**BÀI TẬP LỚN**

**MÔN: AN TOÀN BẢO MẬT HỆ THỐNG THÔNG TIN**

**ĐỀ TÀI: TÌM HIỂU VỀ GIẢI THUẬT MÃ HÓA CÔNG KHAI RSA**

***THÀNH VIÊN:***

1. ***NGUYỄN HUY HOÀNG MSV: B13DCCN372***
2. ***NGUYỄN NHẬT HỒNG MSV: B13DCCN020***
3. ***VIÊN TUẤN HÙNG MSV: B13DCCN470***
4. ***NGUYỄN THỊ THU HƯƠNG MSV:B13DCCN146***
5. ***NGUYỄN THỊ HƯƠNG MSV:B13DCCN370***

Mục Lục

[**I.** **Giới thiệu chung** 3](#_Toc463644215)

[**II.** **Giới thiệu về giải thuật RSA** 3](#_Toc463644216)

[*1.* *Khái niệm* 3](#_Toc463644217)

[*2.* *Một tả hoạt động* 3](#_Toc463644218)

[**III.** **Giải thuật RSA** 4](#_Toc463644219)

[*1.* *Cách tạo khóa* 4](#_Toc463644220)

[*2.* *Mã hóa, Giải mã* 6](#_Toc463644221)

[*3.* *Sơ đồ toàn bộ quá trình* 7](#_Toc463644222)

[**IV.** **Điểm yếu** 8](#_Toc463644223)

[**V.** **Các dạng tấn công** 9](#_Toc463644224)

[*1.* *Modul chung* 9](#_Toc463644225)

[*2.* *Sử dụng số mũ hóa e nhỏ:* 10](#_Toc463644226)

[*3.* *Sử dụng số mũ giải mã d nhỏ* 10](#_Toc463644227)

[**VI.** **Ứng dụng demo** 10](#_Toc463644228)

[*1.* *Cài đặt RSA trong thực tế* 10](#_Toc463644229)

[*2.* *Cài đặt trong java* 10](#_Toc463644230)

[*3.* *Demo ứng dụng* 11](#_Toc463644231)

[**VII.** **Kết luận** 14](#_Toc463644232)

[**VIII.** **Tài liêu tham khảo** 14](#_Toc463644233)

1. **Giới thiệu chung**

Giải thuật được [Ron Rivest](https://vi.wikipedia.org/w/index.php?title=Ron_Rivest&action=edit&redlink=1), [Adi Shamir](https://vi.wikipedia.org/w/index.php?title=Adi_Shamir&action=edit&redlink=1) và [Len Adleman](https://vi.wikipedia.org/w/index.php?title=Len_Adleman&action=edit&redlink=1) mô tả lần đầu tiên vào năm [1977](https://vi.wikipedia.org/wiki/1977) tại [Học viện Công nghệ Massachusetts](https://vi.wikipedia.org/wiki/H%E1%BB%8Dc_vi%E1%BB%87n_C%C3%B4ng_ngh%E1%BB%87_Massachusetts) (MIT). Tên của thuật toán lấy từ 3 chữ cái đầu của tên 3 tác giả. Nó đánh dấu một sự tiến hóa vượt bậc của lĩnh vực mật mã học trong việc sử dụng mã công khai. RSA đang được sử dụng phổ biến trong thương mại điện tử và được cho là đảm bảo an toàn với điều kiện độ dài khóa đủ lớn.

Trước đó, năm 1973, Clifford Cocks, một nhà toán học người Anh làm việc tại GCHQ đã mô tả mô tả một thuật toán tương tự. Với khả năng tính toán tại thời điểm đó thì thuật toán này không khả thi và chưa bao giờ được thực nghiệm. Tuy nhiên, phát minh này chỉ được công bố vào năm [1997](https://vi.wikipedia.org/wiki/1997) vì được xếp vào loại tuyệt mật.

