Lecture 9. RSA public-key encryption and signatures

Bruno Grenet

https://membres-ljk.imag.fr/Bruno.Grenet/IntroCrypto.html

Introduction to cryptology
Université Grenoble Alpes – IM²AG
M1 INFO, MOSIG & AM

Programming Techniques — S.L. Graham, R.L. Rivest* Editors

A Method for Obtaining Digital Signatures and Public-Key Cryptosystems

R. L. Rivest, A. Shamir, and L. Adleman
MIT Laboratory for Computer Science
and Department of Mathematics

An encryption method is presented with the novel property that publicly revealing an encryption key does not thereby reveal the corresponding decryption key. This has two important consequences:
(1) Couriers or other secure means are not needed to transmit keys, since a message can be enciphered using an encryption key publicly revealed by the intended recipient. Only he can decipher the message, since only he knows the corresponding decryption key.
(2) A message can be "signed" using a privately held decryption key. Anyone can verify this signature using the corresponding publicly revealed encryption key. Signatures cannot be forged, and a signer cannot later

I. Introduction

The era of "electronic mail" [10] may soon be upon us; we must ensure that two important properties of the current "paper mail" system are preserved: (a) messages are *private*, and (b) messages can be *signed*. We demonstrate in this paper how to build these capabilities into an electronic mail system.

At the heart of our proposal is a new encryption method. This method provides an implementation of a "public-key cryptosystem", an elegant concept invented by Diffie and Hellman [1]. Their article motivated our research, since they presented the concept but not any practical implementation of such a system. Readers familiar with [1] may wish to skip directly to Section V for a description of our method.

II. Public-Key Cryptosystems

In a "public-key cryptosystem" each user places in a public file an encryption procedure E. That is, the public file is a directory giving the encryption procedure of each user. The user keeps secret the details of his corresponding decryption procedure D. These procedures have the following four properties:

(a) Deciphering the enciphered form of a message M

A Method for Obtaining Digital Signatures and Public-Key Cryptosystems

R. Rivest, A. Shamir & L. Adleman (1978)

- Basics of RSA encryption scheme
- Signature using the encryption scheme in *reverse mode*

Pros

- First proposal of a public-key encryption scheme
- Use of computational difficulty as security

Cons

- As presented, the encryption scheme is completely unsafe!
- The signature is not a good idea!

Remark

- Already known to GHCQ (UK) in 1973, declassified only in 1997 Clifford Cocks

Contents of this lecture

1. The maths of RSA: the trapdoor permutation
   - $\mathbb{Z}/N\mathbb{Z}$ where $N = p \times q$
   - Designing a *trapdoor permutation*

   $\rightarrow \pm$ the content of the original paper

2. RSA encryption scheme
   - What should be added to obtain a proper encryption scheme?

3. RSA signatures
   - How to obtain a proper signature scheme?

Contents

Representation and ring operations

General context

$N = p \times q$ where p, q are prime numbers; computations *modulo* N

Representation and modular operations

- $\mathbb{Z}/N\mathbb{Z} = \{0, 1, \ldots, N-1\}$ with *modular* addition, subtraction and multiplication:
  1. Perform the operation in the integers
  2. Reduce the result *modulo* N
- Complexity: $O(\log^2 N)$ or $O(\log N \log \log N)$

Example: $\mathbb{Z}/35\mathbb{Z}$

$21 + 18 = 39 = 4$ $\qquad -12 = 23$

$5 \times 10 = 50 = 15$

Detour by a fundamental algorithm

The extended Euclidean Algorithm (xGCD)

Input: $a, b \in \mathbb{Z}, a > b \geq 0$

Output: $g, u, v \in \mathbb{Z}$ s.t. $g = au + bv$ and $g = \gcd(a, b)$

1. $(r_0, u_0, v_0) \leftarrow (a, 1, 0)$
2. $(r_1, u_1, v_1) \leftarrow (b, 0, 1)$
3. $i \leftarrow 2$
4. while $r_{i-1} \neq 0$:
5. $\quad (q_i, r_i) \leftarrow \textsc{QuoRem}(r_{i-2}, r_{i-1})$
6. $\quad (u_i, v_i) \leftarrow (u_{i-2} - q_i u_{i-1}, v_{i-2} - q_i v_{i-1})$
7. $\quad i \leftarrow i + 1$
8. return $(r_{i-2}, u_{i-2}, v_{i-2})$

