Table of contents

Asymmetric Cryptography (Public Key Cryptography)

Mathematical Foundations

Fundamental Theorem of Arithmetic and Euler's Totient Function

Asymmetric cryptography relies on solid mathematical foundations from number theory. Two concepts are essential:

Fundamental Theorem of Arithmetic: Every positive integer greater than 1 can be written uniquely (up to order) as a product of prime powers:

$$n = p_1^{e_1} \cdot p_2^{e_2} \cdot p_3^{e_3} \cdots p_m^{e_m}$$

Euler's Totient Function $\phi(n)$: Number of positive integers smaller than n that are coprime with n.

To compute $\phi(n)$:

$$\phi(n) = \prod_{i=1}^{m} p_i^{e_i} \cdot \left(1 - \frac{1}{p_i}\right)$$

Important special case: If $n = p \cdot q$ with p and q prime, then:

$$\phi(n) = (p-1)(q-1)$$

Original Text

Mathematical Foundations

Fundamental Theorem of Arithmetic: Every positive integer n can be written uniquely (up to order) as a product of powers of distinct prime numbers p_i:

$$n = p_1^{e_1} \cdot p_2^{e_2} \cdot p_3^{e_3} \cdots p_m^{e_m}$$

Euler's Totient Function: Let $n \in \mathbb{Z}^+$, the **Euler's totient function** $\phi(n)$ is equal to the number of positive integers smaller than n that are **relatively prime** to n.
Calculation of Euler's totient function: According to the fundamental theorem of arithmetic, every integer $n > 1$ can be written as:

$$n = \prod_{i=1}^{m} p_i^{e_i}$$

then $\phi(n)$ is calculated as:

$$\phi(n) = \prod_{i=1}^{m} \left(p_i^{e_i} - p_i^{e_i-1}\right)$$

In particular, if $n = p \cdot q$ with p and q prime, then:

$$\phi(n) = (p-1)(q-1)$$

Quick Revision

- **Unique decomposition**: every integer = product of prime numbers
- $\phi(n)$: counts integers $< n$ coprime with n
- **Key for RSA**: if $n = pq$ (primes) then $\phi(n) = (p-1)(q-1)$

Euler's Theorem and Fermat's Little Theorem

These theorems are at the heart of RSA and other asymmetric algorithms.

Euler's Theorem: If $n \in \mathbb{Z}^+$ and $a \in \mathbb{Z}$ with $\gcd(a, n) = 1$, then:

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

Fermat's Little Theorem (special case if $n = p$ prime): If $a \in \mathbb{Z}$ and p prime does not divide a:

$$a^{p-1} \equiv 1 \pmod{p}$$

Important applications:

1. **Exponent reduction**: If n is a product of distinct primes and $r \equiv s \pmod{\phi(n)}$, then:

$$a^r \equiv a^s \pmod{n}$$

2. **Calculation of inverses**: $a^{\phi(n)-1}$ is the inverse of a modulo n. In particular, if p is prime, a^{p-2} is the inverse of a modulo p.

Original Text

Mathematical Foundations (II)

Euler's Theorem: Let $n \in \mathbb{Z}^+$ and $a \in \mathbb{Z}$ with $\gcd(a, n) = 1$, then we have:

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

Fermat's Little Theorem (special case of Euler's theorem if n is prime): Let $a \in \mathbb{Z}$ and p a prime number such that p does not divide a, then we have:

$$a^{p-1} \equiv 1 \pmod{p}$$

Note that since p is prime, we have $\phi(p) = p - 1$.
Exponent reduction $\bmod \phi(n)$: If n is the product of distinct primes and $r, s \in \mathbb{Z}$ such that $r \equiv s \pmod{\phi(n)}$ then $\forall a \in \mathbb{Z}$:

$$a^r \equiv a^s \pmod{n}$$

Application of Euler's Theorem to inverse calculation: From Euler's theorem, we have that:

$$a \cdot a^{\phi(n)-1} \equiv 1 \pmod{n}$$

which means that $a^{\phi(n)-1}$ is the **inverse of a modulo n**. In particular, a^{p-2} is the inverse of a modulo n if p is prime.

Quick Revision

- **Euler's Theorem**: $a^{\phi(n)} \equiv 1 \pmod{n}$
- **Fermat**: special case if p prime: $a^{p-1} \equiv 1 \pmod{p}$
- **Modular inverse**: $a^{-1} \equiv a^{\phi(n)-1} \pmod{n}$
- **Base of RSA**: enables encryption/decryption with exponents

---

Multiplicative Groups and Generators

Multiplicative group $\mathbb{Z}_n^*$: Set of elements of $\mathbb{Z}_n$ coprime with n:

$$\mathbb{Z}_n^* = \{a \in \mathbb{Z}_n \mid \gcd(a, n) = 1\}$$

If n is prime: $\mathbb{Z}_n^* = \{1, 2, \ldots, n-1\}$

Order of an element: Smallest positive integer t such that $a^t \equiv 1 \pmod{n}$

Generator: An element α is a generator of $\mathbb{Z}_n^*$ if its order is $\phi(n)$. Then $\mathbb{Z}_n^*$ is said to be **cyclic**.

