

# RSA in OpenSSL

**Alfredo De Santis**

Dipartimento di Informatica  
Università di Salerno

**ads@unisa.it**



**Maggio 2020**

# Outline

- Concetti Preliminari
- Cifratura Asimmetrica in OpenSSL

# Outline

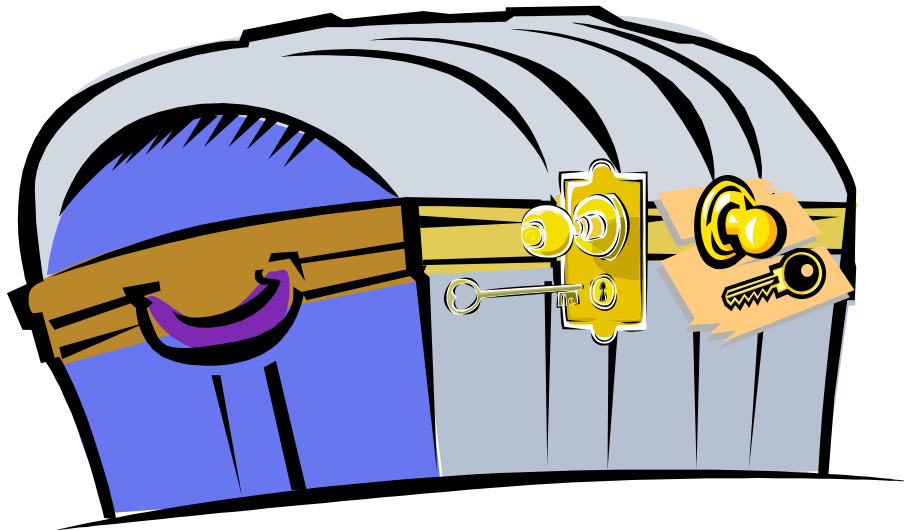
- Concetti Preliminari
- Cifratura Asimmetrica in OpenSSL

# Cifrari Asimmetrici

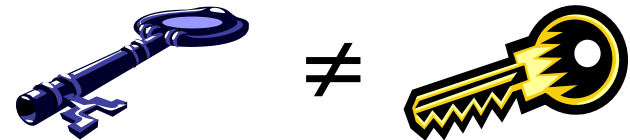
- Non ci sono chiavi condivise tra gli utenti
- Un utente A (Alice) genera da solo la propria coppia di chiavi, chiamate chiave pubblica e chiave privata
  - La chiave pubblica è resa pubblica
  - La chiave privata è memorizzata in modo sicuro dall'utente
- Chiunque può cifrare un messaggio per Alice
- Solo Alice può decifrare un messaggio cifrato per lei

# Cifrari Asimmetrici

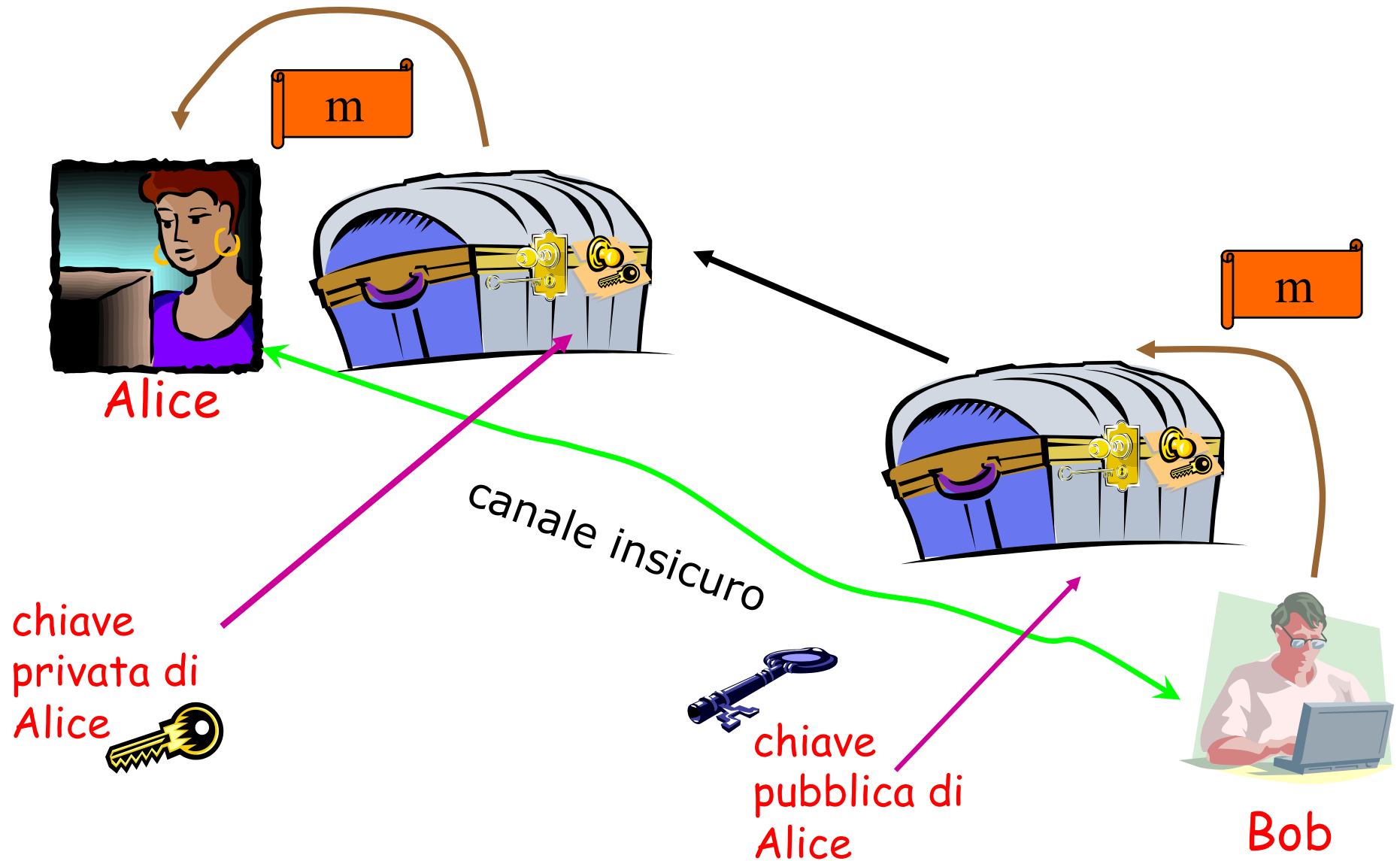
- Usano una cassaforte con due lucchetti
  - Con una chiave (**pubblica**) chiudiamo la cassaforte
  - Con l'altra chiave (**privata**) apriamo la cassaforte