21 15

i	r_i		u_i	v_i	q_i
0	21	=	1 × 21 +	0 × 15	✓
1	15	=	0 × 21 +	1 × 15	✓
2	6	=	1 × 21 +	−1 × 15	1
3	3	=	−2 × 21 +	3 × 15	2
4	0	=	5 × 21 +	−7 × 15	2

$\hookrightarrow 3 = -2 \times 21 + 3 \times 15$

$3 = \gcd(21, 15)$

Detour by a fundamental algorithm

The extended Euclidean Algorithm (xGCD)

Input: $a, b \in \mathbb{Z}, a > b \geq 0$
Output: $g, u, v \in \mathbb{Z}$ s.t. $g = au + bv$ and $g = \gcd(a, b)$

1. $(r_0, u_0, v_0) \leftarrow (a, 1, 0)$
2. $(r_1, u_1, v_1) \leftarrow (b, 0, 1)$
3. $i \leftarrow 2$
4. while $r_{i-1} \neq 0$:
5. $\quad (q_i, r_i) \leftarrow \textsc{QuoRem}(r_{i-2}, r_{i-1})$
6. $\quad (u_i, v_i) \leftarrow (u_{i-2} - q_i u_{i-1}, v_{i-2} - q_i v_{i-1})$
7. $\quad i \leftarrow i + 1$
8. return $(r_{i-2}, u_{i-2}, v_{i-2})$

Correction

- For all i, $\gcd(a, b) = \gcd(r_i, r_{i+1})$
- For all i, $r_i = a \cdot u_i + b \cdot v_i$

Consequence

$\gcd(a, b) = 1 \iff$
there exists $u, v \in \mathbb{Z}$ s.t. $1 = a \cdot u + b \cdot v$

Complexity

The bit complexity of the extended Euclidean Algorithm is $O(\log(a) \log(b))$

Inversion and division in $\mathbb{Z}/N\mathbb{Z}$

Definition

$a \in \mathbb{Z}/N\mathbb{Z}$ is invertible if there exists $b \in \mathbb{Z}/N\mathbb{Z}$ s.t. $a \times b = 1$ *modular* $\times$

- one can *divide by a in* $\mathbb{Z}/N\mathbb{Z}$

Theorem

$a \in \mathbb{Z}/N\mathbb{Z}$ is invertible modulo N $\iff$ $\gcd(a, N) = 1$ $\iff$ $\exists u, v$ s.t. $1 = au + Nv$

(previous slide)

Proof

$\exists u, v,\ 1 = au + Nv \iff \exists u,\ 1 = au \bmod N \iff u$ is the inverse of $a \bmod N$

Algorithms

Inverse: Use the extended Euclidean Algorithm
Running time: $O(\log^2 N)$ or $O(\log N \log^2 \log N)$

Division: Use multiplication and inverse
Same running time

Invertible elements of $\mathbb{Z}/N\mathbb{Z}$

Definition

- The multiplicative group $\mathbb{Z}/N\mathbb{Z}^\times$ is the set of invertible elements of $\mathbb{Z}/N\mathbb{Z}$
- Its number of elements is denoted $\varphi(N)$

