

Appunti dei corsi di Programmazione di Rete Sistemi di elaborazione: Reti II

PROF. G. BONGIOVANNI

4) LA SICUREZZA	2
4.1) Controllo dei diritti di accesso	2
4.1.1) Basic authentication in HTTP 1.0	2
4.1.2) Digest authentication in HTTP 1.1	4
4.1.3) Firewall	4
4.2) Protezione delle risorse da danneggiamento	7
4.2.1) La sicurezza e le estensioni del Web.....	7
4.2.2) La sicurezza e Java	8
4.3) Protezione delle informazioni durante il transito sulla rete.....	9
4.3.1) Crittografia.....	10
4.3.1.1) Crittografia a chiave segreta (o simmetrica).....	13
4.3.1.1.1) Il DES	14
4.3.1.1.2) IDEA	15
4.3.1.1.3) AES	16
4.3.1.2) Crittografia a chiave pubblica	16
4.3.1.2.1) RSA.....	18
4.3.1.2.2) Premesse matematiche	18
4.3.1.2.3) L'algoritmo RSA.....	19
4.3.1.2.4) Correttezza dell'algoritmo RSA	20
4.3.1.2.5) Considerazioni su RSA	21
4.3.2) Funzioni hash e firme digitali.....	21
4.3.3) Protocolli crittografici.....	25
4.3.3.1) Chiave segreta di sessione.....	25
4.3.3.2) Centro di distribuzione delle chiavi.....	27
4.3.3.3) Secure Socket Layer e Secure-HTTP.....	29

4) La sicurezza

In generale, la sicurezza ha a che fare con i seguenti aspetti:

- controllo del diritto di accesso alle informazioni;
- protezione delle risorse da danneggiamenti (volontari o involontari);
- protezione delle informazioni mentre esse sono in transito sulla rete;
- verifica che l'interlocutore sia veramente chi dice di essere.

La rete Internet è nata con la finalità originaria di offrire un efficace strumento per lo scambio di informazioni fra gruppi di ricercatori sparsi per il mondo. Di conseguenza le problematiche relative alla sicurezza non sono state prese in considerazione nel progetto dell'architettura TCP/IP, né tantomeno in quello dei protocolli di livello application.

L'interesse mostrato da chi sfrutta commercialmente Internet, però, sta cambiando la tipologia di utilizzo della rete, e i problemi legati alla scarsa sicurezza diventano sempre più pesanti, per cui ci sono molte attività in corso (compresa la riprogettazione di alcuni protocolli fondamentali quali IP) per incorporare nell'architettura meccanismi di sicurezza.

Nel caso specifico del Web, il fatto che esso sia nato come sistema aperto e disponibile a tutti lo rende particolarmente vulnerabile dal punto di vista della sicurezza (ovviamente, un sistema chiuso è più facile da proteggere). Ciò nonostante, alcuni meccanismi sono disponibili e verranno brevemente descritti nel seguito.

4.1) Controllo dei diritti di accesso

4.1.1) Basic authentication in HTTP 1.0

Nel protocollo HTTP è presente un servizio detto *Basic Authentication* per fornire selettivamente l'accesso a informazioni private, sulla base di un meccanismo di gestione di password.

Sul server si mantengono, in opportuni file di configurazione:

- una lista di *realm*, ossia di porzioni del file system gestito dal server Web per accedere alle quali ci vuole un permesso;
- per ogni realm, una lista degli utenti abilitati con le relative password.

Un realm è di fatto una stringa di testo. Tutti i documenti la cui URL contiene quella stringa fanno parte del realm.

Quando arriva una richiesta GET per un documento che appartiene a un realm, il server non restituisce il documento, ma un messaggio come questo:

```
HTTP/1.0 401 Unauthorized
WWW-Authenticate: Basic realm="NomeRealm"
Server: .....
Date: .....
Content-type: .....
Content-length: 0
```

Quando il client riceve questo messaggio, fa apparire automaticamente una finestra di dialogo predisposta per l'immissione di una *username* e di una *password*.

L'utente immette i dati, e poi preme OK. A questo punto il client invia una nuova richiesta al server, simile alla seguente:

```
GET url(la stessa di prima) HTTP/1.0
Authorization: Basic *****
User-agent: .....
Accept: .....
ecc.
```

dove il testo rappresentato con gli asterischi contiene la username e la password immesse dall'utente, codificate con il metodo *base64 encoding* (uno standard del mondo Unix, definito negli rfc 1341 e 1521, che non costituisce una cifratura ma serve solo a poter trasmettere caratteri ASCII estesi).

Quando il server riceve la richiesta, applica l'algoritmo di decodifica alla stringa di username-password, le confronta con quelle in suo possesso per il realm `NomeRealm` e:

- se è tutto OK restituisce il documento richiesto;
- altrimenti, restituisce un messaggio di errore (`403 forbidden`).

Il client di norma ricorda in una cache la coppia username-password immessa dall'utente e la utilizza anche per i successivi accessi a documenti dello stesso realm; il server deve comunque decodificare tale coppia ogni volta che arriva e verificarne la corrispondenza con quelle in suo possesso.

4.1.2) Digest authentication in HTTP 1.1

Il problema con questo approccio è che username e password di fatto viaggiano in chiaro sulla rete, dato che gli algoritmi usati per la codifica e la decodifica sono noti a tutti, e quindi può essere intercettata.

In proposito c'è una proposta (*Digest Authentication*, rfc 2069) per istituire un meccanismo di cifratura di username e password basato sul meccanismo di *Message Digest*, una sorta di funzione hash facile da calcolare ma difficile da invertire (la vedremo più avanti).

Questo protocollo è di tipo *challenge-response* dove:

- il challenge, inviato dal server al client, contiene un valore detto *nonce* che è ogni volta diverso;
- la response, inviata dal client al server, è il Message Digest (calcolato con l'algoritmo *MD5*) di:
 - nonce ricevuto dal server;
 - username dell'utente;
 - password dell'utente.

In questo modo i dati riservati dell'utente (username e password) non viaggiano mai in chiaro sulla rete.

Quando il server riceve il Message Digest dal client, effettua anch'esso un identico calcolo e confronta i due valori. Se sono uguali è tutto OK, altrimenti no.

4.1.3) Firewall

In molte situazioni in cui esiste una rete aziendale connessa con una rete esterna (ad esempio Internet), può sorgere la necessità di:

- proteggere le informazioni riservate (ad esempio piani strategici, dati finanziari, ecc.) da accessi provenienti dall'esterno della rete aziendale, consentendo solo l'accesso a informazioni pubbliche (ad esempio listino prodotti);
- limitare l'accesso, da parte degli elaboratori posti sulla rete aziendale, alle informazioni presenti sulla rete esterna.

Questo si può ottenere per mezzo di un *firewall* (parete tagliafuoco), che è l'incarnazione moderna del fossato pieno d'acqua (e magari anche di coccodrilli) e del ponte levatoio che proteggevano gli antichi castelli.

Il principio è lo stesso:

- forzare il passaggio di tutto ciò che transita (esseri umani nell'antichità, traffico di rete oggi) attraverso un unico punto di ingresso e uscita, dove si provvede ad effettuare gli opportuni controlli.

Il firewall si inserisce fra la rete aziendale e quella esterna. In tal modo, tutto il traffico dall'una all'altra parte deve per forza transitare attraverso il firewall.

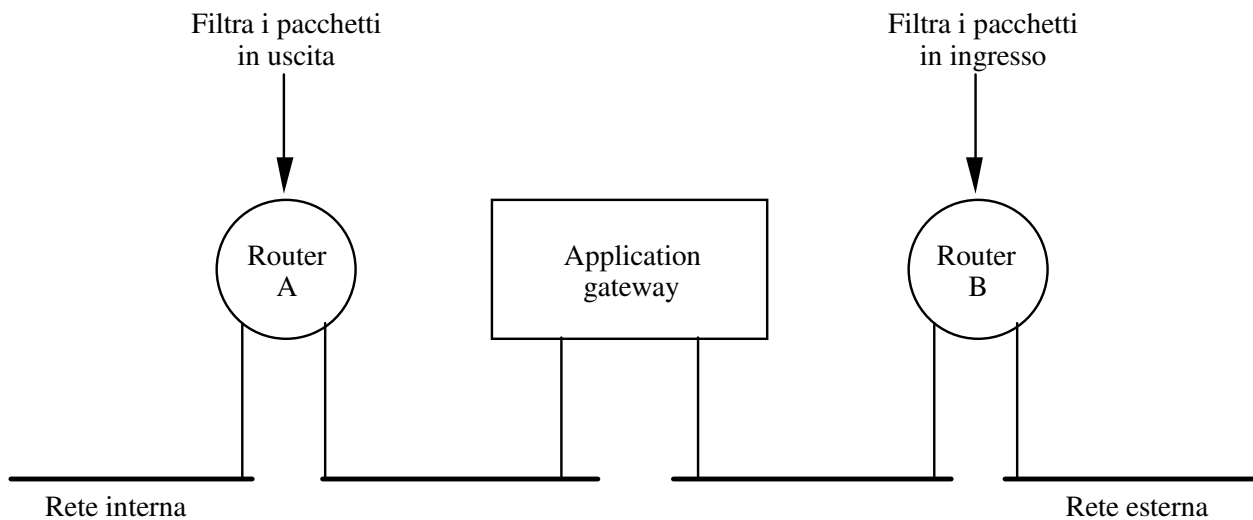


Figura 4-1: Tipica configurazione di un firewall

Esistono molte configurazioni di firewall, più o meno raffinate. In quella sopra illustrata si ricorre a due tipi di componenti:

- due **router** che filtrano i pacchetti (A filtra in uscita, B filtra in ingresso): ogni pacchetto in transito viene esaminato secondo criteri opportunamente impostati; se li soddisfa viene lasciato passare, altrimenti no;
- un **application gateway**: questa componente opera a livello application, e quindi entra nel merito del contenuto dei dati in transito. Ad esempio, un **mail gateway** può decidere se lasciar passare un messaggio di posta elettronica sulla base di subject, o destinatario, o addirittura esaminando il contenuto del messaggio (ad esempio, se c'è la parola "bomb" lo ferma).