Properties of generators:

1. $\mathbb{Z}_n^*$ has a generator iff $n = 2, 4, p^k$ or $2p^k$ (with p prime, $p \neq 2$ and $k \geq 1$)
2. If p is prime, $\mathbb{Z}_p^*$ always has a generator
3. If α is a generator, all elements can be written as: $\mathbb{Z}_n^* = \{\alpha^i \bmod n \mid 0 \leq i < \phi(n)\}$
4. The number of generators is $\phi(\phi(n))$

Generator test

- α is a generator of $\mathbb{Z}_n^*$ iff for every prime p dividing $\phi(n)$, $\alpha^{\phi(n)/p} \not\equiv 1 \pmod{n}$
- if $n = 2p + 1$ is a "safe prime" with p prime: α is a generator iff $\alpha^2 \not\equiv 1 \pmod{n}$ and $\alpha^p \not\equiv 1 \pmod{n}$

> **i** Original Text
>
> **Mathematical Foundations (III)**
>
> **Definition**: The **multiplicative group of** $\mathbb{Z}_n$, denoted $\mathbb{Z}_n^*$ is:
>
> $$\mathbb{Z}_n^* = \{a \in \mathbb{Z}_n \mid \gcd(a, n) = 1\}$$
>
> In particular, if n is prime: $\mathbb{Z}_n^* = \{a \mid 1 \leq a \leq n - 1\}$
> The **number of elements or order** of the multiplicative group $\mathbb{Z}_n^*$ is $\phi(n)$ (by definition of ϕ).
> **Definition**: Let $a \in \mathbb{Z}_n$, the **order of** a is the smallest positive integer t for which:
>
> $$a^t \equiv 1 \pmod{n}$$
>
> **Definition**: Let $\alpha \in \mathbb{Z}_n^*$, if the order of α is $\phi(n)$, then α is a **generator of** $\mathbb{Z}_n^*$. When a group $\mathbb{Z}_n^*$ has a generator, it is said to be **cyclic**.
> **Properties of generators**:
>
> - $\mathbb{Z}_n^*$ has a generator iff $n = 2, 4, p^k$ or $2p^k$, with p prime, $p \neq 2$ and $k \geq 1$. In particular, if p is prime, $\mathbb{Z}_p^*$ has a generator.
> - If α is a generator of $\mathbb{Z}_n^*$, then all elements of $\mathbb{Z}_n^*$ can be written as:
>
> $$\mathbb{Z}_n^* = \{\alpha^i \bmod n \mid 0 \leq i \leq \phi(n) - 1\}$$
>
> - The number of generators of $\mathbb{Z}_n^*$ is $\phi(\phi(n))$.

- α is a generator of $\mathbb{Z}_n^*$ iff for every prime p dividing $\phi(n)$, we have:

$$\alpha^{\phi(n)/p} \not\equiv 1 \pmod{n}$$

In particular if n is a prime of the form $n = 2p + 1$ with p prime (such n is called a **safe prime**), α is a generator of $\mathbb{Z}_n^*$ iff $\alpha^2 \not\equiv 1 \pmod{n}$ and $\alpha^p \not\equiv 1 \pmod{n}$.

Quick Revision

- $\mathbb{Z}_n^*$: elements coprime with n, cardinality $= \phi(n)$
- **Generator**: element of order $\phi(n)$ (generates the entire group)
- **Crucial for DH and ElGamal**: security based on discrete logarithm in cyclic group
- **Safe prime**: $n = 2p + 1$ with p and n prime

Fast Exponentiation

Efficient computation of $a^k \bmod n$ in polynomial time, essential for all asymmetric algorithms.

Principle: Use the binary representation of the exponent k.

Example: Computation of $2^{644} \bmod 645$

1. Binary representation: $(644)_{10} = (1010000100)_2$
2. Compute successive powers of 2 modulo 645:
   - $2^1 \bmod 645$
   - $2^2 \bmod 645$
   - $2^4 \bmod 645$
   - $2^8 \bmod 645$
   - ...
   - $2^{512} \bmod 645$
3. Combine according to bits set to 1: $2^{644} = 2^{512} \cdot 2^{128} \cdot 2^4$

Complexity: $O(\log^3 n)$ - very efficient!

Application: Computation of the inverse using Euler's theorem in polynomial time.

Alternative: **Extended Euclidean algorithm** to find x such that $ax \equiv 1 \pmod{n}$ by solving $ax - kn = 1 = \gcd(a, n)$. Complexity also $O(\log^3 n)$.

Original Text

Fast Exponentiation

Fast exponentiation: Using the binary representation of a number, we can compute powers very efficiently.
Example: computation of $2^{644} \bmod 645$
$(644)_{10} = (1010000100)_2$
Now, we compute the exponents corresponding to the powers of 2, namely:

$$2^1 \bmod 645, \quad 2^2 \bmod 645, \quad 2^4 \bmod 645, \quad \dots, \quad 2^{512} \bmod 645$$

From the binary representation, we compute:

$$2^{644} = 2^{512+128+4} = 2^{512} \cdot 2^{128} \cdot 2^4 = 160 \cdot 153 \cdot 6 \bmod 645$$

The complexity of this algorithm fast exponentiation is $O(\log^3 n)$.
By relying on **Euler's theorem**, the computation of the **inverse of a number** in such a group is therefore performed in polynomial time.
The extended Euclidean algorithm can also be used to find an x such that:

$$ax \equiv 1 \pmod n$$

since this congruence can be written as: $ax - 1 = kn$ and therefore:

$$ax - kn = 1 = \gcd(a, n)$$

The complexity of this algorithm is also $O(\log^3 n)$.

Quick Revision

- **Idea**: binary representation of the exponent
- **Complexity**: $O(\log^3 n)$ - polynomial!
- **Essential**: makes RSA, ElGamal, DH practical
- **Alternative**: extended Euclidean algorithm for inverses

Chinese Remainder Theorem (CRT)

The CRT allows solving systems of simultaneous congruences, with important applications in cryptography.

Theorem: Let $n_1, n_2, \dots, n_t \in \mathbb{Z}^+$ pairwise coprime ($\gcd(n_i, n_j) = 1$ if $i \neq j$) and $a_1, a_2, \dots, a_t \in \mathbb{Z}$. Then the system:

$$\begin{cases} x \equiv a_1 \pmod{n_1} \\ x \equiv a_2 \pmod{n_2} \\ \vdots \\ x \equiv a_t \pmod{n_t} \end{cases}$$

has a unique solution $x \bmod N$ with $N := n_1 \cdot n_2 \cdots n_t$.