Public key  $\neq$  Private key



# Cifrari Asimmetrici



# Cifrari Asimmetrici

chiave privata  
kpriv

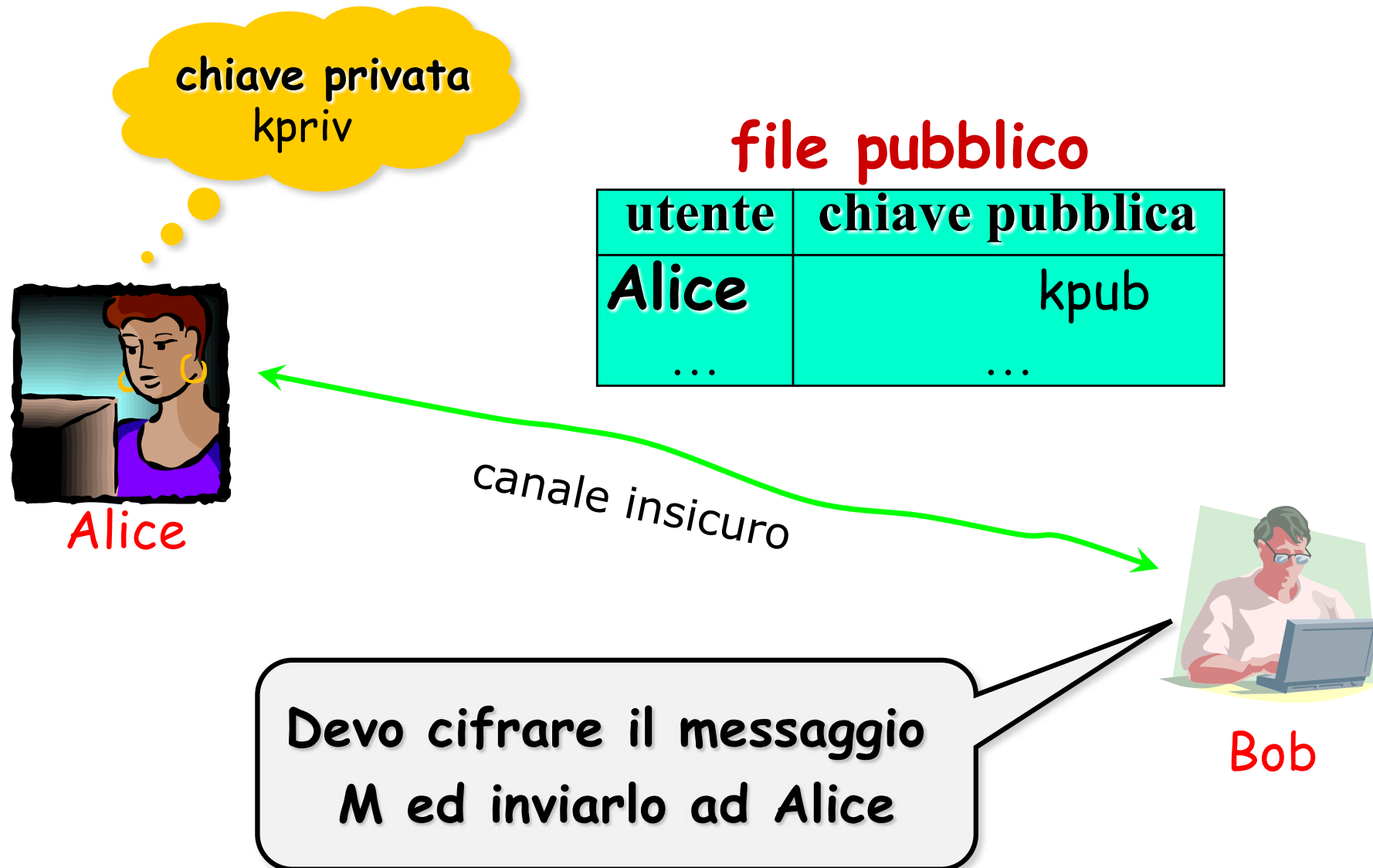


Alice

file pubblico

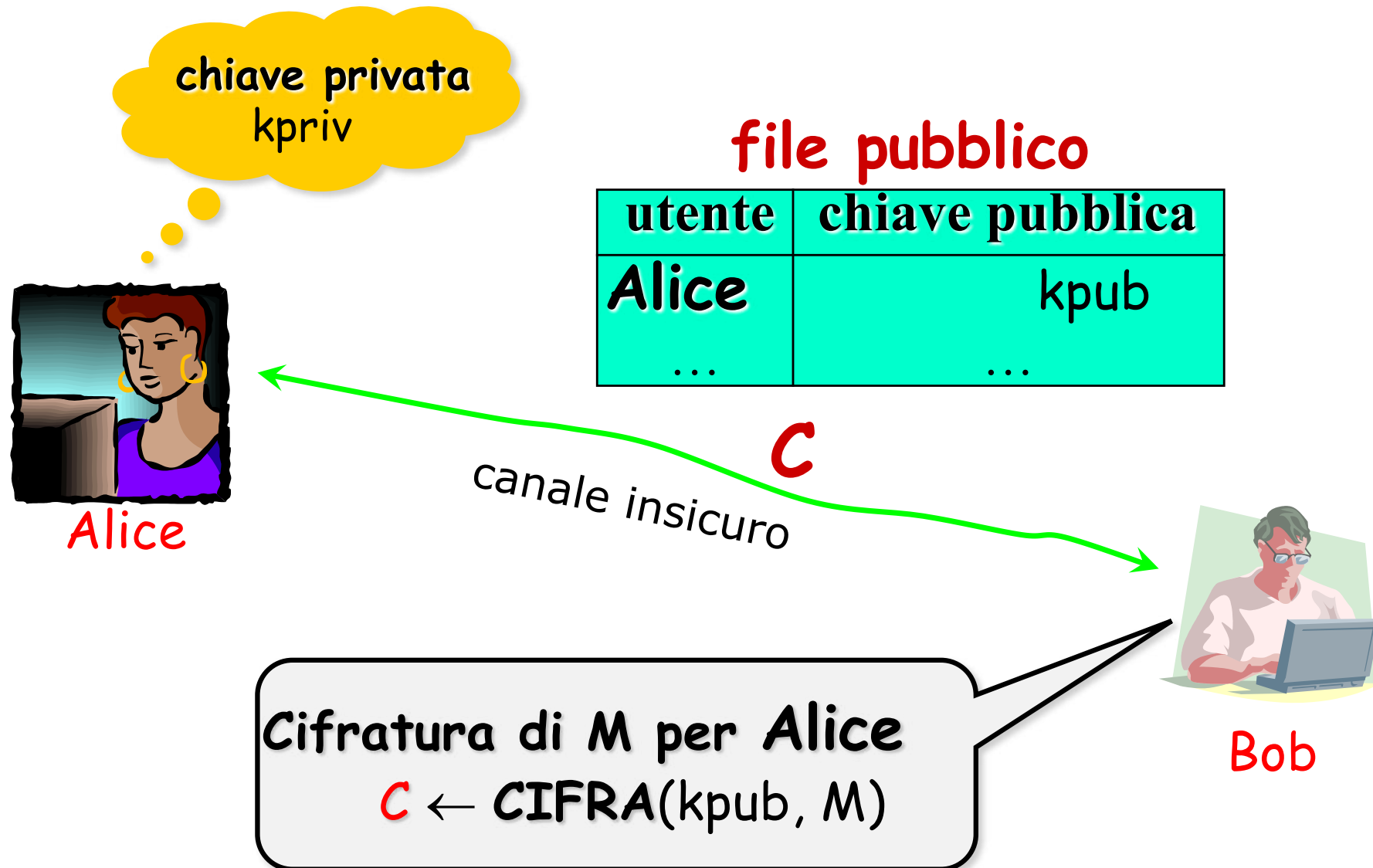
utente	chiave pubblica
Alice	kpub
...	...