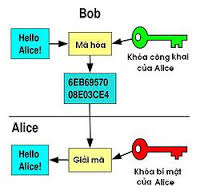
Thuật toán RSA được MIT đăng ký bằng sáng chế tại Hoa Kỳ vào năm [1983](https://vi.wikipedia.org/wiki/1983) (Số đăng ký 4.405.829). Bằng sáng chế này hết hạn vào ngày [21 tháng 9](https://vi.wikipedia.org/wiki/21_th%C3%A1ng_9) năm [2000](https://vi.wikipedia.org/wiki/2000). Tuy nhiên, do thuật toán đã được công bố trước khi có đăng ký bảo hộ nên sự bảo hộ hầu như không có giá trị bên ngoài Hoa Kỳ. Ngoài ra, nếu như công trình của Clifford Cocks đã được công bố trước đó thì bằng sáng chế RSA đã không thể được đăng ký.

1. **Giới thiệu về giải thuật RSA**
2. *Khái niệm*

Trong mật mã học, **RSA** là một thuật toán mật mã hóa khóa công khai. Đây là thuật toán đầu tiên phù hợp với việc tạo ra chữ ký điện tử đồng thời với việc mã hóa. Nó đánh dấu một sự tiến bộ vượt bậc của lĩnh vực mật mã học trong việc sử dụng khóa công cộng. RSA đang được sử dụng phổ biến trong thương mại điện tử và được cho là đảm bảo an toàn với điều kiện độ dài khóa đủ lớn.

1. *Một tả hoạt động*

Thuật toán RSA có hai khóa: khóa công khai (hay khóa công cộng) và khóa bí mật (hay khóa cá nhân). Mỗi khóa là những số cố định sử dụng trong quá trình mã hóa và giải mã. Khóa công khai được công bố rộng rãi cho mọi người và được dùng để mã hóa. Những thông tin được mã hóa bằng khóa công khai chỉ có thể được giải mã bằng khóa bí mật tương ứng. Nói cách khác, mọi người đều có thể mã hóa nhưng chỉ có người biết khóa cá nhân (bí mật) mới có thể giải mã được.

**

Ta có thể mô phỏng trực quan một hệ mật mã khoá công khai như sau: Bob muốn gửi cho Alice một thông tin mật mà Bob muốn duy nhất Alice có thể đọc được. Để làm được điều này, Alice gửi cho Bob một chiếc hộp có khóa đã mở sẵn và giữ lại chìa khóa. Bob nhận chiếc hộp, cho vào đó một tờ giấy viết thư bình thường và khóa lại (như loại khoá thông thường chỉ cần sập chốt lại, sau khi sập chốt khóa ngay cả Bob cũng không thể mở lại được-không đọc lại hay sửa thông tin trong thư được nữa). Sau đó Bob gửi chiếc hộp lại cho Alice. Alice mở hộp với chìa khóa của mình và đọc thông tin trong thư. Trong ví dụ này, chiếc hộp với khóa mở đóng vai trò khóa công khai, chiếc chìa khóa chính là khóa bí mật.

1. **Giải thuật RSA**
2. *Cách tạo khóa*
3. Giải thuật

Thủ tục sinh khóa RSA:

* Tạo 2 số nguyên tố p và q;
* Tính n = p x q
* Tính Φ(n)= (p-1)(q-1)  .
* Chọn số e sao cho 0 < e < Φ(n) và gcd(e, Φ (n)) = 1
* Chọn số d sao cho d = e -1 mod Φ(n), hoặc (d x e) mod Φ(n) = 1 (d là molulo nghịch đảo của e)
* Ta có (n, e) là khóa công khai, (n, d) là khóa riêng.

1. Ví dụ

* Chọn 2 số nguyên tố p=3 và q=11
* Tính n = p x q = 3 x 11 = 33
* Tính Φ(n) = (p-1) x (q-1) = 2 x 10 = 20
* Chọn số e sao cho 0 < e < 20, và e và Φ(n) là số nguyên tố cùng nhau (Φ(n) không chia hết cho e). Chọn e = 7
* Tính (d x e) mod Φ(n) 🡪 (d x 7) mod 20 = 1; d = (20\*k +1)/7 🡪 d = 3 (k=1)

=>Khóa công khai (33, 7), khóa bí mật (33, 3).

