Exploring RSA: Attacks and Enhancements

Zhou Zihan 12310814

Intoduction:

I would like to choose the topic RSA and do some further searches. What we learn in the class is the most simple way to generate RSA key and may not have security targeted to specific attacks, **so I want to find some common attack measures to learn the mechnism and danger, so that I could search for some ways to optimize the traditional RSA.** What I write in this article is the thing I learned from some essays and teaching videos, also there are CPP codes written by myself to implement some algorithm. From this way, I have a deeper understanding about the advanced cryptography.

Content:

First I search for the common ways to attack, and I found two of most common approaches, which are factorization attack and Chosen Plaintext Attack. I'll intoduce these two ways and find some optimazation to fix these security problems.

Factorization Attack:

I found several ways to do the factorization. I get insipired by the video

入门算法课(4)-质数和筛法 | 质数 | 埃氏筛 | 线性筛 | 二次筛法 | 编程 | 算法竞赛 | 数学*哔哩哔哩* bilibili

and I conclude them with CPP codes below.

Trial Division

```
void trialDivision(int n) {
    while (n % 2 == 0) {
        cout << 2 << " ";
        n /= 2;
    }
    for (int i = 3; i <= sqrt(n); i += 2) {
        while (n % i == 0) {
            cout << i << " ";
            n /= i;
        }
    }
    if (n > 2) {
        cout << n << " ";
    }
}
```

Sieve of Eratosthenes

```
void sieveOfEratosthenes(int N) {
    vector<bool> isPrime(N + 1, true);
    isPrime[0] = isPrime[1] = false;

    for (int p = 2; p * p <= N; ++p) {
        if (isPrime[p]) {
            for (int i = p * p; i <= N; i += p) {
                isPrime[i] = false;
            }
        }
    }
    for (int p = 2; p <= N; ++p) {
        if (isPrime[p]) {
            cout << p << " ";
        }
    }
}
```

Linear Sieve

```
void linearSieve(int N) {
    vector<int> prime;
    vector<bool> isPrime(N + 1, true);
    isPrime[0] = isPrime[1] = false;

    for (int i = 2; i <= N; ++i) {
        if (isPrime[i]) {
            prime.push_back(i);
        }

        for (int j = 0; j < prime.size() && prime[j] <= N / i; ++j) {
            isPrime[prime[j] * i] = false;
            if (i % prime[j] == 0) {
                break;
            }
        }
    }

    for (int p : prime) {
        cout << p << " ";
    }
}
```

Quadratic Sieve

```cpp
void quadraticSieve(int N) {
    // Choose Factor base
    vector<int> factorBase;
    for (int i = 2; i <= sqrt(N); ++i) {
        if (isPrime(i)) {
            factorBase.push_back(i);
        }
    }

    // Construct Squre numbers and filter
    vector<int> squares;
    for (int i = 1; i <= sqrt(N); ++i) {
        int square = i * i % N;
        if (isFactor(square, factorBase))
            squares.push_back(square);
        }
    }

    // Find proper factor
    for (int i = 0; i < squares.size(); ++i) {
        if (findFactor(squares[i])) {
            cout << "Factor found: " << squares[i] << endl;
        }
    }
}

bool isPrime(int num) {
    if (num <= 1) return false;
    for (int i = 2; i <= sqrt(num); ++i) {
        if (num % i == 0) return false;
    }
    return true;
}

bool isFactor(int num, const vector<int>& factorBase) {
    for (int p : factorBase) {
        if (num % p == 0) {
            return true;
        }
    }
    return false;
}
```

For their runtime and proper usage:

Trial Division: Suitable for factorizing integers within a small range;

Sieve of Eratosthenes: Its time complexity is $O(N \log \log N)$, making it efficient for finding primes in a moderate range.

Linear Sieve: More efficient than the Sieve of Eratosthenes, making it suitable for finding primes in a large range. Its time complexity is $O(N)$, which is optimal for prime sieving.

Quadratic Sieve: Suitable for factorizing large integers. It is an advanced factorization algorithm with a time complexity of:

$$O\left(\exp\left(\frac{1}{2}\sqrt{\ln N \ln \ln N}\right)\right)$$

It is particularly effective for large number factorization tasks like RSA key cracking.

Through there approches to attack, the most Optimal Algorithm for lager integer is Quadratic Sieve, which is always the main approach among factorization attacks. Next, let me talk about the advantages and disadvantages I have collected about **Factorization Attack**.