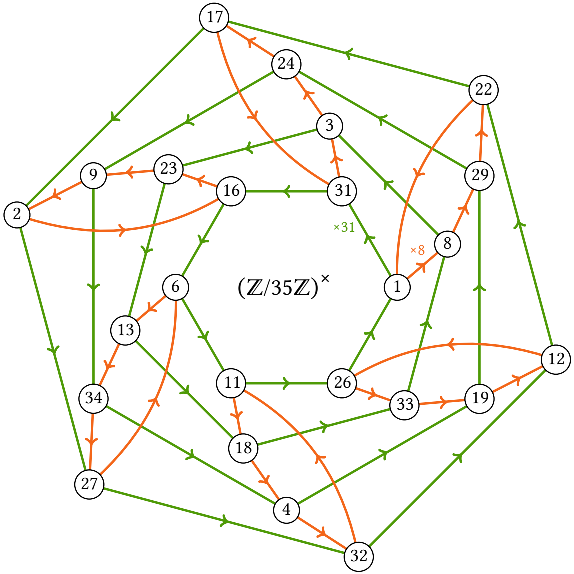
Proposition

If $N = p \times q$ with primes $p \neq q$, $\varphi(N) = (p-1)(q-1)$

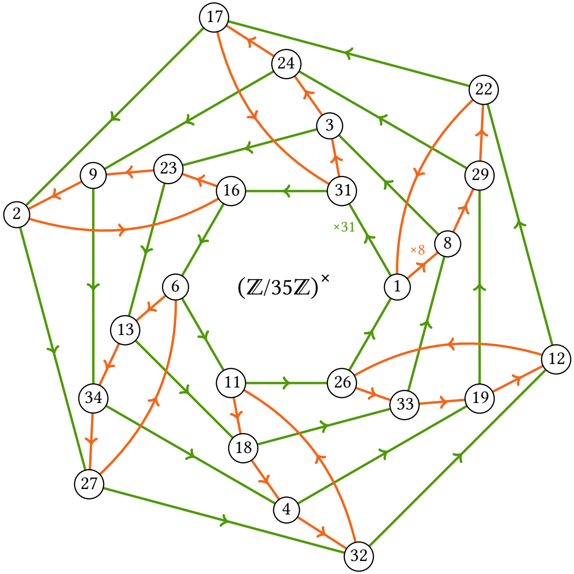
Proof

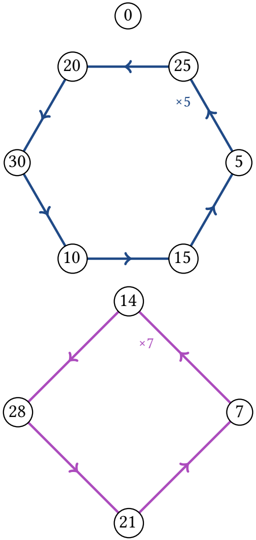
- a invertible $\iff \gcd(a,N) = 1 = \gcd(a, pq) \iff$ neither p nor q divides a
- Multiples of p: $0, p, 2p, 3p, \dots, (q-1)p$
  q: $0, q, 2q, 3q, \dots, (p-1)q$ } total: $q + p - 1$
- $\varphi(N) = pq - (p+q-1) = (p-1)(q-1)$

The multiplicative group is **not** cyclic!



The multiplicative group is **not** cyclic!





Non-invertible elements

The "RSA theorem"

Theorem

Let $N = p \times q$ with primes $p \neq q$. Then for all $a \in \mathbb{Z}/N\mathbb{Z}$, $a^{1+\varphi(N)} = a$.

Proof

(1) Fermat's little theorem: $\forall a \in \{1, \dots, p-1\}$, $a^{p-1} \bmod p = 1$

$$\{ax \bmod p : 1 \le x \le p-1\} = \{y : 1 \le y \le p-1\}$$

$$\Rightarrow \prod_{x=1}^{p-1} (ax) = \prod_{y=1}^{p-1} y \Rightarrow a^{p-1}(p-1)! = (p-1)! \Rightarrow a^{p-1} = 1 \quad (\text{all mod } p)$$

(2) $a^{1+\varphi(N)} = a^{1+(p-1)(q-1)}$:

$+ \; a^{p-1} = 1 \bmod p \Rightarrow a^{1+\varphi(N)} \bmod p = a \Rightarrow p$ divides $a^{1+\varphi(N)} - a$

$+ \; a^{q-1} = 1 \bmod q \Rightarrow a^{1+\varphi(N)} \bmod q = a \Rightarrow q$ divides $a^{1+\varphi(N)} - a$

$\Rightarrow pq$ divides $a^{1+\varphi(N)} - a \Rightarrow a^{1+\varphi(N)} \bmod N = a$

The RSA trapdoor permutation

The original (unsafe!) RSA encryption scheme

Definition as an encryption scheme

Public key: (N, e) where $N = p \times q$ with primes $p \neq q$ and $\gcd(e, \varphi(N)) = 1$

Private key: (N, d) where $d \times e \bmod \varphi(N) = 1$

Encryption: Given $m \in \mathbb{Z}/N\mathbb{Z}$, compute $c = m^e \bmod N$

Decryption: Given $c \in \mathbb{Z}/N\mathbb{Z}$, compute $m = c^d \bmod N$

Correction

$c^d = m^{ed}$. $ed \bmod \varphi(N) = 1 \Leftrightarrow \exists\, k$ s.t. $ed = 1 + k \cdot \varphi(N)$

$\hookrightarrow c^d = m^{1+k\varphi(N)} = m^{1+\varphi(N)} \cdot m^{(k-1)\varphi(N)} = m^{1+(k-1)\varphi(N)} = \cdots = m^{1+\varphi(N)} = m$.

