attack
- 2.1.4 Protocollo interlock
- 2.1.5 Scambio delle chiavi con firme digitali

2.2 Servizi di timestamping (o di certificazione del tempo)

2.2.1 Soluzioni con arbitro

2.2.1.2 Soluzione con arbitro migliorata

2.2.2 Protocollo di collegamento (o di linking)

2.2.2 Protocollo distribuito

2.3 Firme digitali

2.3.1 Firma di documenti mediante crittografia a chiave privata con arbitro

2.3.2 Firma di documenti mediante crittografia a chiave pubblica

2.3.3 Firma di documenti mediante crittografia a chiave pubblica e documento in cifra

2.3.4 Firma di documenti mediante crittografia a chiave pubblica e one-way hash function

2.3.5 Resend attack

2.3.6 Difendersi dal resend attack

2.3.7 Firme multiple

2.3.8 Firme digitali e possibilità di ripudiare

2.3.9 Undeniable signature

2.3.10 Firme di gruppo

2.3.10.1 Firme di gruppo con arbitro

2.3.11 Firme digitali fail-stop

2.4 Key Distribution Center

2.5 Autenticazione

2.5.1 Autenticazione mediante one-way function

2.5.2 Dictionary attack e salt

2.5.3 Autenticazione mediante crittografia a chiave pubblica

2.6 Bit commitment

2.6.1 Bit commitment mediante crittografia simmetrica

2.6.2 Bit commitment mediante generatori di sequenze pseudo-casuali

2.6.3 Bit commitment mediante hashing

2.7 Prove a conoscenza nulla

2.7.1 Protocollo di base

2.7.2 Isomorfismo tra grafi

2.8 Firme alla cieca (blind signature)

2.8.1 Firme completamente alla cieca

2.8.2 Firme alla cieca

2.9 Digital cash

2.9.1 Protocollo #1

2.9.2 Protocollo #2

2.9.3 Protocollo #3

2.9.4 Protocollo #4

2.10 Voto elettronico

2.10.1 Protocollo #1

2.10.2 Protocollo #2

2.10.3 Protocollo #3

1

Gli elementi fondamentali

1.1 Introduzione ai protocolli

La crittografia ha come fine ultimo la risoluzione di problemi pratici che coinvolgono la segretezza, l'autenticazione e l'integrità. Gli algoritmi crittografici, però, da soli non servono a nulla e per qualunque applicazione concreta sono necessari dei protocolli.

Un **protocollo** è una serie di passi che coinvolge due o più parti e che serve a portare a termine un compito. Quindi un protocollo ha una sequenza, con un inizio ed una fine, ed ogni passo deve essere eseguito in ordine; un protocollo, per essere tale, deve coinvolgere più di una persona e, soprattutto, deve servire a fare qualcosa, altrimenti è solo una perdita di tempo.

Oltre a queste caratteristiche fondamentali, un protocollo deve averne altre:

- Chiunque partecipi al protocollo deve conoscerlo.
- Chiunque partecipi al protocollo deve accettare di seguirlo.
- Il protocollo deve essere privo di ambiguità, ogni passo deve essere descritto con chiarezza e non ci devono essere possibilità di incomprensioni.
- Il protocollo deve essere completo, quindi deve essere prevista un'azione per qualunque situazione.

Per essere un protocollo di “security” deve avere un'ulteriore proprietà:

- Non deve fornire più informazioni di quelle per cui è stato concepito

Un **protocollo crittografico** è un protocollo che usa la crittografia. Per questo fa uso di un qualche algoritmo di crittografia, ma, come vedremo, solitamente l'obiettivo del protocollo va al di là della semplice segretezza. In generale, comunque, l'uso della crittografia all'interno di un protocollo serve a prevenire o almeno individuare l'eavesdropping (to eavesdrop significa origliare) e il cheating (to cheat significa ingannare).

1.1.1 Lo scopo dei protocolli

Nella vita di tutti i giorni ci sono protocolli informali per quasi ogni cosa: giocare a poker, ordinare dei beni telefonicamente, votare alle elezioni. Al giorno d'oggi però, un numero sempre crescente di relazioni umane non avviene più faccia a faccia, ma su reti di computer, ed i computer, per fare quello che le persone fanno senza neanche pensarci, hanno bisogno di protocolli formali.

Nelle relazioni faccia a faccia la sicurezza e la correttezza sono fondate sulla presenza della persona, ma questo naturalmente non è possibile quando si comunica con reti di computer. Per questo nel progettare un protocollo si dovranno considerare potenzialmente disonesti, non solo gli utilizzatori, ma anche gli amministratori ed i progettisti della rete.

1.1.2 I personaggi

Per illustrare i vari protocolli utilizzeremo alcuni personaggi elencati qui sotto. Alice e Bob sono i più importanti e sono sempre presenti, mentre gli altri appaiono solo in alcuni casi.

- ALICE: prima partecipante in tutti i protocolli.
- BOB: secondo partecipante in tutti i protocolli.
- CAROL: partecipante ai protocolli con tre o quattro parti.
- DAVE: partecipante ai protocolli con quattro parti.
- EVE: eavesdropper, attacca passivamente il protocollo.
- MALLORY: malicious active attacker, attacca attivamente il protocollo.
- TRENT: trusted arbitrator, persona di cui tutti si fidano.
- PEGGY: prover, in alcuni algoritmi deve dimostrare di sapere qualcosa.
- VICTOR: verifier, verifica le prove fornite da Peggy.

1.1.3 Protocolli con arbitro

Un **arbitro** (Trent) è una terza parte disinteressata di cui tutti si fidano. Disinteressata significa che non ha particolari legami con nessuna delle parti coinvolte; fidata significa che tutte le persone che partecipano al protocollo accettano come vero ciò che dice e come corretto ciò che fa. Gli arbitri aiutano a completare protocolli tra parti reciprocamente diffidenti. Nella vita reale esempi comuni di arbitri possono essere avvocati, notai o banche; purtroppo però quello che avviene in questi casi non può essere direttamente trasportato nel mondo dei computer.

I principali problemi sono:

- È più facile fidarsi di una terza parte neutrale se la si conosce e se la si può vedere in faccia; inoltre è probabile che due parti reciprocamente diffidenti diffidino anche di un arbitro senza volto che si trova da qualche parte nella rete.
- La rete di computer deve sostenere il costo di mantenimento dell'arbitro (e tutti conoscono le parcelle degli avvocati).
- I protocolli con arbitro introducono inevitabilmente dei ritardi.

- L'arbitro deve intervenire in ogni transizione, quindi rappresenta un potenziale collo di bottiglia, soprattutto nelle implementazioni su larga scala. Si può pensare di aumentare il numero di arbitri, ma ciò fa crescere il costo.
- L'arbitro rappresenta un punto molto vulnerabile per chi vuole sovvertire la rete.

Ciononostante i protocolli con arbitro hanno le loro applicazioni pratiche.

1.1.4 Protocolli con giudice

Un **giudice** è una terza parte disinteressata e fidata, come l'arbitro, ma a differenza di questo, non partecipa direttamente ad ogni protocollo: viene chiamato in causa solo in caso di una disputa. Questi protocolli si basano sull'onestà delle parti in causa: se qualcuno però sospetta che ci siano stati degli imbrogli, chiama in causa il giudice. Se il protocollo è progettato correttamente, il giudice è in grado di stabilire con certezza se ci sono stati degli imbrogli e da parte di chi: la possibilità di essere smascherati funziona quindi da deterrente.

1.1.5 Protocolli self-enforcing

Un protocollo self-enforcing è il miglior tipo di protocollo, in quanto è il protocollo stesso a garantire l'imparzialità e la correttezza. Non sono necessari né arbitri, né giudici. Il protocollo è progettato in maniera che non ci possano essere dispute, infatti se una delle parti prova a imbrogliare, l'altra se ne accorge immediatamente e il protocollo termina. In un mondo ideale ogni protocollo sarebbe self-enforcing, ma sfortunatamente non ne esiste uno per ogni situazione.

1.1.6 Attacchi contro i protocolli

Gli attacchi crittografici possono essere condotti contro gli algoritmi crittografici, contro le tecniche usate per implementare gli algoritmi o contro i protocolli crittografici: noi consideriamo i primi due livelli sicuri e analizziamo solo gli attacchi contro i protocolli.

Per prima cosa qualcuno non coinvolto nel protocollo (Eve) può ascoltare di nascosto ciò che viene trasmesso (eavesdropping): questo viene chiamato **passive attack** perché l'aggressore non influisce sul protocollo.

Alternativamente un aggressore (Mallory) può cercare di modificare il protocollo a proprio vantaggio: può fingere d'essere qualcun altro, cancellare messaggi, sostituire un messaggio con un altro, ripetere vecchi messaggi, interrompere un canale di comunicazione o alterare informazioni memorizzate in un computer. Questi vengono chiamati **active attack** perché richiedono un intervento attivo.

I passive attacker possono solo raccogliere messaggi e cercare di decifrarli tramite crittoanalisi. Gli active attacker invece possono fare molto di più: possono essere interessati ad acquisire informazioni, a degradare le prestazioni del sistema, a corrompere dati esistenti o ad ottenere accessi non autorizzati.

E' anche possibile che l'attacker sia una delle parti coinvolte nel protocollo: in tal caso può mentire durante lo svolgimento del protocollo o non seguirlo affatto. Questo tipo di attacker è chiamato **cheater**. I passive cheater seguono il protocollo, ma cercano di ottenere più informazioni di quanto gli sia concesso. Gli active cheater invece infrangono il protocollo per cercare di imbrogliare.

Purtroppo è molto difficile mantenere sicuro un protocollo, soprattutto quando quasi tutte le parti coinvolte sono active cheater, ma talvolta si riesce almeno ad individuare la presenza di active cheating. Naturalmente i protocolli dovrebbero essere sicuri contro il passive cheating.

1.2 One-way functions

La nozione di **one-way function** è fondamentale per la crittografia a chiave pubblica. Sebbene non siano protocolli, le one-way function sono un elemento fondamentale per la maggior parte dei protocolli. Le one-way function sono relativamente facili da calcolare, ma molto più difficili da invertire. In questo contesto "difficile" ha più o meno questo significato: sarebbero necessari milioni di anni per calcolare x nota $f(x)$, anche se fossero utilizzati tutti i computer del mondo. La rottura di un piatto è un buon esempio di one-way function. E' facile frantumare un piatto in centinaia di pezzi, ma non altrettanto rimetterli insieme per ottenere nuovamente un piatto.

Tutto ciò è interessante, ma da un punto di vista strettamente matematico non ci sono prove che le one-way function esistano o possano essere costruite. Ciononostante, molte funzioni sembrano essere di questo tipo: possiamo calcolare in maniera efficiente e, almeno finora, non conosciamo nessun modo per invertirle agevolmente. Ad esempio x^2 è facile da calcolare, mentre $x^{1/2}$ è molto più complesso. Nel seguito, ad ogni modo, si supporrà che le one-way function esistano.

Le one-way function, comunque, non possono essere usate direttamente: un messaggio cifrato con una one-way function è inutile, in quanto nessuno può decifrarlo. Quindi, per la crittografia a chiave pubblica, abbiamo bisogno di qualcos'altro.

Una **trapdoor one-way function** è un particolare tipo di one-way function, con un trabocchetto segreto. E' facile da calcolare in una direzione e difficile nell'altra, ma, se si conosce il segreto, l'inversione non è più un problema.

1.3 One-way hash functions

Una **one-way hash function** ha molti nomi (to hash significa tritare): funzione di compressione, funzione di contrazione, impronta digitale, checksum (somma di controllo) crittografica, verifica di integrità del messaggio, codice di individuazione di manipolazione. Comunque la si chiami, questo tipo di funzione è fondamentale nella crittografia moderna.

Le hash function sono usate da molto tempo nel mondo dei computer. Una hash function è una funzione che prende una stringa di ingresso di lunghezza variabile (chiamata **pre-image**) e la converte in una di uscita (chiamata **hash value**) di lunghezza fissa (generalmente inferiore). Un semplice esempio di hash function è una funzione che prende la pre-image e restituisce un byte ottenuto dallo XOR di tutti i byte d'ingresso.