Gauss's algorithm (1801) to compute x:

$$x = \sum_{i=1}^{t} a_i N_i M_i \bmod N$$

with:

- $N_i = N/n_i$
- $M_i = N_i^{-1} \bmod n_i$ (modular inverse)

Complexity: $O(\log^3 n)$ - polynomial!

Cryptographic applications:

1. Acceleration of RSA computations (use p and q separately)
2. Secret sharing (secret sharing schemes)
3. Certain attacks on RSA (if small exponent and multiple messages)

> **Original Text**
>
> **Chinese Remainder Theorem**
>
> The **Chinese Remainder Theorem** (3rd century!) allows solving linear systems of simultaneous congruences. It solves problems raised in ancient Chinese puzzles. It was, for example, about finding a number that produces a remainder of 1 when divided by 3, of 2 when divided by 5 and of 3 when divided by 7… It was also used to calculate the exact moment of alignment of several celestial bodies having different orbits (and therefore periods).
> **Chinese Remainder Theorem**: Let $n_1, n_2, \dots, n_t \in \mathbb{Z}^+$ be pairwise coprime (i.e., $\gcd(n_i, n_j) = 1,\ \forall i \neq j$) and $a_1, a_2, \dots, a_t \in \mathbb{Z}$. Then, the system of congruences:

$$\begin{cases} x \equiv a_1 \pmod{n_1} \\ x \equiv a_2 \pmod{n_2} \\ \vdots \\ x \equiv a_t \pmod{n_t} \end{cases}$$

has a unique solution $x \bmod N := n_1 n_2 \cdots n_t$
Gauss's algorithm (1801) for the computation of x:

$$x = \sum_{i=1}^{t} a_i N_i M_i \bmod N$$

with $N_i = N/n_i$ and $M_i = N_i^{-1} \bmod n_i$.
The **complexity** of this algorithm is $O(\log^3 n)$.
It is therefore possible in **polynomial time** to go from congruences mod n_i to congruences mod N!

Quick Revision

- **Solves**: systems of congruences with pairwise coprime moduli
- **Unique solution**: modulo product of moduli
- **Complexity**: $O(\log^3 n)$ (polynomial)
- **Crypto usage**: RSA optimization, attacks if small exponent

---

Basic Problems and Complexity

Classification of Hard Problems

The security of asymmetric cryptography relies on mathematical problems reputed to be hard:

Generic problems:

1. **Factorization (FACTP)**: Given n, find its factorization into prime numbers
   - Base of **RSA** and **Rabin**
2. **Discrete Logarithms (DLP)**: Given prime p, a generator $\alpha \in \mathbb{Z}_p^*$ and $\beta \in \mathbb{Z}_p^*$, find x such that:
   $$\alpha^x \equiv \beta \pmod{p}$$
   - Base of **ElGamal** and **Diffie-Hellman**

3. **Square Root modulo composite (SQROOTP)**: Given composite n and a quadratic residue a, find $\sqrt{a} \bmod n$
    - Base of **Rabin**

Specific problems:

1. **RSA Problem (RSAP)**: Given $n = pq$, e with $\gcd(e, \phi(n)) = 1$ and c, find m such that $m^e \equiv c \pmod{n}$
2. **Diffie-Hellman Problem (DHP)**: Given prime p, generator α, $\alpha^a \bmod p$ and $\alpha^b \bmod p$, find $\alpha^{ab} \bmod p$

Proven equivalences:

- **DHP DLP** (equivalent under certain conditions)
- **RSAP FACTP** (proven equivalent for the generic case)
- **SQROOTP FACTP**

> Original Text
>
> **Basic Problems**
>
> **Main generic problems**:
>
> - **Factorization (FACTP)**: Given a positive integer n, find its factorization into prime numbers.
> - **Discrete Logarithms (DLP)**: Given a prime number p, a generator $\alpha \in \mathbb{Z}_p^*$ and an element $\beta \in \mathbb{Z}_p^*$, find the integer x, $0 \leq x \leq p-2$, such that: $\alpha^x \equiv \beta \pmod{p}$.
> - **Square Root in $\mathbb{Z}_n$ if n is composite (SQROOTP)**: Given a composite integer n and a quadratic residue a, find the square root of $a \bmod n$.
>
> **Specific problems** (proper to an encryption system):
>
> - **RSA (RSAP)**: Given a positive integer $n = pq$, a positive integer e with $\gcd(e, (p-1)(q-1)) = 1$ and an integer c, find an integer m with $m^e \equiv c \pmod{n}$.
> - **Diffie-Hellman (DHP)**: Given a prime number p, a generator $\alpha \in \mathbb{Z}_p^*$ and the elements $\alpha^a \bmod p$ and $\alpha^b \bmod p$, find $\alpha^{ab} \bmod p$.
>
> **Proven results**:
>
> - **DHP DLP** (Equivalent under certain conditions)
> - **RSAP FACTP** (Proven equivalent for the generic problem)
> - **SQROOTP FACTP**

Quick Revision

- **FACTP**: factor $n \to$ base of RSA/Rabin
- **DLP**: find discrete logarithm $\to$ base ElGamal/DH
- **SQROOTP**: square root mod composite $\to$ Rabin
- **Equivalences**: breaking = solving the base problem

Factorization Techniques

The security of RSA depends on the difficulty of factoring large numbers.