1. Lưu ý khi tạo khóa

- Việc phân tích n (n = pq) là không khả thi về mặt tính toán

- p và q nên có cùng độ lớn (tính bằng bit), là các số đủ lớn

- p – q không nên quá nhỏ,

- Khi có được p 🡪 tính q, và tìm ra d

- Nếu p và q được chọn ngẫu nhiên thì p – q đủ lớn

- Số mũ mã hóa (e) nhỏ có thể tăng tốc độ mã hóa;

- Kẻ tấn công có thể nghe trộm và lấy được bản mã, từ đó phân tích bản mã để khôi phục bản rõ. Do số mũ nhỏ nên chi phí cho phân tích/vét cạn không quá lớn;

->Phòng chống:

+ Sử dụng số mũ e lớn;

+ Thêm chuỗi ngẫu nhiên vào khối rõ trước khi mã hóa.

- Khi sử dụng số mũ giải mã (d) nhỏ, có thể tăng tốc độ giải mã;

- Nếu d nhỏ và gcd(p-1, q-1) (gcd: ước số chung lớn nhất) cũng nhỏ thì d có thể tính được tương đối dễ dàng từ khóa công khai (n, e);

Phòng chống: Sử dụng số mũ d đủ lớn.

1. Quá trình tạo khóa thực tế

* Việc tìm ra 2 số nguyên tố đủ lớn *p* và *q* thường được thực hiện bằng cách thử xác suất các số ngẫu nhiên có độ lớn phù hợp
* *p* và *q* còn cần được chọn không quá gần nhau để phòng trường hợp phân tích *n* bằng phương pháp phân tích Fermat
* Tránh sử dụng các phương pháp tìm số ngẫu nhiên mà kẻ tấn công có thể lợi dụng). Yêu cầu các số được lựa chọn cần đồng thời ngẫu nhiên và không dự đoán được
* Khóa bí mật *d* phải đủ lớn. Năm 1990, Wiener chỉ ra rằng nếu giá trị của *p* nằm trong khoảng *q* và 2*q* (khá phổ biến) và *d* < *n*1/4/3 thì có thể tìm ra được *d* từ *n* và *e*.
* Hiện nay các số mũ nhỏ không còn được sử dụng do có thể tạo nên những lỗ hổng
* Giá trị thường dùng hiện nay là 65537 vì được xem là đủ lớn và cũng không quá lớn ảnh hưởng tới việc thực hiện hàm mũ.

1. *Mã hóa, Giải mã*
2. Chuyển đổi văn bản rõ

Trước khi thực hiện mã hóa, ta phải thực hiện việc chuyển đổi văn bản rõ (chuyển đổi từ *M* sang *m*) sao cho không có giá trị nào của M tạo ra văn bản mã không an toàn. Nếu không có quá trình này, RSA sẽ gặp phải một số vấn đề sau:

* Nếu *m* = 0 hoặc *m* = 1 sẽ tạo ra các bản mã có giá trị là 0 và 1 tương ứng
* Khi mã hóa với số mũ nhỏ (chẳng hạn *e* = 3) và *m* cũng có giá trị nhỏ, giá trị {\displaystyle m^{e}}me  cũng nhận giá trị nhỏ (so với *n*). Như vậy phép môđun không có tác dụng và có thể dễ dàng tìm được *m* bằng cách khai căn bậc *e* của *c* (bỏ qua môđun).
* RSA là phương pháp [mã hóa xác định](https://vi.wikipedia.org/w/index.php?title=M%C3%A3_h%C3%B3a_x%C3%A1c_%C4%91%E1%BB%8Bnh&action=edit&redlink=1) (không có thành phần ngẫu nhiên) nên kẻ tấn công có thể thực hiện [tấn công lựa chọn bản rõ](https://vi.wikipedia.org/w/index.php?title=T%E1%BA%A5n_c%C3%B4ng_l%E1%BB%B1a_ch%E1%BB%8Dn_b%E1%BA%A3n_r%C3%B5&action=edit&redlink=1) bằng cách tạo ra một bảng tra giữa bản rõ và bản mã. Khi gặp một bản mã, kẻ tấn công sử dụng bảng tra để tìm ra bản rõ tương ứng.