Una one-way hash function è una hash function che funziona in un solo verso: è facile calcolare il valore di hash da una pre-image, ma è difficile generare una pre-image che faccia ottenere un ben preciso valore di hash. La hash function vista prima non è di tipo one-way, infatti, dato un byte, è banale generare una stringa i cui byte abbiano come XOR il byte desiderato. Con una one-way hash function ciò non è possibile; inoltre una buona one-way hash function è anche **collision-free** (senza collisioni), cioè è difficile generare due pre-image con lo stesso valore di hash.

La hash function è pubblica, la sicurezza risiede nella sua unilaterialità. L'uscita dipende dall'ingresso in modo indistinguibile. La variazione di un bit nella pre-image, modifica in media metà dei bit del valore di hash. Dato un valore di hash è computazionalmente impossibile trovare una pre-image che lo generi.

Grazie a queste funzioni si può pensare di prendere le impronte digitali ad un file: se Alice vuole verificare che Bob abbia un certo file (che anche lei ha), ma non vuole che glielo invii, allora gli può chiedere solo il valore di hash. Se Bob invia il corretto valore di hash, Alice può essere praticamente certa che anche lui possiede quel file. Questo genere di tecnica è particolarmente utile nelle transazioni finanziarie, in cui non si vuole che, da qualche parte nella rete, un prelievo di \$100 diventi un prelievo di \$1.000.

Bisogna notare che con questo schema chiunque può verificare la correttezza del valore di hash.

1.3.1 Message Authentication Codes

Un **message authentication code** (MAC), noto anche come data authentication code (DAC), è una one-way hash function con l'aggiunta di una chiave segreta. Il valore di hash è funzione sia della pre-image, sia della chiave. La teoria è la stessa delle hash function, a parte il fatto che, solo chi conosce la chiave, può verificare il valore di hash. Si può creare un MAC da una hash function o da un algoritmo di crittografia, ma ci sono anche MAC dedicati.

1.4 Generazione di sequenze casuali e pseudo-casuali

Sembrerebbe inutile perdere tempo parlando della generazione di sequenze casuali, poiché qualunque compilatore integra già un generatore di questo tipo. Sfortunatamente, però, tali generatori di numeri casuali non sono affatto sicuri per la crittografia e, probabilmente, non sono nemmeno casuali.

I generatori di numeri casuali non sono veramente casuali perché spesso non è necessario che lo siano. La maggior parte delle applicazioni comuni, come i giochi per computer, ha bisogno di così pochi numeri casuali, che non si accorge di eventuali difetti nel generatore. Al contrario, la crittografia è estremamente sensibile alle proprietà del generatore. Usando un generatore di numeri casuali scadente, si ottengono bizzarre correlazioni e strani risultati. Se la sicurezza dipende dal generatore di numeri casuali, bizzarre correlazioni e strani risultati sono l'ultima cosa che si desidera.

Il problema è che un generatore di numeri casuali, in realtà, non produce numeri casuali. Probabilmente, non produce niente di lontanamente simile ad una sequenza di numeri casuali, anche perché è impossibile produrre qualcosa di veramente casuale su un computer.

I computer, infatti, sono oggetti deterministici: dei dati entrano da una parte, all'interno avvengono delle operazioni completamente prevedibili e altri dati escono dall'altra parte. Se si inseriscono gli stessi dati in due diverse occasioni, entrambe le volte si ottiene lo stesso risultato. Se si inseriscono gli stessi dati in due computer identici, i risultati sono identici. Un computer può essere solo in un numero finito di stati (un numero molto grande, ma comunque finito) e, ciò che ne esce, sarà sempre una funzione deterministica di ciò che è stato inserito e dello stato corrente. Ciò significa che ogni generatore di numeri casuali su un computer è, per definizione, periodico. Tutto ciò che è periodico è, per definizione, prevedibile. E se qualcosa è prevedibile, non può essere casuale. Un vero generatore di numeri casuali ha bisogno di un ingresso casuale, cosa che un computer non può fornire.

1.4.1 Sequenze pseudo-casuali

La cosa migliore che un computer può produrre è un generatore di sequenze pseudo-casuali. In modo informale, si può definire pseudo-casuale una sequenza che sembra casuale. Il periodo della sequenza dovrebbe essere abbastanza lungo da garantire che una sequenza finita di lunghezza ragionevole (che poi è quella effettivamente utilizzata) sia non periodica. Se si ha bisogno di un miliardo di bit casuali, non si può usare un generatore che ripete la stessa sequenza ogni sedicimila bit. In generale, queste sottosequenze non periodiche dovrebbero essere indistinguibili dalle sequenze casuali. Per esempio, dovrebbero avere all'incirca lo stesso numero di uni e zeri; circa metà delle serie (sequenze dello stesso bit) dovrebbero avere lunghezza uno, un quarto lunghezza due, un ottavo lunghezza tre, e così via. Non dovrebbero essere possibile comprimerle. Le distribuzioni delle lunghezze delle serie dovrebbero essere le stesse per gli uni e per gli zeri.

Per i nostri scopi, un generatore di sequenze è pseudo-casuale se ha la proprietà di sembrare casuale. Questo significa che supera tutti i test statistici di casualità che si possono trovare.

Sono stati fatti grossi sforzi per produrre delle buone sequenze pseudo-casuali su computer. Tutti i generatori sono periodici, ma con potenziali periodi di 2^{256} bit ed oltre, possono essere usati nella maggior parte delle applicazioni. Resta il problema delle bizzarre correlazioni e degli strani risultati. Ogni generatore di numeri pseudo-casuali ha questi problemi, se usato in un certo modo: questo è ciò che il crittanalista userà per attaccare il sistema.

1.4.2 Sequenze pseudo-casuali crittograficamente sicure

Per le applicazioni crittografiche la casualità statistica non è sufficiente. Una sequenza pseudo-casuale, per essere crittograficamente sicura, deve avere anche la seguente proprietà:

E' imprevedibile. Deve essere computazionalmente impossibile prevedere il prossimo bit casuale, data una completa conoscenza dei bit precedenti e dell'algoritmo che genera la sequenza.

Per concludere ci chiediamo se esistano delle sequenze veramente casuali. Filosofia a parte, per noi, un generatore di sequenze è veramente casuale se ha anche questa proprietà:

Non può essere riprodotto. Se si usa il generatore di sequenze due volte con lo stesso ingresso, i due risultati saranno completamente correlati.

Alla fine il problema rimane sempre quello di determinare se una data sequenza è veramente casuale.

1.5 Protocolli che utilizzano la crittografia simmetrica

Se due persone vogliono comunicare in sicurezza, naturalmente cifrano i messaggi, ma il protocollo non è solo questo; vediamo cosa deve fare Alice per inviare un messaggio cifrato a Bob:

1. Alice e Bob si accordano su un **cryptosystem** (sistema crittografico).
2. Alice e Bob si accordano su una chiave.
3. Alice prende il **plaintext message** (messaggio in chiaro) e lo cifra usando l'algoritmo crittografico e la chiave. In questo modo ottiene un **ciphertext message** (messaggio cifrato).
4. Alice invia il ciphertext message a Bob.
5. Bob decifra il ciphertext message con lo stesso algoritmo e la stessa chiave, quindi lo legge.

Eve, che si trova tra Alice e Bob, può sferrare un **ciphertext-only attack**, ma ci sono algoritmi che, per quanto ne sappiamo, resistono a qualunque potenza di calcolo realisticamente a disposizione di Eve. Naturalmente Eve non è stupida, quindi può cercare di ascoltare anche i passi 1 e 2 del protocollo: in tal caso, quando intercetta il ciphertext message, non fa altro che decifrarlo e leggerlo. In un buon cryptosystem la sicurezza dipende esclusivamente dalla conoscenza della chiave, mentre l'algoritmo può anche essere reso pubblico. Per questo la gestione delle chiavi riveste un ruolo molto importante. La chiave deve restare segreta prima, durante e dopo l'esecuzione del protocollo, o comunque fino a quando si vuole che il messaggio rimanga segreto. Come conseguenza di quanto detto, il passo 1 può essere eseguito in pubblico, mentre il passo 2 deve essere necessariamente realizzato in segreto.

Mallory, un active attacker, può fare parecchie altre cose. Può provare ad interrompere le comunicazioni al passo 4, impedendo ad Alice di comunicare con Bob. Può intercettare i messaggi di Alice, sostituendoli con i propri. Nel caso conosca la chiave (intercettandola al passo 2 o violando il cryptosystem), può cifrare i propri messaggi e inviarli a Bob: a questo punto Bob penserà che i messaggi arrivino da Alice. Oppure può creare un nuovo messaggio anche senza conoscere la chiave: in questo caso Bob, decifrando il messaggio, otterrà un discorso senza senso e penserà che Alice (o la rete) abbia dei seri problemi.

Naturalmente anche gli stessi Bob e Alice possono violare il protocollo, ma la crittografia simmetrica assume che i due si fidino l'uno dell'altro.

Per concludere i cryptosystem simmetrici hanno i seguenti problemi:

- Le chiavi devono essere distribuite segretamente. Questo è un problema soprattutto nelle comunicazioni a livello mondiale. Spesso le chiavi sono consegnate a mano da dei corrieri.
- Se una chiave è compromessa (rubata, indovinata, estorta, comprata...), Eve può decifrare tutti i messaggi. Può anche fingere di essere una delle parti e produrre falsi messaggi per ingannare le altre parti.
- Assumendo che sia usata una diversa chiave per ogni coppia di utenti in una rete, il numero totale di chiavi cresce rapidamente al crescere del numero di utenti: in generale una rete con n utenti richiede $n(n-1)/2$ chiavi.

1.6 Protocolli che utilizzano la crittografia a chiave pubblica

Si può pensare ad un algoritmo simmetrico come ad una cassaforte. La chiave è la combinazione. Qualcuno che conosce la combinazione può aprire la cassaforte, metterci un documento e quindi richiuderla. Qualcun altro che conosce la combinazione può riaprire la cassaforte e prendere il documento. Chi non conosce la combinazione deve invece imparare a scassinare casseforti.

Nel 1976 Diffie e Hellman cambiarono per sempre il mondo della crittografia, descrivendo la crittografia a chiave pubblica (la NSA ha sostenuto di conoscere il concetto dal 1966, ma non ne ha fornito prova). Questo rivoluzionario sistema crittografico usa due chiavi, una pubblica ed una privata. E' computazionalmente difficile dedurre la chiave privata da quella pubblica. Chiunque, usando la chiave pubblica, può cifrare un messaggio, ma non decifrarlo. Solo chi possiede la chiave privata può decifrare. E' come se la cassaforte fosse diventata una cassetta postale: chiunque può imbucare una lettera, ma solo chi ha la chiave della cassetta può ritirare la posta.

Matematicamente il processo è basato sulle trapdoor one-way function: la cifratura è la direzione facile e la chiave pubblica rappresenta le istruzioni per la cifratura, così chiunque può cifrare un messaggio. La decifratura è la direzione difficile, talmente difficile che, anche chi possiede dei computer Cray e migliaia (se non milioni) di anni, non può realizzarla, se non conosce il segreto. Il segreto, o trapdoor, è la chiave privata: se la si conosce, la decifratura è facile come la cifratura.

Questo è lo schema con cui Alice invia un messaggio a Bob usando la crittografia a chiave pubblica:

1. Alice e Bob si accordano su un cryptosystem a chiave pubblica.
2. Bob invia ad Alice la propria chiave pubblica.
3. Alice cifra il proprio messaggio usando la chiave pubblica di Bob e glielo invia.
4. Bob decifra il messaggio di Alice usando la propria chiave privata.

Si noti come la crittografia a chiave pubblica risolva il problema di gestione delle chiavi della crittografia simmetrica. Prima Alice e Bob dovevano accordarsi segretamente su una chiave, ora invece possono scambiarsi messaggi segreti senza accordi preventivi.

Solitamente gli utenti di una rete si accordano su un comune cryptosystem a chiave pubblica. Ogni utente ha la propria chiave pubblica e la propria chiave privata, e le chiavi pubbliche sono pubblicate in un database. Ora il protocollo è ancora più semplice:

1. Alice ottiene la chiave pubblica di Bob dal database.
2. Alice cifra il proprio messaggio usando la chiave pubblica di Bob e glielo invia.
3. Bob decifra il messaggio di Alice usando la propria chiave privata.