# Cifratura





# Cifratura



# Decifratura

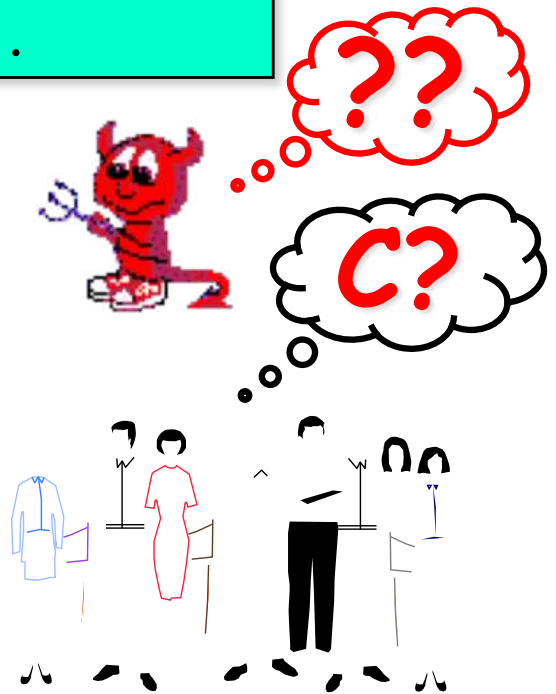
Devo decifrare il  
messaggio cifrato **C**

file pubblico

utente	chiave pubblica
Alice	kpub
...	...



Alice



# Decifratura

chiave privata  
 $k_{priv}$

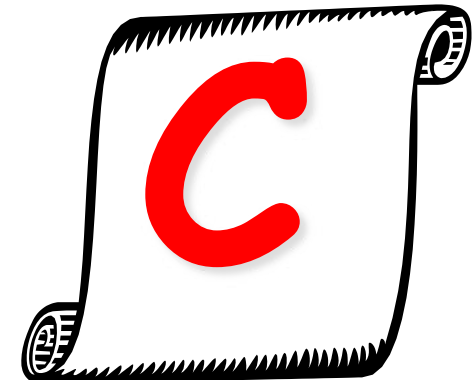
file pubblico

utente	chiave pubblica
Alice	$k_{pub}$
...	...



Alice

Decifratura di  $C$   
 $M \leftarrow \text{DECIFRA}(k_{priv}, C)$



# Outline

- Concetti Preliminari
- Cifratura Asimmetrica in OpenSSL

# Cifratura Asimmetrica in OpenSSL

## Descrizione dei Dati - ASN.1

- Chiavi e parametri utilizzati da OpenSSL si basano su ASN.1
  - Abstract Syntax Notation One (ASN.1)
- ASN.1 è un modo per descrivere i dati
  - Partendo da tipi primitivi e costruendo tipi più complessi
- ASN.1 specifica tipo e struttura dei dati
  - Tipo di valore
    - Intero, booleano, stringa di caratteri, etc.
  - Struttura
    - Contenimento, ordine, opzioni
- ASN.1 non specifica la codifica (rappresentazione) dei dati

# Cifratura Asimmetrica in OpenSSL

## PKCS

- Public Key Cryptography Standards (PKCS)
- Insieme di standard sviluppati e pubblicati da RSA Security Inc
- Descrivono, mediante ASN.1, gli oggetti coinvolti in operazioni crittografiche
- Ne esistono vari, ad es.,
  - **PKCS #1** RSA Cryptography Standard
  - **PKCS #3** Diffie-Hellman Key Agreement Standard
  - **PKCS #5** Password-based Encryption Standard
  - **PKCS #7** Cryptographic Message Syntax Standard
  - **PKCS #8** Private-Key Information Syntax Standard
  - **PKCS #12** Personal Information Exchange Syntax Standard

# ASN.1

## Chiave privata RSA in PKCS #1 - Esempio

```
RSAPrivateKey ::= SEQUENCE {
    version      Version,
    modulus      INTEGER, -- n
    publicExponent INTEGER, -- e
    privateExponent INTEGER, -- d
    prime1       INTEGER, -- p
    prime2       INTEGER, -- q
    exponent1    INTEGER, -- d mod (p-1)
    exponent2    INTEGER, -- d mod (q-1)
    coefficient   INTEGER, -- (inverse of q) mod p
    otherPrimeInfos OtherPrimeInfos OPTIONAL
}

Version ::= INTEGER { two-prime(0), multi(1) } (CONSTRAINED BY {
    -- version must be multi if otherPrimeInfos present --
})

OtherPrimeInfos ::= SEQUENCE SIZE(1..MAX) OF OtherPrimeInfo

OtherPrimeInfo ::= SEQUENCE {
    prime      INTEGER, -- ri
    exponent   INTEGER, -- di
    coefficient INTEGER -- ti
}
```

# Cifratura Asimmetrica in OpenSSL

## Codifica dei Dati

- ASN.1 è relativo ad un insieme di regole di codifica che specificano come rappresentare i suoi dati
- Regole sono caratterizzate da uno specifico formato
  - Basic Encoding Rules (BER)
  - Canonical Encoding Rules (CER)
  - Distinguished Encoding Rules (DER)
  - Privacy Enhanced Mail (PEM)
  - Etc.
- Progettate inizialmente per la trasmissione seriale
  - Dove i dati sono inviati un bit alla volta, in modo sequenziale



# Cifratura Asimmetrica in OpenSSL

## Codifica dei Dati - DER vs. PEM

- Distinguished Encoding Rules (DER)
  - Codifica in binario i dati descritti mediante ASN.1
  - Ogni file può contenere un solo oggetto codificato in DER
- Privacy Enhanced Mail (PEM)
  - Formato di default usato da OpenSSL
  - Usato per codificare in formato testuale (Base64) oggetti codificati tramite DER
  - Può contenere commenti
  - Ogni file può contenere oggetti multipli codificati in PEM
    - Uno sotto l'altro
  - Sintassi

```
-----BEGIN keyword-----  
Base 64 encoding of BER/DER  
-----END keyword-----
```

# Cifratura Asimmetrica in OpenSSL

## Codifica dei Dati - DER vs. PEM

- Distinguished Encoding Rules (DER)
  - Sottoinsieme di BER
  - Codifica in binario i dati descritti mediante ASN.1
  - Ogni file può contenere un solo oggetto codificato in DER
- Privacy Enhanced Mail (PEM)
  - Formato di default usato da OpenSSL
  - Usato per codificare in formato testuale (Base64) oggetti codificati tramite DER
  - Può contenere commenti
  - Ogni file può contenere oggetti codificati
  - Uno sotto l'altro
  - Sintassi

keyword può essere una chiave pubblica, una chiave privata, etc

```
-----BEGIN keyword-----  
Base 64 encoding of BER/DER  
-----END keyword-----
```

# Cifratura Asimmetrica in OpenSSL

## Codifica dei Dati - DER vs. PEM

- Distinguished Encoding Rules (DER)
  - Sottoinsieme di BER
  - Codifica in binario i dati descritti mediante ASN.1
  - Ogni file può contenere un solo oggetto codificato in DER
- Privacy Enhanced Mail (PEM)
  - Formato di default usato da OpenSSL
  - Usato per codificare in formato testuale (Base64) oggetti codificati tramite DER
  - Può contenere commenti
  - Ogni file può contenere
    - Uno sotto l'altro
  - Sintassi

**keyword** può essere una chiave pubblica, una chiave privata, etc

PEM

```
-----BEGIN keyword-----  
Base 64 encoding of BER/DER  
-----END keyword-----
```

# Cifratura Asimmetrica in OpenSSL

## Codifica dei Dati - DER vs. PEM

### ➤ Distinguished Encoding Rules (DER)

- Sottoinsieme di BER
- Codifica in binario i dati descritti mediante ASN.1
- Ogni file può contenere un solo oggetto codificato in DER

### ➤ Privacy Enhanced Mail (PEM)

- Formato di default usato da OpenSSL
- Usato per codificare in formato testuale (Base64) oggetti codificati tramite DER
- Può contenere commenti
- Ogni file può contenere oggetti multipli codificati in PEM
  - Uno sotto l'altro
- Sintassi

```
-----BEGIN keyword-----  
Base 64 encoding of BER/DER  
-----END keyword-----
```

