Appunti dei corsi di

Programmazione di Rete
Sistemi di elaborazione: Reti II

PROF. G. BONGIOVANNI

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 piu 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, include:
  - username (in chiaro);
  - nonce ricevuto dal server (in chiaro);
  - il Message Digest (calcolato con l'algoritmo ***MD5***) di:
    - nonce ricevuto dal server;
    - username dell'utente;
    - password dell'utente.

In questo modo la password dell'utente non viaggia 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) Network Address Translation

Per alleviare il crescente problema della scarsità di indirizzi IP, prodotto dalla progressiva ed inarrestabile crescita della rete Internet, varie tecniche sono state sviluppate.

Innanzitutto sono stati definiti dei range di ***indirizzi IP privati***, che possono essere riutilizzati su un numero illimitato di reti IP, purché nessuna di esse usi tali indirizzi per accedere alla rete Internet. Si dice che tali indirizzi ***non sono ruotabili su Internet***, cioè non possono comparire in pacchetti che viaggiano nella rete Internet, che è una ***rete pubblica***, ossia è una rete sulla quale vengono impiegati indirizzi *ufficialmente assegnati da enti presposti*. Tali indirizzi si dicono ***indirizzi pubblici***.

Naturalmente, per un'organizzazione che disponga di una ***rete privata*** (cioè dotata di indirizzi privati) il fatto di non poter accedere ad Internet dalle proprie postazioni costituisce un serio inconveniente. Per questo è stata introdotta una tecnica apposita, nota col nome di ***Network Address Translation*** (***NAT***).

Essa offre la possibilità di accedere alla rete pubblica anche agli host collocati su una rete privata e, come vedremo, fornisce anche un meccanismo rudimentale di protezione della rete privata da accessi indesiderati provenienti dalla rete pubblica.

Il NAT viene eseguito su un router che è collocato fra la rete privata e la rete pubblica ed effettua opportune trasformazioni degli indirizzi (sorgente e/o destinatario) nei pacchetti che transitano dalla rete privata verso quella pubblica e/o viceversa.

Vi sono diversi tipi di NAT, i più diffusi dei quali sono:

1. ***Outbound NAT*** (detto anche ***NAT unidirezionale*** o ***NAT tradizionale***);
2. ***NAT basato sulle porte*** (detto anche ***NAPT*** o ***PAT***).
3. ***Inbound NAT*** (detto anche ***NAT a due vie*** o ***NAT bidirezionale***);
4. ***NAT sovrapposto*** (detto anche ***NAT doppio***);

Vedremo le caratteristiche di tutti e quattro, ma prima è necessario introdurre un po' di terminologia.

Innanzitutto va detto che, rispetto ad un router che esegue il NAT, sono identificabili due lati: il ***lato inside*** è quello della rete privata, il ***lato outside*** è quello della rete pubblica.

Una prima classificazione degli indirizzi è relativa al *lato nel quale si trova l'host* cui l'indirizzo si riferisce. Gli indirizzi sono suddivisi in:

- ***indirizzi inside***: sono gli indirizzi relativi agli host che si trovano nel lato inside;
- ***indirizzi outside***: sono gli indirizzi relativi agli host che si trovano nel lato outside.

Una seconda classificazione degli indirizzi è relativa al *lato nel quale si trova il pacchetto* che li contiene. Essi sono suddivisi in:

- ***indirizzi locali***: sono gli indirizzi presenti in un pacchetto che si trova nel lato inside (sia che tale indirizzo si riferisca ad un host del lato inside che del lato outside);
- ***indirizzi globali***: sono gli indirizzi presenti in un pacchetto che si trova nel lato outside (sia che tale indirizzo si riferisca ad un host del lato inside che del lato outside).

Combinando queste due classificazioni, otteniamo quattro categorie di indirizzi:

- ***indirizzo inside local***: è l'indirizzo di un host che si trova nella rete privata, espresso utilizzando l'*indirizzo IP privato* che gli è stato assegnato; in altre parole, gli indirizzi inside local rappresentano la "visione" che la rete IP privata ha di se stessa; essi sono il modo normale, o nativo, con cui nella rete privata si fa riferimento agli host della rete privata stessa;
- ***indirizzo inside global***: è l'indirizzo di un host che si trova nella rete privata, espresso utilizzando un *indirizzo IP pubblico* che gli è (permanentemente o temporaneamente) assegnato; in altre parole, gli indirizzi inside global rappresentano la "visione" che la rete pubblica ha della rete privata; si noti che lo spazio di questi indirizzi, al limite, può essere costituito da un *singolo indirizzo* (come vedremo dopo);
- ***indirizzo outside global***: è l'indirizzo di un host che si trova nella rete pubblica, espresso semplicemente utilizzando l'*indirizzo IP pubblico* che gli è stato assegnato; essi sono il modo normale, o nativo, con cui nella rete pubblica si fa riferimento agli host della rete pubblica stessa;
- ***indirizzo outside local***: è l'indirizzo di un host che si trova nella rete pubblica, espresso secondo il modo con cui gli host della rete privata vi fanno riferimento; in

altre parole, gli indirizzi outside local rappresentano la "visione" che la rete privata ha della rete pubblica.

Quello che fa il NAT è sostanzialmente tradurre l'identità di risorse del lato inside o del lato outside mediante trasformazione di indirizzi da local a global e viceversa. Quali indirizzi vengono trasformati, e come, dipende dallo specifico tipo di NAT in uso. Ad esempio, nell'outbound NAT gli host del lato inside fanno di norma riferimento a quelli del lato outside utilizzando direttamente i loro indirizzi pubblici, per cui in questo caso gli indirizzi outside local coincidono con quelli outside global.

Concettualmente il NAT opera le trasformazioni di indirizzi secondo quanto illustrato nella figura seguente.

Nel lato inside gli indirizzi sono sempre local, nel lato outside sono sempre global; un pacchetto di ritorno che sia la risposta ad un pacchetto precedentemente inviato avrà ovviamente, rispetto ad esso, scambiati gli indirizzi di mittente e destinatario, e questo indipendentemente dal fatto che il primo pacchetto sia nato nel lato inside o in quello outside.

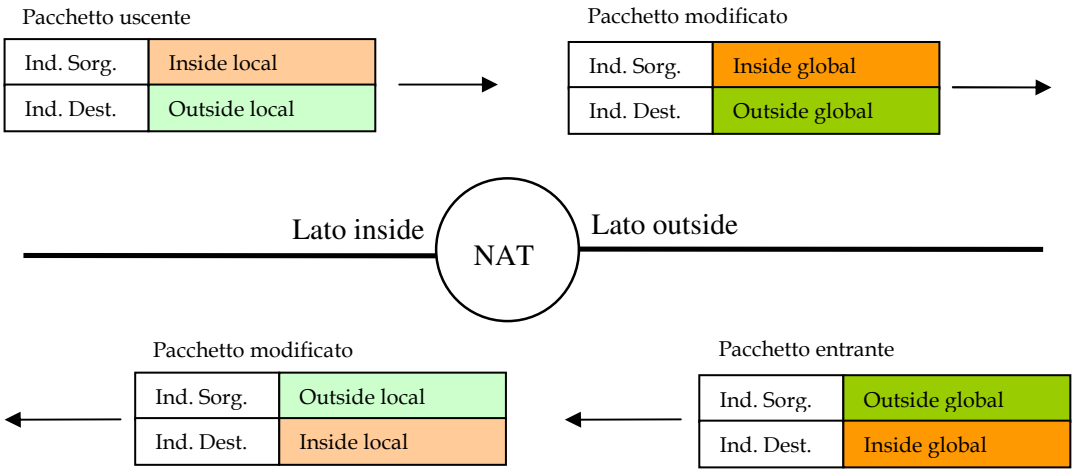

Figura 4-1: funzionamento concettuale del NAT

Ma come riesce il NAT ad operare le opportune trasformazioni di indirizzi sui pacchetti che, muovendosi in entrambe le direzioni, attraversano il confine fra lato inside e lato outside?

Un principio è comune a tutte le versioni di NAT: deve essere mantenuta una (o più d'una) ***tabella di traduzione*** (***translation table***) nella quale vengono gestite tutte le necessarie informazioni di corrispondenza fra un indirizzo originariamente presente nel pacchetto e quello che il NAT deve scrivere al suo posto prima di far proseguire il pacchetto.

Ciò si può applicare tanto all'indirizzo del mittente (ed in questo caso si parla di ***source NAT***) quando all'indirizzo del destinatario (***destination NAT***).

Ogni entrata della tabella rappresenta una specifica corrispondenza; inoltre, le entrate possono essere di due tipi differenti:

- ***entrate statiche***: esse rappresentano delle corrispondenze prefissate ed immutabili nel tempo; ogni volta che si incontra un indirizzo per il quale esiste una entrata statica, esso viene tradotto sempre nello stesso modo, secondo quanto specificato dall'entrata stessa;
- ***entrate dinamiche***: esse rappresentano delle corrispondenze che vengono stabilite automaticamente di volta in volta, secondo le necessità.

Iniziamo ora a vedere il reale funzionamento dei tipi di NAT che abbiamo citato in precedenza, cominciando dal più semplice (che è anche il più diffuso), ossia l'outbound NAT.

Outbound NAT

E' il NAT che molto spesso viene utilizzato su una Lan collegata ad Internet per mezzo di un router con capacità di NAT.

La situazione più semplice è la seguente:

- gli host della Lan hanno ciascuno un indirizzo privato;
- all'organizzazione che gestisce la Lan sono stati assegnati indirizzi pubblici in numero pari o superiore al numero di host della Lan (rilasceremo in seguito questa assunzione, ma essa per il momento è utile a fini di chiarezza);
- nella tabella di traduzione del NAT sono inserite tante entrate *statiche* quanti sono gli host, e ciascuna di tali entrate stabilisce la corrispondenza fra l'indirizzo inside local e l'indirizzo inside global di quell'host;
- gli host della rete interna usano indirizzi pubblici per fare riferimento agli host di Internet (non c'è quindi destination NAT sui pacchetti in uscita).

Ora, supponiamo che fra le altre vi sia l'entrata:

Inside local	*Inside global*
…	…
10.0.0.207	194.54.21.11
…	…

e che l'host 10.0.0.207 invii un pacchetto di richiesta ad un server esterno (diciamo all'indirizzo pubblico 204.51.16.12), dopodiché ci aspettiamo che arrivi un pacchetto di risposta inviato dal server che andrà consegnato all'host 10.0.0.207.

Il NAT compie le seguenti operazioni:

1. riceve il pacchetto che deve uscire, cui va applicato il *source NAT*:
   1.1. legge l'indirizzo sorgente (inside local), che è 10.0.0.207;
   1.2. consulta la sua tabella di traduzione, dove trova che a tale indirizzo corrisponde l'indirizzo (inside global) 194.54.21.11;
   1.3. sostituisce, nel pacchetto, l'indirizzo sorgente 10.0.0.207 con l'indirizzo 194.54.21.11;
   1.4. invia tale pacchetto sul lato outside;
2. quando arriva il pacchetto di risposta del server gli applica il *destination NAT*:
   2.1. legge l'indirizzo (inside global) di destinazione, che è 194.54.21.11;
   2.2. consulta la tabella di traduzione, dove trova che a tale indirizzo corrisponde l'indirizzo (inside local) 10.0.0.207;
   2.3. sostituisce, nel pacchetto, l'indirizzo di destinazione 194.54.21.11 con l'indirizzo 10.0.0.207;
   2.4. inoltra il pacchetto modificato sul lato inside.

Il tutto è illustrato nella figura seguente, che va letta in senso orario partendo da sinistra in alto.

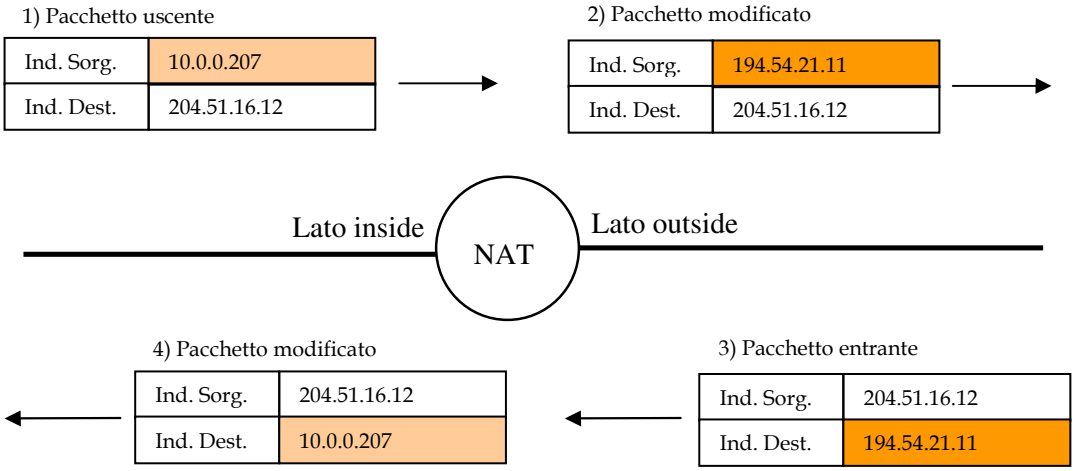

Figura 4-2: un semplice outbound NAT

L'assunzione che esista un numero di indirizzi pubblici assegnati alla Lan pari a quello degli indirizzi privati non è però realistica, perché di fatto vanificherebbe uno degli scopi principali del NAT, che è quello di ridurre il numero di indirizzi pubblici da impiegare.

