BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ TP. HCM

BÙI HỮU ĐÔNG

THIẾT KẾ VÀ CÀI ĐẶT THƯ VIỆN SỐ LỚN ỨNG DỤNG TRONG MẬT MÃ

LUẬN VĂN THẠC SỸ

Chuyên ngành: Công nghệ thông tin

Mã số ngành: 60480201

TP. HỒ CHÍ MINH, tháng năm 2015

BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ TP. HCM

BÙI HỮU ĐÔNG

THIẾT KẾ VÀ CÀI ĐẶT THƯ VIỆN SỐ LỚN ỨNG DỤNG TRONG MẬT MÃ

LUẬN VĂN THẠC SỸ

Chuyên ngành: Công nghệ thông tin

Mã số ngành: 60480201

CÁN BỘ HƯỚNG DẪN KHOA HỌC: **PGS.TSKH. NGUYỄN XUÂN HUY**

TP. HỒ CHÍ MINH, tháng năm 2015

CÔNG TRÌNH ĐƯỢC HOÀN THÀNH TẠI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ TP. HCM

Cán bộ hướng dẫn khoa học: PGS.TSKH. Nguyễn Xuân Huy
(*Ghi rõ họ, tên, học hàm, học vị và chữ ký*)

Nguyễn Xuân Huy

Luận văn thạc sĩ được bảo vệ tại Trường Đại học Công nghệ TP. HCM ngày … tháng … năm 2015

Thành phần Hội đồng đánh giá luận văn thạc sĩ gồm:
(Ghi rõ họ, tên, học hàm, học vị của Hội đồng chấm bảo vệ luận văn thạc sĩ)

TT	**Họ và tên**	**Chức danh Hội đồng**
1		Chủ tịch
2		Phản biện 1
3		Phản biện 2
4		Ủy viên
5		Ủy viên, Thư ký

Xác nhận của Chủ tịch Hội đồng đánh giá luận sau khi luận văn đã được sửa chữa (nếu có).

Chủ tịch Hội đồng đánh giá LV

TRƯỜNG ĐH CÔNG NGHỆ TP. HCM
PHÒNG QLKH – ĐTSĐH

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập – Tự do – Hạnh phúc

*TP. HCM, ngày tháng năm 20*15

NHIỆM VỤ LUẬN VĂN THẠC SĨ

Họ tên học viên: Bùi Hữu Đông.....Giới tính: Nam .....

Ngày, tháng, năm sinh: 19/02/1983.....Nơi sinh: Hải Dương .....

Chuyên ngành: Công nghệ thông tin.....MSHV: 1341860034 .....

I- Tên đề tài:

THIẾT KẾ VÀ CÀI ĐẶT THƯ VIỆN SỐ LỚN ỨNG DỤNG TRONG MẬT MÃ

II- Nhiệm vụ và nội dung:

- Nghiên cứu về lý thuyết mã hóa, bài toán an toàn thông tin, hệ mã hóa khóa công khai.
- Nghiên cứu về lý thuyệt độ phức tạp của thuật toán. Sơ lược lý thuyết về hàm băm.
- Nghiên cứu về cách thức tổ chức, biểu diễn số nguyên lớn.
- Nghiên cứu về cơ sở toán học xử lý các số nguyên lớn có độ dài hàng trăm thậm chí hàng nghìn chữ số ứng dụng trong mật mã.
- Nghiên cứu các giải thuật toán học áp dụng nhằm tối ưu hóa các phép toán số học.
- Nghiên cứu cách kiểm tra và sinh số nguyên tố từ một số nguyên lớn cho trước.
- Tìm hiểu về cơ chế hoạt động, quá trình mã hóa và giải mã của hệ mật mã RSA.
- Nghiên cứu thực hiện mở rộng hệ mật RSA nhằm tăng cường tính an toàn.
- Lập trình cài đặt lớp số lớn (*BigNumbers*), kiểm thử và ứng dụng *BigNumbers* kết hợp với hàm băm để xây dựng hệ mật mã RSA.

III- Ngày giao nhiệm vụ: 03/04/2015.

IV- Ngày hoàn thành nhiệm vụ: .....

V- Cán bộ hướng dẫn: PGS.TSKH. Nguyễn Xuân Huy .....

CÁN BỘ HƯỚNG DẪN
(*Họ tên và chữ ký*)

KHOA QUẢN LÝ CHUYÊN NGÀNH
(*Họ tên và chữ ký*)

LỜI CAM ĐOAN

Học viên xin cam đoan đây là công trình nghiên cứu, tìm hiểu của riêng mình dưới sự hướng dẫn khoa học của **PGS.TSKH. Nguyễn Xuân Huy**. Các số liệu, kết quả thực nghiệp nêu trong luận văn là trung thực và chưa từng được ai công bố trong bất kỳ công trình nào khác.

Học viên xin cam đoan rằng mọi sự giúp đỡ cho việc thực hiện luận văn này đã được cảm ơn và các thông tin trích dẫn trong luận văn đã được chỉ rõ nguồn gốc.

Học viên thực hiện luận văn
(Ký và ghi rõ họ tên)

Bùi Hữu Đông

LỜI CẢM ƠN

Lời đầu tiên học viên xin chân thành cảm ơn tới các thày, cô giảng viên của trường đã tận tình truyền đạt cho học viên những kiến thức quý báu trong suốt quá trình học tập, nghiên cứu và rèn luyện tại trường.

Đặc biệt, học viên xin bày tỏ lời cảm ơn chân thành và lòng biết ơn sâu sắc nhất tới **PGS. TSKH. Nguyễn Xuân Huy.** Người thày đã chỉ bảo và hướng dẫn tận tình cho học viên trong suốt quá trình nghiên cứu khoa học và thực hiện đề tài luận văn. Bên cạnh kiến thức khoa học, thày còn giúp học viên những kỹ năng trình bày, phong cách làm việc thực tế và kinh nghiệm sống quý báu.

Học viên cũng xin gửi lời cảm ơn chân thành tới **TS. Nguyễn An Khương** - *Khoa KH&KT Máy tính - Đại học Bách Khoa TPHCM* đã giúp đỡ học viên các kiến thức toán học quý giá cho học viên áp dụng vào luận văn.

Ngoài ra, học viên cũng bày tỏ lòng biết ơn tới gia đình, bạn bè, người thân và đồng nghiệp cơ quan đã động viên, khuyến khích tinh thần, giúp đỡ để học viên hoàn thành luận văn này.

TP Hồ Chí Minh, ngày 15 tháng 09 năm 2015

Học viên

Bùi Hữu Đông

TÓM TẮT

Trong quá trình trao đổi thông tin, vấn đề an toàn bảo mật thông tin luôn được đặt lên hàng đầu. Ngoài vấn đề bảo mật thì vấn đề xác thực thông tin cũng rất quan trọng, nó ứng dụng hầu hết các giao dịch thương mại điện tử trên môi trường mạng, đặc biệt như mạng internet như hiện nay.

Hệ mật khóa công khai, cụ thể như RSA có độ phức tạp phá mã với chi phí thời gian dạng hàm mũ, vì RSA được xây dựng dựa vào độ khó của quá trình phân tích một số nguyên lớn thành tích các thừa số nguyên tố và bài toán *logarit rời rạc* trong *modulo* hợp số. Do đó, để tăng độ an toàn của RSA cần tăng độ lớn của các số nguyên tham gia.

Khi độ lớn của các số nguyên càng lớn thì thời gian xử lý của quá trình mã hóa và giải mã cũng tăng lên, các thông tin cần mã hóa ngày càng đa dạng, có khối lượng lớn. Do đó, vấn đề đặt ra là hệ mã hóa cần giảm thời gian xử lý.

Đồng thời, các phương pháp tấn công phá mã ngày càng được cải tiến, thực hiện trên nhiều phương pháp tấn công khác nhau. Do vậy, hệ mật mã càng phải được tăng tính bảo mật và an toàn.

Từ vấn đề an toàn bảo mật nói trên, luận văn này tập trung vào việc xây dựng cấu trúc dữ liệu, cách thức biểu diễn và xử lý số lớn. Nghiên cứu về cơ sở toán học, lý thuyết độ phức tạp tính toán, các phương pháp xác định tính nguyên tố của số nguyên lớn và các giải thuật tối ưu cho từng phép toán số học để xây dựng thư viện số lớn với mục đích tăng tốc độ xử lý ứng dụng khi cài đặt hệ mật mã RSA.

Ngoài ra, nhằm tăng cường tính an toàn, bảo mật cho hệ mật RSA, luận văn cũng nghiên cứu và đề xuất cài đặt mở rộng hệ mật mã RSA sử dụng với lớn hơn hai số nguyên tố p và q ban đầu.

Ngoài các mục Mở đầu, tổng quan, danh mục và kết luận. Nội dung chính của luận văn được trình bày qua ba chương như sau:

- Chương I: Giới thiệu chung và cơ sở toán học của mật mã.
- Chương II: Lớp thư viện tính toán số với nguyên lớn.
- Chương III: Ứng dụng thư viện số lớn cho hệ RSA mở rộng.

ABSTRACT

Nowadays, the Internet provides essential communication tools for billions of people, and it is being increasingly used as a tool for e-commerce. So the security in using Internet becomes a very important issue to deal with. One of the solutions to secure the communications is cryptography.

The RSA public-key cryptosystem has two keys. One key is used for encryption and the other key which is only known to authentic receiver so that he/she can decrypt the cipher message. RSA with its security is based on the intractability of the large integer factorization problem ($n = pq$).

However, when the size of the integers used in the system increases, the processes of encryption and decryption in RSA cyptosystem will become costly as well. Since RSA algorithm is based on arithmetic modulo of large numbers, which requires a very large number of computations, fast implementation of RSA algorithm becomes vitally important for the performance of the cryptosystems.

On the other hand, software solutions are inherently flexible for all kinds of emerging cryptosystems but comparatively slow. Hence it is necessary to develop efficient methods to implement RSA over software platforms. The software implementations of RSA are generally based on 2-prime RSA and 2-prime implementations require two major operations: squaring and multiplication reduction.

In this thesis, we give an overview on number theory, and study data structures and algorithms to effectively implement large integer numbers to build up a library of large numbers. And then we use this library to speed – up the implementation of RSA cryptosystem.

Moreover, in order to increase the safety and security of RSA cryptosystem, this thesis also presents and extension of RSA algorithm for more than two primes instead of using the standard RSA algorithm with only two primes.

This thesis consists of three chapters structured as follows.

- In Chapter 1, we present an introduction to number theory and cryptography.
- Chapter 2 is devoted to build a class library of large integer numbers.
- Chapter 3 is an application of the above class library for multi-prime RSA algorithm.

MỤC LỤC

DANH MỤC CÁC TỪ VIẾT TẮT

Ký hiệu, viết tắt	Ý nghĩa tiếng Anh	Ý nghĩa tiếng Việt
RSA	Ron Rivest, Adi **S**hamir và Len Adleman	Tên của ba nhà khoa học đã đưa ra hệ mật mã khóa công khai. Hệ mật này được viết tắt bởi ba ký tự tên đầu của ba nhà khoa học này.
GCD	Greatest Common Divisors	Ước chung lớn nhất
MD5	Message-Digest algorithm 5	Tên của một hàm băm
SHA1	Secure Hash Algorithm 1	Tên của một hàm băm
$\varphi(n)$	Euler' Function	Phi hàm Euler
MIPS-Years	Million Instructions Per Second	Số các phép tính đã hoàn thành bởi một máy trong thời gian một năm, với tốc độ khoảng 10^6 phép tính trên một giây, ta có MIPS-Years ≈ 31.5×10^{12} phép tính
AKS	Agrawal – Kayal – Saxena	Tên thuật toán kiểm tra xác suất một số nguyên có phải là nguyên tố không?
Euclid		Tên nhà toán học. Thuật toán Euclid cho phép tìm GCD của hai số nguyên cho trước.
NFS	Number Field Sieve	Một thuật toán phân tích thành thừa số nguyên tố của hợp số N
MPQS	Multiple Polynomial Quadratic Sieve	Một thuật toán phân tích thành thừa số nguyên tố của hợp số n

DANH MỤC CÁC KÝ HIỆU TOÁN HỌC

Ký hiệu	Ý nghĩa
$\mathbb{Z}$	Tập các số nguyên. Đó là tập {…,-2, -1, 0, 1, 2, …}.
$[a, b]$	Ký hiệu là số nguyên x thỏa mãn $a \le x \le b$.
$]a, b[$	Ký hiệu là số nguyên x thỏa mãn $a < x < b$.
$A \leftarrow B$	Biểu diễn sự gán giá trị B sang A.
$\lfloor x \rfloor$	Ký hiệu của số nguyên lớn nhất $\le x$.
$\Rightarrow$	Suy ra.
$\Leftrightarrow$	Tương đương.
$\ln(x)$	Logarit tự nhiên của x. Đó là logarit của x với cơ số e ($e \approx 2.71828$).
$\log(x)$	Logarit của x với cơ số 2.
$\sum_{i=1}^{n} a_i$	Ký hiệu là tổng của $a_1 + a_2 + ... + a_n$.
$\prod_{i=1}^{n} a_i$	Ký hiệu là tích của $a_1 a_2 ... a_n$.

DANH MỤC CÁC HÌNH

DANH MỤC CÁC BẢNG

MỞ ĐẦU

1. Lý do chọn đề tài

Trong quá trình trao đổi thông tin, vấn đề an toàn và bảo mật thông tin luôn là được quan tâm hàng đầu. Theo đó mật mã học là một phần vô cùng quan trọng. Trong mật mã học việc bảo mật luôn đi cùng với vấn đề xác thực thông tin, đặc biệt là trong hệ thống mã hóa khóa công khai. Việc xác thực thông tin ngày nay thường sử dụng dưới dạng chữ ký số. Chữ ký số không chỉ ứng dụng trong các ngành công nghệ thông tin, mật mã học mà còn sử dụng trong lĩnh vực ngân hàng, giao dịch thẻ tín dụng, xác thực nộp thuế qua mạng Internet...

Hệ mã hóa khóa công khai như hệ mật RSA (Ron Rivest, Adi **S**hamir và Len Adleman) thực hiện tính toán với các số nguyên lớn có thể dài hàng nghìn chữ số. Độ phức tạp trong việc giải mã của hệ mã RSA tỉ lệ hàm mũ với các số nguyên tham gia vào việc tạo khóa mã hóa và giải mã khóa công khai. Do vậy, để tăng độ an toàn của hệ mã khóa công khai, cần tăng độ lớn của các số nguyên tham gia.

Thuật toán của RSA được xây dựng dựa vào độ khó của bài toán phân tích một số nguyên lớn ra các thừa số nguyên tố, bài toán khai căn và *logarit rời rạc* trong modulo hợp số. Nếu vấn đề trên là khó, tức là không có một thuật toán hiệu quả để giải chúng thì không thể phá mã toàn bộ hệ mật RSA. Khi độ lớn của các số nguyên càng lớn thì thời gian xử lý của quá trình mã hóa và giải mã cũng tăng lên. Các thông tin cần mã hóa ngày càng có khối lượng lớn và đa dạng, do đó vấn đề đặt ra là hệ mật mã cần giảm thiểu thời gian xử lý. Đồng thời, các phương pháp tấn công nhằm phá mã ngày càng được cải tiến và thực hiện trên nhiều phương pháp tấn công khác nhau. Do đó hệ mật mã càng phải được tăng tính bảo mật.

Để giải quyết các vấn đề đã được đặt ra ở trên, trong nội dung nghiên cứu của đề tài này sẽ tập trung vào việc xây dựng cấu trúc dữ liệu và thuật toán tối ưu hóa các phép toán để xây dựng thư viện xử lý số học với các số nguyên lớn, từ đó làm tăng tốc độ xử lý và tăng tính bảo mật của hệ mã hóa mà cụ thể là hệ mật mã RSA.

Từ tính cấp thiết của vấn đề tối ưu hóa nêu trên của hệ mã hóa RSA, đồng thời được sự gợi ý và hướng dẫn của **PGS.TSKH. Nguyễn Xuân Huy**, học viên đã chọn đề tài: **"Thiết kế và cài đặt thư viện số lớn ứng dụng trong mật mã"** cho luận văn thạc sĩ chuyên ngành Công nghệ thông tin.

2. Nội dung chính

Bố cục của của luận văn gồm các nội dung chính sau:

- Nội dung của đề tài là nghiên cứu về lý thuyết mã hóa, cơ sở toán học xử lý các số nguyên lớn có độ dài hàng trăm, hàng nghìn chữ số ứng dụng trong mật mã.
- Nghiên cứu về cách thức tổ chức, biểu diễn số nguyên lớn.
- Nghiên cứu và áp dụng các thuật toán tối ưu để xử lý các phép toán số học.
- Tìm hiểu về cơ chế hoạt động, quá trình mã hóa và giải mã của RSA, kiểm tra tính nguyên tố của một số nguyên theo các thuật toán Miller - Rabin, Fermat...
- Nghiên cứu mở rộng hệ mật mã khóa công khai RSA nhằm tăng tính an toàn.
- Lập trình cài đặt lớp số lớn (*BigNumbers*), kiểm thử và ứng dụng *BigNumbers* kết hợp với hàm băm để xây dựng RSA.

Ngoài phần mở đầu, tổng quan lĩnh vực nghiên cứu và kết luận, bố cục của luận văn gồm ba chương chính như sau:

- *Chương I: Giới thiệu chung và cơ sở toán học của mật mã.*
- *Chương II: Lớp thư viện tính toán số nguyên lớn.*
- *Chương III: Ứng dụng thư viện số lớn cho hệ RSA mở rộng.*

3. Mục đích của đề tài

- Nghiên cứu về cấu trúc dữ liệu để biểu diễn số nguyên lớn.
- Nghiên cứu về cách thức của quá trình tạo khóa, quá trình thực hiện mã hóa và giải mã của hệ mã khóa công khai RSA.
- Nghiên cứu khả năng mở rộng cho hệ mật RSA nhằm tăng cường độ an toàn.
- Tìm hiểu về cơ sở toán học và các thuật toán xử lý số học dùng trong RSA.
- Tìm ra các giải thuật tính toán cần tối ưu hóa và thực hiện giải pháp tối ưu.
- Cài đặt lớp thư viện xử lý số nguyên lớn cùng với các phép toán cụ thể.
- Ứng dụng lớp thư viện xử lý số lớn vào hệ mật RSA kết hợp với phương pháp hàm băm để xác thực thông tin.
- Đánh giá, so sánh kết quả sau khi tối ưu so với ban đầu.

4. Đối tượng nghiên cứu

- Lý thuyết mã hóa và an toàn thông tin, hệ mã hóa khóa công khai RSA.
- Cơ sở lý thuyết số học gồm: phép cộng, trừ, nhân, chia, lũy thừa, so sánh, lý thuyết đồng dư, cấu trúc đại số, các phép toán, tính nguyên tố, ước chung lớn nhất, hàm một chiều, hàm cửa sập... cho các số nguyên lớn ứng dụng trong hệ mật mã RSA.

- Cách thức tổ chức biểu diễn dữ liệu cho các số nguyên lớn.
- Sơ lược về lý thuyết hàm băm.
- Sơ lược lý thuyết về độ phức tạp tính toán.
- Nghiên cứu các thuật toán số học số nguyên nhằm tối ưu hóa các phép tính toán.

5. Phạm vi nghiên cứu

- Đề tài tiếp cận cách thức tổ chức, biểu diễn số nguyên lớn theo phương pháp hướng đối tượng. Thực hiện sử dụng các giải thuật toán học nhằm tối ưu hóa các phép toán số học trên đối tượng số nguyên lớn này.
- Cơ chế hoạt động cụ thể của hệ mật RSA.
- Khả năng mở rộng cho hệ mật RSA.
- Thử nghiệm áp dụng cài đặt cho hệ mật mã RSA nhằm so sánh hiệu năng xử lý của hệ mã trước và sau khi tối ưu.
- Đề tài giới hạn trong phạm vi nghiên cứu các giải pháp nhằm tối ưu hóa tốc độ hệ mật RSA thông qua các giải thuật tính toán số học. Việc ứng dụng rộng rãi trong thực tế cần có điều kiện về thời gian kiểm thử và tùy theo quy mô sử dụng.

