RSA

Cryptology (1)

Martin Stanek

2025

KI FMFI UK Bratislava

Public-key (asymmetric) cryptography

- problems with secret-key (symmetric) cryptography
  - encryption – all parties need to know the key
  - distributing the key
- idea of public-key cryptography
  - user generates a related pair of keys – public and private
  - private key can’t be computed from the public key efficiently
- public-key schemes
  - encryption schemes, digital signature schemes, key agreement protocols, …
  - often built on computationally hard problems: factorization, discrete logarithm, learning with errors, …

Public vs. private key

- public-key
  - encryption (in asymmetric encryption schemes)
  - verification of signatures (in digital signature schemes), etc.
  - can be distributed freely, anyone can encrypt data for the user or verify user's signatures
  - How to ensure the authenticity of the public key? PKI?
- private-key
  - decryption (asymmetric encryption schemes)
  - signing (digital signature schemes), etc.
  - should be kept private

Public-key encryption scheme (informally): $\langle \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec} \rangle$

- $\mathsf{Gen}(1^k)$ – a PPT algorithm; it produces a key pair $(\mathsf{pk}, \mathsf{sk})$
  - k is a security parameter
  - plaintext space is fixed or implied by pk
- $\mathsf{Enc}_{\mathsf{pk}}(m)$ – a PPT algorithm; it computes a ciphertext from a plaintext m and pk
- $\mathsf{Dec}_{\mathsf{sk}}(c)$ – (deterministic) PT algorithm; computes a plaintext from a ciphertext c and sk
- requirements:
  - correctness: $\forall(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^k)\ \forall m : \mathsf{Dec}_{\mathsf{sk}}\big(\mathsf{Enc}_{\mathsf{pk}}(m)\big) = m$
  - efficiency: (probabilistic) polynomial time
  - security

Initialization (key generation)

1. choose large, distinct primes p, q (e.g. 1024 bits)
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 $e \cdot d \equiv 1 \pmod{\varphi(n)}$

Ron Rivest, Adi Shamir, Leonard Adleman (1977)

- Clifford Cocks (1973), declassified in 1997

encryption scheme & digital signature scheme

- 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 computation

Encryption and decryption

- textbook/plain RSA
- encryption and 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 $D(111) = 111^{103} \bmod 209 = 100$

Correctness of RSA

Basic facts from the number theory (1)

- notation (for positive integer n):
  - $\mathbb{Z}_n = \{0, 1, \ldots, n-1\}$
  - $\mathbb{Z}_n^* = \{a \mid a \in \mathbb{Z}_n \wedge \gcd(a, n) = 1\}$
- 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
- $a \equiv b \pmod{n} \quad \Leftrightarrow \quad n \mid (a - b)$

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. $ka \equiv kb \pmod{n} \Rightarrow n \mid k(a-b)$; since n and k are coprime, we have $n \mid (a-b)$ ■

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. $ka \equiv kb \pmod{n} \Rightarrow n \mid k(a-b)$; since n and k are coprime, we have $n \mid (a-b)$ ■

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

Theorem (Euler). Let n be a positive integer. Then for an arbitrary integer a coprime to n, i.e., $\gcd(a, n) = 1$:

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

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

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

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 $a \in \mathbb{Z}$ coprime to n
  - $\lambda(p) = p - 1, \lambda(p^l) = p^{l-1}(p-1)$ for a prime number p
  - $\lambda(n) = \mathrm{lcm}(\lambda(p_1^{l_1}), \lambda(p_2^{l_2}), \ldots, \lambda(p_k^{l_k}))$, where $n = p_1^{l_1} p_2^{l_2} \cdot \ldots \cdot p_k^{l_k}$ is a prime decomposition
  - generalization of Euler's theorem ($a^{\lambda(n)} \equiv 1 \pmod{n}$ for a coprime to n)
  - sometimes RSA is specified in terms of $\lambda(n)$; $\lambda(p \cdot q) = \mathrm{lcm}(p-1, q-1)$

Theorem (Correctness of RSA). Let E and D be the encryption and decryption functions in RSA scheme. Then

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

Remarks:

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

Proof.

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

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

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 $\gcd(m', n) = 1$
- $D(E(m)) = \left(m'p^l\right)^{ed} \bmod n = m' \cdot \left(p^{1+k\varphi(n)}\right)^l \bmod n$ (using Case 1)
- evaluating expression $p^{1+k\varphi(n)} \bmod n$:

$$p^{q-1} \equiv 1 \pmod{q} \quad \text{(FLT)}$$

$$p^{k(q-1)(p-1)} \equiv p^{k\varphi(n)} \equiv 1 \pmod{q}$$

$$p^{k\varphi(n)} = 1 + t \cdot q \quad \Rightarrow \quad p^{1+k\varphi(n)} = p + t \cdot n$$

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

Implementation

Implementation – how?

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

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

$a^{21} \bmod n$: (k, v, a) values before and after iteration	
$(21, 1, a)$	$(10, a, a^2)$
$(10, a, a^2)$	$(5, a, a^4)$
$(5, a, a^4)$	$(2, a^5, a^8)$
$(2, a^5, a^8)$	$(1, a^5, a^{16})$
$(1, a^5, a^{16})$	$(0, a^{21}, a^{32})$

- other improvements: sliding window, Montgomery reduction

- 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: $\pi(n) \approx n/\ln(n)$
- experiment (average from 50 samples):

bit length	avg. tests
256	137
512	171
786	325
1024	435

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, ...