Criteri tipici di filtraggio dei pacchetti, che possono anche essere combinati fra loro, sono:

- indirizzo IP (o range di indirizzi) di partenza o di destinazione: questo può servire quando si vogliono connettere fra loro due reti aziendali remote attraverso una rete esterna, ottenendo una cosiddetta **extranet**;
- numero di port di destinazione: questo può servire per abilitare certi servizi e altri no (ad esempio, si disabilita in ingresso il port 23: nessuno dalla rete esterna può fare login su un

elaboratore della rete interna). Un problema in proposito è che alcuni servizi (come il Web) sono spesso offerti anche su porte non standard;

- tipo di connessione usata: è abbastanza comune bloccare tutto il traffico UDP, più difficile da analizzare.

Possono esistere vari application gateway, uno per ogni protocollo di livello application che si vuole controllare.

Spesso, per semplificare il controllo, si installa sulla rete interna un *server proxy* (che può anche coincidere col gateway), cioè un oggetto che agisce da intermediario fra i clienti della rete interna ed i server della rete esterna.

Ad esempio, nel caso di un server proxy per il protocollo HTTP (*HTTP proxy*) si ha tipicamente una situazione come questa:

- i client della rete interna vengono configurati in modo da fare riferimento all'HTTP proxy;
- il firewall viene configurato per lasciar transitare il traffico HTTP da e per l'HTTP proxy.

Quando un utente attiva un link che punta a un server Web della rete esterna succede questo:

1. il client apre una connessione col proxy e gli invia la richiesta;
2. il proxy (che può passare dal firewall) apre una connessione con il server Web esterno e gli invia la richiesta del client;
3. il server Web esterno invia la risposta al proxy;
4. il proxy "gira" la risposta al client.

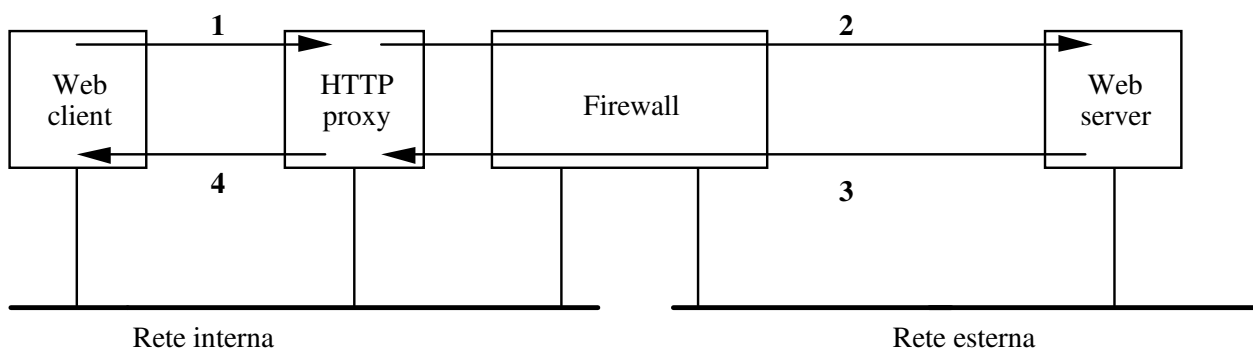


Figura 4-2: Uso di un HTTP proxy

I proxy server hanno anche una funzione di *caching* delle pagine più recenti, in modo da poterle offrire immediatamente se vengono richieste più di una volta, aumentando così l'efficienza e diminuendo l'uso di banda trasmissiva.

Infine, un'ultima nota: se l'azienda vuole pubblicare all'esterno sue informazioni (ad esempio con un server Web) basterà che collochi tale server all'esterno del firewall.

4.2) Protezione delle risorse da danneggiamento

Di norma i server Web non accettano altri metodi che GET (e POST in relazione alle form), quindi impediscono operazioni pericolose quali la scrittura o la cancellazione di file. Inoltre, di norma i server Web non considerano legali le URL che fanno riferimento a porzioni del file system esterne alla parte di competenza del server Web stesso.

Dunque, per lo meno quando si chiedono al server servizi standard (per il recupero di pagine Web) non ci sono grandi pericoli.

4.2.1) La sicurezza e le estensioni del Web

Il discorso però cambia completamente quando si allargano le funzionalità rese disponibili:

- sul server, con programmi CGI;
- sul client, con applicazioni helper.

In entrambi i casi, le opportunità per azioni che causano danneggiamenti travalicano le possibilità di controllo di client e server, e dipendono esclusivamente dalle caratteristiche dei programmi esterni. Si possono aprire delle voragini nella sicurezza!

Essenzialmente, si può dire questo:

- più è potente il programma (helper sul client e CGI sul server) e maggiori sono i pericoli ai quali si è esposti.

Lato client

Supponiamo che l'utente abbia configurato il suo client per lanciare un *interprete PostScript* quando riceve un file PostScript, che di norma contiene un insieme di comandi per la formattazione di testo e grafica indipendenti dalla piattaforma.

In questo scenario, l'interprete PostScript viene lanciato ed esegue uno a uno i comandi contenuti nel file, mostrando sul video (o stampando) il documento.

Ora, PostScript è un completo linguaggio di programmazione, e contiene anche comandi per operazioni sul file system. Se l'autore del documento PostScript ha sfruttato tali potenzialità per recare danni, l'utente ne sopporterà le conseguenze: ad esempio, il file PostScript potrebbe contenere delle istruzioni che cancellano tutti i file dal disco rigido dell'elaboratore.

Lato server

Uno scenario tipico, come abbiamo già detto, è questo:

- il programma CGI compone, in base ai dati immessi nei campi della form, un comando destinato ad un altro programma esterno;
- passa il comando a tale programma esterno e gli chiede di eseguirlo.

Abbiamo visto che nel caso di una interrogazione a una base dati:

- il comando è la formulazione di una query;
- il programma esterno è il gestore della base dati.

Ora, se invece:

- il programma esterno è molto potente (ad esempio: la *shell*);
- il programma CGI non entra a sufficienza nel merito del comando che viene costruito;
- il server Web ha i privilegi di *root*, e lancia con tali privilegi anche il programma CGI (e quindi, di riflesso, anche la shell);

allora si corrono enormi rischi: un utente remoto può inviare un comando per distruggere dei file, ricevere il file delle password, eccetera.

Alcune delle possibili precauzioni per evitare le situazioni sopra descritte sono le seguenti:

- il server in ascolto sulla porta 80 deve girare come root (altrimenti non può aprire un socket su nessuna well-known port), ma i suoi figli (o i thread) che gestiscono le singole richieste devono avere i minimi privilegi necessari per poter svolgere il loro compito (e questo vale anche per i programmi CGI ed i programmi esterni);
- i programmi CGI devono controllare la potenziale pericolosità di ogni comando che ricevono prima di consegnarlo al programma esterno;
- il programma esterno deve essere il meno potente possibile: ad esempio la shell è certamente da evitare, se possibile.

4.2.2) La sicurezza e Java

Una grande attenzione è stata posta, nel progetto del linguaggio e della JVM, ai problemi di sicurezza derivanti potenzialmente dal fatto di mandare in esecuzione sulla propria macchina un codice proveniente da una fonte ignota (e perciò non affidabile in linea di principio).

Ad esempio, si potrebbero ipotizzare questi scenari certamente indesiderabili:

- un applet cifra tutti i file del disco, e chi lo ha programmato chiede un riscatto per fornire la chiave di decifrazione;
- un applet ruba informazioni riservate e le invia a qualcun altro;
- un applet cancella tutti i file del disco.

La prima linea di difesa è stata incorporata nel linguaggio, che è:

- fortemente tipato;
- con controlli sui limiti degli array;
- senza puntatori.

In tal modo è impossibile accedere a zone di memoria esterne a quelle allocate all'applet.

Tuttavia, *Trudy* (un personaggio che conosceremo di più in seguito) si diverte a modificare un compilatore C per produrre dei bytecode in modo da aggirare i controlli effettuati dal compilatore Java.

Per questa ragione, la JVM offre la seconda linea di difesa sotto forma di una componente, detta *bytecode verifier*, che effettua numerosi controlli sui bytecode prima di mandarli in esecuzione, verificando ad esempio che non si cerchi di:

- costruire puntatori;
- chiamare metodi con parametri non validi;
- usare variabili non inizializzate.

La terza linea di difesa è rappresentata dal *class loader*, il meccanismo di caricamento delle classi. Esso impedisce, ad esempio, che una classe dell'applet vada a sostituirsi a una delle classi di sistema in modo da aggirare i meccanismi di sicurezza di quest'ultima.

Infine, un'ulteriore linea di difesa è il *security manager*, una classe che ha il compito di stabilire dei limiti a ciò che il programma (applet o application) può fare.

In particolare, di norma i client Web (e gli AppletViewer) caricano all'avvio un security manager che impedisce a tutti gli applet di:

- accedere al file system locale;
- aprire connessioni di rete con host diversi da quello di provenienza;
- lanciare altri programmi.

Viceversa, un'applicazione Java viene avviata di norma senza alcun security manager associato (a meno che non venga programmata diversamente), e quindi non ha alcuna delle limitazioni sopra citate.

4.3) Protezione delle informazioni durante il transito sulla rete

Esistono ulteriori problemi legati alla sicurezza, che non si possono risolvere semplicemente con meccanismi basati sull'uso di password.

Essi possono essere divisi nelle seguenti aree, collegate fra loro:

- **segretezza**: si desidera inviare delle informazioni riservate, in modo che solo il destinatario sia in grado di leggerle;
- **autenticazione del mittente**: si vuole essere sicuri che colui col quale si dialoga sia veramente chi dice di essere;
- **integrità del messaggio**: si vuole esseri sicuri che il messaggio che arriva non sia stato manomesso durante il viaggio.