The algorithms and complexities

Key generation

1. Generate two random primes $p \neq q$ — $O(\log^4 N)$
   - Sample random (odd) integers — $O(\log N)$ samples
   - Test their primality — $O(\log^3 N)$
2. Compute $N = p \times q$ and $\varphi(N) = (p-1) \times (q-1)$ — $O(\log^2 N)$
3. Generate e, d such that $e \times d \bmod \varphi(N) = 1$ — $O(\log^2 N)$
   - Sample random integers e — $1 + O(1/\sqrt{N})$ samples
   - Apply xGCD$(e, \varphi(N))$ to test invertibility and get d — $O(\log^2 N)$

Encryption and decryption

- Modular exponentiation in $\mathbb{Z}/N\mathbb{Z}$ — $O(\log^3 N)$
  - *Binary powering*, using $a^n = \begin{cases} a^{\lfloor n/2 \rfloor} \cdot a^{\lfloor n/2 \rfloor} & \text{for even } n \\ a \cdot a^{\lfloor n/2 \rfloor} \cdot a^{\lfloor n/2 \rfloor} & \text{for odd } n \end{cases}$

Remark

- All complexities can be slightly improved

Attacks on the trapdoor

Possible goals

Key recovery: Given (N, e), compute d s.t. $d \times e \bmod \varphi(N) = 1$

Plaintext recovery: Given (N, e) and c, compute m s.t. $m^e \bmod N = c$

Computational problems

Modular e-th root: Given N, c, e, compute m s.t. $m^e \bmod N = c$

Computation of φ: Given $N = p \times q$ (for unknown p, q), compute $\varphi(N) = (p-1)(q-1)$

Factorization: Given $N = p \times q$, compute p and q

Reductions between problems

- Plaintext recovery $\iff$ modular e-th root
- Computation of φ $\implies$ Key recovery $\implies$ plaintext recovery
- Computation of φ $\iff$ Factorization of N: ($\Leftarrow$) Compute p and q and return $(p-1)(q-1)$

($\Rightarrow$) $\varphi(N) = (p-1)(q-1) = N - (p+q) + 1$
↳ We know $N = pq$ and $S = p+q$

Consider $(X-p)(X-q) = X^2 - (p+q)X + pq = X^2 - SX + N$
↳ Compute the roots of the poly. to get p and q.

Integer factorization

Complexity of integer factorization

- Brute force algorithm: $O(\sqrt{N}) = O(2^{\frac{\log N}{2}})$
- ...
- General Number Field Sieve: $2^{O(\log^{\frac{1}{3}} N \log^{\frac{2}{3}} \log N)}$ *Lenstra, Lenstra (1993) and others...*
- Quantum algorithms:
  - $O(\log^3 N) = O(2^{3 \log \log N})$ *Shor (1994)*
  - $O(\log^{2.5} N) = O(2^{2.5 \log \log N})$ *Regev (2025)*

(Remark: no known NP-hardness result $\rightarrow$ could be polynomial in $\log N$)

Current record: 829-bit (250-digit) integer factorization

- Boudot, Gaudry, Guillevic, Heninger, Thomé, Zimmermann (Feb. 2020)
- Software: CADO-NFS
- Hardware: (mainly) academic clusters
- Approx. 2,700 core-years in a few months

Contents

The original RSA scheme is unsafe!

