Lecture Notes: RSA & Factorization Algorithms

Course: INFO-F-537 Cryptanalysis (Topic 4)
Document compiled by Luca Volonterio

Based on Standard Cryptanalysis Curriculum

January 8, 2026

Abstract

This document covers **Topic 4** for the oral exam. It focuses on the cryptanalysis of Public-Key systems, specifically RSA. The core security of RSA relies on the difficulty of the **Integer Factorization Problem**. We analyze algorithms to break this assumption, ranging from exponential time (Trial Division) to sub-exponential time (Quadratic Sieve and Number Field Sieve).

Contents

1 The RSA Primitive

1.1 Setup and Definition

RSA is based on the arithmetic of modular exponentiation.

- **Key Generation:**
  1. Choose two large distinct primes p and q.
  2. Compute $N = p \cdot q$ (the modulus).

3. Compute $\phi(N) = (p-1)(q-1)$.
4. Choose public exponent e such that $\gcd(e, \phi(N)) = 1$.
5. Compute private exponent $d \equiv e^{-1} \pmod{\phi(N)}$.

- **Encryption:** $C \equiv M^e \pmod{N}$.
- **Decryption:** $M \equiv C^d \pmod{N}$.

1.2 The Security Assumptions

> **Exam Tip: Important Distinction:** The *RSA Problem* (finding M from C, e, N) is not proven to be equivalent to the *Factorization Problem* (finding p, q from N). However, factorization is the most direct way to break RSA because knowing p, q allows computing $\phi(N)$ and thus d.

1.3 From Knowing d to Factoring N

Assume an attacker somehow obtains the private exponent d for a public key (N, e) with $N = pq$ and odd primes p, q. Define

$$k = ed - 1.$$

Then k is a multiple of $\phi(N) = (p-1)(q-1)$, hence in particular

$$k = 2^s \cdot t \quad \text{with } t \text{ odd}.$$

For any a coprime to N, one has

$$a^{ed} \equiv a \pmod{N} \quad \Rightarrow \quad a^k \equiv 1 \pmod{N}.$$

Consider the sequence

$$z_0 = a^t \bmod N, \quad z_{i+1} = z_i^2 \bmod N, \quad 0 \le i < s.$$

If for some i one gets

$$z_i \not\equiv \pm 1 \pmod{N} \quad \text{and} \quad z_{i+1} \equiv 1 \pmod{N},$$

then $z_i^2 \equiv 1 \pmod{N}$ but $z_i \not\equiv \pm 1$, so $z_i^2 - 1$ is a non-trivial multiple of N. This yields

$$\gcd(z_i - 1, N)$$

as a non-trivial factor of N. Repeating with random a until a non-trivial square root of 1 modulo N is found gives a probabilistic reduction from knowing d to factoring N.

> **Exam Tip: Key Idea:** Any efficient algorithm that outputs a non-trivial square root of 1 modulo N can be turned into a factoring algorithm. The above construction shows that knowing d gives such square roots with good probability.

2 Factorization Algorithms and L-Notation

2.1 The L-Notation

For many factorization algorithms, the running time on an integer N is well described by the sub-exponential L-notation. For parameters $0 \le \alpha \le 1$ and $c > 0$, define

$$L_N[\alpha, c] = \exp\Big((c + o(1))(\ln N)^\alpha (\ln \ln N)^{1-\alpha}\Big).$$

Some special cases are:

- $\alpha = 0$: $L_N[0, c] = (\ln N)^{c+o(1)}$, essentially polynomial in $\ln N$.
- $\alpha = 1$: $L_N[1, c] = N^{c+o(1)}$, i.e. exponential in $\ln N$.
- $\alpha = \frac{1}{2}$: "classical" sub-exponential regime, used for the Quadratic Sieve and Elliptic Curve Method.
- $\alpha = \frac{1}{3}$: faster sub-exponential regime of the Number Field Sieve.

For sufficiently large N, if α is the same and $c < c'$, then $L_N[\alpha, c]$ is asymptotically faster than $L_N[\alpha, c']$.