Nel primo protocollo, Bob deve inviare ad Alice la propria chiave pubblica prima che lei possa inviargli un messaggio. Il secondo invece è più simile alla posta tradizionale, infatti Bob non è coinvolto nel protocollo finché non vuole leggere il messaggio.

1.7 Sistemi crittografici ibridi

Nel mondo reale gli algoritmi a chiave pubblica non sostituiscono quelli a chiave privata. Infatti non sono usati per cifrare i messaggi, bensì per cifrare le chiavi. Questo, principalmente, per due ragioni:

1. Gli algoritmi a chiave pubblica sono lenti, almeno 1.000 volte più lenti di quelli a chiave privata.
2. I cryptosystem a chiave pubblica sono vulnerabili ai **chosen-plaintext attack**. Se $C=E(P)$, quando P è un plaintext scelto da un insieme di n elementi, un crittanalista deve solo cifrare tutti gli n possibili plaintext e confrontare i risultati con C . In questo modo non potrà ottenere la chiave privata, ma potrà determinare P .

Un chosen-plaintext attack può essere particolarmente efficace se ci sono pochi messaggi cifrati. Per esempio, se P è una cifra minore di 1.000.000, il crittanalista prova un milione di cifre diverse. Questo attacco può essere molto efficace anche quando P non è così ben definito. In certi casi, infatti, anche solo sapere che un ciphertext non corrisponde ad un particolare plaintext, può essere un'informazione utile. I sistemi simmetrici non sono vulnerabili a questo attacco, perché un crittanalista non può effettuare cifrature di prova con una chiave sconosciuta. Nella maggior parte delle implementazioni pratiche la crittografia a chiave pubblica è usata per distribuire in modo sicuro le **session key**. Tale chiavi vengono poi usate per scambiarsi messaggi tramite algoritmi simmetrici. Questo è spesso chiamato **hybrid cryptosystem**.

1. Bob invia ad Alice la propria chiave pubblica.
2. Alice genera una session key casuale, K , la cifra usando la chiave pubblica di Bob e la invia a Bob.

$$E_B(K)$$

3. Bob decifra il messaggio di Alice usando la propria chiave privata per recuperare la session key.

$$D_B(E_B(K))=K$$

4. Entrambi cifrano le loro comunicazioni usando la stessa session key.

Nota: E_B deriva dall'inglese **to encrypt** (cifrare); D_B deriva dall'inglese **to decrypt** (decifrare); il pedice B dice che si stanno usando le chiavi (pubblica e privata) di Bob.

L'uso della crittografia a chiave pubblica per la distribuzione delle chiavi risolve un importante problema di gestione delle chiavi. Con la crittografia simmetrica, la chiave esiste già prima di essere usata: se Eve riesce a metterci le mani sopra, può decifrare tutti i messaggi. La session key, invece, è creata quando serve e distrutta quando non serve più. Questo riduce drasticamente il rischio di compromettere la session key.

1.8 Firme digitali

Le firme scritte a mano sono usate da molto tempo come prova di paternità di un documento, o almeno di accordo con esso. La firma è convincente per vari motivi:

1. La firma è autentica. La firma convince il destinatario del documento che l'autore ha deliberatamente firmato il documento.
2. La firma non si può falsificare. La firma è la prova che il firmatario, e nessun altro, ha deliberatamente firmato il documento.
3. La firma non è riutilizzabile. La firma è parte del documento; nessuno può spostare la firma su un altro documento.
4. Il documento firmato è inalterabile. Dopo che il documento è stato firmato, non può essere alterato.
5. La firma non può essere ripudiata. Il firmatario non può sostenere, in seguito, di non aver firmato.

In realtà, nessuna di queste cose è proprio vera, ma siamo pronti ad accettare questa situazione perché imbrogliare non è facile e si rischia di essere scoperti.

Il nostro obiettivo è realizzare la firma tramite computer, ma ci sono dei problemi.

1.8.1 Algoritmi e terminologia

Tutti gli algoritmi di firma digitale sono a chiave pubblica, con una informazione segreta per firmare i documenti ed una informazione pubblica per verificare la firma. Talvolta il processo di firma è detto **cifratura con chiave privata** e il processo di verifica è detto **decifratura con chiave pubblica**. Tutto ciò è ingannevole ed è vero per il solo algoritmo RSA. In generale, ci riferiremo ai processi di firma e di verifica senza alcun dettaglio sugli algoritmi coinvolti. La firma di un messaggio con la chiave privata K è:

$$S_K(M)$$

La verifica di una firma con la corrispondente chiave pubblica è:

$$V_K(M)$$

La stringa di bit unita al documento quando è firmato, sarà chiamata **firma digitale**, o semplicemente **firma**. L'intero protocollo, con cui il destinatario è convinto dell'identità del mittente e dell'integrità del messaggio, è chiamato **autenticazione**.

2

Protocolli

2.1 Scambio delle chiavi

Una tecnica crittografica usata comunemente consiste nell'usare una diversa chiave per ogni conversazione. In tal caso la chiave è chiamata **session key**, perché è usata per una sola sessione. L'uso di una session key ha molti vantaggi, ma anche un grande problema, cioè far pervenire di volta in volta la chiave alle parti.

2.1.1 Scambio delle chiavi mediante crittografia simmetrica

Questo protocollo prevede che Alice e Bob, utilizzatori di una rete, condividano una chiave segreta (una per Alice e una per Bob) con il Key Distribution Center (KDC), che in questo caso è rappresentato da Trent, una entità di cui tutti si fidano. Tali chiavi devono essere già al loro posto prima dell'inizio del protocollo e, anche se questo è in realtà un grosso problema, ora non ce ne preoccupiamo.

1. Alice chiama Trent e richiede una session key K_{ab} per comunicare con Bob.
2. Trent genera una session key K_{ab} casuale. Ne cifra due copie: una con la chiave di Alice, l'altra con la chiave di Bob.

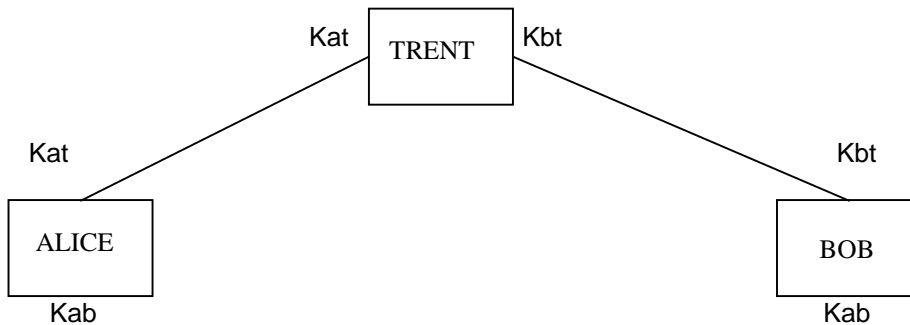
$$E_{K_{bt}}(K_{ab}) \text{ e } E_{K_{at}}(K_{ab})$$

3. Trent invia entrambe le chiavi ad Alice.
4. Alice decifra la propria copia della session key.
$$D_{K_{at}} [E_{K_{at}}(K_{ab})] = K_{ab}$$
5. Alice invia a Bob la sua (di lui) copia della session key.
$$E_{K_{tb}}(K_{ab})$$

6. Bob decifra la propria copia della session key.
$$D_{K_{bt}} [E_{K_{tb}}(K_{ab})] = K_{ab}$$

Alice e Bob usano questa session key per comunicare con sicurezza.

Schematicamente:



Questo protocollo si basa sull'assoluta affidabilità di Trent: se Mallory corrompe Trent, tutta la rete è compromessa.

Inoltre in questo sistema Trent rappresenta un potenziale collo di bottiglia, in quanto è coinvolto in ogni scambio di chiavi.

2.1.2 Scambio delle chiavi mediante crittografia a chiave pubblica (o a chiave asimmetrica)

Alice e Bob usano la crittografia a chiave pubblica per accordarsi su una session key, che poi usano per cifrare i dati. In pratica, le chiavi pubbliche firmate degli utenti spesso si trovano in un database, per cui Alice può inviare un messaggio sicuro a Bob, anche se lui non la conosce.

1. Alice ottiene la chiave pubblica di Bob dal KDC.
 $Get e_B$
2. Alice genera una session key casuale K_{ab} .
3. Alice cifra K_{ab} usando la chiave pubblica di Bob e la invia a Bob.
 $E_B(K_{ab}) \rightarrow Bob$
4. Bob decifra il messaggio di Alice usando la propria chiave privata.
 $D_B[E_B(K_{ab})] = K_{ab}$
5. Entrambi cifrano le loro comunicazioni usando la stessa session key.

Pregi:

- Non c'è bisogno della presenza di Trust

Difetti:

- Costoso computazionalmente

2.1.3 Man-in-the-middle attack

Eve può solo provare a forzare l'algoritmo a chiave pubblica o tentare un ciphertext-only attack sul messaggio cifrato. Mallory, invece, può fare molto di più: può modificare i messaggi, può

cancellarli, può crearne di nuovi; può imitare Bob quando parla con Alice, e viceversa. Vediamo come funziona l'attacco:

1. Alice invia a Bob la propria chiave pubblica. Mallory intercetta questa chiave e invia a Bob la propria chiave pubblica.
2. Bob invia ad Alice la propria chiave pubblica. Mallory intercetta questa chiave e invia ad Alice la propria chiave pubblica.
3. Quando Alice invia un messaggio a Bob, cifrato con la chiave pubblica "di Bob", Mallory lo intercetta. Siccome il messaggio è cifrato con la sua chiave pubblica, lo decifra con la sua chiave privata, lo cifra con la chiave pubblica di Bob e lo invia a Bob.
4. Quando Bob invia un messaggio ad Alice, cifrato con la chiave pubblica "di Alice", Mallory lo intercetta. Siccome il messaggio è cifrato con la sua chiave pubblica, lo decifra con la sua chiave privata, lo cifra con la chiave pubblica di Alice e lo invia ad Alice.

Questo attacco funziona anche quando le chiavi pubbliche di Alice e Bob sono memorizzate in un database. Mallory può intercettare la richiesta di Alice al database e sostituire la propria chiave pubblica a quella di Bob. Naturalmente può fare lo stesso anche con Bob. O, ancora meglio, può introdursi clandestinamente nel database e sostituire la propria chiave sia a quella di Alice che a quella di Bob.

Quest'attacco funziona perché Alice e Bob non hanno modo di verificare se stanno realmente parlando fra loro due.

2.1.4 Protocollo interlock

Il **protocollo interlock** (sincronizzazione) ha buone probabilità di sventare il **man-in-the-middle attack**. Ecco come funziona:

1. Alice invia a Bob la propria chiave pubblica.
 $e_A \rightarrow \text{Bob}$
2. Bob invia ad Alice la propria chiave pubblica.
 $e_B \rightarrow \text{Alice}$
3. Alice cifra il proprio messaggio usando la chiave pubblica di Bob, quindi invia metà del messaggio cifrato a Bob.
 $[E_B (Mab)]/2 \rightarrow \text{Bob}$
4. Bob cifra il proprio messaggio usando la chiave pubblica di Alice, quindi invia metà del messaggio cifrato ad Alice.
 $[E_A (Mab)]/2 \rightarrow \text{Alice}$
5. Alice invia l'altra metà del messaggio cifrato a Bob.
 $[E_B (Mab)]/2 \rightarrow \text{Bob}$
6. Bob riunisce le due metà del messaggio di Alice e lo decifra con la propria chiave privata.
 $D_B [E_B (Mab)]$
7. Bob invia l'altra metà del proprio messaggio ad Alice.
 $[E_A (Mab)]/2 \rightarrow \text{Alice}$
8. Alice riunisce le due metà del messaggio di Bob e lo decifra con la propria chiave privata.
 $D_A [E_A (Mab)]$

L'importante è che mezzo messaggio da solo è inutile. Bob non può leggere niente del messaggio di Alice fino al passo 6; Alice non può leggere niente del messaggio di Bob fino al punto 8. Ci sono vari sistemi per fare ciò:

- Se l'algoritmo di cifratura è un algoritmo a blocchi, la metà di ogni blocco può essere spedita in ognuna delle due metà del messaggio.
- La decifratura del messaggio potrebbe dipendere da un vettore di inizializzazione, da inviare con la seconda metà del messaggio.
- La prima metà del messaggio potrebbe essere una one-way hash function del messaggio cifrato e il messaggio cifrato vero e proprio potrebbe essere la seconda metà.

