RSA Cryptosystem

Gerardo Pelosi

Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB)
Politecnico di Milano

gerardo.pelosi - at - polimi.it

Overview

Lesson contents

- Public-key Cryptosystems
- RSA Cryptosystem
  - Speed up RSA primitives
  - Numerical examples
- RSA Key Length and Performances
- Security of the RSA cryptoscheme [part-I]
  - Integer Factorization Problem (FP)
  - RSA Problem (RSAP)
- Security of the RSA cryptoscheme [part-II]

Public-Key Cryptography (1)

- PKC is also known as *asymmetric-key cryptography*, it employs two separate keys:
  - Private key (for decryption) – kept secret and never shared
  - Public key (for encryption) – advertised publicly as part of a "digital certificate" that includes also the name of the owner, the details of the PKC scheme and some other information

$$\langle \mathcal{A}, \mathcal{M}, \mathcal{C}, \mathcal{K}, \{Enc_{k_{\mathrm{pub}}}(\cdot)\}, \{Dec_{\mathrm{priv}}(\cdot)\} \rangle$$

- If Alice wants to communicate confidentially with Bob, she can encrypt a message using Bob's publicly available key.

  Such a communication is only decipherable by Bob as only Bob has access to the corresponding private key.

Public-Key Cryptography (2)

- PKC schemes are substantially slower than symmetric-key schemes. They are most commonly used for:
    - the transport of either a "shared secret" or "session key" that is subsequently used for bulk data encryption with a symmetric algorithm
    - encryption of small data payloads such as credit card numbers and PINs
    - providing "data integrity" and "data origin authentication", i.e. ensuring that the data have not been tampered with, and come from the rightful sender
    - "entity authentication" and "authenticated key establishment" protocols: typical examples are challenge-response protocols and the SSL/TLS protocols

Public-Key Cryptography (3)

Outline of the authentication procedure:

- Party A, willing to send an authenticated message m to party B, applies the decryption function to m with A's own private key to obtain a digital signature s.
  The authenticity of the message will be checked through applying the encryption function with A's public key to s and comparing the result with the message m.

  The result of the validity checking operation will match the message only if the signature was produced with the correct secret key!
- Usually, only a digest of the message is authenticated in order to get computational and communication efficiency.
- Therefore the typical authentication will make use of publicly known and cryptographically strong hash function $h(\cdot)$. A computes:

$$s \leftarrow Dec_{k_{priv,A}}(h(m)), \text{ then sends: } \langle m, s \rangle \text{ to B}$$

Public-Key Cryptography (4)

- if A wants to send a message m to B with both authentication and confidentiality she computes:

  $$s \leftarrow Dec_{k_{priv,A}}(h(m)),\ c \leftarrow Enc_{k_{pub,B}}(\langle m,\ s\rangle)$$

  and sends c to B

  The processing steps undertaken by B to recover m from c are

  $$\text{Obtain: } \langle m,\ s\rangle \leftarrow Dec_{k_{priv,B}}(c), \quad \text{Check: } h(m) \overset{?}{=} Enc_{k_{pub,A}}(s)$$

The alternative approach based on computing $s \leftarrow Dec_{k_{priv,A}}(h(m))$ and $c \leftarrow Enc_{k_{pub,B}}(m)$ to transmit $\langle c, s\rangle$ to B is not secure against a passive adversary who can peel off the part denoted by s from the message $\langle c, s\rangle$ and illegally re-use it for sending messages with the same hash image (if possible) on behalf of A.

Public-Key Cryptography (5)

The way to implement the Enc and Dec functions of a PKC in practice is to make use of (old) “number theoretic problems”.
The most common ones are:

- Integer Factorization Problem:
  given a composite integer n, compute its factorization $\prod_i p_i^{e_i},\ e_i \geq 1$
- Discrete Logarithm extraction in a cyclic group:
  given $(\langle g \rangle, \cdot)$ and $g_1 = g^x$, find $x \in \{0, 1, \ldots, |g| - 1\}$

Other possible, non strictly number theoretic, problems are:

- Finding the shortest vector in a l-dimensional lattice (vector space with scalar coefficients over $\mathbb{Z}^l$ or $\mathbb{Q}^l$), given a basis for the space and a notion of distance (norm)
- Decoding a “general linear code”

Public-Key Cryptography (6)

- These problems allow both the PK encryption and decryption functions ($Enc_{k_{\text{pub}}}(\cdot)$, $Dec_{k_{\text{priv}}}(\cdot)$) to be defined in such a way that
  1. the public key and the private key are linked in a mathematical way
  2. the knowledge of the public key tells you nothing about the private key
  3. the knowledge of the private key allows you to decrypt messages encrypted with the public key
- The encryption functions are also called **one-way trapdoor functions**
- These functions are:
  - easy to compute in one direction (i.e., knowing the public key)
  - computationally hard to invert, without knowing some secret information (i.e., the private key)