6. Ý nghĩa thực tiễn và ý nghĩa khoa học của luận văn

Ý nghĩa thực tiễn

- Mô tả được cách thức tổ chức, biểu diễn đối tượng số nguyên lớn - BigNumber.
- Cài đặt đầy đủ được các phép toán số học trên đối tượng số nguyên lớn cỡ hàng trăm thậm chí hàng ngàn chữ số.
- Xây dựng được ứng dụng kiểm tra thử nghiệm các giải thuật toán học trong hệ mã hóa công khai RSA mở rộng.
- Đánh giá, so sánh hiệu năng trước và sau khi tối ưu của hệ mã RSA.
- Đã vận dụng vào việc xác thực (trust) thông tin trong hệ thống ngân hàng điện tử của Ngân hàng Xây dựng.

Ý nghĩa khoa học

- Trình bày chi tiết về cơ chế hoạt động, độ an toàn bảo mật và khả năng bị tấn công của hệ mã hóa khóa công khai RSA.
- Trình bày sơ lược về hàm băm, ứng dụng trong việc xác thực thông tin.
- Trình bày được cơ sở lý thuyết toán học, cấu trúc đại số, lý thuyết về độ phức tạp thuật toán, các giải thuật toán học dùng trong hệ mật mã công khai RSA.

- Tối ưu hóa các phép xử lý số học với số nguyên lớn (hàng ngàn chữ số). Nhằm giúp tăng tốc, xử lý an toàn của hệ mật mã RSA.
- Mở rộng và chứng minh tính đúng đăn hệ mã hóa RSA khi dùng với *multi-prime*.

7. Tổng quan về lĩnh vực nghiên cứu

Từ những năm 60, lý thuyết tính toán ra đời đã cho một cách thích hợp để quy yêu cầu bí mật hoặc ngẫu nhiên về một yêu cầu có thể định nghĩa được đó là yêu cầu về *độ phức tạp tính toán.* Một giải pháp mật mã là đảm bảo bí mật nếu mọi thuật toán thám mã nếu có, đều phải được thực hiện với độ phức tạp tính toán cực lớn. Cực lớn ở đây muốn nói tới đó là vượt giới hạn khả năng tính toán (kể cả dùng máy tính) mà người thám mã có thể có. Về lý thuyết, có thể xem đó là những độ phức tạp tính toán với tốc độ tăng vượt quá hàm mũ.

Thuật toán mã hóa khóa công khai – RSA được mô tả vào năm 1977 có hai khóa đó là: *Khóa công khai* và *khóa bí mật.* 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. Tính bảo mật và an toàn của thuật toán RSA dựa trên độ khó (độ phức tạp) của bài toán phân tích một số các thừa số nguyên tố, bài toán khai căn và bài toán *logarit rời rạc* trong *modulo* hợp số.

Trong RSA để tính toán ra cặp khóa công khai và khóa cá nhân, ban đầu chúng ta phải có hai số nguyên tố p và q đủ lớn để từ $n = pq$ hoặc giá trị e ta không thể phân tích ngược ra p và q. Khi đó độ an toàn trong việc mã hóa thông tin là rất cao.

Trên thực tế, việc tạo ra các số nguyên tố lớn và mạnh theo tiêu chuẩn an toàn. Tức là các số nguyên tố có độ lớn hàng trăm, hàng ngàn chữ số giúp tăng độ phức tạp tính toán với độ tăng hàm mũ để tăng thời gian phá mã của các phương pháp tấn công.

Khi độ lớn của các số nguyên tố tăng lên vượt quá kiểu dữ liệu số cơ bản đòi hỏi phải có phương pháp tổ chức, cách thức biểu diễn các số nguyên tố lớn này. Đồng thời, trong quá trình thực hiện mã hóa và giải mã của thuật toán RSA đòi hỏi chi phí thời gian thực hiện phải nhỏ và tốc độ xử lý các phép toán số học phải nhanh bởi thông tin cần mã hóa rất đa dạng và có khối lượng lớn.

Để thực hiện các vấn đề nêu trên, trong nội dung của luận văn này sẽ tập trung vào việc nghiên cứu cách thức tổ chức lưu trữ số nguyên lớn, nghiên cứu và xây dựng

một số thuật toán tối ưu hóa nhằm tăng hiệu quả các phép tính toán thực hiện trên các số nguyên lớn.

8. Tổng quan về các đề tài đã nghiên cứu liên quan

Tình hình một số nghiên cứu trong nước liên quan

Đối với đề tài trong nước liên quan tới an toàn, bảo mật, tối ưu hệ mã hóa công khai RSA thì đã được đề cập tới rất nhiều. Dưới đây trích dẫn một số đề tài tiêu biểu.

Stt	Tên đề tài	Tác giả	Nội dung
1	Luận văn thạc sĩ: "Nghiên cứu hệ mật RSA với số mũ giải mã lớn và ứng dụng cho chữ ký số".	Tô Danh Dũng - Học viện Bưu chính viễn thông năm 2013.	Nghiên cứu hệ mật RSA với số mũ giải mã lớn nhằm tăng độ an toàn bào mật với các phương pháp tấn công hiện nay.
2	Luận văn thạc sĩ: "Các thuật toán tối ưu hóa trong bảo mật thông tin".	Nguyễn Ngọc Trung - Đại học Thái nguyên năm 2008.	Xây dựng một số thuật toán tối ưu hóa nhằm tăng hiệu quả các phép tính thực hiện số nguyên.
3	Luận văn thạc sĩ: "Nghiên cứu và phát triển hệ mật mã khóa công khai ứng dụng trong bảo mật dữ liệu và xác thực các giao dịch điện tử".	Trần Đăng Hiên - Đại học công nghệ - Đại học Quốc gia Hà nội – 2010.	Nghiên cứu, tổng hợp và phát triển một số hệ mã hóa công khai và thuật toán kiểm tra và sinh số nguyên tố lớn.
4	Xây dựng chương trình xử lý song song để xác định một số nguyên lớn có phải là nguyên tố hay không?	Nguyễn Thị Hoài Thương - Lớp ĐHCNTT10.	Nghiên cứu phương pháp xử lý song song tăng khả năng tính toán của CPU đáp ứng yêu cầu mã hóa trong lĩnh vực an toàn và bảo mật thông tin.

Tình hình nghiên cứu quốc tế liên quan

Trên thế giới có rất nhiều bài báo, tạp chí khoa học nghiên cứu về vấn đề an toàn, bảo mật thông tin, tính nguyên tố của số lớn và các vấn đề liên quan tới hệ mật mã RSA. Dưới đây trích dẫn một số bài báo tiêu biểu.

Stt	Tên đề tài	Tác giả	Nội dung
1	An efficient implementation of multi-prime RSA on DSP processor.	Anand Krishnamurthy, Yiyan Tang, Cathy Xu and Yuke Wang, 2003.	Nghiên cứu phương thức mới để thực hiện Montgomery reduction nhằm tăng thời gian xử lý lũy thừa theo modulo.
2	On the Security of Multi-prime RSA.	M. Jason Hinek- David R. Cheriton School of Computer Science University of Waterloo Waterloo, Ontario, N2L 3G1, Canada, June 13, 2006.	Nghiên cứu về sự an toàn bảo mật của RSA khi sử dụng nhiều số nguyên tố và các phương pháp tấn công.
3	The RSA Algorithm.	Evgeny Milanov 3 June 2009.	Nghiên cứu về hệ mật RSA.

Các vấn đề còn tồn tại chung của các đề tài đã thực hiện nêu trên

Về cơ bản các đề tài nghiên cứu trong nước nêu trên đều nghiên cứu, đánh giá khá chi tiết về hệ mật mã công khai RSA, các kiến thức cơ sở toán học mà RSA sử dụng, đồng thời đưa ra các giải pháp nhằm tăng tốc, nhằm tối ưu cho quá trình tạo khóa, quá trình mã hóa và giải mã của RSA.

Tuy nhiên, các đề tài trên chưa đưa ra giải thuật tối ưu cụ thể từng phép xử lý số học, chưa phân tích cụ thể các phép toán mà hệ mã hóa RSA sử dụng, đồng thời cũng chưa đưa ra đánh giá hiệu năng thực hiện cho các phép toán học.

Khả năng xử lý số lớn còn ở mức hạn chế, nếu số cực lớn (cỡ hàng ngàn chữ số) thì tốc độ các phép toán số học cũng như quá trình mã hóa và giải mã của RSA còn chậm.

CHƯƠNG 1
GIỚI THIỆU CHUNG VÀ CƠ SỞ TOÁN HỌC CỦA MẬT MÃ

1.1. Giới thiệu chung về mật mã [5]

1.1.1. Sơ lược lịch sử

Từ khi con người biết trao đổi và truyền đưa thông tin cho nhau, nhu cầu sử dụng mật mã của con người đã xuất hiện. Đặc biệt khi các thông tin đó đã được thể hiện dưới hình thức ngôn ngữ, thư từ. Mật mã trước hết là một loại hoạt động thực tiễn, nội dung chính của nó là để giữ bí mật thông tin từ một người gửi A đến một người nhận B.

Vì bản gửi đi thường được chuyển qua các con đường công khai nên người ngoài có thể "lấy trộm" được, nên thay vì gửi bản rõ thì A gửi cho B bản mã mật của bản rõ người ngoài không hiểu được. Nhưng B có thể giải bản mã mật này thành bản rõ để hiểu được là do giữa hai người đã có một thỏa thuận về một chìa khóa chung.

Chìa khóa chung đó được gọi đơn giản là *khóa mật mã*. Tất nhiên để thực hiện được một phép mật mã, còn cần có một thuật toán biến bản rõ, cùng với khóa mật mã, thành bản mã mật, và một thuật toán ngược lại, biến bản mã mật, cùng với khóa mật mã, thành bản rõ. Các thuật toán đó được gọi là tương ứng là thuật toán *lập mật mã* và thuật toán *giải mật mã*.

Các thuật toán này thường không nhất thiết phải giữ bí mật, mà cái cần được giữ tuyệt mật luôn luôn là *khóa mật mã*. Trong thực tiễn, đã có hoạt động bảo mật thì cũng có hoạt động ngược lại là khám phá bí mật từ các bản mật mã "lấy trộm" được, ta thường gọi là *mã thám*, hoạt động này quan trọng không kém gì hoạt động bảo mật. Vì các thuật toán lập mật mã và giải mật mã không nhất thiết là bí mật, nên *mã thám* thường được tập trung vào việc tìm khóa mật mã, do đó cũng có người gọi công việc đó là *phá khóa*.

Trước đây *mật mã* chỉ được hiểu theo nghĩa hẹp, chủ yếu dùng *bảo mật dữ liệu*, người ta quan niệm: *Mật mã học là khoa học nghiên cứu mật mã về tạo và thám mã.*

Vào những thập niên đầu của thế kỷ 20, các bản mật mã trước đây được viết bằng ngôn ngữ thông thường nay được chuyển bằng kỹ thuật số thành các dãy tín hiệu nhị phân, tức các dãy *bit* và các phép biến đổi trên các dãy ký tự được chuyển thành các phép biến đổi trên các dãy *bit*, hay các dãy số, việc thực hiện các phép lập mã, giải mã trở thành việc thực hiện các hàm số số học. Toán học và kỹ thuật tính toán bắt đầu trở

thành công cụ cho việc phát triển khoa học về mật mã. Khái niệm trung tâm của khoa học mật mã là khái niệm bí mật.

Khái niệm *bí mật* thoạt đầu được gắn với khái niệm *ngẫu nhiên,* rồi về sau trong những thập niên gần đây, với khái niệm *độ phức tạp tính toán,* việc sử dụng lý thuyết xác suất và ngẫu nhiên làm cơ sở để nghiên cứu mật mã đã giúp C.Shannon đưa ra khái niệm *bí mật hoàn toàn* của một hệ mật mã từ năm 1948, khởi đầu cho một lý thuyết xác suất về mật mã. Trong thực tiễn làm mật mã, các *dãy bit ngẫu nhiên* được dùng để trộn với bản rõ (dưới dạng một dãy *bit* xác định) thành ra bản mật mã.

Lý thuyết về độ phức tạp tính toán ra đời từ giữa những năm 1960 đã cho một cách thích hợp để quy yêu cầu bí mật hoặc ngẫu nhiên về một yêu cầu có thể định nghĩa được là yêu cầu về *độ phức tạp tính toán.* Bây giờ có thể nói: Một giải pháp mật mã là bảo đảm bí mật, nếu mọi thuật toán thám mã nếu có, đều phải được thực hiện với độ phức tạp tính toán cực lớn! cực lớn là bao nhiêu? là vượt quá giới hạn khả năng tính toán (bao gồm cả máy tính) mà người thám mã có thể có.

Về lý thuyết, có thể xem đó là những độ phức tạp tính toán với tốc độ tăng vượt quá hàm mũ, hoặc thuộc loại *NP-khó*. Năm 1978, Rivest, Shamir và Adleman tìm ra một hệ mã hóa khóa công khai và một sơ đồ *chữ ký điện tử* hoàn toàn có thể ứng dụng trong thực tiễn, tính bảo mật và an toàn của chúng được bảo đảm bằng độ phức tạp của một bài toán số học nổi tiếng là bài toán phân tích số nguyên thành các thừa số nguyên tố.

Sau khi phát minh ra hệ mã RSA, việc nghiên cứu để phát minh ra các hệ mã hóa khóa công khai khác và ứng dụng vào các bài toán khác nhau của an toàn thông tin đã được tiến hành rộng rãi. Lúc này, mật mã được hiểu theo nghĩa rộng, nó là một trong những công cụ hiệu quả bảo đảm an toàn thông tin nói chung: Bảo mật, bảo toàn, xác thực, chống chối cãi.

1.1.2. Một số khái niệm cơ bản về mật mã

- *Bản rõ:* Là nội dung của thông điệp cần gửi đi yêu cầu phải đảm bảo an toàn, nó có thể là file văn bản, file cấu trúc, xâu các *bit*.
- *Mã hóa:* Là quá trình chuyển đổi thông tin từ bản rõ sang bản mã. Trong quá trình này thông tin trong bản rõ được ẩn đi và do đó bất kỳ người nào đọc thông điệp này cũng không hiểu được trừ phi người đó có thể giải mã.
- *Bản mã:* Là kết quả thu được khi mã hóa bản rõ.
- *Giải mã:* Là quá trình xử lý ngược, tiến hành giải mã bản mã để thu lại bản rõ.

Thí dụ 1.1 (*Mã hóa một đoạn văn bản*).

Ta có một đoạn văn bản cần mã hóa là "ABC" với luật mã là tịnh tiến vòng một đối với mã ASCII của mỗi ký tự. Vậy ta có như sau:

- *Bản rõ:* "ABC".
- *Mã hóa*: Biến đổi các ký tự thành các số theo mã ASCII của ký tự đó.
  A → 65, B → 66, C → 67. Tiến hành tịnh tiến một giá trị ta có 66-67-68 tương ứng với ký tự "BCD".
- *Bản mã:* "BCD".
- *Giải mã:* Giải mã bản mã "BCD" ta được "ABC".

1.1.3. Hệ mật mã

Một hệ mật mã là một bộ gồm năm thành phần đó là (*P, C, K, E, D*) với:

- *P* (Plaintext): Tập hợp hữu hạn các bản rõ (không gian bản rõ).
- *C* (Ciphertext): Tập hợp hữu hạn các bản mã có thể (không gian bản mã).
- *K* (Key): Tập hợp các khóa có thể.
- *E* (Encrytion): Tập hợp các quy tắc mã hóa có thể, nói cách khác *E* là một ánh xạ từ *KxP* vào *C* được gọi là *phép lập mã* (mã hóa).
- *D* (Decrytion): Tập hợp các quy tắc giải mã, nói cách khác *D* là một ánh xạ từ *KxC* vào *P*, gọi là *phép giải mã*. Với mỗi $k \in K$ ta định nghĩa e_k sao cho hai ánh xạ $P \to C$ và d_k sao cho $C \to P$ là hai hàm cho bởi: $\forall x \in P,\ e_k(x) = E(K,\ x)$ *và* $\forall y \in C,\ d_k(y) = D(K,\ y)$.

Trong đó e_k *và* d_k được gọi lần lượt là *hàm lập mã* và *hàm giải mã* ứng với khóa mã hóa *K*. Các hàm này phải thoải mãn hệ thức: $\forall x \in P,\ d_k(e_k(x)) = x$.

Nội dung cần mã hóa thể hiện dưới dạng bản rõ (*P*). Người gửi sử dụng qui tắc (*E*) và khóa (*K*) mã hoá bản rõ (*P*), kết quả thu được gọi là *bản mã* $(EK(P) = C)$. Bản mã này được gửi đi trên một đường truyền tới người nhận, sau khi nhận được bản mã (*C*) người nhận sử dụng qui tắc (*D*) và khóa (*K*) giải mã nó để hiểu được nội dung thông điệp gốc. Hệ mật mã hiện đại cần đảm bảo theo các yêu cầu sau:

- *Tính bảo mật* (*Confidentiality*): Đảm bảo dữ liệu được truyền đi một cách an toàn và không bị lộ nếu như ai đó cố tình muốn có được thông điệp gốc ban đầu. Chỉ những người được phép mới có khả năng đọc được nội dung thông tin ban đầu.

- *Tính xác thực* (*Authentication*): Giúp cho người nhận thông điệp xác định được chắc chắn thông điệp mà họ nhận là thông điệp gốc ban đầu. Kẻ giả mạo không thể giả dạng một người khác hay nói cách khác không thể mạo danh để gửi thông điệp. Người nhận có khả năng kiểm tra nguồn gốc thông điệp mà họ nhận được.
- *Tính toàn vẹn* (*Integrity):* Người nhận thông điệp có thể kiểm tra thông điệp không bị thay đổi trong quá trình truyền đi. Kẻ giả mạo không thể có khả năng thay thế dữ liệu ban đầu bằng dữ liệu giả mạo.
- *Tính không thể chối bỏ (Non – repudation)*: Người gửi, người nhận không thể chối bỏ sau khi đã gửi hoặc nhận thông điệp.

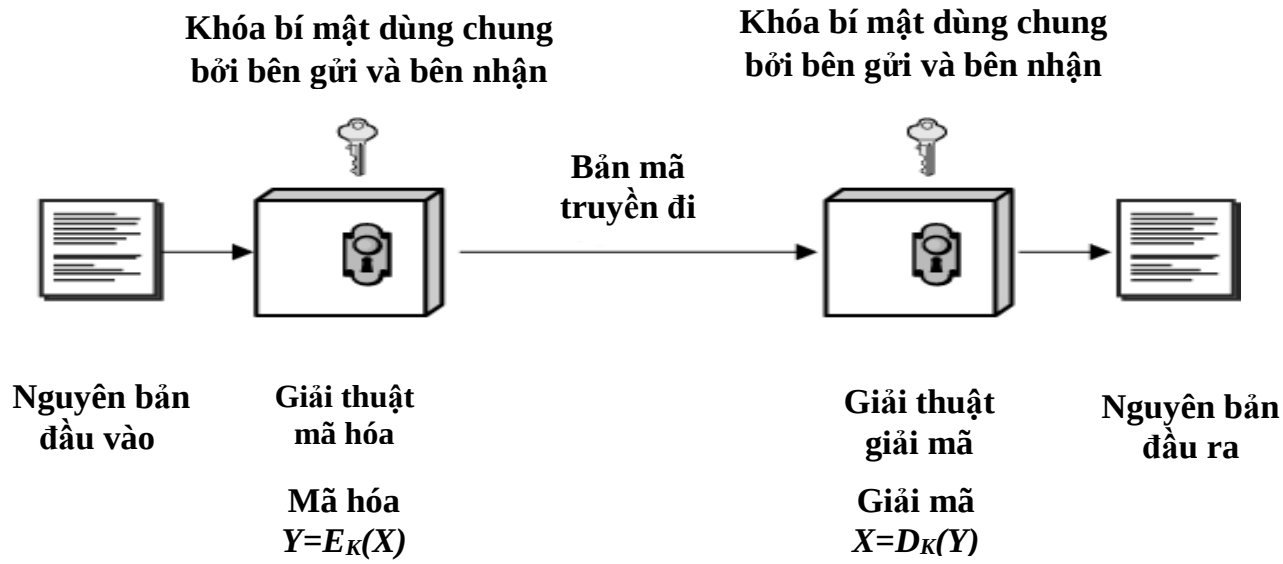
1.1.4. Phân loại các hệ mật mã

Có hai mã hóa phổ biến là mã hóa đối xứng và mã hóa bất đối xứng

1.1.4.1. Mã hóa đối xứng

Là phương thức mã hóa mà trong đó cả người gửi và người nhận đều sử dụng chung một khóa bí mật để mã hóa và giải mã thông điệp. Do đó, khi bị mất khóa bí mật này thì tính bảo mật của hệ mã bị phá vỡ.

