RSA

Martin Stanek

Department of Computer Science
Comenius University
stanek@dcs.fmph.uniba.sk

Cryptology 1 (2020/21)

Content

Initialization

parameters, private and public transformations

Correctness of RSA

basic facts from number theory

Euler's theorem

Implementation

modular exponentiation, choosing primes, etc.

choosing public exponent, optimization of private transformation

real-world implementation (openssl examples)

paddings

Introduction

- RSA – Ron Rivest, Adi Shamir, Leonard Adleman (1977)
- Clifford Cocks (1973), declassified in 1997
- one of the most frequently used public-key schemes
- encryption scheme / digital signature scheme

Initialization (key generation)

1. choose large, distinct primes p, q (e.g. 1024 bits long)
2. let $n = p \cdot q$ (public modulus)
3. choose e such that $\gcd(e, \varphi(n)) = 1$
   - φ is Euler's totient function
   - $\varphi(n) = (p-1)(q-1)$
4. compute d such that $e \cdot d \equiv 1 \pmod{\varphi(n)}$

- public key: (e, n); e public exponent
- private key: (d, n); d private exponent
- additional values are often stored as a part of the private key to speed up the private transformation

Encryption and decryption

- textbook/plain RSA
- encryption, decryption: $E, D : \mathbb{Z}_n \to \mathbb{Z}_n$:

$$E(m) = m^e \bmod n$$
$$D(c) = c^d \bmod n$$

- small example:
  - $p = 11$, $q = 19$, $n = 11 \cdot 19 = 209$, $\varphi(209) = 10 \cdot 18 = 180$
  - $e = 7$, $d = 7^{-1} \bmod 180 = 103$
  - public key: $(7, 209)$; private key: $(103, 209)$
  - let $m = 100$: encryption $E(100) = 100^7 \bmod 209 = 111$
  - decryption $D(111) = 111^{103} \bmod 209 = 100$

Basic facts from number theory 1

- Notation (for positive integer n):
  - $\mathbb{Z}_n = \{0, 1, \ldots, n-1\}$
  - $\mathbb{Z}_n^* = \{a \,;\; a \in \mathbb{Z}_n \;\&\; \gcd(a, n) = 1\}$
- Euler's totient function: $\varphi(n) = |\mathbb{Z}_n^*|$
  - $\varphi(8) = |\{1, 3, 5, 7\}| = 4$
  - $\varphi(p) = |\{1, 2, \ldots, p-1\}| = p - 1$ for prime p
  - $\varphi(p \cdot q) = (p-1)(q-1)$ for product of two distinct primes
- $a \equiv b \pmod{n} \;\Leftrightarrow\; n \mid (a - b)$

Basic facts from number theory 2

Lemma 1

Let $ka \equiv kb \pmod{n}$ *for positive integer* n *and integers* a, b, k. *Let* $\gcd(k, n) = 1$. *Then* $a \equiv b \pmod{n}$.

Proof.

$$ka \equiv kb \pmod{n} \implies n \mid k(a-b)$$

Since n and k are coprime, we have $n \mid (a-b)$ □

Basic facts from number theory 3

Lemma 2

Let $\mathbb{Z}_n^* = \{a_1, \ldots, a_{\varphi(n)}\}$. *Let* k *be an integer such that* $\gcd(k, n) = 1$.
Then $\{ka_1 \bmod n, \ldots, ka_{\varphi(n)} \bmod n\} = \mathbb{Z}_n^*$

Proof.

1. $\gcd(a_i, n) = 1,\ \gcd(k, n) = 1 \implies \gcd(ka_i, n) = 1$
   Hence $\{ka_1 \bmod n, \ldots, ka_{\varphi(n)} \bmod n\} \subseteq \mathbb{Z}_n^*$
2. $\gcd(k, n) = 1,\ ka_i \equiv ka_j \pmod{n} \implies a_i \equiv a_j \pmod{n}$ (Lemma 1)
   $\implies i = j$, and therefore all elements in the set are distinct.

□

Euler's theorem

Theorem 3 (Euler)

Let n be a positive integer. Then for an arbitrary integer a coprime to n:

$$a^{\varphi(n)} \equiv 1 \pmod{n}.$$

Proof.