1. Giải thuật mã hóa

* Mã hóa đoạn thông tin M cần gửi thành 1 số m<n theo 1 hàm có thể đảo ngược theo 1 phương pháp thỏa thuận(được mô tả ở phần chuyển đổi bản rõ).
* Tính *c* là bản mã hóa của *m* theo công thức: **c= me mod n**

1. Giải thuật giải mã

Sau khi nhận c, sử dụng khóa bí mật d tìm được m từ c theo công thức: **m= cd mod n.**

1. Ví dụ

* Mã hóa:

• Với m = 6,

• c = me mod n = 67 mod 33 = 279936 % 33 = 30

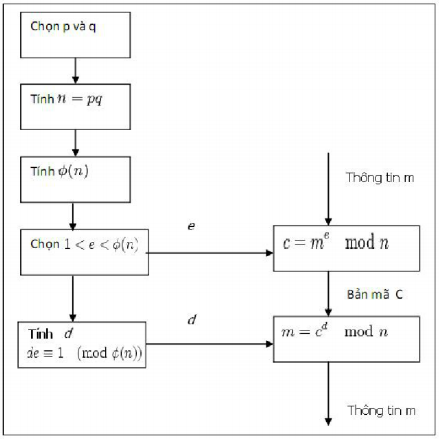
• => c = 30

* Giải mã:

• m = cd mod n = 303 mod 33 = 27000 % 33 = 6

• => m = 6

1. *Sơ đồ toàn bộ quá trình*



1. **Điểm yếu**
   * + Điểm yếu lớn nhất của giải thuật mã hóa công khai(bất đối xứng) RSA là tốc độ mã hóa và giải mã rất chậm so với mã hóa đối xứng, nếu dùng mã hóa RSA để mã hóa dữ liệu truyền-nhận giữa hai bên thì sẽ tốn rất nhiều chi phí.
     + Một khóa công khai nào đó có thể liên quan tới một số lượng lớn và khó xác định người sử dụng. Vì thế sẽ tốn rất nhiều thời gian khi muốn thu hồi hoặc thay thế một khóa vì lý do an ninh.
     + Sự an toàn của thành phần khóa mật (private key) phụ thuộc vào tính khó của việc phân tích thừa số nguyên tố (PTTSNT) các số lớn. Tuy nhiên, độ an toàn của RSA chưa chắc hoàn toàn tương đương với tính khó của bài toán, tức là có thể tồn tại phép tấn công phá vỡ được RSA mà không cần phải biết PTTSNT đối với n.
     + Trong hệ RSA, không phải tất cả các thông tin đều được che giấu tốt, tức là mọi khoá đều tốt và đều làm bản rõ thay đổi hoàn toàn.

Ví dụ 1: n = 35 = 5 x 7; m = 4 x 6

* e = 5 (GCD (5,24) = 1)
* X = 8
* Đối với bất kỳ khoá nào tồn tại ít nhất 9 bản rõ bị ‘phơi mặt’, tuy nhiên đối với n ≥ 200 điều đó không còn quan trọng. Mặc dù vậy phải chú ý là nếu e không được chọn cẩn thận thì có thể gần đến 50% bản rõ bị lộ.
* Ví dụ 2: Với n = 35, e = 17 {1, 6, 7, 8, 13, 14, 15, 20, 21, 27, 28, 29, 34} không che được
* Người ta cho rằng có thể tránh được tình huống này nếu số nguyên tố được chọn là an toàn. Một số nguyên tố được gọi là an toàn nếu p=2p’ +1 trong đó p’ cũng là số nguyên tố.
  + - Tồn tại khả năng một người nào đó có thể tìm ra được khóa bí mật. Không giống với hệ thống mật mã sử dụng một lần (one-time pad) hoặc tương đương, chưa có thuật toán mã hóa khóa bất đối xứng nào được chứng minh là an toàn trước các tấn công dựa trên bản chất toán học của thuật toán. Khả năng một mối quan hệ nào đó giữa 2 khóa hay điểm yếu của thuật toán dẫn tới cho phép giải mã không cần tới khóa hay chỉ cần khóa mã hóa vẫn chưa được loại trừ. An toàn của các thuật toán này đều dựa trên các ước lượng về khối lượng tính toán để giải các bài toán gắn với chúng. Các ước lượng này lại luôn thay đổi tùy thuộc khả năng của máy tính và các phát triển toán học mới.
    - Một điểm yếu tiềm tàng trong việc sử dụng khóa bất đối xứng là khả năng bị tấn công dạng **kẻ tấn công đứng giữa** (man in the middle attack): kẻ tấn công lợi dụng việc phân phối khóa công khai để thay đổi khóa công khai. Sau khi đã giả mạo được khóa công khai, kẻ tấn công đứng ở giữa 2 bên để nhận các gói tin, giải mã rồi lại mã hóa với khóa đúng và gửi đến nơi nhận để tránh bị phát hiện. Dạng tấn công kiểu này có thể phòng ngừa bằng các phương pháp trao đổi khóa an toàn nhằm đảm bảo nhận thức người gửi và toàn vẹn thông tin. Một điều cần lưu ý là khi các chính phủ quan tâm đến dạng tấn công này: họ có thể thuyết phục (hay bắt buộc) nhà cung cấp chứng thực số xác nhận một khóa giả mạo và có thể đọc các thông tin mã hóa.

