Reti di Calcolatori

Esercitazione su: Crittografia &
Java Cryptographic Architecture (JCA)

ABC, Alice Bob e Clara

Alice deve inviare un documento a Bob e a Clara.

Il documento deve garantire Bob e Clara riguardo l'identità del mittente (Alice), e deve essere leggibile solo da Bob e Clara.

Descrivere le operazioni che Alice deve effettuare per realizzare i suddetti obiettivi, supponendo che Alice, Bob e Clara dispongano di una comune infrastruttura a chiave pubblica (PKI).

Soluzione teorica

Passi da eseguire:

1. Alice, Bob e Clara generano una coppia di chiavi RSA e pubblicano le loro rispettive chiavi pubbliche.
2. Alice cifra il messaggio con la sua chiave privata, per garantire l'identificazione del mittente.
3. Alice cifra una seconda volta il messaggio cifrato al passo 2 con la chiave pubblica di Bob, per garantire che solo Bob possa decifrare il messaggio.
4. Alice cifra una seconda volta il messaggio cifrato al passo 2 con la chiave pubblica di Clara, per garantire che solo Clara possa decifrarlo.
5. Bob riceve il messaggio del passo 3 e lo decifra, prima con la sua chiave privata e dopo con la chiave pubblica di Alice.
6. Clara riceve il messaggio del passo 4 e lo decifra, prima con la sua chiave privata e dopo con la chiave pubblica di Alice.

Soluzione implementata: Passo 1

- “Alice”, “Bob” e “Clara” eseguono la classe java “GeneraChiavi.java”.

- “GeneraChiavi.java” genera una coppia di chiavi (pubblica e privata) e richiede all’utente i nomi dei file in cui salvarle (es.: Alice.PublicKey e Alice.PrivateKey).

- “Alice”, “Bob” e “Clara”, dopo aver generato le chiavi, simulano la pubblicazione su una “Certification Authority” copiando la loro rispettiva chiave pubblica nella cartella “PublicKeyStore”.

Soluzione implementata: Passo 2-3

"Alice" esegue la classe java "CodificaMessaggio.java", la quale richiede nell'ordine:

1. La chiave privata del mittente (Alice.PrivateKey).

1. La chiave pubblica del destinatario (Bob.PublicKey).

1. Il nome del file da criptare (TestoInChiaro.txt).

1. Il nome del file in cui salvare i dati cifrati (TestoCodificatoPerBob.txt) .

Il file con i dati cifrati è inviato a Bob come allegato di una @mail.

Soluzione implementata: Passo 2-3

```
//Codice java per cifrare messaggi più lunghi di 128 Byte (1024 bit).

Cipher cipher = Cipher.getInstance("RSA");
cipher.init(Cipher.ENCRYPT_MODE, publicKey);

byte[] coded = new byte[(((plain.length-1)/100+1))*128];
int ixplain = 0;
int ixcoded = 0;
while((plain.length - ixplain) > 100)
{
        ixcoded += cipher.doFinal(plain, ixplain, 100, coded, ixcoded);
        ixplain += 100;
}
cipher.doFinal(plain, ixplain, plain.length - ixplain, coded, ixcoded);
```

Soluzione implementata: Passo 2-4

"Alice" esegue la classe java "CodificaMessaggio.java", la quale richiede nell'ordine:

1. La chiave privata del mittente (Alice.PrivateKey).

1. La chiave pubblica del destinatario (Clara.PublicKey).

1. Il nome del file da criptare (TestoInChiaro.txt).

1. Il nome del file in cui salvare il dati cifrati (TestoCodificatoPerClara.txt) .

Il file con i dati cifrati è inviato a Clara come allegato di una @mail.

Soluzione implementata: Passo 5

Dopo aver ricevuto la @mail da Alice, “Bob” salva l’allegato ed esegue la classe java “DecodificaMessaggio.java”, la quale richiede nell’ordine:

1. La chiave privata del destinatario (Bob.PrivateKey).

1. La chiave pubblica del mittente (Alice.PublicKey).

1. Il nome del file da decriptare (TestoCodificatoPerBob.txt).

1. Il nome del file in cui salvare i dati decifrati (TestoDecodificato.txt).

Soluzione implementata: Passo 5

```
// Codice java per decifrare messaggi più lunghi di 128 Byte (1024 bit).

Cipher cipher = Cipher.getInstance("RSA");
cipher.init(Cipher.DECRYPT_MODE, privateKey);

byte[] plain = new byte[code.length];
int ixplain = 0;
int ixcode = 0;
while((code.length - ixcode) > 128)
{
        ixplain += cipher.doFinal(code, ixcode, 128, plain, ixplain);
        ixcode += 128;
}
ixplain += cipher.doFinal(code, ixcode, code.length - ixcode, plain, ixplain);
```

Soluzione implementata: Passo 5

```
// Codice java per decifrare messaggi più lunghi di 128 Byte (1024 bit).

byte[] plain = new byte[code.length];
…
ixplain += cipher.doFinal(code, ixcode, code.length - ixcode, plain, ixplain);


// La dimensione reale dei dati decodificati è “ixplain“ e non “plain.length”
// quindi “plain” va troncato alla dimensione “ixplain“.

byte[] tempPlain = new byte[ ixplain ];
for(int i=0; i<ixplain ; i++)
        tempPlain[ i ] = plain[ i ];
plain = tempPlain;
```

Soluzione implementata: Passo 6

Dopo aver ricevuto la @mail da Alice, “Clara” salva l’allegato ed esegue la classe java “DecodificaMessaggio.java”, la quale richiede nell’ordine:

1. La chiave privata del destinatario (Clara.PrivateKey).

1. La chiave pubblica del mittente (Alice.PublicKey).

1. Il nome del file da decriptare (TestoCodificatoPerClara.txt).

1. Il nome del file in cui salvare i dati decifrati (TestoDecodificato.txt).