Ban đầu, bản rõ được người gửi (*Bob*) mã hóa với khóa *K*. Sau đó bản mã được gửi tới người nhận (*Alice*). Khi nhận bản mã, *Alice* sử dụng khóa *K* giải mã. Do đó, nếu một người khác có được khóa *K* thì hệ thống mã hóa này sẽ bị tấn công.



Hình 1.1. Sơ đồ hoạt động của mã hóa khóa đối xứng

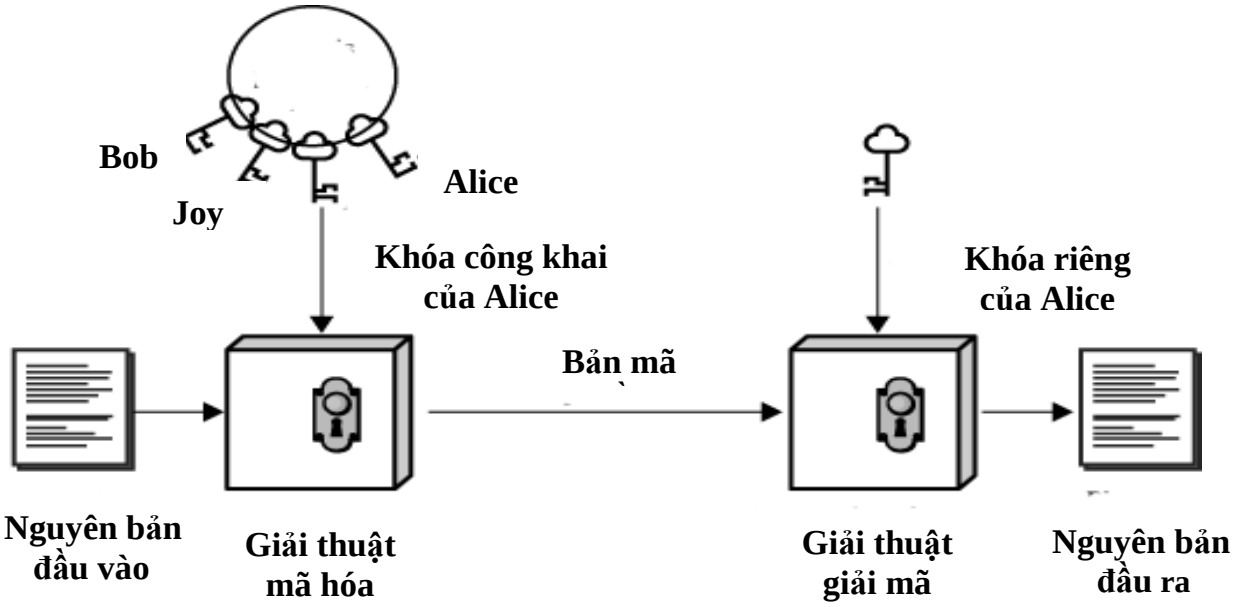
Các hệ mật mã hiện nay như: DES, 3DES-Triple, AES được xây dựng trên phương pháp mã hóa khóa đối xứng.

1.1.4.2. Mã hóa bất đối xứng

Hệ thống mã hóa bất đối xứng hay còn gọi là mã hóa với khóa công khai. Hệ mã này được áp dụng các kết quả của toán học đã khắc phục được các hạn chế của các phương pháp mã hóa khóa đối xứng. Phương pháp mã hóa bất đối xứng sử dụng hai loại khóa trong cùng một cặp khóa: Khóa công khai (public key) được công bố rộng rãi và sử dụng để mã hóa các thông điệp, khóa riêng (private key) chỉ do chủ thể nắm giữ và được sử dụng để giải mã thông điệp đã được mã hóa bằng khóa công khai.

Lý thuyết toán học dựa trên cơ sở khai thác những ánh xạ f mà việc thực hiện ánh xạ ngược f^{-1} rất khó so với việc thực hiện ánh xạ f sử dụng trong các phương pháp mã hóa này. Thực hiện ánh xạ ngược f^{-1} chỉ tiến hành được khi biết khóa riêng.

Khi thực hiện mã hóa bất đối xứng, *Bob* sử dụng khóa công khai do *Alice* tạo để mã hóa thông điệp và gửi cho *Alice*. Do biết khóa riêng nên *Alice* giải mã được thông điệp mà *Bob* đã mã hóa. Trong trường hợp bản mã bị người thứ ba có được, nếu chỉ kết hợp với thông tin về khóa công khai đã được công bố, rất khó giải mã được bản mã này trong khoảng thời gian chấp nhận được do không nắm được khóa riêng của *Alice*.

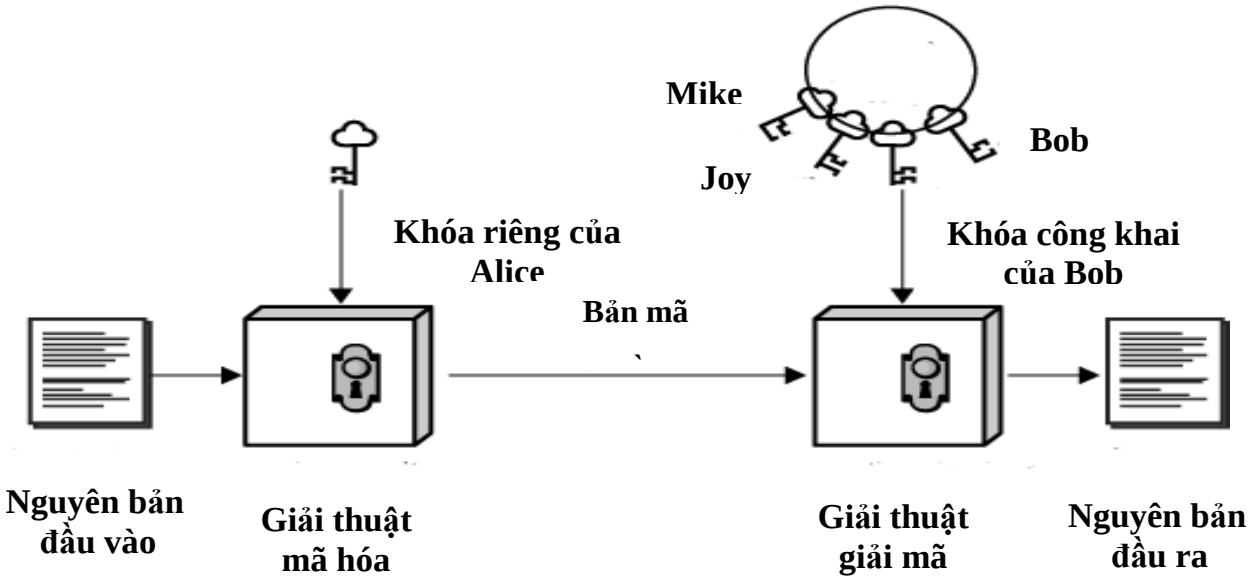


Hình 1.2. Sơ đồ hoạt động của mã hóa khóa bất đối xứng

Khóa công khai và khóa riêng có quan hệ toán học với nhau theo nghĩa từ khóa riêng có thể tính toán để suy ra được khóa công khai, nhưng để từ khóa công khai suy ra khóa riêng sẽ rất phức tạp vì số lượng phép tính toán là rất lớn dẫn đến thời gian thực hiện để giải mã là không khả thi khi chiều dài của khóa đủ lớn.

Đây cũng là mấu chốt của vấn đề bảo mật và tấn công trong các hệ mã khóa công khai. Đề tài này sẽ đề cập đến vấn đề an toàn của hệ mã công khai. Nghiên cứu đưa ra các giải pháp hỗ trợ làm tăng tính an toàn của các hệ mã này bằng cách áp dụng cách thức biểu diễn và các thuật toán xử lý nhanh với số lớn. Từ đó có thể tăng chiều dài của khóa mà vẫn đảm bảo yếu tố thời gian mã hóa và giải mã chấp nhận được. Ngoài sử dụng khóa công khai và khóa bí mật để thực hiện mã hóa. Hiện nay, sử dụng phổ biến trong việc xác thực thông tin.

Theo mô hình dưới đây, *Alice* dùng mã khóa bí mật của mình để mã hóa thông tin. Sau đó chuyển bản mã cho *Bob*. *Bob* dùng khóa công khai của *Alice* để xác thực thông tin xem có phải *Alice* là người gửi tin hay không. Vấn đề xác thực này được sử dụng rất phổ biến hiện nay khi các ứng dụng giao dịch thương mại điện tử, ngân hàng điện tử, khai báo nộp thuế...trên Internet.



Hình 1.3. Sơ đồ hoạt động xác thực thông tin khóa bất đối xứng

1.1.5. Bài toán an toàn thông tin

Có nhiều bài toán khác nhau về an toàn thông tin tùy theo những trường hợp khác nhau, nhưng chung lại có một số bài toán mà ta thường gặp trong thực tế như sau:

- *Bảo mật:* Giữ thông tin được bí mật đối với tất cả mọi người.
- *Toàn vẹn thông tin:* Bảo đảm thông tin không bị thay đổi hay xuyên tạc bởi những kẻ không có thẩm quyền hoặc bằng những phương tiện không được phép.
- *Nhận thực một thực thể:* Xác nhận danh tính của một thực thể chẳng hạn như một người, một máy tính cuối trong mạng...

- *Nhận thực một thông báo:* Xác nhận nguồn gốc của một thông báo gửi đến.
- *Chữ ký:* Một cách để gắn kết một thông tin với một thực thể, thường dùng trong bài toán nhận thực một thông báo cũng như trong nhiều bài toán nhận thực khác.
- *Ủy quyền:* Chuyển thực thể khác quyền được đại diện hoặc được làm.
- *Cấp chứng chỉ:* Cấp một sự xác nhận thông tin bởi một thực thể được tín nhiệm.
- *Báo nhận:* Xác nhận một thông báo đã được nhận hay một việc đã được thực hiện.
- *Làm chứng:* Kiểm thử tồn tại một thực thể khác với người chủ sở hữu thông tin.
- *Không chối bỏ được:* Ngăn ngừa việc chối bỏ trách nhiệm đối với một cam kết.
- *Ẩn danh:* Che giấu danh tính của một thực thể tham gia trong một tiến trình.
- *Thu hồi:* Rút lại một giấy chứng chỉ hay ủy quyền đã cấp.

Các giải pháp cho các bài toán an toàn thông tin như trên đều trên cơ sở các phương pháp mã hóa, đặc biệt là phương pháp mã hóa khóa công khai.

1.1.6. Thám mã và tính an toàn của hệ mật mã

1.1.6.1. Vấn đề về thám mã

Có thể quy ước xem bài toán thám mã cơ bản là bài toán tìm khóa mã hóa *K* (hay khóa giải mã *K''*). Để giải bài toán đó, giả thiết người thám mã biết thông tin về sơ đồ hệ mã hóa được dùng, kể cả các phép lập mã và giải mã tổng quát *E* và *D*. ngoài ra, người thám mã có thể biết thêm một số thông tin khác. Ta có thể phân loại bài toán thám mã thành các bài toán cụ thể như sau:

- Bài toán thám mã *chỉ biết bản mã*: Là bài toán phổ biến nhất, khi người thám mã chỉ biết một bản mã hóa *Y*.
- Bài toán thám mã khi *biết cả bản rõ*: Người thám mã biết một bản mã hóa *Y* cùng với bản rõ tương ứng *X*.
- Bài toán thám mã khi *có bản rõ được chọn*: Người thám mã có thể chọn một bản rõ *X* và biết bản mã hóa tương ứng *Y*. Điều này có thể xảy ra khi người thám mã chiếm được (tạm thời) máy lập mã.
- Bài toán thám mã khi *có bản mã được chọn*: Người thám mã có thể chọn một bản mã hóa *Y* và biết bản rõ tương ứng *X*. Điều này có thể xảy ra khi người thám mã chiếm được tạm thời máy giải mã.

1.1.6.2. Tính an toàn của một hệ mật mã

Tính an toàn của một hệ thống mã hóa phụ thuộc vào độ khó của bài toán thám mã khi sử dụng hệ mã hóa đó. Sau đây xin giới thiệu vài cách hiểu thông dụng nhất:

- *An toàn vô điều kiện*: Giả thiết người thám mã có được thông tin về bản mã, nếu những hiểu biết về bản mã không thu hẹp được độ bất định về bản rõ đối với người thám mã, thì hệ mã hóa là an toàn vô điều kiện hay *bí mật hoàn toàn.*
- *An toàn được chứng minh*: Một hệ thống mã hóa có độ an toàn được chứng minh nếu có thể chứng minh được là bài toán thám mã đối với hệ thống đó *khó tương đương* với một bài toán khó đã biết, thí dụ bài toán phân tích một số nguyên thành tích các thừa số nguyên tố, bài toán tìm *logarit rời rạc* theo một *modulo* nguyên tố.
- *An toàn tính toán:* Nếu mọi phương pháp thám mã đều đòi hỏi một nguồn năng lực tính toán vượt mọi khả năng tính toán của một kẻ thù giả định.

1.1.7. Hệ mã hóa khóa công khai

1.1.7.1. Giới thiệu

Hệ mã hóa khóa công khai là một dạng *mật mã hóa* cho phép người sử dụng trao đổi các thông tin mật mà không cần phải trao đổi các khóa chung bí mật trước đó. Điều này được thực hiện bằng cách sử dụng một cặp khóa có mối quan hệ toán học với nhau là khóa công khai và khóa bí mật. *Nó* có thể sử dụng với mục đích như:

- *Mã hóa:* Giữ bí mật thông tin và chỉ có người có khóa bí mật mới giải mã được.
- *Xác thực:* Cho phép kiểm tra một văn bản có phải đã được tạo với một khóa bí mật nào đó hay không (*chữ ký số*).
- *Trao đổi khóa:* Cho phép chia sẻ khóa phiên trong mã hóa đối xứng.

Hiện nay, có một số hệ mật mã khóa công khai. Tuy nhiên, trong phạm vi của luận văn này chỉ nghiên cứu chi tiết về hệ mật RSA.

1.1.7.2. Hệ mật mã khóa công khai RSA

Thuật toán RSA có hai khóa: Khóa công khai và khóa bí mật. 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.

RSA dựa vào độ khó của bài toán phân tích một số nguyên lớn ra các thừa số nguyên tố, *logarit rời rạc* trong *modulo* hợp số (Bài toán RSA).

Về mặt tổng quát RSA là một hệ mã hóa theo khối, trong đó bản rõ *m* và bản mã *c* là các số nguyên trong khoảng 0 đến 2^i với *i* là số *bit* của khối. Độ lớn của *i* thường dùng là 1024 *bit.*

1.1.7.3. Cơ chế hoạt động của RSA

- *Quá trình tạo khóa*

Algorithm 1.1 (*Key generation for RSA*).

Để sử dụng hệ mật mã RSA, đầu tiên phải tạo ra một cặp khóa công khai và khóa bí mật, việc tạo cặp khóa được thực hiện theo các bước dưới đây:

1. Chọn hai số nguyên tố lớn p và q sao cho $p \neq q$, cùng kích thước và thông điệp m cùng tích pq thỏa $m < 2^{i-1} < pq < 2^i$.
2. Tính $n = pq$ và $\varphi(n) = (p-1)(q-1)$;
3. Chọn số tự nhiên e, sao cho $1 < e < \varphi(n)$ và $gcd(e, \varphi(n)) = 1$;
4. Tính số nguyên duy nhất d, $1 < d < \varphi(n)$ và $de = 1 \pmod{\varphi(n)}$;
5. Cặp khóa công khai gồm (e, n) và khóa bí mật là (d, n).

Theo các Bước của *Algorithm 1.1* thì số nguyên e và d được gọi là số mũ mã hóa và số mũ giải mã còn n được gọi là *modulo* của RSA.

- *Quá trình mã hóa và giải mã*

Algorithm 1.2 (*RSA encryption and decryption*).

Giả sử *Bob* chuyển một thông điệp M cho *Alice, Alice* có thể giả mã M.

1. Quá trình mã hóa: *Bob* sẽ thực hiện các bước sau:
   1.1. *Bob* có được bộ khóa công khai gồm (e, n) của *Alice*;
   1.2. Chuyển M thành số nguyên m, $m \in [0, n-1]$;
   1.3. Tính giá trị bản mã bằng: $c = m^e \pmod{n}$;
   1.4. *Bob* gửi bản mã hóa C cho *Alice*;
2. Quá trình giải mã: *Alice* khôi phục bản rõ m từ C. *Alice* sử dụng khóa bí mật d để khôi phục, cụ thể: $m = c^d \pmod{n}$.

Chứng minh Algorithm 1.2. Ta cần chứng minh $c^d \equiv m \pmod{n}$.

Từ $c = m^e \pmod{n} \rightarrow c^d \equiv m^{ed} \pmod{n}$. Do $n = pq$ nên ta cần tính $m^{ed} \pmod{p}$ và $m^{ed} \pmod{q}$. Nếu $p \mid m$ thì $m^{ed} \pmod{p} = 0 = m \pmod{p}$.

Ngược lại, theo $ed \equiv 1 \pmod{\varphi(n)}$ thì tồn tại một số nguyên *k* sao cho $ed = 1 + k\varphi(n)$.

Do đó, ta có như sau

$m^{ed} \pmod{p} \equiv m^{1+k\varphi(n)}$

$$\equiv mm^{k\varphi(p)\varphi(q)} \pmod{p}$$

$$\equiv m(m^{\varphi(p)})^{k\varphi(q)} \pmod{p}$$

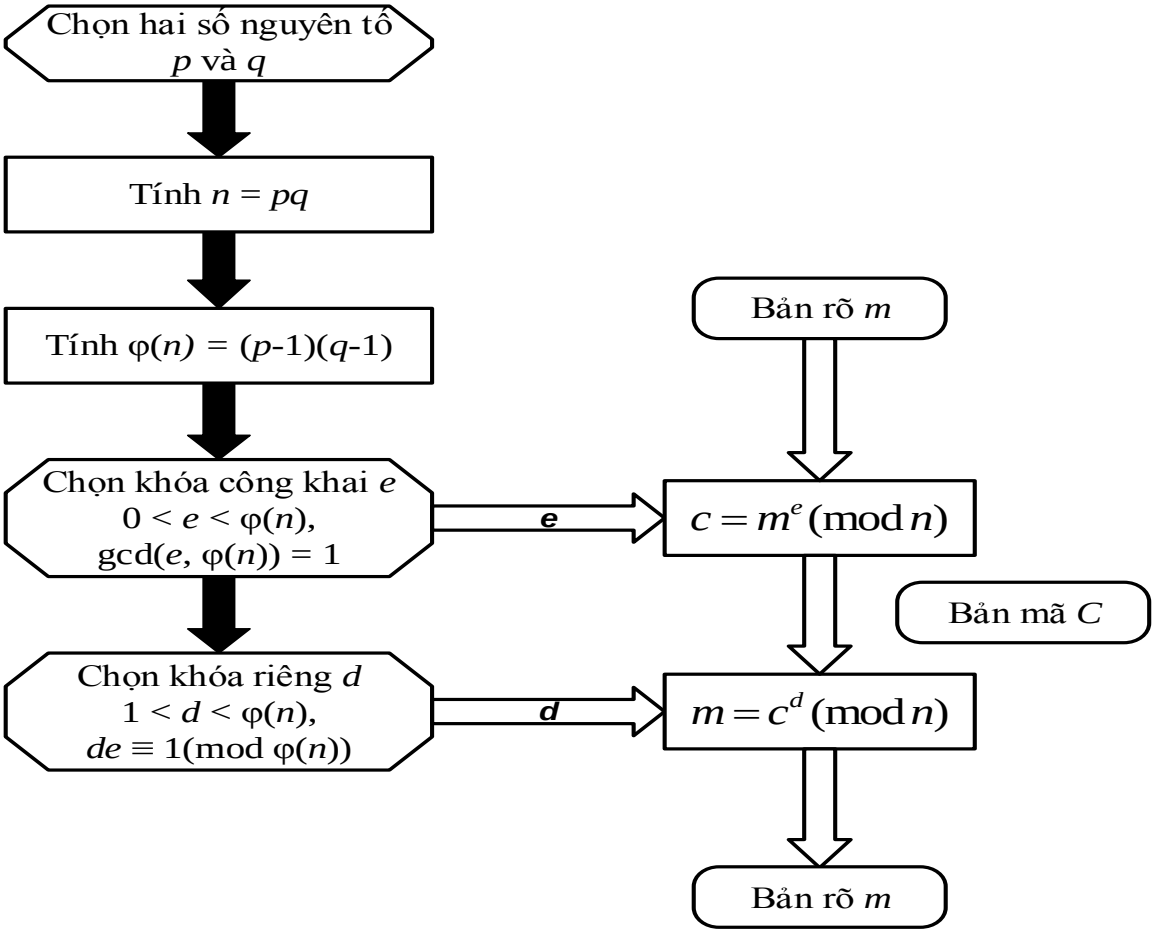
$$\equiv m(1)^{k\varphi(q)} \pmod{p} \text{ (}\textit{Theo Định lý Fermat về số nguyên tố}\text{)}$$

$$\equiv m \pmod{p}.$$

Tương tự ta có: $m^{ed} \equiv m \pmod{q}$. Theo định lý về phần dư Trung Hoa, ta có:

$$m^{ed} \equiv m \pmod{pq} \equiv m \pmod{n} \Rightarrow c^d \equiv m.$$

Tổng quát lại cơ chế hoạt động của hệ mật mã RSA như sơ đồ dưới đây



Hình 1.4. Sơ đồ cơ chế hoạt động của RSA chuẩn.