Public-Key Cryptography (7)

- Desiderata for the actual definition of both encryption and decryption transformations require some thoughts
- Indeed, the concept of PKC was so strange that
  - it was not until 1976 that anyone thought of it:
    W. Diffie and M. Hellman. 2006. New directions in cryptography. IEEE Trans. Inf. Theor. 22, 6 (Sep. 1976), 644-654
  - while in 1977, the first full PKC cryptosystem (**RSA**) was invented at MIT by Ron **R**ivest, Adi **S**hamir and Leonard **A**dlemann
  - English mathematicians, developed a system equivalent to RSA in 1973, but it wasn't declassified until 1997

Rivest Shamir Adleman (RSA) - Cryptoscheme

Public Key: k_{pub}

Let p, q be two prime integers ($p \approx q$) – randomly chosen

$$\begin{array}{ll} \text{RSA public modulus:} & n \leftarrow p \cdot q \\ \text{RSA public exponent:} & e \overset{\text{Random}}{\leftarrow} \mathbb{Z}^*_{\varphi(n)} \\ & e \in \{1 \leq i \leq \varphi(n) - 1 \text{ s.t. } \gcd(e, \varphi(n)) = 1\} \end{array}$$

$$k_{pub} = \langle e, n \rangle$$

Private Key: k_{priv}

$$\text{RSA private exponent:} \quad d \leftarrow e^{-1} \mod \varphi(n) \qquad d \in \mathbb{Z}^*_{\varphi(n)}$$

$$k_{priv} = \langle p, q, \varphi(n), d \rangle$$

Rivest Shamir Adleman (RSA) - Cryptoscheme

One-way Function with Trapdoor

Encryption Function

Given a RSA public key $k_{pub} = \langle n, e \rangle$, the message $\mathcal{M}$ and ciphertext $\mathcal{C}$ spaces are defined as elements of $\mathbb{Z}_n$; i.e., $m, c \in \mathbb{Z}_n$

$$c \leftarrow Enc_{k_{pub}}(m) = m^{e \bmod \varphi(n)} \bmod n$$

Decryption Function

Given a RSA private key $k_{priv} = \langle p, q, \varphi(n), d \rangle$, and a proper ciphertext $c \in \mathbb{Z}_n$

$$m \leftarrow Dec_{k_{priv}}(m) = c^{d \bmod \varphi(n)} \bmod n$$

Rivest Shamir Adleman (RSA) - Cryptoscheme

In order to employ the previous definitions in a full cryptosystem it is necessary to prove:

$$Dec(Enc(m)) = m,\ \forall\, m \in \mathbb{Z}_n$$
$$(m^e)^{d \bmod \varphi(n)} \bmod n \equiv m^{ed \bmod \varphi(n)} \bmod n \overset{?}{\equiv} m \bmod n$$

$$Enc(Dec(c)) = c,\ \forall\, c \in \mathbb{Z}_n$$
$$(c^d)^{e \bmod \varphi(n)} \bmod n \equiv c^{ed \bmod \varphi(n)} \bmod n \overset{?}{\equiv} c \bmod n$$

- The symmetry of the encryption and decryption functions allows us to restrict the correctness proof only to the encryption transformation.

Rivest Shamir Adleman (RSA) - Cryptoscheme

Given $n = p \cdot q, m \in \mathbb{Z}_n$; $e, d \in \mathbb{Z}^*_{\varphi(n)}$ $(e \cdot d \equiv_{\varphi(n)} 1)$, we need to prove that:

$$(m^e)^{d \bmod \varphi(n)} \bmod n \equiv m \bmod n,\ \forall\, m \in \mathbb{Z}_n$$

we need to distinguish two cases:

1st case: $\gcd(n, m) = 1$

In this case m has a multiplicative inverse in $\mathbb{Z}_n$: $m \in \mathbb{Z}^*_n$, where $|\mathbb{Z}^*_n| = \varphi(n)$ thus, for some integer t, we can write the following:

$$(m^e)^d \equiv_n m^{1+t\varphi(n)} \equiv_n m \cdot (m^{\varphi(n)})^t \equiv_n m$$

Therefore,

$$(m^e)^{d \bmod \varphi(n)} \bmod n \equiv m^{ed \bmod \varphi(n)} \bmod n \equiv m \bmod n \qquad \text{(cvd.)}$$

2nd case: $\gcd(n, m) \neq 1$

Being $\gcd(n, m) \neq 1$ we can write (without loss of generality) that $\gcd(n, m) = p$, that is, we can assume $\mathbf{m} = \mathbf{u} \cdot \mathbf{p}$, for some integer u.