Exponential time methods: $O(\exp(c \cdot \ln(n)))$

- Trial Division (successive division)
- Sieve of Eratosthenes (2nd century BC)
- Fermat's Method (~1650)
- Pollard's ρ Method (1975)
- Pollard's $p-1$ Method (1974)

Sub-exponential time methods: $O(\exp(c \cdot (\ln(n))^{1/3}))$

- Continued Fractions (1975)
- **Quadratic Sieve (1981)** - very effective in practice
- **Number Field Sieve - NFS (1990)** - currently the fastest
- General Number Field Sieve - GNFS (2006)

Polynomial time methods:

- **Shor's Algorithm** (1994): $O(\log^c n)$ on **quantum computer**

Current records (2020):

- Largest number factored: **RSA-829** (250 digits, 829 bits)
- Computation time: 2700 core-years (Intel Xeon Gold 6130 CPUs)
- Method: General Number Field Sieve

Implications:

- RSA keys < 1024 bits: **vulnerable**
- RSA keys 1024 bits: **limits** (states with significant resources)
- Recommendation: **2048 bits minimum** (3072-4096 for long term)

Original Text

Classical Factoring Techniques and New Developments

Exponential time: $O(\exp(c \cdot \ln(n)))$

- Trial Division
- Eratosthenes' Sieve (II B.C.)
- Fermat's Difference of Squares Method (~1650)
- Square Form Factorization (1971)
- Pollard's p-1 method (1974)
- Pollard's Rho Method (1975)

Sub-exponential time: $O(\exp(c \cdot (\ln(n))^{1/3}))$

- Continued Fractions (1975)
- **Quadratic Sieve (1981)**
- **Number Field Sieve - NFS (1990)**
- **General Number Field Sieve - GNFS (2006)**

Polynomial time:

- **Shor's Algorithm in a Quantum Computer (1994)**: $O(\log^c n)$

Recent developments:

- Bernstein's specific NFS computer to factor a 1536-bit number would take the same time as a 512-bit computation on a conventional machine
- **Largest factorization to date (2020)**: RSA-829 (250-digit number) using NFS
- Total computation time: **2700 core-years** (Intel Xeon Gold 6130 CPUs at 2.1GHz)

Factorization on quantum computer:

- Significant problems (errors, dispersion, etc.)
- 2001: 7-qubit computer (IBM Almaden)
- Feasibility of a computer with millions of qubits... ?

Quick Revision

- **Sub-exponential**: NFS currently the fastest
- **Record 2020**: RSA-829 (829 bits) in 2700 core-years
- **Recommendation**: keys 2048 bits for RSA
- **Future threat**: quantum computers (Shor)

The RSA Algorithm

RSA Operation (Encryption/Decryption)

RSA (Rivest-Shamir-Adleman, 1978) is the most used asymmetric algorithm.

Key generation:

1. Choose two **large** prime numbers p and q (1024 bits each)
2. Compute $n := p \cdot q$ and $\phi(n) = (p-1)(q-1)$
3. Choose encryption exponent e with:
   - $1 < e < \phi(n)$
   - $\gcd(e, \phi(n)) = 1$
4. Compute decryption exponent d such that:

$$e \cdot d \equiv 1 \pmod{\phi(n)}$$

   (using extended Euclidean algorithm or fast exponentiation)

Resulting keys:

- **Public** key: (n, e)
- **Private** key: d (keep p and q secret too!)

Encryption (by Bob, to Alice):

1. Obtain authentic public key (n, e) of Alice
2. Transform plaintext into integers $m_i \in [0, n-1]$
3. Compute ciphertexts: $c_i := m_i^e \bmod n$
4. Send the c_i to Alice

Decryption (by Alice):

- Use private key d to compute:
$$m_i = c_i^d \bmod n$$

Proof of operation:

$$c^d \equiv (m^e)^d \equiv m^{ed} \pmod{n}$$

Since $ed \equiv 1 \pmod{\phi(n)}$, there exists k such that $ed = 1 + k\phi(n)$, therefore:

$$c^d \equiv m^{1+k\phi(n)} \equiv m \cdot (m^{\phi(n)})^k \equiv m \cdot 1^k \equiv m \pmod{n}$$

(by Euler's theorem)

Original Text

RSA Encryption/Decryption Procedure and Proof

Key generation:

- Each entity (A) creates a key pair (public and private) as follows:
  - A chooses the size of the modulus n (e.g., size(n) = 1024 or size(n) = 2048).
  - A generates two prime numbers p and q of large size ($n/2$).
  - A computes $n := pq$ and $\phi(n) = (p-1)(q-1)$.
  - A generates the encryption exponent e, with $1 < e < \phi(n)$ such that $\gcd(e, \phi(n)) = 1$.
  - A computes the decryption exponent d, such that: $ed \equiv 1 \pmod{\phi(n)}$ using the extended Euclidean algorithm or fast exponentiation.
- The pair (n, e) is A's **public** key; d is A's **private** key.

Encryption:

- Entity B obtains (n, e), the **authentic** public key of A.
- B transforms its plaintext into a series of integers m_i, such that $m_i \in [0, n-1]\ \forall i$.
- B computes the ciphertext $c_i := m_i^e \bmod n$, $\forall i$ using fast exponentiation.
- B sends to A all the ciphertexts c_i.

Decryption:

- A uses its private key to compute the plaintexts $m_i = c_i^d \bmod n$.