Nel campo dell'informatica e soprattutto delle reti, dove da un lato è possibile facilmente creare copie (e anche modificarle) di documenti e dall'altro non si può escludere che Trudy (ovvero un intruso) intercetti le informazioni in transito sulla rete, tutto è più difficile che nella vita quotidiana, dove esistono al proposito meccanismi consolidati (buste sigillate, documenti di identità, autenticazione dei documenti).

E ciò è soprattutto vero in una rete come Internet, dove:

- esiste la necessità di dialogare con entità precedentemente sconosciute;
- ci sono potenzialmente in ogni momento molte Trudy all'ascolto, pronte a rubare informazioni e a sfruttarle a proprio vantaggio.

I problemi precedenti possono essere risolti con un **protocollo crittografico**, che consiste in una serie di passi nei quali si utilizzano le tecnologie seguenti:

- **crittografia**;
- **firme digitali**.

4.3.1) Crittografia

La crittografia (o scrittura nascosta) è la disciplina che si occupa di studiare le tecniche per scrivere un messaggio in modo tale che solo il legittimo destinatario sia in grado di leggerlo. Si occupa dunque del problema della segretezza.

I requisiti principali di tale tecnica sono:

- ragionevole efficienza nella creazione del messaggio;
- estrema difficoltà nella interpretazione del messaggio da parte di chi non è autorizzato;
- possibilità di cambiare con estrema rapidità il metodo usato.

Una prima possibilità è stabilire un metodo di trasformazione (**cifratura** o **encryption**) del messaggio originale e un corrispondente metodo di interpretazione (**decifratura** o **decryption**) che vengono tenuti gelosamente segreti, e sono noti solo alle persone autorizzate.

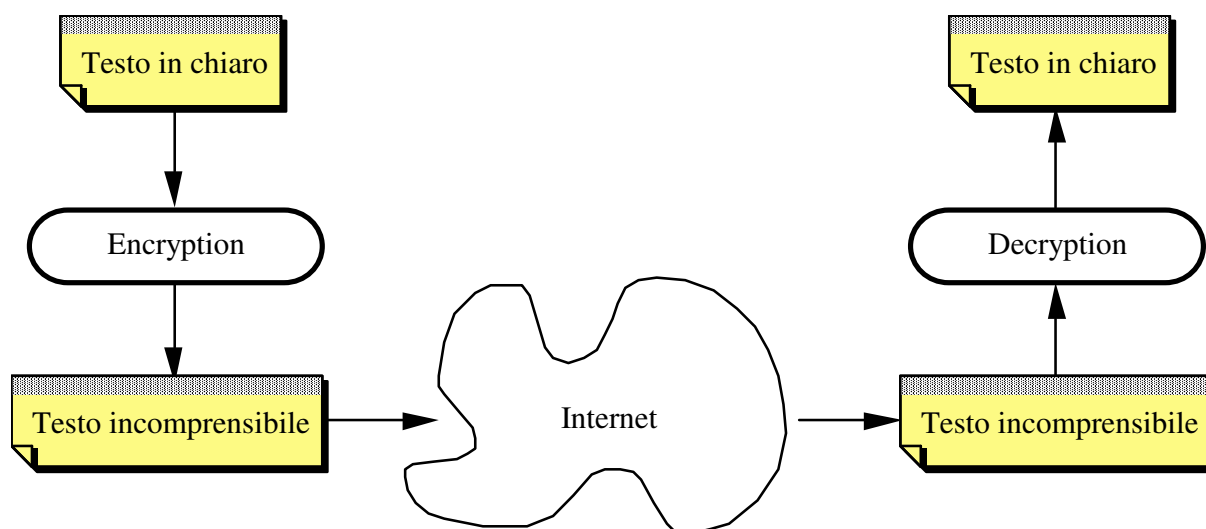


Figura 4-3: Cifratura e decifratura

Ad esempio, un banale metodo di trasformazione (segreto) può essere il seguente: sostituire ogni carattere con quello che, nell'alfabeto, lo segue immediatamente (con wrap-around).

Questo approccio però non soddisfa il terzo requisito, perché per cambiare metodo lo si deve riprogettare completamente. Per questo motivo, si ricorre invece a uno schema diverso, che consiste in:

- un *metodo* di cifratura e uno di decifratura che sono noti a tutti, ma sono parametrizzati da una chiave che deve essere data loro in input assieme al messaggio;
- una sequenza di bit, detta *chiave*, che è nota solo alle persone autorizzate.

Di fatto il metodo di cifratura è una *funzione* E che accetta in ingresso il *testo in chiaro* (*plaintext*) P e una chiave k , producendo il *testo cifrato* (*ciphertext*) C :

$$C = E(P,k)$$

e che normalmente si indica come:

$$C = E_k(P)$$

Quindi, per ogni valore possibile della chiave si ottiene un diverso metodo di cifratura.

A titolo di sempio:

- metodo di cifratura (pubblico): sostituire ogni carattere con quello che lo segue a distanza k (con wrap-around);
- chiave (segreta): il valore k .

Il metodo di decifratura è un'altra funzione (ovviamente collegata alla prima) che accetta in ingresso il testo cifrato C, una chiave k e restituisce il testo in chiaro originale P:

$$P = D_k(C)$$

Ovviamente, si dovrà avere che:

$$D_k(E_k(P)) = P$$

Come vedremo, non è detto che si debba usare la stessa chiave nelle due fasi. Il modello risultante è il seguente:

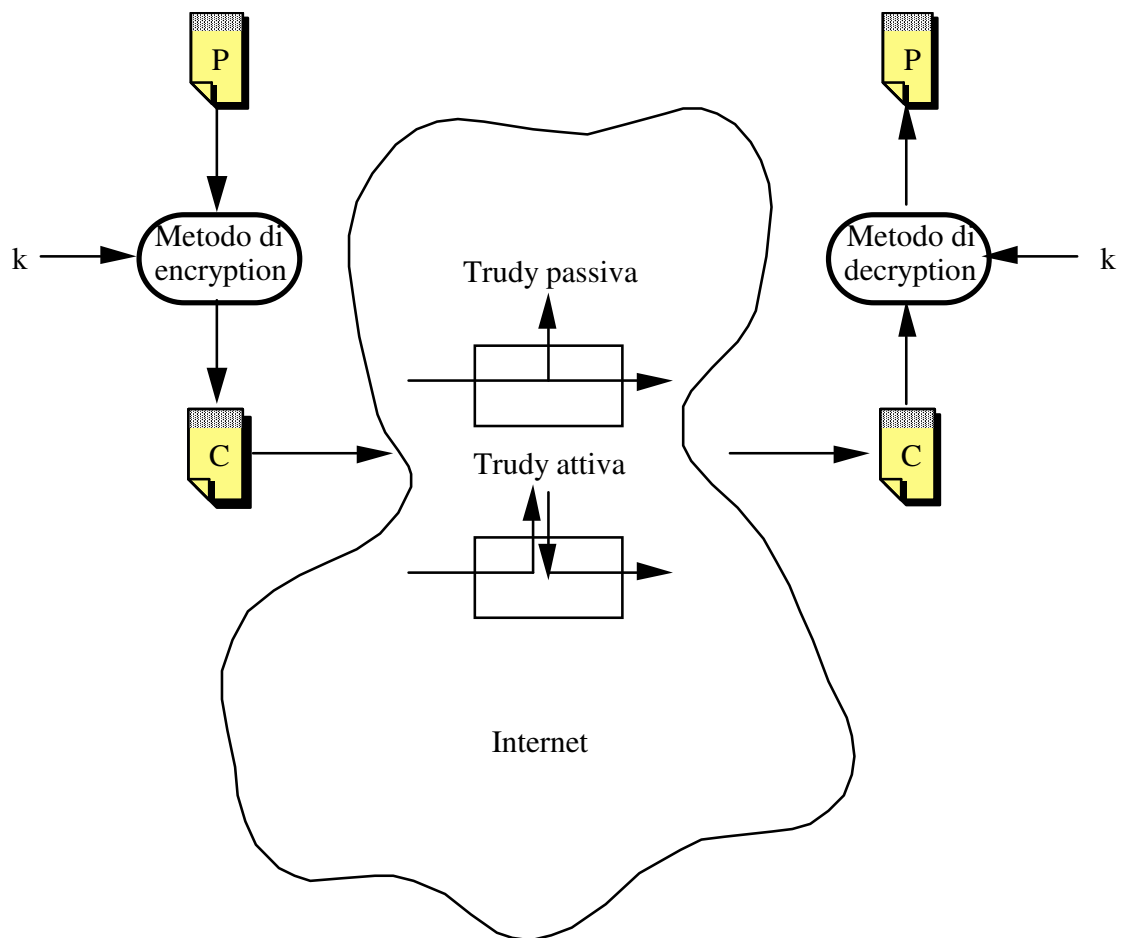


Figura 4-4: Cifratura e decifratura basate su chiave

Trudy può essere passiva (ascolta solamente) o attiva (ascolta ed altera i messaggi che riesce ad intercettare).

La crittografia, come abbiamo detto, si occupa di trovare buoni metodi per effettuare la cifratura e la decifratura. La **criptoanalisi** (*cryptoanalysis*) invece si occupa di scoprire modi per decifrare i messaggi pur non conoscendo le regole note alle persone autorizzate (in particolare, la chiave).

Questa operazione può essere portata avanti in due modi:

- provare ad applicare al testo cifrato il metodo di decifratura con tutti i possibili valori della chiave. Questo approccio, detto di **forza bruta**, produce certamente il testo in chiaro originale prima o poi, ma è ovviamente molto oneroso, ed anzi lo è tanto più quanto è lunga la chiave (ad esempio con 128 bit di chiave, ci sono 2^{128} prove da effettuare);
- scoprire le eventuali debolezze delle funzioni E e D, per ridurre lo spazio delle chiavi da esplorare.