Consider that:

$$\begin{array}{l} m^{\varphi(q)} \bmod q \equiv m^{q-1} \bmod q \equiv 1 \bmod q \qquad (obs.: \gcd(q, m) = 1) \\ (m^{\varphi(n)})^t \bmod q \equiv (m^{(q-1)})^{(p-1)t} \bmod q \equiv 1 \bmod q \\ (\mathbf{m}^{\varphi(\mathbf{n})})^{\mathbf{t}} = \mathbf{1} + \mathbf{s}\,\mathbf{q}, \text{for some integers } \mathbf{s} \text{ and } \mathbf{t}. \end{array}$$

Thus,

$$\begin{array}{l} (\mathbf{m}^{\mathbf{e}})^{\mathbf{d}} \equiv_n m^{1+t\varphi(n)} \equiv_n m \cdot (m^{\varphi(n)})^t \equiv_{\mathbf{n}} \mathbf{m} \cdot (\mathbf{1} + \mathbf{s}\,\mathbf{q}) \\ m \cdot (1 + s\,q) \equiv_n m + \mathbf{m}\,s\,q \equiv_n m + \mathbf{u}\,\mathbf{p}\,s\,q \equiv_n m + (u\,s)\,\mathbf{n} \equiv_{\mathbf{n}} \mathbf{m} \end{array}$$

Hence:

$$(m^e)^{d \bmod \varphi(n)} \bmod n \equiv m^{ed \bmod \varphi(n)} \bmod n \equiv m \bmod n \qquad \text{(cvd.)}$$

Speeding Up of the RSA Encryption Function

If a repeated S&M strategy is used to compute modular exponentiations, the complexity of performing the RSA encryption is of $\lceil \log_2 \varphi(n) \rceil + \mathrm{HW}(e)$ multiplications, with $\mathrm{HW}(e)$ being the Hamming Weight of the public exponent.

- Typically, the RSA public exponent is chosen as one of the following prime integers: $\{3 = 2^1 + 1, 17 = 2^4 + 1, 65537 = 2^{16} + 1\}$, checking that $e \in \mathbb{Z}^*_{\varphi(n)}$.
- The security of a RSA public key $k_{pub} = \langle 65537, n \rangle$ is still guaranteed by both the secrecy of the prime factors p, q and the secrecy of the value of the Totient function: $\varphi(n) = (p-1)(q-1)$.
- Even if e is small (f.i., e=3), the secret exponent $d = e^{-1} \bmod \varphi(n)$ will have (with high probability) approximately the same bit-length of $\varphi(n)$ with a non-predictable Hamming weight.

Speeding Up of the RSA Decryption Function

To reduce the computational load of the repeated S&M of the decryption function, we can exploit the CRT to obtain a $\times 4$ speed-up. Given:

$$k_{priv} = \langle d, p, q, \varphi(n) \rangle, \quad m \leftarrow Dec_{k_{priv}}(c) = c^{d \bmod \varphi(n)} \bmod n$$

We compute:

$$\begin{aligned} m_p &= c^d \bmod p \equiv_p c^{d \bmod (p-1)} \\ m_q &= c^d \bmod q \equiv_q c^{d \bmod (q-1)} \end{aligned}$$

and recombine the values through the following relations (CRT):

$$m = c^{d \bmod \varphi(n)} \bmod n \Leftrightarrow \begin{cases} m = m_p \bmod p \\ m = m_q \bmod q \end{cases} \Leftrightarrow \begin{cases} m \equiv_p c^{d \bmod \varphi(p)} \\ m \equiv_q c^{d \bmod \varphi(q)} \end{cases}$$

$$m \equiv \left(q(q^{-1} \bmod p) m_p + p(p^{-1} \bmod q) m_q \right) \bmod n$$

Speeding Up of the RSA Decryption Function

Assuming $n \approx \varphi(n)$, $t = \lceil \log_2 n \rceil$, and $\lceil \log_2 p \rceil \approx \lceil \log_2 q \rceil \approx \frac{t}{2}$

- $d \bmod \varphi(p)$, and $d \bmod \varphi(q)$ will be $\frac{t}{2}$ bit long, with roughly $\frac{t}{4}$ bits set to "1"
- The computational complexity of a modular multiplication is quadratic in the number of bits, reducing by a factor of 2 the size of the operands yields a 4 times speedup, i.e. $\mathtt{mul}_{\frac{t}{2}} = \frac{1}{4}\mathtt{mul}_t$.
- Assuming the same cost for both a squaring and a multiplication, the computation of m_p, m_q requires on average: $\frac{3}{2}\frac{t-1}{2}\ \mathtt{mul}_{\frac{t}{2}}$ op.s modulo p and $\frac{3}{2}\frac{t-1}{2}\ \mathtt{mul}_{\frac{t}{2}}$ op.s modulo q

$$m \leftarrow Dec_{k_{priv}}(c) \quad \text{avg. cost: } \mathcal{O}\left(\frac{3}{2}(t-1)\ \mathtt{mul}_t\right)$$

$$m \overset{\text{CRT}}{\leftarrow} Dec_{k_{priv}}(c) \quad \text{avg. cost: } \mathcal{O}\left(2\,\frac{3}{2}\,\frac{t-1}{2}\,\mathtt{mul}_{\frac{t}{2}}\right) = \mathcal{O}\left(\frac{1}{4}\,\frac{3}{2}(t-1)\mathtt{mul}_t\right)$$