Per capire come ciò provochi dei problemi a Mallory, esaminiamo il suo tentativo di sovvertire il protocollo. Egli può ancora sostituire la propria chiave pubblica a quelle di Alice e Bob, ma ora, quando intercetta la prima metà del messaggio di Alice, non può decifrarlo e ricifrarlo, quindi non gli resta che inventare un messaggio completamente nuovo e inviarne metà a Bob. Quando intercetta la prima metà del messaggio di Bob, ha lo stesso problema. Quando poi intercetta le seconde metà, non può più modificare i messaggi che ha inventato in precedenza. La conversazione tra Alice e Bob sarà quindi completamente diversa.

Mallory potrebbe anche andare avanti in questo modo. Se conosce abbastanza bene Alice e Bob da imitarli entrambi in una conversazione tra loro, i due potrebbero non accorgersi di venire ingannati. Ma naturalmente questo è molto più difficile di quello che poteva fare prima.

2.1.5 Scambio delle chiavi con firme digitali

L'uso delle firme digitali in un protocollo per lo scambio di session key, elude il man-in-the-middle attack. Trent firma sia la chiave pubblica di Alice che quella di Bob. Le chiavi firmate includono un certificato di proprietà firmato. Quando Alice e Bob ricevono le chiavi, verificano entrambi la firma di Trent. Ora sanno a chi appartiene la chiave pubblica e il protocollo di scambio delle chiavi può procedere.

Mallory ha dei seri problemi. Le uniche cose che può fare sono ascoltare il traffico cifrato o distruggere le linee di comunicazione.

Questo protocollo usa Trent, ma il rischio di compromettere il KDC è minore che nel primo caso. Se Mallory corrompe Trent, tutto ciò che ottiene è la chiave privata di Trent. Questa chiave gli permette di firmare nuove chiavi; questo però non gli permette direttamente di decifrare delle session key o di leggere dei messaggi cifrati. Per leggere le comunicazioni, Mallory deve impersonare un utente della rete ed effettuare un man-in-the-middle attack.

Quest'attacco funziona, ma va ricordato che Mallory deve essere in grado di intercettare e modificare i messaggi. In alcune reti ciò è molto difficile: su un canale broadcast è quasi impossibile sostituire un messaggio (sebbene sia relativamente facile disturbare le comunicazioni per renderle incomprensibili). Sulle reti di computer, invece, questi tipi di attacco diventano ogni giorno più facili (IP spoofing, attacchi ai router, e così via).

2.2 Servizi di timestamping (o di certificazione del tempo)

In molte situazioni c'è la necessità di provare che un documento esisteva già in una certa data. Si pensi, ad esempio, ad una controversia su un brevetto: la parte che esibisce la copia meno recente del lavoro, vince la causa. Con i documenti cartacei, i notai possono firmare e gli avvocati tutelare le copie. Se sorge una disputa, il notaio o l'avvocato testimoniano che il documento esisteva già in una certa data.

Nel mondo digitale, le cose sono molto più complicate. Non è possibile individuare segni di manomissione in un documento digitale, in quanto può essere copiato e modificato a piacere senza che nessuno se n'accorga. E' banale modificare la data in un documento per computer. Quello che si vuole ottenere è un protocollo di timestamping (letteralmente "timbro datario") digitale con le seguenti proprietà:

- I dati stessi devono essere timbrati, indipendentemente dal mezzo fisico su cui risiedono.
- Deve essere impossibile cambiare anche un solo bit del documento, senza che la modifica sia evidente.
- Deve essere impossibile eseguire il timestamping di un documento con una data e un'ora diverse dalle attuali.

2.2.1 Soluzione con arbitro

Questo protocollo fa uso di Trent, che è un fidato servizio di timestamping, e di Alice, che desidera eseguire il timestamping su di un documento.

1. Alice trasmette una copia del documento a Trent.
2. Trent memorizza la data e l'ora in cui ha ricevuto il documento e tiene in custodia una copia del documento.

Ora, se qualcuno mette in dubbio il momento di creazione del documento, Alice deve semplicemente chiamare in causa Trent. Egli mostrerà la propria copia del documento e verificherà data e ora.

Questo protocollo funziona, ma ha degli ovvi problemi. Per prima cosa non c'è privacy. Alice deve fornire una copia del documento a Trent. Chiunque in ascolto sul canale di comunicazione può leggerlo. Alice può cifrarlo, ma in ogni caso una copia del documento deve restare nel database (non si sa quanto affidabile) di Trent.

Il secondo problema, è che il database deve essere enorme; inoltre, la larghezza di banda necessaria a spedire file di grandi dimensioni a Trent può non essere disponibile.

Il terzo problema riguarda potenziali errori. Un errore di trasmissione o la rottura del computer di Trent, possono annullare completamente la validità del timestamp sul documento di Alice. Infine, può non esistere una persona abbastanza onesta da gestire il servizio di timestamping.

2.2.1.2 Soluzione con arbitro migliorata

Le one-way hash function e le firme digitali possono risolvere facilmente la maggior parte dei problemi:

1. Alice calcola il valore di one-way hash del documento e trasmette il valore di hash a Trent.
 $H(M) \rightarrow \text{Trent}$
2. Trent appone al valore di hash la data e l'ora in cui lo ha ricevuto e quindi firma digitalmente il risultato e invia il valore di hash (con timestamp) firmato ad Alice.
 $D_{Ts} [H(M) + T] \rightarrow \text{Alice}$

Volendo verificare la data conoscendo il messaggio M , servirà la chiave pubblica di Trent, E_{Ts} :
 $E_{Ts} \{ D_{Ts} [H(M) + T] \} = H(M) + T$

In tale modo si verifica che T coincida. Per verificare il messaggio M basterà farne la hash e confrontare il risultato con la hash del messaggio depositato.

La data T si può modificare solo conoscendo la chiave privata di Trent quindi Alice e Trent possono sempre agire in collusione per produrre timestamp falsi e comunque Trent potrebbe sempre venire corrotto da una terza parte..

2.2.2 Protocollo di collegamento (o di linking)

Un modo per risolvere questo problema consiste nel collegare il timestamp di Alice con quelli precedentemente generati da Trent. Tali timestamp saranno molto probabilmente generati per persone diverse da Alice. Siccome l'ordine in cui Trent riceve le diverse richieste di timestamp non è noto a priori, il timestamp di Alice deve essere avvenuto dopo quello precedente. E siccome la richiesta che è arrivata dopo è collegata con il timestamp di Alice, allora la richiesta di Alice deve essere avvenuta prima. Ciò incastra la richiesta di Alice in un preciso slot temporale.

Se è il nome di Alice, il valore di hash di cui Alice vuole il timestamp è il timestamp precedente è T_{n-1} , allora il protocollo è:

1. Alice invia a Trent H_n e A .
2. Trent invia ad Alice:

$$T_n = S_K(n, A, H_n, t_n, I_{n-1}, H_{n-1}, T_{n-1}, L_n)$$

dove L_n è composto dalle seguenti informazioni di collegamento:

$$L_n = H(I_{n-1}, H_{n-1}, T_{n-1}, L_{n-1})$$

S_K indica che il messaggio è firmato con la chiave privata di Trent. Il nome di Alice la identifica come l'origine della richiesta. Il parametro n indica l'ordine della richiesta: questo è il timestamp n -esimo emesso da Trent. Il parametro t_n è il riferimento temporale. L'informazione addizionale è costituita dall'identificativo, dal valore di hash originale, dal

riferimento temporale e dal timestamp con hash del precedente documento certificato da Trent.

3. Dopo avere timbrato il documento successivo, Trent invia ad Alice l'identificativo di chi lo ha richiesto: I_{n+1} .

Se qualcuno mette in dubbio il timestamp di Alice, lei deve solo contattare gli autori del documento precedente e di quello successivo: I_{n-1} and I_{n+1} . Se i loro documenti sono messi in discussione, possono mettersi in contatto con I_{n-2} and I_{n+2} , e così via. Tutti possono dimostrare che il loro documento è stato timbrato dopo il precedente e prima del successivo.

Questo protocollo rende molto difficili eventuali collusioni tra Alice e Trent. L'unico modo per violare questo sistema consiste nel creare una catena fittizia di documenti, sia prima che dopo quello di Alice, abbastanza lunga da esaurire la pazienza di chiunque stia mettendo in dubbio il timestamp.

2.2.3 Protocollo distribuito

Le persone muoiono; i timestamp vanno persi. Possono succedere tante cose che rendono impossibile per Alice ottenere una copia del timestamp di I_{n-1} . Questo problema può essere alleviato inserendo nel timestamp di Alice i timestamp delle 10 persone precedenti e, poi, inviandole gli identificativi delle 10 persone successive. Alice ha così una probabilità molto maggiore di trovare qualcuno che abbia ancora il proprio timestamp.

Il seguente è un protocollo del tipo appena visto, che, però, fa a meno di Trent:

1. Usando H_n come ingresso, Alice genera una sequenza di valori casuali, per mezzo di un generatore di numeri pseudo-casuali crittograficamente sicuro:

$$V_1, V_2, V_3, \dots, V_k$$

2. Alice interpreta questi valori come gli identificativi di altre persone, quindi invia H_n ad ognuna di queste persone.
3. Ognuna di queste persone, allega la data e l'ora al valore di hash, firma il tutto e lo rispedisce ad Alice.
4. Alice memorizza tutte le firme come timestamp.

Il generatore di numeri pseudo-casuali crittograficamente sicuro del passo 1 impedisce ad Alice di scegliere dei controllori corrotti. Anche se prova a fare dei banali cambiamenti nel documento per ottenere un gruppo di identificativi corrotti, le probabilità di riuscita sono trascurabili.

Questo protocollo funziona perché, per Alice, l'unico modo di falsificare un timestamp sarebbe convincere tutte le k persone a collaborare. Siccome queste sono scelte casualmente, le probabilità sono tutte contro Alice.

2.3 Firme digitali

Requisiti:

- Autentica
- Non copiabile
- Non riusabile
- Non ripudiabile
- Il documento firmato deve essere inalterabile

2.3.1 Firma di documenti mediante crittografia a chiave privata con arbitro

Un primo elementare protocollo di firma digitale fa uso della crittografia simmetrica e prevede la presenza di un arbitro. Non approfondiamo quest'argomento perché, anche se il protocollo funziona, in pratica non è realizzabile. Innanzi tutto Trent (l'arbitro) rappresenta sempre un collo di bottiglia, perché deve partecipare ad ogni comunicazione; inoltre, deve essere infallibile e tutti devono fidarsi ciecamente di lui, il che è in pratica irrealizzabile.

2.3.2 Firma di documenti mediante crittografia a chiave pubblica

Ci sono algoritmi a chiave pubblica che possono essere utilizzati per le firme digitali. In alcuni di questi (ad esempio RSA) sia la chiave pubblica che la privata possono essere usate per la cifratura. Se si cifra un documento con la propria chiave privata si ottiene una firma digitale sicura. In altri casi (ad esempio DSA) c'è un algoritmo per la firma digitale distinto da quello per la cifratura.

Il protocollo di base è semplice:

1. Alice cifra il documento con la propria chiave privata, firmando così il documento.
 $D_A (M)$
2. Alice invia il documento firmato a Bob.
 $D_A (M) \rightarrow \text{Bob}$
3. Bob decifra il documento con la chiave pubblica di Alice, verificando così la firma.
 $E_A [D_A (M)]$

Usando lo schema precedente, in alcune circostanze, Bob può truffare Alice, riusando il documento firmato. Immaginiamo che Alice invii a Bob un assegno digitale firmato; Bob lo porta alla banca, la quale verifica la firma e gli paga la somma. Bob, che è una persona senza scrupoli, conserva una copia dell'assegno digitale e dopo una settimana torna in banca (anche in una banca diversa). La banca verifica la firma e paga nuovamente l'assegno; se Alice non si accorge di quello che sta succedendo, Bob può continuare la truffa per anni.