Thí dụ 1.2 (*Quá trình tạo khóa, mã hóa và giả mã theo RSA*).

Tạo khóa công khai và khóa bí mật: Alice chọn hai số nguyên tố p = 2357 và q = 2551 và tính $n = pq$ = 6012707 và $\varphi(n) = (p-1)(q-1)$ = 6007800.

Chọn e = 3674911 và sử dụng giải thuật Euclid mở rộng theo *Algorithm 1.11* để tìm được d = 422191 thỏa mãn $de \equiv 1 \pmod{\varphi(n)}$.

Ta có cặp khóa với khóa công khai gồm (e = 3674911, n = 6012707) và khóa bí mật gồm (d = 422191, n = 6012707).

Quá trình mã hóa: Để thực hiện mã hóa một bản tin m = 5234673, *Bob* sử dụng biểu thức $c \equiv m^e \pmod n$. = $5234673^{3674911} \pmod{6072707}$ = 3650502 và gửi cho *Alice*.

Quá trình giải mã: Để thực hiện giải mã bản mã c mà *Bob* gửi đến *Alice* thực hiện tính $m \equiv c^d \pmod n$. = $3650502^{422191} \pmod{6072707}$ = 5234673.

1.1.7.4. Độ an toàn của RSA

Độ an toàn của hệ thống RSA dựa trên hai vấn đề của toán học: Bài toán phân tích ra thừa số nguyên tố các số nguyên lớn và bài toán RSA.

Theo [1] bảng chi phí thời gian cần thiết để phân tích những hệ mật mã RSA có kích cỡ số *modulo n* khác nhau, bằng những thuật toán phân tích tốt nhất hiện nay.

Ở đây chi phí tính toán được tính bằng đơn vị MIPS-Years (đó là số các phép tính đã hoàn thành bởi một máy trong thời gian một năm, với tốc độ khoảng 10^6 phép tính trên một giây, ta có 1 MIPS-Years ≈ 31.5×10^{12} phép tính).

Bảng 1.1. Chi phí thời gian cần thiết để phân tích số nguyên n

Số chữ số thập phân	**Số phép tính bit**	**Chi phí thời gian**
50	$1{,}4.10^{10}$	3,9 giờ
75	9.10^{12}	104 ngày
100	$2{,}3.10^{15}$	74 năm
200	$1{,}2.10^{23}$	$3{,}8.10^9$ năm
300	$1{,}5.10^{29}$	$4{,}9.10^{15}$ năm
500	$1{,}3.10^{39}$	$4{,}2.10^{25}$ năm

Vào ngày 22/8/1999 một nhóm các nhà nghiên cứu đã hoàn thành việc phân tích số n dùng trong hệ RSA-155 có 155 chữ số thập phân (512-*bit*) bằng thuật toán General Number Field Sieve (GNFS). Sau một thời gian 7 tháng trên mạng máy tính có 300 workstation yêu cầu khoảng 8000 MIPS-Years.

Việc phân tích số nguyên n thành các thừa số nguyên tố p, q nhằm mục đích bẻ gãy hệ mật mã RSA là điều khó có thể tính toán nổi, nếu như trong quá trình thiết kế hệ

RSA ta chọn số nguyên *n* lớn. Tuy nhiên, độ dài của số nguyên *n* là lớn hay nhỏ còn phụ thuộc vào mối liên hệ giữa tốc độ giải mã và độ an toàn của hệ RSA. Số *n* sử dụng cho *modulo* RSA hiện nay phải có ít nhất khoảng 232 chữ số thập phân (768-*bit*) sẽ cho ta một độ an toàn đủ để chống lại những tấn công sử dụng các kỹ thuật phân tích hiện nay. Trong thực tế, nếu trong quá trình thiết kế sử dụng số *n* có kích cỡ khoảng 1024 *bit* là rất an toàn, có thể chống lại các kỹ thuật phân tích hiện tại và các kỹ thuật khác phát triển trong tương lai.

1.1.7.5. Hạn chế của hệ mật mã RSA

Hệ mật mã RSA có độ an toàn cao khi số nguyên $n = pq$ có kích cỡ càng lớn khoảng trên 1024 *bit,* có thể chống lại các kỹ thuật tấn công hiện nay. Tuy nhiên, hệ mật RSA vẫn tồn tại những hạn chế sau đây:

- *Tốc độ xử lý:* RSA chủ yếu dùng các phép nhân chậm hơn nhiều so với giải thuật đối xứng, không thích hợp với mã hóa với dữ liệu lớn, thường dùng trao đổi khóa bí mật hoặc dùng kết hợp với giá trị băm để dùng để xác thực thông tin.
- *Tính xác thực của khóa công khai*: Ai cũng có thể tạo ra một khóa công bố đó là khóa của người khác. Việc giả mạo chưa bị phát hiện có thể đọc được nội dung các thông báo gửi cho người kia. Cần đảm bảo những người đăng ký khóa là đáng tin.

1.1.7.6. Các phương pháp tấn công hệ mật mã khóa công khai RSA

Từ khi được công bố, RSA đã được phân tích tính an toàn bởi nhiều nhà khoa học. Đã có một số cuộc tấn công lên hệ mật RSA xong chúng chủ yếu là minh họa cho việc sử dụng RSA không đúng cách. Vấn đề thám mã đối với RSA hiện nay đang được các nhà khoa học tập trung vào các kẽ hở của RSA.

- *Brute-force attack:* Phương pháp tấn công bằng cách thử tất cả những khóa có thể có. Đây là phương pháp tấn công thô sơ nhất và cũng khó khăn nhất. Theo lý thuyết, tất cả các thuật toán mã hóa hiện đại đều có thể bị đánh bại bởi brute-force nhưng trong thực tiễn việc này chỉ có thể thực hiện được trong thời gian hàng triệu, thậm chí hàng tỉ năm. Vì thế có thể coi một thuật toán mã hóa là an toàn nếu như không còn cách nào khác để tấn công nó dễ hơn là brute-force.
- *Dùng chung modulo n:* Giả sử bên gửi và bên nhận cùng sử dụng cùng một *modulo* chung *n*. Chẳng hạn *Alice* sử dụng khóa công khai (*n, e*) và giữ khóa bí mật d_1, trong khi *Bob* sử dụng khóa công khai (*n, a*) và giữ khóa bí mật d_2. *Mike* là người thứ ba sử dụng khóa công khai của cả *Alice* và *Bob* để gửi văn bản cần bảo mật *x* đến cả *Alice* và *Bob*.

Lúc này tại phía *Alice*, bản mã sẽ có dạng $y_1 = x^e \pmod n$ và ở phía *Bob* sẽ có dạng $y_2 = x^a \pmod n$. Lúc này người thám mã sẽ dùng các thông tin về n, e, a, y_1, y_2 để tìm ra bản mã *x* như sau:

- Tính $c = e^{-1} \pmod a$.
- Tính $h = (ce^{-1})/a$ và $x = y_1^c (y_2^h)^{-1} \pmod n$.

Việc tính toán này có thể tìm được bản rõ bởi vì ta có

$$y_1^c (y_2^h)^{-1} \pmod n = x^{ce} (x^{a(ce^{-1})/a})^{-1} \pmod n = x^{ce} (x^{ce^{-1}})^{-1} \pmod n = x.$$

- *Dùng số mũ lập mã e bé:* Trong quá trình sinh khóa RSA, để rút ngắn thời gian thực hiện, người dùng thường chọn số mũ *e* bé, giả sử *e* = 3. Tuy nhiên nếu các bên tham gia cùng chọn số mũ *e* bé giống nhau thì sẽ có nguy cơ bị thám mã tấn công. Cụ thể như sau:

Giả sử có ba người tham gia chọn ba khóa công khai $(e, n_1), (e, n_2), (e, n_3)$ với cùng số mũ *e* = 3. Vậy *Alice* sẽ gửi bản mã có dạng $c_i \equiv m^3 \pmod{n_i}$ với *i* = 1, 2, 3 tương ứng với ba người nhận. Bằng các cách thức nghe lén trên kênh truyền, kẻ thám mã sẽ có được c_1, c_2, c_3 và tiến hành tìm *x* với $0 \le x \le n_1, n_2, n_3$, bằng hệ phương trình sau

$$\begin{cases} x \equiv c_1 \pmod{n_1} \\ x \equiv c_2 \pmod{n_2} \\ x \equiv c_3 \pmod{n_3}. \end{cases}$$

Vì $x \le n_i$ nên $x^3 \le n_1 n_2 n_3$, theo định lý phần dư Trung Hoa ta có $x = m^3$. Đây là bài toán số học có thể giải được bằng cách tìm căn bậc 3 của *x* để được bản rõ.

Tương tự ở trên, trong quá trình sinh khóa, để cải thiện tốc độ của thuật toán, *d* được chọn là một số nhỏ. Tuy nhiên nếu $\gcd(p-1, q-1)$ là một số nhỏ thì có thể tính toán ra số *d* từ khóa công khai (n, e).

Điều này không thực hiện được khi *d* có độ dài bằng với *n*, do đó để ngăn chặn điểm yếu này của RSA, ta thường chọn *d* có độ dài bằng với *n*.

Do đó, RSA được xem là không đảm bảo an toàn nếu ta dùng các khóa có số *e* hay *d* là các số nguyên tố bé mặc dù điều đó giúp cho thuật toán làm việc nhanh hơn. Vì vậy, khi sử dụng hệ mật RSA, số mũ *e* và *d* luôn luôn được chọn là những số nguyên tố lớn để đảm bảo độ an toàn cho hệ mật.

- *Tấn công dựa vào cách lặp phép mã:* Ta có hàm $c \equiv x^e \pmod n$ có tính chất hoán vị trong tập $\mathbb{Z}_n$. Do đó, việc lặp phép mã thực hiện như sau.

$$c_0 \equiv c,\ c_1 \equiv c^e \pmod n,\ c_2 \equiv c^{e^2} \pmod n, \ldots \equiv c^{e^i} \pmod n, \ldots$$

Sẽ tồn tại $k \geq 1$ sao cho $c_k \equiv c^{e^k} \pmod x = c.$ Vậy người thám mã sẽ sử dụng *c* và thực hiện phép lặp mã liên tục cho tới khi $c_k = c$, khi đó số hạng trước c_{k-1} chính là bản mã cần tìm. Tuy nhiên, thông thường số *k* sẽ là một số rất lớn vào khoảng *n*! và yêu cầu thực hiện rất nhiều phép lặp mã trong khi *n* thường có độ dài khoảng 200 chữ số. Vì vậy RSA vẫn tỏ ra an toàn đối phương pháp lặp phép mã.

- *Tấn công dựa trên thời gian***:** Năm 1995, Paul Kocher mô tả một dạng tấn công mới lên RSA. Khi kẻ tấn công nắm đủ thông tin về phần cứng thực hiện mã hóa và xác định được thời gian giải mã đối với một số bản mã lựa chọn thì có thể nhanh chóng tìm ra khóa *d*. Dạng tấn công này có thể áp dụng đối với hệ thống chữ ký điện tử sử dụng RSA.
- *Phương pháp sử dụng φ(n):* Giả sử người tấn công biết được giá trị $\varphi(n)$. Khi đó việc xác định *p* và *q* được đưa về việc giải hai phương trình sau: $n = pq$. Thay $q = n/p$ được phương trình bậc hai: $\varphi(n) = (p-1)(q-1) \Leftrightarrow p^2 - (n - \varphi(n) + 1)p + n = 0.$

Với *p, q* chính là hai nghiệm của phương trình bậc hai này. Vấn đề phát hiện được giá trị $\varphi(n)$ còn khó hơn việc xác định hai thừa số nguyên tố của *n*.

Từ cơ sở toán học của RSA, hầu hết các phương pháp tấn công đều chưa thực sự hiệu quả. Các cố gắng bẻ khóa RSA thành công được công bố đều phải dựa trên thực hiện trong khoảng thời gian dài.

Tuy nhiên, để đảm bảo tính an toàn và tăng tốc độ khả năng xử lý của RSA với số nguyên lớn là điều hết sức cấp thiết đặt ra.

1.1.8. Mở rộng hệ mã hóa RSA

1.1.8.1. Mở rộng sinh khóa cho hệ mã hóa RSA với lớn hơn hai số nguyên tố

Để tăng độ an toàn cho hệ mật RSA luận văn nghiên cứu thực hiện mở rộng nhiều hơn hai số nguyên tố (*multi–prime RSA*) so với *RSA chuẩn* chỉ có hai số.

Algorithm 1.3 (*Key generation for multi-prime RSA - extend of the standard RSA*).

Mở rộng RSA bằng việc sử dụng số lượng số nguyên tố *r*, sao cho $r > 2$. Quá trình sinh khóa khi dùng *multi-prime RSA* được thực hiện như sau:

1. Chọn ngẫu nhiên r số nguyên tố $p_1 p_2 ... p_r$. Các p_i độc lập, cùng kích thước với nhau và thông điệp m cùng tích các số nguyên tố p_i thỏa $m < 2^{i-1} < p_1 p_2 ... p_i < 2^i$;
2. Tính $n = \prod_{i=1}^{r}(p_i)$ và $\varphi(n) = \prod_{i=1}^{r}(p_i - 1)$;
3. Chọn số e, sao cho $1 < e < \varphi(n)$ và $\gcd(e, \varphi(n)) = 1$;
4. Tính số d *với* $1 < d < \varphi(n)$ sao cho $de \equiv 1 (\mathrm{mod}\ \varphi(n))$;
5. Cặp khóa công khai gồm (e, n) và khóa bí mật là (d, n).

Theo các bước của *Algorithm 1.3* thì số nguyên *e* và *d* được gọi là số mũ mã hóa và số mũ giải mã còn *n* được gọi là *modulo* của *multi-prime* RSA.

1.1.8.2. Quá trình mã hóa và giải mã với multi-prime RSA

Quá trình mã hóa và giải mã của RSA mở rộng cũng như chứng minh tính đúng đắn với *r* số nguyên tố hoàn toàn tương tự với các Bước đã mô tả ở *Algorithm 1.2*.

Thí dụ 1.3 (*Multi-prime RSA với ba số nguyên tố độc lập* p_1, p_2 *và* p_3).

- *Tạo khóa cặp khóa công khai và bí mật*

Alice chọn ba số nguyên tố $p_1 = 11$, $p_2 = 23$ và $p_3 = 41$ và $n = p_1 p_2 p_3 = 10373$ và $\varphi(n) = (p_1 - 1)(p_2 - 1)(p_3 - 1) = 8800$. Ta chọn số $e = 7$ và dùng Euclid mở rộng để tìm $d = 7543$ thỏa mãn $de \equiv 1 (\mathrm{mod}\ \varphi(n))$.

Ta có khóa công khai ($e = 7$, $n = 10373$) và khóa bí mật ($d = 7543$, $n = 10373$).

- *Quá trình mã hóa*

Để thực hiện mã hóa một bản tin $m = 35$, *Bob* sử dụng lũy thừa theo *modulo* để tính $c = m^e (\mathrm{mod}\ n) = 35^7 (\mathrm{mod}\ 10373) = 7146$ và gửi cho *Alice*.

- *Quá trình giải mã*: *Alice* tính $m = c^d (\mathrm{mod}\ n) = 7146^{7543} (\mathrm{mod}\ 10373) = 35$.

1.1.8.3. Nhận xét về multi-prime RSA

Như trên ta đã sử dụng *r* số nguyên tố với $r > 2$ để thực hiện sinh cặp khóa để mã hóa và giải mã cho *multi-prime RSA* và kết quả cho chính xác bản rõ. Tùy từng mục đích sử dụng ta sẽ dùng cụ thể *r* = 3, *r* = 4… số nguyên tố.

Khi mở rộng như vậy thì tính an toàn sẽ được tăng lên vì như ta thấy việc phá mã của RSA khó tương đương với bài toán phân tích một số nguyên thành tích các thừa số nguyên tố. Chẳng hạn, nếu có số $n = pq$ là tích của hai số nguyên tố thì chỉ cần tìm ra một số sẽ tìm ra số còn lại. Nhưng nếu $n = pqr$ là tích của ba số nguyên tố thì khi tìm được thừa số nguyên tố đầu tiên sẽ chưa tìm được thừa số thứ hai mà phải tiếp tục phân tích. Như thế sẽ phải tốn rất nhiều thời gian để tính toán hơn.

Tuy nhiên, theo [13] độ an toàn của *multi-prime RSA* không nhỏ hơn *RSA chuẩn* với cùng kích cỡ *modulo n*. Do đó, số lượng số nguyên tố *r* tham gia vào *modulo n* phải vừa đủ. Bảng dưới đây liệt kê số lượng số nguyên tố lớn nhất để đảm bảo an toàn cho *multi-prime RSA* với kích cỡ *modulo* khác nhau. Dữ liệu trong bảng dưới được lấy từ [13], độ an toàn của *multi-prime RSA* với số lượng số nguyên tố *r* tham gia thông qua thời gian chạy phân tích của phương pháp phân tích thừa số NFS.

Bảng 1.2. Số lượng số nguyên tố lớn nhất an toàn cho các cỡ modulo khác nhau

Modulo size (*bit*)	1024	2048	4096	8192
Maximum number of prime (*r*)	3	3	4	5

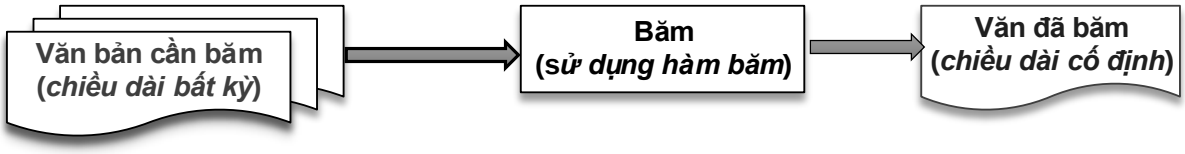
Theo [13] sự khác biệt giữa *multi-prime RSA* với *RSA chuẩn* (hai số nguyên tố) là *multi-prime RSA* sử dụng số lượng số nguyên tố nhiều hơn ($r > 2$). Tuy nhiên, ta có thể tìm thấy sự khác biệt trong việc thực hiện *giải mã* bản mã có một số thuận lợi. Thứ nhất là thời gian, sử dụng định lý phần dư Trung Hoa (Chinese Remainder Theorem) và thực hiện phép tính toán song song, số phép *bit* cần để giải mã là $\frac{3}{2r^3}(\log(n))^3$. Như vậy, thời gian cần để giải mã giảm khi thêm mỗi số nguyên tố trong *modulo*. Thuận lợi thứ hai là về không gian, cũng sử dụng định lý phần dư Trung Hoa, không gian cần cho tất cả các tính toán trong quá trình giải mã cũng chỉ yêu cầu $\log_2(p_r)$, trong đó p_r là số nguyên tố lớn nhất trong *modulo n*. Nếu tất cả các số nguyên tố p_i có chiều dài *bit* cỡ $\frac{\log(n)}{r}$ (cùng độ lớn), không gian cần để thực hiện giải mã sẽ giảm mỗi số nguyên tố

được thêm vào trong *modulo n.* Tuy nhiên nếu sử dụng quá nhiều số nguyên tố tham gia trong *modulo n* sẽ làm tăng rủi ro khi thực hiện phân tích nhân tử của *n*, do đó cần cân bằng độ lớn và số lượng các số nguyên tố tham gia trong *modulo n.*

1.1.9. Sơ lược về hàm băm

Hàm băm (hash function) là hàm chuyển đổi một thông điệp có độ dài bất kỳ thành một dãy *bit* có độ dài cố định. Các hàm băm nhận một chuỗi *bit* có chiều dài tùy ý (hữu hạn) làm dữ liệu đầu vào và tạo ra một chuỗi *bit* mới có chiều dài cố định *n bit*, được gọi là *giá trị băm*. Trong mật mã học hàm băm có một số tính chất bảo mật nhất định để phù hợp trong nhiều ứng dụng như: Chứng thực, kiểm tra tính nguyên vẹn của thông điệp.