1. **Các dạng tấn công**
   * + 1. *Modul chung*

Khi một nhóm user sử dụng các khoá công khai Z=(e,n) khác nhau ở thành phần e nhưng giống nhau ở modul đồng dư n. Khi đó, nếu kẻ thù tóm được hai đoạn bản mã mà:

+ Của cùng một bản rõ được mã hoá bởi khoá PK khác nhau (từ hai user khác nhau)

+ Hai thành phần e tương ứng là nguyên tố cùng nhau

thì nó sẽ có cách để giải được bản mã. Cụ thể là nếu kẻ thù biết ,,n,,

Vì ()=1 nên nó có thể tìm được a và b sao cho:

a\* +b\* = 1

Suy ra kẻ thù có thể tìm được X từ

Tóm lại nên tránh sử dụng chung modul đồng dư (common moduls) giữa những user cùng một nhóm làm việc nào đó.

* + - 1. *Sử dụng số mũ hóa e nhỏ*
* Tấn công này xảy ra với điều kiện là giá trị e đã được chọn nhỏ (e mà nhỏ thì thuật toán mã hoá trong truyền tin mật cũng như kiểm định chữ ký sẽ nhanh hơn).
* Nếu kẻ thù có thể tìm được e(e+1)/2 bản mã mà được mã hoá từ những bản rõ phụ thuộc tuyến tính thì hệ thống sẽ bị nguy hiểm. Tuy nhiên nếu các bản rõ này mà không có quan hệ với nhau thì không sao.
* Phòng chống:
  + Ghép thêm vào các bản rõ những xâu nhị phân ngẫu nhiên để đảm bảo cho chúng là không bị phụ thuộc.
  + Sử dụng số mỹ e lớn.

1. *Sử dụng số mũ giải mã d nhỏ*

* Khi sử dụng số mũ giải mã (d) nhỏ, có thể tăng tốc độ giải mã;
* Nếu d nhỏ và gcd(p-1, q-1) (gcd: ước số chung lớn nhất) cũng nhỏ thì d có thể tính được tương đối dễ dàng từ khóa công khai (n, e);
* Phòng chống: Sử dụng số mũ d đủ lớn

1. **Ứng dụng demo**
2. *Cài đặt RSA trong thực tế*

* Do kích thước cặp khóa RSA rất lớn(n cỡ 2048 bit – khoảng 150 số thập phân) việc thực hiện RSA trực tiếp co chi phí tính toán và lưu trữ rất lớn

- Mã hóa c = me mod n

- Giải mã m = cd mod n

- Do m,e và d thường rất lớn nên giá trị mũ me và cd thường rất rất lớn => cần có giải thuật hiệu quả để giảm chi phí tính toán => cài đặt trên máy tính

1. *Cài đặt trong java*

* Ngôn ngữ lập trình java định nghĩa lớp BigInteger cung cấp hầu hết các hàm dựng và các hàm số học cho phép thao tác thuận lợi với số nguyên lớn
* Một số hàm có thể dùng để cài đặt RSA

