



Riservatezza

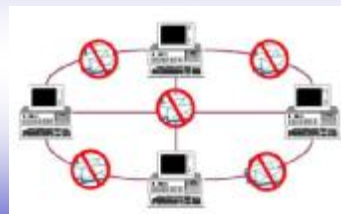
Soluzione: - **tecniche di cifratura e decifratura**

- 1) SISTEMI CRITTOGRAFICI A CHIAVE PRIVATA (SIMMETRICA)
- 2) SISTEMI CRITTOGRAFICI A CHIAVE PUBBLICA (ASIMMETRICA)
- 3) SISTEMI CRITTOGRAFICI MISTI.



Riservatezza (cont.)

Una rete sicura:



Riservatezza (cont.)

- 1) SISTEMI CRITTOGRAFICI A CHIAVE PRIVATA (SIMMETRICA)



- Mittente e destinatario utilizzano la stessa chiave per comunicare



Riservatezza (cont.)

- 2) SISTEMI CRITTOGRAFICI A CHIAVE PUBBLICA



- Due chiavi differenti: pubblica e privata
- La chiave pubblica cifra il messaggio
- La chiave privata lo decifra
- Senza conoscere la chiave privata non si può decodificare



Riservatezza (cont.)

- 3) SISTEMI CRITTOGRAFICI MISTI



- Ogni utente ha una chiave pubblica e una privata
- Creazione di una chiave simmetrica per ogni sessione
- La chiave simmetrica viene cifrata e allegata al messaggio
- Il destinatario decifra la chiave simmetrica con la sua privata
- La usa per decifrare il messaggio



Autenticazione



L'autenticazione avviene tramite l'applicazione al nostro documento elettronico di una firma digitale.



Mappa testuale

- ├ Introduzione
- └ **RSA**
 - OpenSSL
 - cPongo
 - Bibliografia



RSA

RSA si basa sul prodotto di due numeri primi di grandi dimensioni, che possono superare le 300 cifre.

- due fasi:
- generazione della coppia di chiavi
 - utilizzo delle stesse.



RSA (cont.)

Prima fase:

1. vengono scelti due numeri primi **p**, **q** molto grandi;
2. viene calcolato $n=pq$, e la funzione di Eulero $\Phi(n) = (p-1)(q-1)$ dopo di che i due primi **p**, **q** vengono eliminati;
3. si sceglie un intero **e** minore di $\Phi(n)$ e primo con esso;
4. utilizzando la versione estesa dell'algoritmo di Euclide viene calcolato l'intero **d** così da avere $e * d = 1 \text{ mod } \Phi(n)$;
5. vengono resi pubblici i valori **e**, **n** che costituiscono la chiave pubblica e mantenuto segreto **d** che, utilizzato con **n** rappresenta la chiave privata.



RSA (cont.)

Seconda fase:

C = messaggio cifrato

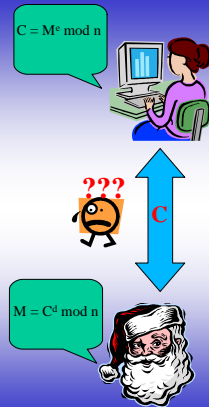
M = messaggio in chiaro

- Cifratura:

$$C = M^e \text{ mod } n$$

- Decifratura:

$$M = C^d \text{ mod } n$$



RSA (cont.)

- Metodo più efficiente di decifratura:

- Basato sul Teorema Cinese del Resto
- Dati **p**, **q** calcolo **a**, **b** t.c. $ap + bq = 1$
- $c = bq$, $d = ap$

- Dato un messaggio cifrato **C**
- $y = C^d \text{ mod } p-1 \text{ mod } p$
- $z = C^d \text{ mod } q-1 \text{ mod } q$
- $M = (cy + dz) \text{ mod } n$

- ü al posto di un esponenziale modulo **n** abbiamo due esponenziali modulo **p**, **q**
- ü gli esponenti calcolati possono essere conservati per altre operazioni di decifratura



Firma Digitale RSA

Firma— $M^d \text{ mod } n$ // Firmiamo **M** utilizzando la *chiave privata (d ed n)*

Verifica (Firma)
if $M = \text{Firma}^e \text{ mod } n$ // Verifichiamo la firma utilizzando la *chiave pubblica (e ed n)*

return TRUE
else
return FALSE





Mapa testuale

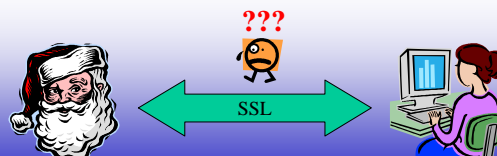
- Introduzione
- RSA
- OpenSSL
- cPongo
- Bibliografia



OpenSSL

Lo strumento di cui ci siamo avvalsi per lo sviluppo del nostro programma è l'OpenSSL

- Potente strumento per la sicurezza delle reti
- GNU Public Licence
- Multiplatforma



OpenSSL (cont.)