Một hàm băm nhận đầu vào là một xâu ký tự (thông điệp) có độ dài tùy ý và tạo ra kết quả một xâu ký tự có độ dài cố định.



Hình 1.5. Minh họa về hàm băm

Đối với một hàm băm với mọi dữ liệu đầu vào x_1 cho trước thì không thể tính toán để tìm ra được bất kỳ một đầu vào x_2 nào ($x_1 \neq x_2$) sao cho giá trị băm $h(x_1) = h(x_2)$. Cũng như với mọi đầu ra *y* cho trước không thể tính toán để tìm được bất kỳ dữ liệu đầu vào *x* nào sao cho giá trị băm $h(x)$ bằng giá trị đầu vào *y* ban đầu. Nói cách khác hàm băm là một hàm một chiều.

Hàm băm phổ biến hiện nay là MD5 và SHA-1.

Hình 1.6. Minh họa về giá trị băm

1.2. Cơ sở toán học của mật mã [1], [5], [8], [9], [10], [11], [14]

Trong mục này sẽ nhắc lại một số kiến thức toán học cơ bản liên quan. Để nội dung của luận văn không quá chi tiết, các kiến thức đề cập đến chủ yếu là các khái niệm, định nghĩa, định lý..., còn các phần chứng minh sẽ được lược bỏ.

1.2.1. Ước chung lớn nhất

Nếu số nguyên *a* chia hết cho số nguyên dương *d* thì *d* gọi là ước của *a* và *a* được gọi là bội của *d*, ký hiệu *d*|*a*.

Ước chung lớn nhất của hai số nguyên *a* và *b* là số nguyên *d* lớn nhất sao cho $d \mid a$ và $d \mid b$. Ký hiệu là $\gcd(a, b) = d$.

Thí dụ 1.4. Ước chung lớn nhất của hai số nguyên 24 và 84 là gcd(24, 84) = 12.

1.2.2. Đồng dư thức

Định nghĩa 1.1. *Cho a là một số nguyên và m là một số nguyên dương, khi đó ta ký hiệu a mod m là số dư khi chia a cho m.*

Định nghĩa 1.2. *Cho a và b là hai số nguyên và n là số nguyên dương thì a được gọi là đồng dư với b theo modulo n nếu a – b chia hết cho n. Có nghĩa là hai số nguyên a, b có cùng số dư khi chia cho n. Ký hiệu:* $a \equiv b \pmod{n}$ *gọi là đồng dư thức và n được gọi là modulo của đồng dư.*

Định lý 1.1. *Cho n là một số nguyên dương. Các số nguyên a và b đồng dư theo modulo n nếu và chỉ nếu tồn tại một số nguyên k sao cho a = b + kn.*

Quan hệ đồng dư giữa *a* và *b* theo *modulo m* có tính chất phản xạ, đối xứng và bắc cầu, tức quan hệ đồng dư là một quan hệ tương đương.

1.2.3. Lớp tương đương

Quan hệ đồng dư là một quan hệ tương đương. Khi đó nó tạo ra một phân hoạch trên tập các số nguyên Z thành các lớp tương đương. Nghĩa là nó chia tập số nguyên Z thành các tập con không rỗng và rời nhau.

Hai số nguyên thuộc cùng một lớp tương đương khi và chỉ khi chúng cho cùng số dư nếu chia cho *n*. Nếu $a \in Z$, lớp tương đương chứa *a* được ký hiệu $[a]_n$ là tập các số nguyên có cùng số dư *a* khi chia cho *n*

Cho *m* là số nguyên dương, khi đó có *m* lớp đồng dư *modulo m*, được ký hiệu là $[0]_m, [1]_m, ..., [m-1]_m$. Mỗi lớp tương đương được gọi là số nguyên *modulo m.* Ta có

$$[0]_m = [m] = [2m] = ...$$

$$[1]_m = [m+1]_m = [2m+1]_m = ...$$

.....

$$[m-1]_m = [2m-1]_m = [3m-1]_m = ...$$

Tập hợp các số theo *modulo m* này được ký hiệu bởi $\mathbb{Z}_m = \{[0]_m, [1]_m, ..., [m-1]_m\}$.

1.2.4. Khái niệm cơ bản trong cấu trúc đại số

1.2.4.1. Nửa nhóm và vị nhóm

Cho tập hợp *X* với phép toán nhân. Ta nói $(X, *)$ (Gọi tắt là *X*) là:

- Một *nửa nhóm* nếu phép toán nhân có tính kết hợp trên *X*.
- Một *vị nhóm* nếu phép toán nhân kết hợp trên *X* và có phần tử trung hòa trên *X*.

Một *nửa nhóm* được gọi là *giao hoán* hay *Abel* nếu phép toán tương ứng giao hoán.

Thí dụ 1.5. Phép cộng trên các tập $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$ trở thành các vị nhóm giao hoán.

1.2.4.2. Nhóm

Nhóm là một vị nhóm mà mọi phần tử đều khả đối xứng. Nói cách khác, một tập *G* khác rỗng với phép toán nhân được gọi là một *nhóm* nếu thỏa các tính chất sau:

- Với mọi $x, y, x \in G$, $(xy)z = x(yz)$.
- Tồn tại phần tử $e \in G$, sao cho với mọi $x \in G$, $ex = xe = x$.
- Với mọi $x \in G$, tồn tại $x^{-1} \in G$ sao cho $xx^{-1} = x^{-1}x = e$.

Nếu phép toán trên G là phép cộng thì các tính chất trên trở thành:

- Với mọi $x, y, x \in G$, $(x+y)+z = x+(y+z)$;
- Tồn tại phần tử $0 \in G$, sao cho với mọi $x \in G$, $0+x = x+0 = x$;
- Với mọi $x \in G$, tồn tại $-x \in G$ sao cho $x+(-x) = (-x)+x = 0$.

Nếu phép toán trên nhóm G giao hoán thì ta nói G là *nhóm giao hoán* hay *nhóm Abel*. Nhóm G được gọi là *nhóm hữu hạn* nếu G *hữu hạn*. Khi đó, số phần tử của G được gọi là *cấp* của G. Nếu G không hữu hạn thì gọi là *nhóm vô hạn*.

Định lý 1.2 (*Định lý Lagrange*). Cho G là một nhóm cấp *n* và $g \in G$. Khi đó, cấp của g là ước của *n*.

Thí dụ 1.6. Tập hợp các số nguyên Z cùng với phép cộng thông thường là một *nhóm* giao hoán mà ta thường gọi là *nhóm* cộng các số nguyên.

1.2.4.3. Nhóm con

Cho G là một *nhóm*, cho $S \subset G$ và $S \neq \varnothing$. S được gọi là *nhóm con* của G nếu:

- Phần tử trung hòa *e* của G nằm trong S.
- Tập S là tập đóng trong G. Tức là với mọi $x, y \in S$ thì $xy \in S$.

- Tập S là tập đóng đối với phép lấy nghịch đảo trong G. Tức là với mọi $x \in S$ thì $x^{-1} \in S$.

1.2.4.4. Nhóm Cyclic

Cho G là một nhóm cùng với phép toán nhân, ký hiệu $(G, *)$ gọi là *nhóm Cyclic* nếu như nó sinh ra bởi một trong các phần tử của nó. Tức là tồn tại $g \in G$ mà mỗi phần tử $a \in G$ đều tồn tại một số nguyên $n \in N$ để $g^n = a$. Khi đó, g được gọi là phần tử *sinh* hay phần tử *nguyên thủy* của G.

Thí dụ 1.7. Nhóm $(Z^+, +)$ là nhóm *Cyclic*, với phần tử sinh $g = 1$.

Cho $(G, *)$ là nhóm *Cyclic* với phần tử sinh là *g* và phần tử trung hòa *e*. Nếu tồn tại số tự nhiên nhỏ nhất *n* mà $g^n = e$ thì G sẽ chỉ gồm có *n* phần tử khác nhau gồm $e, g, g^2, g^3 ... g^{n-1}$. Khi đó G được gọi là *nhóm Cyclic* hữu hạn cấp *n*.

Phần tử $\alpha \in G$ có cấp *d*, nếu *d* là số nguyên dương nhỏ nhất sao cho $\alpha^d = e$. Trong đó, *e* là phần tử trung hòa trong G. Như vậy, phần tử α có cấp 1, nếu $\alpha = e$.

1.2.5. Các lớp thặng dư

Định nghĩa 1.3. *Các lớp tương đương theo quan hệ đồng dư theo modulo n gọi là các lớp thặng dư theo modulo n. Ký hiệu là* $\mathbb{Z}_n$.

Thí dụ 1.8. Cho tập $\mathbb{Z}_3$ gồm các lớp sau:

$$[0]_4 = \{..., 0, 4, 8...\}; [1]_4 = \{..., 1, 5, 9, ...\}; [2]_4 = \{..., 2, 6, 10, ...\}$$

Định nghĩa 1.4. *Nếu từ mỗi lớp thặng dư theo modulo n lấy ra một đại diện thì tập hợp các đại diện đó được gọi là một hệ thặng dư đầy đủ theo modulo n.*

Thí dụ 1.9. Tập $\{0, 1, 2, 3, 4\}$ là một hệ thặng dư đầy đủ *modulo* 5.

Tổng quát, tập $\mathbb{Z}_n = \{0, 1, 2, ..., n-1\}$ thường được gọi là *tập các thặng dư đầy đủ theo modulo n.* Vì mọi số nguyên bất kỳ đều có thể tìm được trong $\mathbb{Z}_n$ một số đồng dư với mình. Tất cả các phép toán đều thực hiện được theo *modulo*.

Các phép toán trong tập $\mathbb{Z}_n$:

- Cho *n* là số nguyên dương, nếu *a* và $b \in \mathbb{Z}_n$ thì

$$(a+b) \pmod{n} = \begin{cases} (a+b), & \text{nếu } (a+b) < n \\ (a+b-n), & \text{nếu } (a+b) > n. \end{cases}$$

- Phép nhân *modulo* của *a* và *b* được thực hiện như phép nhân thông thường, sau đó lấy phần dư khi chia cho *n*.
- Phép tính nghịch đảo nhờ sử dụng thuật toán Euclid mở rộng.

Tập $\mathbb{Z}_n$ là tập đóng đối với phép cộng, phép trừ và phép nhân nhưng không đóng đối với phép chia. Vì phép chia cho *a* theo *modulo n* chỉ thực hiện được khi $\gcd(a,n)=1$.

Ta xét tập $\mathbb{Z}_n^* = \{a \in \mathbb{Z}_n \mid \gcd(a, n) = 1\}$. Có nghĩa là $\mathbb{Z}_n^*$ là tập con của $\mathbb{Z}_n$ bao gồm tất cả các phần tử nguyên tố cùng nhau với *modulo n* và không có hai phần tử nào đồng dư với nhau *modulo n*. Ta gọi tập $\mathbb{Z}_n^*$ là tập *các thặng dư thu gọn theo modulo n.* Mọi số nguyên tố cùng nhau với *n* đều có thể tìm thấy trong $\mathbb{Z}_n^*$ một đại diện đồng dư với mình theo *modulo n.*

Nếu *p* là số nguyên tố thì $\mathbb{Z}_p^* = \{1, 2, ..., p-1\}$. Tập $\mathbb{Z}_n^*$ lập thành một *nhóm con* đối với phép nhân vì trong $\mathbb{Z}_n^*$ phép chia luôn thực hiện được. Ta cũng thường gọi $\mathbb{Z}_n^*$ là *nhóm nhân* của $\mathbb{Z}_n$.

Thí dụ 1.10. $\mathbb{Z}_{10}$ = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} thì $\mathbb{Z}_{10}^*$ = {1, 3, 7, 9 }.

1.2.6. Phi hàm Euler và phần tử nguyên thủy

Định nghĩa 1.5. *Cho một số nguyên n > 1. Phi hàm Euler là số các số nguyên nằm trong khoảng* $[1...n]$ *và nguyên tố cùng nhau với n. Ký hiệu:* $\varphi(n)$. $\varphi(n)$ *có tính chất:*

- Phi hàm Euler $\varphi(n)$ là *cấp* của $\mathbb{Z}_n^*$.
- Nếu *p* là số nguyên tố thì nhóm $\mathbb{Z}_p^*$ có cấp là $p-1$ tức $\varphi(n) = p-1$.
- Nếu có hai số nguyên tố *p, q* khác nhau và $n = pq$ thì $\varphi(n) = (p-1)(q-1)$.
- Nếu *n* được phân tích thành tích các thừa số nguyên tố $n = p_1^{a_1} p_2^{a_2} ... p_k^{a_k}$. Trong đó p_i là các số nguyên tố và a_i là các số nguyên dương thì phi hàm Euler của *n* được tính như sau $\varphi(n) = (p_1 - 1) p_1^{(a_1-1)} (p_2 - 1) p_2^{(a_2-1)} ... (p_k - 1) p_k^{(a_k-1)}$.

Một phần tử $g \in \mathbb{Z}_n^*$ có cấp *m*, nếu *m* là số nguyên dương bé nhất sao cho $g^m = 1$ trong $\mathbb{Z}_n^*$. Theo *Định lý Lagrange* thì $\varphi(n)$ chia hết cho *m.* Vì vậy, với mọi $b \in \mathbb{Z}_n^*$ ta luôn có $b^{\varphi(n)} \equiv 1 \pmod{n}$. Nếu *p* là số nguyên tố thì do $\varphi(n) = p-1$, với mọi $b \in \mathbb{Z}_p^*$ ta

luôn có $b^{p-1} \equiv 1 \pmod{p}$ (*). Nếu *b* có cấp *p*-1, tức *p*-1 là số mũ bé nhất thỏa mã (*) thì các phần tử $b, b^2, ..., b^{p-1}$ đều khác nhau và theo *modulo p* chúng lập thành *nhóm* $\mathbb{Z}_p^*$.

Theo khái niệm đại số, ta nói $\mathbb{Z}_n^*$ là một *nhóm Cyclic* và *b* là phần tử sinh hay còn gọi là *phần tử nguyên thủy*.

Trong lý thuyết số, *phần tử nguyên thủy* có tính chất sau:

- Với mọi số nguyên tố *p*, $\mathbb{Z}_p^*$ là *nhóm Cyclic* và có $\varphi(p-1)$ *phần tử nguyên thủy*.
- Nếu *g* là *phần tử nguyên thủy* theo *modulo p*, thì khi đó $\beta \equiv g^i \pmod{p}$ với mọi *i* mà $\gcd(i, \varphi(n)) = 1$, cũng là phần tử nguyên thủy theo *modulo p*.
- Nếu $p-1 = p_1^{\alpha_1} p_2^{\alpha_2} ... p_s^{\alpha_s}$ là dạng khai triển chính tắc của $p-1$ và nếu $a^{\frac{p-1}{p_1}} \equiv 1 \pmod{p}, ..., a^{\frac{p-1}{p_s}} \equiv 1 \pmod{p}$, thì a là phần tử *sinh* theo *modulo* $p \in \mathbb{Z}_p^*$.
- Nhóm $\mathbb{Z}_n^*$ có *phần tử nguyên thủy* khi và chỉ khi $n = 2, 4, p^k$ hay $2p^k$ khi p là số nguyên tố lẻ và $k \geq 1$. Nếu n là số nguyên tố thì chắc chắn $\mathbb{Z}_n^*$ có *phần tử nguyên thủy*.

Định lý 1.3 (*Định lý Euler*). *Nếu* $a \in \mathbb{Z}_n^*$. *Tức* $gcd(a, n) = 1$ *thì* $a^{\varphi(n)} \equiv 1 \pmod{n}$. *Nếu* $r \equiv s \pmod{\varphi(n)}$ *thì* $a^r \equiv a^s \pmod{n}$.

1.2.7. Phần tử nghịch đảo

Cho $a \in \mathbb{Z}_n$, nghịch đảo nhân của *a* theo *modulo n* là một số nguyên $x \in \mathbb{Z}_n$ sao cho $ax \equiv 1 \pmod{n}$. Số nguyên *x* là duy nhất và *a* được gọi là khả nghịch, nghịch đảo của *a*. Ký hiệu là a^{-1}. *Phần tử nghịch đảo có một số tính chất sau:*

- Cho $a \in \mathbb{Z}_n$, *a* là khả nghịch khi và chỉ khi gcd(*a*, *n*) = 1.
- Cho $a, b \in \mathbb{Z}_n$, thì phép chia của *a* cho *b* theo *modulo n* là tích của *a* và b^{-1} theo *modulo n* và chỉ được xác định khi *b* có nghịch đảo theo *modulo n*.
- Nếu $d = gcd(a, n)$ thì $ax = b \pmod{n}$ có nghiệm *x* chỉ nếu *d* chia hết cho *b*.

Thí dụ 1.11. Cho *a* = 550, *modulo n* = 1759. Khi đó nghịch đảo của *a* và *n* là 355 vì gcd(550, 1759) = 1 và 550.355 = 1 (mod 1759).

1.2.8. Khái niệm logarit rời rạc

Cho *p* là số nguyên tố, α là phần tử nguyên thủy của $\mathbb{Z}_p$ và $\beta \in \mathbb{Z}_p^*$. *Logarit rời rạc* chính là việc giải phương trình $x = \log_\alpha \beta \pmod p$ với ẩn *x*. Nói cách khác là phải tìm ẩn *x* duy nhất sao cho $\alpha^x \equiv \beta \pmod p$.

1.2.9. Số nguyên tố và một số vấn đề liên quan

1.2.9.1. Số nguyên tố và nguyên tố cùng nhau

Định nghĩa 1.6. Số nguyên tố là số nguyên chỉ chia hết cho 1 và chính nó. Số nguyên lớn hơn 1 không phải là số nguyên tố thì gọi là *hợp số*. Số nguyên tố là vô hạn. Hai số *a* và *b* được gọi là *nguyên tố cùng nhau* nếu $\gcd(a, b) = 1$.

Thí dụ 1.12. Các số nguyên tố là: *2, 3, 5, 7, 11, 13…*

Thí dụ 1.13. Số 6 và 35 là nguyên tố cùng nhau vì gcd(6, 35) = 1.

1.2.9.2. Một số định lý về số học liên quan tới số nguyên tố

Định lý 1.4 (*Định lý cơ bản của số học*). *Mỗi số tự nhiên* $n > 0$ *đều có thể biểu diễn duy nhất nếu không tính đến thứ tự các thừa số dưới dạng* $n = p_1^{\alpha_1} p_2^{\alpha_2} \ldots p_k^{\alpha_k}$.

Trong đó, p_i là các số nguyên tố (*i*=1, 2, 3..., *k*) và $\alpha_1, \alpha_2 \ldots$ là các số tự nhiên dương.

Thí dụ 1.14. Cho $n = 6936 = 2^3 \times 3 \times 17^2$.

Định lý 1.5. *Mọi hợp số n đều có ước nguyên tố nhỏ hơn* $\sqrt{n}$.

Bổ đề 1.1. *Nếu n không có bất kỳ ước số a nào* $\in \left]1, \sqrt{n}\right]$ *thì n là số nguyên tố.*

Định lý 1.6 (*Định lý Fermat bé*). *Cho p là số nguyên tố và a là số nguyên dương, nếu gcd(a, p) = 1 thì* $a^{p-1} \equiv 1 \pmod p$ *hay* $a^p \equiv a \pmod p$.

Hệ quả 1.1. *Nếu p là số nguyên tố và a là số nguyên dương thì* $a^p \equiv a \pmod p$.

1.2.9.3. Phương pháp kiểm tra và sinh số nguyên tố

Phương pháp cổ điển để kiểm tra tính nguyên tố của số nguyên *n* cho trước như Trial Division hay phương pháp sàng Eratosthenes tuy kiểm tra chính xác nhưng nếu số nguyên *n* rất lớn thì hai phương pháp này không khả thi. Do đó, một trong những phương pháp hiệu quả để kiểm tra tính nguyên tố của một số nguyên lớn *n* cho trước đó là phương pháp thử xác suất theo các thuật toán được mô tả như dưới đây.

- *Thuật toán Fermat*

Thuật toán Fermat là kiểm tra xác suất dựa trên *Định lý 1.6*.

Thí dụ 1.15. Ta có $2^{6-1} = 32 \equiv 2 \pmod 6$.