Proof: Let m be the plaintext and c the ciphertext with $c := m^e \bmod n$, we need to prove: $m \overset{!}{=} c^d \bmod n$
Substituting c by its value we obtain:

$$c^d \bmod n = m^{ed} \bmod n \quad (*)$$

but, we know that:

$$ed \equiv 1 \pmod{\phi(n)}$$

and therefore by definition of congruences, there exists an integer k with:

$$ed - 1 = k\phi(n)$$

substituting in (*):

$$c^d \equiv m^{k\phi(n)+1} \equiv m^{k\phi(n)} \cdot m \pmod{n}$$

If $\gcd(m, n) = 1$, we have by **Euler's theorem**:

$$m^{\phi(n)} \equiv 1 \pmod{n}$$

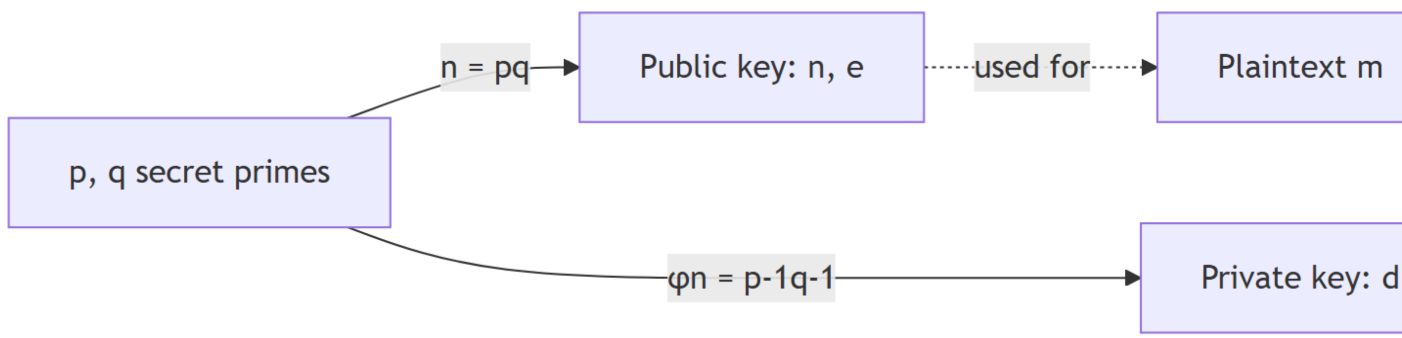
therefore:

$$c^d \equiv (m^{\phi(n)})^k \cdot m \equiv m \pmod{n}$$

Q.E.D. !
If $\gcd(m, n) \neq 1$, m is necessarily a multiple of p or q (very unlikely case…), we can show by doing the calculations mod p and mod q that the congruence remains true.

Quick Revision

- **Public key**: (n, e) with $n = pq$
- **Private key**: d such that $ed \equiv 1 \pmod{\phi(n)}$
- **Encryption**: $c = m^e \bmod n$
- **Decryption**: $m = c^d \bmod n$
- **Security**: based on difficulty of factoring n



RSA Security

Equivalence RSA problem Factorization:

- Finding d factoring n (proven equivalent)
- Decrypting without d is **not proven** as hard as factoring, but…
- No method faster than factoring is known

Factorization complexity:

- Fastest methods: $O(\exp(c \cdot (\ln(n))^{1/3}))$ (sub-exponential)
- Computationally impossible for $n \geq 1024$ bits
- **Current recommendation**: 2048 bits minimum (3072-4096 for long-term security)

Choice of exponents:

- **Encryption exponent** e:
  - Often **small** for speed: $e = 3, 17, 65537$ (common)
  - Caution: if e too small AND $m < n^{1/e}$, attack possible (e-th root in $\mathbb{Z}$)
  - Solution: **randomization** (padding) of the message
- **Decryption exponent** d:
  - Must be **large**: at least half the size of n
  - If d small: vulnerable to Wiener's attack

Performance consequence:

- **Fast encryption** (e small)
- **Slow decryption** (d large)

> Original Text
>
> **RSA: Security**
>
> The **RSAP** problem of finding m from c is not proven to be as hard as factorization but…:
>
> - We can prove that if we find d we can easily compute p and q. This is equivalent to saying that **factoring n and finding d require equivalent computational effort**.
> - We know that the fastest methods for factoring have a **sub-exponential complexity** $O(\exp(c \cdot (\ln(n))^{1/3}))$. The problem therefore remains **computationally impossible** for modulus ≥ 1048 bits (2048 bits is a frequent choice for long-term security…).
> - To improve encryption speed, we tend to choose **relatively small exponents** e (typically: $e := 3$, $e := 17$ and $e := 19$). However, it has been proven that computing an i-th root (with small i) modulo a composite n can be significantly easier than factoring n. On the other hand, in 2008 it was proven that the generic RSA problem is equivalent to factorization.
> - The **decryption exponent d must imperatively be large** (at least half the size

of n) to guarantee the system's security.

- Consequently, **encryption is normally significantly faster than decryption** since the exponents used are much smaller!

Quick Revision

- **Security**: based on difficulty of FACTP (factorization)
- **Recommended size**: $n \geq 2048$ bits
- **Small** e: fast encryption (3, 17, 65537)
- **Large** d: at least size(n)/2
- **Separate keys**: encryption signature

Attacks on RSA

Attack on small exponent with same message

If the same message m is sent to 3 recipients with $e = 3$:

- $c_1 \equiv m^3 \pmod{n_1}$
- $c_2 \equiv m^3 \pmod{n_2}$
- $c_3 \equiv m^3 \pmod{n_3}$

The **Chinese Remainder Theorem** gives a unique solution x mod $n_1 n_2 n_3$ such that:

$$x \equiv c_1 \pmod{n_1}, \quad x \equiv c_2 \pmod{n_2}, \quad x \equiv c_3 \pmod{n_3}$$

If $m^3 < n_1 n_2 n_3$ (often true), then $x = m^3$ in $\mathbb{Z}$ and we can compute m by simply taking the integer cube root!

Protection: always randomize the message before encryption (OAEP padding)

Attack if message small

If $m < n^{1/e}$, then $m^e < n$, so $c = m^e$ (in $\mathbb{Z}$, not modulo). We can directly compute the e-th root!