Che succede quindi se alla Lan è assegnato un pool di indirizzi pubblici la cui cardinalità è inferiore al numero degli host che vi si trovano?

Una prima soluzione è costituita dalla *gestione dinamica*, e non più statica, delle entrate della tabella di traduzione:

1. quando arriva il pacchetto che deve uscire, il NAT seleziona dal pool *un indirizzo (inside global) che al momento non sia già in uso* e crea una nuova entrata nella tabella, dove memorizza la corrispondenza fra l'indirizzo (inside local) dell'host mittente e l'indirizzo (inside global) appena scelto nel pool;
2. opera la sostituzione nel pacchetto e lo inoltra sul lato outside;
3. quando arriva la risposta, consulta come nel caso precedente la tabella, dove però stavolta troverà l'entrata dinamica testé creata;
4. opera la trasformazione corrispondente dell'indirizzo di destinazione ed inoltra il pacchetto sul lato inside.

L'entrata dinamica viene cancellata quando la connessione Tcp fra l'host interno ed il server esterno termina, e di norma anche se trascorre un tempo prefissato (dell'ordine di alcuni minuti) senza che sia transitato alcun pacchetto riferibile a tale entrata.

Si noti che, già con questa semplice modifica di funzionamento del NAT, si ottiene una importante funzionalità di controllo degli accessi alla rete inside: infatti, se un pacchetto si presenta spontaneamente al NAT proveniente dal lato outside senza che un pacchetto di richiesta sia precedentemente uscito, il NAT non troverà nella sua tabella di traduzione alcuna entrata associata all'indirizzo (inside global) contenuto nel campo destinazione del pacchetto. In tal caso, invariabilmente, il pacchetto viene scartato.

Con questo schema, però, permane un problema non trascurabile: il numero di host che possono contemporaneamente accedere alla rete pubblica è limitato dal numero di indirizzi pubblici del pool. Se, ad esempio, nel pool vi sono 10 indirizzi che sono già stati tutti inseriti in altrettante entrate dinamiche, un ulteriore host che cercasse di collegarsi alla rete pubblica non riuscirebbe in tale intento, perché il NAT non potrà assegnargli un indirizzo inside global finché uno di quelli in uso non verrà rilasciato.

Per risolvere questo ulteriore problema si ricorre ad un altro tipo di NAT.

NAT basato sulle porte

In questo tipo di NAT si fa uso anche dei numeri di porta oltre che degli indirizzi IP. Per questa ragione si chiama anche NAPT (Network Address Port Translation) o PAT (Port Address Translation).

L'idea di base è semplice: nella tabella di traduzione, che continua ad essere gestita in modo dinamico, si tiene traccia della coppia (indirizzo IP, numero di porta) sia per gli indirizzi originari che per quelli trasformati. Si parla in questo caso di ***porta sorgente*** (quella relativa all'indirizzo inside local) e ***porta nattata*** (quella relativa all'indirizzo inside global).

Dato che i numeri di porta disponibili sono decine di migliaia, diviene possibile associare molti indirizzi inside local con lo stesso indirizzo inside global, differenziando tali associazioni semplicemente per mezzo del numero di porta.

Ciò permette di avere al limite *un solo indirizzo inside global* nel pool, e di gestire comunque un numero elevato di accessi ad Internet contemporanei.

Ad esempio, supponiamo che l'unico indirizzo inside global disponibile sia 194.54.21.100. Esso apparirà in tutte le entrate dinamiche della tabella di traduzione, ma in ogni entrata esso sarà accompagnato da un diverso numero di porta.

Immaginiamo che in un determinato momento la tabella contenga, fra le altre, le seguenti entrate dinamiche:

Inside local	*Inside global*
…	…
10.0.0.10:1030	194.54.21.100:900
10.0.0.10:1040	194.54.21.100:901
10.0.0.10:1050	194.54.21.100:902
10.0.0.11:600	194.54.21.100:903
10.0.0.11:610	194.54.21.100:904
10.0.0.14:500	194.54.21.100:905
10.0.0.20:770	194.54.21.100:906
10.0.0.22:430	194.54.21.100:907
…	…

Questa situazione corrisponde al fatto che:

- l'host 10.0.0.10 ha tre connessioni attive;
- l'host 10.0.0.11 ha due connessioni attive;
- gli altri host hanno una connessione attiva ciascuno.

All'arrivo di un pacchetto dal lato outside, il NAT come nel caso precedente consulterà la tabella, e se troverà un'entrata relativa all'indirizzo (inside global) e al numero di porta che appaiono come indirizzi di destinazione rispettivamente nell'header IP e nell'header TCP del pacchetto in arrivo, opererà la corrispondente sostituzione ed inoltrerà il pacchetto sul lato inside.

La figura seguente rappresenta questo scenario, ed è relativa all'entrata che appare per ultima nella tabella precedente.

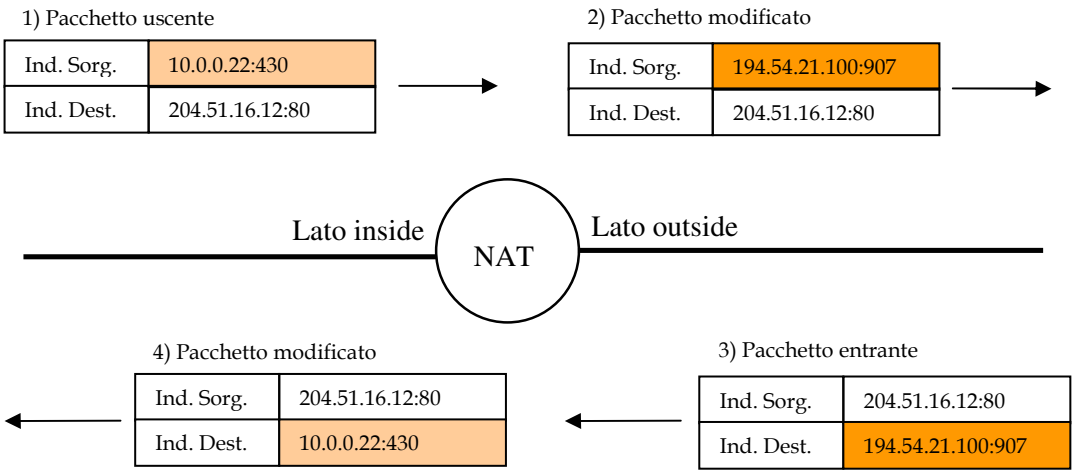

Figura 4-3: NAT basato sulle porte

Ovviamente il discorso continua a valere se il pool di indirizzi inside global ne contiene più di uno. In tal caso, semplicemente, il NAT potrà scegliere in ogni momento un indirizzo libero del pool, ma continuerà comunque ad usarlo assieme ad un numero di porta nattata.

Ancora, come nel caso precedente, qualunque pacchetto si presenti al NAT proveniente dal lato outside e per il quale non si trovi un'entrata nella tabella viene scartato.

Inbound NAT

Peraltro, può sorgere l'esigenza di rendere accessibile un servizio offerto sul lato inside (ad esempio un server WWW) anche alle macchine del lato outside, cosa ovviamente impossibile se non vengono mai trasferiti sul lato inside pacchetti originati spontaneamente sul lato outside.

Questa esigenza affianca (ma non sostituisce) quella di consentire agli host del lato inside di accedere ad Internet. Per tale ragione il NAT che la soddisfa (l'inbound NAT) si chiama anche ***a due vie***, perché deve gestire:

- l'accesso a Internet da parte degli host del lato inside;
- l'accesso da parte degli host del lato outside a servizi offerti nel lato inside.

Il secondo problema è leggermente più complicato del primo, perché vi è una sorta di asimmetria nelle conoscenze che i due lati hanno ciascuno dell'altro:

- gli host del lato inside conoscono gli indirizzi delle apprecchiature sul lato outside, dato che essi sono pubblici;
- gli host del lato outside invece non hanno alcuna conoscenza degli indirizzi del lato inside; oltretutto, anche se conoscessero tali indirizzi non potrebbero usarli dato che essi non sono ruotabili su Internet.

Per accedere a un qualunque servizio offerto sul lato inside, un host del lato outside quindi non può che usare un indirizzo inside global, e sarà compito del NAT preoccuparsi di effettuare le opportune trasformazioni sul pacchetto in transito. Perché ciò sia possibile, dovrà essere inserita nella tabella del NAT un'*entrata statica* che associ un indirizzo inside global ed un numero di porta all'indirizzo inside local ed al numero di porta sui quali viene effettivamente offerto il servizio.

Ad esempio, la seguente entrata statica:

Inside local	*Inside global*
…	…
10.0.0.10:5000	194.54.21.100:80
…	…

consente agli host della rete Internet di accedere ad un server WWW che è in esecuzione sull'host di indirizzo privato 10.0.0.10 ed accetta connessioni TCP sulla porta 5000.

Il funzionamento dell'inbound NAT è simile al caso precedente, ed è illustrato nella figura che segue. Si noti che l'ordine degli eventi cambia (la figura va letta in senso antiorario, partendo da destra in alto).

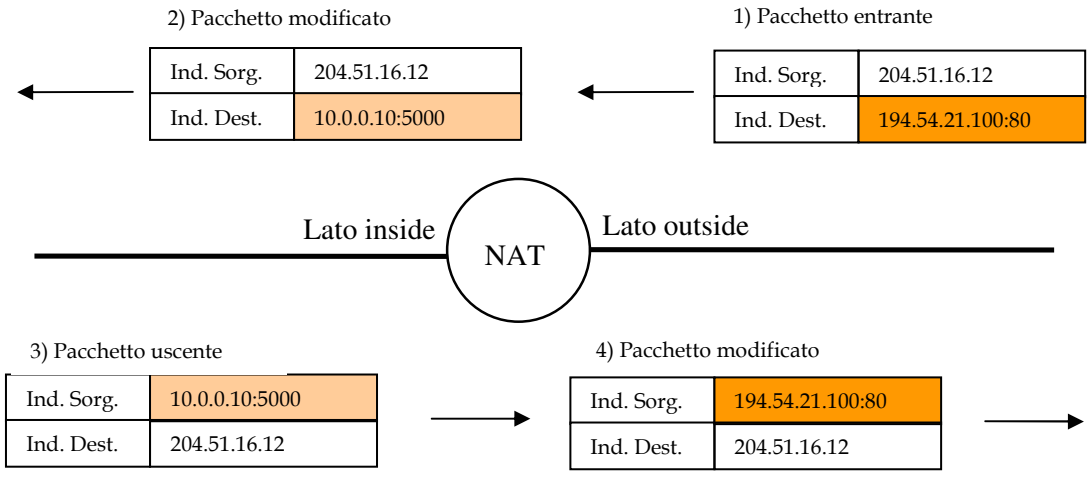

Figura 4-4: inbound NAT

Una variante di questo tipo di NAT si ha quando esiste sul lato inside un server DNS in grado di cooperare col NAT. In questo caso i passi sono i seguenti:

1. arriva dall'esterno una richiesta DNS per il nome del server sul lato inside che offre il servizio, richiesta che il NAT passa al server DNS della rete interna;
2. il server DNS risponde fornendo l'indirizzo *inside local* del server;
3. tale risposta contenente l'indirizzo inside local viene intercettata dal NAT, che:
   3.1. inserisce nella sua tabella un'entrata dinamica che associa l'indirizzo inside local con un opportuno indirizzo inside global;
   3.2. costruisce una nuova risposta DNS, che invia al richiedente, nella quale appare non più l'indirizzo inside local ma l'indirizzo inside global stabilito nel passo precedente.
4. All'arrivo di un pacchetto IP che contenga come indirizzo di destinazione tale indirizzo inside global, si applica quanto visto nella figura precedente: la sola differenza è che l'entrata pertinente nella tabella è ora quella dinamica e non una statica.

NAT sovrapposto

Un'ultima considerazione è la seguente. In tutti gli schemi che abbiamo visto finora si effettua:

- un source NAT sui pacchetti che transitano dal lato inside al lato outside;
- un destination NAT sui pacchetti che transitano dal lato outside al lato inside.

L'unica differenza fra gli schemi che abbiamo visto è relativa all'uso di entrate statiche o dinamiche, ed alla sequenza della fasi:

- nell'outbound NAT prima si fa source NAT sui pacchetti uscenti dal lato inside, poi il destination NAT su quelli entranti nel lato inside;
- nell'inbound NAT prima si fa destination NAT sui pacchetti entranti nel lato inside, poi il source NAT su quelli uscenti dal lato inside;

Viceversa, non abbiamo finora considerato l'idea di applicare le operazioni complementari, e cioè un destination NAT ai pacchetti uscenti dal lato inside e diretti al lato outside e/o un source NAT a quelli provenienti dal lato outside e diretti al lato inside.

