Sicurezza dei Sistemi Informatici

Esercitazioni OpenSSL

Esercitazione 3

Cifratura asimmetrica

Obiettivo

- Scambio file su applicazione client-server.
- Il server possiede la propria chiave privata.
- Il client ha la chiave pubblica del server.
- Il client vuole inviare al server un file, garantendo la confidenzialità.

Funzioni client

- #include <openssl/rsa.h>
- #include <openssl/pem.h>
- Allocazione contesto
  RSA* rsa=RSA_new();
- Cifratura
  RSA* PEM_read_RSAPublicKey(…);
  RSA_public_encrypt(…);
- Deallocazione contesto
  RSA_free(rsa);

Funzioni client

- RSA* PEM_read_RSAPublicKey
(FILE *fp, RSA **x, pem_password_cb *cb, void *u);

fp = file contenente la chiave pubblica
x = contesto RSA

Funzioni client

• RSA_public_encrypt
(int flen, unsigned char *from, unsigned char *to, RSA *rsa, int padding);

flen = dimensione del testo in chiaro in byte
from = buffer contenente il testo in chiaro
to = buffer per contenere il testo cifrato prodotto
rsa = contesto
padding = tipo padding (usare RSA_PKCS1_PADDING)

• La dimensione del buffer *to* deve essere pari **esattamente** a *RSA_size(rsa)*.

• *RSA_public_encrypt()* ritorna la dimensione in byte del testo cifrato prodotto, cioè *RSA_size(rsa)*.

Funzioni server

- #include <openssl/rsa.h>
- #include <openssl/pem.h>
- Allocazione contesto
  RSA* rsa=RSA_new();
- Decifratura
  OpenSSL_add_all_algorithms();
  RSA* PEM_read_RSAPrivateKey(…);
  RSA_private_decrypt(…);
- Deallocazione contesto
  RSA_free(rsa);

Funzioni server

- RSA* PEM_read_RSAPrivateKey
(FILE *fp, RSA **x, pem_password_cb *cb, void *u);

fp = file contenente la chiave privata
x = contesto RSA
u = stringa contentente la password della chiave privata

Funzioni server

• RSA_private_decrypt
(int flen, unsigned char *from, unsigned char *to, RSA *rsa, int padding);

flen = dimensione del testo cifrato in byte
from = buffer contenente il testo cifrato
to = buffer per contenere il testo in chiaro prodotto
rsa = contesto
padding = tipo padding (usare RSA_PKCS1_PADDING)

• I byte scritti nel buffer *to* saranno **non più di** *RSA_size(rsa)*.

• *RSA_public_decrypt()* ritorna la dimensione in byte del testo in chiaro prodotto.

Altre funzioni utili

- RSA_generate_key(…)

Genera una coppia di chiavi RSA

- PEM_write_RSAPrivateKey(…)

Scrive su file una chiave privata RSA

- PEM_write_RSAPublicKey(…)

Scrive su file una chiave pubblica RSA

Generazione delle chiavi (1)

```
#include <stdio.h>
#include <string.h>
#include <openssl/rsa.h>
#include <openssl/rand.h>
#include <openssl/pem.h>

…

char *file_pem = "priv.pem";
char *file_pem_pub = "pub.pem";
FILE *fp;

int bits = 1024;
unsigned long exp = RSA_F4;

RSA *rsa;

rsa=RSA_generate_key(bits,exp,NULL,NULL);
```

Generazione delle chiavi (2)

```
fp = fopen(file_pem, "w");

unsigned char *kstr = "password";

PEM_write_RSAPrivateKey
(fp,rsa,EVP_des_ede3_cbc(),kstr,strlen(kstr),NULL,NULL));

close(fp);

fp = fopen(file_pem_pub, "w");

PEM_write_RSAPublicKey(fp, rsa))

close(fp);

RSA_free(rsa);

…
```

Esercizio

- Si consideri il cifrario RSA.

- Si consideri il metodo *digital envelope*
  - Il client invia al server $\{E_k(F), E_e(k)\}$
  - *k* chiave simmetrica stabilita dal client
  - *e* chiave pubblica del server

- Generare una coppia di chiavi RSA utilizzando la funzione *RSA_generate_key()*. Fornire la chiave pubblica al client, e la chiave privata al server.

- Il client legge un file *F*, e invia il suo contenuto al server, secondo il metodo digital envelope. Il server decifra quanto ricevuto dal client, e lo salva su file locale.

- Confrontare il miglioramento delle prestazioni rispetto all'invio del file cifrato con chiave simmetrica. Utilizzare il comando *time* di UNIX.