Per questo motivo, le firme digitali spesso contengono un **timestamp**, cioè una registrazione della data e dell'ora in cui il documento è stato firmato. In pratica, il timestamp è unito al messaggio originale, che solo a quel punto viene firmato. La banca memorizza i timestamp in un database. Ora, quando Bob prova ad incassare una seconda volta l'assegno, la banca controlla il proprio database e si accorge di aver già pagato un assegno di Alice con quel timestamp.

Quindi per non essere copiabile e riusabile Alice invierà a Bob il messaggio cifrato con la propria chiave privata, ma contenente anche un timestamp:

$$D_A (M + Ts) \rightarrow \text{Bob}$$

Questo protocollo verifica le proprietà richieste:

1. La firma è autentica; quando Bob verifica il messaggio con la chiave pubblica di Alice, sa che è stata lei a firmare.
2. La firma non è falsificabile; solo Alice conosce la propria chiave privata.
3. La firma non è riutilizzabile; la firma è funzione del documento e non può essere trasferita ad un altro documento.
4. Il documento firmato è inalterabile; se avviene qualche modifica, il documento non può più essere verificato con la chiave pubblica di Alice.
5. La firma non può essere ripudiata; Bob non ha bisogno di Alice per verificare la firma.

Difetti:

- Documento in chiaro
- Cifrare un messaggio di grandi dimensioni richiede molto tempo \rightarrow pesante computazionalmente

2.3.3 Firma di documenti mediante crittografia a chiave pubblica e documento in cifra

Con il seguente protocollo si elimina il difetto del precedente relativo al documento in chiaro:

1. Alice cifra il documento più il timestamp con la propria chiave privata, firmando così il documento.

$$D_A (M + Ts)$$

2. Alice codifica il risultato del passo precedente usando la chiave pubblica di Bob

$$E_B [D_A (M + Ts)]$$

3. Alice invia il documento firmato a Bob.

$$E_B [D_A (M + Ts)] \rightarrow \text{Bob}$$

4. Bob decifra il documento con la chiave pubblica di Alice, verificando così la firma.

$$D_B \{E_A [D_A (M + Ts)]\} = M + Ts$$

In tale modo si elimina un difetto ma si aggrava l'altro, infatti con tale protocollo aumenta il costo computazionale.

2.3.4 Firma di documenti mediante crittografia a chiave pubblica e one-way hash function

Nelle implementazioni pratiche, gli algoritmi a chiave pubblica sono troppo inefficienti per firmare documenti lunghi. Per risparmiare tempo, i protocolli di firma digitale sono spesso implementati usando delle one-way hash function: invece di firmare il documento, Alice firma il valore di hash del documento. In questo protocollo, sia la one-way hash function che l'algoritmo di firma digitale devono essere concordati anticipatamente.

1. Alice genera il valore di hash di un documento con il relativo timestamp.
 $H(M + Ts) = h$
2. Alice cifra il valore di hash con la propria chiave privata, firmando così il documento.
 $D_A(h)$
3. Alice invia a Bob il documento e il valore di hash firmato.
 $D_A(h) \rightarrow \text{Bob}$
4. Bob genera il valore di hash del documento che Alice gli ha inviato.
 $H(M + Ts) = h'$
5. Bob, usando l'algoritmo di firma digitale, decifra il valore di hash con la chiave pubblica di Alice.
 $E_A[D_A(h)] = h$
6. Se il valore di hash firmato, cioè h , coincide con quello generato, cioè h' , la firma è valida.
Se $h = h'$ ok

La velocità aumenta in modo drastico e, usando un hash di 160 bit, le probabilità di avere lo stesso hash per due documenti diversi sono solo una su 2^{160} ; per questo, chiunque, può considerare la firma del valore di hash come la firma del documento, senza correre rischi. Bisogna notare che, se si usasse una hash function non one-way, sarebbe facile creare più documenti con lo stesso valore di hash; in tal modo firmando un documento, rischieremmo di vederci attribuire una moltitudine di documenti diversi.

Questo protocollo ha altri benefici. Innanzi tutto, la firma può essere tenuta separata dal documento. Inoltre, lo spazio richiesto al destinatario per conservare il documento e la firma, è molto minore. Un sistema di archivio può utilizzare questo protocollo per verificare l'esistenza di un documento senza doverne conservare il contenuto. Il database centrale potrebbe conservare solamente i valori di hash dei file. Gli utenti presentano i valori di hash al database; questo applica un timestamp alle richieste e le memorizza. Se in futuro ci fosse disaccordo su chi ha creato un documento e su quando lo ha fatto, il database potrebbe risolvere la disputa. Questo sistema ha vaste implicazioni concernenti la privacy: Alice potrebbe registrare i diritti d'autore di un documento, mantenendo segreto il documento stesso. Alice dovrebbe rendere pubblico il documento solo per provarne la paternità.

2.3.5 Resend attack

Consideriamo un'implementazione di questo protocollo in cui si utilizzano anche dei messaggi di conferma. Quando Bob riceve un messaggio, lo rispedisce al mittente come conferma dell'avvenuta ricezione.

1. Alice firma un messaggio con la propria chiave privata, lo cifra con la chiave pubblica di Bob e lo invia a Bob.

$$E_B(S_A(M))$$

2. Bob decifra il messaggio con la propria chiave privata e verifica la firma con la chiave pubblica di Alice.

$$V_A(D_B(E_B(S_A(M)))) = M$$

3. Bob firma il messaggio con la propria chiave privata, lo cifra con la chiave pubblica di Alice e lo rimanda ad Alice.

$$E_A(S_B(M))$$
4. Alice decifra il messaggio con la propria chiave privata e verifica la firma con la chiave pubblica di Bob. Se il messaggio risultante è lo stesso che ha inviato a Bob, sa che Bob ha ricevuto il messaggio.

Se per la cifratura e per la firma digitale viene usato lo stesso algoritmo, esiste un possibile attacco, detto **resend attack**. In questi casi, l'operazione di firma è l'inversa di quella di cifratura: $V_X = E_X$ e $S_X = D_X$.

Supponiamo che Mallory sia un legittimo utente del sistema, con le proprie chiavi (pubblica e privata). Vediamo come Mallory può leggere la posta di Bob.

1. Mallory registra il messaggio di Alice a Bob.

$$E_B[D_A(M)] = Z$$
2. Mallory invia tale messaggio a Bob, sostenendo che arriva da lui (Mallory).

$$Z \rightarrow \text{Bob}$$
3. Bob pensa che sia un regolare messaggio da parte di Mallory, così lo decifra con la propria chiave privata e poi prova a verificare la firma con la chiave pubblica di Mallory.

$$E_M[D_B(Z)] = E_M[D_B[E_B[D_A(M)]]] = E_M[D_A(M)]$$
4. Bob continua il protocollo e invia a Mallory una ricevuta:

$$E_M[D_B[E_M[D_A(M)]]] \rightarrow \text{Mallory}$$
5. Ora a Mallory non resta che decifrare il messaggio con la propria chiave privata, cifrarlo con la chiave pubblica di Bob, decifrarla di nuovo con la propria chiave privata e cifrarlo con la chiave pubblica di Alice. A questo punto Mallory ha M .

$$E_A[D_M[E_B[D_M[E_M[D_B[E_M[D_A(M)]]]]]]] = M$$

E' ragionevole immaginare che Bob invii automaticamente a Mallory una ricevuta. Questo protocollo potrebbe essere integrato nel suo software di comunicazione e prevedere l'invio automatico della ricevuta. Se Bob controllasse la comprensibilità del messaggio prima di inviare una ricevuta, potrebbe evitare questo problema di sicurezza.

2.3.6 Difendersi dal resend attack

Questo attacco funziona perché l'operazione di cifratura è uguale a quella di verifica della firma, e l'operazione di decifratura è uguale a quella di apposizione della firma. Per avere un

protocollo sicuro è sufficiente usare operazioni anche solo lievemente differenti per la cifratura e le firme digitali. Questo si può ottenere in vari modi: usando chiavi diverse per ogni operazione; usando differenti algoritmi per ogni operazione; usando i timestamp, che rendono il messaggio in arrivo diverso da quello in partenza; usando le firme digitali con le one-way hash function.

2.3.7 Firme multiple

In alcuni casi, Alice e Bob potrebbero avere la necessità di firmare lo stesso documento. Senza le one-way hash function, le firme multiple sono possibili, ma con qualche problema. Usando le one-way hash function, invece, il procedimento è semplicissimo:

1. Alice firma il valore di hash del documento.
2. Bob firma il valore di hash del documento.
3. Bob invia la propria firma ad Alice.
4. Alice invia il documento, la propria firma e quella di Bob a Carol.
5. Carol verifica entrambe le firme.

I passi 1 e 2 possono essere eseguiti in serie o in parallelo. Nel passo 5, Carol può verificare una firma senza dover verificare l'altra.

2.3.8 Firme digitali e possibilità di ripudiare

Alice può imbrogliare con le firme digitali e purtroppo non c'è niente da fare (o quasi). Può firmare un documento e poi sostenere di non averlo fatto. Per prima cosa firma normalmente il documento; quindi diffonde in modo anonimo la propria chiave privata. A questo punto, dichiara che la propria firma è stata compromessa e che anche altre persone la stanno usando, fingendo d'essere lei. Alice nega, così, di aver firmato il documento di cui sopra, e tutti gli altri documenti firmati con la propria chiave privata. Questa è chiamata **repudiation**.

I timestamp possono ridurre gli effetti di questo tipo d'imbroglio, ma Alice può sempre sostenere che la propria chiave era già compromessa. E' per questo che si fa un gran parlare di chiavi private racchiuse in moduli anti-manomissione.

Benché non si possa fare nulla per questo tipo d'abuso, ci si può cautelare dall'invalidazione di vecchie firme. Il protocollo è il seguente:

1. Alice firma il messaggio.
2. Alice genera un'intestazione contenente alcune informazioni d'identificazione, la concatena al messaggio firmato, firma il tutto e lo invia a Trent.
3. Trent verifica la firma esterna e conferma le informazioni d'identificazione. Quindi aggiunge un timestamp al messaggio firmato di Alice e alle informazioni d'identificazione. Alla fine firma il tutto e lo invia sia a Bob che ad Alice.
4. Bob verifica la firma di Trent, le informazioni d'identificazione e la firma di Alice.
5. Alice verifica il messaggio che Trent ha inviato a Bob. Se non è stata lei a crearlo, lo dice al più presto.

2.3.9 Undeniable signature

In certe situazioni è necessario avere delle firme che possano essere dimostrate valide, ma che il destinatario non possa mostrare a terze parti senza il consenso dell'autore. Una firma di questo tipo viene detta **undeniable signature**, in quanto non può essere negata dall'autore. Sebbene forse sarebbe stato più corretto chiamare queste firme "non trasferibili", il nome è giustificato dal fatto che, se Alice è obbligata a riconoscere o a ricusare una firma (ad esempio in tribunale), non può falsamente rinnegare la propria firma.

L'aspetto matematico è complesso, ma l'idea di base è semplice:

1. Alice si presenta a Bob con una firma.
2. Bob genera un numero casuale e lo invia ad Alice.
3. Alice effettua un calcolo, usando il numero casuale e la propria chiave privata, quindi invia a Bob il risultato. Alice può effettuare questo calcolo solo se la firma è valida.
4. Bob conferma.

Esiste anche un ulteriore protocollo, tale che Alice può dimostrare di non aver firmato un certo documento, ma non può rinnegare falsamente una firma. Questa soluzione, comunque, non è perfetta, perché è stato dimostrato che, in alcuni casi, Bob può convincere Carol (una terza parte) della validità della firma di Alice. Ciononostante, le firme non rinnegabili hanno un grande numero di applicazioni.

Una nozione simile è quella di **entrusted undeniable signature**: in questo caso, l'unica differenza è che il protocollo di disconoscimento può essere portato a termine solo da Trent. Bob non può pretendere che Alice esegua tale protocollo, ma può farlo Trent. E, se Trent rappresenta il sistema giudiziario, allora farà eseguire il protocollo solo per risolvere una disputa formale.

2.3.10 Firme di gruppo

Viene definita firma di gruppo, una firma che ha le seguenti (spesso utili) proprietà:

- Solo i membri del gruppo possono firmare messaggi.
- Chi riceve la firma può verificare che appartiene veramente a quel gruppo.
- Chi riceve la firma non può determinare qual membro del gruppo sia l'autore.
- In caso di disputa è possibile conoscere l'identità dell'autore.