Dato che il NAT è in grado di effettuare anche tali trasformazioni, sorge spontaneo chiedersi quale situazione le possa richiedere.

Fin qui abbiamo sempre supposto che gli host del lato inside abbiano completa conoscenza degli indirizzi pubblici presenti sulla rete Internet, e quindi possano indirizzare direttamente qualunque risorsa disponibile sul lato outside.

Questo però non è sempre detto. In particolare, un'organizzazione può avere la necessità di connettere fra loro varie reti private (ciascuna collocata dietro un apposito NAT), o magari vuole limitare l'accesso ad Intenet da parte degli host del lato inside, rendendolo possibile solo verso un sottoinsieme di server considerati "di fiducia".

Una possibilità per risolvere entrambe queste problematiche è di configurare il NAT in modo tale che, *in aggiunta* ai controlli visti precedentemente:

- mantenga in un'apposita tabella (che chiameremo tabella B, per distinguerla dalla tabella vista finora che d'ora in poi chiameremo tabella A) una serie di *entrate statiche* che associano ciascuna un indirizzo *outside local* con un corrispondente indirizzo *outside global*;
- venga applicato a tutti i pacchetti uscenti dal lato inside (oltre al solito outbound NAT) anche un destination NAT: se nella tabella B c'è l'entrata corrispondente all'indirizzo (outside local) contenuto nel pacchetto l'indirizzo viene trasformato nel corrispondente indirizzo outside global ed il pacchetto passa, altrimenti esso viene scartato.
- venga applicato ai pacchetti di ritorno provenienti dal lato outside anche un source NAT sulla base dell'opportuna entrata statica della tabella B, se essa è presente.

In questo modo, gli host non possono spedire con successo all'esterno pacchetti contenenti indirizzi outside global (che vengono scartati dal NAT), ma solo pacchetti contenenti indirizzi outside local (che vengono accettati dal NAT e trasformati in outside global): così si consentono accessi a Internet solo verso i server di fiducia.

Questo stesso meccanismo permette di realizzare anche la comunicazione fra due reti private R1 ed R2, ciascuna collegata alla rete pubblica con un NAT: nella rete R1 basterà inserire (nella tabella B) almeno un'entrata che stabilisca la corrispondenza fra uno specifico indirizzo outside local e un indirizzo outside global che corrisponda al router NAT della rete R2. Una simmetrica operazione verrà effettuata nella rete R2.

Ad esempio, supponiamo che la tabella A contenga l'entrata dinamica:

Inside local	*Inside global*
…	…
10.0.0.22:430	194.54.21.100:907
…	…

e che la tabella B contenga l'entrata statica:

Outside local	*Outside global*
…	…
10.1.1.100	204.51.16.12
…	…

L'applicazione (in aggiunta al solito outbound NAT) di un destination NAT ai pacchetti uscenti e di un corrispondente source NAT a quelli di ritorno è illustrata dalla figura seguente. Come si vede entrambi gli indirizzi (sorgente e destinazione) vengono trasformati, ragion per cui questo tipo di NAT si chiama sovrapposto (o doppio).

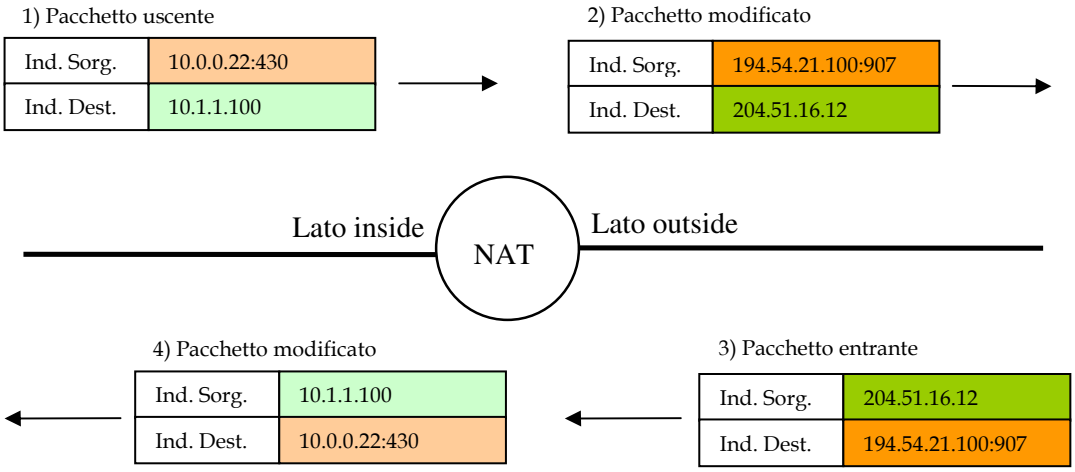

Figura 4-5: NAT sovrapposto

Si noti che in questo scenario, se viene gestito anche un inbound NAT, si presenta un problema: un pacchetto proveniente spontaneamente dal lato outside per l'accesso ad un servizio del lato inside non troverà ovviamente un'entrata statica appropriata nella tabella B (l'indirizzo outside global del mittente può essere un qualunque indirizzo pubblico).

Quindi, perché la risposta del server interno possa successivamente uscire, si dovrà:

- all'arrivo del pacchetto di richiesta creare un'entrata *dinamica* nella tabella B che associ l'indirizzo outside global dell'host mittente (quello che vuole accedere al servizio) con un indirizzo outside local;
- effettuare la corrispondente operazione di source NAT sul pacchetto prima di inoltrarlo sul lato inside;
- alla successiva risposta del server interno (che arriva dal lato inside) applicare il destination NAT sulla base dell'entrata dinamica testé inserita nella tabella B, sostituendo l'indirizzo outside local di destinazione col corrispondente indirizzo outside global, prima di inoltrarlo sul lato outside.

.

4.1.4) Firewall

In molte situazioni una rete aziendale privata connessa con una rete esterna pubblica (Internet) ha bisogno di essere protetta perché, ad esempio, può sorgere la necessità di:

- proteggere le informazioni riservate (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 in parte ottenere, come abbiamo visto, col NAT. Una apparecchiatura più specializzata e sofisticata da questo punto di vista è il ***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, ossia al ***filtraggio*** del traffico. Solamente quello che rispetta determinate caratteristiche viene lasciato transitare da una parte all'altra.

Il firewall (che spesso implementa anche funzioni di NAT) 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 dove può essere controllato.

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;
- contenuti di livello applicativo trasportati.

Categorie di firewall

Esistono due categorie di firewall: quelli ***packet filter*** e quelli ***stateful***. La differenza è la seguente:

- i firewall packet filter non mantengono alcuna memoria del traffico che li ha attraversati nel recente passato, e filtrano i pacchetti singolarmente, cioè applicando a ciascuno di essi indipendentemente i ***criteri*** (o ***regole***) ***di filtraggio*** impostate nella configurazione;
- i firewall stateful, invece, mantengono in memoria la storia del traffico recente, ossia sono in grado di ricondurre ogni pacchetto che li raggiunge ad un flusso di comunicazione (se esso è in atto) di cui quel pacchetto fa parte.

Ovviamente i primi sono più semplici da implementare dei secondi, ma soffrono di varie limitazioni dovute proprio al fatto che ogni pacchetto è valutato di per se e non viene ricondotto ad un flusso di comunicazione in atto.

Si considerino le seguenti situazioni, a titolo di esempio.

1. Si desidera consentire il transito di un pacchetto proveniente dalla rete esterna e diretto a quella interna solo se esso è relativo ad una comunicazione che è stata avviata da un host della rete interna; viceversa, non si accettano pacchetti provenienti "spontaneamente" dall'esterno:
   - se il pacchetto contiene un segmento TCP entrambi i tipi di firewall possono facilmente gestire questo requisito: infatti, dato che una connessione TCP si avvia con un segmento nel quale i bit ***syn*** e ***ack*** sono rispettivamente a 1 e a 0, basterà definire una regola per la quale tutti i pacchetti di tale tipo provenienti dall'esterno devono essere scartati;
   - se viceversa il pacchetto contiene un segmento UDP, protocollo per il quale non vi è una fase di attivazione della connessione, i firewall stateful possono comunque risolvere la situazione verificando se, in precedenza, un segmento UDP con gli stessi indirizzi (ovviamente scambiati) di quelli contenuti nel pacchetto da filtrare è transitato dalla rete interna a quella esterna. Per i firewall packet filter, invece, questa situazione non è gestibile, in quanto essi non possiedono tale conoscenza. Di conseguenza, è pratica comune per questi tipi di firewall non consentire in alcun caso il passaggio di traffico UDP.

2. Si desidera riconoscere se è in atto una ***sequenza di attacco***, ossia una successione di pacchetti provenienti dall'esterno con caratteristiche (che sono conosciute e quindi identificabili) tali da rappresentare una minaccia per le risorse della rete interna:
   - i firewall stateful, se opportunamente configurati, riconoscono tali sequenze man mano che esse si presentano, e provvedono quindi a bloccare ulteriori pacchetti in ingresso non appena si rendono conto che essi appartengono ad una di tali sequenze;
   - i firewall packet filter, invece, poiché ispezionano i pacchetti singolarmente, non sono in grado di ricondurli a possibili sequenze d'attacco.

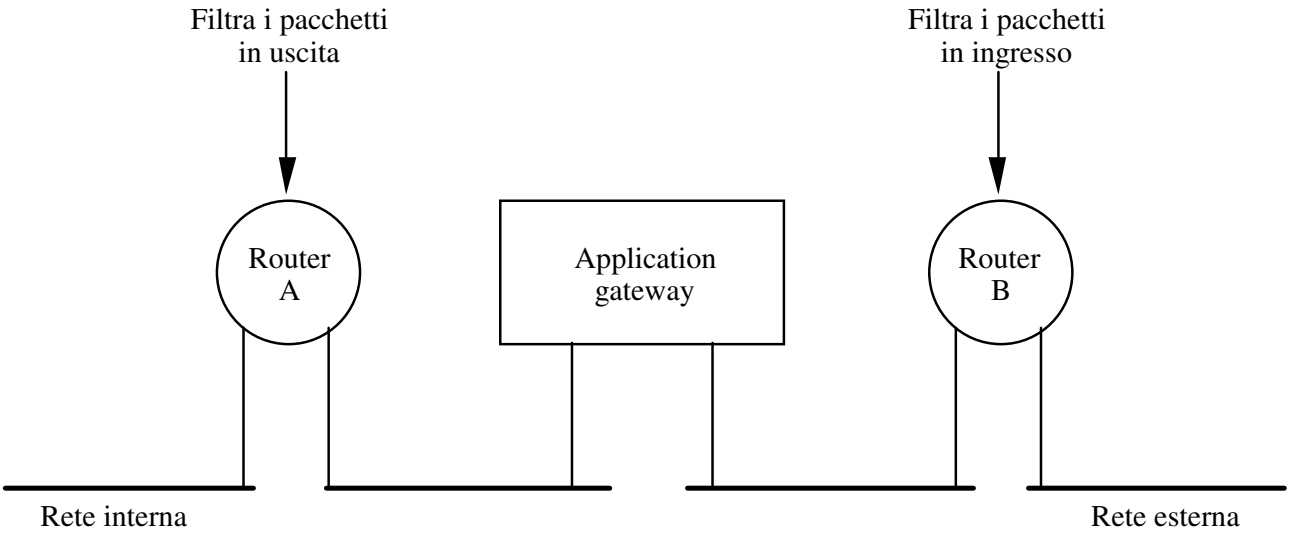

Figura 4-6: Tipica configurazione di un firewall

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

Server proxy

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 serventi 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.

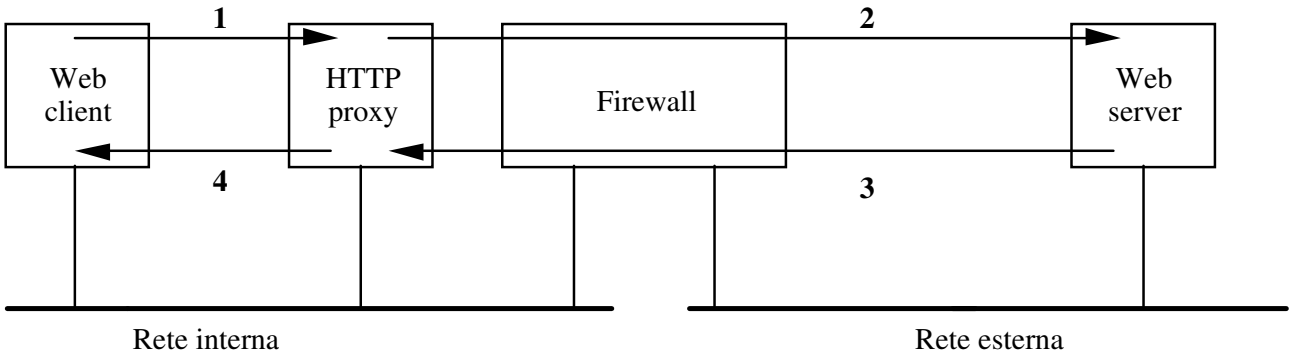

Figura 4-7: Uso di un HTTP proxy

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.