Deterministic encryption

- Two ciphertexts are equal iff the corresponding messages are equal
- The scheme cannot be IND-CPA/CCA secure

Examples of other difficulties

Small exponent: If e and m are small: $m^e \bmod N = m^e$ in $\mathbb{Z} \to \sqrt[e]{c}$ in $\mathbb{Z}$

Related messages: Given the ciphertexts of m and $m + \delta$ with small $\delta \to m$

Multiple receivers: Given the ciphertexts of m with several distinct keys $\to m$

> The original RSA encryption scheme is severely flawed and **should never be used!**

- Solution: use (random) padding

The padded RSA encryption scheme: overview

Construction

Parameters: n: number of bits of N; ℓ: length of the messages

$\mathrm{Gen}_n()$:
1. $p, q \twoheadleftarrow$ two random primes s.t. $p \times q$ has bit-length n
2. $N \leftarrow p \times q$, $\varphi(N) \leftarrow (p-1) \times (q-1)$
3. $e \twoheadleftarrow$ random integer invertible modulo $\varphi(N)$, $d \leftarrow e^{-1} \bmod \varphi(N)$
4. return $pk = (N, e)$, $sk = (N, d)$

$\mathrm{Enc}_{pk}(m)$: $m \in \{0,1\}^{\ell}$
1. $r \twoheadleftarrow \{0,1\}^{n-\ell}$
2. while $\hat{m} = r \| m \notin \mathbb{Z}/N\mathbb{Z}$: $r \twoheadleftarrow \{0,1\}^{n-\ell}$
3. return $c = \hat{m}^e \bmod N$

$\mathrm{Dec}_{sk}(c)$:
1. $\hat{m} \leftarrow c^d \bmod N$
2. Return $m = \hat{m}_{[n-\ell..n[}$

Correction

- As for the original RSA

Security of padded RSA

The security depends on $n - \ell$ — number of padding bits

Small values of $n - \ell$

- $2^{n-\ell}$ possible paddings
- Sufficient to break $2^{n-\ell}$ original RSA instances

$\rightarrow$ Not secure!

Very large value of $n - \ell$: $\ell = 1$

- If computing e-th root in $\mathbb{Z}/N\mathbb{Z}$ is hard, IND-CPA secure encryption scheme
- Very inefficient secure encryption scheme, one bit at a time
- Slightly better if used as a KEM — *still useless!*

Medium values of $n - \ell$

- Open problem!

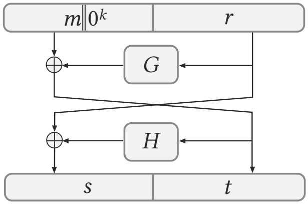
Padded RSA in practice

RSA PKCS1

- Standardized by RSA laboratories
- Padding: $m \rightarrow \texttt{0x00} \| \texttt{0x02} \| r \| \texttt{0x00} \| m$ where r is random
- Attack using failure of the unpadding procedure *Bleichenbacher (1998)*
  - Used against SSL 3.0
  - Workaround: in case of failure, return a random value
  - Prevents IND-CCA security

RSA Optimal Asymmetric Encryption Padding (OAEP) *Bellare, Rogaway (1994)*

- Padding: $m \rightarrow s \| t$ where
  - G, H: hash functions
  - r: random bits
- Standardized as PKCS1 v2
- IND-CCA secure under two assumptions
  - RSA trapdoor is *one-way*
  - G and H are random oracles



Source : J. Katz, Y. Lindell. Introduction to modern cryptography. 3rd ed, CRC Press, 2021. (modif.)

Contents

Original (broken…) version

Signatures are not "encryption with the private key"

Construction

$\mathsf{Gen}_n()$:
1. $p, q \twoheadleftarrow$ two random primes s.t. $p \times q$ has bit-length n
2. $N \leftarrow p \times q$, $\varphi(N) \leftarrow (p-1) \times (q-1)$
3. $e \twoheadleftarrow$ random integer invertible modulo $\varphi(N)$, $d \leftarrow e^{-1} \bmod \varphi(N)$
4. return $pk = (N, e)$, $sk = (N, d)$

$\mathsf{Sign}_{sk}(m)$:
1. return $m^d \bmod N$ $\qquad m \in \mathbb{Z}/N\mathbb{Z}$

$\mathsf{Vrfy}_{pk}(m, \sigma)$:
1. test whether $m = \sigma^e \bmod N$

Correction

- As for the original RSA encryption scheme

Attacks

existential forgeries

1. The adversary chooses σ and computes $m = \sigma^e \bmod N$
2. The adversary sees (m_1, σ_1) and (m_2, σ_2) and computes $m = m_1 \cdot m_2$ and $\sigma = \sigma_1 \cdot \sigma_2$

RSA FDH (*Full Domain Hash*)

Construction

$\text{Gen}_n()$:
1. Compute $pk = (N, e)$, $sk = (N, d)$ as previously
2. Choose a hash function $H : \{0,1\}^* \to \mathbb{Z}/N\mathbb{Z}$

$\text{Sign}_{sk}(m)$:
1. return $H(m)^d \bmod N$ — $m \in \{0,1\}^*$

$\text{Vrfy}_{pk}(m, \sigma)$:
1. test whether $H(m) = \sigma^e \bmod N$

What should H satisfy to avoid attacks?