2.3.10.1 Firme di gruppo con arbitro

Questo protocollo fa uso di un arbitro:

1. Trent genera una grande quantità di coppie chiave pubblica / chiave privata e dà ad ogni membro del gruppo una diversa lista di chiavi private uniche. Non ci sono doppioni.
2. Trent rende pubblica la lista di tutte le chiavi pubbliche del gruppo, in ordine casuale. Trent conserva un elenco segreto dei proprietari delle diverse chiavi.

3. Quando un membro del gruppo vuole firmare un documento, sceglie una chiave a caso dalla propria lista personale.
4. Quando qualcuno vuole verificare che la firma appartiene al gruppo, controlla la lista completa delle chiavi pubbliche e verifica la firma.
5. In caso di disputa, Trent conosce il proprietario di ogni chiave pubblica.

Il problema di questo protocollo è che richiede una parte fidata, infatti, Trent conosce tutte le chiavi private e può falsificare le firme.

2.3.11 Firme digitali fail-stop

Supponiamo che in qualche modo Eve, potentissimo avversario dotato di una potenza di calcolo enorme, riesca a forzare la chiave privata di Alice: a questo punto, Eve può impersonare Alice, falsificandone la firma senza problemi. Le firme digitali **fail-stop** riescono a prevenire questo tipo di truffa. L'idea di base è che per ogni chiave pubblica ci sono molte possibili chiavi private valide: queste chiavi, però, producono tutte firme diverse. Supponiamo che Eve voglia forzare la chiave privata di Alice: Eve raccoglie messaggi firmati e, usando una schiera di supercomputer, prova a recuperare la chiave privata di Alice. Anche se Eve riesce a ottenere una chiave privata valida, questa è solo una delle tante possibili e molto probabilmente non è quella giusta. Le probabilità di trovare proprio la chiave privata di Alice possono essere rese trascurabili.

Ora, quando Eve falsifica la firma di Alice con la chiave privata che ha generato, ottiene una firma diversa da quella che otterrebbe Alice. Quindi Alice può provare che la firma falsificata non è sua, generando, con lo stesso messaggio e la stessa chiave pubblica, una firma diversa.

Naturalmente questo protocollo non impedisce a Mallory di entrare in casa di Alice e rubarle la chiave privata.

2.4 Key Distribution Center

In tutti questi protocolli di crittografia a chiave pubblica, non abbiamo chiarito come Alice può ottenere la chiave pubblica di Bob. La maniera più semplice di ottenere la chiave pubblica di qualcuno è di richiederla ad un database sicuro, situato da qualche parte nella rete. Il database deve essere pubblico: in questo modo chiunque può ottenere la chiave pubblica di chiunque altro. Il database deve anche essere protetto in scrittura da tutti eccetto Trent, altrimenti Mallory potrebbe sostituire la chiave pubblica di Bob. A quel punto Bob non potrebbe più leggere i messaggi indirizzati a lui, mentre potrebbe farlo Mallory.

Anche se le chiavi pubbliche fossero memorizzate in un database sicuro, Mallory potrebbe sostituirle durante la trasmissione. Per prevenirlo, Trent può firmare ogni chiave pubblica con la propria chiave privata; in questo caso, Trent è noto come **Key Certification Authority** o **Key Distribution Center (KDC)**. Nelle implementazioni pratiche, il KDC firma un messaggio composto dal nome dell'utente, la sua chiave pubblica ed altre eventuali informazioni utili. Questo messaggio composto è memorizzato nel database del KDC. Quando Alice riceve la chiave pubblica di Bob verifica la firma del KDC per assicurarsi della validità della chiave.

Quanto visto, in pratica, non rende le cose impossibili per Mallory, solo molto più difficili e rischiose.

2.5 Autenticazione

Quando Alice utilizza un computer host, questo in qualche modo deve riconoscerla. Tradizionalmente il problema è risolto utilizzando delle password: Alice inserisce la propria password ed il computer host conferma che è corretta.

2.5.1 Autenticazione mediante one-way function

In realtà non è necessario che il computer host conosca le password: è sufficiente che sappia distinguere quelle valide da quelle non valide. Ciò è facile se si utilizzano le one-way function. Invece di memorizzare le password, il computer host ne memorizza le rispettive one-way function.

1. Alice invia al computer host la propria password.
2. Il computer host calcola la one-way function per la password.
3. Il computer host confronta il risultato della one-way function con il valore memorizzato.

Siccome il computer host non deve più conservare l'elenco delle password valide, diminuisce il rischio che qualcuno s'intrufoli nel sistema e le rubi. La lista di password cifrate dalla one-way function è inutile, perché la funzione non può essere invertita per recuperare le password.

2.5.2 Dictionary attack e salt

Un file di password cifrate con una one-way function è ancora vulnerabile. Mallory compila una lista con le password più comuni, con 1.000.000 di elementi. A questo punto usa la one-way function su tutti gli elementi della lista e memorizza i risultati. In seguito, Mallory ruba un file di password cifrate e lo confronta con il suo file, trovando le eventuali corrispondenze.

Questo è un **dictionary attack**, ed è sorprendentemente efficace. Il **salt** (letteralmente sale, ma forse rende più l'idea la parola condimento) è un accorgimento per rendere il dictionary attack più difficoltoso. Il salt è una stringa casuale che viene concatenata con le password, prima che queste vengano passate alla one-way function. Quindi, sia il valore del salt che i risultati della one-way function sono conservati nel database del computer host. Se il numero di possibili valori del salt è abbastanza grande, in pratica viene eliminata la possibilità di un dictionary attack contro le parole più comuni, perché Mallory dovrebbe generare il one-way hash per ogni possibile valore di hash.

Il vero problema è assicurarsi che Mallory debba effettuare un tentativo di cifratura per ogni password del suo dizionario ogni volta che prova a violare la password di un'altra persona, piuttosto che poter fare un unico massiccio calcolo (fuori linea) per tutte le possibili password.

Naturalmente la situazione migliora al crescere della dimensione del salt. La maggior parte dei sistemi UNIX, ad esempio, usa solo 12 bit di salt. A riguardo, Daniel Klein ha sviluppato un

programma per indovinare le password che spesso, entro una settimana, riesce a violare il 40% delle password di un dato host. Feldmeier e Karn, invece, hanno compilato una lista di circa 732.000 password comuni concatenate con tutti i 4096 possibili valori del salt, e stimano di poter violare circa il 30% di password su qualunque host.

Il salt non è miracoloso; aumentandone il numero di bit non si risolve tutto. Il salt protegge solo da dictionary attack generici contro un file di password, non da un attacco contro una singola password. Protegge le persone che usano la stessa password su macchine diverse, ma non migliora le password scelte male.

2.5.3 Autenticazione mediante crittografia a chiave pubblica

Persino con il salt, il primo protocollo ha dei seri problemi di sicurezza. Quando Alice invia la propria password al computer host, chiunque abbia accesso al suo canale di comunicazione, può leggerla. Eve potrebbe essere da qualche parte lungo il percorso, pronta ad ascoltare la sequenza di login. Se Eve ha accesso alla memoria del processore del computer host, può vedere la password prima che sia passata alla one-way function.

La crittografia a chiave pubblica può risolvere questo problema. Il computer host conserva un file con la chiave pubblica di ogni utente; ogni utente custodisce la propria chiave privata. Ecco un semplice esempio di protocollo.

Quando si effettua il login il protocollo procede così:

1. Il computer host invia ad Alice una stringa casuale.
2. Alice cifra la stringa con la propria chiave privata e la rimanda al computer host, insieme al proprio nome.
3. Il computer host cerca la chiave pubblica di Alice nel proprio database e la usa per decifrare il messaggio.
4. Se la stringa decifrata è uguale a quella che il computer host ha inviato ad Alice, il computer host permette ad Alice di accedere al sistema.

La cosa più importante è che Alice non invia mai la sua chiave privata sulla linea di trasmissione; il risultato è che Eve, in questo protocollo, non può fare nulla. La chiave privata è lunga e difficile da ricordare e, di solito, è usata direttamente dal software (o hardware) di comunicazione dell'utente. Ciò richiede un terminale intelligente di cui Alice si possa fidare, ma, perlomeno, il canale e il computer host non devono necessariamente essere sicuri.

Resta il fatto che, cifrare stringhe arbitrarie, soprattutto quando sono inviate da una terza parte di cui non ci si fida, è una cosa sciocca e può esporre ad alcuni tipi d'attacco. I protocolli di autenticazione sicuri hanno la seguente, più complessa, forma:

1. Alice esegue un calcolo basato su alcuni numeri casuali e sulla propria chiave privata ed invia il risultato al computer host.
2. Il computer host invia ad Alice un diverso numero casuale.
3. Alice esegue alcuni calcoli basati sui numeri casuali (sia quelli che ha generato che quello che ha ricevuto) e sulla propria chiave privata ed invia il risultato al computer host.

4. Il computer host esegue alcuni calcoli sui vari numeri ricevuti da Alice e sulla chiave pubblica di Alice per verificare se lei (Alice) conosce la sua (di Alice) chiave privata.
5. Se è così, la sua identità è verificata.

Se Alice non si fida del computer host più di quanto questo si fidi di lei, allora Alice pretenderà che il computer host provi la propria identità alla stessa maniera.

2.6 Bit commitment

La situazione è questa: Alice vuole fare una previsione (ad esempio un bit o una serie di bit), ma non la vuole rivelare fino a quando non si è avverata; Bob, d'altra parte, vuole assicurarsi che Alice non possa cambiare idea dopo essersi impegnata.

2.6.1 Bit commitment mediante crittografia simmetrica

Questo protocollo di bit commitment usa la crittografia simmetrica:

1. Bob genera una sequenza di bit casuali, R , e la invia ad Alice.
 $R \rightarrow \text{Alice}$
2. Alice crea un messaggio costituito da i bit che vuole impegnare, b , e dalla sequenza casuale di Bob. Lo cifra con una chiave casuale, K , e invia il risultato a Bob.
 $E_K(R + b) \rightarrow \text{Bob}$

Questa è la parte del protocollo in cui Alice si impegna. Bob non può decifrare il messaggio e così non può conoscere i bit.

Quando per Alice arriva il momento di rivelare i bit, il protocollo continua:

3. Alice invia a Bob la chiave.
 $K \rightarrow \text{Bob}$
4. Bob decifra il messaggio per svelare i bit e controlla la sua sequenza casuale per verificare la validità dei bit.
 $D_K[E_K(R + b)] = R + b$

Se il messaggio non contenesse la sequenza casuale di Bob, Alice potrebbe decifrare il messaggio inviato a Bob con diverse chiavi, fino a trovare quella che le dà il bit desiderato. Siccome un bit può assumere due soli valori, Alice troverà una chiave adatta dopo pochi tentativi. La sequenza di bit casuali previene questo attacco, perché Alice è costretta a produrre un messaggio con il suo bit invertito, ma con tutti quelli di Bob immutati. Se l'algoritmo di cifratura è buono, le sue possibilità di riuscita sono trascurabili.

2.6.2 Bit commitment mediante generatori di sequenze pseudo-casuali

Questo è un protocollo di bit commitment che usa i generatori di sequenze pseudo-casuali:

1. Bob genera una sequenza di bit casuali, R_B , e la invia ad Alice.
2. Alice genera un seme casuale per un generatore di sequenze pseudo-casuali; quindi, per ogni bit della sequenza di Bob, invia a Bob l'output del generatore, se il bit di Bob è 0, oppure lo XOR dell'output del generatore e del proprio bit, se il bit di Bob è 1.

Quando per Alice arriva il momento di rivelare i bit, il protocollo continua:

3. Alice invia a Bob il proprio seme.
4. Bob completa il passo 2 per confermare che Alice ha agito correttamente.

2.6.3 Bit commitment mediante hashing

1. Alice genera due stringhe di bit random, R_1 e R_2 .
2. Alice crea un messaggio costituito dalle due stringhe e dai bit che vuole impegnare, b ; di tale messaggio ne fa la hash.
$$H(R_1 + R_2 + b) = y$$
3. Alice invia a Bob sia y che la stringa R_1 .
$$\{y, R_1\} \rightarrow \text{Bob}$$

Questa è la parte del protocollo in cui Alice si impegna.