2.2 Overview of Main Algorithms

The following table summarizes typical complexity classes and rough practical ranges for several important algorithms:

Algorithm	Asymptotic complexity	Rough practical range
Trial division	$O(\sqrt{N}) = L_N[1, \frac{1}{2}]$	Only very small N or tiny factors
Pollard's $p - 1$	Heuristic $L_N[0, c]$	Good when $p - 1$ is smooth
Pollard's ρ	$O(N^{1/4}) = L_N[1, \frac{1}{4}]$	Medium-size factors (e.g. up to ~ 60 bits)
Lenstra ECM	$L_N[\frac{1}{2}, c]$	Very good for medium primes (e.g. < 170 bits)
Quadratic Sieve (QS)	$L_N[\frac{1}{2}, 1]$	General-purpose up to $\sim 100-110$ digits
GNFS	$L_N[\frac{1}{3}, (64/9)^{1/3}]$	State-of-the-art for large RSA (e.g. RSA-250 and beyond)

The QS and GNFS are general-purpose algorithms, while Pollard's $p - 1$ and ECM exploit special structure such as smoothness of group orders or medium-sized prime factors.

3 Generic Factorization Algorithms

These algorithms do not rely on the size of N directly, but often on the size of the smallest prime factor p.

3.1 Pollard's $p - 1$ Method

This algorithm is effective when $p - 1$ is **B-smooth** (i.e., all prime factors of $p - 1$ are small, $\leq B$).

Concept: By Fermat's Little Theorem, $a^{p-1} \equiv 1 \pmod{p}$. If $p - 1$ divides some integer K (e.g., $K = \text{lcm}(1, 2, \ldots, B)$), then

$$a^K \equiv 1 \pmod{p} \implies a^K - 1 = k \cdot p.$$

Thus, $\gcd(a^K - 1, N)$ will reveal the factor p.

Algorithm:

1. Choose a smoothness bound B.
2. Compute $K = \text{lcm}(1, 2, \ldots, B)$.
3. Pick a random base a (e.g., $a = 2$).
4. Compute $x \equiv a^K \pmod{N}$.
5. Compute $g = \gcd(x - 1, N)$.
6. If $1 < g < N$, then g is a factor.

Countermeasure: Use "safe primes" where $p = 2p' + 1$ with p' prime, so that $p - 1$ is not very smooth.

3.2 Pollard's ρ Algorithm

Pollard's ρ is a heuristic algorithm based on random walks and the birthday paradox. It finds a factor p in expected time about $O(\sqrt{p}) \approx O(N^{1/4})$.

Concept: Define an iteration function $f : \{0, \ldots, N-1\} \to \{0, \ldots, N-1\}$, typically $f(x) = x^2 + 1 \bmod N$. Starting from some x_0, consider the sequence

$$x_{i+1} = f(x_i) \bmod N.$$

Modulo a prime factor $p \mid N$, this sequence eventually becomes periodic, and collisions $x_i \equiv x_j \pmod{p}$ imply that p divides $x_i - x_j$. Computing $\gcd(|x_i - x_j|, N)$ can then yield p.

Floyd's Cycle-Finding (Tortoise and Hare) Instead of storing all previous values, two sequences are run:

$$x_{i+1} = f(x_i), \quad y_{i+1} = f(f(y_i)),$$

and at each step one computes

$$g = \gcd(|x_i - y_i|, N).$$

If $1 < g < N$, then g is a non-trivial factor of N. If $g = N$, one restarts with a different function or seed.

Optimizations (Brent and Accumulation) In practice, several optimizations improve performance:

- **Brent's cycle-finding** uses blocks of iterations and fewer gcd calls to detect cycles more efficiently.
- **Product accumulation** multiplies many differences $(x_i - y_i)$ modulo N and performs a gcd only occasionally, which reduces the number of expensive gcd computations.

3.3 Lenstra's Elliptic Curve Method (ECM)

The Elliptic Curve Method is particularly efficient for finding medium-sized prime factors of N. It generalizes the idea of Pollard's $p-1$ from the multiplicative group $(\mathbb{Z}/p\mathbb{Z})^\times$ to the group of points on random elliptic curves over $\mathbb{Z}/p\mathbb{Z}$.

Idea:

- Choose a random elliptic curve E modulo N and a random point P on E.
- Compute multiples $[k]P$ where k has many small prime factors (e.g. $k = \mathrm{lcm}(1, \ldots, B)$).
- Group operations require inversions modulo N; when an inversion fails, the corresponding denominator is not invertible and shares a non-trivial gcd with N, revealing a factor.

The expected running time to find a factor p depends mainly on the size of p, and is roughly

$$L_p[\tfrac{1}{2}, c]$$

for some constant c. ECM is usually the method of choice to strip off medium-size prime factors before applying QS or GNFS to the remaining cofactor.