Let $\mathbb{Z}_n^* = \{a_1, \ldots, a_{\varphi(n)}\}$, so $\mathbb{Z}_n^* = \{aa_1 \bmod n, \ldots, aa_{\varphi(n)} \bmod n\}$ (Lemma 2).
We compute the product of all elements:

$$\prod_{i=1}^{\varphi(n)} a_i \equiv \prod_{i=1}^{\varphi(n)} a \cdot a_i \equiv a^{\varphi(n)} \prod_{i=1}^{\varphi(n)} a_i \pmod{n}$$

Since $\gcd(a_i, n) = 1$, the product is coprime to n as well. Applying Lemma 1 we get $a^{\varphi(n)} \equiv 1 \pmod{n}$. □

Euler's theorem – remarks

- Fermat's little theorem:
  Let p be a prime, and a be an integer. If $p \nmid a$ then $a^{p-1} \equiv 1 \pmod{p}$.
- FLT is direct corollary of Euler's theorem:
  - $p \nmid a \Leftrightarrow \gcd(a, p) = 1$
  - $\varphi(p) = p - 1$
- Carmichael's function $\lambda(n)$: smallest positive integer such that $a^{\lambda(n)} \equiv 1 \pmod{n}$ for every integer a coprime to n
  - $\lambda(p) = p - 1$, $\lambda(p^l) = p^{l-1}(p-1)$ for prime p
  - $\lambda(n) = \operatorname{lcm}(\lambda(p_1^{l_1}), \lambda(p_2^{l_2}), \ldots, \lambda(p_k^{l_k}))$, where $n = p_1^{l_1} p_2^{l_2} \ldots p_k^{l_k}$ is a prime decomposition
  - generalization of Euler's theorem ($a^{\lambda(n)} \equiv 1 \pmod{n}$ for a coprime to n)
  - sometimes the RSA is specified in terms of $\lambda(n)$
  - $\lambda(p \cdot q) = \operatorname{lcm}(p-1, q-1)$

Correctness of RSA

Theorem 4 (Correctness of RSA)

Let E and D be the encryption and decryption functions in RSA scheme. Then

$$\forall m \in \mathbb{Z}_n : \ D(E(m)) = m.$$

Proof.

Case 1: $\gcd(m, n) = 1$; the most frequent case

$$\begin{aligned} D(E(m)) &= (m^e)^d \bmod n = m^{1+k\varphi(n)} \bmod n \\ &= m \cdot (\underbrace{m^{\varphi(n)}}_{1})^k \bmod n \qquad \text{(Euler's theorem)} \\ &= m \bmod n = m \end{aligned}$$

Correctness of RSA 2 (proof cont.)

Case 2: $\gcd(m, n) > 1$, rare event
(you can factorize n if this happens for $m \neq 0$)

- trivially valid if $m = 0$
- wlog we assume $m = m' \cdot p^l$ for $l \geq 1$ and $\gcd(m', n) = 1$
- $D(E(m)) = (m'p^l)^{ed} \bmod n = m' \cdot (p^{1+k\varphi(n)})^l \bmod n$ using case 1
- evaluating expression $p^{1+k\varphi(n)} \bmod n$:

$$
\begin{aligned}
p^{(q-1)} &\equiv 1 \pmod{q} \qquad \text{(FLT)} \\
p^{k(q-1)(p-1)} \equiv p^{k\varphi(n)} &\equiv 1 \pmod{q} \\
p^{k\varphi(n)} &= 1 + t \cdot q \quad \Rightarrow \quad p^{1+k\varphi(n)} = p + t \cdot n
\end{aligned}
$$

- therefore $p^{1+k\varphi(n)} \equiv p \pmod{n}$ and $D(E(m)) = m' \cdot p^l \bmod n = m$

□

Correctness – remarks

- E, D are two mutually inverse bijections
- some fixed points: $E(0) = 0$, $E(1) = 1$, $E(n-1) = n-1$

Implementation – “how” & efficiency issues

- modular exponentiation
- primality testing
- computing private exponent (modular inverse)
- choosing public exponent for efficiency
- improving performance of private transformation by Chinese remainder theorem

Modular exponentiation

- compute $a^k \bmod n$ for (positive) integers a, k, n
- note that $\underbrace{a \cdot a \cdot \ldots \cdot a}_{k} \bmod n$ is an exponential algorithm w.r.t. $|k|$

polynomial time algorithm:

$v = 1$
while $(k > 0)$
    if k is odd: $v = v \cdot a \bmod n$
    $a = a^2 \bmod n$
    $k = k/2$ // *integer division*
return v

example: $a^{21} \bmod n$

$(k, v, a)_{\text{before}}$	$(k, v, a)_{\text{after}}$
$(21, 1, a)$	$(10, a, a^2)$
$(10, a, a^2)$	$(5, a, a^4)$
$(5, a, a^4)$	$(2, a^5, a^8)$
$(2, a^5, a^8)$	$(1, a^5, a^{16})$
$(1, a^5, a^{16})$	$(0, a^{21}, a^{32})$