La struttura di una chiave RSA:

```
struct RSA
{
    BIGNUM *n;           // public modulus
    BIGNUM *e;           // public exponent
    BIGNUM *d;           // private exponent
    BIGNUM *p;           // secret prime factor
    BIGNUM *q;           // secret prime factor
    BIGNUM *dmp1;       // d mod (p-1)
    BIGNUM *dmq1;       // d mod (q-1)
};
```

- p, q, dmp1, dmq1 vengono usati per velocizzare le operazioni di decifratura
- nella chiave privata possono essere anche NULL
- nella chiave pubblica l'esponente privato e i relativi fattori segreti sono NULL



OpenSSL (cont.)

Queste funzioni effettuano le operazioni di cifratura e decifratura di un messaggio:

```
int RSA_public_encrypt(int flen, unsigned char *from, unsigned
char *to, RSA *rsa, int padding);
flen: numero di byte da cifrare
from: stringa da crittografare
to: buffer dove viene scritto l'output cifrato
rsa: chiave pubblica RSA
padding: specifica quale tipo di padding debba essere
usato per completare la taglia del messaggio
Ritorna la grandezza dei dati crittografati
```

```
int RSA_private_decrypt(int flen, unsigned char *from, unsigned
char *to, RSA *rsa, int padding);
flen: numero di byte da decifrare
from: stringa da decrittare
to: buffer che conterrà l'output in chiaro
rsa: chiave privata RSA
padding: questo parametro è posto uguale al valore di
padding della funzione encrypt
Ritorna la grandezza del testo in chiaro
```



OpenSSL (cont.)

Queste funzioni implementano l'operazione di firma e di verifica del messaggio:

```
int RSA_sign(int type, unsigned char *m, unsigned int m_len, unsigned
char *sigret, unsigned int *siglen, RSA *rsa);
type: indica il tipo di algoritmo che esegue la funzione hash
m: buffer che contiene i dati da firmare
m_len: numero di byte del buffer
sigret: buffer che conterrà la firma
siglen: numero di byte della firma
rsa: puntatore alla chiave privata

int RSA_verify(int type, unsigned char *m, unsigned int m_len, unsigned
char *sigbuf, unsigned int siglen, RSA *rsa);
type: indica il tipo di algoritmo che esegue la funzione hash
m: buffer che contiene i dati da verificare
m_len: numero di byte del buffer
sigbuf: buffer contenente la firma da verificare
siglen: numero di byte della firma
rsa: puntatore alla chiave pubblica
```



OpenSSL (cont.)

OpenSSL supporta due formati standard per conservare le chiavi e per scambiarle:

- DER (Distinguished Encoding Rules):
 - Formato binario
 - Utilizzato per i trasferimenti sulle reti
 - Non ideale per comunicazioni testuali (es. E-mail)
- PEM (Privacy Enhanced Mail):
 - Definito in RFCs 1421, 1422, 1423, e 1424.
 - Codificato in base 64 (Formato testuale)
 - Ideale per comunicazioni testuali

In generale se abbiamo la necessità di scrivere i dati sul disco, dovremmo utilizzare il formato PEM



OpenSSL (cont.)

Ecco come risulterebbe una chiave privata in formato PEM a 1024 bit :

```
-----BEGIN RSA PRIVATE KEY-----
MIICXQIBAAKBgQDwxE5+qNqE20NQWgRi71vkqroI+52CgzGmKv42BsE6kk1QiVSS
9Qm75bKfEP7Jz83NO/YO5DPH6Zvr38nY6HCY3Ta/fNU00/xNE3UiOqSK8dn7lbw
7itshIwjPpK09cgL6wos4Sm+lcHjdrk5DyTKcRNadWYXsXnsdrR5Fwt.ImwIBAwKB
gQCPLYmpxecDPNeK5xhB90ftxyawp75XAIEZcf7Oryt8YYjgW423TgZ9Q8xqC1SG
ioKzFU60mCKFGm7jy1TE7RaAXuiq84NDz71wmFjOggGzePH14CH5WnP10GTwffsI
cEBr5WjUu2DTNKAudk+1jLJFXFY6UIws6tzB1SPNNagewJBAPAvdSm765vsMcOL
zfIks0rMpx9c4QZrXAgm+IoAN3EQOvZ9KNAK1xkUYDPEyJ93GY5ZjyPaFnM9t+X
n971lr8CQQDk6GVm8oN9Z5aMPNDSzNGrj+f+xxngEjxDzcH2YfDcVY8cb8T4Daqt
vWSP2JWz4YAWJzACuuQ+zAMBAJnQ4E1AkeAeOB+jcSfyZ/LL17KJTBh3hzMaFOiW
BEeSsBn7BqrPoLV8pFNwirHku2LqzTC3bFPORCZEKF+a5oik1SEVP0PkfwJBAJia
7kShrP5FDwgoizczNnJf7/8vZpVhfYKJK/mWoJLkyhL1Lfqqzxx5+QwqY7u9BAA7
Eyx8mCndV1YAZotAMMCQDYz4eFgmTJuq/J/Ls7NusdBxuwH0fKRt2KPhCzW5r
5mEeCDf1Q3ATBjeUt6Z77JG0aEmQuq5NOqd/julVtBst
-----END RSA PRIVATE KEY-----
```



OpenSSL (cont.)