È possibile convertire un formato nell'altro e viceversa

# Cifratura Asimmetrica in OpenSSL

## Codifica dei Dati - DER vs. PEM

### ➤ Distinguished Encoding Rules (DER)

- Sottoinsieme di BER
- Codifica in binario i dati descritti mediante ASN.1
- Ogni file può contenere un solo oggetto codificato in DER

È possibile convertire un formato nell'altro e viceversa

### ➤ Privacy Enhanced Mail (PEM)

- Formato di default usato da OpenSSL
- Usato per codificare in formato testuale (Base64) oggetti codificati tramite DER
- Può contenere commenti
- Ogni file può contenere oggetti multipli codificati in PEM
  - Uno sotto l'altro
- Sintassi

```
-----BEGIN keyword-----  
Base 64 encoding of BER/DER  
-----END keyword-----
```

# Cifratura Asimmetrica in OpenSSL

## Codifica dei Dati - DER vs. PEM

### ➤ Distinguished Encoding Rules (DER)

- Sottoinsieme di BER
- Codifica in binario i dati descritti mediante ASN.1
- Ogni file può contenere un solo oggetto codificato in DER

Entrambe le codifiche sono supportate da OpenSSL

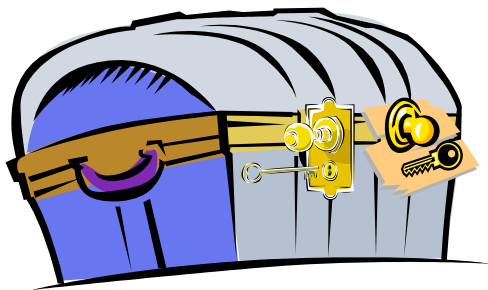
### ➤ Privacy Enhanced Mail (PEM)

- Formato di default usato da OpenSSL
- Usato per codificare in formato testuale (Base64) oggetti codificati tramite DER
- Può contenere commenti
- Ogni file può contenere oggetti multipli codificati in PEM
  - Uno sotto l'altro
- Sintassi

```
-----BEGIN keyword-----  
Base 64 encoding of BER/DER  
-----END keyword-----
```

# Cifratura RSA in OpenSSL

- OpenSSL fornisce
  - I comandi `genrsa` ed `rsa` per generare, esaminare, manipolare ed usare chiavi RSA
  - Il comando `rsautl` per cifrare (e firmare) mediante RSA



# Cifratura RSA in OpenSSL

## Generazione Chiavi

- La coppia di chiavi RSA può essere generata in OpenSSL mediante il comando `genrsa`

### Opzioni principali del comando `genrsa`

```
openssl genrsa [options] [numbits]
```

#### ➤ **options**

- `-des, -des3, -aes128, -aes192, -aes256...` Cifra le chiavi generate, utilizzando DES o 3DES, oppure utilizzando AES a 128, 192 o 256 bit, etc
- `-out file` Scrive in un file le chiavi generate
- `-passout arg` Password usata per la cifratura delle chiavi
- `-passin arg` Password usata per la decifratura delle chiavi
- `-F4, -3` Esponente pubblico: in OpenSSL può essere 65537 oppure 3. Il valore di default è 65537 e non sono ammessi altri valori

#### ➤ **numbits**

- Numero di bit del modulo RSA, di default è 2048



# Cifratura RSA in OpenSSL

## Generazione Chiavi

- La coppia di chiavi RSA può essere generata in OpenSSL mediante il comando `genrsa`

### Opzioni principali del comando `genrsa`

```
openssl genrsa [options] [numbits]
```

- **options**

- `-des, -des3, -aes128, -aes192, -aes256...` Cifra le chiavi generate, utilizzando DES o 3DES, oppure utilizzando AES a 128, 192 o 256 bit
- `-out file` Scrive in un file le chiavi generate
- `-passout arg` Password usata per la cifratura delle chiavi
- `-passin arg` Password usata per la decifratura delle chiavi
- `-F4, -3` Esponente pubblico: in OpenSSL può essere 65537 oppure 3. Il valore di default è 65537 e non sono ammessi altri valori

- **numbits**

- Numero di bit del modulo RSA, di default è 2048

# Cifratura RSA in OpenSSL

## Generazione Chiavi

- La coppia di chiavi RSA può essere generata in OpenSSL mediante il comando `genrsa`

### Opzioni principali del comando `genrsa`

`openssl genrsa [options] [numbits]`

- **options**

- `-des, -des3, -aes128,` utilizzando DES o 3DES, o 256 bit, etc
- `-out file` Scrive in un file
- `-passout arg Password`
- `-passin arg Password` usata per la decifratura delle chiavi
- `-F4, -3` Esponente pubblico: in OpenSSL può essere 65537 oppure 3. Il valore di default è 65537 e non sono ammessi altri valori

- **numbits**

- Numero di bit del modulo RSA, di default è 2048

Per ottenere la lista completa delle opzioni del comando `genrsa` è possibile utilizzare `man genrsa`

# Cifratura RSA in OpenSSL

## Esempio Generazione Chiavi

- Mediante il seguente comando è possibile generare una coppia di chiavi RSA a 1024 bit
  - Tale coppia sarà cifrata attraverso AES a 128 bit, usando la stringa "P1pp0B4ud0" come password. Il risultato verrà scritto nel file `rsaprivatekey.pem`

```
openssl genrsa -out rsaprivatekey.pem -passout  
pass:P1pp0B4ud0 -aes128 1024
```

- Se la generazione andrà a buon fine, verrà mostrato il seguente messaggio

```
Generating RSA private key, 1024 bit long modulus  
.....++++++  
.....++++++  
e is 65537 (0x10001)
```

# Cifratura RSA in OpenSSL

## Esempio Generazione Chiavi

- È possibile visualizzare il contenuto del file `rsaprivatekey.pem` mediante il seguente comando

```
openssl rsa -in rsaprivatekey.pem -text
```

N.B. Per visualizzare il contenuto `rsaprivatekey.pem` è necessario conoscere la sua password di cifratura

Le chiavi RSA in OpenSSL sono rappresentate secondo lo standard PKCS #1 (RFC3447)

- <https://tools.ietf.org/html/rfc3447>

# Cifratura RSA in OpenSSL

## Esempio Generazione Chiavi

- È possibile visualizzare il contenuto di `rsaprivatekey.pem` mediante il seguente comando:

Stampa in formato testuale le varie componenti della chiave pubblica o privata, oltre alla versione codificata in formato PEM di tali componenti

```
openssl rsa -in rsaprivatekey.pem -text
```

N.B. Per visualizzare il contenuto `rsaprivatekey.pem` è necessario conoscere la sua password di cifratura

Le chiavi RSA in OpenSSL sono rappresentate secondo lo standard PKCS #1 (RFC3447)

- <https://tools.ietf.org/html/rfc3447>

# Cifratura RSA in OpenSSL

## Esempio Generazione Chiavi

- È possibile visualizzare il contenuto del file `rsaprivatekey.pem` mediante il seguente comando

```
openssl rsa -in rsaprivatekey.pem -text
```

N.B. Per visualizzare il contenuto del file `rsaprivatekey.pem` è necessario conoscere la sua password

Per ottenere la lista completa delle opzioni del comando `rsa` è possibile utilizzare  
`man rsa`

Le chiavi RSA in OpenSSL sono rappresentate secondo lo standard PKCS #1 (RFC3447)

- <https://tools.ietf.org/html/rfc3447>

Enter pass phrase for rsaprivatekey.pem:

Private-Key: (1024 bit)

modulus:

00:b6:d3:9a:24:04:87:57:db:88:d4:5e:19:f2:6e:  
31:2d:be:88:3f:78:a4:5c:59:1d:62:aa:fa:a5:ee:  
24:56:f9:9c:57:65:8c:92:66:a7:99:4b:31:0e:45:  
10:65:b3:c4:76:23:e1:66:ca:34:ba:59:6c:d7:38:  
ce:6c:20:e6:d7:59:be:92:c5:0f:87:0e:29:d2:18:  
fb:d4:3e:f9:8c:f1:09:89:c2:b5:fd:d8:6c:64:fa:  
a8:3e:cf:11:60:b3:fa:c0:97:a3:65:41:5a:d1:a5:  
e6:b4:a4:12:e0:11:c6:45:84:23:e9:2f:ee:52:b9:  
f5:53:69:98:82:73:4e:e1:01

publicExponent: 65537 (0x10001)

privateExponent:

79:15:33:64:df:4a:f3:b9:05:aa:00:98:96:a2:bc:  
17:5b:b4:c8:b4:91:c6:47:8e:da:a5:fa:52:a2:a1:  
96:a8:c2:0e:6f:c6:e5:be:ef:08:eb:6f:95:cf:cf:  
c8:4b:82:b5:eb:95:80:8a:32:d9:a8:18:19:82:19:  
d4:2b:06:36:42:3a:47:1b:e1:80:7d:54:9c:97:f5:  
1f:08:ac:cb:ff:7a:a3:42:51:d5:dd:1e:fa:33:bc:  
50:25:89:3d:42:18:8e:d5:f8:e5:2c:0c:d8:19:98:  
7e:50:94:ae:d4:05:95:ba:9c:9e:87:5f:10:52:50:  
33:14:e8:7b:3e:06:0b:f1

prime1:

00:eb:50:2b:07:f5:0c:9e:77:3c:35:d9:6c:fe:fd:  
51:24:3a:ff:cf:87:15:17:41:f9:cb:17:00:37:4b:  
1f:d6:5c:e7:ce:a2:13:36:a0:e8:f6:47:d8:78:61:  
4e:39:7d:63:2e:31:94:bd:e5:95:30:36:86:53:33:  
fd:e5:3c:bf:13

prime2:

00:c6:e6:32:f3:32:6c:19:5d:ec:cc:43:6a:36:44:  
29:9b:d6:54:ee:b1:ba:36:8e:56:07:f6:f3:9b:0b:  
ab:45:57:58:f8:85:89:68:4e:11:52:37:13:fb:d4:  
ad:b4:05:5a:47:1d:e5:ea:64:84:27:4a:a5:e4:e6:  
09:e2:43:9e:1b

exponent1:

39:9f:e3:39:ca:9f:b1:7a:bf:8a:ec:58:cd:c4:d9:  
07:07:4f:b7:d1:7b:af:bb:5f:61:85:9b:6e:ee:fd:  
b7:ef:fe:70:52:ae:63:d5:59:d7:5d:d6:bb:fc:10:  
3c:f1:e0:c4:e2:2d:6b:a7:7a:36:59:53:e1:b2:3c:  
bd:d3:77:73

exponent2:

0f:f7:78:cd:97:15:a0:6f:ae:cb:b2:f9:ba:c3:7c:  
07:9b:8c:13:e1:46:8e:8c:9c:91:65:1c:a7:2c:a7:  
a2:18:61:f7:09:59:3f:7b:4c:de:a9:b7:3b:f7:15:  
be:a6:d3:59:74:27:f9:c9:f9:e5:e1:93:31:ad:d9:  
cb:45:1c:53

coefficient:

56:39:ae:68:3f:99:f7:26:c0:e3:00:3c:95:2f:42:  
f1:b2:f9:14:c7:65:7a:15:f0:04:f2:4b:25:15:18:  
cc:64:27:07:58:47:7e:27:a8:9a:aa:ab:2e:22:57:  
68:e0:47:a4:68:54:23:cb:7d:78:46:01:4e:0f:71:  
60:ea:cb:04

Modulo n di 1024 bit

Esponente pubblico e

Esponente privato d

Primo p

Primo q

$d_p = d \bmod (p - 1)$

$d_q = d \bmod (q - 1)$

$q_{inv} = q^{-1} \bmod p$

Contenuto del file

rsaprivatekey.pem – 1/2

Chiave privata RSA, espressa  
secondo la notazione definita  
da PKCS #1

Formula di Garner  
per la decifratura

Calcolo di  $c^d \bmod n$

$m_1 = c^{d_p} \bmod p$

$m_2 = c^{d_q} \bmod q$

$h = q_{inv} \cdot (m_1 - m_2) \bmod p$

$m = m_2 + h \cdot q$



```
writing RSA key
-----BEGIN RSA PRIVATE KEY-----
MIICWwIBAAKBgQC205okBIIdX24jUXhnybjEtvog/eKRcWR1iqvql7iRW+ZxXZYyS
ZqeZSzEORRBls8R2I+FmyjS6WWzX0M5sI0bXWb6SxQ+HDinSGPvUPvmM8QmJwrX9
2Gxk+qg+zxFgs/rAl6NlQVrRpea0pBLgEcZFhCPpL+5SufVTaZiCc07hAQIDAQAB
AoGAeRUzZN9K87kFqgCYlqK8F1u0yLSRxke02qX6UqKhIqjCDm/G5b7vC0tvlc/P
yEuCteuVgIoy2agYGYIZ1CsGNkI6RxvhgH1UnJf1Hwisy/96o0JR1d0e+j08UCWJ
PUIYjtX45SwM2BmYfLCUrtQFlbqcnodfEFJQMxToez4GC/ECQQDrUCsH9Qyedzw1
2Wz+/VEk0v/PhxUXQfnLFwA3Sx/WX0f0ohM2o0j2R9h4YU45fWMuMZS95ZUwNoZT
M/3lPL8TAkEAxuYy8zJsGV3szENqNkQpm9ZU7rG6No5WB/bzmwurRVdY+IWJaE4R
UjcT+9SttAVaRx3l6mSEJ0ql50YJ4k0eGwJA0Z/j0cqfsXq/iuxYzcTZBwdPt9F7
r7tfYYWbbu79t+/+cFKuY9VZ113Wu/wQPPHgX0Ita6d6NlLT4bI8vdN3cwJAD/d4
zZcVoG+uy7L5usN8B5uME+FGjoyckWUcPyynohhh9wLZP3tM3qm30/cVvqbTWXQn
+cn55eGTMa3Zy0UcUwJAVjmuaD+Z9ybA4wA8lS9C8bL5FMdleHxwBPJLJRUYzGQn
B1hHfieomqqrLiJXa0BHpGhUI8t9eEYBTg9xY0rLBA==
-----END RSA PRIVATE KEY-----
```

Codifica PEM della chiave  
privata RSA



# Cifratura RSA in OpenSSL

## Esempio Generazione Chiavi

- Mediante il seguente comando è possibile estrarre la chiave pubblica RSA memorizzata nel file `rsaprivatekey.pem`
  - Decifrando il contenuto di tale file attraverso la password **"P1pp0B4ud0"** e scrivendo la chiave pubblica nel file `rsapublickey.pem`

```
openssl rsa -in rsaprivatekey.pem -passin pass:P1pp0B4ud0  
-pubout -out rsapublickey.pem
```

- Per visualizzare il contenuto del file `rsapublickey.pem` è possibile utilizzare il seguente comando

```
openssl rsa -pubin -in rsapublickey.pem -text
```

# Cifratura RSA in OpenSSL

## Contenuto del file rsapublickey.pem

Modulus (1024 bit):