Numerical Example (1)

Consider an RSA cryptosystem where $k_{pub} = \langle n, e \rangle = \langle 143, 77 \rangle$. Now, knowing that $n = p \cdot q = 13 \cdot 11$:

1. Compute the corresponding private key, $k_{priv} = \langle p, q, d, \varphi(n) \rangle$.
2. Given the plaintext message $m = 101 \in \mathbb{Z}_n$, compute the corresponding ciphertext through applying a repeated S&M strategy.
3. Given the ciphertext computed at step **(2)**, show the computations necessary to retrieve the original message through applying the Chinese Remainder Theorem (CRT).

Numerical Example (2)

Consider an RSA cryptosystem where $k_{pub} = \langle n, e \rangle = \langle 143, 77 \rangle$. Now, knowing that $n = p \cdot q = 13 \cdot 11$:

- Compute the corresponding private key, $k_{priv} = \langle p, q, d, \varphi(n) \rangle$.

Solution

$\varphi(n) = (p-1)(q-1) = 120 = 2^3 \cdot 3 \cdot 5$
$d = e^{-1} \bmod \varphi(n) = e^{\varphi(\varphi(n))-1} \bmod \varphi(n) = 77^{\varphi(120)-1} \bmod 120 \Rightarrow$
$d = e^{-1} \bmod \varphi(n) = 77^{31} \bmod 120$
$d \equiv_{120} 77^{31} \equiv_{120} 77^{11111_2} \equiv_{120} ((((77^2) \cdot 77)^2 \cdot 77)^2 \cdot 77)^2 \cdot 77 \equiv_{120} \cdots$
$d = 53 \bmod 120.$

The same result may be obtained through applying the extended Euclid's algorithm for the computation of $\xi, d \in \mathbb{Z}_{\varphi(n)}$ in the following relation:

$$1 = \gcd(\varphi(n), e) = \varphi(n) \cdot \xi + d \cdot e$$

Numerical Example (3)

Consider an RSA cryptosystem where $k_{pub} = \langle n, e\rangle = \langle 143, 77\rangle$.

1. Given the plaintex message $m = 101 \in Z_n$, compute the corresponding ciphertext through applying a repeated S&M strategy.

Solution

$m = 101 \in Z_n$, $n = 143$, $e = 77 = 1001101_2$:

$c = m^e \bmod n \equiv_{143} 101^{1001101_2} \equiv_{143} ((((101^2)^2)^2 \cdot 101)^2 \cdot 101)^2)^2 \cdot 101 \Rightarrow$
$\ldots$

$c = 101^{77} \bmod n \equiv_{143} 95.$

Numerical Example (4)

Consider an RSA cryptosystem where $k_{pub} = \langle n, e \rangle = \langle 143, 77 \rangle$.

- Given $c \equiv_n m^e \equiv_{143} 95$, and $n = p \cdot q = 13 \cdot 11 = 143$; $\varphi(n) = 120, d \equiv_{120} 53$, derive the plaintext applying the CRT.

Solution

$$m = c^{d \bmod \varphi(n)} \bmod n \Leftrightarrow \begin{cases} m = m_p \bmod p \\ m = m_q \bmod q \end{cases} \text{where:}$$

$$m_p = c^{d \bmod \varphi(p)} \bmod p \equiv_{13} 95^{53 \bmod 12} \equiv_{13} 4^{5 \bmod 12} \equiv_{13} 10 \text{ and}$$
$$m_q = c^{d \bmod \varphi(q)} \bmod q \equiv_{11} 95^{53 \bmod 10} \equiv_{11} 7^{3 \bmod 10} \equiv_{11} 2$$
$$m \equiv_n \left(M_p \cdot M'_p \cdot m_p + M_q \cdot M'_q \cdot m_q\right) \bmod n$$
$$M_p = q = 11,\ M'_p = q^{-1} \bmod p = 11^{-1} \bmod 13 \equiv_{13} -2^{-1} \equiv_{13} 6$$
$$M_q = p = 13,\ M'_q = p^{-1} \bmod q = 13^{-1} \bmod 11 \equiv_{11} 2^{-1} \equiv_{11} 6$$

$$\Rightarrow m \equiv_{143} (11 \cdot 6 \cdot 10 + 13 \cdot 6 \cdot 2) \equiv_{143} 101$$

RSA Keypair Generation with Common Tools

OpenSSL

- The SSL/TLS transport layer employs asymmetric key primitives to perform both endpoint authentication and session key establishment
- To generate an RSA keypair to be used with SSL/TLS with OpenSSL: `openssl genrsa -out <filename.pem> <modulus_size>`
  - OpenSSL will store both keys in the `filename.pem`, in ASCII-encoded DER format and print the private key on screen
  - The output file contains the private key, with all the required values for CRT-RSA, plus the public exponent
- To export only the public key to a separate file:

```
openssl rsa -in <filename.pem> -pubout -out <public_filename.pem>
```