- 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:

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

- prime n: strong pseudoprime to every base
- composite n: the probability that n is strong pseudoprime to a random base is $\leq 1/4$
  - probability of error after k independent choices of the base is $\leq 4^{-k}$
  - much smaller for most n
- repeated squaring for the second part of $(\star)$

Miller-Rabin test – remarks

- 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:
  $x^2 \equiv 1 \pmod n \quad \Rightarrow \quad n \mid (x+1)(x-1) \quad \Rightarrow \quad x \equiv \pm 1 \pmod n$
- $(\star)$ … we get 1 at the end, and we get it a “corect way”

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 $a, b \geq 0$):

$$
\begin{aligned}
&\mathrm{EEA}(a, b): \\
&\qquad \text{if } b = 0 : \text{return } (a, 1, 0) \\
&\qquad (d, x, y) = \mathrm{EEA}(b, a \bmod b) \\
&\qquad \text{return } (d, y, x - y \cdot \lfloor a/b \rfloor)
\end{aligned}
$$

- $\mathrm{EEA}(e, \varphi(n)) \mapsto (1, x, y)$: $\quad xe + y\varphi(n) = 1 \Rightarrow d = x \bmod \varphi(n)$

Choosing public exponent

- improving performance: small public exponent
- common choice $e = 65537 = (1000...0001)_2$
  - 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; small number of ones)

– used in various constructions and implementations with modular arithmetic

Theorem (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):

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

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

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 $j \in \{1, \ldots, k\}$):

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

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

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

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

Corollary of the CRT

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

$$\begin{matrix} x \equiv a \pmod{p} \\ x \equiv a \pmod{q} \end{matrix} \quad \Longleftrightarrow \quad x \equiv a \pmod{n}$$

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

Optimization of $D(c)$

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

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

  we can obtain m as follows:

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

  where $m_p = c^d \bmod p = c^{d \bmod (p-1)} \bmod p$, and $m_q = c^d \bmod q = c^{d \bmod (q-1)} \bmod q$
- two “half-size” modular exponentiations are faster than one full-size
- p, q – part of private key; pre-computed inverses

- 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. $m = m_q + q\big(q_{\text{inv}}(m_p - m_q) \bmod p\big)$
- the correctness of the result can be easily verified by checking:

$$m \bmod p = m_p$$

$$m \bmod q = m_q$$

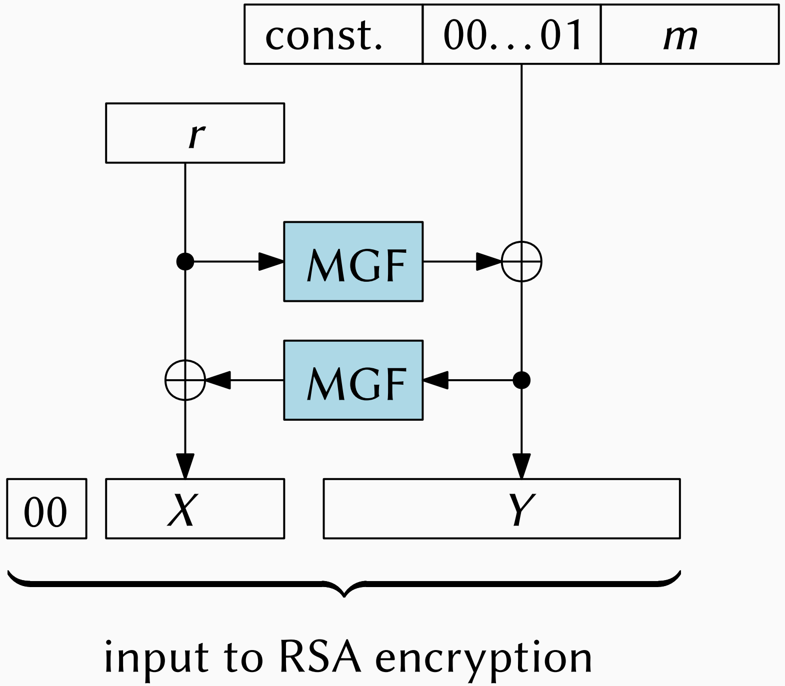
  and noticing that $0 \leq m < pq$

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: $00 \parallel 02 \parallel PS \parallel 00 \parallel m$
- PS – string of pseudorandom nonzero bytes of length ≥ 8
- various recommendation on using this padding (see RFC 8017)

OAEP (Optimal Asymmetric Encryption Padding)

- recommended, PKCS#1 v2.2, RFC 8017
- provable secure (in some sense, in some security model)
- slightly simplified presentation of OAEP (empty "label")
- 2 round Feistel
  - r – random seed
  - MGF – mask generation function, hash function based (RO)
- padding verified in decryption
- impossible to create valid ciphertext without encryption



Exercises

1. *Use* `openssl` *CLI to generate RSA key pair.*
   a) *Inspect the text representation of both keys.*
   b) *Encrypt and decrypt a short text.*
2. *Find a non-trivial fixed point for RSA encryption scheme. Use a concrete parameters for your computation. Non-trivial means “not in* $\{0, 1, -1\}$*”.*
3. *Perform an experiment in your preferred programming language, and test if two “half-size” exponentiations are faster than one full-size exponentiation.*