Protection: padding mandatory

Multiplicative property

$$E(m_1) \cdot E(m_2) \equiv (m_1 \cdot m_2)^e \equiv E(m_1 \cdot m_2) \pmod{n}$$

Allows chosen-ciphertext attacks and blind signatures.

General attack

The most effective method remains **factoring** n (if parameters well chosen and implementation correct).

> **Original Text**
>
> **RSA: Attacks**
>
> When we want to encrypt the **same message for a group of correspondents**, it is advisable to introduce variations (**randomization**) before encryption to avoid the following attack:
> Assume we compute ciphertexts c_1, c_2, c_3 from the same plaintext m and the same exponent $e := 3$ addressed to three entities with modulus: n_1, n_2, n_3.
> The **Chinese Remainder Theorem** tells us that there exists a solution $x \bmod n_1 n_2 n_3$, such that:
>
> $$x \equiv c_1 \pmod{n_1}, \quad x \equiv c_2 \pmod{n_2}, \quad x \equiv c_3 \pmod{n_3}$$
>
> But if m does not change for the three encryptions, we have that $x = m^3 \bmod n_1 n_2 n_3$ and, moreover: $m^3 < n_1 n_2 n_3$. We can, therefore, find m by computing the **integer cube root** of m^3, knowing that for this calculation there exist efficient algorithms!
> More generally, if $m < n^{1/e}$, we can apply fast algorithms (in $\mathbb{Z}$) to compute the e-th roots of m^e. It is therefore advisable to perform **"randomization" of m before encrypting**!
> **The multiplicative property of RSA**: $(m_1 m_2)^e \equiv m_1^e \cdot m_2^e \equiv c_1 \cdot c_2 \pmod{n}$ gives rise to **dangerous vulnerabilities** (see blind signatures).
> Assuming parameters are correctly chosen and the implementation has no flaws, **the most effective method to "break" the generic RSA algorithm remains factoring n.**

> **Quick Revision**
>
> - **Same message, small e**: CRT allows extracting m!
> - **Message too small**: $m < n^{1/e} \to$ direct root
> - **Multiplicative property**: $E(m_1) \cdot E(m_2) = E(m_1 m_2)$
> - **Protection**: always padding/randomization (OAEP)

The ElGamal Algorithm

Asymmetric system (1985) based on the **discrete logarithm problem (DLP)**.

Keys:

- Choose prime p, generator $\alpha \in \mathbb{Z}_p^*$, secret a
- Compute $y = \alpha^a \bmod p$
- **Public**: (p, α, y) | **Private**: a

Encryption: For message m, choose unique random k

- $\gamma = \alpha^k \bmod p$
- $\delta = m \cdot y^k \bmod p$
- Send (γ, δ)

Decryption: $m = \delta \cdot \gamma^{-a} \bmod p$

Original Text

ElGamal Encryption/Decryption Procedure

Key generation
Each entity (A) creates a key pair (public and private) as follows:

- A generates a prime number p (len(p)   1024 bits) and a **generator** α of the multiplicative group $\mathbb{Z}_p^*$
- A generates a random number a, such that $1 \leq a \leq p{-}2$ and computes $y := \alpha^a \bmod p$
- The **public key** of A is (p, α, y), the **private key** of A is a

Encryption

- Entity B obtains $(p, \alpha, \alpha^a \bmod p)$, the authentic public key of A
- B transforms its plaintext into a series of integers m_i, such that $m_i \in [0, p-1]\ \forall i$
- For each message m_i:
  - B generates a **unique** random number k, such that $1 \leq k \leq p - 2$
  - B computes $\gamma := \alpha^k \bmod p$ and $\delta := m_i \cdot (\alpha^a)^k \bmod p$ and sends the ciphertext $c := (\gamma, \delta)$

Decryption

- A uses its private key a to compute $\gamma^{p-1-a} \bmod p$ (note that: $\gamma^{p-1-a} \equiv \gamma^{-a} \equiv \alpha^{-ak} \bmod p$)
- A retrieves the plaintext by computing: $\delta \cdot \gamma^{-ak} \bmod p$

Quick Revision

Base: DLP in $\mathbb{Z}_p^*$
Ciphertext: $(\alpha^k, m \cdot y^k)$
Security: k must be unique and large
Disadvantage: doubles message size

Essential Remarks

- **Proof**: $\delta \cdot \gamma^{-a} = m \cdot (\alpha^a)^k \cdot (\alpha^k)^{-a} = m \bmod p$
- **Security**: based on DLP (complexity sub-exponential close to factorization)
- **Exponents**: k and a must be large (otherwise vulnerable to baby-step giant-step)
- **Reuse prohibited**: if k repeated, $\delta_1/\delta_2 = m_1/m_2$ reveals the messages
- **Major disadvantage**: $\times 2$ expansion of ciphertext size
- **Generalization**: works on $GF(2^n)$ or elliptic curves

Original Text - Remarks

Proof that the scheme works: If $s \equiv k^{-1}(m_h - ar) \bmod (p-1)$, we have that: $m_h \equiv (ar + ks) \bmod (p-1)$ and $v_2 = \alpha^{H(m)} \bmod p$. If, as we wish to show $m_h = H(m)$, by reducing exponents mod $(p-1)$, we can rewrite v_2: $v_2 \equiv \alpha^{ar+ks} \bmod p$. On the other hand: $v_1 = y^r \alpha^{rs} \equiv \alpha^{ar}\alpha^{ks} \equiv \alpha^{ar+ks} \bmod p$.
The ElGamal procedure is based on the difficulty of computing **discrete logarithms modulo a prime number** (DLP problem) even though it has not been proven to be strictly equivalent to this problem.
The **most efficient algorithms** known have a sub-exponential complexity very close to that of factorization (we often use the same algorithms).
The **chosen exponents** (k, a) must be large because there exist efficient algorithms to compute discrete logarithms modulo a prime number when the exponent is small (baby-step giant-step algorithm).
A **disadvantage of ElGamal** is that it multiplies the ciphertext length by 2.
It is **essential** for the security of the procedure that the random number k is not repeated, otherwise: let (γ_1, δ_1) and (γ_2, δ_2) be the two generated ciphertexts, we have that $\delta_1/\delta_2 = m_1/m_2$ and consequently, it is trivial to recover one plaintext from the other.
The ElGamal procedure can be **generalized** to other groups like $GF(2^n)$ or elliptic curves.