Ci sono due principi fondamentali che ogni metodo di cifratura deve rispettare:

- i messaggi originali devono contenere della ridondanza: se così non fosse, ogni possibile messaggio cifrato sarebbe comunque valido e Trudy potrebbe quindi inviarne a piacere, dato che a destinazione essi verrebbero considerati autentici dopo essere stati decifrati;
- i messaggi originali devono contenere qualcosa (ad esempio un **time stamp**) che impedisca a Trudy di rispeditare nuovamente un messaggio valido. Altrimenti, ogni volta che Trudy intercetta un messaggio può inviarlo nuovamente per i propri scopi.

4.3.1.1 Crittografia a chiave segreta (o simmetrica)

In questo tipo di crittografia, il mittente e il destinatario (Alice e Bob) si accordano, ad esempio incontrandosi di persona lontano da occhi indiscreti, su una singola chiave che verrà usata sia in fase di cifratura che di decifratura.

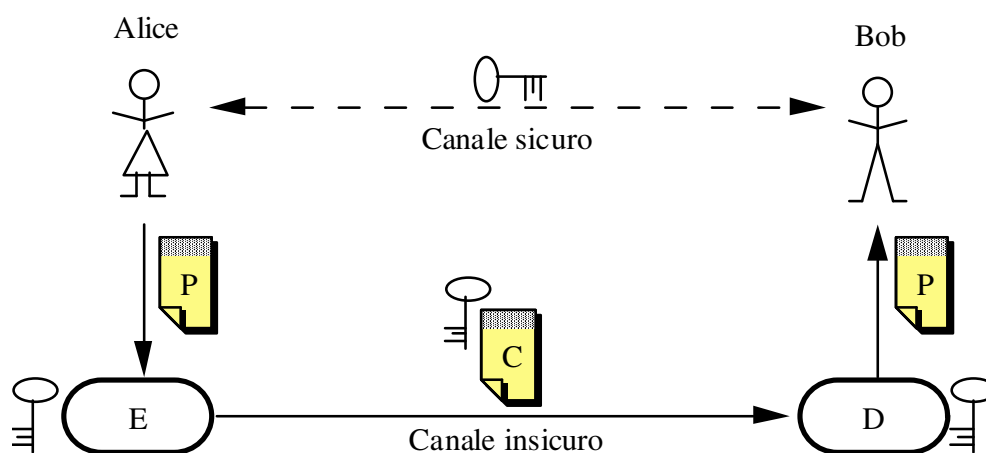


Figura 4-5: Crittografia a chiave segreta

Tutti i moderni cifrari a chiave segreta condividono alcune caratteristiche di fondo.

Essi operano su un blocco di bit alla volta (*cifratura a blocchi*) e sono costituiti da un certo numero di stadi, ciascuno dei quali riceve in ingresso i valori prodotti in uscita dallo stadio precedente e fornisce in uscita i valori allo stadio successivo. In molti di tali stadi la trasformazione operata sui bit è basata sull'uso di raffinati meccanismi di calcolo detti *reti di sostituzione e permutazione* (*substitution-permutation network*) che, ricevendo in ingresso i bit da elaborare e la chiave di cifratura (o un sottoinsieme dei suoi bit) producono i bit in uscita. È soprattutto la struttura delle reti di sostituzione e permutazione (spesso chiamate *S-box*) che costituisce il cuore del progetto di un codice e ne caratterizza la robustezza rispetto a due moderni tipi di criptoanalisi: la *criptoanalisi lineare* e la *criptoanalisi differenziale*, che cercano di ricostruire la struttura del codice indagando matematicamente le eventuali regolarità e correlazioni che si dovessero riscontrare nel testo cifrato.

4.3.1.1.1) Il DES

L'algoritmo più diffuso in questa categoria è il *DES (Data Encryption Standard)*, inventato dall'IBM e adottato come standard del governo U.S.A. nel 1977.

Il testo in chiaro è codificato in blocchi di 64 bit, che producono ciascuno 64 bit di testo cifrato (*cifratura a blocchi*).

L'algoritmo è parametrizzato da una chiave di 56 bit e consiste di ben 19 stadi, in ciascuno dei quali si opera una trasformazione dell'output dello stadio precedente.

Inoltre, in 16 dei 19 stadi la trasformazione è effettuata mediante S-box parametrizzati ciascuno da un diverso sottoinsieme di bit della chiave.

Il DES è stato al centro di controversie sin dal giorno in cui è nato:

- il progetto originale IBM prevedeva chiavi da 128 bit invece che da 56 bit, ma i militari U.S.A. "suggerirono" attraverso l'*NSA (National Security Agency)*, detta anche malignamente *No Such Agency* tale riduzione. Secondo molti, la riduzione fu motivata dall'esigenza di mantenere la capacità (con opportune potenti macchine) di rompere il codice;
- oggi il DES non è più considerato sicuro, in quanto recenti tecniche di criptoanalisi differenziale hanno ridotto lo spazio di ricerca a 2^{43} possibilità;

Una sua variante, il *Triple DES*, è però a tutt'oggi considerato sicuro, in quanto non si conosce alcun modo di romperlo. Il meccanismo è il seguente.

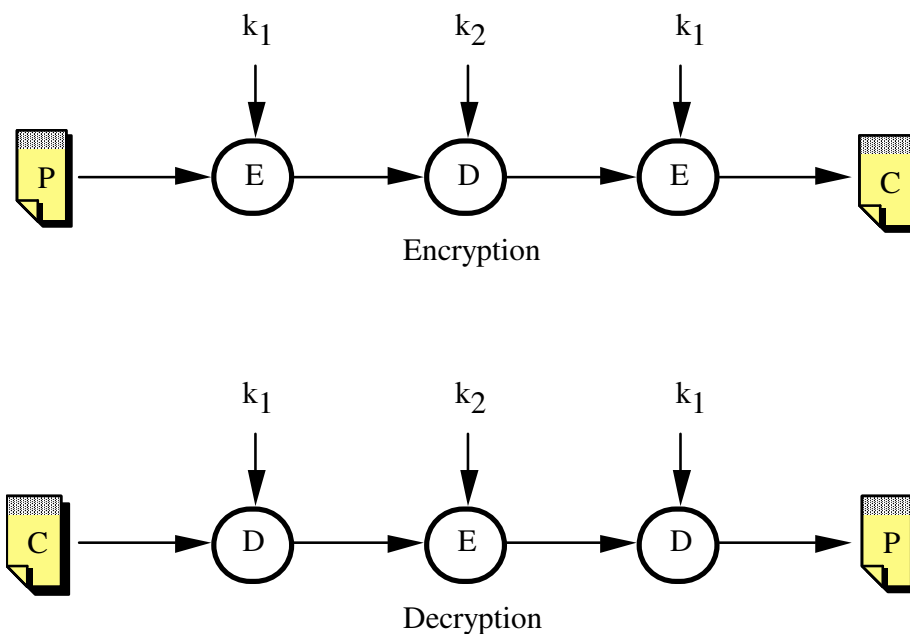


Figura 4-6: Triple DES

Questo schema, ponendo $k_1=k_2$, garantisce la compatibilità all'indietro col normale DES.

Effettivamente il Triple DES costituisce un codice per il quale l'approccio della forza bruta richiede 2^{112} tentativi: anche con un miliardo di chip che effettuano un miliardo di operazioni al secondo, ci vorrebbero 100 milioni di anni per la ricerca esaustiva.

4.3.1.1.2) IDEA

Un altro importante algoritmo a chiave segreta è **IDEA** (*International Data Encryption Algorithm*).

Esso fu progettato nel '90 in Svizzera, e per questa ragione non è soggetto alle limitazioni sull'uso e sull'esportazione che esistono in U.S.A. (dove gli algoritmi di cifratura sono a tutti gli effetti di legge considerati armi da guerra).

Come il DES, IDEA effettua una cifratura a blocchi (di 64 bit), ma usa una chiave di 128 bit e consiste di otto stadi, nei quali ogni bit di output dipende da tutti i bit in input (il che non vale per il DES).

Non sono noti risultati di criptanalisi che lo indeboliscono.

4.3.1.1.3) AES

Il 2 gennaio 1997 l'ente americano NIST (National Institute of Standards and Technology) iniziò il procedimento per designare il successore del DES, denominato *Advanced Encryption Standard (AES)*. Tale procedimento si sviluppò attraverso varie fasi, tutte gestite con la massima diffusione internazionale e circolazione delle idee e delle proposte, e si concluse il 4 dicembre 2001 con la scelta del cifrario Rijndael, il cui nome deriva da quello dei due autori, i belgi J. Daemen e V. Rijmen.

AES (ossia il cifrario Rijndael) effettua la cifratura su blocchi di 128 bit, e può utilizzare chiavi di 128, 192 oppure 256 bit. Il numero di stadi dipende dalla lunghezza della chiave usata: si impiegano 10 stadi con chiavi da 128 bit, 12 stadi con chiavi da 192 bit e 14 stadi con chiavi da 256 bit.

Gli S-box sono stati progettati con grande attenzione e sono stati oggetto di approfondite indagini da parte della comunità scientifica internazionale. Come per IDEA, non sono noti risultati di crittanalisi che indeboliscano AES, e la lunghezza delle chiavi utilizzabili (soprattutto nel caso del valore 256) rende privo di speranze un attacco di forza bruta.

4.3.1.2) Crittografia a chiave pubblica

Un problema di fondo affligge la crittografia a chiave segreta quando aumenta il numero di persone che vogliono essere in grado di comunicare fra loro.

Poiché ogni coppia di persone deve essere in possesso di una corrispondente chiave, se N persone desiderano comunicare fra loro ci vogliono $N(N-1)/2$ chiavi, cioè una per ogni coppia.

Ciò rende estremamente difficile il problema della distribuzione delle chiavi, che resta il punto debole di tutto il sistema.

Nella seconda metà degli anni '70 fu introdotto (Diffie e Hellmann, Stanford University) un tipo di crittografia radicalmente nuovo, detto a *chiave pubblica* (o *asimmetrica*).