Nếu *p* là số nguyên tố thì $a^{p-1} \equiv 1 \pmod p$ (*), nếu không thỏa (*) thì có thể đoán *p* không phải là số nguyên tố. Tuy nhiên, một số trường hợp nó lại không đúng, như *n*=341=11.31 không phải là số nguyên tố nhưng nó vẫn thỏa (*). Do đó, *Định lý 1.3* có thể dùng để kiểm tra hợp số. Các số mà thoả mãn Định lý 1.3 mà không phải là số nguyên tố gọi là *số giả nguyên tố (pseudoprime*).

Định nghĩa 1.7 (*Số giả nguyên tố*). *Một số giả nguyên tố cơ sở a là một hợp số nguyên n sao cho* $a^{n-1} \equiv 1 \pmod n$.

Nếu số giả nguyên tố cơ sở *a* không tồn tại thì Định lý Fermat là cách đơn giản để kiểm tra tính nguyên tố của *n* nếu và chỉ nếu $a^{n-1} \equiv 1 \pmod n$. Thật đáng tiếc số giả nguyên tố cơ sở *a* lại tồn tại với mọi cơ sở, vì vậy định lý Fermat chỉ cho một cách kiểm tra thiên về hợp số.

Algorithm 1.4 (*Thuật toán xác định số giả nguyên tố*).

```
Input:  Number n, b is number integer which that base
Output: True if pseudoprime else False.
 If (b^(n-1)% n = 1) return True else then return False.
```

Nếu một số vượt qua kiểm tra *số giả nguyên tố* với vài cơ sở thì khả năng nó là số nguyên tố khá chắc chắn (vẫn có một xác suất nào đó). Thuật toán Fermat sẽ kiểm tra một số *n* là *giả nguyên tố* với *k* cơ sở được chọn một cách ngẫu nhiên và kết luận là một số nguyên tố với xác suất nào đó nếu và chỉ nếu nó vượt qua *k* lần kiểm tra.

Algorithm 1.5. (*Thuật toán Fermat kiểm tra k lần số n để kiểm tra tính nguyên tố*)

```
Input: Number n, k is number integer which that base.
Output: True if n is prime else False.
 1. For i from 1 to k do the following:
    1.1. b ← random(2, n - 1);
    1.2. If (!pseudoprime(n, b)) then return(False);
 2. Return(True).
```

- *Thuật toán Miller - Rabin*
  - *Cơ sở lý thuyết*

Cho tập hữu hạn $\mathbb{Z}_p$, trong đó *p* là số nguyên tố. Chắc chắn rằng 1 và -1 luôn là căn bậc hai của 1 theo *modulo p*. Khi đó

$$x^2 \equiv 1 \pmod{p}$$

$$\Leftrightarrow x^2 - 1 \equiv 0 \pmod{p}$$

$$\Leftrightarrow (x-1)(x+1) \equiv 0 \pmod{p}.$$

Từ đó $x-1$ hoặc $x+1$ chia hết cho *p*. Giả sử *p* là số nguyên tố lẻ, khi đó *p* - 1 là số chẵn và ta viết *p* - 1 dưới dạng $m.2^s$. Trong đó $s \geq 1$ và *m* lẻ, có nghĩa ta rút hết các thừa số 2 khỏi *p* -1. Với số $a \in [1...p-1]$. Xét dãy $x_k = a^{2^k m}$, với *k* = 1, 2,…*s*. Khi đó, $x_k = (x_{k-1})^2$ với *k* = 1,2,…*s* và $x_s = a^{p-1}$. Theo Định lý Fermat nhỏ ta có $a^{p-1} \equiv 1 \pmod{p}$ hay $x_s \equiv 1 \pmod{p}$ hay $x_{s-1}^2 \equiv 1 \pmod{p}$.

Do đó hoặc $x_{s-1} \equiv 1 \pmod{p}$ hoặc $x_{s-1} \equiv -1 \pmod{p}$. Nếu $x_{s-1} \equiv -1 \pmod{p}$ ta dừng lại, nếu ngược lại ta tiếp tục với x_{s-2}, sau một số bước hữu hạn ta có.

Hoặc ta có một chỉ số $k \in [0, s-1]$ sao cho $x_k \equiv -1 \pmod{p}$ hoặc với *k* = 0 ta có

$$x_k \equiv 1 \pmod{p}.$$

Có mệnh đề *Q* (*p*, *a*): Nếu *p* là số nguyên tố lẻ và $p-1 = m.2^s$, với $\forall a \in]0,\ p-1[$ thì hoặc tồn tại $x_k = a^{2^k m} \equiv 1 \pmod{p}$ với $\forall k \in [0, s]$ hoặc $\exists k \in [0, s]$ sao cho tồn tại $x_k = a^{2^k m} \equiv -1 \pmod{n}$.

Định lý 1.7. *Nếu n là một hợp số dương lẻ thì tồn tại không quá* $\frac{n-1}{4}$ *cơ sở b, với giá trị* $b \in [1, n-1]$, *sao cho n trải qua được kiểm tra Miller đối với cơ sở đó.*

Từ Định lý 1.7 ta suy ra, nếu số b được chọn ngẫu nhiên trong khoảng $1 \leq b \leq n-1$ thì *n* trải qua kiểm tra Miller cơ sở *b* với xác xuất bé hơn 1/4. Như vậy, nếu ta chọn *k* số ngẫu nhiên thì xác suất để *n* trải qua kiểm tra Miller đối với *k* cơ sở đó sẽ không quá $\frac{1}{4^k}$. Khi *k* đủ lớn (chẳng hạn *k* = 20) thì xác suất này quá nhỏ, nên với *n* trải qua với 20 cơ sở ngẫu nhiên thì có thể tin chắc chắn rằng *n* là số nguyên tố.

o *Giải thuật Miller – Rabin*

Algorithm 1.6 (*Thuật toán Miller-Rabin kiểm tra tính nguyên tố của số nguyên n*).

Input: An odd integer n >1 and a positive security para t.

Output: The answer "COMPOSITE" or "PRIME".

1. Write $(n-1) \leftarrow 2^s r$ such that r is odd, $s > 1$;
2. Repeat from 1 to t
   Choose a random integer a which satisfies in [2, n-1];
3. Compute $y \leftarrow a^r \pmod n$;
4. If $y \neq 1$ and $y \neq (n-1)$ then do the following:
   4.1. $j \leftarrow 1$;
   4.2. While $j < s$ and $y \neq (n-1)$ then do the following:
        $y \leftarrow y^2 \pmod n$ If $y = 1$ then return("COMPOSITE");
        $j \leftarrow j + 1$;
   4.3. If $y \neq (n-1)$ then return("COMPOSITE");
5. Return("PRIME").

Thí dụ 1.16. Cho $n = 221$.Ta có $n - 1 = 220 = 2^2.55$, vậy $s = 2$ và $d = 55$.

Bảng 1.3. Kiểm tra tính nguyên tố của số nguyên n theo thuật toán Miller-Rabin

Thử lần 1	1	Chọn ngẫu nhiên số $a < n$, $a = 174$.
	2	Tính $a^{2^0.d} \pmod n = 174^{55} \pmod{221} = 47 \neq 1, n-1$.
	3	Tính $a^{2^1.d} \pmod n = 174^{110} \pmod{221} = 220 = n-1$.
	Sau lần 1, ta tạm kết luận $n = 221$ là số nguyên tố, tiếp tục thử lần 2.	
Thử lần 2	1	Chọn ngẫu nhiên $a < n$, $a = 137$.
	2	Tính $a^{2^0.d} \pmod n = 137^{55} \pmod{221} = 188 \neq 1, n-1$.
	3	Tính $a^{2^1.d} \pmod n = 137^{110} \pmod{221} = 205 \neq n-1$.
	Sau 2 lần thử ta kết luận $n = 221$ là hợp số, do a =137 là bằng chứng.	

1.2.10. Thuật toán Euclid

Thuật toán Euclid giúp tính ước chung lớn nhất của hai số nguyên *a* và *b*. Nó tạo ra một cặp số nguyên dương mới số nhỏ hơn và phần dư của của phép chia hai số ban đầu. Thuật toán Euclid được mô tả chi tiết như sau.

Algorithm 1.7 (*Thuật toán Euclid*).

Input: Two positive integers a and b with $a \geq b$.

Output: Return gcd(a, b).

1. While $b > 0$ do the following:

   1.1. $r \leftarrow a \bmod b$;

   1.2. $a \leftarrow b$, $b \leftarrow r$;

2. Return (a).

Thí dụ 1.17. Tìm gcd(1406, 646). Các bước thực hiện được mô tả như bảng sau.

Bảng 1.4. Mô tả các bước tính gcd(a, b) theo thuật toán Euclid

	a	*b*	*r*
1406 = 2.646 + 114	1406	646	646
646 = 5.114 + 76	646	114	114
114 = 1.76 + 38	114	76	76
76 = 2.38 + 0	76	38	38
	38	0	0

Ta biết rằng nếu $\gcd(a, b) = d$ thì phương trình bất định $ax + by = d$ có nghiệm nguyên (x, y) và có thể tìm được bằng *Thuật toán Euclid* mở rộng như sau

Algorithm 1.8 (*Thuật toán Euclid mở rộng*).

Input: Two positive integers a and b with $a \geq b$.

Output: Positive $d = \gcd(a, b)$ and x, y with $ax + by = d$.

1. If $b = 0$ then do the following:

   1.1. $d \leftarrow a$, $x \leftarrow 1$, $y \leftarrow 0$;

   1.2. Return (d, x, y);

2. $x_2 \leftarrow 1, x_1 \leftarrow 0, y_2 \leftarrow 0, y_1 \leftarrow 1$;

3. While $b > 0$ then do the following:

   3.1. $q \leftarrow a \; div \; b, r \leftarrow a \bmod b, x \leftarrow (x_2 - qx_1), y \leftarrow (y_2 - qy_1)$

   3.2. $a \leftarrow b, b \leftarrow r, x_2 \leftarrow x_1, x_1 \leftarrow x, y_2 \leftarrow y_1, y_1 \leftarrow y$;

4. $d \leftarrow a, x \leftarrow x_2, y \leftarrow y_2$;

5. Return (d, x, y).

Thí dụ 1.18. Dùng thuật toán Euclide mở rộng cho các số a = 4864 và b = 3458, lần lượt được các giá trị sau đây cho các biến $a, b, q, r, x, y, x_1, x_2, y_1, y_2$.

Bảng 1.5. Mô tả các bước tính theo Euclid mở rộng

a	b	q	r	x	y	x_1	x_2	y_1	y_2
4864	3458					0	1	1	0
3458	1406	1	1406	1	-1	1	0	-1	1
1406	646	2	646	-2	3	-2	1	3	-1
646	114	2	114	5	-7	5	-2	-7	3
114	76	5	76	-27	38	-27	5	38	-7
76	38	1	38	32	-45	32	-27	-45	38
38	0	2	0	-91	128	-91	32	128	-45

Theo bảng trên khi kết thúc các vòng lặp (với b = 0), thực hiện tiếp lệnh 4 ta được kết quả d = 38 và y = -45, cặp số (32, -45) thỏa mãn: 4864.32 + 3458. (-45) = 38.

1.2.11. Định lý Trung Hoa về phần dư

Định lý 1.8 (*Phần dư Trung Hoa*).

Giả sử các số nguyên $n_1, n_2, .., n_k$ là từng cặp nguyên tố cùng nhau. Khi đó, hệ phương trình đồng dư tuyến tính dưới đây có một nghiệm duy nhất theo mod n

$$\begin{cases} x_1 \equiv a_1 \pmod{n_1} \\ x_2 \equiv a_2 \pmod{n_2} \\ ..... \\ x_k \equiv a_k \pmod{n_k}. \end{cases}$$

Ký hiệu: $n = n_1 n_2 ... n_k, N_i = \dfrac{n}{n_i}$. Nghiệm duy nhất được cho bởi biểu thức

$$x = \sum_{i=1}^{k} a_i N_i M_i \pmod{n}.$$

Trong đó: $M_i = N_i^{-1} \pmod{n_i}$, (có M_i vì N_i và n_i nguyên tố cùng nhau).

Thí dụ 1.19. Cặp $x \equiv 3 \pmod 7$ và $x \equiv 7 \pmod{13}$ có một nghiệm $x \equiv 59 \pmod{91}$.

Nếu $gcd(n_1, n_2) = 1$ thì cặp phương trình $x \equiv a \pmod{n_1}$ và $x \equiv a \pmod{n_2}$ có nghiệm duy nhất $x \equiv a \pmod{n}$ theo mod n với $n = n_1.n_2$.

1.2.12. Hàm một phía và hàm cửa sập một phía

Hàm số số học $y = f(x)$ được gọi là *hàm một phía* (one-way function), nếu việc tính thuận từ *x* ra *y* là "dễ", nhưng việc tính ngược lại từ *y* tìm lại *x* là rất "khó", có thể hiểu chẳng hạn dễ là tính được trong thời gian đa thức (với đa thức bậc thấp), còn khó là không tính được thời gian đa thức. Thực tế thì cho đến hiện nay, việc tìm và chứng minh một hàm số nào đó là không tính được trong thời gian đa thức còn là việc rất khó, cho nên "khó" thường khi chỉ được hiểu một cách đơn giản chưa tìm được thuật toán tính nó trong thời gian đa thức. Với cách hiểu tương đối như vậy về "dễ" và "khó", người ta đã đưa ra một số thí dụ sau đây về các hàm một phía.

Thí dụ 1.20. Tích $n = pq$, với *p* và *q* là số nguyên tố lớn . Hàm số $y = x^2 \pmod n$ cũng được xem là một hàm một phía.

Thí dụ 1.21. Cho $n = pq$ là tích của hai số nguyên tố lớn, và *a* là một số nguyên sao cho $\gcd(a, \varphi(n)) = 1$. Hàm số $y = x^a \pmod n$ (từ $\mathbb{Z}_n$ vào $\mathbb{Z}_n$) cũng là một hàm một phía, nếu giả thiết là biết *n* nhưng không biết *p, q*.

Hàm *y = f(x)* gọi là *hàm cửa sập một phía* (trapdoor one-way function), nếu việc tính thuận từ *x* ra *y* là "dễ", việc tính ngược từ *y* tìm lại *x* là rất "khó", nhưng có một cửa sập *z* để với sự trợ giúp của cửa sập *z* thì việc tính *x* từ *y* và *z* lại trở thành dễ.

Thí dụ 1.22. Xét hàm số lũy thừa theo *modulo* $y = x^a \pmod n$ khi biết *p* và *q* là hàm cửa sập một phía. Từ *x* tính *y* là dễ, từ *y* tìm *x* (nếu chỉ biết *n, a*) là rất khó, nhưng vì biết *p* và *q* nên biết $\varphi(n) = (p-1)(q-1)$, và dùng thuật toán Euclid mở rộng được trình bày tại *Algorithm 1.8*, tìm được *b* sao cho $ab \equiv 1 \pmod{\varphi(n)}$, từ đó dễ tính được $x = y^b \pmod n$. Ở đây có thể xem *b* là cửa sập.

1.2.13. Lý thuyết cơ bản về độ phức tạp của thuật toán

Thuật toán hay giải thuật là tập hợp các quy tắc và thủ tục theo trật tự nhất định để giải quyết một vấn đề nào đó. Thuật toán có những đặc trưng sau:

- *Tính hữu hạn:* Một thuật toán bao giờ cũng phải kết thúc sau hữu hạn bước.
- *Tính xác định:* Mọi bước của thuật toán bao giờ cũng được xác định rõ ràng, chính xác và do đó luôn thực hiện được.
- *Giá trị vào và ra:* Một thuật toán bao giờ cũng có giá trị nhập vào và kết quả ra.

- *Tính hiệu quả:* Để đạt được kết quả mong muốn và tổng thời gian thực hiện thuật toán phải đủ nhỏ. Thực tế người ta thường qui về các đơn vị tính sơ cấp. Nó được đo bởi số lượng các đơn vị tính sơ cấp mà thuật toán yêu cầu thực hiện.

Khái niệm về *độ phức tạp tính toán* (về không gian hay thời gian) của một tiến trình tính toán là số ô nhớ được dùng hay số các phép toán sơ cấp dùng trong tiến trình đó.

Độ phức tạp về mặt không gian: Gắn liền với các cấu trúc dữ liệu được sử dụng.

Độ phức tạp về mặt thời gian: Độ phức tạp này được thể hiện qua số lượng các câu lệnh về các phép gán, phép tính số học, phép so sánh…

Độ tăng của hàm: Cho $f(x)$ và $g(x)$ là hai hàm từ tập số nguyên hoặc tập số thực vào tập số thực, ta nói $f(x)$ là $O(g(x))$ hay $f(x)$ có quan hệ big – O với $g(x)$, ký hiệu $f(x)=O(g(x))$, nếu tồn tại hai hằng số C và k sao cho: $|f(x)| \leq C.|g(x)|, \ \forall x \geq k$.

Số các phép toán của thuật toán là $f(n)$ trong đó n là kích thước đầu vào.

Ta thường quy độ phức tạp về thời gian về các mức sau: Độ phức tạp hằng số ($O(1)$), logarit ($O(\log_a n)$), tuyến tính ($O(n)$), đa thức ($O(n^k)$), hàm mũ ($O(a^n)$)…

Thuật toán $f(n)$ có độ phức tạp thời gian đa thức, nếu có một đa thức $p(n)$ sao cho với mọi n đủ lớn ta luôn có $f(n) <= p(n)$, với $f(n)$ là độ phức tạp tính toán theo thời gian.

Dễ nhận thấy rằng với một thuật toán dưới mũ, thời gian làm việc với các số nguyên lớn vẫn không khả thi. Do đó, với các ứng dụng xử lý số lớn, ta thường phải cố gắng biến đổi để thu được một thuật toán có thời gian tính toán đa thức.

CHƯƠNG 2
LỚP THƯ VIỆN TÍNH TOÁN SỐ NGUYÊN LỚN

2.1. Cấu trúc dữ liệu và tổ chức biểu diễn số nguyên lớn

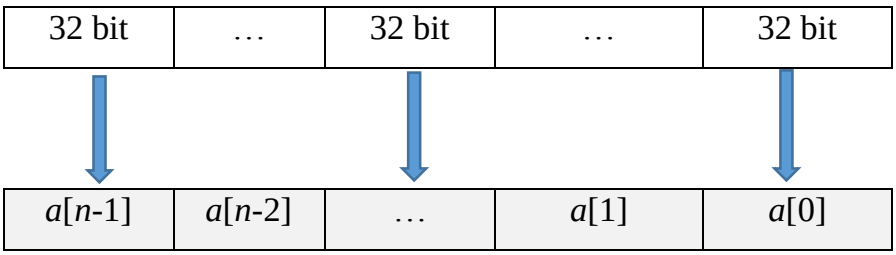
2.1.1. Cấu trúc dữ liệu của số lớn

Có rất nhiều cách tổ chức, biểu diễn số nguyên lớn như: chuỗi ký tự (xâu), mảng một chiều lưu trữ các số liên tiếp…Trong luận văn này số nguyên lớn được xây dựng dưới dạng một đối tượng *BigNumInteger*. Đối tượng này có các thuộc tính và các phương thức cho đối tượng số lớn. Hai thuộc tính của *BigNumInteger* là:

- *Data:* Mảng một chiều có kiểu dữ liệu là *uint*. Mỗi phần tử của mảng lưu trữ tối đa 32 *bit*, tương ứng với miền giá trị xác định trong khoảng $[0\ldots 2^{32}-1]$.
- *Lengthdata:* Có kiểu dữ liệu là ***int*.** Thuộc tính này dùng để xác định chiều dài thực của số lớn. Nó cho biết cần bao nhiêu phần tử 32 *bit* để lưu trữ số lớn.

2.1.2. Biểu diễn số nguyên lớn

Nguyên tắc tổng quát để biểu diễn số nguyên là dùng *m bit* để biểu diễn số nguyên không dấu *A*. Thuộc tính *Data* và *Lengthdata* của số nguyên lớn được biểu diễn như sau



Hình 2.1. Mô tả cách biểu diễn số nguyên lớn.

Với hình trên ta thấy được cách biểu diễn một số nguyên lớn của mảng *Data* là lưu một dãy các *bit* của số nguyên lớn, mỗi phần tử *a*[*i*] với $i \in [0...n-1]$ chứa các dãy 32 *bit* liên tiếp nhau đến khi hết các *bit* biểu diễn của số lớn *A*.