Quick Revision - Remarks

Equivalence: based on DLP (not proven equivalent)
k **unique**: CRITICAL - otherwise m_1/m_2 revealed
Key size: large exponents necessary
Extensions: $GF(2^n)$, elliptic curves

Rabin Algorithm

Asymmetric system **equivalent to factorization** (provably secure).

Keys:

- Generate two primes p, q (1024 bits total), compute $n = pq$
  - **Public**: n
  - **Private**: (p, q)

Encryption: $c = m^2 \bmod n$

Decryption:

- Compute the 4 square roots of $c \bmod n$ (via roots mod p and mod q)
- Identify the correct message by redundancy

Original Text

Rabin Encryption/Decryption Procedure

Key generation
Each entity (A) creates a key pair (public and private) as follows:

- A generates two random prime numbers p and q of large size (len(pq) 1024)
- A computes $n := pq$
- The **public key** of A is n, the **private key** of A is (p, q)

Encryption

- Entity B obtains n, the authentic public key of A
- B transforms its plaintext into a series of integers m_i, such that $m_i \in [0, n-1]\ \forall i$
- B computes $c_i = m_i^2 \bmod n$ for each message m_i
- B sends all the ciphertexts c_i to A

Decryption

- A uses its private key (p, q) to retrieve the **4 solutions** of the equation: $c_i = x^2 \bmod n$ using **efficient algorithms** to compute square roots mod$p$ and mod$q$
- A determines either by an **additional indication** from B, or by **redundancy analysis** which of the 4 messages m_1, m_2, m_3, m_4 is the original plaintext

Quick Revision

Base: SQROOTP (square root mod composite)
Advantage: proven equivalent to factorization
Problem: 4 possible solutions, requires redundancy
Vulnerability: chosen-ciphertext attack reveals factors

Essential Remarks

- **Proven security**: SQROOTP   FACTP (only algorithm with proven equivalence)
- **Chosen-ciphertext attack**: if A decrypts $c = m^2 \bmod n$ chosen by adversary M
  - M receives a root m_x among 4 possible
  - If $m \neq m_x \bmod n$ (prob. 0.5), then $\gcd(m - m_x, n)$ gives a factor of n
- **Solution**: require sufficient redundancy to identify unique solution without ambiguity

Original Text - Remarks

The Rabin procedure is based on the **impossibility of finding square roots modulo a composite of unknown factorization** (SQROOTP problem).
The **main interest** of this algorithm lies in the fact that it has been **proven to be equivalent to factorization** (SQROOTP   FACTP). This algorithm therefore belongs to the **provably secure** category for any passive attack.
Active attacks can, in some cases, compromise the algorithm's security. More precisely, if we mount the following **chosen ciphertext** attack:

- The attacker M generates an m and sends to A the ciphertext $c = m^2 \bmod n$.
- A responds with a root m_x among the 4 possible m_1, m_2, m_3, m_4.
- If $m \neq m_x \bmod n$ (probability 0.5), M repeats with a new m.
- Otherwise, A computes $\gcd(m - m_x, n)$ and thus obtains one of the two factors of n.

This attack could be **avoided** if the procedure required **sufficient redundancy** in the plaintexts allowing A to identify without ambiguity which of the possible solutions is the original plaintext. In this case, A would always respond with m and discard the other solutions that do not have the predefined level of redundancy.

Quick Revision - Remarks

Unique: only algorithm proven equivalent to FACTP
Attack: chosen-ciphertext gives factors (prob. 0.5)
Countermeasure: mandatory redundancy in messages

Comparison RSA - ElGamal - Rabin

Criterion	RSA	ElGamal	Rabin
Problem	RSAP	DLP	SQROOTP
Security	Equiv. factorization (generic case)	Based on DLP	**Proven** factorization
Expansion	1:1	**1:2**	1:1
Decryption	Deterministic	Deterministic	**4 solutions**
Signature	Yes	Yes	Yes (with precautions)

Elliptic Curves (Basic Idea)

Fundamental Concept

An **elliptic curve** E is defined by: $y^2 = x^3 + ax + b$ (with discriminant $4a^3 + 27b^2 \neq 0$).

Key operation: Point addition

- Geometrically: draw a line between two points P and Q, find the 3rd intersection point, then take its symmetric
- Forms a **commutative group** with point at infinity $\mathcal{O}$ as identity
- **Scalar multiplication**: $kP = P + P + ... + P$ (k times)

Cryptographic advantage:

- The **ECDLP problem**: finding k such that $Q = kP$ is very difficult (exponential effort)
- **Shorter keys** for same security as in $\mathbb{Z}_p^*$

> Original Text - Definition
>
> An **elliptic curve** is a set of points E defined by the equation: $y^2 = x^3 + ax + b$, with x, y, a and b rational numbers, integers or integers modulo m ($m > 1$). The set E also contains a "point at infinity" denoted $\mathcal{O}$. The point $\mathcal{O}$ is not on the curve but it is the identity element of E.
> We will choose for our calculations elliptic curves that do not have multiple roots or, in other words, curves where the **discriminant** $4a^3 + 27b^2 \neq 0$.