1. $\sigma \to h = \sigma^e \to H(m) = h$ — *first preimage resistance*
2. $m_1, m_2 \to H(m) = H(m_1) \cdot H(m_2) \bmod N$ — *"non-multiplicative"*
3. If $H(m_1) = H(m_2)$, $\sigma_1 = \sigma_2$ — *collision resistance*
4. The image of H should be the full $\mathbb{Z}/N\mathbb{Z}$ — *full domain*

Bad and good news

- We *do not know* how to build a satisfying H — *no security proof*
- Security proof if RSA trapdoor is *one-way* and H is a random oracle

Proof sketch of RSA FDH

(Informal) Theorem

If e-th roots in $\mathbb{Z}/N\mathbb{Z}$ are hard to compute and H is random, RSA FDH is secure

Proof sketch : Prove that if $\mathcal{A}$ attacks RSA-FDH, then $\mathcal{A}$ is able to compute eth roots in $\mathbb{Z}/N\mathbb{Z}$

- What is an adversary $\mathcal{A}$?
  - → Requests m_i, gets $\sigma_i = H(m_i)^d \bmod N$
  - → Hyp.: for each m_i, $\mathcal{A}$ also requests $h_i = H(m_i)$
- Build $\mathcal{A}'$ that given c, computes m s.t. $c = m^e \bmod N$, by playing the EUF-CMA game
  - → Let t be the total number of requests to H made by $\mathcal{A}$.

  1. $\mathcal{A}'$ guesses a value $j \in \{1, \ldots, t\}$
  2. $\mathcal{A}'$ simulates $\mathcal{A}$:
     - for a request $H(m_i)$ $(i \neq j)$, $\mathcal{A}'$ samples σ_i and returns $h_i = \sigma_i^e \bmod N$
     - for $H(m_j)$, $\mathcal{A}'$ returns c. [if $\mathcal{A}$ returns (m_j, σ) then $\sigma = H(m_j)^d = c^d$]

Proof sketch of RSA FDH

(Informal) Theorem

If e-th roots in $\mathbb{Z}/N\mathbb{Z}$ are hard to compute and H is random, RSA FDH is secure

Proof sketch

<u>Analysis</u>:

1. If $\mathcal{A}'$ makes a correct guess j, then $\mathcal{A}$ returns a pair (m_j, σ) where $H(m_j)^d = \sigma$. Since $\mathcal{A}'$ decided to set $H(m_j) = c$, $\mathcal{A}'$ knows σ s.t. $c^d = \sigma$ $(\Leftrightarrow c = \sigma^e)$

   ↳ $\mathcal{A}'$ has computed the eth root of c.

2. If $\mathcal{A}'$ makes an incorrect guess, it fails.

$$\Pr[\text{Success}(\mathcal{A}')] = \frac{1}{t} \Pr[\text{Success}(\mathcal{A})].$$

Conclusion

RSA is a *one-way* trapdoor function

- One direction is easy to compute: $(m, e) \to m^e \bmod N$
- The other direction is (hopefully!) hard to compute: $(c, e) \to \sqrt[e]{c} \bmod N$
- But there is a trapdoor: given $d = e^{-1} \bmod \varphi(N)$, easy to compute $m = c^d \bmod N$

Use of RSA trapdoor function

- No direct use!
- Public-key encryption scheme $\to$ RSA OAEP
- Digital signatures $\to$ RSA FDH

Security

- No formal proof that RSA is *one-way* *assumption*
- Related but not equivalent to the difficulty of integer factorization
- Typical key sizes: N with $\geq$ 2048 bits
- Many other pitfalls: implementation, randomness quality, dependent keys, …