Các chuỗi 32 *bit* lần lượt của số nguyên lớn được lưu trữ theo thứ tự ngược từ vị trí *a*[0] tới *a*[*n*-1], từ phải sang trái để thuận tiện cho các thao tác tính toán sau này. Cũng theo hình minh họa trên cụ thể như sau:

- Phần tử *a*[0] sẽ lưu 32 *bit* có trọng số thấp nhất (32 *bit* phải nhất của chuỗi *bit*).
- Phần tử *a*[1] sẽ lưu 32 *bit* tiếp theo. 32 *bit* này có trọng số lớn hơn 32 *bit a*[0].
- Phần tử *a*[*n*-1] sẽ lưu 32 *bit* cuối cùng (các *bit* có trọng số lớn nhất – *bit* trái nhất).

- *Số bù chín và số bù mười:* Cho số thập phân *A* gồm *n* chữ số thập phân, ta có:
  - Số bù chín của *A* là $\left(10^n - 1\right) - A$.
  - Số bù mười của *A* là $\left(10^n\right) - A$. Số bù mười của *A* = (Số bù chín của *A*) + 1.

Thí dụ 2.1. Với *n* = 4 và *A* = 3265 thì:

- Số bù chín của *A* là $\left(10^4 - 1\right) - \left(A\right)$ = 9999 - 3265 = 6734.
- Số bù mười của *A* là $\left(10^4\right) - \left(A\right)$ = 10000 - 3265 = 6735.

- *Số bù một và số bù hai:* Cho số nhị phân *A* được biểu diễn bằng *n bit*. Ta có:
  - Số bù một của *A* là $\left(2^n - 1\right) - A$.
  - Số bù hai của *A* là $\left(2^n - A\right)$. Số bù hai = số bù một + 1.

Thí dụ 2.2. Với *n* = 8 *bit*, cho *A* = 0010 0101 thì:

- *Số bù một của A* là $\left(2^8 - 1\right) - \left(A\right)$ = 1111 1111 - 0010 0101 = 1101 1010.
- *Số bù hai của A* là $\left(2^8\right) - \left(A\right)$ = 1 0000 0000 - 0010 0101 = 1101 1011.

Số âm (số nguyên có dấu) được thể hiện theo *số bù hai ở trên*. Khi đó nếu *bit* ngoài cùng bên trái bằng 1 thì sẽ thể hiện là số âm, ngược lại nếu bằng 0 thì là số *dương*.

Với cách tổ chức và biểu diễn theo kiểu này ta sẽ dễ dàng sử dụng các *thao tác bit* như AND, OR, NOT, XOR, dịch *bit*... để thực hiện các phép toán số học.

Thí dụ 2.3. Giả sử có số *a* = 1844674407370955161146744. Dạng nhị phân của *a* là

"000…0011000011010011111111111111111111111111111111111111111111111111111111100100010101011110000"

Khi đó chuỗi nhị phân của số dương *a* được lưu trữ trong đối tượng số lớn như sau

a[*n*-1]	*a*[2]	*a*[1]	*a*[0]
000…000	00000000000000110000110100111111	11111111111111111111111111111111	11111111111100100010101011110000
32 bit	32 bit	32 bit	32 bit

Hình 2.2. Minh họa cấu trúc và cách biểu diễn chuỗi bit của số lớn không dấu a

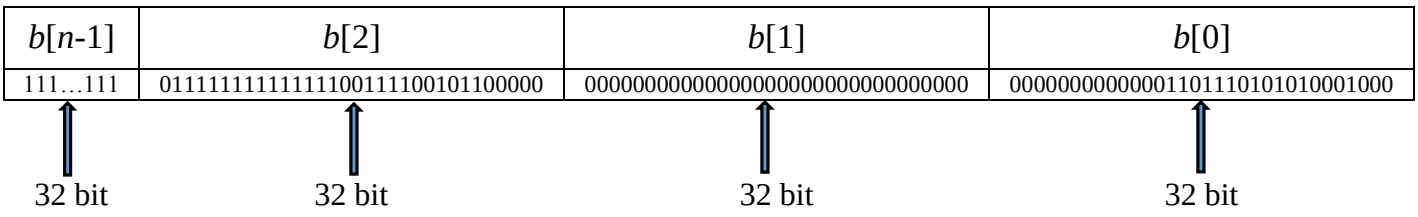
Theo cách biểu diễn này thì *Lengthdata* = 3. Như vậy, với số *dương a* trên thì nếu chiều dài tối đa của không gian biểu diễn (mảng *Data*) là *n* thì biểu diễn được tối đa 32*n bit* và do *a* là số không dấu nên *bit* ngoài cùng bên trái (vị trí *a*[*n*-1]) bằng 0.

Thí dụ 2.4.

Giả sử có số *b* = -184467440737095516114674 4. Dạng nhị phân của *b* là

"1...11101111111111111100111100101100000000000000000000000000000000000000000000000000000000000011011101010100010 00".

Khi đó chuỗi nhị phân của số có dấu *b* được lưu trữ trong đối tượng số lớn như sau



Hình 2.3. Mô phỏng cách biểu diễn chuỗi bit của số có dấu lớn b

Theo thí dụ này thì *Lengthdata* = *n*. Như vậy, với *số âm* (số nguyên có dấu) *b* ở trên thì nếu chiều dài tối đa của không gian biểu diễn (mảng *Data*) là *n* thì có thể biểu diễn được tối đa 32.*n bit* và do *b* là *số âm* nên *bit* ngoài cùng bên trái (vị trí *a*[*n*-1]) = 1.

Trong ngôn ngữ lập trình C#, khi xem các giá trị tại các *b*[*i*] với *i* = [0...*n*-1] thì sẽ hiển thị giá trị thực thập phân theo *index* từ 0 đến 31 như *Thí dụ 2.5* dưới đây.

Thí dụ 2.5. Hiển thị giá trị thập phân trong mỗi phần tử của mảng

b[0] = 00000000000001101110101010001000 = 453256.

b[1] = 00000000000000000000000000000000 = 0.

b[2] = 11111111111111100111100101100000 = 4294867296.

.....

b[*n*-1] = 11111111111111111111111111111111 = 4294967295 = $2^{32} - 1$.

Để khởi tạo giá trị cho các số lớn ta xây dựng các hàm khởi tạo áp dụng phương thức nạp chồng (Overloading).

2.1.3. Các hàm khởi tạo số nguyên lớn

Các hàm khởi tạo đối tượng số lớn trong luận văn này sử dụng sử dụng ngôn ngữ lập trình C# (Version Studio 2013) để thực hiện.

Function 2.1 (*Hàm khởi tạo đối tượng số lớn không tham số*).

Để khởi tạo đối tượng số lớn không tham số ta thực hiện khai báo như sau

```
Public BigNumInteger()
{
   Data = new uint[MaxLength];
   LengthData = 1;
};
```

Function 2.2 (*Hàm khởi tạo đối tượng số lớn với tham số là số lớn BigNumInteger*).

Tạo đối tượng số lớn bằng cách sao chép dữ liệu từ đối tượng số lớn truyền vào bao gồm chiều dài và mảng *Data* tương ứng. Ta thực hiện khai báo như sau

```
Pubic BigNumInteger (BigNumInteger BigNum)
{
   Data = new uint[MaxLength];
   LengthData = BigNum.LengthData;
   For (int i =0; i < LengthData; i++)
   Data[i] = BigNum.Data[i];
};
```

Function 2.3 (*Hàm khởi tạo đối tượng số lớn với tham số vào kiểu long*).

Ban đầu khởi tạo thuộc tính *Data* với số *index* lớn nhất có thể ban đầu. Tiếp theo sử dụng phép toán AND *bit* của số đầu vào (kiểu *long*) với mặt nạ (mask) để "cắt" 32 *bit* đầu tiên, đưa 32 *bit* đầu tiên này vào *index* =0 của mảng *Data*.

Tiếp theo, ta dịch phải 32 *bit* để "cắt" *bit* tiếp theo đưa vào mảng *Data* có *index* chạy từ 1, rồi cứ thế tiếp tục cho tới hết số *long* ban đầu. Ta khởi tạo như sau

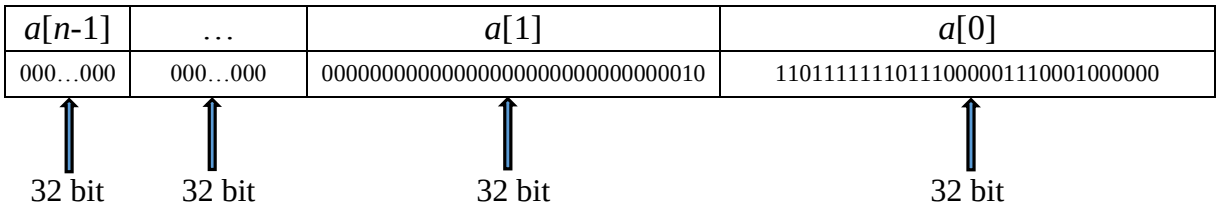
```
pubic BigNumInteger (long value)
{
   Data = new uint[MaxLength];
   long tempVal = value;
   LengthData = 0;
   While (value != 0 && LengthData < MaxLength)
   {
      Data[LengthData]=(uint)(value & 0xFFFFFFFF);
      value >>= 32;
      LengthData++;
   }
};
```

Thí dụ 2.6. Khởi tạo số lớn *a* = 12345678912.

Ta thực hiện gọi hàm khởi tạo như sau

BigNumInteger a = new *BigNumInteger*(12345678912).

Khi đó đối tượng số lớn *a* được tổ chức như sau



Hình 2.4. Minh họa cách tổ chức lưu trữ dữ liệu của số lớn a

Với mảng biểu diễn trên thì thuộc tính *LengthData* = 2. Thể hiện thập phân trong C# khi view lần lượt là *a*[0] = 3755744320 và *a*[1] = 2.

Function 2.4 (*Hàm khởi tạo đối tượng số lớn với tham số vào kiểu String*).

Đối với số thập phân, chẳng hạn số 123456 thì số bên trái hơn số bên phải 10 lần bắt từ bên phải nhất. Có nghĩa 123456 = 100000+20000+3000+400+50+6. Do đó, để đưa một chuỗi số dạng *String* vào đối tượng số lớn thì ta thực hiện duyệt từng ký tự số từ phải sang trái cho đến khi hết chuỗi số, mỗi ký tự số duyệt được sẽ chuyển sang dạng số sau đó cộng với phần tử bên trái sau khi đã nhân 10 lần. Nếu phần tử trái nhất của chuỗi mà là ký tự "-" thì đổi dấu.

Ta thực hiện khởi tạo như sau

```
Public BigNumInteger (String value, int radix)
{
  BigNumInteger multiplier = new BigNumInteger(1);
  BigNumInteger result = new BigNumInteger();
  int limit = 0;
  If ( value[0] == '-' ) limit = 1;
  For (int i = value.Length – 1; i >= limit; i--)
   {
    int posVal = (int)value[i];
    If (posVal >= '0' && posVal <= '9')posVal = posVal - '0';
    If (postVal <= radix)
     {
       If (value[0] = '-' )
       posVal = - posVal;
       result = result + (multiplier * posVal);
       if ((i - 1) >= limit) multiplier = multiplier *radix;
     }
    }
```

```
    Data = new uint[MaxLength];
    For (int i = 0; i < result.LengthData; i++)
         Data[i] = result.Data[i];
    LengthData = result.LengthData;
};
```

Phép cộng (+), phép trừ (-), phép nhân (*), các phép so sánh... đã được nạp chồng (Overload) trong ngôn ngữ C# để sử dụng cho số tính toán với số lớn.

Thí dụ 2.7. Khởi tạo số lớn *a* từ chuỗi "123456879548545454564654121 2".

BigNumInteger a = new *BigNumInteger*("123456879548545454564654121 2", 10)

2.2. Các phép toán trên số lớn

Các phép toán xử lý số lớn trong luận văn này sử dụng các phép thao tác trên các *bit*. Ta có một số các phép thao tác *bit* cơ bản sau đây: Phép cộng (+) , phép trừ (-), phép nhân (*), phép AND, phép OR, phép XOR, phép NOT, dịch trái, dịch phải.

2.2.1. Phép gán giá trị ngầm cho đối tượng số lớn

Trong ngôn ngữ C# cho phép khai báo gán ẩn (*Implicit*) cho phép gán giá trị cho một đối tượng. Cụ thể ta kết hợp phép gán ẩn với việc khởi tạo số lớn đã nêu ở *Mục 1.3 của Chương II* luận văn này.

Function 2.5 (*Hàm định nghĩa phép gán đối tượng số lớn a với kiểu long*).

```
Public static implicit operator BigNumInteger (long value)
{
  Return (new BigNumInteger(value));
};
```

Thí dụ 2.8. Khởi tạo gán ngầm định kiểu *long*: *BigNumInteger x* =12345678912.

Function 2.6 *(Định nghĩa gán đối tượng số lớn a với kiểu String*).

```
Public static implicit operator BigNumInteger(String value)
{
  return (new BigNumInteger(value, 10));
};
```

Thí dụ 2.9. Khởi tạo gán ngầm kiểu *String*: *BigNumInteger* a = "123456879548".

2.2.2. Phép cộng hai số lớn không dấu

Việc cộng hai số lớn không dấu (unsigned) là việc cộng nhóm 32 *bit* của hai số

lớn *a* và *b* từ bên phải qua. Nếu mỗi lần cộng có dư *bit* thì sẽ cộng phần dư *bit* này sang phần cộng 32 nhóm *bit* tiếp theo. Cuối cùng trả về số lớn *c* là kết quả của phép cộng giữa *a* và *b*. Đối tượng số lớn *a* và *b* có thuộc tính *Data* như sau:

a[*n*-1]	*a*[*n*-2]	.....	*a*[1]	*a*[0]

b[*n*-1]	*b*[*n*-2]	.....	*b*[1]	*b*[0]

Hình 2.5. Minh họa biểu diễn cấu trúc thuộc tính Data của hai số lớn a và b

Số *a* và *b* có cùng *n* là độ dài lớn nhất khởi tạo ban đầu (*Capacity*), số *a* có thuộc tính *a.LengthData* và *b.LengthData* (*a.LengthData* < *n*-1 và *b.LengthData* < *n*-1). Việc cộng hai số *a* và *b* được thực hiện như các bước sau:

- **Bước 1:** Khởi tạo số lớn lưu kết quả, giả sử là *Result* và *Result.LengthData*= Max(a.*LengthData*, *b.LengthData*), khởi tạo biến nhớ *c* = 0;
- **Bước 2:** Duyệt lần lượt từ vị trí *i* = [0] đến vị trí *i*= [*Result.LengthData* -1] thực hiện phép cộng *bit* và cộng thêm *bit* nhớ *c* của kết quả trước.

  Trong đó biến lưu kết quả tổng trung gian (sum) có kiểu dữ liệu là *long* đủ để lưu phép cộng hai số nguyên có kiểu *uint.*

  - Phần tử vị trí thứ 0: *c* = 0, *sum* = 0; //(*Sum* có kiểu dữ liệu là *long*)
    - *sum* = *a*[0]+ *b*[0] + *c; //Thực hiện phép cộng nhóm 32 bit theo từng vị trí.*
    - *c* = *sum* >> 32; //*Dịch phải 32 bit để lấy bit dư khi cộng a[0] và b[0].*
    - *Result.Data*[0] = *sum* & *0xFFFFFFFF*; // *Đưa 32 bit vào mảng kết quả..*
  - Tiếp cho phần tử thứ nhất:
    - *sum* = *a*[1]+ *b*[1] + *c*; //*c này được tính ở bước trên*
    - *c* = *sum* >> 32;
    - *Result.Data*[1] = *sum* & *0xFFFFFFFF*; // *Đưa 32 bit vào kết quả*

    .....
  - Phần tử thứ *i: với i=Result.LengthData -1.*
    - *sum* = *a*[*i*]+ *b*[*i*] + *c; //c này được tính ở bước trên (i-1)*
    - *c* = *sum* >> 32;
    - *Result.Data*[*i*] = *sum* & *0xFFFFFFFF*; // *Đưa 32 bit vào kết quả*
- **Bước 3:** Nếu ở Bước 2 đến vị trí *Result.LengthData -1* mà *c* =1 thì đưa vào vị trí có trọng số cao nhất trong *Result.Data*.

- **Bước 4:** Trả về đối tượng số lớn *c* kết quả là tổng của *a* và *b* ban đầu.

Algorithm 2.1 (*Cộng hai số lớn không dấu*).

```
Input: Two big number integer same a, b.
Output: Return big number integer with r = a + b.
  1. r.LengthData ← Max(a.LengthData, b.LengthData), c ← 0;
  2. For i from 0 to r.LengthData – 1 Step 1 do the following:
     2.1. sum ← a.Data[i] + b.Data[i] + c;
     2.2. c ← (sum >> 32);
  3. If (c != 0 && r.LengthData < MaxLength) then do:
     3.1. r.Data[r.LengthData] ← c;
     3.2. r.LengthData++;
  4. Return(r).
```

- *Đánh giá độ phức tạp*

Giả sử chiều dài lớn nhất các *bit a* và *b* là *n* thì khi đó phép cộng hai số lớn *a* và *b* là phép cộng liên tiếp các cặp *bit* của *a*, *b* và thêm phép cộng *bit* nhớ *c*. Do đó cần *n* phép cộng *bit*. Vì thế, độ phức tạp của thuật toán cộng *bit* này là O(*n*).

Thí dụ 2.10. Hai số lớn *a* =12345678932 và *b* =987654321286 biểu diễn như sau

a[*n*-1]	…	*a*[1] (2)	*a*[0](3755744340)
000…000	000…000	00000000000000000000000000000010	11011111110111000001110001010100

b[*n*-1]	…	*b*[1] (229)	*b*[0](4106810502)
000…000	000…000	00000000000000000000000011100101	11110100110010001111010010000110

Hình 2.6. Minh họa biểu diễn số lớn a và b cho thuật toán cộng

Trong đó: *n* là dung lượng lớn nhất của mảng *a* và *b* và chiều dài của *a* và *b* đều < *n*.

Bảng 2.1. Mô tả các bước thực hiện phép cộng hai số lớn không dấu a và b

[*n*-1]	…	[3]	[2]	[1]	[0]	
0	0	0	0	2	3755744340	*a*
0	0	0	0	229	4106810502	*b*
0	0	0	0	1	0	*c*
0	0	0	0	232	3567587546	*result*

Result[*n*-1]	…	*Result*[1] (232)	*Result*[0](3567587546)
000…000	000…000	00000000000000000000000011101000	11010100101001010001000011011010

Hình 2.7. Minh họa biểu diễn kết quả phép cộng hai số lớn không dấu

2.2.3. Phép trừ hai số lớn không dấu

Như trong cách biểu diễn số nguyên ta đã có khái niệm *bù chín, bù mười* đối với số thập phân và phương pháp *bù hai* với số nhị phân. Việc thực hiện trừ hai số lớn a và b được thực hiện như sau:

- *Đảm bảo số a và b có độ dài như nhau. Phần nào còn thiếu ở số trừ thì thêm số 0.*
- *Cộng số bị trừ với số bù mười của số trừ.*
- *Bỏ bít nhớ cuối cùng của kết quả ta có kết quả của phép trừ giữa hai số a và b.*

Thí dụ 2.11. Có số lớn *c* = 56789 – 456, số bị trừ là 56789, số trừ là 456

- Thêm số 0 vào số trừ cho đủ chiều dài với số 56789: 00456
- Lấy *bù mười* của số trừ: 99544.
- Cộng số bị trừ với *bù mười* của số trừ: 56789 + 99544 = 156333.
- Bỏ số nhớ cuối cùng (số 1) ta được kết quả là: 56333.

Tuy nhiên, số lớn *a* và *b* được biểu diễn với cấu trúc dữ liệu và xử lý trong luận văn này sử dụng các phép toán thao tác trên dãy *bit* nhị phân nên ta áp dụng như sau:

- *Ta chuyển số trừ b thành số bù hai (theo phương pháp bù hai).*
- *Lấy số a cộng số bù hai của số b.*

Algorithm 2.2 (*Lấy bù hai của số lớn BigNuminteger*).

```
Input: Object big integer n to get complement.
Output: Object big integer r = -(n)
  1. r ← n;
  2. For i from 0 to MaxLength - 1 step 1 do the following
     r.Data[i] ← (~(n.Data[i]));   //1'complement.
  3. val ← 0, c ← 1, index ← 0;    //Get 2'complement.
  4. While c != 0 and  index < MaxLength, do the following:
     4.1. val ← r.Data[index];
     4.2. val++;
     4.3. r.Data[index] = val & 0xFFFFFFFF;
     4.4. c ← val >> 32; index++;
  5. r.LengthData ← MaxLength;
     While r.LengthData > 1
     and r.Data[r.LengthData - 1] = 0 do
     r.LengthData--;
6. Return (r). // Kết quả phép lấy bù n
```

Algorithm 2.3 (*Giải thuật trừ hai số lớn a và b*).

```
Input: Two big unsign integer a, b.
Output: Big integer r = a – b.
  1. c ← -(b); //Using Algorithm 2.2 of chapter II.
  2. r ← a + c; //Using Algorithm 2.1 of chapter II.
  3. Return (r).
```

- *Đánh giá độ phức tạp***:** $O(n)$. Với *n* là chiều dài dãy *bit* truyền vào.