Zona demilitarizzata

Infine, c'è un'ultimo problema importante in relazione ai firewall: se un'azienda vuole rendere disponibili all'esterno dei servizi o delle informazioni (ad esempio consentendo l'accesso dall'esterno ad un server Web), dovrà far si che il firewall consenta il passaggio di traffico originato dalla rete esterna e diretto a tale server Web.

Questo però apre una falla rilevante nella sicurezza, poiché tale server diviene bersaglio di potenziali attacchi dato che è accessibile dall'esterno, e quindi potrebbe venire compromesso nelle sue funzionalità.

Già di per se ciò è molto poco desiderabile, ma oltretutto espone ad un pericolo ancora maggiore: una volta che un attacco a tale server sia portato a compimento con successo, l'attaccante potrebbe guadagnare il controllo della macchina, e da essa avviare ulteriori attacchi a qualunque risorsa della rete interna. Contro tali attacchi il firewall non ha alcuna possibilità di intervento, perché essi si sviluppano integralmente sulla rete interna e quindi il relativo traffico non attraversa il firewall stesso.

Per ovviare a tale inconveniente si ricorre ad una terza zona di rete, detta ***Zona Demilitarizzata*** (***DMZ***, ***DeMilitarized Zone***), sulla quale vengono collocati tutti i server accessibili dalla rete esterna.

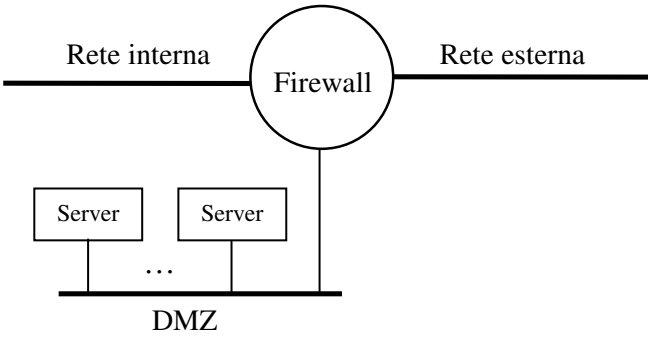

Figura 4-8: la DMZ

La DMZ ha un livello di sicurezza intermedio, più basso di quello della rete interna (ad esempio, il traffico originato dall'esterno viene lasciato passare se è diretto alla DMZ ma viene bloccato se è diretto verso la rete interna). Inoltre, il traffico proveniente dalla DMZ e diretto alla rete interna deve comunque transitare attraverso il firewall, dove può quindi essere sottoposto a controlli. Di conseguenza, un eventuale attacco che guadagni il controllo di uno o più server della DMZ non potrà successivamente arrecare danni alla rete interna.

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 decifratura;
- 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.

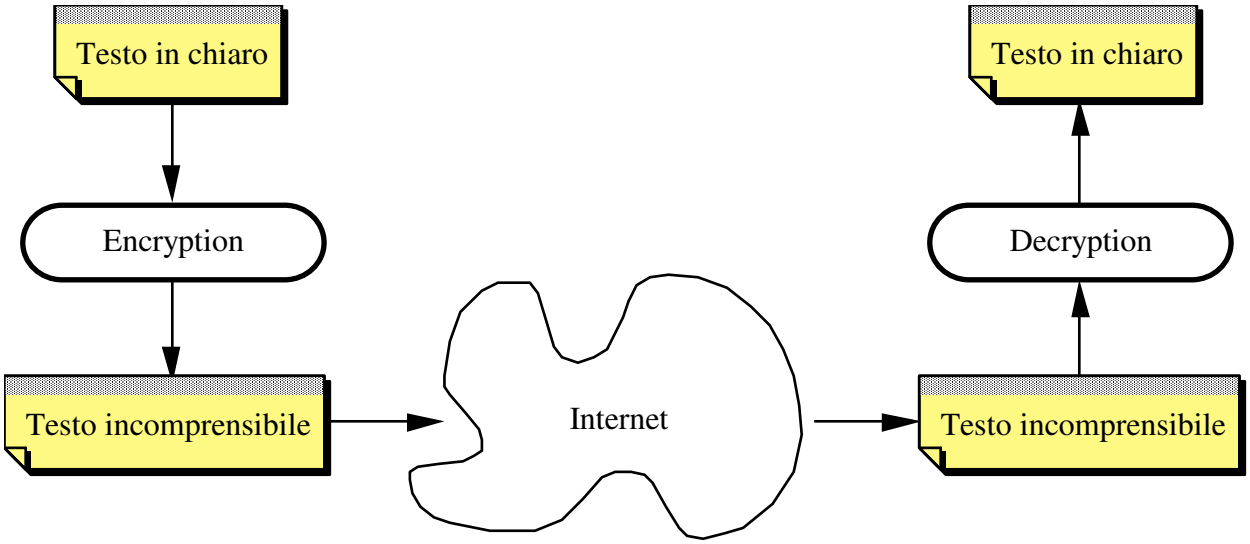

Figura 4-9: Cifratura e decifratura

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:

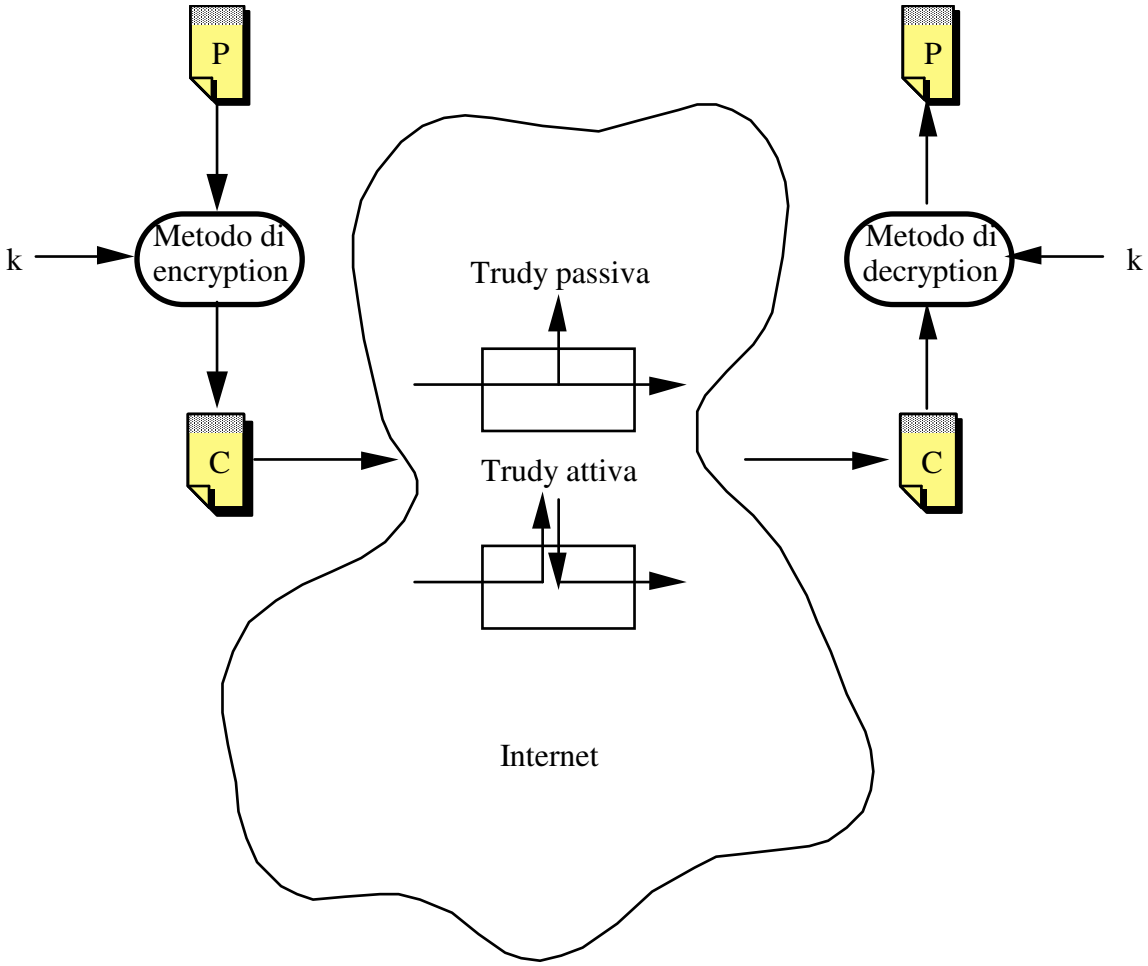

Figura 4-10: Cifratura e decifratura basate su chiave

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 ***cryptoanalisi*** (***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 cifrature 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 rispedire nuovamente un messaggio valido. Altrimenti, ogni volta che Trudy intercetta un messaggio puo inviarlo nuovamente per i propri scopi.

4.3.1.1) Cenni storici

La crittografia è una disciplina antica, nata inizialmente per garantire la segretezza delle comunicazioni in campo militare. Il primo esempio di codice crittografico noto è il ***codice di Cesare***, utilizzato da Giulio Cesare per le comunicazioni con i suoi generali. Il codice di Cesare è definito nel seguente modo:
- Sostituire ogni lettera del testo in chiaro con quella che si trova, nell'ordinamento alfabetico, tre posizioni dopo (la A viene sostituita dalla D, la B con la E, e così via). L'alfabeto si considera chiuso circolarmente, cioè la lettera Z viene sostituita con la lettera C.

Si noti che utilizzando questo codice uno stesso testo in chiaro produce, tutte le volte che viene cifrato, lo stesso testo cifrato.

Un'evoluzione del codice di Cesare è il ***codice di Cesare generalizzato***, definito come segue:
- Sostituire ogni lettera del testo in chiaro con quella che si trova, nell'ordinamento alfabetico, k posizioni dopo, considerando anche in questo caso l'alfabeto chiuso circolarmente. Il valore di k può essere variato di volta in volta, ad ogni operazione di cifratura, e funge da chiave di cifratura. Ovviamente, la stessa chiave deve essere utilizzato nella cifratura e nella corrispondente decifratura di un messaggio.

Il lavoro per il criptoanalista è comunque ancora molto semplice: egli dovrà provare semplicemente ad usare tutti i possibili valori della chiave, che sono solo 21, per ottenere il testo in chiaro (che riconoscerà subito, non appena il testo decifrato sarà effettivamente intelligibile).

Un ulteriore miglioramento è costituito dal cosiddetto ***cifrario monoalfabetico***: anch'esso sostituisce ogni lettera del testo in chiaro con un'altra, ma in modo meno regolare. Infatti qualsiasi lettera può essere sostituita da qualunque altra, purché in una operazione di cifratura ciascuna lettera, ogni volta che appare, sia sempre sostituita dalla stessa lettera. La chiave di cifratura, in questo caso, è molto più articolata che nel caso precedente perché è costituita dalla sequenza che definisce la sostituzione da applicare per ciascuna lettera del testo in chiaro. Ad esempio:
Lettera del testo in chiaro: `a b c d e f g h i l m n o p q r s t u v z`
Lettera del testo cifrato: `m n b v c z a s d f g h l p o i u r t e q`
In questo caso un attacco di forza bruta è molto più oneroso, perché ci sono 21! (= 5 x 10^{19}) possibili valori della chiave da applicare. E' possibile però un altro tipo di attacco contro un codice di questo tipo: se il criptoanalista conosce la natura del messaggio originale (ad esempio sa che si tratta di un testo scritto in italiano) può sfruttare le regolarità della lingua italiana. Può misurare le frequenze di occorrenza delle varie lettere nel testo cifrato e, confrontandole con quelle note per la lingua italiana, dedurre quali siano le corrispondenti lettere del testo in chiaro.

Circa quattro secoli or sono fu introdotto un nuovo concetto, quello di ***cifrario polialfabetico***, detto anche ***cifrario di Vigenère***: l'idea di fondo è di utilizzare più cifrari monoalfabetici in una singola operazione di cifratura e nella corrispondente decifratura. L'innovazione principale è che una stessa lettera, che appaia in differenti posizioni del testo in chiaro, può essere sostituita di volta in volta da lettere diverse.

Un esempio di semplice cifrario polialfabetico è il seguente:
- Si scelgono tre diversi cifrari monoalfabetici C1, C2 e C3;
- Si definisce una sequenza di applicazione dei cifrari, ad esempio C3 C1 C2 C2 C1;
- Le lettere del testo in chiaro si trasformano a gruppi di 5, applicando alla prima lettera di ogni gruppo il terzo dei tre cifrari scelti (C3), alla seconda lettera il primo cifrario (C1), alla terza ed alla quarta lettera il secondo cifrario (C2), alla quinta lettera il primo cifrario (C1).