Advantages:

1. **Simplicity and Directness**:
    - Factorization attacks are simple and directly target the core security assumption of RSA encryption: the difficulty of factorizing large composite numbers. This makes the attack method easier to understand and implement.
2. **Known Success for Small Key Sizes**:
    - For smaller RSA key sizes (e.g., 512-bit or 768-bit keys), factorization attacks are quite effective. Historically, factorization attacks have been successful against these weaker key sizes, demonstrating their potential for breaking RSA encryption when it is improperly implemented with small keys.

Disadvantages:

1. **Highly Computationally Expensive for Large Key Sizes**:
    - Factorization attacks become impractical as the size of the RSA key grows. For key sizes of 2048 bits and above (currently considered secure), the computational cost of factorization grows exponentially, making such attacks infeasible with current computational resources.
2. **Not a Universal Solution**:
    - Factorization attacks are limited to breaking RSA-based systems. However, modern cryptography uses a variety of algorithms (e.g., elliptic curve cryptography, post-quantum algorithms) that do not rely on factorization for security, making factorization attacks a less universal threat to encryption schemes as a whole.
3. **Vulnerability to Optimizations and Safeguards**:
    - Modern RSA implementations often include safeguards like padding schemes (e.g., PKCS#1), key length recommendations, and random key generation, which reduce the effectiveness of factorization attacks. Attackers need to bypass these additional measures to perform a successful factorization, which makes the attack more difficult and less likely to succeed.

Chosen Plaintext Attack (CPA)

Definition: A Chosen Plaintext Attack (CPA) is a type of cryptographic attack where the attacker has the capability to **choose arbitrary plaintexts and obtain their corresponding ciphertext**s. The goal of the attacker is to gain some useful information about the encryption key or the structure of the encryption algorithm, which can then be used to decrypt other ciphertexts or to encrypt additional plaintexts.

Here is a little example about CPA:
Consider a simple substitution cipher where each letter in the plaintext is replaced by a fixed letter in the ciphertext. An attacker using a CPA might choose plaintexts like "A", "B", "C", etc., and observe the corresponding ciphertexts. By analyzing the mappings, the attacker can deduce the entire substitution table, allowing them to decrypt any ciphertext.

Importantly, we should notice that RSA doesn't have CPA security, because RSA is determinism, which means every decryption acquires the smae result. As a result, if a hacker could use encryption do generate some ciphertexts according to some known plaintexts, the RSA will be broken.

From the video 10.4 公钥加密与选择明文攻击*哔哩哔哩*bilibili I learn the related proof and knowleges about RSA and CPA. Below is the proof why RSA doesn't have CPA security.

Proof: Suppose the adversary knows that the user has encrypted one of the messages, either m_1 or m_2. The adversary also knows the user's public key, which is e and n. When the adversary is given a ciphertext c and asked to determine whether the corresponding plaintext m is m_1 or m_2, the adversary only needs to compute: $c\prime = m_1^e mod n$

If $c\prime = c$, then the adversary knows that $m = m_1$. Otherwise, the adversary knows that $m = m_2$.

Here are the advantages and disadvantages I collected about CPA:

Advantages of Chosen Plaintext Attack (CPA):

1. **Effective against weak or unprotected implementations**: Can break RSA encryption if no proper padding schemes or safeguards are in place.
2. **Can reveal key information**: If the attacker has access to a large number of plaintext-ciphertext pairs, it can help in extracting private keys or cracking the encryption.

Disadvantages of Chosen Plaintext Attack (CPA):

1. **Requires access to plaintext-ciphertext pairs**: The attacker needs to control or access plaintexts and their corresponding encrypted outputs, which isn't always possible.
2. **Limited by modern security measures**: Proper padding schemes (e.g., PKCS#1) and hybrid encryption systems make CPA less effective or infeasible in practice.

How can we improve RSA according to the two attacking ways mentioned above?

For Factorization Attack security:

Firstly, I want to find a standard to calculate the security rank of RSA, and I find two concepts called **Key Security Coefficient** and **Security Strength Threshold** and I'll do some analyses on them. Maybe by meeting the standards, the RSA could handle factorization attacks.

Definition of Key Security Coefficient

Definition: Given two **strong primes** p and q, and an integer u such that

$gcd(u, (p-1)(q-1)) = 1$, if there exists the **smallest positive integer s** such that $us \equiv 1 mod (p-1)(q-1)$, then s is defined as the security strength of the key (u, pq).

Explanation:

- **Strong Primes**: Strong primes are primes of the form $p = 2p_1 + 1$ and $q = 2q_1 + 1$, where p_1 and q_1 are also primes. These primes enhance the security of RSA keys because they ensure that $(p-1)(q-1)$ contains large prime factors, making factorization more difficult.
- **Smallest Positive Integer s**: This integer s is the smallest value such that us is congruent to 1 modulo $(p-1)(q-1)$. This value reflects the periodicity of the key u under modulo $(p-1)(q-1)$, i.e., the number of modular exponentiation operations required to recover the original plaintext.

