Crittografia a chiave pubblica

Esercizi con OpenSSL

Alfredo De Santis

Dipartimento di Informatica
Università di Salerno

ads@unisa.it

http://www.dia.unisa.it/professori/ads

Marzo 2017

Cifratura e Decifratura

- **Esercizio 1** Generare una coppia di chiavi RSA a 2048 bit e memorizzarle nel file rsakey.pem
- **Esercizio 2** Estrarre la chiave pubblica dal file rsakey.pem e mostrare il relativo contenuto
- **Esercizio 3** Creare file testuale, cifrarlo con la chiave pubblica e memorizzare il risultato in un file
- **Esercizio 4** Decifrare il file creato al passo precedente

Cifratura e Decifratura

- **Esercizio 5** Creare una chiave privata RSA di 160 bit e salvarla nel file smallkey.pem
- **Esercizio 6** Mostrare le informazioni relative alla chiave generata
- **Esercizio 7** La chiave privata dovrebbe essere memorizzata in chiaro o in modo cifrato?
  - Argomentare e motivare la risposta
- **Esercizio 8** La chiave pubblica dovrebbe essere memorizzata in chiaro o in modo cifrato?
  - Argomentare e motivare la risposta

Cifratura e Decifratura

- **Esercizio 9** Creare una chiave privata private1.pem di 1024 bit e memorizzarla in chiaro
- **Esercizio 10** Creare una chiave privata private1.pem di 1024 bit e memorizzarla in modo cifrato, mediante DES Triplo
- **Esercizio 11** Mostrare le informazioni relative alle chiavi private private1.pem e private2.pem
- **Esercizio 12** Creare una una chiave pubblica associata alle chiavi private private1.pem e private2.pem, rispettivamente
- **Esercizio 13** Mostrare le informazioni relative alle due chiavi pubbliche

Cifratura e Decifratura

- **Esercizio 14** Chiedere ad un compagno di corso la sua chiave pubblica. Cifrare un file con tale chiave ed inviarlo al compagno. Chiedere al compagno di decifrare il file che ha ricevuto
- **Esercizio 15** Chiedere ad un compagno di corso la propria chiave pubblica. Cifrare un file di grandi dimensioni mediante un cifrario simmetrico, usando una password arbitraria. Cifrare la password mediante la chiave pubblica del compagno di corso. Inviare al compagno di corso sia la password cifrata che il file cifrato. Chiedere al compagno di decifrare il file

Recupero chiave privata RSA

- Nel seguente esempio verrà mostrato come, a partire da una chiave pubblica RSA (di dimensioni "troppo piccola"), è possibile recuperare la relativa chiave privata
- 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
  1. *yafu - Automated integer factorization*
     - https://sourceforge.net/projects/yafu/
  2. genPriv.py – Script *Python* per la codifica di chiavi in formato PKCS1

Recupero chiave privata RSA

1) 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-----
```

Recupero chiave privata RSA

1) 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
writ
---
MCIwDQYJKoZIhvcNAQEBBQADEQAwDgIHAVIhTDIDtQIDAQAB
-----END PUBLIC KEY-----
```