- other improvements: sliding window, Montgomery reduction

Choosing primes

- primes should be secret (otherwise an attacker can easily compute d)
- procedure: random choice of odd integer + primality testing
- density of primes:
  - $\pi(n)$ – number of primes less than or equal to n
  - Prime number theorem: $\pi(n) \sim n/\ln(n)$
  - bounds for $\pi(n)$, e.g. for $n \geq 55$: $n/(\ln n + 2) < \pi(n) < n/(\ln n - 4)$
  - these bounds can be used to estimate the probability that random odd number of certain length is a prime
- real experiment (average from 300 samples):

bit length	count
128	44
512	180
786	271
1024	353

Primality testing

- deciding primality is in **P**
  - AKS primality test (2002); slow, not used in practice
- probabilistic tests offer better performance: Miller-Rabin, Lucas, etc.
- Miller-Rabin test (and its variants or combination with other tests) is the most common choice
  - FIPS 186-4 Digital Signature Standard
  - openssl implementation, ...

Miller-Rabin test

- input: odd n; let $n - 1 = t \cdot 2^s$ for odd integer t
- n is *strong pseudoprime* to a base a (where $1 \leq a < n$) if:

$$a^t \equiv 1 \pmod{n} \quad \vee \quad \exists r \in \mathbb{Z}_s : a^{t \cdot 2^r} \equiv -1 \pmod{n}$$

- if n is prime, then n is strong pseudoprime to every base
- if n is composite, then the probability that n is strong pseudoprime to a random base is at most $1/4$
  - probability of error after k independent choices of the base is 4^{-k}
  - much smaller for most n
- repeated squaring for the second part of strong pseudoprime property
- some insight into the property:
  - if n is prime: $a^{n-1} \equiv a^{t \cdot 2^s} \equiv 1 \pmod{n}$ for all a not divisible by n (FLT)
  - if n is prime: 1 has exactly two square roots modulo n;
    $x^2 \equiv 1 \pmod{n} \implies n \mid (x+1)(x-1) \implies x \equiv \pm 1 \pmod{n}$

Computing private exponent

- $d = e^{-1} \bmod \varphi(n)$
- Extended Euclidean algorithm
  - input: integers a, b
  - output: $\gcd(a, b)$, integers x, y such that $xa + yb = \gcd(a, b)$
  - remark: returning gcd is redundant if x, y are known
  - simple recursive version (for integers $a, b \geq 0$):

    EEA(a,b)
    if $b = 0$: return$(a, 1, 0)$
    $(d, x, y) \leftarrow$ EEA$(b, a \bmod b)$
    return $(d, y, x - y \cdot \lfloor a/b \rfloor)$
- EEA$(e, \varphi(n)) \mapsto (1, x, y)$: $xe + y\varphi(n) = 1 \implies d = x \bmod \varphi(n)$

Choosing public exponent

- improving performance: small public exponent
- common choice $e = 65537 = (1000 \ldots 0001)_2$
  - it is a prime and with high probability coprime to $\varphi(n)$
  - if $e = 65537$ is desired, we can test $\gcd(e, p-1) = 1$ (and for q as well) while generating the primes
  - nice binary representation (short and small number of ones)

Chinese remainder theorem

- used in various constructions and implementations with modular arithmetic

Theorem 5 (CRT)

Let $n_1, \ldots, n_k$ are pairwise coprime positive integers. Then the following system of congruences (where $a_1, \ldots, a_k$ are arbitrary integers):

$$\begin{aligned} x &\equiv a_1 \pmod{n_1} \\ &\ldots \\ x &\equiv a_k \pmod{n_k} \end{aligned}$$

has a solution. Additionally, all solutions of the system are mutualy congruent modulo $N = n_1 \cdot \ldots \cdot n_k$.

Chinese remainder theorem – proof

Proof.