Definition of Security Strength Threshold

Definition: Given two strong primes p and q, the Euler function value

$\phi((p-1)(q-1))$ is recursively divided by 2 until an odd number is obtained. This odd number, denoted as $f_a(p,q)$, is called the security strength threshold of the keys derived from the prime pair (p,q).

Calculation Method:

- **Euler Function Value**:

  $\phi((p-1)(q-1))$ calculates the number of integers coprime to

  $(p-1)(q-1)$. For strong primes p and q, $\phi((p-1)(q-1))$ can be expressed as:

  $\phi((p-1)(q-1)) = \frac{(p-1)(q-1)}{2} = 2(p_1 - 1)(q_1 - 1)$
- **Recursive Division by 2**: Starting from

  $\phi((p-1)(q-1))$, repeatedly divide by 2 until the result is an odd number. This odd number $f_a(p,q)$ is the security strength threshold.

After learning the definitions of these two concepts, I also learn **the Relationship Between Key Security Coefficient and RSA Security** through an article.

1. **Quantification of Key Security Coefficient**:
   - The key security coefficient s quantifies the periodicity of the key u under modulo $(p-1)(q-1)$. A larger s means that an attacker needs more modular exponentiation operations to recover the plaintext, thus increasing the difficulty of the attack. For example, if s is small, an attacker can find the key's periodicity with fewer operations, thereby recovering the plaintext. Therefore, a larger s significantly enhances the security of the RSA key.
2. **Role of Security Strength Threshold**:
   - The security strength threshold $f_a(p,q)$ provides a **benchmark** for evaluating the security of the key. The key's security coefficient s should be greater than or equal to this threshold to ensure sufficient security.For example, if s is much smaller than $f_a(p,q)$, the key is considered insecure because an attacker can find the key's periodicity with fewer operations, thereby recovering the plaintext.
3. **Practical Application**:
   - When generating RSA keys, it is crucial to compute the security coefficient of each key and ensure it is above the security strength threshold. Such keys are called secure keys, and their introduction and application in security systems can effectively defend against known plaintext attacks. For example, in the paper, a 512-bit RSA key pair with a security coefficient of 2 times the threshold was mentioned. Attacking it would require approximately 10^{143} years of encryption operations, which significantly reduces the

probability of generating weak keys randomly, thereby effectively enhancing the security of RSA applications.

Additionally, I try to implement the algorithm mentioned by using CPP:

```
#include <iostream>
#include <random>
#include <cassert>
#include <gmpxx.h> // Use for big numbers

// Generate a strong prime
mpz_class generate_strong_prime(int bits) {
    mpz_class p, p1;
    do {
        p = mpz_class::random(bits);
        p1 = (p - 1) / 2;
    } while (!mpz_probab_prime_p(p.get_mpz_t(), 25) ||
!mpz_probab_prime_p(p1.get_mpz_t(), 25));
    return p;
}

// Calculate ϕ((p-1)(q-1))
mpz_class euler_phi(mpz_class p, mpz_class q) {
    mpz_class p1 = p - 1;
    mpz_class q1 = q - 1;
    return (p1 * q1) / mpz_gcd(p1.get_mpz_t(), q1.get_mpz_t());
}

// Evaluate the security coefficient of RSA
mpz_class evaluate_security_coefficient(mpz_class p, mpz_class q, mpz_class u) {
    mpz_class phi = euler_phi(p, q);
    mpz_class s = 1;
    mpz_class mod = mpz_powm(u.get_mpz_t(), s.get_mpz_t(), phi.get_mpz_t());

    while (mod != 1) {
        s += 1;
        mod = mpz_powm(u.get_mpz_t(), s.get_mpz_t(), phi.get_mpz_t());
    }
    return s;
}

int main() {
    // Generate two strong primes
    mpz_class p = generate_strong_prime(128);
    mpz_class q = generate_strong_prime(128);

    // Generate public key, normally 65537
    mpz_class u = 65537;

    // Ensure u and ϕ((p-1)(q-1)) are coprime
    mpz_class phi = euler_phi(p, q);
    assert(mpz_gcd(u.get_mpz_t(), phi.get_mpz_t()) == 1);

    // Assess the security coeffienct of RSA
    mpz_class security_coefficient = evaluate_security_coefficient(p, q, u);
    std::cout << "Security Coefficient: " << security_coefficient << std::endl;

    return 0;
}
```

As a result, we figure out there is a method to ensure the security of RSA and we have the algorithm to acquire that. Next, I'll find some commom approaches to break RSA.