00:b6:d3:9a:24:04:87:57:db:88:d4:5e:19:f2:6e:  
31:2d:be:88:3f:78:a4:5c:59:1d:62:aa:fa:a5:ee:  
24:56:f9:9c:57:65:8c:92:66:a7:99:4b:31:0e:45:  
10:65:b3:c4:76:23:e1:66:ca:34:ba:59:6c:d7:38:  
ce:6c:20:e6:d7:59:be:92:c5:0f:87:0e:29:d2:18:  
fb:d4:3e:f9:8c:f1:09:89:c2:b5:fd:d8:6c:64:fa:  
a8:3e:cf:11:60:b3:fa:c0:97:a3:65:41:5a:d1:a5:  
e6:b4:a4:12:e0:11:c6:45:84:23:e9:2f:ee:52:b9:  
f5:53:69:98:82:73:4e:e1:01

Modulo n di 1024 bit

Exponent: 65537 (0x10001)

Esponente pubblico e

writing RSA key

-----BEGIN PUBLIC KEY-----

MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQC205okBI dX24jUXhnybjEtvog/  
eKRcWR1iqvql7iRW+ZxXZYySZqeZSzEORRBl s8R2I+FmyjS6WWzXOM5sIObXWb6S  
xQ+HDinSGPvUPvmM8QmJwrX92Gxk+qg+zxFgs/rAl6NLQVrRpea0pBLgEcZFhCPp  
L+5SufVTaZiCc07hAQIDAQAB

-----END PUBLIC KEY-----

Chiave pubblica RSA  
codificata in Base64

# Cifratura RSA in OpenSSL

## Esempio Generazione Chiavi

- Mediante il seguente comando è possibile estrarre la chiave pubblica RSA memorizzata nel file `rsaprivatekey.pem`
  - Decifrando il contenuto di tale file attraverso la password **"P1pp0B4ud0"** e scrivendo la chiave pubblica nel file `rsapublickey.pem`

```
openssl rsa -in rsaprivatekey.pem -pubout -out rsapublickey.pem
```

**N.B.** L'opzione `-pubin` deve essere sempre inclusa nel comando OpenSSL ogniqualvolta il file preso in input sia una chiave pubblica

- Di default OpenSSL si aspetta di ricevere in input una chiave privata

- Per visualizzare il contenuto della chiave pubblica in `rsapublickey.pem` è possibile utilizzare il seguente comando

```
openssl rsa -pubin -in rsapublickey.pem -text
```

# Cifratura RSA in OpenSSL

## Esempio Generazione Chiavi

- Mediante il seguente comando è possibile estrarre la chiave pubblica RSA memorizzata nel file `rsaprivatekey.pem`
  - Decifrando il contenuto di tale file attraverso la password "P1pp0B4ud0" e scrivendo la chiave pubblica nel file `rsapublickey.pem`

```
openssl  
-pubout
```

```
unable to load Private Key  
139762194448088:error:0906D06C:PEM routines:PEM_read_bio:no  
start line:pem_lib.c:701:Expecting: ANY PRIVATE KEY
```

- Per visualizzare il contenuto del file `rsapublickey.pem` è possibile utilizzare il seguente comando

```
openssl rsa -in rsapublickey.pem -text
```

N.B. Se si prova a leggere una chiave pubblica RSA non specificando l'opzione `-pubin` viene restituito un errore

# Cifratura RSA in OpenSSL

## Il comando `rsautl`

- Il comando `rsautl` consente di usare le chiavi RSA per la cifratura
  - Permette di specificare opzioni per cifrare e decifrare i dati
- Non può essere usato per cifrare grosse moli di dati
  - Per tale scopo va invece usata la cifratura simmetrica (comando `enc`)
- Cifratura e decifratura RSA sono onerose

# Cifratura RSA in OpenSSL

## Opzioni Principali del Comando `rsautl`

```
openssl rsautl [options]
```

### ➤ options

- `-in file` File di input
- `-out file` File di output
- `-encrypt` Cifra con la chiave pubblica
- `-decrypt` Decifra con la chiave privata
- `-sign` Firma con la chiave privata
- `-verify` Verifica con la chiave pubblica
- `-inkey` Chiave presa in input
- `-passin arg` Sorgente da cui deve essere letta la password
- `-pubin` Specifica che l'input è una chiave pubblica RSA
- `-pkcs`, `-oaep`, `-ssl`, `-raw` Padding da usare: PKCS#1 v1.5 (default), PKCS#1 OAEP (Optimal Asymmetric Encryption Padding), modalità speciale di padding usata in SSL v2, nessun padding, rispettivamente

# Cifratura RSA in OpenSSL

## Opzioni Principali del Comando `rsautl`

```
openssl rsautl [options]
```

### ➤ options

- `-in file` File di input
- `-out file` File di output
- `-encrypt` Cifra con la chiave pubblica
- `-decrypt` Decifra con la chiave privata
- `-sign` Firma con la chiave privata
- `-verify` Verifica con la chiave pubblica
- `-inkey` Chiave presa in input
- `-passin arg` Sorgente da cui deve essere letta la password
- `-pubin` Specifica che l'input è una chiave pubblica RSA
- `-pkcs`, `-oaep`, `-ssl`, `-raw` Padding da usare: PKCS#1 v1.5 (default), PKCS#1 OAEP (Optimal Asymmetric Encryption Padding), modalità speciale di padding usata in SSL v2, nessun padding, rispettivamente

Li vedremo nelle prossime lezioni,  
quando verrà trattata la  
firma RSA

# Cifratura RSA in OpenSSL

## Opzioni Principali del Comando `rsautl`

`openssl rsautl [options]`

### ➤ **options**

- `-in file` File di input
- `-out file` File di output
- `-encrypt` Cifra con la chiave privata
- `-decrypt` Decifra con la chiave privata
- `-sign` Firma con la chiave privata
- `-verify` Verifica con la chiave pubblica
- `-inkey` Chiave presa in input
- `-passin arg` Sorgente da cui deve essere letta la password
- `-pubin` Specifica che l'input è una chiave pubblica RSA
- `-pkcs`, `-oaep`, `-ssl`, `-raw` Padding da usare: PKCS#1 v1.5 (default), PKCS#1 OAEP (Optimal Asymmetric Encryption Padding), modalità speciale di padding usata in SSL v2, nessun padding, rispettivamente

Per ottenere la lista completa delle opzioni del comando `rsautl` è possibile utilizzare `man rsautl`



# Cifratura RSA in OpenSSL

## Esempio di Cifratura e Decifratura

`testoInChiaro.txt`

Benvenuti al corso di Sicurezza, A.A. 2019/2020!