> Quick Revision - Concept
>
> **Equation**: $y^2 = x^3 + ax + b$
> **Structure**: group with $\mathcal{O}$
> **Operation**: geometric addition
> **Hard problem**: ECDLP

Addition on Elliptic Curves

Let $P := (x, y) \in E$, we define $-P := (x, -y)$ (symmetric with respect to the x-axis). We have $P + (-P) = \mathcal{O}$.

For two points $P, Q \in E$ with $Q \neq -P$, we define $P + Q := R$ where $-R$ is the 3rd intersection point between the curve and the line passing through P and Q.

For **doubling**: $2P = R$ where $-R$ is the intersection point of the curve with the tangent to the curve at point P.

> Original Text - Addition
>
> Let $P := (x, y) \in E$, we define $-P$ as $-P := (x, -y)$. Graphically, $-P$ is the symmetric point of P with respect to the x-axis. Note that $P + (-P) = \mathcal{O}$.
> Let two points $P, Q \in E$, such that $Q \neq -P$, we define the addition $P + Q := R$ where $R \in E$ such that $-R$ is the 3rd intersection point between the curve and the line passing through P and Q.
> The set E with $\oplus$ defines a **commutative group** for addition.
> Let $P \in E$, the point $2P = R$, such that $-R$ is the intersection point of the curve with the line tangent to the curve at point P.

Quick Revision - Addition

Inverse: $-P = (x, -y)$
Addition: 3rd intersection point + symmetry
Doubling: tangent + symmetry
Property: commutative group

ECDLP and Cryptographic Advantages

When the elliptic curve is defined over the field $\mathbb{Z}_p$ with p a large prime ($y^2 \equiv x^3 + ax + b \bmod p$), the computation of $k \in \mathbb{Z}_p$ such that $Q = kP$ with (P, Q) known is **very difficult** (exponential effort). This problem is the **Elliptic Curve Discrete Logarithm Problem (ECDLP)**.

Main advantage: key sizes much smaller for equivalent security.

Original Text - ECDLP and Advantages

When the elliptic curve is defined over the field $\mathbb{Z}_p$ with p a large prime number ($y^2 \equiv x^3 + ax + b \bmod p$), the computation of $k \in \mathbb{Z}_p$ such that $Q = kP$ with (P, Q) known, is very difficult (requires exponential effort). This problem is known as: **Elliptic Curve Discrete Logarithm Problem (ECDLP)**.
The **main advantage** of public cryptography based on elliptic curves is that the size of the numbers used (and therefore, keys) is smaller.
This is due to the **increased complexity** of computations on E_p (elliptic curve defined over field $\mathbb{Z}_p$) compared to usual fields such as $\mathbb{Z}_p$ or $GF(2^m)$.
The **representation of a plaintext as points** of the curve remains a complex operation.
In October 2003, the **US National Security Agency (NSA)** purchased a patent from Certicom for the use of elliptic curve cryptography.
In September 2013 Claus Diem showed that under certain conditions the ECDLP problem could be solved in **sub-exponential time**.

Quick Revision - ECDLP

Problem: finding k in $Q = kP$ (exponential)
Gain: keys ~6-10× shorter
Limit: representing messages as points difficult
NSA: adopted in 2003

Key Size Comparison Table

AES (symmetric)	RSA/DH	Elliptic Curves	Ratio
56 bits	512 bits	112 bits	1:4.6
80 bits	1024 bits	160 bits	1:6.4
112 bits	2048 bits	224 bits	1:9.1
128 bits	3072 bits	256 bits	1:12
256 bits	15360 bits	512 bits	1:30

i Original Text - Table

This table shows the key size ratios compared to RSA for equivalent security.
(Table extracted from original document)

ElGamal on Elliptic Curves

Direct Adaptation

Replace operations in $\mathbb{Z}_p^*$ with operations on E_p

Keys:

- Choose curve E_p and point $P_0 \in E_p$ of large order
- Secret x, compute $P_a = xP_0$
- **Public**: (E_p, P_0, P_a) | **Private**: x

Encryption: For message $m_i \in E_p$

- Choose random k
- $\gamma = kP_0$, $\delta = kP_a + m_i$
- Send (γ, δ)

Decryption: $m_i = \delta - x\gamma$

i Original Text - ElGamal EC

Key generation
Each entity (A) creates a key pair (public and private) as follows:

- A chooses an elliptic curve E_p with p, a large prime number (len(p) bits) and a point $P_0 \in E_p$.
- A generates a random number x, such that $1 \le x \le p$ and computes $P_a = xP_0$

(multiplication by a scalar on E_p, for which efficient algorithms exist).

- The public key of A is (E_p, P_0, P_a), the private key of A is x.

Encryption

Entity B obtains (E_p, P_0, P_a), the authentic public key of A.

- B transforms its plaintext into a series of integers m_i, such that $m_i \in E_p$ for all i.
- For each message m_i:
  - B generates a **unique** random number k, such that $1 \leq k \leq p$.
  - B computes $\gamma := kP_0$ and $\delta := kP_a + m_i$ and sends the ciphertext $c := (\gamma, \delta)$.

Decryption

- A uses its private key x to compute: $x\gamma = xkP_0 = kP_a$.
- A retrieves the plaintext by computing: $\delta - kP_a = kP_a + m_i - kP_a = m_i$.

The security of the scheme relies on **ECDLP**!
It is also necessary to **authenticate** the exchanged public parts to avoid the previously described man-in-the-middle attacks.
The properties of the protocol are identical to the $\mathbb{Z}_p^*$ case.

Quick Revision - ElGamal EC

Principle: same as ElGamal on E_p
Operations: + and scalar multiplication on points
Security: ECDLP
Authentication: necessary against MitM
Advantage: short keys