For RSA security:

1. Padding Schemes

Padding schemes are used to prevent attacks like **Chosen Plaintext Attacks (CPA)** by modifying the plaintext before encryption. This ensures that even if the attacker selects the same plaintext, the resulting ciphertext will differ, preventing valuable information from being inferred.

Why Padding is Needed:

RSA encryption is deterministic, meaning the same plaintext always produces the same ciphertext. Padding ensures that even if an attacker controls the plaintext, the ciphertext will vary, making it harder to exploit.

Common Padding Schemes:

1. **PKCS#1 (Public Key Cryptography Standards #1)**: PKCS#1 **adds random padding** to the plaintext, making the ciphertext different each time the same plaintext is encrypted. This scheme prevents CPA and makes reverse decryption difficult.
2. **OAEP (Optimal Asymmetric Encryption Padding)**: OAEP is a more advanced version of PKCS#1 that uses **hashing and randomness** to generate more complex padding, offering better security against attacks.

Padding schemes like PKCS#1 and OAEP modify plaintext before encryption, ensuring that **identical plaintexts do not produce the same ciphertext**. This prevents attacks like CPA and makes RSA encryption more secure.

2. Hybrid Encryption

Hybrid encryption combines **symmetric encryption** (fast and efficient) and **asymmetric encryption** (secure for key exchange) to optimize security and performance.

How Hybrid Encryption Works:

1. **Generate a Symmetric Key**: A random symmetric key (e.g., AES) is generated to encrypt the data.
2. **Encrypt the Data**: The data is encrypted using the symmetric key.
3. **Encrypt the Symmetric Key**: The symmetric key is encrypted with the recipient’s public key (e.g., RSA).
4. **Send Ciphertext and Key**: The encrypted data and the encrypted symmetric key are sent to the recipient.
5. **Decryption**:
    - The recipient decrypts the symmetric key with their private key.
    - They then use the decrypted symmetric key to decrypt the data.

Hybrid encryption combines **the security of asymmetric encryption** for key exchange and **the speed of symmetric encryption for data**, making it ideal for secure communication systems like HTTPS.

Conclusion

This article discusses RSA encryption, highlighting its principles, attack methods, and optimization strategies. It explains vulnerabilities like factorization and chosen plaintext attacks, detailing common attack algorithms. The article also explores RSA optimization, focusing on key security metrics such as the Key Security Coefficient and Security Strength Threshold, with CPP code implementations. It emphasizes the importance of padding schemes, like PKCS#1 and OAEP, in enhancing security.

What really makes me interested on this topic is a historic story in World War II: During World War II, the U.S. Navy initially faced skepticism from high-level officials about a cryptographic message intercepted by their codebreakers. The message indicated that Japan was planning to attack Midway. To confirm this, U.S. forces on Midway sent a false message stating that the island lacked fresh water, prompting the Japanese to encrypt it. The U.S. intercepted the encrypted message, verifying the authenticity of the information. This intelligence breakthrough allowed the U.S. Navy to prepare for the Japanese attack, leading to a decisive victory at the Battle of Midway. The story reveals the successful implementation of CPA and let me consider the imortance of security of RSA and more encryptions and decryptions.

Reference

Wenxue, T., Xiping, W., Jinju, X., & Meisen, P. (2010). A mechanism of quantitating the security strength of RSA key. In *2010 Third International Symposium on Electronic Commerce and Security* (pp. 357-361). IEEE. https://doi.org/10.1109/ISECS.2010.85

Bilibili. (2021, May 11). *入门算法课(4)-质数和筛法 | 质数 | 埃氏筛 | 线性筛 | 二次筛法 | 编程 | 算法竞赛 | 数学* [Video]. Bilibili. https://www.bilibili.com/video/BV1LC4y117H4/?spm_id_from=333.337.search-card.all.click&vd_source=ab21e6557ef94cfe52401dd41fa38ad3

Bilibili. (2021, June 9). *10.4 公钥加密与选择明文攻击* [Video]. Bilibili. https://www.bilibili.com/video/BV1Ub4y137rJ/?spm_id_from=333.337.search-card.all.click&vd_source=ab21e6557ef94cfe52401dd41fa38ad3

I would like to express my gratitude to professor for reading this far. I am still inexperienced, and if there are any mistakes, I kindly ask for your understanding. Additionally, I sincerely appreciate the professor's excellent teaching and thoughtful guidance throughout this semester, which has sparked my interest in discrete mathematics.