- To print out the contents of a keyfile in human-readable format

```
openssl rsa -noout -text -in <keyfile>
```

RSA Keypair Generation with Common Tools (OpenSSL)

Stored `filename.pem`

```
-----BEGIN RSA PRIVATE KEY-----
lqOqtbGu9W0JPHDKAUsr83fs+xweeY8vccRkMOlwoAdmbR/4
HXTttsuGxyEKkDdu71y4IVJCDEd+ULRs70ZeRz5baayd3ROO
e5rt+VAe/O8+A63WXijZi/GpDG2m8R49hEjAjc8mXQIDAQAB
dIZ6HEzm75ZtgJM7aK38ulzVOPA5A9hahnPS8m+YbAKGTmRu
kTVf6hm0+iN4Ck96bgCAiW36M22v4inRFipn6fOZQwSUXLE5
/Nv2tWPIadse+UDLsFgr/e3WYKyOnk0CQQDYIG9VjnvfojOz
pFgARS1csSG1/Ap99dYMu6Xyy1cDnghod4DClx12GC0mhwc7
pCvjtnGlFHmVC4Xby8W/jh8qOxAzrYRzFBrCyjBsfCDW7q9P
J6Ly4peoEUZZ2jB1RwJAJmcVAVHRBvqv/OTwcY/FBOje8mQm
MiaFgzuwkAS7kGP7QHzDf1AUpxBCrY7epQjo0+2TUwJAGNXy
r5qZAs5by1Y4SXVAisMjVGcAgKh05AL0aSTqsw75hHMKQlWj
4QJAJ2NTs1yULGPdOdzGzzjCd6nHy/rbqXnkSOHj5VrjGG1d
O2JeXon9tqTUmI99Tf7E64crRA==

-----END RSA PRIVATE KEY-----
```

Stored `public_filename.pem`

```
-----BEGIN PUBLIC KEY-----
3DQEBAQUAA4GNADCBiQKBgQDC8QNqoUgjx5t+kjE1JSLzrROe
DFUSOoAFErWdCI8T9jzQySZKQ0XTU1Wp1Fyz+h1l6xMgKOw4f
PTQIDAQAB

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

Human Readable `filename.pem`

```
Private-Key:  (1024 bit)
modulus:
00:a3:96:a3:aa:b5:b1:ae:f5:6d:09:3c:70:ca:01:
...
publicExponent:  65537 (0x10001)
privateExponent:
47:bd:b7:77:15:09:a0:65:e7:74:86:7a:1c:4c:e6:
...
prime1:
00:d8:20:6f:55:8e:7b:df:a2:3d:33:9d:c5:45:46:
...
prime2:
00:c1:c4:d8:a4:2b:e3:b6:71:a5:14:79:95:0b:85:
...
exponent1:
26:67:15:01:51:d1:06:fa:af:ff:44:f0:71:8f:c5:
...
exponent2:
18:d5:f2:2e:98:b1:87:20:e9:d7:c5:fd:a4:8c:73:
...
coefficient:
27:63:53:b3:5c:94:2c:63:dd:39:dc:c6:cf:38:c2:

...
```

RSA Keypair Generation with Common Tools

SSH - Secure SHell

- An alternative to password-based user authentication over the SSH secure transport protocol is based on asymmetric cryptosystems
- Generate an RSA keypair to perform secure login via SSH with:
```
ssh-keygen -t rsa -b <modulus_size_bits> -f <filename>
```
- Public and private key will be available in printable-ASCII encoded format in `filename.pub` and `filename` respectively
- By default your keys are stored in the `.ssh` directory in your home
- If you have more than one keypair, you can select one during your login as `user` on `host` as `ssh -i <private_key_filename> user@host`
- You can regenerate the public key from the private one as
```
ssh-keygen -f <filename> -y > filename.pub
```

Mathematical Security of the RSA cryptoscheme

The Security of RSA cryptosystem is related to the computational complexity of the integer factoring problem.

Factoring Problem (FP)

Given an integer n as the product of only two prime factors $n = p \cdot q$, find these factors

RSA Problem (RSAP)

Given a ciphertext $c = m^e \bmod n \in \mathbb{Z}_n$, where $n = p \cdot q$, $e \in \mathbb{Z}^*_{\varphi(n)}$, find $m \in \mathbb{Z}_n$

- If you solve the FP, then you solve also RSAP! (...practically critical)
- If you solve the RSAP, it is not known if you can solve FP!

Factoring Problem (FP)

Notation

To measure the complexity of algorithms to factor an integer n the following function is often employed:

$$L_n(\alpha, \beta) = \exp\Big(\beta + o(1)\big((\log n)^{\alpha}(\log \log n)^{1-\alpha}\big)\Big)$$

Notice that:

- a complexity of $\mathcal{O}(L_n(0, \beta))$ corresponds to a polynomial time algorithm (the input size of the problem is $\log n$).
- a complexity of $\mathcal{O}(L_n(1, \beta))$ corresponds to an algorithm which runs in exponential time.
- Hence, for $0 < \alpha < 1$, $L_n(\alpha, \beta)$ interpolates between polynomial and exponential time and is a.k.a. *sub-exponential* function.

Factoring Problem (FP) - (1)

Trivial Division

- Works quite well for removing primes from an integer n up to 12 digits (that is, numbers $\leq 10^{12}$).
  - Try every prime number up to $\sqrt{n}$ and check if it is a factor of $n \Rightarrow$ exponential complexity: $L_n(1, 1)$
- A variation is to form a product $x = p_1 p_2 p_3 \dots p_r$ of r primes and to compute $\gcd(n, x)$ for finding the largest prime factor in n. Here is a product of all primes $p \leq 97$:

  2305567963945518424753102147331756070

Factoring Problem (FP) - (2)

Elliptic Curve Method

This has a sub-exponential complexity $L_p(1/2, c)$, where p denotes the smallest factor of n.
It is appropriate when $p \leq 2^{50}$

Quadratic Sieve

This is the fastest method for factoring integers with size between 2^{240} and 2^{300}. It has complexity $L_n(1/2, 1)$

General Number Field Sieve (GNFS)

This is currently the most successful method for numbers greater than 2^{300} and has complexity $L_n(1/3, 1.923)$

Recommended Key Lengths - (1)

Currently 768-bit numbers are the largest RSA moduli that have been factored

(2000 machine-years on AMD-Opteron sigle core CPU 2.2 GHz, 2GiB RAM [ref. www.iacr.org/2010/006.pdf])

- it is recommended to employ moduli $n = p \cdot q$ of size at least 1024 bits to ensure medium-term security.
- For long-term security the recommendations suggest modulus sizes greater than 2048 bits.

Recommended Key Lengths - (2)

NIST recommended key lengths, considering the foreseen technological and theoretical cryptanalysis advancements [updated 2011]

Date	Security margin	Symmetric cipher	RSA
2010	80	2TDEA*	1024
2011 – 2030	112	3TDEA	2048
$>$ 2030	128	AES-128	3072
$>>$ 2030	192	AES-192	7680
$>>>$ 2030	256	AES-256	15360

- **Security margin**: Minimum computational effort expressed as the log_2 of the number of DES computations
- **Symmetric cipher**: Suggested cipher to achieve the minimum adequate level of security (Note: nTDEA $=$ Triple DES Algorithm with n keys)
  - (*) The assessment of at least 80 bits of security for 2TDEA is based on the assumption that an attacker has no more than 2^{40} matched ptx/ctx blocks
- **RSA**: size of the RSA modulus to achieve the adequate security margin, considering the best known integer factoring method (i.e., the GNFS)

RSA Performance

Overlook

RSA Hardware implementations are 1000$\times$ slower than both AES and DES, and at least 100$\times$ slower in software (for moduli $\geq$ 512 bits).

Table: OpenSSL Performances on an AMD Opteron^{TM} Processor-8378, 2.4Ghz (one core used)