Rompere un cifrario polialfabetico è sicuramente molto più difficile, perché le regolarità del linguaggio sottostante vengono offuscate dal fatto che una stessa lettera viene cifrata con lettere diverse di volta in volta. Però se il criptoanalista riesce a dedurre la lunghezza della sequenza di applicazione dei cifrari, ossia il ***periodo*** del codice (5 nel nostro esempio) può sfruttare le regolarità del linguaggio esaminando le lettere che distano 5 posizioni fra loro, dato che esse sono state cifrate tutte con lo stesso codice monoalfabetico.

Si noti che in tutti i codici visti finora si applica al testo in chiaro una sola fase di trasformazione per produrre il corrispondente testo cifrato. Una scoperta che ha permesso lo sviluppo della moderna crittografia fu che applicare diverse fasi di cifratura (anziché una sola) ad uno stesso testo in chiaro, per ottenere il testo cifrato, rende enormemente più difficile il tentativo di ricostruire il testo in chiaro senza conoscere la chiave di cifratura e decifratura.

Un esempio di applicazione di questa idea viene dalle ***macchine a rotori***, largamente utilizzate durante la seconda guerra mondiale per cifrare le comunicazioni militari, fra cui la celebre Enigma, utilizzata dall'esercito e dalla marina tedesca.

Figura 4-11: Un esemplare di Enigma a tre rotori

Un rotore è un disco con 21 ingressi e 21 uscite, e dotato di un cablaggio interno che collega ogni ingresso con una uscita secondo una permutazione fissata. Di fatto un singolo rotore, se rimane immobile, realizza un cifrario monoalfabetico. Una ipotetica macchina a rotori composta di un solo rotore funziona in questo modo: ad ogni pressione di un tasto da parte dell'operatore, la lettera corrispondente al tasto premuto, che è elettricamente collegato ad uno dei 21 ingressi del rotore, viene codificata con la lettera corrispondente all'uscita cui quell'ingresso è collegato. Contestualmente il cilindro ruota di una posizione, definendo quindi un nuovo cifrario monoalfabetico, che sarà usato per cifrare la prossima lettera, e così via. Dunque, una macchina con un singolo rotore realizza un cifrario polialfabetico con periodo 21, che di per se non rappresenta, come abbiamo visto, un ostacolo insormontabile. Ma la vera potenza di queste macchine risiede nel fatto che venivano usati vari rotori, ciascuno cablato con una differente permutazione fra ingressi e uscite. Le uscite di ogni rotore erano collegate agli ingressi del rotore successivo. Inoltre, per ogni giro completo del primo rotore il secondo rotore avanzava di uno scatto, per ogni giro completo del secondo rotore il terzo rotore avanzava di uno scatto, e così via. Una macchina siffatta e munita di tre rotori definisce un codice polialfabetico con periodo 21 x 21 x 21 = 9261; con quattro rotori il periodo sale a 194481, e con cinque rotori a 4084101.

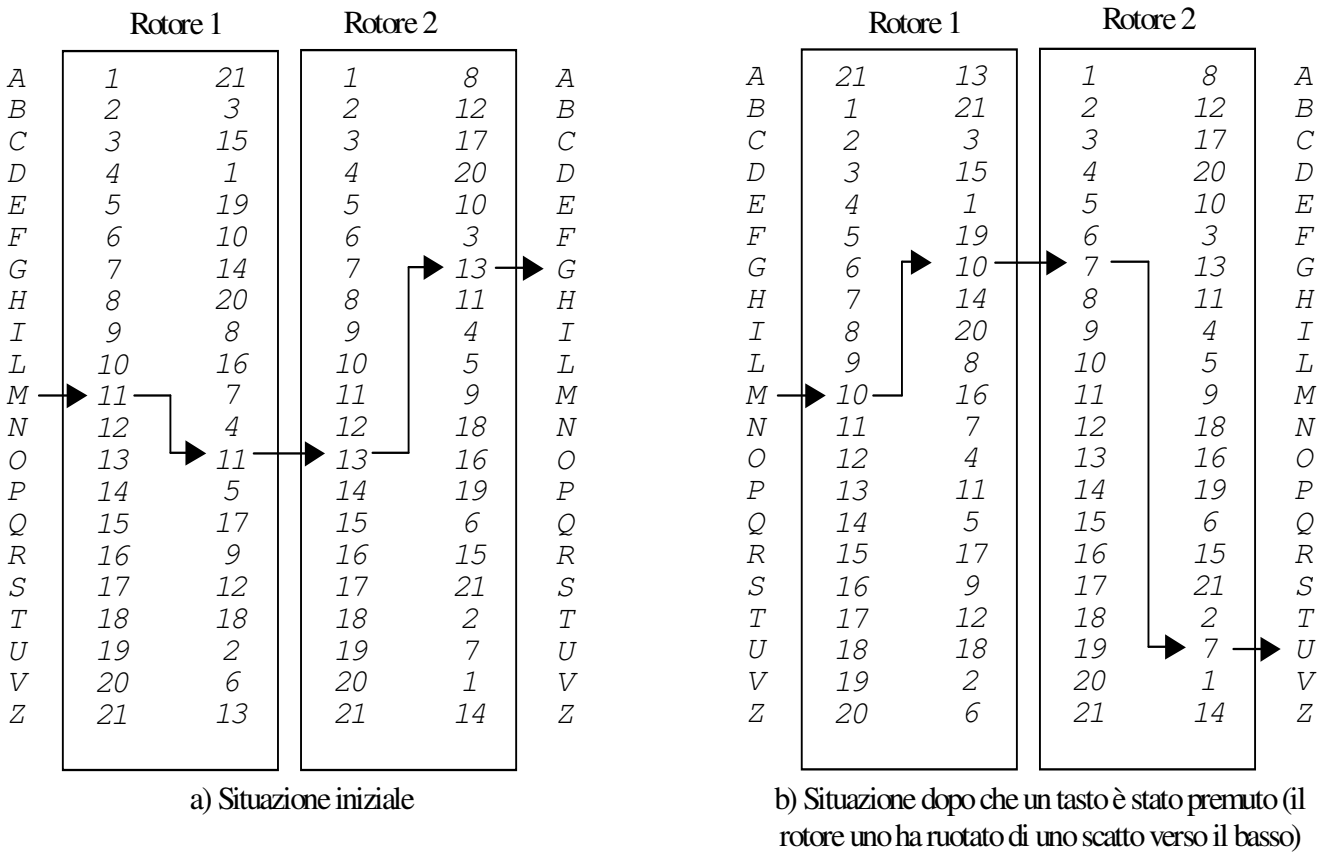

Figura 4-12: Macchina a due rotori

Periodi di tale lunghezza rendono inapplicabile per il criptoanalista lo studio delle regolarità del linguaggio, a meno che egli non disponga di messaggi di dimensioni veramente enormi.

L'importanza di queste macchine risiede nel fatto che esse realizzavano, con tecnologie elettromeccaniche, idee che sono alla base degli attuali cifrari progettati per essere eseguiti da elaboratori elettronici.

4.3.1.2) 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.

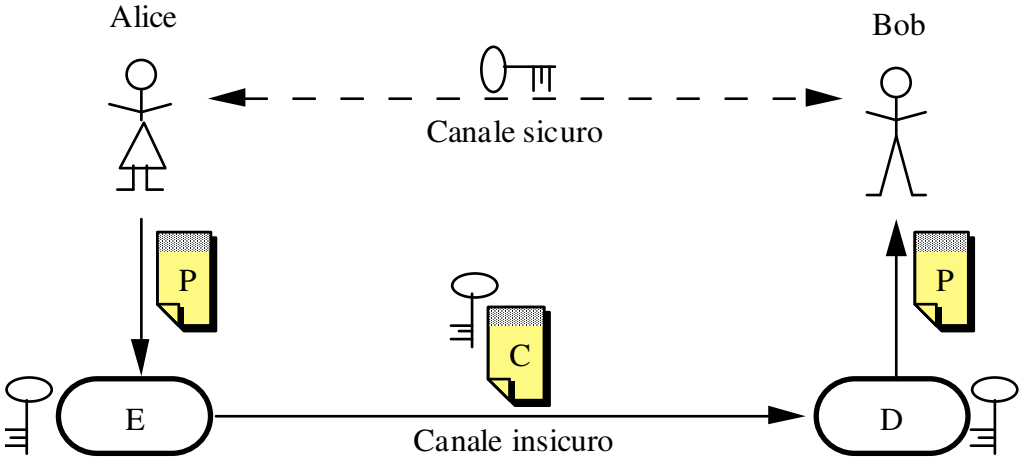

Figura 4-13: Crittografia a chiave segreta

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

Essi operano su un blocco di bit alla volta (***cifratura a blocchi***), al quale applicano la funzione di cifratura parametrizzata da una chiave. Dunque essi sono di fatto codici monoalfabetici, anche se l'alfabeto dei simboli (i valori dei possibili blocchi) è molto grande.

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. E' sopratutto 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.

4.3.1.2.1) 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.

Le caratteristiche salienti dell'algoritmo sono le seguenti:

- l'algoritmo è parametrizzato da una chiave di 56 bit (la chiave nel suo complesso è di 64 bit, ma 8 di questi sono bit di parità, uno per ciascun byte);
- il testo in chiaro è codificato in blocchi di 64 bit, che producono ciascuno 64 bit di testo cifrato (***cifratura a blocchi***);
- il DES consiste di 19 stadi, ciascuno dei quali opera una specifica trasformazione del blocco di 64 bit che riceve in input. L'output di ogni stadio è passato in input allo stadio successivo;
- in 16 di tali stadi la trasformazione viene parametrizzata da un sottoinsieme di 48 bit della chiave (il sottoinsieme è diverso in ciascuno stadio);
- è un codice monoalfabetico (uno stesso input cifrato con la stessa chiave produce sempre lo stesso output).

Più in particolare:

- nel primo stadio ($i = 0$) si effettua una permutazione (fissa, non dipendente dalla chiave) dei 64 bit del blocco, che vengono poi divisi in due blocchi da 32 bit (sinistro e destro, L_0 ed R_0 rispettivamente);
- in ciascuno dei 16 stadi successivi ($i = 1, 2, ..., 16$):
    - il blocco R_{i-1} in input ricevuto dallo stadio precedente viene passato inalterato allo stadio successivo, dove diviene L_i;
    - il blocco L_{i-1} in input ricevuto dallo stadio precedente viene combinato in XOR con il risultato dell'applicazione di una funzione $f(R_{i-1}, K_i)$, e passato allo stadio successivo, dove diviene R_i;
    - nella funzione $f(R_{i-1}, K_i)$, che sarà illustrata nel seguito, abbiamo che:
        - R_{i-1} sono i 32 bit del blocco da trasformare;
        - K_i sono i 48 bit tratti dai 56 bit della chiave usati per la trasformazione;
- nel penultimo stadio i due blocchi prodotti dallo stadio precedente vengono scambiati fra loro;
- nell'ultimo stadio ai 64 bit viene applicata la permutazione inversa di quella applicata nel primo stadio.

La figura seguente illustra lo schema complessivo del DES.