L'idea è questa:

- ognuno possiede due chiavi, legate una all'altra:
 - una è la *chiave privata*, nota solo al proprietario;
 - l'altra è la *chiave pubblica*, nota a tutti;

- ciò che viene cifrato con la prima chiave può essere decifrato con l'altra (e di solito viceversa);
- è quasi impossibile, e comunque estremamente oneroso, derivare la prima chiave anche se si conosce la seconda.

Dunque, per un gruppo di N persone sono necessarie solo 2N chiavi.

Il funzionamento, per ottenere la *segretezza*, è questo:

- Alice cifra il messaggio con la chiave pubblica di Bob (che è nota a tutti);
- Bob decifra il messaggio con la propria chiave privata (che è nota solo a lui).

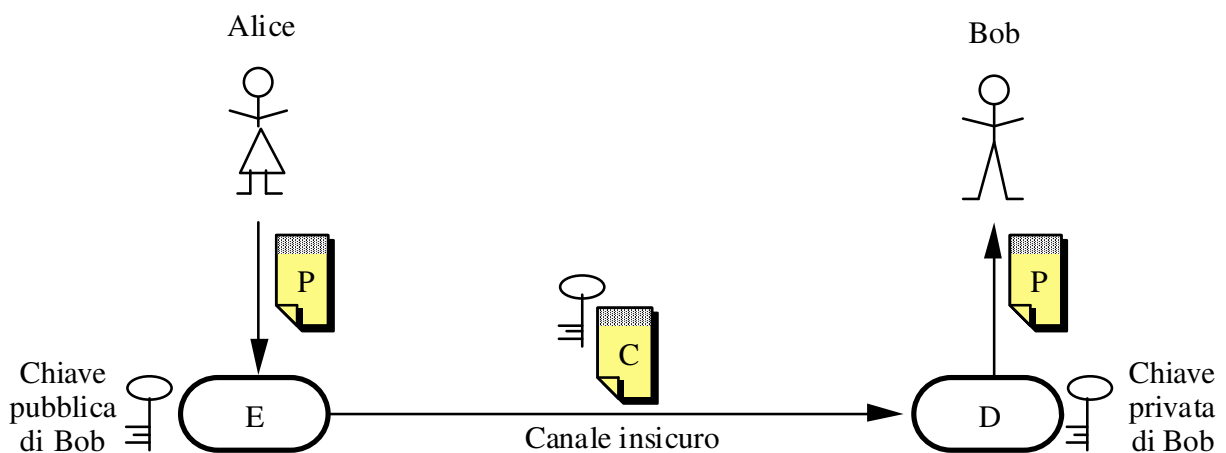


Figura 4-7: Riservatezza mediante crittografia a chiave pubblica

La crittografia a chiave pubblica fornisce anche un meccanismo per garantire l'*autenticazione del mittente*, cioè la garanzia che esso provenga veramente dall'autore e non da qualcun altro, e l'*integrità del messaggio*, cioè la garanzia che il messaggio non sia stato alterato.

In questo caso si opera alla rovescia:

- Alice cifra il messaggio con la propria chiave privata;
- Bob lo decifra con la chiave pubblica di Alice.

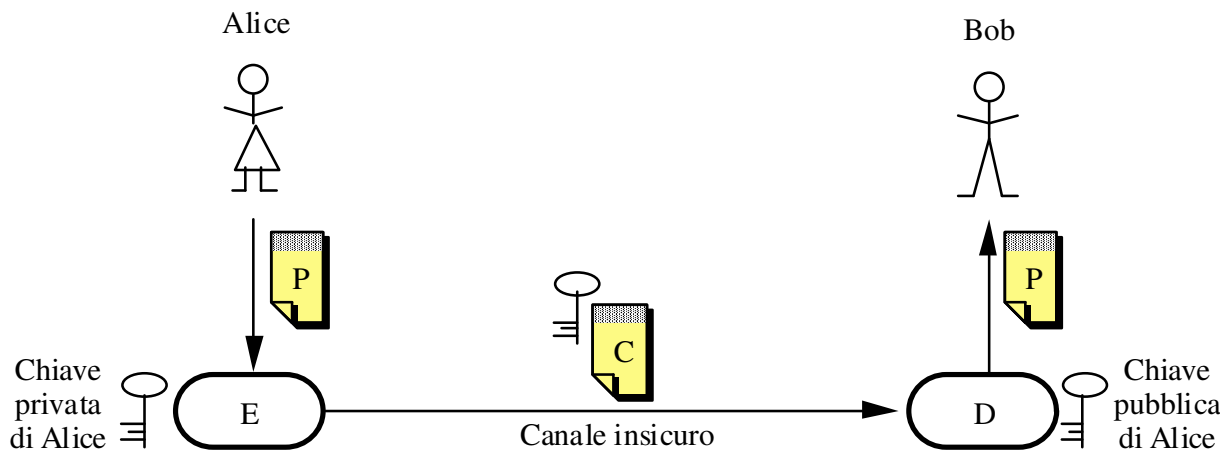


Figura 4-8: Autenticazione mediante crittografia a chiave pubblica

In questo caso non c'è segretezza, dato che chiunque può decifrare il messaggio, ma nessuno se non Alice avrebbe potuto costruirlo, ed inoltre nessuno può averlo alterato.

4.3.1.2.1) RSA

L'algoritmo a chiave pubblica più noto ed usato è l'algoritmo **RSA** (dalle iniziali degli autori Rivest, Shamir e Adleman), nato nel 1978.

Esso trae la sua efficacia dalla enorme difficoltà di trovare la fattorizzazione di un grande numero (si stima che serva un miliardo di anni di tempo macchina per fattorizzare un numero di 200 cifre, e 10^{25} anni per un numero di 500 cifre).

4.3.1.2.2) Premesse matematiche

Aritmetiche finite

Si opera in un'aritmetica finita (o circolare) *modulo n*, cioè a valori interi $0, 1, \dots, n-1$. In tale aritmetica:

- Somma algebrica e prodotto si calcolano normalmente, applicando poi al risultato l'operazione di modulo n ;
- Dato un numero x , *l'inverso di x* è quel numero y tale che

$$xy \equiv 1 \pmod{n}.$$

Funzione di Eulero

Dato un intero n , la funzione di Eulero $\Phi(n)$ rappresenta il numero di numeri primi con n e minori di n . Casi particolari sono:

- Se n è primo, $\Phi(n) = n - 1$;

- Se n e' il prodotto di due numeri primi s e t , $\Phi(n) = (s - 1)(t - 1)$.

Teorema di Eulero

Dati due numeri m ed n primi fra loro, si ha che

$$m^{\Phi(n)} \equiv 1 \pmod{n}$$

Se n e' il prodotto di due numeri primi s e t , si ha (vedi funzione di Eulero) che

$$m^{(s-1)(t-1)} \equiv 1 \pmod{n}$$

4.3.1.2.3) L'algoritmo RSA

1. Scegliere *due grandi numeri primi s e t* (tipicamente maggiori di 10^{100}) e calcolare $n = st$;
2. Calcolare $z = \Phi(n)$, ossia $z = (s - 1)(t - 1)$;
3. Scegliere un numero d *primo con $\Phi(n)$* ;
4. Trovare il numero e *inverso di d nell'aritmetica finita modulo $\Phi(n)$* , ossia il numero e per cui:

$$de \equiv 1 \pmod{\Phi(n)}$$

- Nota 1: questo passo, come vedremo, serve a garantire che cifratura e decifratura siano due funzioni inverse;
- Nota 2: In teoria e si puo' calcolare (grazie al teorema di Eulero) come:

$$e = (d^{-1} \pmod{\Phi(n)})$$

Infatti:

$$(de)_{\pmod{\Phi(n)}} = (d d^{-1} \pmod{\Phi(n)})_{\pmod{\Phi(n)}} = (d^{-1} \pmod{\Phi(n)})_{\pmod{\Phi(n)}}$$

ma poiche' d e' $\Phi(n)$ sono primi fra loro, per il teorema di Eulero si ha

$$d^{-1} \pmod{\Phi(n)} \equiv 1 \pmod{\Phi(n)}$$

e quindi anche

$$de \equiv 1 \pmod{\Phi(n)}$$

dunque e e' l'inverso di d .

In pratica pero' questo approccio e' computazionalmente inapplicabile per grandi numeri, per cui si va per tentativi scegliendo e come il piu' piccolo valore x per il quale $(dx - 1) / \Phi(n)$ e' intero.

5. Porre:

- Come **chiave pubblica** la coppia (n,e) ;
- Come **chiave privata** la coppia (n,d) .

Ovviamente s e t vanno tenuti segreti, anzi vanno distrutti.

6. Suddividere il testo da cifrare in **blocchi di k bit** tali che, considerando ogni blocco P come un intero non negativo, il suo valore sia minore di n . Ossia, prendere k tale che:

$$2^k - 1 < n.$$

7. Utilizzare le seguenti funzioni per la cifratura e la decifratura:

- La **cifratura** produce a partire da P un blocco C tale che:

$$C = (P^e)_{\text{mod } n}$$

- La **decifratura** produce a partire da C un blocco P tale che:

$$P = (C^d)_{\text{mod } n}$$

4.3.1.2.4) Correttezza dell'algorithm RSA

Come visto sopra, si ha:

$$C = (P^e)_{\text{mod } n}$$

$$P = (C^d)_{\text{mod } n}$$

Dunque, possiamo scrivere che il blocco P' risultante dalla decifratura di un blocco C che a sua volta e' la cifratura di un blocco P vale:

$$P' = (P^{ed})_{\text{mod } n}$$

Ricordiamo che d, e sono inversi nell'aritmetica modulo $\Phi(n)$, per cui possiamo scrivere:

$$P' = (P^{k\Phi(n)+1})_{\text{mod } n} = ((P^{\Phi(n)})^k P)_{\text{mod } n}$$

Se P e' primo con n , per il teorema di Eulero si ha:

$$P^{\Phi(n)} \equiv 1 \text{ mod } n$$

Da cui discende che:

$$P' = (1^k P)_{\text{mod } n} = P$$

Se invece P non e' primo con n , valgono le seguenti considerazioni:

- Poiche' $n = st$, P *deve* avere fra i suoi divisori s oppure t (ma non entrambi, altrimenti sarebbe uguale ad n);
- Supponendo che $P = qs$ (cioe' che P abbia s fra i suoi divisori) si avra' che $q < t$, altrimenti P sarebbe uguale o maggiore di n .