1. Let $N_i = N/n_i$ for $i = 1, \ldots, k$, and $M_i = N_i^{-1} \bmod n_i$.
   Solution $x = \sum_{i=1}^{k} a_i N_i M_i$ can be easily verified (for $j \in \{1, \ldots, k\}$):

$$x \equiv a_j \underbrace{N_j M_j}_{1 \bmod n_j} + \sum_{\substack{i=1 \\ i \neq j}}^{k} a_i M_i \underbrace{N_i}_{0 \bmod n_j} \equiv a_j \pmod{n_j}.$$

2. Let x and x' are two solutions. Therefore

$$\begin{aligned} x &\equiv x' \pmod{n_1} \\ &\ldots \\ x &\equiv x' \pmod{n_k} \end{aligned} \qquad \Rightarrow \quad \forall i : \; n_i \mid (x - x')$$

   Since n_i are pairwise coprime we have $N \mid (x - x')$.

□

Corollary of the CRT

- $n = p \cdot q$, $\gcd(p, q) = 1$

$$\begin{aligned} x &\equiv a \pmod{p} \\ x &\equiv a \pmod{q} \end{aligned} \qquad \Longleftrightarrow \qquad x \equiv a \pmod{n}$$

- ($\Rightarrow$) value a is a solution;
  according to the CRT, if x is also a solution, then $x \equiv a \pmod{n}$
- ($\Leftarrow$) trivial: $x = a + t \cdot pq \;\Rightarrow\; x = a + (tq) \cdot p \;\Rightarrow\; x \equiv a \pmod{p}$
  (similarly for q)

Optimization of private transformation $D(c)$

- idea: instead of $c^d \bmod n$ compute $c^d \pmod p$, $c^d \pmod q$ and combine results using the CRT
- for unknown m:

$$\begin{aligned} m &\equiv c^d \pmod p \\ m &\equiv c^d \pmod q \end{aligned} \iff m \equiv c^d \pmod n$$

we can obtain m as follows:

$$m = m_p \cdot q(q^{-1} \bmod p) + m_q \cdot p(p^{-1} \bmod q) \bmod n$$

where $m_p = c^d \bmod p = c^{d \bmod (p-1)} \bmod p$ and similarly $m_q = c^d \bmod q = c^{d \bmod (q-1)} \bmod q$

- two "half-size" modular exponentiations are faster than one "full-size"
- p, q – part of private key; pre-computed inverses

Optimization in real world

- private key includes: $p, q, d_p = d \bmod (p-1), d_q = d \bmod (q-1), q_{\text{inv}} = q^{-1} \bmod p$
- computation of $D(c)$:
  1. $m_p = c^{d_p} \bmod p$, $m_q = c^{d_q} \bmod q$
  2. $m = m_q + q(q_{\text{inv}}(m_p - m_q) \bmod p)$

- the correctness of the result can be easily verified by checking:

$$m \bmod p = m_p$$
$$m \bmod q = m_q$$

and noticing that $0 \leq m < pq$

Real-world implementation (playing with openssl)

- creating 2048-bit RSA instance:
  `openssl genrsa -out myrsa.pem 2048`
- What is inside?
  `openssl rsa -in myrsa.pem -text`
  - modulus: n
  - publicExponent: e
  - privateExponent: d
  - prime1, prime2: p, q
  - exponent1, exponent2: $d \bmod (p-1)$, $d \bmod (q-1)$
  - coefficient: $q_{\text{inv}} = q^{-1} \bmod p$
- extracting public key:
  `openssl rsa -in myrsa.pem -pubout -out myrsa-pub.pem`

PKCS#1 v1.5 padding

- why padding at all?
  - to randomize encryption (plain RSA is deterministic)
  - prove the security of the scheme (OAEP)
- PKCS#1 v1.5 (still used, potential implementation problems, not recommended)
- padded plaintext m:

$$0x00 \,\|\, 0x02 \,\|\, \mathrm{PS} \,\|\, 0x00 \,\|\, m$$

- PS – string of pseudo-random nonzero bytes of length ≥ 8
- various recommendation on using this padding (see RFC 8017)

Encrypting with PKCS#1 v1.5 padding (openssl)