Security Margin	**Modulus [bit]**	**Dec [ms]**	**Enc ($\mathbf{e = 2^{16} + 1}$) [ms]**
80	1024	0.299	0.019
112	2048	2.021	0.062
128	3072	25.1	12.2
192	7680	113.3	14.1
256	15360	720.2	20.4

RSA is mainly used to exchange an ephemeral "session key" (shared secret value). The session key is then employed to transmit data enciphered via a symmetric-key encryption scheme

Security of the RSA cryptoscheme

Lemma 1

The RSA problem is no harder than the "factoring" problem.

Proof.

Using a factoring oracle we first find the factorization of n.
We can now compute $\varphi(n) = (p-1)(q-1)$ and then $d = \frac{1}{e} \bmod \varphi(n)$.
Once d has been computed it is easy to recover m via
$c^d \equiv_n m^{ed} \equiv_n m^{1 \bmod \varphi(n)}) \equiv_n m$.
Hence, the RSA problem is no harder than "factoring". □

Fact

Major open question: how much easier is the problem of breaking RSA (a.k.a. RSA-Problem), w.r.t. the Integer Factorization Problem (FP) ?

Security of the RSA cryptoscheme

Lemma 2

Knowing the private exponent d corresponding to the public key $k_{pub} = \langle n, e \rangle$ it is possible to efficiently factor n

Proof.

$ed - 1 = s(p-1)(q-1)$ for some integer s.
Picking $x \in \mathbb{Z}_n^*$, we know that $x^{ed-1} \equiv_n 1 \Rightarrow$ we can compute $y = \sqrt{x^{ed-1}} = x^{\frac{ed-1}{2}}$ ($ed-1$ is known and will be obviously even).
Therefore, we have the identity $y^2 - 1 \equiv_n 0$

- If $y \neq \pm 1 \bmod n$, a factor of n is obtained as: $\gcd(y-1, n)$
- If $y = -1 \bmod n$, we repeat this procedure from the beginning and pick another value for x
- If $y = +1 \bmod n$, we iterate the whole procedure starting from the square root of y

$\square$

Security of the RSA cryptoscheme

Lemma 3

Given a RSA modulus n and the value of $\varphi(n)$, one can efficiently factor n

Proof.

Being $\varphi(n) = (p-1)(q-1) = n - (p+q) + 1$,

we have that

$$\begin{aligned} -(p+q) &= \varphi(n) - n - 1 \\ p\,q &= n \end{aligned}$$

Therefore, through trivial algebra we can define the 2nd degree equation in the unknown Z:

$$Z^2 + (\varphi(n) - n - 1)\,Z + n = 0$$

the roots of this equation gives the prime factors p, q. □

Security of the RSA cryptoscheme

Since modular arithmetic is expensive it may be tempting to have more users sharing the same public modulus n, but employing different public/private exponents: (e_i, d_i). *Very fast HW implementations are obtained through tuning the modular arithmetic to the specific value of n*

Lemma 4

Given two RSA ciphertexts $c_1 = Enc_{\langle n,e_1\rangle}(m)$, $c_2 = Enc_{\langle n,e_2\rangle}(m)$, one can easily recover the original message m if $\gcd(e_1, e_2) = 1$.

Proof.

$1 = \gcd(e_1, e_2) = t_1\, e_1 + t_2\, e_2$ for proper values t_1, t_2.
A pair of values for t_1 and t_2 can be computed by the Euclidean Algorithm.

Then,

$$c_1^{t_1}\, c_2^{t_2} \bmod n = (m^{e_1})^{t_1}(m^{e_2})^{t_2} = m \bmod n$$

$\square$

Security of the RSA cryptoscheme

Practical RSA systems often use a “small” fixed public exponent (e.g., $e = 3$) so as to cut down the computational cost for the sender.

Lemma 5

The use of a small public exponent is unsafe in case multicast communications take place.

Proof.

Consider three RSA public keys: $\langle n_1, 3_1\rangle$, $\langle n_2, 3\rangle$, $\langle n_3, 3\rangle$, and suppose someone employs them to send the same message m thrice. Then, a passive attacker can recover the message as:

$$\begin{cases} c_1 = m^3 \bmod n_1 \\ c_2 = m^3 \bmod n_2 \\ c_3 = m^3 \bmod n_3 \end{cases} \overset{\text{CRT}}{\Longrightarrow} X \equiv m^3 \bmod (n_1\, n_2\, n_3)$$

Since $m^3 < (n_1\, n_2\, n_3)$, we must have $X = m^3$ identically over the integers. Thus, in polynomial time the attacker computes $m = \sqrt[3]{X}$ $\square$

Security of the RSA cryptoscheme

Fact 6

Generating two moduli sharing a prime exposes both of them to trivial factoring. In fact, if $n_1 = p\,q_1$ and $n_2 = p\,q_2$, p can be obtained as $\gcd(n_1, n_2)$.

Lemma 4, and Lemma 5 show that there are particular situations where it is possible to break RSA without solving the factoring problem!
Key points to bring home:

- the plaintext should be randomly padded before transmission. That way the "same message" is never encrypted to two different people.
- In addition, very small public exponents should be avoided: $e = 65537 = 2^{16} + 1$ is the usual choice currently in use.
- the primes for moduli generations should be picked through employing a sound (P)RNG initialized <u>with at least a 128-bit long seed</u>