- Usando la chiave pubblica contenuta in `rsapublickey.pem` il contenuto di `testoInChiaro.txt` è cifrato e scritto in `testoCifrato.txt`

```
openssl rsautl -encrypt -pubin -inkey rsapublickey.pem  
-in testoInChiaro.txt -out testoCifrato.txt
```

- Usando la chiave privata contenuta in `rsaprivatekey.pem` il contenuto di `testoCifrato.txt` è decifrato e scritto in `testoDecifrato.txt`

```
openssl rsautl -decrypt -inkey rsaprivatekey.pem  
-in testoCifrato.txt -out testoDecifrato.txt
```

# Cifratura RSA in OpenSSL

## Dimensione dei Dati da Cifrare

- RSA permette di cifrare dati la cui dimensione è al più pari a quella della chiave (modulo  $n$ )
- Nel caso in cui venga utilizzata cifratura con padding, assumendo l'utilizzo di una chiave di  $X$  bit, possono essere cifrati
  - $(X / 8) - 11 = Y$  Byte, se la modalità di padding è PKCS #1 v1.5
    - <https://www.ietf.org/rfc/rfc2313.txt>
  - $(X / 8) - 42 = Y$  Byte, se la modalità di padding è OAEP
    - <https://tools.ietf.org/html/rfc3447#section-7.1>

# Cifratura RSA in OpenSSL

## Dimensione dei Dati da Cifrare

- Se si prova a cifrare un file di dimensioni non conformi rispetto a quelle stabilite dal modulo e dalla modalità di padding si ottiene un errore
- Esempio
  - Supponiamo di cifrare il file `bigFile.tar.gz`, di 22952 bit, usando PKCS#1 v1.5 come modalità di padding

```
openssl rsautl -encrypt -pubin -inkey rsapublickey.pem  
-in bigFile.tar.gz -out testoCifrato.txt
```



```
RSA operation error  
140102711398104:error:0406D06E:rsa  
routines:RSA_padding_add_PKCS1_type_2:data too large for key  
size:rsa_pk1.c:153:
```

# Cifratura RSA in OpenSSL

## Esempio di Recupero Chiave Privata RSA

- È possibile recuperare una chiave privata RSA a partire dalla relativa chiave pubblica (di piccole dimensioni)?



# Cifratura RSA in OpenSSL

## Esempio di Recupero Chiave Privata RSA

- Libreria da installare
  - ASN.1 library for Python
    - `sudo apt-get install python-pyasn1 python3-pyasn1 python-pyasn1-modules python3-pyasn1-modules pypy-pyasn1`
- Strumenti necessari
  - *YAFU - Automated integer factorization*
    - <https://sourceforge.net/projects/yafu/>
  - *genPriv.py* - Script *Python* per la codifica di chiavi in formato PKCS1



# Cifratura RSA in OpenSSL

## Esempio di Recupero Chiave Privata RSA

- Generiamo una chiave RSA di 50 bit, estraiamo da essa la relativa chiave pubblica e ne stampiamo il contenuto

```
openssl genrsa 50 > smallkey.pem  
openssl rsa -pubout -in smallkey.pem > smallkey_pub.pem  
openssl rsa -in smallkey_pub.pem -pubin -text -modulus
```



```
Modulus (50 bit): 836321605190581 (0x2f8a14c3203b5)  
Exponent: 65537 (0x10001)  
Modulus=2F8A14C3203B5  
writing RSA key  
-----BEGIN PUBLIC KEY-----  
MCIwDQYJKoZIhvcNAQEBBQADEQAwDgIHAVihTDIDtQIDAQAB  
-----END PUBLIC KEY-----
```

# Cifratura RSA in OpenSSL

## Esempio di Recupero Chiave Privata RSA

- Generiamo una chiave RSA di 50 bit, estraiamo da essa la relativa chiave pubblica e ne stampiamo il contenuto

```
openssl genrsa 50 > smallkey.pem  
openssl rsa -pubout -in smallkey.pem > smallkey_pub.pem  
openssl rsa -in smallkey_pub.pem -pubin -text -modulus
```

```
Modulus (50 bit):  
Exponent: 65537 (0  
Modulus=2F8A14C320555
```

```
writing RSA key
```

```
-----BEGIN PUBLIC KEY-----
```

```
MCIwDQYJKoZIhvcNAQEBBQADEQAwDgIHAvihTDIDtQIDAQAB
```

```
-----END PUBLIC KEY-----
```

- N.B. OpenSSL 1.1.1 consente di generare chiavi aventi lunghezza minima pari a 512 bit
  - Per generare chiavi di lunghezza inferiore è necessario utilizzare versioni precedenti di OpenSSL

# Cifratura RSA in OpenSSL

## Esempio di Recupero Chiave Privata RSA

- Generiamo una chiave RSA di 50 bit, estraiamo da essa la relativa chiave pubblica e ne stampiamo il contenuto

```
openssl genrsa 50 > smallkey.pem  
openssl rsa -pubout -in smallkey.pem > smallkey_pub.pem  
openssl rsa -in smallkey_pub.pem -pubin -text -modulus
```



Modulus (50 bit): 836321605190581 (0x2f8a14c3203b5)

Exponent: 65537 (0x10001) Base 10

Modulus=2F8A14C3203B5

wri Modulo RSA di 50 bit

--- 10111110001010000101001100001100100000001110110101

MCiWDQYJKOZiNVCNAQEBBQADEQAWDgIHAViNTDIDtQIDAQAB Base 2

-----END PUBLIC KEY-----



# Cifratura RSA in OpenSSL

## Esempio di Recupero Chiave Privata RSA

- Generiamo una chiave RSA di 50 bit, estraiamo da essa la relativa chiave pubblica e ne stampiamo il contenuto

```
openssl genrsa  
openssl rsa -  
openssl rsa -
```

- È possibile convertire in decimale il valore esadecimale del modulo n, mediante vari strumenti

- Ad es. tramite convertitori online

<http://www.rapidtables.com/convert/number/hex-to-decimal.htm>

Modulus (50 bit): 321605190581 (0x2f8a14c3203b5)

Exponent: 65537 (0x10001)

Modulus=2F8A14C3203B5

writing RSA key

Base 16

-----BEGIN PUBLIC KEY-----

MCiwdQYJKoZIhvcNAQEBBQADEQAwDgIHAVihTDIDtQIDAQAB

-----END PUBLIC KEY-----

# Cifratura RSA in OpenSSL

## Esempio di Recupero Chiave Privata RSA

1) Fattorizziamo il modulo  $n$  (836321605190581) per trovare i relativi fattori primi  $p$  e  $q$

- Considereremo due metodi alternativi per farlo
  - Metodo Online, utilizzando <http://factordb.com/>
  - Metodo Offline, utilizzando YAFU

# Cifratura RSA in OpenSSL

Esempio di Recupero Chiave Privata RSA -  
factordb.com

Numero che  
si intende  
fattorizzare

[Sequences](#)

[Recent results](#)

[Factor tables](#)

[Status](#)

836321605190581

Factorize!

Result:

digits

number

15 [\(show\)](#)

836321605190581

$\times_{15} =$

27346349

30582569

$n$

$p$

$q$

More information 

ECM 

# Cifratura RSA in OpenSSL

## Esempio di Recupero Chiave Privata RSA - YAFU

- Consente la fattorizzare automatica di numeri interi presi in input
- Utilizza alcuni tra i più potenti algoritmi di fattorizzare
  - Special Number Field Sieve (SNFS)
  - General Number Field Sieve (GNFS)
  - Self-Initializing Quadratic Sieve (SIQS)
  - Elliptic Curve factorization Method (ECM)
  - Multiple Polynomial Quadratic Sieve (MPQS)
  - Etc.
- Combina gli algoritmi di fattorizzazione in modo dinamico e adattivo
  - Per minimizzare i tempi necessari a trovare i fattori primi
- L'implementazione di molti algoritmi è multi-thread
  - Permette di sfruttare processori multi-core o many-core

# Cifratura RSA in OpenSSL







## Esempio di Recupero Chiave Privata RSA - YAFU

### ➤ Scaricare YAFU

Looking for the latest version? [Download yafu-1.34.zip \(4.1 MB\)](#)

[Home](#) / 1.34



Name ↕	Modified ↕	Size ↕	Downloads / Week ↕
<a href="#">↑ Parent folder</a>			
yafu-1.34.zip	<a href="#">2013-03-06</a>	4.1 MB	139  
yafu-1.34-src.zip	<a href="#">2013-02-27</a>	686.7 kB	12  
<a href="#">README.txt</a>	<a href="#">2013-02-25</a>	2.2 kB	0  