Base 10

Modulo RSA di 50 bit

10111110001010000101001100001100100000001110110101

Base 2

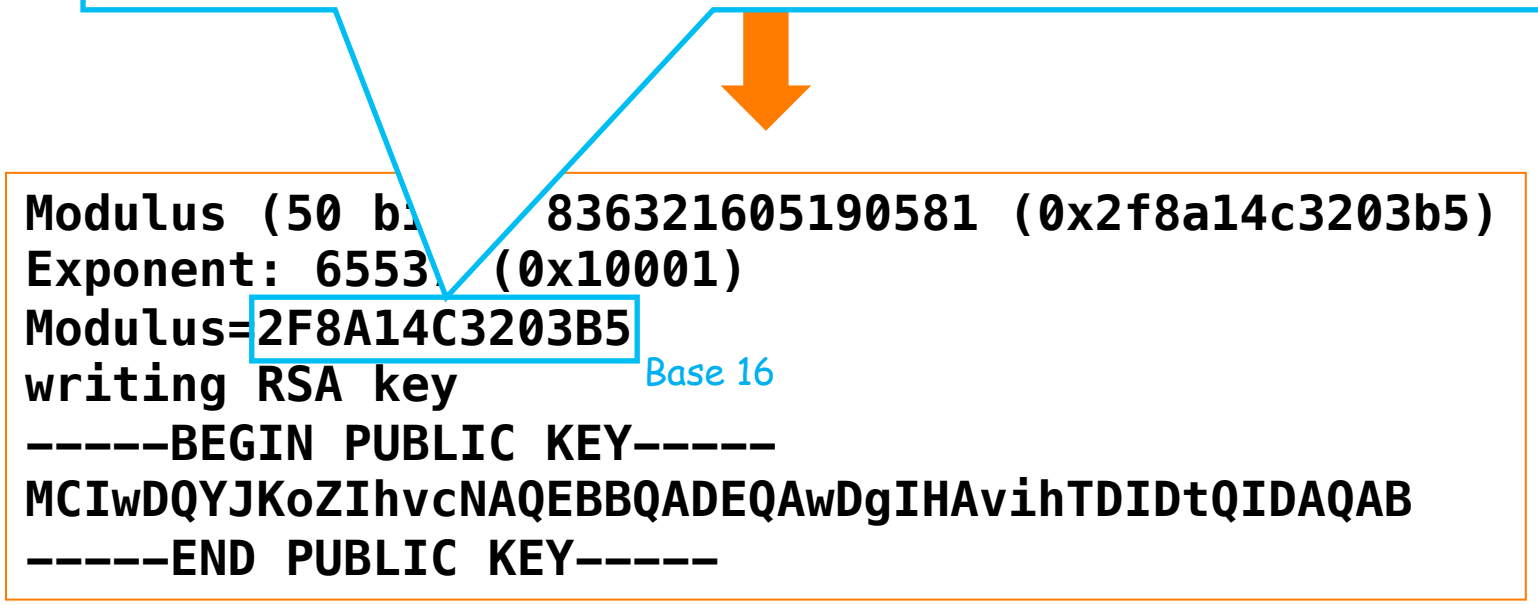
Recupero chiave privata RSA

1) Gene
chiave

```
open
open
open
```

È possibile convertire in decimale il valore esadecimale del modulo n, mediante vari strumenti, come ad es. il seguente convertitore online

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



Recupero chiave privata RSA

- 2) Fattorizziamo il modulo (836321605190581) al fine di trovare i relativi fattori primi, siano essi p e q
- Considereremo due metodi alternativi per farlo
  - Online, utilizzando http://factordb.com/
  - Offline, utilizzando YAFU

http://factordb.com/

Numero che si intende fattorizzare

Sequences | Re[illegible]results | Factor tables | Status

836321605190581 Factorize!

Result:

digits	number
15 (show)	836321605190581 = 27346349 · 30582569

n p q

More information

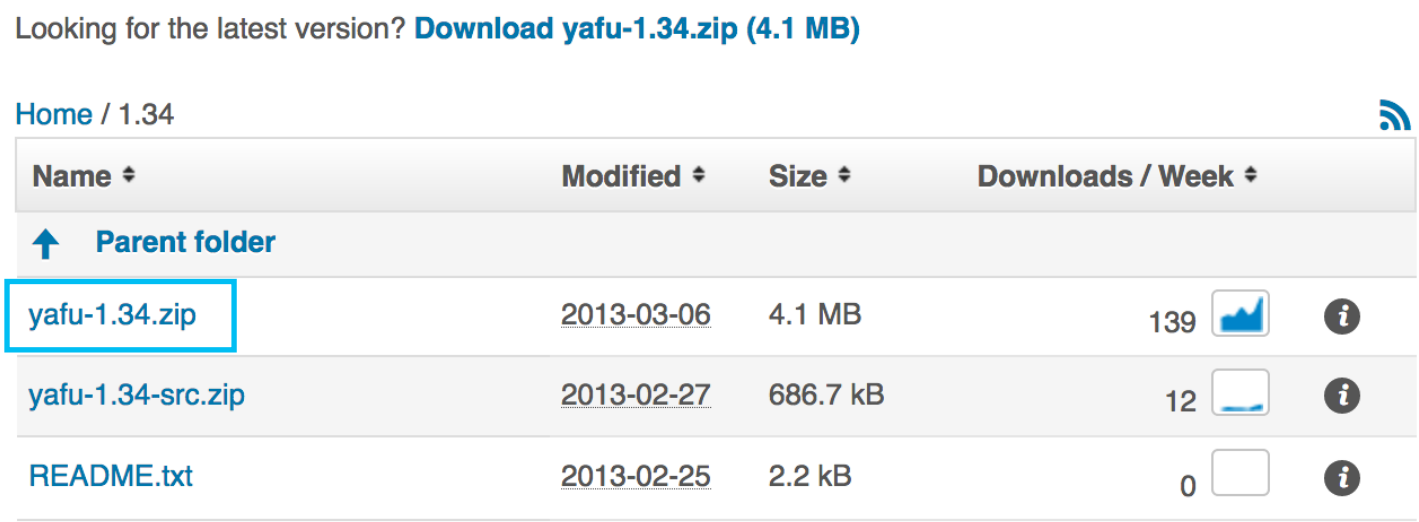
ECM

YAFU

- Permette di fattorizzare, in maniera completamente automatica, numeri interi arbitrari presi in input
- Utilizza alcuni tra i più potenti algoritmi di fattorizzazione
  - 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 a pieno l'utilizzo di processori multi-core o many-core

YAFU
(Installazione)

1. Download YAFU



- `unzip yafu-1.34.zip`
- `chmod +x yafu`

YAFU

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


fac: factoring 836321605190581
fac: using pretesting plan: normal
fac: no tune info: using qs/gnfs crossover of 95 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


***factors found***

P8 = 27346349
P8 = 30582569

ans = 1
```

Numero da fattorizzare

p

q

Recupero chiave privata RSA

3) Dopo aver ottenuto il valore dei numeri primi p e q, possiamo ricostruire il valore della chiave privata e codificarla in PKCS1 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
```

Recupero chiave privata RSA

Infine, visualizziamo la chiave privata ricostruita al passo precedente

```
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-----
```

Recupero chiave privata RSA

- **Esercizio 18** A partire dalla seguente chiave pubblica RSA, trovare la relativa chiave privata e codificarla in PKCS1

```
MCwwDQYJKoZIhvcNAQEBBQADGwAwGAIRALuxEy5a8lwnCny2bTpvGSkCAwEAAQ==
```

  - Suggerimento: per visualizzare i campi della chiave pubblica codificata in Base64 è possibile utilizzare il seguente servizio online
    - https://lapo.it/asn1js/

Sfida RSA

- **Esercizio 19.1** Creare una coppia di chiavi RSA da 300 bit
  - challengePub.pem e challengePriv.pem
- **Esercizio 19.2** Cifrare un file con la chiave pubblica
- **Esercizio 19.3** Inviare il file cifrato e la chiave pubblica ad un compagno di corso e sfidarlo a decifrare il file
- **Esercizio 19.4** Confrontarsi col compagno di corso sul tempo impiegato per il recupero della chiave privata
  - In particolare per la fattorizzazione del modulo n

Tempi di generazione chiavi RSA

- **Esercizio 20** Generare 9 chiavi RSA da 8192 bit ciascuna
  - Riportare in formato tabellare i tempi relativi alla generazione di ciascuna chiave
  - Qual è il maggiore tempo richiesto?
  - Qual è il minore tempo richiesto?
  - Qual è il tempo medio?
  - Da cosa deriva questa differenza tra i vari tempi di esecuzione?

Utilizzare il comando **`time`** per valutare i tempi di esecuzione

Generazione Chiavi RSA
(Tempi Richiesti)

- 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**

Fattorizzazione
(Tempi Richiesti)

- 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

Fattorizzazione
(Tempi Richiesti)