Di seguito c'è la fase di verifica:

4. Alice invia a Bob il messaggio costituito dalle due stringhe e dai bit che vuole impegnare (a Bob mancava solo R_2).
$$(R_1 + R_2 + b) \rightarrow \text{Bob}$$
5. Bob ne fa la hash per verificare se il risultato ottenuto, y' , coincide con quello ricevuto al passo 2, y .
$$H(R_1 + R_2 + b) = y'$$

2.7 Prove a conoscenza nulla

Purtroppo, di solito, l'unico modo di dimostrare di conoscere una certa cosa, consiste nel dirla; a quel punto, però, anche gli altri la sanno. Usando le one-way function, invece, Peggy può fornire una **zero-knowledge proof**. Questo protocollo dimostra a Victor che Peggy ha una certa informazione, ma non concede a Victor la possibilità di conoscere tale informazione.

Questi protocolli sono di tipo interattivo. Victor pone a Peggy una serie di domande: se Peggy conosce il segreto, può rispondere correttamente a tutte le domande; se non lo conosce ha una certa probabilità di indovinare. Dopo un certo numero di risposte corrette, Victor può concludere che le prove di Peggy sono valide. Naturalmente i due possono accordarsi sulle domande, per cui una eventuale registrazione non può essere usata per convincere una terza parte della validità della prova.

2.7.1 Protocollo di base

Supponiamo che Peggy conosca una certa informazione, e che tale informazione sia la soluzione di un problema difficile. Il protocollo zero-knowledge di base è composto da vari passi:

1. Peggy usa la propria informazione e un numero casuale per trasformare il problema difficile in un altro problema difficile, isomorfo rispetto a quello originale. Quindi usa la propria informazione e il numero casuale per risolvere questa nuova istanza del problema difficile.
2. Peggy mette da parte la soluzione della nuova istanza, usando uno schema di tipo bit commitment.
3. Peggy rivela a Victor la nuova istanza. Victor non può usare questo nuovo problema per ottenere informazioni riguardanti l'istanza originale o la sua soluzione.
4. Victor chiede a Peggy di dimostrargli che la vecchia e la nuova istanza sono isomorfe, oppure di rivelare la soluzione messa da parte al punto 2 e di dimostrare che è soluzione della nuova istanza.
5. Peggy fa quanto richiesto.
6. Peggy e Victor ripetono i passi da 1 a 5 per n volte.

Come anticipato, una eventuale registrazione del protocollo non può essere usata come prova, perché Peggy e Victor possono mettersi d'accordo. L'aspetto matematico che sta dietro a questo meccanismo è molto complicato. I problemi e la trasformazione che li lega devono essere scelti accuratamente, in modo che Victor non possa dedurre niente sulla soluzione del problema originale, anche dopo molte iterazioni del protocollo. Non tutti i problemi difficili possono essere usati per questo protocollo, ma, comunque, una buona parte di essi.

2.7.2 Isomorfismo tra grafi

Un grafo è una rete di linee che connettono un insieme di punti. Se due grafi differiscono solo per i nomi dati ai punti, sono detti **isomorfi**. Per un numero di punti grande, determinare se due grafi sono isomorfi può richiedere centinaia di anni; infatti, il problema è di tipo **NP-completo**.

Supponiamo che Peggy conosca l'isomorfismo tra due grafi, G_1 e G_2 . Il seguente protocollo convince Victor del fatto che Peggy conosce una certa informazione:

1. Peggy permuta casualmente G_1 per ottenere un nuovo grafo, H , isomorfo rispetto a G_1 . Siccome Peggy conosce l'isomorfismo tra H e G_1 , conosce anche quello tra H e G_2 . Per chiunque altro, trovare un isomorfismo tra H e G_1 o tra H e G_2 , è difficile come trovarne uno tra G_1 e G_2 .
2. Peggy invia H a Victor.
3. Victor chiede ad Alice di:
 - dimostrare che H e G_1 sono isomorfi, oppure,
 - dimostrare che H e G_2 sono isomorfi.
4. Peggy fa quanto chiesto:
 - dimostra che H e G_1 sono isomorfi, senza dimostrare che anche H e G_2 lo sono, oppure,
 - dimostra che H e G_2 sono isomorfi, senza dimostrare che anche H e G_1 lo sono.
5. Peggy e Victor ripetono i passi da 1 a 4 n volte.

Se Peggy non conosce un isomorfismo tra G_1 e G_2 , non può creare un grafo H isomorfo ad entrambi. Quello che può fare è creare un grafo isomorfo rispetto solo a G_1 o G_2 . In questo modo ha il 50% di probabilità di indovinare quale prova le chiederà Victor al passo 3. Questo protocollo non dà a Victor nessuna informazione utile su come ottenere un isomorfismo tra G_1 e G_2 , perché, per ogni ripetizione del protocollo, Alice genera un nuovo grafo H .

2.8 Firme alla cieca (blind signature)

Una caratteristica essenziale dei protocolli di firma digitale è che, chi firma, sa cosa sta firmando. Questa è un'ottima idea, a parte quando si desidera il contrario. In alcune applicazioni, infatti, è necessario che il firmatario non possa vedere cosa sta firmando, o comunque non “esattamente”.

2.8.1 Firme completamente alla cieca

Bob è un notaio. Alice vuole fargli firmare un documento, ma non vuole fargli sapere niente del contenuto. Bob non è interessato al contenuto del documento, infatti sta solo certificando che il documento è stato registrato in una certa data e ad una certa ora. Il protocollo è il seguente:

1. Alice prende il documento e lo moltiplica per un valore casuale. Tale valore casuale è chiamato **blinding factor** (fattore di accecamento).
2. Alice invia a Bob il documento oscurato.
3. Bob firma il documento oscurato.
4. Alice divide per il blinding factor, ottenendo il documento originale firmato da Bob.

Questo protocollo funziona se le funzioni di firma e di moltiplicazione sono commutative. Se non lo sono, esistono operazioni alternative alla moltiplicazione, adatte allo scopo. Supponiamo, comunque, che l' algoritmo funzioni.

Se il blinding factor è veramente casuale, Bob non può ottenere nessuna informazione sul documento originale. Anche se, al termine del protocollo, mettesse le mani sul documento originale firmato, non potrebbe dimostrare di averlo firmato in quel particolare protocollo.

Le proprietà di una firma completamente alla cieca sono le seguenti:

1. La firma di Bob sul documento è valida. La firma è una prova del fatto che Bob ha firmato il documento.
2. Bob non può mettere in relazione il documento firmato con l'atto della firma di tale documento. Anche se conserva un archivio di tutte le firme di questo tipo fatte, non può determinare quando ha firmato un certo documento.

2.8.2 Firme alla cieca

Il protocollo di firma completamente alla cieca, è inutile per la maggior parte delle applicazioni. Comunque, esiste un modo in cui Bob può sapere cosa sta firmando, mantenendo però le utili caratteristiche della firma alla cieca. La soluzione è di tipo probabilistico: Bob riceve un grande numero di diversi documenti accecati, li esamina tutti tranne uno, quindi firma quest'ultimo.

Per capire come questo sistema funzioni, conviene considerare i documenti accecati come se fossero chiusi in una busta. Il processo di accecamento consiste nel mettere il documento in una busta, il processo di rimozione del blinding factor consiste nell'aprire la busta. Il documento è

firmato grazie ad un pezzo di carta carbone nella busta: quando Bob firma la busta, la sua firma rimane anche sul documento.

Consideriamo un esempio con un gruppo di agenti segreti. Le loro identità sono segrete, nemmeno la loro agenzia sa chi sono. Il direttore dell'agenzia vuole dare ad ognuno di essi un documento firmato del tipo: "Il portatore di questo documento, (inserire qui il nome di copertura dell'agente), ha completa immunità diplomatica". Ogni agente ha la propria lista di nomi di copertura, ma non vuole farla conoscere all'agenzia. D'altra parte, l'agenzia non vuole firmare alla cieca qualunque documento le arrivi. Un agente sveglio potrebbe inviare un messaggio del tipo: "L'agente (nome) si ritira e incassa una pensione di \$1.000.000 all'anno. Firmato, signor Presidente." In questo caso le firme alla cieca possono essere utili.

Supponiamo che ogni agente abbia dieci possibili nomi di copertura, scelti da lui e che nessun altro conosce. Supponiamo anche che agli agenti non importi sotto quale di questi nomi otterranno l'immunità diplomatica. Supponiamo infine che ALICE rappresenti l'agenzia e BOB un particolare agente.

1. BOB prepara n documenti in cui si concede immunità diplomatica, usando per ognuno un differente nome di copertura.
2. BOB acceca ogni documento con un diverso blinding factor.
3. BOB invia gli n documenti accecati ad ALICE.
4. Alice sceglie $n-1$ documenti a caso e per ognuno di questi chiede a BOB il blinding factor.
5. BOB invia ad ALICE i blinding factor richiesti.
6. ALICE apre gli $n-1$ documenti e sia sicura che siano corretti.
7. ALICE firma il documento restante e lo invia a BOB.
8. BOB rimuove il blinding factor e legge il suo nuovo nome di copertura: il documento firmato gli dà immunità diplomatica sotto tale nome.

Questo protocollo è sicuro contro truffe da parte di BOB. Per imbrogliare, deve prevedere quale documento ALICE non esaminerà. Le probabilità a suo favore sono 1 su n , per niente buone. ALICE lo sa e firma tranquillamente il documento che non può esaminare. Con questo documento, il protocollo è lo stesso della firma completamente alla cieca e ne mantiene tutte le proprietà di anonimato.

2.9 Digital cash

Il denaro contante è un problema. Gli assegni e le carte di credito ne hanno ridotto la quantità fisicamente circolante, ma la sua completa eliminazione è ancora lontana. D'altra parte, assegni e carte di credito, permettono di invadere a fondo la privacy delle persone.

Fortunatamente esiste un complicato protocollo che consente di inviare messaggi autenticati, ma non rintracciabili, nel senso che non possono essere attribuiti all'autore. Alice può trasferire del **digital cash** a Bob, senza che Eve venga a sapere l'identità di Alice. Bob può quindi depositare quel denaro elettronico sul proprio conto, anche se la banca non ha idea di chi sia Alice. Al contrario, se Alice prova a comprare con lo stesso digital cash che ha dato a Bob, verrà scoperta dalla banca. E se Bob prova a depositare lo stesso digital cash in due diversi conti, verrà smascherato, ma in tal caso Alice rimarrà anonima. Questo talvolta è chiamato **anonymous digital cash**.

Le caratteristiche del denaro elettronico sono le seguenti:

- Anonimo
- Non riusabile

I protocolli di digital cash sono molto complessi; noi ne svilupperemo uno, un passo alla volta.

2.9.1 Protocollo #1

Questo primo protocollo è un protocollo fisico semplificato per **money order** anonimi:

1. Alice prepara 100 money order anonimi da \$1.000 l'uno.
2. Alice li mette in cento diverse buste, ognuna contenente un pezzo di carta carbone, quindi consegna le buste alla banca.
3. La banca apre 99 buste e conferma che tutte contengono un money order da \$1.000.
4. La banca firma la restante busta (ancora chiusa), firmando così anche il money order, quindi la consegna ad Alice e scala \$1.000 dal suo conto mettendoli in un conto temporaneo.
5. Alice apre la busta e spende il money order da un negoziante.
6. Il negoziante controlla la firma della banca per assicurarsi che il money order sia regolare.
7. Il negoziante porta il money order in banca.
8. La banca verifica la propria firma e versa \$1.000 sul conto del negoziante prendendoli dal conto temporaneo.

Questo protocollo funziona; Alice ha solo l'1% di probabilità di riuscire ad ingannare la banca e se la punizione in caso di truffa è abbastanza severa, non varrà affatto la pena di provare.

2.9.2 Protocollo #2

Il protocollo precedente impedisce ad Alice di scrivere un money order per una cifra più alta di quella dichiarata, ma non le impedisce di fotocopiare il money order e di spenderlo due volte.