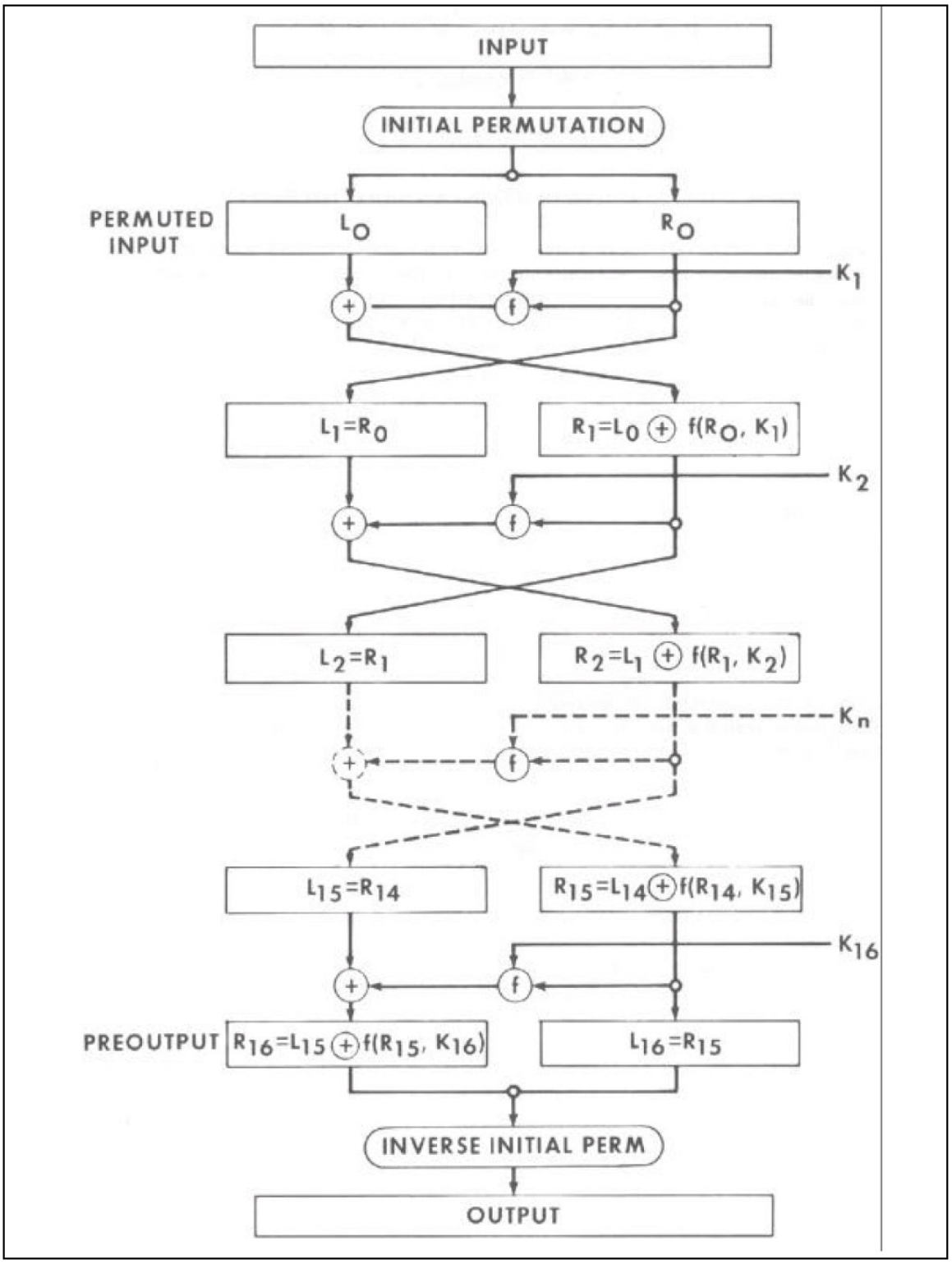

Figura 4-14: Schema complessivo del DES

La funzione $f(R_{i-1}, K_i)$ è realizzata per mezzo di vari S-box che operano secondo il seguente schema:

- i 32 bit del blocco R_{i-1} sono dapprima trasformati in 48 bit da un S-box ***espansore*** (S-box "E" nella figura seguente);
- si effettua l'XOR fra i 48 bit del blocco espanso ed i 48 bit di K_i;
- il risultato di tale XOR, che ha 48 bit, viene suddiviso in 8 blocchi da 6 bit; ciascun blocco di 6 bit viene passato in un S-box ***concentratore*** ("S_1", "S_2", …, "S_8" nella figura seguente) che lo trasforma in un blocco di 4 bit;
- alla sequenza di 8 x 4 = 32 bit risultanti si applica una permutazione (fissa) e si ottiene così l'output da passare allo stadio successivo.

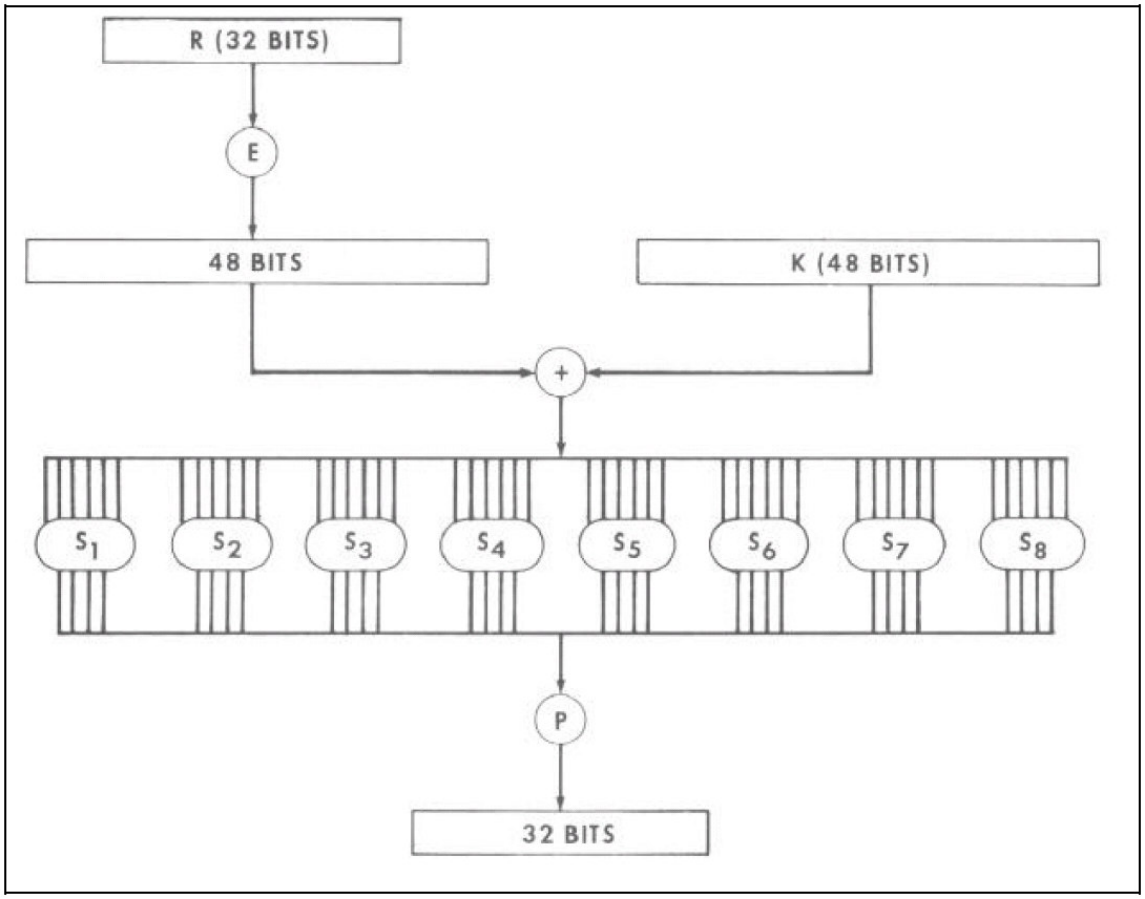

Figura 4-15: La funzione $f(R_{i-1}, K_i)$

Sia lo S-box E che gli S-box S_1, S_2, …, S_8 (che costituiscono il vero cuore del progetto del DES) sono definiti da apposite tabelle.

La tabella di E ha 8 righe e 6 colonne ed è la seguente:

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

Leggendo la tabella per righe, si ottiene via via l'indice di ogni bit da prelevare nella sequenza di 32 bit di R_{i-1} e da inserire nella sequenza di 48 bit in output. Ad esempio:

- il primo dei 48 bit in output è il trentaduesimo bit del blocco R_{i-1};
- il secondo dei 48 bit in output è il primo bit del blocco R_{i-1};
- il terzo dei 48 bit in output è il secondo bit del blocco R_{i-1};
- il diciannovesimo dei 48 bit in output è il dodicesimo bit del blocco R_{i-1}.

La tabella di S_1, costituita di 4 righe e 16 colonne, è la seguente (le altre sono analoghe):

14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

La tabella definisce la rete di sostituzione nel modo seguente:

- il primo e l'ultimo dei 6 bit in input identificano la riga;
- gli altri 4 bit in input identificano la colonna;
- il valore corrispondente della tabella, espresso in binario, costituisce i 4 bit in output (si noti che i valori della tabella sono tutti compresi fra 0 e 15).

Un ultimo aspetto da chiarire è il meccanismo di selezione del sottoinsieme di 48 bit dei 56 della chiave che va passato a ciascuno dei 16 stadi. Esso è illustrato dalla figura seguente.

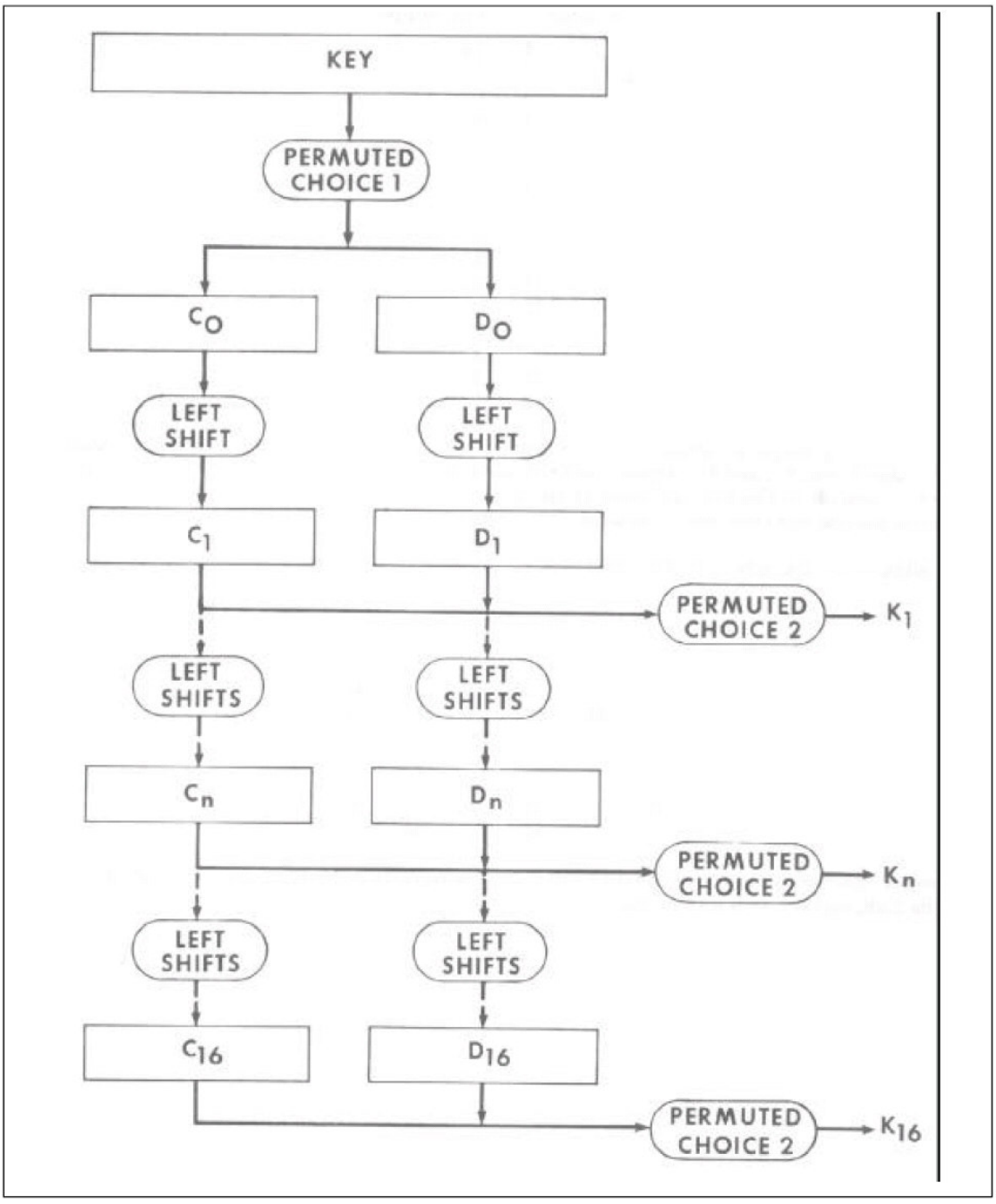

Figura 4-16: Generazione delle chiavi

Il funzionamento è il seguente:

- inizialmente si applica una prima ***scelta permutata***, ossia la selezione di un sottoinsieme di bit della chiave e la loro contestuale permutazione; questa prima scelta permutata (Permuted Choice 1) è definita da un'apposita tabella e produce una permutazione dei soli 56 bit della chiave vera e propria, escludendo cioè da tale operazione gli 8 bit di parità;

- i risultanti 56 bit vengono divisi in due blocchi da 28 bit ciascuno (C_0 e D_0), a ciascuno dei quali si applica uno shift circolare producendo così i blocchi C_1 e D_1;
- in ogni stadio successivo, alla sequenza di 56 bit ottenuta dalla concatenazione dei blocchi C_i e D_i in input dallo stadio precedente si applica una seconda scelta permutata (Permuted Choice 2, identica per tutti gli stadi) che produce i 48 bit da passare alla funzione f. Anche la Permuted Choice 2 è definita da una tabella;
- a ciascuno dei due blocchi C_i e D_i viene poi applicato uno shift circolare (singolo o doppio, a seconda dell'indice dello stadio) prima di passarli come C_{i+1} e D_{i+1} allo stadio successivo.

La tabella della Permuted Choice 2 ha 8 righe e 6 colonne ed è la seguente:

14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

Leggendo la tabella per righe, si ottiene via via l'indice di ogni bit da prelevare dalla sequenza di 56 bit di C_iD_i e da inserire nella sequenza di 48 bit che viene poi passata alla funzione f. Ad esempio:

- il primo dei 48 bit in output è il quattordicesimo bit della sequenza C_iD_i;
- il secondo dei 48 bit in output è il diciassettesimo bit della sequenza C_iD_i;
- il terzo dei 48 bit in output è l'undiciesimo bit della sequenza C_iD_i;
- il diciannovesimo dei 48 bit in output è il sediciesimo bit della sequenza C_iD_i.