Ed ecco la chiave pubblica relativa:

```
-----BEGIN RSA PUBLIC KEY-----
MIGHAoGBANbETn6o2oTbQ1BapGLvW+Squgj7nYKDMAyq/jYgWtqSTVCJVJL1Cbvl
sp8Q/snPzc079g7kMBenplWvfYdJocJjdNr981TQ7/E0TdsI5BIrx2fuVvDuK2yE
jCM+krTlyAvrCizhKVhPJMpxEsdRyYegH1p1bJdLE2x2tHkVa0ibAgED
-----END RSA PUBLIC KEY-----
```




Mappa testuale

- Introduzione
- RSA
- OpenSSL
- **cPongo**
- Bibliografia



cPongo

- Il progetto è composto da due programmi: un server e un client capaci di scambiare messaggi su un canale sicuro
- Tutti i messaggi sono crittografati e firmati per evitare qualsiasi intrusione sul canale
- La crittografia è eseguita con il crittosistema RSA a 1024 bit implementato dalle librerie dell'OpenSSL
- Le chiavi (pubblica e privata) saranno salvate in formato testuale (PEM)
- Per lo scambio dei messaggi sono state usate le API Socket di Berkeley



cPongo (cont.)

Il server è stato progettato per una comunicazione concorrente con infiniti client a cui permette le seguenti operazioni:

- 1) Registrazione
- 2) Autenticazione
- 3) Cancellazione
- 4) Invio messaggi
- 5) Ricezione di tutti i messaggi



cPongo (cont.)




Fase di registrazione

Fase di login

Screenshots

Il server accetta la connessione del client e ne stampa a video la chiave

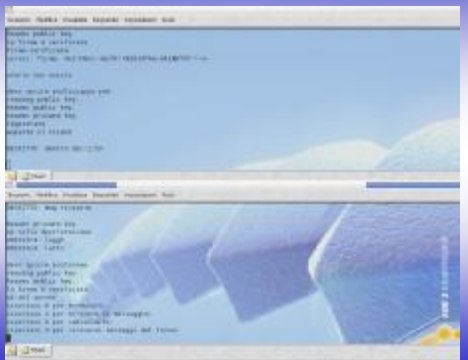
Il client procede con la raccolta dei dati per la registrazione



Screenshots (cont.)


Il server ha appena autenticato il client ed è in attesa di una richiesta

Il client mostra il menù all'utente



Screenshots (cont.)

Validazione di un messaggio scritto nel database



Immagini (cont.)

Questo è ciò che contiene la cartella del server.
Nei file *.pem sono immagazzinate le chiavi pubbliche dei client registrati.



Manuale d'Uso

- Lanciare il server digitando **.Jserver "# porta"**
- Lanciare il client digitando **.Jclient "IP del server"** (Loopback: 127.0.0.1)
- Seguire le indicazioni del client per l'accesso al server
- Utilizzare le opzioni offerte dal menù
- Per una rapida lettura dei database, digitare **.Jleggi**
- Per testare l'efficacia del rilevamento di contraffazioni, digitare: **.Jbrake n**, dove n è il numero delle firme dei messaggi, contenuti nel database del server, da corrompere

Mappa testuale

- Introduzione
- RSA
- OpenSSL
- cPongo
- **Bibliografia**



Bibliografia

- Pravin Chandra, Matt Messier, John Viega
Network Security with OpenSSL
O'Reilly 2002
- William Stallings
Crittografia e Sicurezza delle Reti
McGraw-Hill 2003
- OndaQuadra0A Electronic Magazine:
OpenSSL/RSA di Paolo Ardoino
<http://ardoino.altervista.org/>
- Introduction to Cryptography
University of Haifa – Fall 2004
Benny Pinkas
www.pinkas.net/course.html
- La Sicurezza Delle Transazioni E Comunicazioni Attraverso Internet
Materiale prelevato dal sito <http://www.amicopc.com>
- Appunti del corso di [sicurezza su reti](#) del docente Alfredo De Santis
- Appunti del corso di [reti di calcolatori](#) del docente Roberto De Prisco
- Specifiche del formato PEM: [RFCs 1421, 1422, 1423, e 1424](#)
- Specifiche del formato DER: [RFC 2314](#)
- Sito ufficiale OpenSSL: <http://www.openssl.org>