Questo è detto problema del **double spending**; per risolverlo dobbiamo complicare il protocollo:

1. Alice prepara 100 money order anonimi da \$1.000 l'uno. In ogni money order include una diversa stringa casuale di unicità, abbastanza lunga da rendere trascurabili le probabilità che la usi anche qualcun altro.
2. Alice li mette in cento diverse buste, ognuna contenente un pezzo di carta carbone, quindi consegna le buste alla banca.
3. La banca apre 99 buste e conferma che tutte contengono un money order da \$1.000, verifica inoltre che le stringhe siano tutte diverse.
4. La banca firma la restante busta (ancora chiusa), firmando così anche il money order, quindi la consegna ad Alice e scala \$1.000 dal suo conto mettendoli in un conto temporaneo.
5. Alice apre la busta e spende il money order da un negoziante.
6. Il negoziante controlla la firma della banca per assicurarsi che il money order sia regolare.
7. Il negoziante porta il money order in banca.
8. La banca verifica la propria firma e controlla nel proprio database di non avere già pagato un assegno con la stessa stringa di unicità; se tutto è corretto, accredita \$1.000 sul conto del negoziante (prendendoli dal conto temporaneo) e memorizza la stringa di unicità in un database.
9. Se il money order è già stato pagato, la banca non lo accetta.

Ora, se Alice prova a spendere una fotocopia del money order, o se il negoziante prova a depositarla, la banca lo saprà.

2.9.3 Protocollo #3

Il precedente protocollo protegge la banca dai truffatori, ma non li identifica. La banca non sa se la persona che ha comprato il money order (la banca non sa chi sia Alice) ha cercato di imbrogliare il negoziante o se il negoziante ha cercato di imbrogliare la banca. Questo protocollo risolve il problema:

1. Alice prepara 100 money order anonimi da \$1.000 l'uno. In ogni money order include una diversa stringa casuale di unicità, abbastanza lunga da rendere trascurabili le probabilità che la usi anche qualcun altro.
2. Alice li mette in cento diverse buste, ognuna contenente un pezzo di carta carbone, quindi consegna le buste alla banca.
3. La banca apre 99 buste e conferma che tutte contengono un money order da \$1.000, verificando che le stringhe siano tutte diverse..
4. La banca firma la restante busta (ancora chiusa), firmando così anche il money order, quindi la consegna ad Alice e scala \$1.000 dal suo conto mettendoli in un conto temporaneo.
5. Alice apre la busta e spende il money order da un negoziante.
6. Il negoziante controlla la firma della banca per assicurarsi che il money order sia regolare.
7. Il negoziante chiede ad Alice di scrivere una stringa casuale d'identità sul money order.

8. Alice lo fa.
9. Il negoziante porta il money order in banca.
10. La banca verifica la propria firma e controlla nel proprio database di non avere già pagato un assegno con la stessa stringa di unicità; se tutto è corretto, accredita \$1.000 sul conto del negoziante, quindi memorizza la stringa di unicità e quella d'identità in un database.
11. Se il money order è già stato pagato, la banca non lo accetta. In tal caso, confronta la stringa d'identità sul money order con quella memorizzata nel database. Se è la stessa, la banca sa che la fotocopia è stata fatta dal negoziante; se è diversa, sa che la fotocopia è stata fatta da chi ha comprato il money order.

Questo protocollo suppone che il negoziante non possa modificare la stringa d'identità sul money order. Siccome l'interazione tra il negoziante e la banca ha luogo dopo che Alice ha speso il denaro, il negoziante potrebbe trovarsi con un money order non valido. Nelle implementazioni pratiche, Alice dovrebbe aspettare presso il registratore di cassa durante l'interazione negoziante-banca, come succede per le carte di credito.

Alice può anche incastrare il negoziante, spendendo il money order una seconda volta e fornendo la stessa stringa d'identità. Se il negoziante non conserva un database dei money order già ricevuti, viene imbrogliato. Il seguente protocollo elimina il problema.

2.9.4 Protocollo #4

Se si scopre che chi ha comprato il money order ha cercato di imbrogliare il negoziante, la banca vorrebbe conoscere l'identità del truffatore. Per poterlo fare dobbiamo abbandonare le analogie col mondo fisico ed entrare nel mondo della crittografia.

Per nascondere il nome di Alice nel money order digitale, possiamo usare la tecnica del **secret splitting**:

1. Alice prepara n money order anonimi per una determinata cifra. Ogni money order contiene una diversa stringa casuale di unicità, X , abbastanza lunga da rendere trascurabili le probabilità che la usi anche qualcun altro. In ogni money order ci sono anche n coppie di stringhe di bit d'identità, I_1, I_2, \dots, I_n (sì, n diverse coppie su *ogni* money order). Ogni coppia è generata in questo modo: Alice crea una stringa contenente il proprio nome, il proprio indirizzo ed ogni altra informazione richiesta dalla banca. Quindi la divide in due pezzi, usando il protocollo di secret splitting (non visto), e si impegna su entrambi, usando un protocollo di bit commitment. Per esempio, I_{37} , consiste di due parti: I_{37S} e I_{37D} . Ogni parte è un pacchetto soggetto a bit commitment: può essere chiesto ad Alice di aprirlo e si può verificare se l'operazione di aperture è stata eseguita correttamente. Ogni coppia rivela l'identità di Alice. Ricapitolando, quindi, ogni money order contiene l'importo, la stringa di unicità e le n stringhe d'identità.
2. Alice acceca tutti gli n money order, usando un protocollo di firma alla cieca, quindi li consegna alla banca.
3. La banca chiede ad Alice di mettere in chiaro $n-1$ money order scelti a caso, per confermarne la correttezza. A questo punto la banca controlla l'importo e la stringa di unicità, e chiede ad Alice di rivelare tutte le stringhe d'identità.

4. Se Alice non ha fatto alcun tentativo di truffa, la banca firma il restante money order, lo invia ad Alice e scala l'importo dal suo conto.
5. Alice mette in chiaro il money order e lo spende da un negoziante.
6. Il negoziante controlla la firma della banca per assicurarsi che il money order sia regolare.
7. Il negoziante chiede ad Alice di rivelare a caso o la metà destra o la metà sinistra di ogni stringa d'identità nel money order. In pratica il negoziante consegna ad Alice una stringa casuale di selezione a n bit. Se il bit i -esimo vale 0, Alice rivela la parte sinistra di I_i ; se vale 1, rivela la parte destra.
8. Alice fa quanto chiesto.
9. Il negoziante porta il money order in banca.
10. La banca verifica la firma e controlla nel proprio database di non avere già pagato un assegno con la stessa stringa di unicità; se tutto è corretto, accredita l'importo sul conto del negoziante, quindi memorizza la stringa di unicità e tutte le informazioni d'identità in un database.
11. Se il money order è già stato pagato, la banca non lo accetta. In tal caso, confronta la stringa d'identità sul money order con quella memorizzata nel database. Se è uguale, la banca sa che il negoziante ha copiato il money order. Se è diversa, la banca sa che chi ha acquistato il money order lo ha fotocopiato. Siccome i due negozianti hanno fornito due stringhe di selezione (quasi certamente) diverse, la banca trova una stringa d'identità di cui un negoziante ha ottenuto la parte destra e di cui l'altro ha ottenuto la parte sinistra. Completando il protocollo di secret splitting, la banca rivela l'identità di Alice.

Questo è un protocollo sorprendente, per questo analizziamolo da punti di vista diversi.

Alice può provare a spendere due volte il money order. La truffa, però, funziona solo se le stringhe di selezione sono identiche, e le probabilità che questo accada sono solo una su 2^n , quindi trascurabili. Alice può anche provare a far firmare alla banca un money order con le stringhe d'identità sbagliate, magari con l'identità di qualcun altro; in questo caso le probabilità sono una su n , quindi decisamente migliori, ma se si stabilisce una pena abbastanza severa, Alice non ci proverà.

Il negoziante non può fare praticamente nulla per imbrogliare. Anche in caso di collusione con Alice, la banca, grazie alla stringa di unicità, è tenuta a pagare il money order una sola volta.

La banca, naturalmente, non può conoscere l'identità di Alice (perlomeno se questa non ha tentato truffe), nemmeno con l'aiuto del negoziante.

L'unica che può imbrogliare è Eve. Se ascolta la comunicazione tra Alice e il negoziante e riesce ad arrivare in banca prima di quest'ultimo, può incassare il money order. Inoltre, quando proverà ad incassare il money order, il negoziante verrà ritenuto un truffatore. Eve può anche rubare il money order ad Alice e spenderlo prima di lei; quando Alice cercherà di spenderlo, ignara del furto, verrà ritenuta una truffatrice. Purtroppo non c'è modo di prevenire questo inconveniente, perché è conseguenza diretta dell'anonimato. Sia Alice che il negoziante devono proteggere i loro bit come proteggono le loro banconote.

2.10 Voto elettronico

Le caratteristiche del voto elettronico sono le seguenti:

- Solo gli aventi diritto possono votare
- Un voto per ogni votante (unico)
- Deve essere segreto
- Non può essere duplicabile
- Non può essere alterabile
- Deve essere verificabile
- Deve essere anonimo

Ancora oggi sono in fase di studio diversi protocolli che rispettino tutte le caratteristiche sopra elencate, quelli che vedremo danno l'idea dei principi su cui si basa il voto elettronico.

2.10.1 Protocollo #1

1. Il votante prende la chiave pubblica del CTF (o seggio).
2. Il votante codifica il voto usando tale chiave e lo invia al CTF.

$$E_{\text{CTF}}(v) \rightarrow \text{CTF}$$

3. Il CTF decifra il voto usando la propria chiave privata.

$$D_{\text{CTF}}[E_{\text{CTF}}(v)] = v$$

Tale protocollo non rispetta nessun vincolo di quelli elencati.

2.10.2 Protocollo #2

1. Il votante cifra il voto con la propria chiave privata (lo firma).

$$S_v(v)$$

2. Il votante cifra il voto firmato con la chiave pubblica del CTF e glielo invia.

$$E_{\text{CTF}}[S_v(v)] \rightarrow \text{CTF}$$

3. Il CTF decifra il voto firmato.

$$D_{\text{CTF}}[E_{\text{CTF}}[S_v(v)]] = S_v(v)$$

4. Il CTF verifica la firma e tabula il voto.

$$V_v[S_v(v)] = v$$

L'unico requisito non rispettato è che il voto non è anonimo, il CTF conosce il voto. Per renderlo anonimo si usa la blind signature.

2.10.3 Protocollo #3

1. Il votante genera n (n dà il grado di sicurezza) sets di messaggi, dove un messaggio è così composto:

$$M = \{\text{tutti i voti} + \text{un random number}\}$$

Nota: ogni scheda ha il voto per un partito ed ogni set ha un certo numero di schede, ognuna con un voto diverso

2. Il votante chiude i sets e li invia al CTF.
3. Il CTF verifica il diritto di voto.
4. Il CTF verifica se il votante ha già votato, quindi registra il votante.
5. Il CTF apre $n-1$ sets (ossia si fa dare dal votante la chiave per $n-1$ sets).
6. Il CTF verifica gli $n-1$ sets e che i numeri random al loro interno siano tutti diversi.
7. Il CTF firma l'ennesimo set (quindi firma tutte le schede votate dal set) e lo invia al votante.
8. Il votante apre il set firmato, quindi produce tante schede firmate dal CTF quanti sono i voti possibili.
9. Il votante sceglie la scheda con il voto, aggiunge un numero random e cifra con la chiave pubblica del CTF.
10. Il votante invia al CTF la scheda cifrata (ossia chiusa) contenente anche il numero random.
11. Il CTF verifica la firma e controlla nel database:
 - se il numero random non è presente allora registra il voto ed il numero random.
 - se il numero random è presente allora il voto è un duplicato.
12. Il CTF tabula pubblicamente il numero random ed il voto.

Problemi:

- La generazione dei numeri random deve essere diversa per tutta la popolazione
→ si risolve con la blind signature
- Il voto non è anonimo rispetto al CTF, il problema è nella comunicazione del passo 10.
→ ci vorrà un protocollo #4
- Chi apre le schede deve essere un'altra entità rispetto a chi le riceve
→ due entità diverse, il Central Tabulation Facility (CTF) ed il Central Legitimization Agency (CLA)
Inoltre anche CTF e CLA potrebbero mettersi d'accordo → protocollo #5