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 criptanalisi 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, a parte l'approccio della forza bruta. Il meccanismo è il seguente.

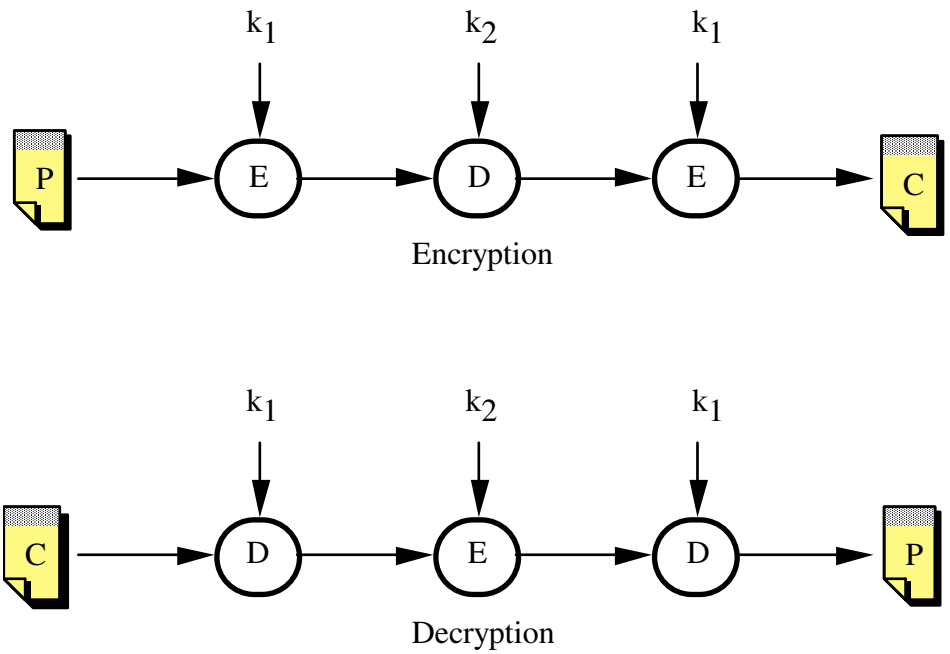

Figura 4-17: Triple DES

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

Si noti che anche il Triple DES è un codice monolfabetico, per il quale però 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.

4.3.1.2.2) 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.

4.3.1.2.3) 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 criptoanalisi che indeboliscano AES, e la lunghezza delle chiavi utilizzabili (soprattutto nel caso del valore 256) rende privo di speranze un attacco di forza bruta.

4.3.1.3) 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).

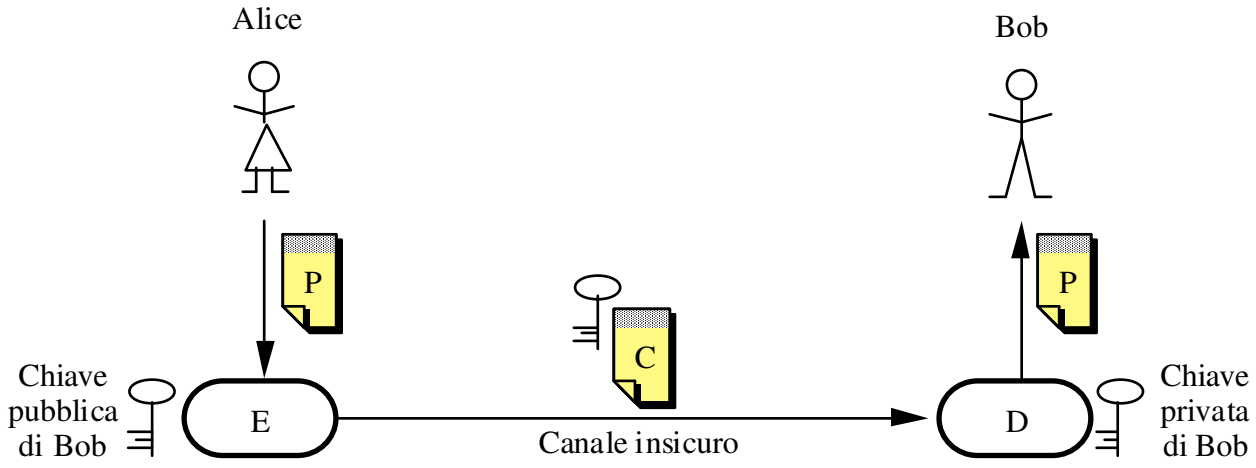

Figura 4-18: Riservatezza mediante crittografia a chiave pubblica

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.

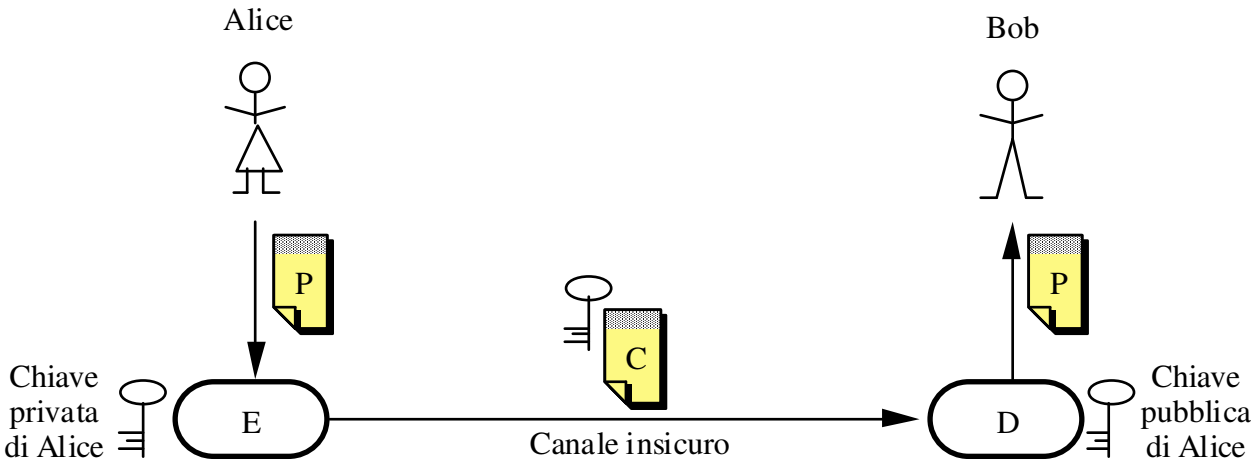

Figura 4-19: Autenticazione mediante crittografia a chiave pubblica

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.

4.3.1.3.1) RSA

L'algoritmo a chiave pubblica piu 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).

4.3.1.3.2) Premesse matematiche

Aritmetiche finite

Si opera in un'aritmetica finita (o circolare) ***modulo n***, cioe' a valori interi 0,1,...,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*** e' quel numero y tale che

$$xy \equiv 1 \bmod n.$$

Funzione di Eulero

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

- Se n e' 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

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

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

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

4.3.1.3.3) L'algoritmo RSA

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

$$de \equiv 1 \bmod \Phi(n);$$

- 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:

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

Infatti:

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

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

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

e quindi anche

$$de \equiv 1 \bmod \Phi(n)$$

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

$$(dx - 1)/\ \Phi(n) \text{ e' intero.}$$

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:

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

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

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

4.3.1.3.4) Correttezza dell'algoritmo RSA

Come visto sopra, si ha:

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

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

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:

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

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

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

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

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

Da cui discende che:

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

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 puo' scrivere:

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

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

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

Che si puo' riscrivere come:

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

Ora, si noti che:

- $t - 1$ e' uguale a $\Phi(t)$;
- $qs^{k(s-1)}$ e' primo con t, poiche' sia q (che e' minore di t) che s (per la scelta iniziale) sono primi con t.

Dunque, per il teorema di Eulero si ha:

$$(qs^{k(s-1)})^{(t-1)} = (qs^{k(s-1)})^{\Phi(t)} \equiv 1 \bmod t$$

ossia

(2) $$(qs^{k(s-1)})^{(t-1)} = xt + 1$$

Sostituendo la (2) nella (1), si ottiene:

$$P' = ((xt + 1)\ qs)\ \bmod n$$
$$P' = (xtqs + qs)\ \bmod n$$

Ponendo $xq = w$ e ricordando che $st = n$, si ottiene:

$$P' = (wn + qs)\ \bmod n = qs = P$$

4.3.1.3.5) Considerazioni su RSA

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

1. trovare **s** e **t**, 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.

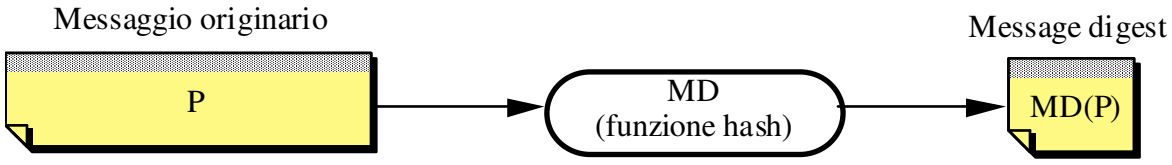

Figura 4-20: Calcolo del riassunto del messaggio

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:

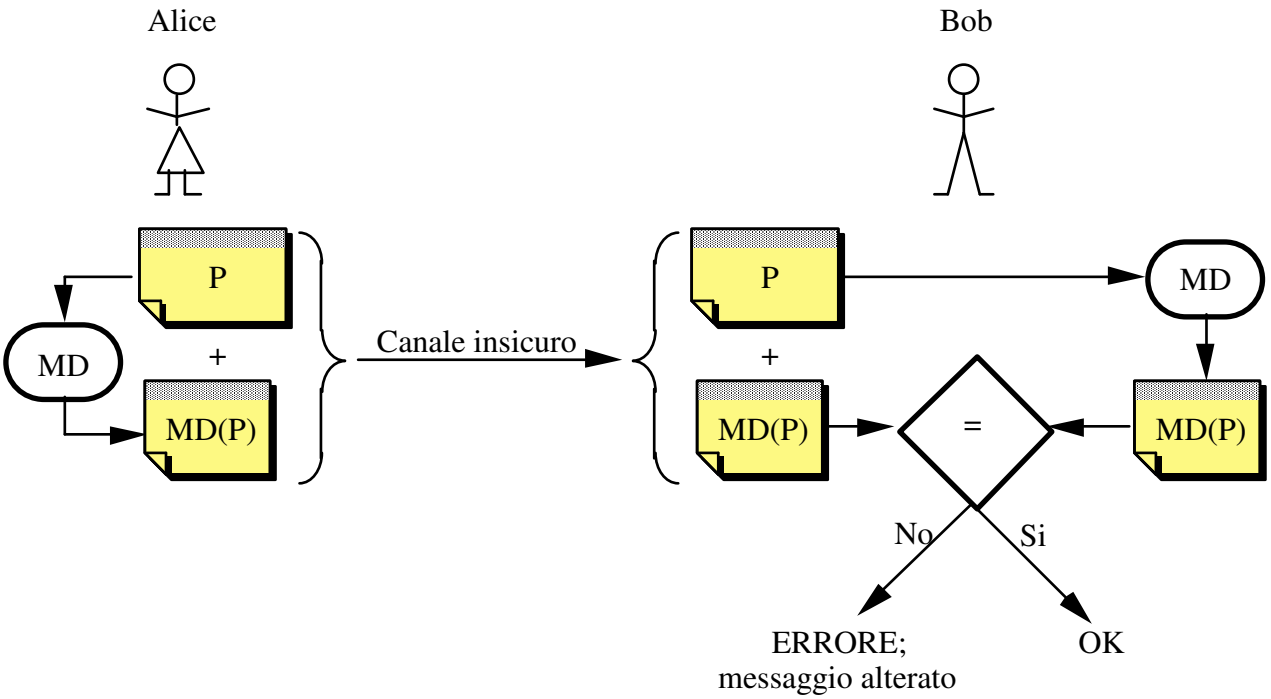

Figura 4-21: Uso del digest per il controllo di integrità del messaggio

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.