➤ `unzip yafu-1.34.zip`

➤ `chmod +x yafu`

# Cifratura RSA in OpenSSL

## Esempio di Recupero Chiave Privata RSA - YAFU

### ➤ Utilizzo di YAFU

```
$ echo "factor(836321605190581)" | ./yafu
```

```
fac: factoring 836321605190581
fac: using pretesting plan: normal
fac: no tune info: using qs/gnfs cro digits
div: primes less than 10000
rho: x^2 + 3, starting 1000 iterations on C15
rho: x^2 + 2, starting 1000 iterations on C15
Total factoring time = 0.0036 seconds
```

Numero da  
fattorizzare

```
***factors found***
```

```
P8 = 27346349 p
```

```
P8 = 30582569 q
```

```
ans = 1
```

# Cifratura RSA in OpenSSL

## Esempio di Recupero Chiave Privata RSA - genPriv.py

2) Dopo aver ottenuto il valore dei numeri primi  $p$  e  $q$ , possiamo ricostruire la chiave privata e codificarla in PKCS#1

➤ Mediante il programma `genPriv.py`

```
$ python2 genPriv.py
<<I valori di p, q ed e vanno inseriti in formato decimale>>
Inserisci p >> 27346349
Inserisci q >> 30582569
Inserisci e >> 65537
Inserire il nome del file dove memorizzare la chiave privata
>> smallkey_priv.pem
```

# Cifratura RSA in OpenSSL

## Esempio di Recupero Chiave Privata RSA

3) Visualizziamo la chiave privata ricostruita

```
openssl rsa -in rsaprivatekey.pem -text
```

```
Private-Key: (50 bit)
modulus: 836321605190581 (0x2f8a14c3203b5)
publicExponent: 65537 (0x10001)
privateExponent: 456450332239841 (0x19f239fe927e1)
prime1: 27346349 (0x1a145ad)
prime2: 30582569 (0x1d2a729)
exponent1: 22529849 (0x157c739)
exponent2: 30560169 (0x1d24fa9)
coefficient: 25370811 (0x18320bb)
writing RSA key
-----BEGIN RSA PRIVATE KEY-----
MDgCAQACBwL4oUwyA7UCAwEAAQIHAZ8jn+kn4QIEAaFFrQIEAdKnKQIEAVfHOQIE
AdJPqQIEAYMguw==
-----END RSA PRIVATE KEY-----
```



# Cifratura RSA in OpenSSL

## Generazione Chiavi RSA - Prestazioni

- Configurazione Hardware
  - Processore: Intel Core i7 2,8GHz a 64 bit
  - RAM: 4GB
  - Hard Disk: Apple SSD SM1024G

Dimensione Chiave (in bit)	Tempo Richiesto
1024	0m0.028s
2048	0m0.177s
4096	0m0.361s
8192	0m16.171s
16384	1m50.871s
32768	28m41.811s

Tempi restituiti dal comando **time**

# Cifratura RSA in OpenSSL

## Fattorizzazione Modulo RSA - Prestazioni

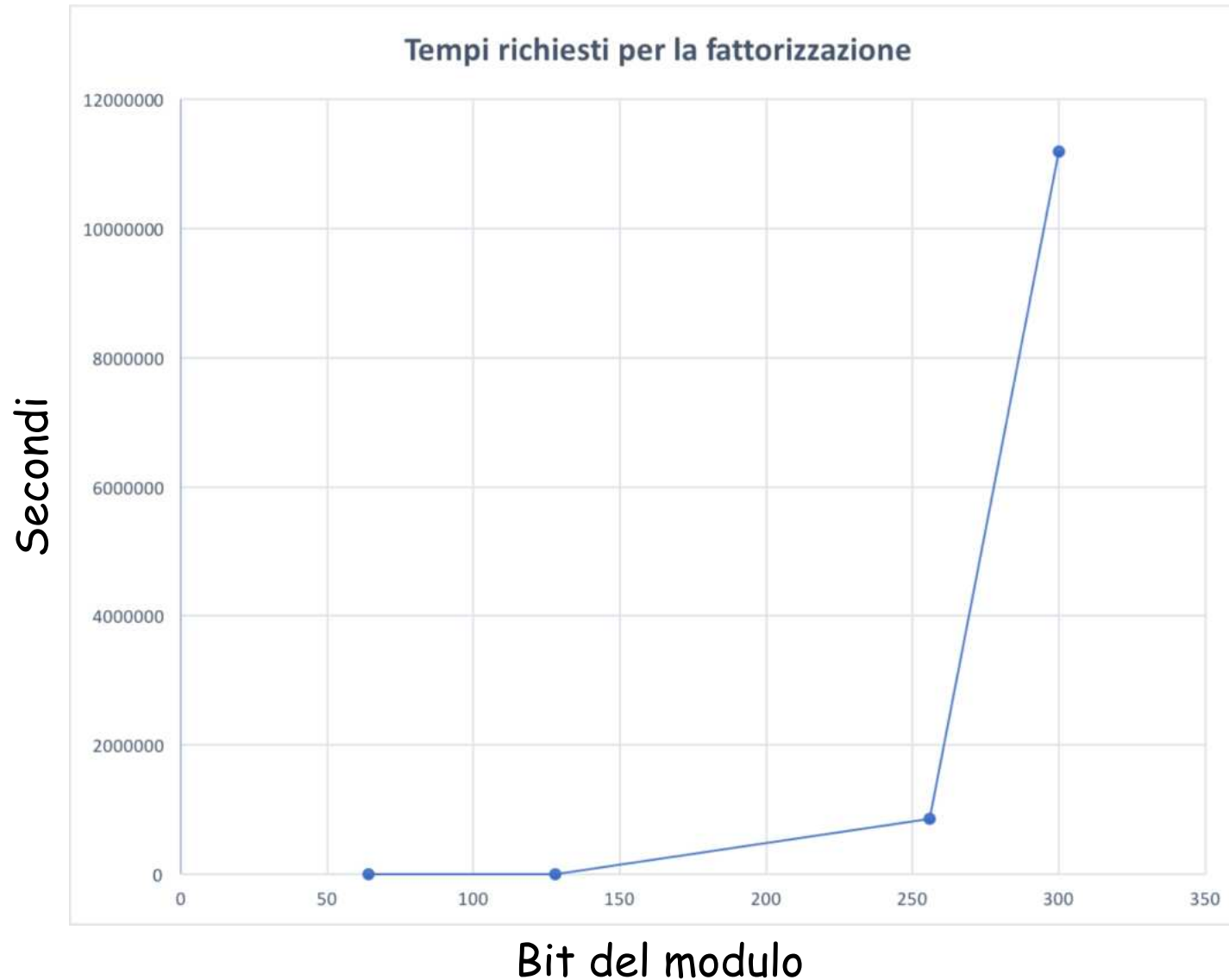
- Configurazione Hardware
  - Processore: Intel Core i7 2,8GHz a 64 bit
  - RAM: 4GB
  - Hard Disk: Apple SSD SM1024G

Dimensione Modulo (in bit)	Tempo Richiesto
64	0.0517s
128	0.4841s
256	85.1588s
300	1119.4591s

Tempi restituiti da YAFU

# Cifratura RSA in OpenSSL

## Fattorizzazione Modulo RSA - Prestazioni



# Bibliografia

- **Network Security with OpenSSL**  
Pravir Chandra, Matt Messier and John Viega (2002),  
O'Reilly
  - Cap. 2.4.3
  - Appendix A. Command-Line Reference
- **Presentazione Lezione Corso di Sicurezza, Prof. De Santis**
  - Cifrari Asimmetrici
- **Documentazione su OpenSSL**
  - <https://www.openssl.org/docs/>