4 Factorization via Congruence of Squares

4.1 The Core Idea

For large general integers (where factors are not small or special), methods based on congruences of squares are used. The central idea is:

If there exist integers x, y such that:

$$x^2 \equiv y^2 \pmod{N} \quad \text{and} \quad x \not\equiv \pm y \pmod{N}, \tag{1}$$

then N divides $x^2 - y^2 = (x - y)(x + y)$. Consequently,

$$\gcd(x - y, N)$$

is a non-trivial factor of N with high probability.

4.2 Dixon's Algorithm & The Quadratic Sieve (QS)

Finding a single congruence of squares directly is hard. Instead, Dixon's algorithm and the Quadratic Sieve build many relations and combine them to produce a square on each side.

High-level Steps:

1. **Factor base:** Choose a set of small primes $\mathcal{P} = \{p_1, \ldots, p_k\}$, called the factor base.
2. **Relation collection:** Search for integers x such that

   $$z = x^2 \bmod N$$

   is **B-smooth**, i.e. it factors completely over $\mathcal{P}$:

   $$x^2 \equiv \prod_{i=1}^{k} p_i^{e_i} \pmod{N}.$$

3. **Linear algebra:** Each relation gives a vector of exponents modulo 2,

   $$v = [e_1 \bmod 2, \ldots, e_k \bmod 2].$$

   Once more relations than primes in the factor base are collected, there must be a non-trivial linear dependency among these vectors modulo 2. Gaussian elimination over $\mathbb{F}_2$ yields a subset of relations whose exponents sum to the zero vector modulo 2.

4. **Building the squares:** Multiplying the corresponding congruences yields

   $$X^2 \equiv Y^2 \pmod{N},$$

   where X is the product of the x's and Y is the product of primes in $\mathcal{P}$ with halved exponents (since the exponents are even). This gives a congruence of squares and thus a chance to factor N.

4.3 Practical Quadratic Sieve: Sieving with a Polynomial

In practice, the Quadratic Sieve uses a specific polynomial and a sieving step to find many smooth values efficiently.

Setup Let
$$M = \left\lfloor \sqrt{N} \right\rfloor,$$
and define the quadratic polynomial
$$y(x) = (M + x)^2 - N.$$
For each integer x, set
$$a = M + x, \quad y(x) \equiv a^2 - N \pmod{N}.$$
If $y(x)$ is B-smooth over the factor base $\mathcal{P}$, then $(a, y(x))$ gives a useful relation:
$$a^2 \equiv y(x) \pmod{N}.$$

Sieving Step The sieving procedure works as follows:

- Precompute the factor base $\mathcal{P} = \{p_1, \ldots, p_k\}$ consisting of primes up to B for which N is a quadratic residue modulo p_j.
- For each $p_j \in \mathcal{P}$, solve
$$(M + x)^2 \equiv N \pmod{p_j}$$
for x. This quadratic congruence has either zero or two solutions modulo p_j, say $x \equiv r_j$ and $x \equiv r'_j \pmod{p_j}$.
- Initialize an array $V[x]$ with values $y(x)$ for x in some interval (e.g. $-A \leq x \leq A$).
- For each prime p_j, and for integers n, the values $x = r_j + np_j$ and $x = r'_j + np_j$ satisfy
$$y(x) \equiv 0 \pmod{p_j},$$
so p_j divides $y(x)$. One updates the array by repeatedly dividing $V[x]$ by p_j (and incrementing the exponent counter for p_j).
- After sieving with all primes in the factor base, entries where $|V[x]| = 1$ correspond to values $y(x)$ that are fully B-smooth, and their recorded exponents give relations for the linear algebra step.

Complexity The Quadratic Sieve has heuristic running time
$$L_N\left[\tfrac{1}{2}, 1\right] \approx \exp\left(\sqrt{\ln N \, \ln \ln N}\right),$$
making it significantly faster than Pollard's ρ for large general N, but slower than GNFS on very large inputs.

Exam Tip: Exam Context: The **Number Field Sieve (NFS)** further improves on QS and achieves complexity in $L_N[1/3, c]$. It is the current state-of-the-art for factoring large RSA moduli (e.g. RSA-768). For the exam, it is usually sufficient to know it exists, its $L_N[1/3]$ complexity, and that it outperforms QS on very large moduli.