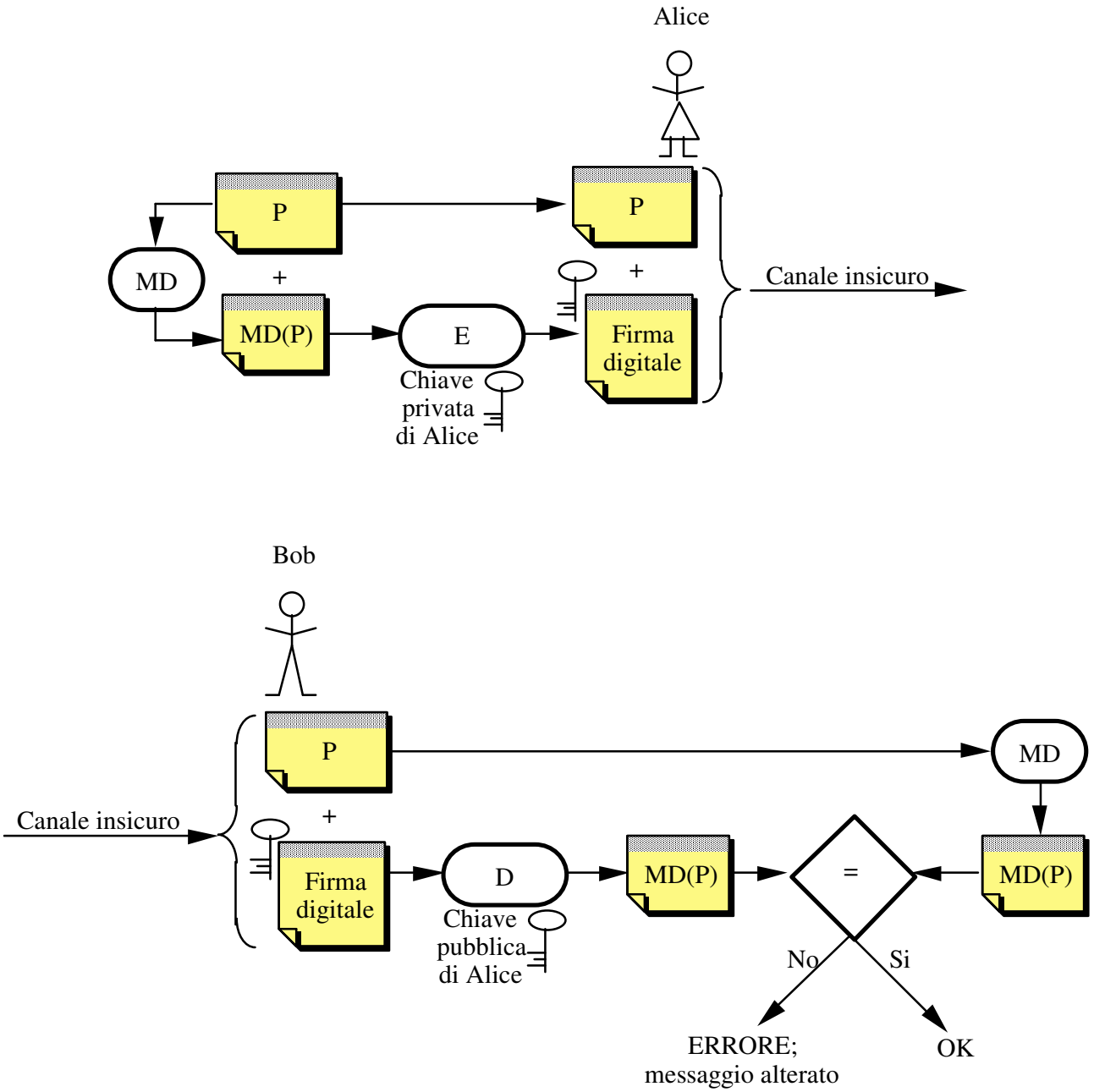

Figura 4-22: Firma digitale

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.

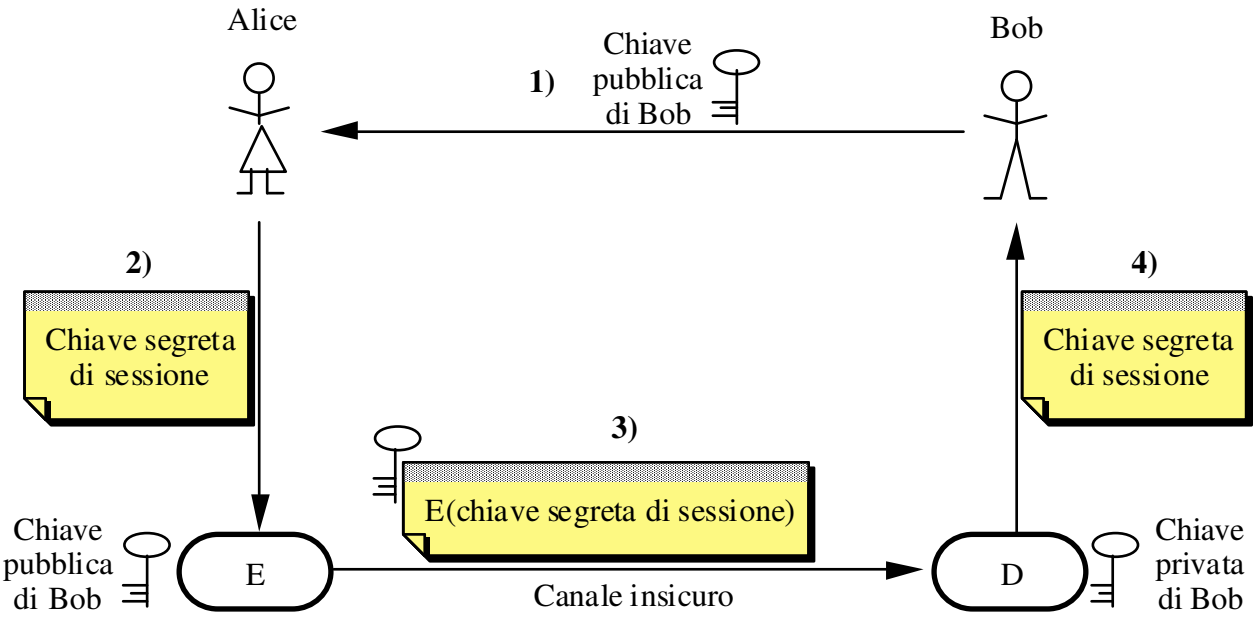

Figura 4-23: Determinazione della chiave segreta di sessione

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 interporsi nella successiva comunicazione e decifrare tutto (***man in the middle attack***).

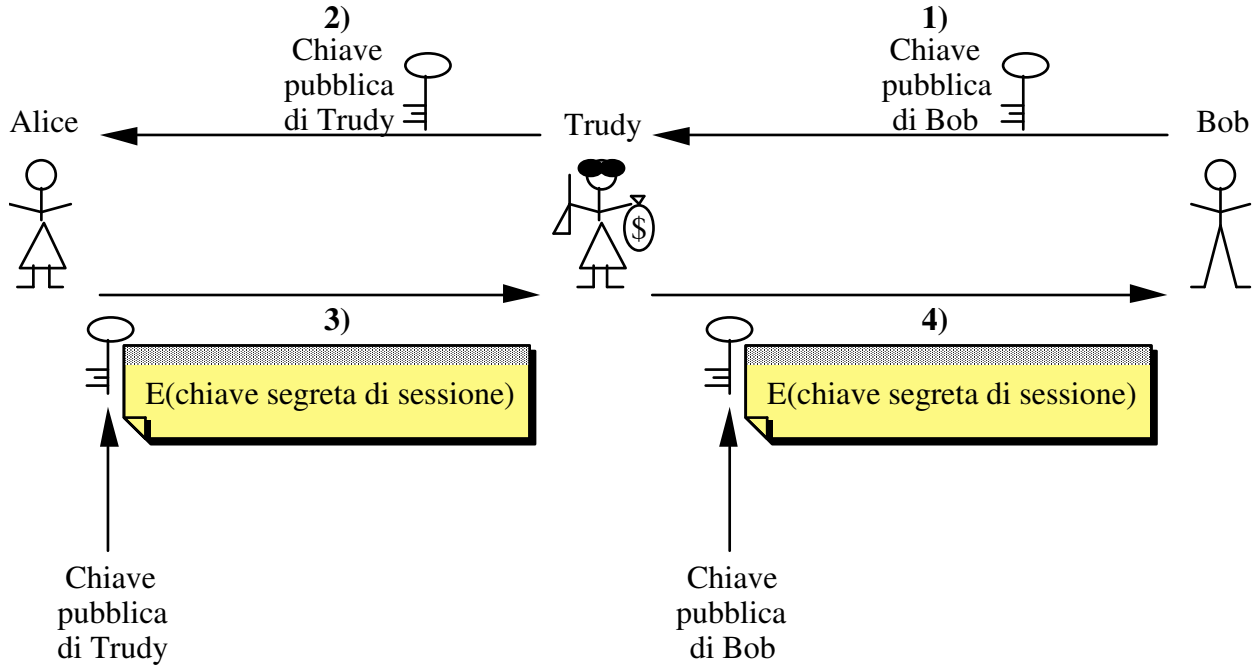

Figura 4-24: Trudy si interpone fra Alice e Bob

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 Bob invia ad Alice il proprio certificato che gli è stato precedentemente rilasciato da una CA; in questo modo Alice ha la garanzia che si tratta proprio della chiave di Bob e non di quella di Trudy.

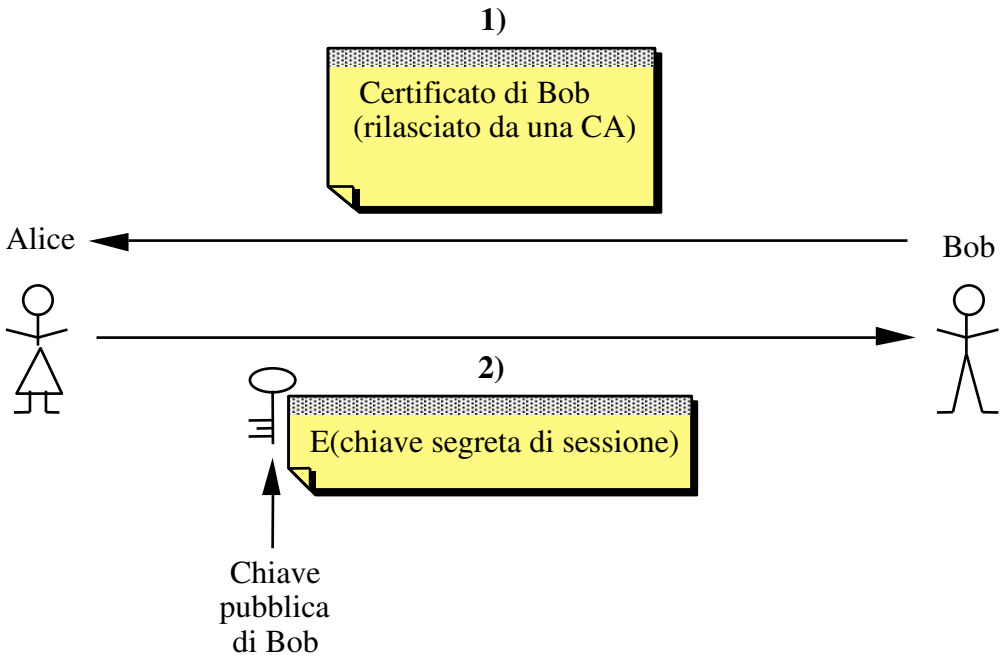

Figura 4-25: Ricorso ad una CA per avere la chiave pubblica di Bob

4.3.3.3) Secure Socket Layer

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:

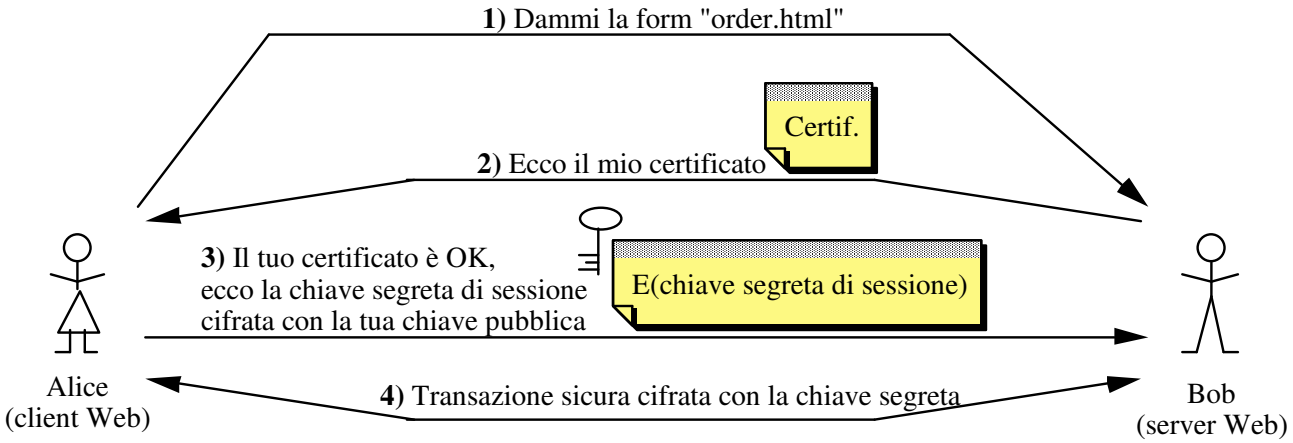

Figura 4-26: Funzionamento del protocollo SSL

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.