•Hàm dựng BigInteger(int bitLength, int certainty, Random rnd): sinh số

nguyên tố ngẫu nhiên với số bit cho trước;

• Hàm BigInteger add(BigInteger val): cộng hai số nguyên lớn;

• Hàm BigInteger gcd(BigInteger val): tìm ƯSC lớn nhất của 2 số nguyên lớn;

• Hàm BigInteger mod(BigInteger m): tính modulo (phần dư) của phép chia

nguyên;

• Hàm BigInteger modInverse(BigInteger m): tính modulo nghịch đảo (this-1

mod m);

• BigInteger modPow(BigInteger exponent, BigInteger m):

tính (thisexponent mod m).

1. *Demo ứng dụng*

* AlgorithmRSA:

package rsa;

import java.math.BigInteger;

import java.security.SecureRandom;

public class AlgorithmRSA {

//d: private key, public key

private BigInteger n, d, e;

public BigInteger getN() {

return n;

}

public void setN(BigInteger n) {

this.n = n;

}

public BigInteger getD() {

return d;

}

public void setD(BigInteger d) {

this.d = d;

}

public BigInteger getE() {

return e;

}

public void setE(BigInteger e) {

this.e = e;

}

/\*\*

\* Create an instance that can encrypt using someone elses public key.

\*/

public AlgorithmRSA(BigInteger n1, BigInteger e1) {

n = n1;

e = e1;

}

/\*\*

\* Create an instance that can both encrypt and decrypt.

\*/

public AlgorithmRSA() {

}

public void KeyRSA(int bits){

SecureRandom r = new SecureRandom();//create BigInteger r random

BigInteger p = new BigInteger(bits , 100, r);

BigInteger q = new BigInteger(bits , 100, r);

n = p.multiply(q);

BigInteger m = (p.subtract(BigInteger.ONE)).multiply(q

.subtract(BigInteger.ONE));

boolean check = false;

do {

e = new BigInteger(bits , 50, r);

if (m.gcd(e).equals(BigInteger.ONE) && e.compareTo(m) < 0) {

check = true;

}

} while (!check);

d = e.modInverse(m);

}

// Encrypt the given plaintext message.Use public key decrypt

public synchronized String encrypt(String str) {

return (new BigInteger(str.getBytes())).modPow(e, n).toString();

}

//Encrypt the given plaintext message.Use public key decrypt

public synchronized BigInteger encrypt(BigInteger b) {

return b.modPow(e, n);

}

// Decrypt the given ciphertext message.Use private key decrypt

public synchronized String decrypt(String str) {

return new String((new BigInteger(str)).modPow(d, n).toByteArray());

}

// Decrypt the given ciphertext message.Use private key decrypt

public synchronized BigInteger decrypt(BigInteger b) {

return b.modPow(d, n);

}

}

* Hình ảnh chạy ứng dụng



1. **Kết luận**

Do thời gian có hạn nên chúng em chưa thể đi sâu phân tích các khía cạnh của giải thuật mã hóa khóa bất đối xứng RSA. Tuy nhiên có thể nói, báo cáo đã nói lên được những phần khái quát nhất và đem lại một cái nhìn tổng thể cho người đọc. Tài liệu đã giải thích được cách thức, quá trình sinh khóa, mã hóa, giải mã cùng với ví dụ để giúp người đọc hiểu hơn về giải thuật. Bên cạnh đó, chúng em cũng đưa ra các điểm yếu, các dạng tấn công và phòng chống của giải thuật RSA.

1. **Tài liêu tham khảo**

Trong quá trình làm báo cáo, chúng em đã tham khảo những tài liệu sau:

* Trang web [www.wikipedia.org](http://www.wikipedia.org)
* Trang web [www.google.com](http://www.google.com)
* Quyển “*RSA tấn công và phòng thủ*”, Tác giả: Nguyễn Thành Nhân, NXB Thanh Niên.
* Quyển “*Giáo trình an toàn bảo mật hệ thống thông tin*” , Tác giả: Hoàng Xuân Dậu.