Allora, si può scrivere:

$$P' = (qs^{ed})_{\text{mod } n} = (qs^{k\Phi(n)+1})_{\text{mod } n}$$

Espandendo $\Phi(n)$, si ha:

$$P' = (qs^{k(s-1)(t-1)} qs)_{\text{mod } n}$$

Che si può riscrivere come:

$$(1) \quad P' = ((qs^{k(s-1)})^{(t-1)} qs)_{\text{mod } n}$$

4.3.1.2.5) Considerazioni su RSA

Si noti che se non fosse difficile fattorizzare n (che è noto a tutti), Trudy potrebbe facilmente:

1. trovare p e q , e da questi z ;
2. una volta determinati z ed e (anch'esso noto a tutti), trovare d con l'*algoritmo di Euclide*.

Nel 1994 RSA è stato rotto, in risposta ad una sfida degli autori pubblicata su Scientific American. Il procedimento si riferì a una chiave di 129 cifre (426 bit), e furono impiegati 1600 elaboratori su Internet per 8 mesi, per un totale di 5000 anni di calcolo a 1 MIPS (milione di istruzioni al secondo).

D'altronde, poiché RSA lavora con i numeri, la dimensione della chiave è variabile e può essere aumentata a piacere, per controbilanciare gli effetti derivanti dal miglioramento delle prestazioni degli elaboratori.

Infine, poiché gli algoritmi a chiave pubblica sono molto più onerosi computazionalmente di quelli a chiave segreta (di un fattore da 100 a 1000), essi sono usati soprattutto per negoziare in modo sicuro (come vedremo fra breve) una chiave segreta, detta *chiave di sessione*, da usare nel corso della comunicazione vera e propria la cui riservatezza viene protetta con un algoritmo quale DES o IDEA.

4.3.2) Funzioni hash e firme digitali

Come abbiamo visto, la crittografia a chiave pubblica può essere usata per autenticare l'origine di un messaggio e per garantirne l'integrità, ossia di fatto per *firmare* un messaggio.

Però, a causa del costo computazionale, questo approccio sembra eccessivo: cifrare tutto il messaggio solo per firmarlo è poco efficiente. Inoltre, è scomodo rendere l'intero documento illeggibile ove solo una firma sia richiesta.

Ambedue i problemi si risolvono con una tecnica diversa e più efficiente, che però introduce un piccolo rischio in termini di sicurezza.

La tecnica in questione si basa sull'uso delle *funzioni hash* (dette anche *funzioni digest*, cioè funzioni riassunto) che vengono applicate al messaggio e ne producono, appunto, un riassunto (*message digest*).

Tale riassunto è in generale di dimensioni ridotte, tipicamente fra i 10 e i 20 byte, indipendentemente dalla lunghezza del messaggio originario.

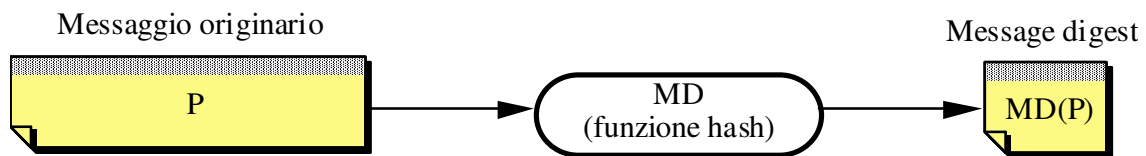


Figura 4-9: Calcolo del riassunto del messaggio

Per essere adatta allo scopo, la funzione hash deve possedere i seguenti requisiti:

- è computazionalmente poco oneroso calcolare $MD(P)$;
- dato $MD(P)$ è praticamente impossibile risalire a P ;
- è praticamente impossibile trovare due documenti P_1 e P_2 tali per cui $MD(P_1) = MD(P_2)$. Si noti che questo requisito non discende dalla proprietà precedente.

Per soddisfare l'ultimo requisito il riassunto deve essere piuttosto lungo, almeno 128 bit. Ad ogni modo, è chiaro che dal punto di vista teorico non è possibile garantire che il requisito sia sempre soddisfatto, poiché in generale la cardinalità dello spazio dei messaggi è molto superiore a quella dello spazio dei riassunti.

L'algoritmo più diffuso per la generazione del message digest è *MD5 (Message Digest 5, Rivest 1992)*, il quinto di una serie, definito nell'RFC 1321. Produce digest di 128 bit, ognuno dei quali è funzione di tutti i bit del messaggio. Funziona a circa 7 MBps su un Pentium Pro a 200 MHz.

Un primo e semplice schema di utilizzo del message digest è il seguente, volto a garantire l'*integrità del messaggio*, ovvero a garantire che il messaggio che giunge a destinazione sia identico a quello che è stato inviato:

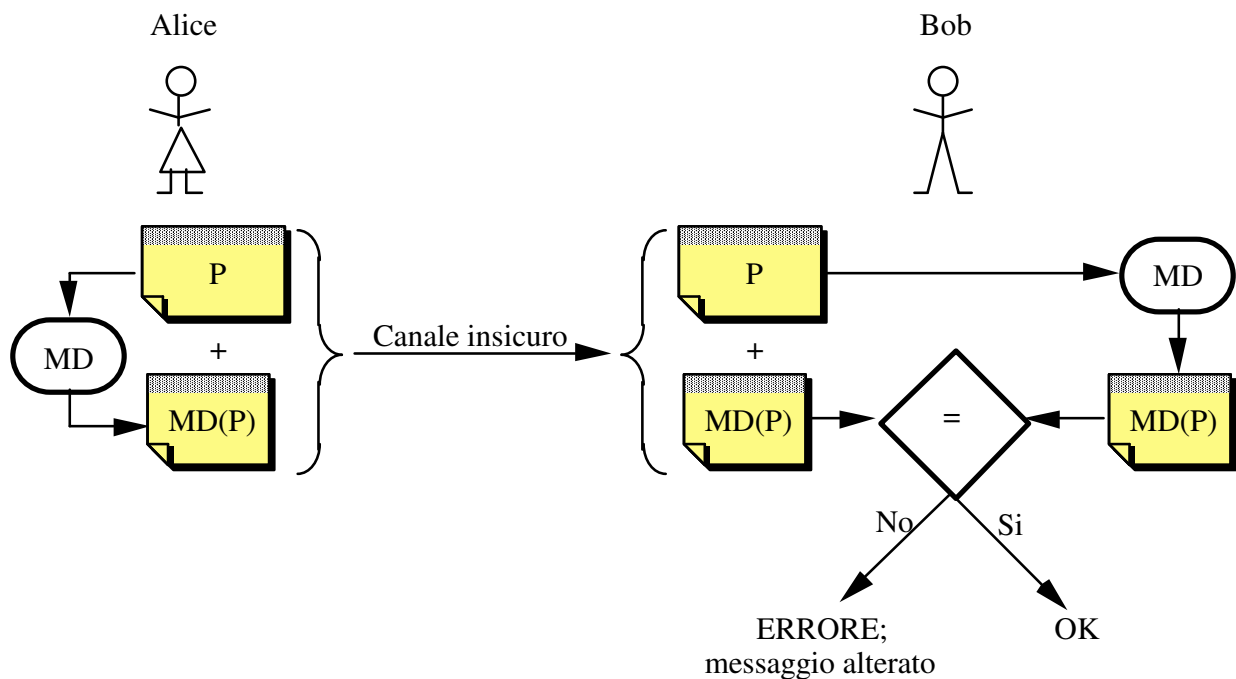


Figura 4-10: Uso del digest per il controllo di integrità del messaggio

Alice invia il messaggio corredato del riassunto; quando Bob riceve il tutto, ricalcola il riassunto e lo confronta con quello ricevuto.

Ovviamente, questa semplice modalità è esposta all'attacco di Trudy, che potrebbe intercettare il messaggio, sostituirlo con uno diverso correlato del relativo digest, e inviarlo a Bob come se fosse quello proveniente da Alice.

Per risolvere questo problema si ricorre a uno schema leggermente più complesso, che fa uso anche della crittografia a chiave pubblica: il riassunto, prima di essere spedito, viene cifrato dal mittente con la propria chiave privata e decifrato dal destinatario con la chiave pubblica del mittente. Un riassunto cifrato in questo modo si dice *firma digitale (digital signature)* del mittente, perché assicura sia l'integrità del messaggio che l'autenticità del mittente, proprio come una firma apposta (in originale) in calce a un documento cartaceo.