Weaknesses w.r.t. Chosen Ciphertext Attacks (1)

We now show a trivial example of the reasons because the presented "textbook RSA" is not secure w.r.t. CCAs.

- In this scenario, an adversary is supposed to choose as many ctxs as she wishes and to obtain their correct decryptions under the same unknown key from an Oracle.

  (You can think to a portable device running a RSA deciphering primitive with the decryption key securely stored in it.
  You can use it only to decipher ctxs different from the one that is interesting for you!)
- From these pieces of information the adversary can attempt to recover either
  - the hidden secret key or
  - the ptx corresponding to a ctx not chosen by him

Weaknesses w.r.t. Chosen Ciphertext Attacks (2)

In the case of RSA scheme:

if an opponent wants to derive the message m corresponding to a certain ctx $c = m^e \bmod n$

- he can generate a random number $x \in \mathbb{Z}_n$
- compute $c' = (c \cdot x^e) \bmod n$
- submit to the Oracle a decryption request for c'

Clearly, the Oracle will return the msg:

$$m' = (c')^d \bmod n$$

Therefore, the opponent can compute the following:

$$m' = c^d \cdot x^{e\,d} \bmod n \Rightarrow m = m'\,x^{-1} \bmod n$$

Weaknesses w.r.t. Chosen Ciphertext Attacks (3)

In order to avoid these kind of problems the RSA standard force to encrypt a proper padding of the original message though a carefully designed bijective function $\phi : \mathbb{Z}_n \mapsto \mathbb{Z}_n$, that is publicly known:

$$\phi(a \cdot b) \neq \phi(a) \cdot \phi(b)$$

In order to encrypt a message $m \in \mathbb{Z}_n$ the standard forces to compute:

$$c = (\phi(m))^e \bmod n$$

Thus if an oracle is asked to decrypt $c' = (c \cdot x^e) \bmod n$, where x is a random number, then the returned value will be:

$$m' = \phi^{-1}(c^d \cdot x^{e\,d} \bmod n) = \phi^{-1}(\phi(m) \cdot x \bmod n)$$

while m would be equal to:

$$m = \phi^{-1}(c^d \bmod n) = \phi^{-1}(\phi(m))$$

Then,

$$m' \neq m \cdot \phi^{-1}(x)$$

Optimized Asymmetric Encryption Padding (OAEP)

By far, the most successful padding scheme in use today (corresponding to the $\phi()$ function, previously introduced) was invented by Bellare and Rogaway and is called OAEP or “Optimized Asymmetric Encryption Padding”.

OAEP is a padding scheme which can be used with any function f which is a *one-way trapdoor permutation* (e.g., RSA).

OAEP satisfies the following two goals:

- Add an element of randomness to each ptx: two equal ptxs (encrypted with the same key) will result in two different ctxs.
- Prevent partial decryption of ciphertexts through ensuring that an adversary cannot recover any portion of the ptx without being able to invert the *trapdoor one-way permutation* f.

OAEP

Let f be any N-bit to N-bit trapdoor one-way permutation, e.g. for $N = 1024$ take f as the RSA function $c \equiv_n m^e$, $N = \lceil \log_2(n) \rceil$.
Let k_0 and k_1 denote numbers such that a computational effort of 2^{k_0} or 2^{k_1} is impossible (e.g. $k_0, k_1 > 80$).
Let H, G be two hash functions:

$$\begin{array}{lll} G: & \{0,1\}^{k_0} & \rightarrow & \{0,1\}^{N-k_0} \\ H: & \{0,1\}^{N-k_0} & \rightarrow & \{0,1\}^{k_0} \end{array}$$

Let m be a message of $N - k_0 - k_1$ bits in length.
We then encrypt using the function

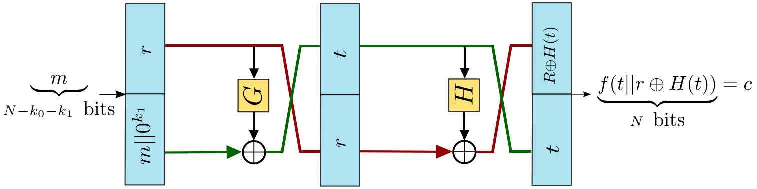
$$Enc(m) = f\left(\{m||0^{k_1} \oplus G(r)\} \, || \, \{r \oplus H(m||0^{k_1} \oplus G(r)))\}\right)$$

- $||$ denotes bit-concatenation
- r is a random bit string of length k_0
- $m||0^{k_1}$ means m followed by k_1 zero bits

OAEP as a 2-stage Feistel network

$G : \{0,1\}^{k_0} \mapsto \{0,1\}^{N-k_0}$
$H : \{0,1\}^{N-k_0} \mapsto \{0,1\}^{k_0}$

Encryption



$\underbrace{m}_{N-k_0-k_1}$ bits
r
$m||0^{k_1}$
G
t
r
H
$R\oplus H(t)$
t
$\underbrace{f(t||r \oplus H(t))}_{N \text{ bits}} = c$

$t{=}m||0^{k_1}\oplus G(r)$: encoded with $N{-}k_0$ bits

Decryption

compute $f^{-1}(c){=}t||r\oplus H(t)\ldots$ extract t and compute $H(t)$, then derive r from $(r\oplus H(t))$, compute $G(r)$ and recover m

Reject the message in case $T\oplus G(r)$ does not have k_1 trailing 0s

Security of the RSA cryptoscheme

Theorem *(RSA-OAEP Security)*

In the random oracle model, if we model the hash functions G and H as "**truly** random functions" then RSA-OAEP is proven to be secure against *adaptive chosen ciphertext attacks* (CCA2) and *chosen plaintext attacks* (CPA) if the RSA assumption holds.

The most common choices for the parameters of RSA-OAEP are:
$k_0 = k_1 = 80$, while G and H are defined through composing either SHA-1 or RIPEMD-160 hash functions