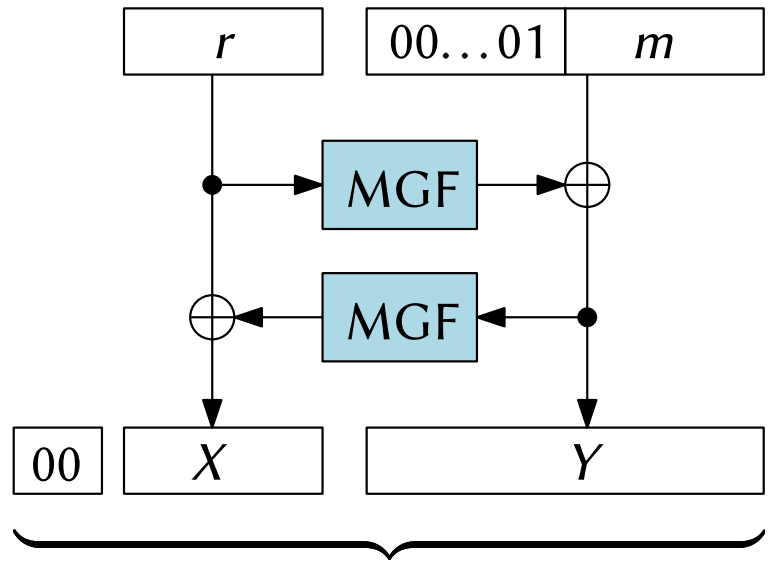
- encryption of short string with RSA using PKCS#1 v1.5 padding (default)
```
echo 'Cryptology - sample plaintext' | openssl
pkeyutl -encrypt -pubin -inkey myrsa-pub.pem -out
cipher.bin
```
- decrypting the ciphertext:
```
openssl pkeyutl -decrypt -inkey myrsa.pem -in
cipher.bin
```
- see the padding:
```
openssl pkeyutl -decrypt -inkey myrsa.pem -in
cipher.bin -hexdump -pkeyopt rsa_padding_mode:none
```

```
0000 - 00 02 e6 88 63 5a d2 e2-57 ee d7 74 6a 78 1a 6a   ....cZ..W..tjx.j
0010 - 75 b2 57 3b 6a d7 bb f8-3b 61 c1 23 bc 77 18 91   u.W;j...;a.#.w..
0020 - bf 2d 36 ed a8 25 30 16-44 d5 ba 88 1e 7c 3e 12   .-6..%0.D....|>.
0030 - d1 c7 b0 7a 3e 5a 47 b3-87 66 7f f0 47 c4 89 c2   ...z>ZG..f..G...
0040 - 69 aa 17 cb e3 96 4f d2-43 8f 4d c5 8c 47 e6 91   i.....O.C.M..G..
0050 - cd e6 0a c0 a3 15 72 9d-0f cb 17 4c 85 43 97 5f   ......r....L.C._
0060 - 52 d1 67 e9 92 8d c5 85-52 ed 15 c7 28 c9 33 b9   R.g.....R...(.3.
0070 - dc 5d 84 b8 6c 26 e3 1a-d7 5a 9a f3 44 cf f1 0d   .]..l&...Z..D...
0080 - 70 b3 3a 22 c1 37 82 e0-7d 24 1d 6d 3e ef e3 b9   p.:".7..}$.m>...
0090 - 3f 2c 56 32 69 27 8a 61-a6 72 3f 98 f3 e7 b2 3a   ?,V2i'.a.r?....:
00a0 - 68 e6 ff c9 cd 90 c6 1b-fc 2a 5e c9 bd cf 38 0c   h........*^...8.
00b0 - 8d 97 08 75 f1 d7 a4 a3-7c 3e 1e 93 31 33 f2 fd   ...u....|>..13..
00c0 - a7 92 f2 1d 61 0d 8c c4-98 df a2 2d ba ca a9 91   ....a......-....
00d0 - 30 f3 a7 89 b5 02 b1 3f-d4 b2 20 e1 b0 b0 6d d0   0......?.. ...m.
00e0 - cd 00 43 72 79 70 74 6f-6c 6f 67 79 20 2d 20 73   ..Cryptology - s
00f0 - 61 6d 70 6c 65 20 70 6c-61 69 6e 74 65 78 74 0a   ample plaintext.
```

OAEP padding

- OAEP – Optimal Asymmetric Encryption Padding
- recommended for new applications (PKCS #1 v2.2, RFC 8017)
- provable secure (in some sense, in some security model)
- slightly simplified OAEP

- 2 round Feistel
- MGF – mask generation function, based on hash function (RO)
- r – random "seed"
- padding verified in decryption
- impossible to create valid ciphertext without encryption



input to RSA encryption

Encrypting with OAEP padding (openssl)

- encrypting a message:
  ```
  echo 'Cryptology - sample plaintext' | openssl
  pkeyutl -encrypt -pubin -inkey myrsa-pub.pem -out
  cipher2.bin -pkeyopt rsa_padding_mode:oaep
  ```
- decrypting the ciphertext:
  ```
  openssl pkeyutl -decrypt -inkey myrsa.pem -in
  cipher2.bin -pkeyopt rsa_padding_mode:oaep
  ```