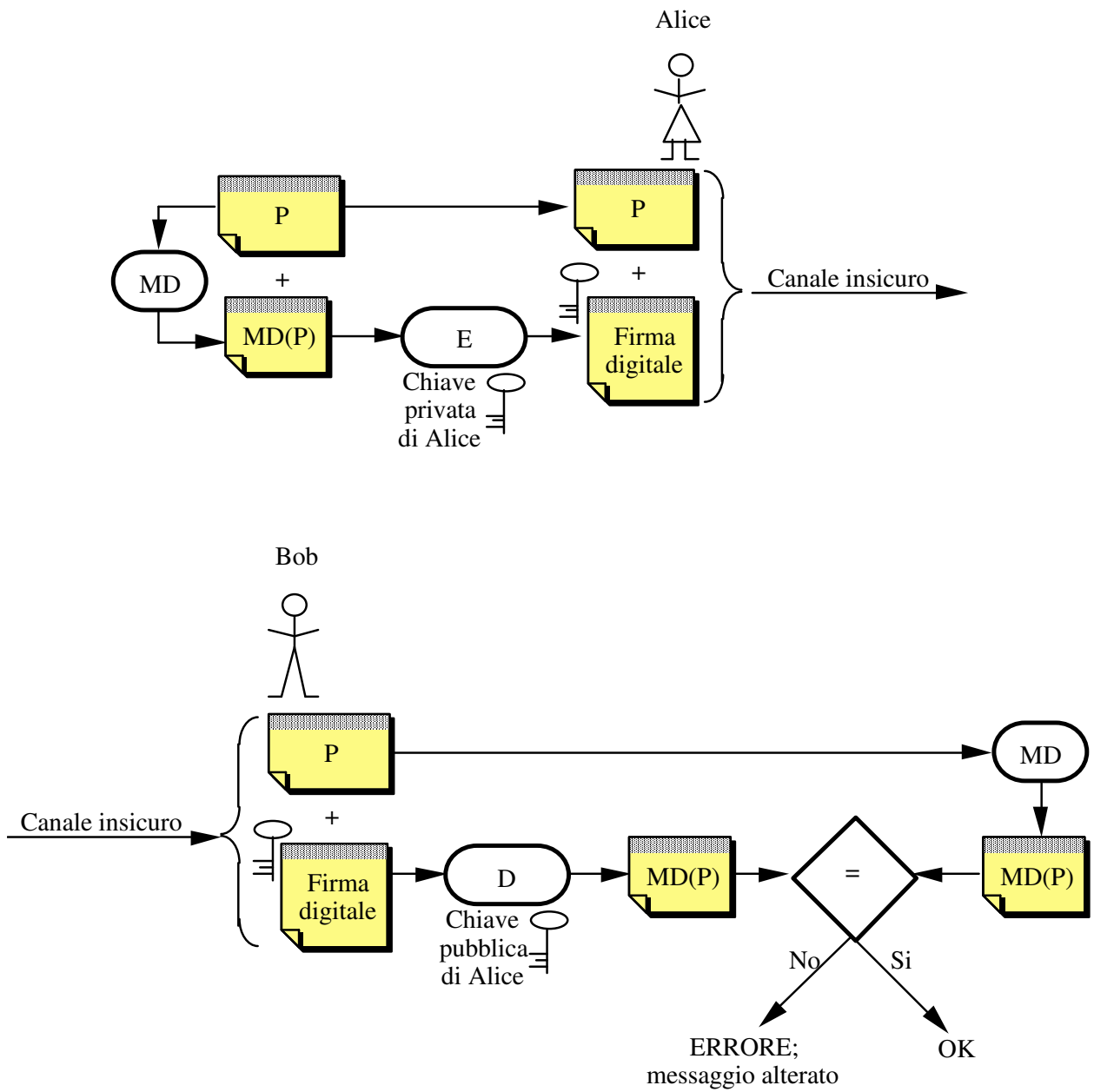


Figura 4-11: Firma digitale

4.3.3) Protocolli crittografici

Quello visto sopra è un esempio di *protocollo crittografico*, cioè di una serie di regole che le parti debbono seguire per assicurarsi una conversazione conforme ai requisiti desiderati.

Un protocollo crittografico in generale non specifica gli algoritmi da usare nei vari passi, ma piuttosto:

- quali tecniche adottare (ad esempio: crittografia a chiave pubblica e/o privata, message digest, ecc.);
- quale successione di passi deve essere seguita, e quale tecnica va adottata in ogni passo.

Esistono vari protocolli crittografici, che si differenziano per:

- il contesto iniziale (ad esempio: i due partecipanti hanno una chiave segreta in comune o no? Conoscono le rispettive chiavi pubbliche o no?);
- gli scopi da raggiungere (ad esempio: autenticazione, segretezza, o entrambi?).

Vedremo ora alcuni protocolli che rivestono un particolare interesse in un contesto come quello del Web, dove:

- è possibile aver bisogno di autenticazione e segretezza nel dialogo con entità mai conosciute prima;
- i canali sono insicuri, e soggetti all'intrusione di Trudy (e magari anche di Gambadilegno!).

4.3.3.1) Chiave segreta di sessione

Un primo problema da affrontare e risolvere mediante un protocollo crittografico è il seguente: poiché la crittografia a chiave segreta è molto più efficiente di quella a chiave pubblica, si vuole usare la prima nel corso della comunicazione effettiva che deve essere portata avanti.

Però, in un contesto distribuito come il Web, è impensabile che ogni potenziale coppia di fruitori disponga di una chiave segreta. Dunque, bisogna trovare un protocollo per concordare, all'inizio della sessione, la chiave segreta da usare durante il resto della sessione, detta per questo chiave *segreta di sessione*.

Un primo protocollo è di per se molto semplice, e sfrutta la crittografia a chiave pubblica:

1. Bob invia la sua chiave pubblica ad Alice;

2. Alice genera una nuova chiave segreta, la cifra con la chiave pubblica di Bob e la invia a Bob;
3. Bob riceve la chiave segeta (cifrata) e la decifra con la propria chiave privata;
4. Alice e Bob a questo punto condividono la chiave segreta di sessione per mezzo della quale possono comunicare in tutta sicurezza.

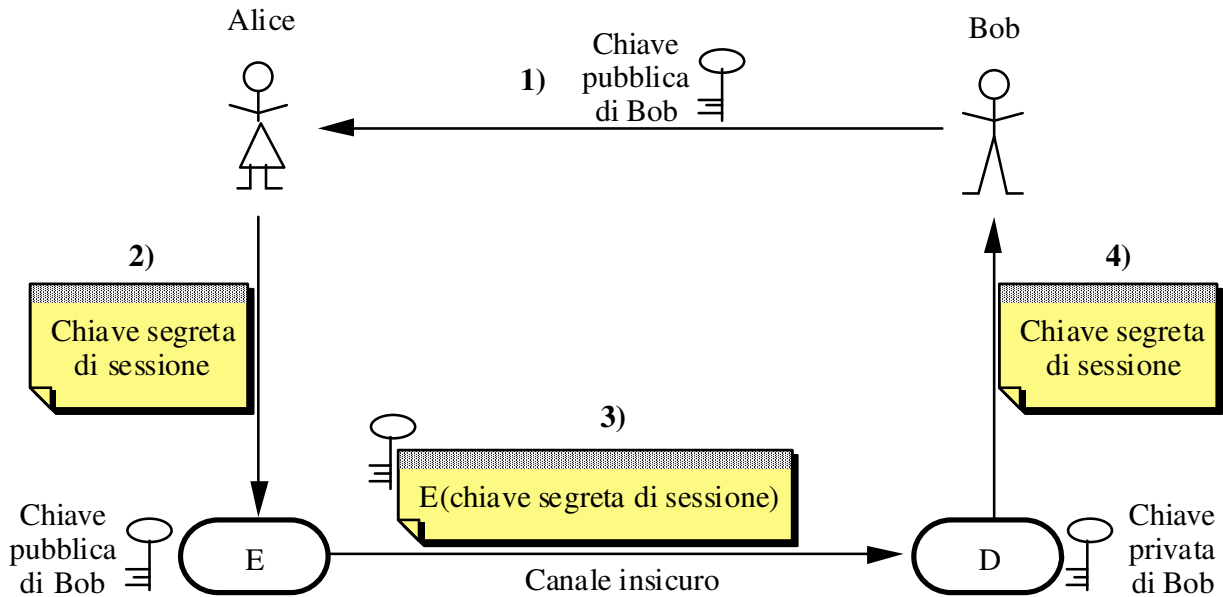


Figura 4-12: Determinazione della chiave segreta di sessione

Per evitare problemi derivanti dalla possibilità che Trudy esegua un *replay attack* (cioè invii duplicati di tutto ciò che intercetta) la chiave segreta di sessione dev'essere ogni volta diversa. Di norma la si calcola per mezzo di un generatore di numeri casuali, che deve essere progettato molto accuratamente (si veda in proposito il clamore suscitato da un bug contenuto in Netscape Navigator, riportato anche sul New York Times del 19/9/95).

4.3.3.2) Centro di distribuzione delle chiavi

Il protocollo precedente però ha un problema di fondo molto serio. Infatti, Trudy può riuscire a fare in modo che Alice riceva, al posto della chiave pubblica di Bob, quella di Trudy, e quindi interpersi nella successiva comunicazione e decifrare tutto (*man in the middle attack*).

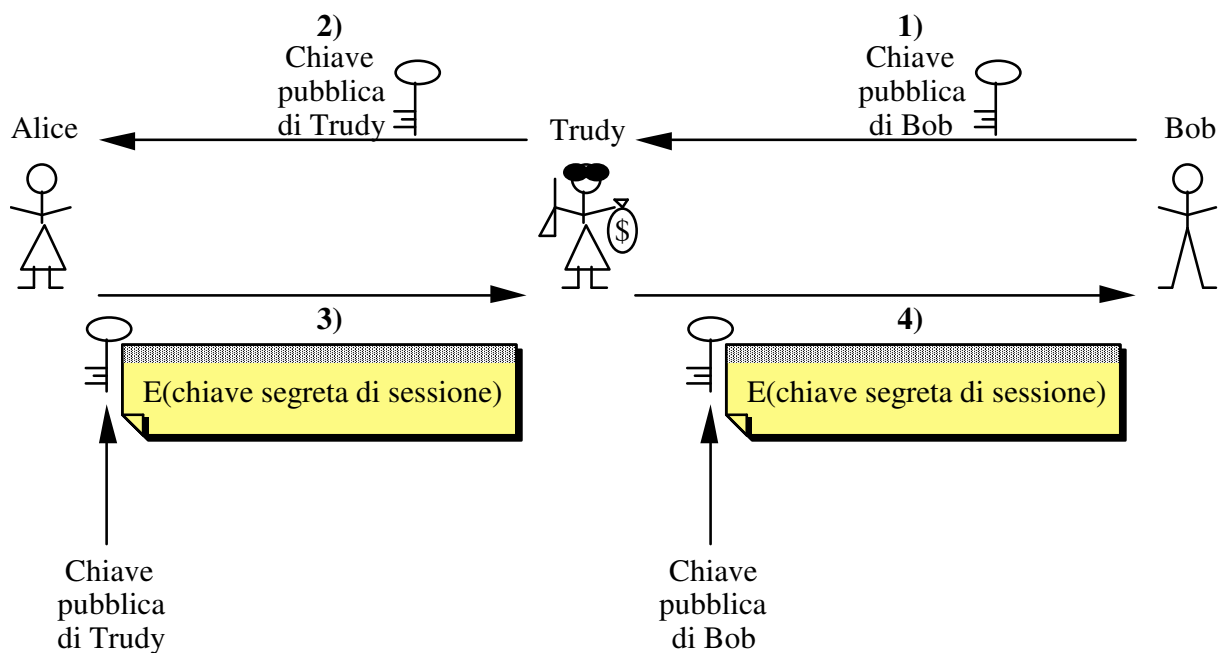


Figura 4-13: Trudy si interpone fra Alice e Bob

Per risolvere questo problema si introduce sulla scena una nuova entità, il *centro di distribuzione delle chiavi*.