5 Specific RSA Attacks (Non-Factorization)

5.1 Wiener's Attack (Small d)

If the private exponent d is too small (roughly $d < \frac{1}{3}N^{1/4}$), RSA is insecure.

- **Mechanism:** Wiener showed that if d is small, then the fraction $\frac{e}{N}$ has convergents $\frac{k}{d}$ in its continued fraction expansion that satisfy the key equation

$$ed - k\phi(N) = 1.$$

Trying convergents gives candidates for (k, d) that can be verified efficiently.

- **Implication:** The private exponent d must be chosen large enough; using a very small d to speed up decryption yields an RSA key that can be recovered without factoring N.

5.2 Coppersmith-Type Attacks (Small e)

Small public exponents e (like $e = 3$) are common in practice to speed up encryption. This remains safe under standard padding, but can be dangerous in simplified or pathological scenarios.

Broadcast Attack If the same message M is sent to e different recipients using the same small public exponent e but different moduli $N_1, \ldots, N_e$, and if no padding is used, then:

- The attacker observes ciphertexts $C_i = M^e \bmod N_i$.
- Using the Chinese Remainder Theorem, the attacker reconstructs the unique integer C such that

$$C \equiv C_i \pmod{N_i}$$

for all i, so $C \equiv M^e \pmod{N_1 \cdots N_e}$.

- If $M^e < N_1 \cdots N_e$, then no modular reduction occurs and $C = M^e$ as an integer, so the attacker can recover M by taking an ordinary e-th root over the integers.

6 Self-Assessment Questions

Level 1: Concepts

Q: Why does finding $x^2 \equiv y^2 \pmod{N}$ help factor N?
Answer: Because $x^2 - y^2 \equiv 0 \pmod{N}$ implies $(x-y)(x+y) = k \cdot N$. If $x \not\equiv \pm y \pmod{N}$, then neither $(x-y)$ nor $(x+y)$ is a multiple of N alone, meaning the factors of N are split between them. Computing $\gcd(x-y, N)$ reveals a non-trivial factor.

Q: What condition makes Pollard's $p-1$ algorithm extremely fast?
Answer: If $p-1$ is **smooth**, meaning it is composed entirely of small prime factors. This ensures that the chosen exponent K is a multiple of $p-1$, so $a^K \equiv 1 \pmod{p}$ and $\gcd(a^K - 1, N)$ reveals the factor.

Q: What is the role of the factor base in the Quadratic Sieve?
Answer: The factor base is the set of small primes over which smooth values $y(x)$ are factored. Only values that factor completely over this base are kept as relations, and their exponent vectors (modulo 2) are used in the linear algebra step to build a congruence of squares.

Level 2: Algorithms

Q: Explain the "Linear Algebra" step in the Quadratic Sieve.
Answer: Each relation $a_i^2 \equiv \prod_j p_j^{e_{j,i}} \pmod{N}$ gives a vector of exponents modulo 2, $v_i = [e_{1,i} \bmod 2, \ldots, e_{k,i} \bmod 2]$. After collecting more relations than primes in the factor base, Gaussian elimination over $\mathbb{F}_2$ yields a non-trivial linear dependency among the v_i; multiplying the corresponding relations produces a perfect square on the right-hand side and a congruence of squares modulo N.

Q: Why is Lenstra's ECM particularly suited for medium-size prime factors?
Answer: The running time of ECM depends mainly on the size of the smallest prime factor p rather than on N itself, with heuristic complexity of the form $L_p[\frac{1}{2}, c]$. This makes it very efficient when p is not too large (e.g. below roughly 170 bits), even if N is much larger.

Level 3: RSA Security

Q: Is breaking RSA equivalent to factoring N?
Answer: It is not proven. Factoring N certainly breaks RSA, because it allows computing $\phi(N)$, inverting e to obtain d, and hence decrypting any ciphertext. However, it remains an open problem whether there exists an efficient algorithm that can solve the RSA problem (or recover d) without explicitly factoring N.

Q: How can knowledge of the private exponent d be used to factor N?
Answer: Given d, write $ed - 1 = 2^s t$ with t odd and choose random a coprime to N. The sequence $z_0 = a^t \bmod N$, $z_{i+1} = z_i^2 \bmod N$ will eventually contain a non-trivial square root of 1 modulo N, i.e. some z_i with $z_i^2 \equiv 1 \pmod{N}$ but $z_i \not\equiv \pm 1 \pmod{N}$. Then $\gcd(z_i - 1, N)$ gives a non-trivial factor of N.