Esso è un ente, di norma governativo o comunque dotato di credibilità internazionale, che:

- possiede adeguati meccanismi di sicurezza (anche fisica) per garantire i dati in proprio possesso;
- possiede una coppia di chiavi (pubblica e privata), e provvede periodicamente a confermare ufficialmente la propria chiave pubblica, ad esempio con la pubblicazione sui principali quotidiani;
- offre a chiunque la richieda una coppia di chiavi (pubblica e privata), che poi provvede a mantenere con sicurezza;
- crea, per ciascuno dei clienti registrati (cioè coloro ai quali ha rilasciato una coppia di chiavi) un *certificato digitale*, che in sostanza è un documento:

- contenente la chiave pubblica del cliente;
- contenente altre informazioni relative al cliente (nome, ecc.);
- cifrato con la chiave privata del centro (ossia, *firmato dal centro*).

Per questa ragione, il centro viene anche detto *Certificate Authority (CA)*.

In generale il software usato da Alice ha cablata al suo interno la chiave pubblica della CA, per cui è in grado di verificare la firma dei certificati provenienti dalla CA, e quindi di essere sicuro della loro autenticità e integrità.

Il protocollo visto precedentemente per stabilire la chiave segreta di sessione viene quindi modificato nel senso che la chiave pubblica di Bob viene consegnata ad Alice sotto forma di un certificato rilasciato a Bob da una CA; in questo modo Alice ha la garanzia che si tratta proprio della chiave di Bob e non di quella di Trudy.

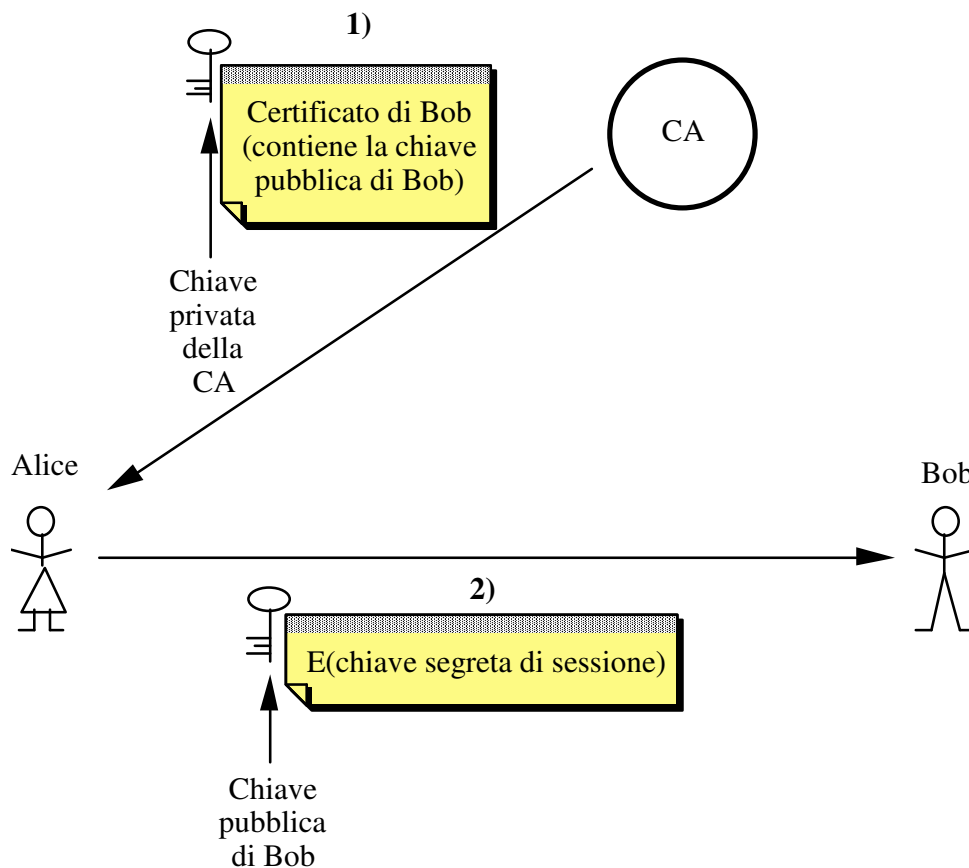


Figura 4-14: Ricorso ad una CA per avere la chiave pubblica di Bob

4.3.3.3) Secure Socket Layer e Secure-HTTP

I certificati prodotti da una CA possono essere conservati dove si desidera. Ad esempio, Bob può tenere una copia del proprio certificato, ed Alice può chiederlo direttamente a lui invece che alla CA.

Questo è precisamente quanto avviene sul Web quando si incontra un link gestito col metodo

`https://`

che viene gestito dal protocollo chiamato *Secure Socket Layer (SSL)*, introdotto da Netscape.

Esso protegge l'intero stream di dati che fluisce sulla connessione TCP, per cui può essere usato sia con HTTP che con FTP o TELNET.

Ad esempio, una form HTML che consente l'ordine di beni da acquistare fornendo il numero della propria carta di credito potrà essere riferita col link:

`https://www.server.com/order.html`

Quando l'utente del client decide di seguire tale link, si sviluppa questa catena di eventi:

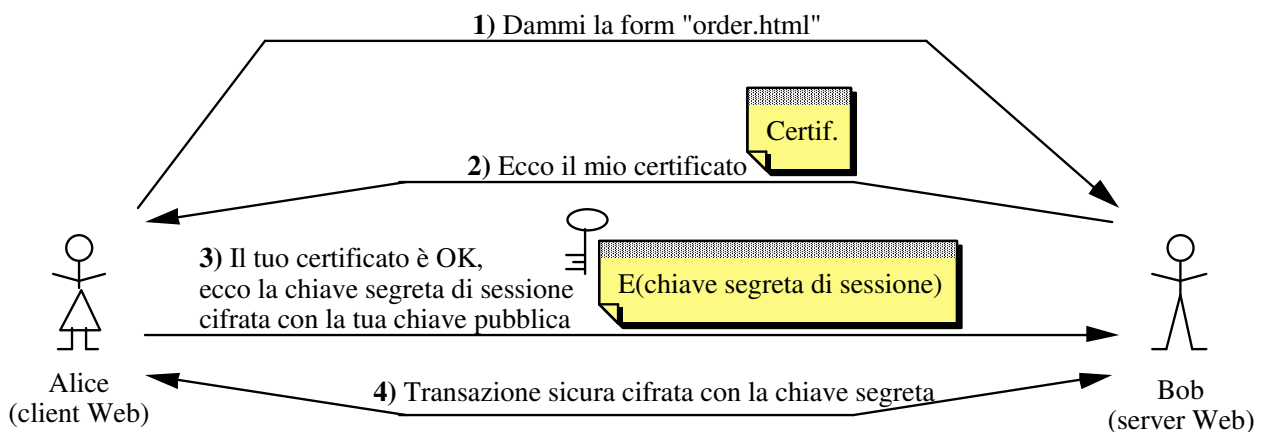


Figura 4-15: Funzionamento del protocollo SSL

Nell'protocollo SSL è previsto l'utilizzo di:

- RSA come algoritmo a chiave pubblica;
- DES oppure RC4 a 128 bit come algoritmo a chiave segreta;
- MD5 oppure SHA per la creazione dei digest.

Per via delle leggi americane, le versioni internazionali del Navigator usano RC4 a soli 40 bit per la cifratura a chiave segreta. Peraltro, una recentissima legge ha diminuito le restrizioni esistenti relativamente all'uso del DES a 56 bit, che verrà quindi incorporato anche nei prodotti destinati all'esportazione.

Infine va citato un altro protocollo sicuro per il Web, *Secure-HTTP (S-HTTP)*. Esso si basa su principi analoghi a SSL, ma si applica solo al traffico HTTP. Infatti, nella sostanza, si cifrano i singoli messaggi HTTP e non lo stream TCP sottostante.

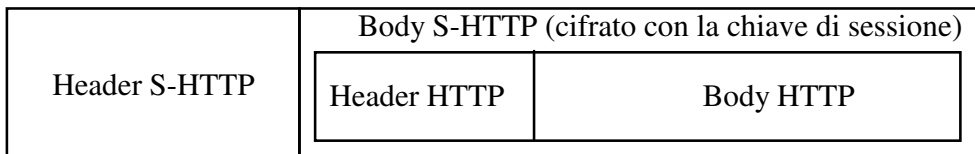


Figura 4-16: Formato di un messaggio S-HTTP

Nell'header S-HTTP, all'inizio, è contenuta la chiave di sessione cifrata con la chiave pubblica del server. Il certificato del server, invece, è incorporato dentro ogni pagina HTML sicura.

[Torna all'indice](#)