Thí dụ 2.12. Cho *a* = 56789, *b* = 456. Tính *Result* = *a* - *b* ? Số *a* và *b* được biểu diễn chi tiết như sau

a[*n*-1]	…	*a*[1]	*a*[0](56789)
000…000	000…000	000…000	00000000000000001101110111010101

Hình 2.8. Minh họa cách tổ chức, biểu diễn số bị trừ a

b[*n*-1]	…	*b*[1]	*b*[0](456)
000…000	000…000	000…000	00000000000000000000000111001000

Hình 2.9. Minh họa cách tổ chức, biểu diễn số trừ b

Lấy *bù hai* theo *Algorithm 2.2* của số *b*, ta được đối tượng *BigNum* như sau

a[*n*-1]	…	*a*[1]	*a*[0](4294966840)
111…111	111…111	111…111	11111111111111111111111000111000

Hình 2.10. Minh họa cách tổ chức, biểu diễn số bù hai của số trừ BigNum.

Thực hiện cộng *a* và *BigNum* (số bù hai của *b*). *Result = a + BigNum.*

Dùng giải thuật *Algorithm 2.1*. Ta được kết quả là 56333, biểu diễn như sau

Result[*n*-1]	…	*Result*[1]	*Result*[0](56333)
000…000	000…000	000…000	00000000000000001101110000001101

Hình 2.11. Minh họa cách tổ chức, biểu diễn kết quả của phép trừ.

2.2.4. Phép nhân hai số không dấu

Có nhiều giải thuật để nhân nhanh hai số, trong [1] có đưa ra giải pháp nhân nhanh hai số *a* và *b* điển hình. Tuy nhiên, trong luận văn này sử dụng giải thuật nhân nhanh Ấn Độ. Tư tưởng của giải thuật Ấn Độ thực hiện phép nhân hai số tự nhiên *a* và *b* như sau:

- Chia *a* cho 2 và nhân *b* cho 2. Lặp tới khi không chia *a* cho 2 được nữa.
- Ở bước chia *a* cho 2 nào mà *a* lẻ thì sẽ cộng *b* tương ứng vào kết quả.

Thí dụ 2.13. Với bài toán trên ta có *a* = 66 và *b* = 25. Tính *k* = *ab* ?

Bảng 2.2. Mô tả các bước nhân hai số a và b theo thuật toán Ấn Độ

a	***b***	***k***	
33	50	50	***a* chia 2 lẻ**
16	100	0	
8	200	0	
4	400	0	
2	800	0	
1	1600	1600	***a* chia 2 lẻ**
0	3200	0	Dừng
Kết quả		**1650**	

Như vậy, ta có công thức tính tổng quát kết quả phép nhân như sau

$$k = ab = \begin{cases} \dfrac{2ba}{2}, & \text{nếu } a \text{ chẵn} \\ \dfrac{2b(a-1)}{2} + b, & \text{nếu } a \text{ lẻ.} \end{cases}$$

Algorithm 2.4 (*Nhân hai số lớn a và b không dấu*).

```
Input: Two object big unsigned integer a, b.
Output: Object big unsigned integer Result = ab.
  1. While a ≠ 0 then do the following:
     1.1. If  a is odd then Result ← Result + b.
     1.2. a ← a right shift 1;    //Chia a cho 2
     1.3. b ← b left shift 1;     //Nhân b với 2
  2. Return (Result).
```

- *Đánh giá độ phức tạp:* Thật vậy theo thuật toán nhân ở trên, *a* và *b* đều được biểu diễn dưới dạng dãy *bit*. Ta gọi T(*n*) là thời gian thực hiện khi $a \neq 0$. $T\left(\frac{n}{2}\right)$ là thời thực hiện khi $\left(\frac{a}{2}\right)$. Ta coi phép chia 2 và nhân 2 là tương đương. Khi đó, trong thân vòng lặp sẽ thực hiện $2T\left(\frac{n}{2}\right)$, ngoài ra còn có một phép cộng khi *a* lẻ. Ta có công thức đệ quy $T(n) = 2T\left(\frac{n}{2}\right) + n$ (*) . Giải phương trình đệ quy (*), ta có

$$T(n) = 2T\left(\frac{n}{2}\right) + n$$

$$= 2\left[2T\left(\frac{n}{4}\right) + \frac{n}{2}\right] + n = 4T\left(\frac{n}{4}\right) + 2n$$

$$= 4\left[2T\left(\frac{n}{8}\right) + \frac{n}{4}\right] + 2n = 8T\left(\frac{n}{8}\right) + 3n$$

.....

$T(n) = 2^k T\left(\frac{n}{2^k}\right) + kn$. Giả sử $n = 2^k \Rightarrow k = \log(n)$.

$T(n) = 2^k T(1) + kn = 2^k + kn = n + n\log(n)$.

Vậy $T(n) \in O(n\log(n))$.

2.2.5. Phép chia hai số lớn không dấu

Xét phép chia *a* cho *b* ta có $a = bt + d$, trong đó *t* là thương, còn *d* là số dư của phép chia. Trong luận văn này sẽ sử dụng giải thuật chia Ấn Độ để áp dụng cho phép của *a* cho *b*. Tư tưởng của giải thuật Ấn Độ như sau:

- Liên tục nhân 2 giá trị của *b* cho đến khi $b \geq a$.
- Liên tục chia *b* cho 2 cho đến khi bằng giá trị ban đầu. Số lần chia và nhân 2 rồi cộng thêm 1 là thương của phép chia, còn phần dư $d = a - bt$.

Theo thuật toán trên việc chia số *a* cho *b* ở trên đơn giản chỉ tự nhân 2, tự chia 2, phép tính cộng và phép trừ đơn thuần. Với cấu trúc dữ liệu lưu trữ của số lớn việc chia 2 một số nguyên *a* là việc dịch phải một lần và nhân 2 là dịch trái một lần.

Algorithm 2.5. (*Chia số lớn không dấu a cho b*)

```
Input: Two object big unsigned integer a, b.
Output: Object big integer t is quotient and d is remainder.
  1. v ← b;
  2. While v <= a then do the following: v ← 2v;
  3. If |v > a|)  then do the following: t ← 0,  d ← a;
  4. While (v > b) do the following:
     4.1. v ← v/2; t ← 2t;
     4.2. If v <= d then do the following: d = d - v, ++t;
  5. Return (t, d).
```

Thí dụ 2.14. Cho số *a* = 75, *b* = 7. Tính phép *a* chia cho *b*.

Áp dụng *Giải thuật 2.5*, ta mô tả các bước của giải thuật theo các bước như bảng sau

Bảng 2.3. Mô tả các bước chia hai số a và b theo thuật toán Ấn Độ

Độ lớn (*a*)	Độ lớn (*b*)	Thương (t)	Dư (d)	Ghi chú
75	7			
	14			
	28			
	56			
	112	0	75	Dừng vì 112 > 75
	56	1	19	19 = 75-56
	28	2	19	19 vì 28 > 19
	14	5	5	5 = 19-14
	7	10	5	Dừng do 7 = 7

- *Đánh giá độ phức tạp*

Theo giải thuật trên để chia số *a* cho số lớn *b* thực chất chỉ là phép tự nhân 2, tự chia 2, phép + và phép -.

Ta gọi $T(n)$ là thời gian thực hiện phép chia của số *a* và *b*. Thời gian để thực hiện phép nhân 2 là $T\left(\frac{n}{2}\right)$, phép chia 2 trong thân vòng lặp while là $T\left(\frac{n}{2}\right)$.

Thời gian để thực hiện phép dịch *bit* với *t* là *n*. Ngoài ra còn một phép so sánh, như vậy ta có: $T(n) = 2T\left(\frac{n}{2}\right) + 2n$. (**)

Giải phương trình đệ quy (**), ta có

$$T(n) = 2T\left(\frac{n}{2}\right) + 2n$$

$$= 2\left[2T\left(\frac{n}{4}\right) + n\right] + 2n = 4T\left(\frac{n}{4}\right) + 4n$$

$$= 4\left[2T\left(\frac{n}{8}\right) + \frac{n}{2}\right] + 4n = 8T\left(\frac{n}{8}\right) + 6n$$

.....

$T(n) = 2^k T\left(\frac{n}{2^k}\right) + 2kn$. Giả sử $n = 2^k \Rightarrow k = \log(n)$.

$T(n) = 2^k T(1) + 2kn = 2^k + 2kn.$

$T(n) = n + 2n\log(n)$. Vậy, $T(n) \in O(n\log(n))$.

Như vậy, để lấy phần dư của phép chia của *a* cho *b* ta cũng dựa vào giải thuật trên. Tuy nhiên ở Bước 5 ta chỉ cần return *d*, với *d* là phần dư khi chia số *a* cho số *b*.

2.2.6. Phép lũy thừa

Ta có hai số nguyên lớn *x* và *n*. Để tính x^n ta sử dụng phương pháp tính nhanh , đó là *phương pháp bình phương và nhân.*

Algorithm 2.6 (*Thuật toán bình phương và nhân – Square and multiply algorithm*).

Ta sử dụng thuật toán *bình phương và nhân* theo phương pháp *Left-to-right binary exponentiation***.** Duyệt từ *bit* có trọng số thấp đến *bit* có trọng số cao.

Input: Two big integer *g* (base number) and $e = (e_{k-1}...e_0)$.

Output: Big integer result $r = g^e$.

```
1. r ← 1;
2. For  i from k - 1 downto 0 step -1
3. r ← r^2;                        // Square
4. If e_i = 1 then r ← rg;         // Multiply (optional)
5. Return (r).
```

Thí dụ 2.15. Tính 2^{64}, trong đó: $g = 2$, $e = 64 = (1000000)_2$.

Bảng 2.4. Mô tả các bước thực hiện tính lũy thừa g^e

i	6	5	4	3	2	1	0
e_i	1	0	0	0	0	0	0
r^e	2	4	16	256	65536	4294967296	18446744073709551616

2.2.7. Xác định ước chung lớn nhất

Ta sử dụng thuật toán Euclid được trình bày tại *Algorithm 1.7* để tìm ước chung lớn nhất của hai số nguyên lớn cho trước. Ta thực hiện cài đặt như sau

Algorithm 2.7 (*Xác định ước chung lớn nhất của hai số nguyên lớn*).

Input: Two positive big integer *a* and *b*.

Output: Big integer *g* = gcd(*a*, *b*).

1. *g* ← *y*;
2. While (*x.LengthData* > 1 || (*x.LengthData* = 1 and *x.Data*[0] != 0)) to do the following:
   2.1. *g* ← *x*;
   2.2. *x* ← *y* mod *x*; //chia lấy dư
   2.3. *y* ← *g*;
3. Return *g*.

2.2.8. Phép cộng theo *modulo*

Algorithm 2.8 (*Cộng hai số nguyên lớn theo modulo n*).

Input: Three big positive integer *a*, *b* and *n*.

Output: Big integer *R* = (*a* + *b*) (mod *n*).

1. If *a* > *n* then do the following *a* ← *a* mod *n*;
2. If *b* > *n*) then do the following *b* ← *b* mod *n*;
3. *R* ← (*a* + *b*);
4. If *R* < *n* then Return (*R*); otherwise Return (*R* - *n*).

2.2.9. Phép nhân theo *modulo*

Việc nhân hai số lớn *a*, *b* theo *modulo n* trong luận văn này được thực hiện theo *phương pháp bình phương và nhân* đã trình bày tại *Algorithm 2.6*.

Algorithm 2.9 (*Nhân hai số nguyên lớn theo modulo n*).

Input: Three big positive integer *a*, *b* and *n*.

Output: Big integer *R* = (*ab*)(mod *n*).

1. If *a* = *n* || *b* = *n* then Return (0);
2. If *a* > *n* then do *a* ← *a* mod *n*;
3. If *b* > *n* then do *b* ← *b* mod *n*;
4. While *a* != 0 then do the following:
   //AddMod use Algorithm 2.8.
   4.1. If *a* is odd then *R* ← AddMod(*R*, *b*, *n*);
   4.2. $a \leftarrow \frac{a}{2}$;
   4.3. *b* ← AddMod(*b*, *b*, *n*);
5. Return (*R*).

2.2.10. Phép cộng các số lớn có dấu

Với cách biểu diễn số lớn với cấu trúc lưu trữ liên tiếp 32 *bit* trong thuộc tính *Data*, với cách biểu diễn này sẽ biểu diễn cho cả số lớn không dấu và có dấu (signed). Do đó việc cộng hai số lớn *a* và *b* có dấu thực chất vẫn là phép cộng các *bit*.

Giải thuật cộng hai số lớn có dấu vẫn sử dụng được *Algorithm 2.1 của Chương II* này.

Thí dụ 2.16. Cho *a* = 12456789987654321 và *b*= -12456789987654321123456.

Tính *r = a + b*. Số lớn *a*, *b* (*số có dấu*) được biểu diễn lần lượt như sau:

a[*n*-1]	…	*a*[1](2900322)	*a*[0](1849785009)
000…000	000…000	00000000001011000100000101100010	01101110010000010111101010110001

Hình 2.12. Minh họa cách biểu diễn số nguyên có dấu a

b[*n*-1]	*b*[2](4294966620)	*b*[1](3075461409)	*b*[0](1570688896)
111…111	11111111111111111111110101011100	10110111010011111101000100100001	01011101100111101100111110000000

Hình 2.13. Minh họa cách biểu diễn số nguyên có dấu b

Trong đó, *n* là dung lượng (capacity) lớn nhất mà có thể lưu trữ của mảng *Data* của *a* và *b* và chiều dài của *a* và *b* đều < *n*. Bảng sau mô tả các bước của phép cộng

Bảng 2.5. Mô tả các bước thực hiện phép cộng hai số lớn có dấu a và b

[*n*-1]	…	[3]	[2]	[1]	[0]	
0	0	0	0	2900232	1849785009	*a*
4294967295	4294967295	4294967295	4294966620	3075461409	1570688896	*b*
0	0	0	0	0	0	*c*
4294967295	4294967295	4294967295	4294966620	3078361731	3420473905	*result*

Theo trên ta thấy tổng của *a* và *b* với *LengthData = n* và *Data* được biểu diễn

r[*n*-1]	*r*[2](4294966620)	*r*[1](3078361731)	*r*[0](3420473905)
111…111	11111111111111111111110101011100	10110111011111000001001010000011	11001011111000000100101000110001

Hình 2.14. Minh họa biểu diễn kết quả phép cộng hai số lớn có dấu

2.2.11. Phép trừ hai số lớn có dấu

Cũng như phép cộng có dấu và với cách biểu diễn dữ liệu số lớn cho cả *số âm* (có dấu) nên việc trừ hai số lớn có dấu thực chất là cộng hai số lớn.

Algorithm 2.10 (*Trừ hai số nguyên lớn có dấu*).

```
Input: Two object big integer signed a and b.
Output: Object big integer r = a - b.
```

1. Using Algorithm 2.2 to do the following: $b \leftarrow (-b)$;
2. Using Algorithm 2.1 to do the following: $r \leftarrow (a + b)$;
3. Return (r).

2.2.12. Phép nhân số lớn có dấu

Phép nhân hai số lớn có dấu dựa trên phép nhân hai số lớn không dấu đã được trình bày ở *Algorithm 2.4 Chương II* này. Chỉ khác là việc lấy dấu cho kết quả của phép nhân. Ta có hàm lấy dấu của một số lớn như sau

Function 2.7 (*Hàm lấy dấu của số lớn*).

Input: Object big integer (include unsigned and signed) a.
Output: Return = 1 if positive and return -1 if nagetive.
If((a.*Data*[*MaxLength*-1] & 0x80000000)!= 0) Return -1 otherwise return 1.

Algorithm 2.11. (*Nhân hai số lớn có dấu*)

Input: Two object big integer signed a and b.
Output: Object big integer $r = ab$.

1. $x \leftarrow$ abs(a), $y \leftarrow$ abs(b), $r \leftarrow 0$; //abs-Get absolute.
2. If (x same sign y) then to do the following: $r \leftarrow xy$;
3. If (x and y not same sign) then to do the following:
   3.1. $R \leftarrow xy$; *//Using Algorithm 2.4*
   3.2. $R \leftarrow (-R)$; *//Using Algorithm 2.2*
4. Return (R).

Thí dụ 2.17. Cho x =10 và y = - 512.

1. r = 0, x = abs(10), y = abs(-512).
2. r = 10.512 = 5120.
3. r = -5120.

2.2.13. Phép chia hai số lớn có dấu

Phép chia (lấy phần nguyên) hai số lớn có dấu dựa trên phép chia hai số lớn không dấu đã được trình bày ở *Algorithm 2.5*. Chỉ khác là việc lấy dấu kết quả.

Algorithm 2.12 (*Chia hai số lớn có dấu lấy phần nguyên*).

Input: Two object big integer signed a and b.

Output: Object big integer $r = \frac{a}{b}$.

1. $x \leftarrow$ abs(a), $y \leftarrow$ abs(b), $r \leftarrow 0$; *//Get absolute of a, b.*
2. Div(x, y ,t ,d); *//Using Algorithm 2.5 to get quotient.*
3. If x same sign with y then do the following: $r \leftarrow t$;
4. If x not same sign with y then do the following:
   4.1. $r \leftarrow t$;
   4.2. $r \leftarrow (-r)$;*// Using algorithm 2.2.*
5. Return (r).

Phép chia (*lấy phần dư*) hai số lớn có dấu dựa trên phép chia hai số lớn không dấu đã được trình bày ở *Algorithm 2.5* trên. Chỉ khác là việc lấy dấu cho kết quả

Algorithm 2.13. (*Chia hai số nguyên lớn có dấu lấy phần dư*)

Input: Two object big integer signed a and b.
Output: Object big integer $r = a\%b$. (get remainder)

1. $x \leftarrow$ abs(a); *//Get absolute a*
2. $y \leftarrow$ abs(b); *//Get absolute b.*
3. $R \leftarrow 0$;
4. Div(x, y, t, d); *//Using Algorithm 2.5.*
5. If (x same sign with y) then to do $r \leftarrow d$;
6. If (x not same sign with y) then to do the following:
   4.1. $r \leftarrow d$;
   4.2. $r \leftarrow (-r)$; *// Using Algorithm 2.2.*
7. Return (r).

2.2.14. Phép so sánh hai số lớn

Phép so sánh hai số lớn được thể hiện cho cả số nguyên lớn có dấu và không dấu. Các số lớn *a* và *b* cần so sánh được biểu diễn dưới dạng mảng liên tiếp các 32 *bit* nhị phân bắt đầu từ *index* thứ 0 tới *n* (*MaxLength*).

Giả sử có số lớn *a* với chiều dài *len*1= *a.LengthData* và *len*2=*b.LenthData.*

Algorithm 2.14. (Greater - *so sánh lớn hơn giữa hai số lớn a và b*)

Input: Two object big integer a and b.
Output: True, if $a > b$ and False if otherwise.

```
1. If  a.sign() = -1 and b.sign() =1) then do return false;
   //Sign() using Function 2.7.
2. If a.sign() = 1 and b.sign() = -1) then return true;
3. If a.sign() = b.sign() then to do the following:
   3.1. Length ← Max(a.LengthData, b.LengthData);
   3.2. Pos ← MaxLength – 1;
   3.3. For Pos from Length-1, Pos >= 0 and a.Data[Pos] =
        b.Data[Pos], Pos--);
   3.4. If Pos >= 0 then to do the following:
           If a.Data[Pos] > b.Data[Pos] then return True
           otherwise return False;
4. Otherwise Return False.
```

Algorithm 2.15 (*Equal -so sánh bằng giữa hai số lớn a và b*).

```
Input: Two object big integer a and b.
Output: True nếu a = b, False if otherwise.
 1. If  a.LengthData != b.LengthData then do return False;
 2. For i from 0 to a.LengthData – 1 step 1 do the following
       If a.Data[i] ! = b.Data[i]) return False;
 3. Return True.
```

Algorithm 2.16 (*GreaterOrEqual - so sánh lớn hơn hoặc bằng giữa hai số lớn*).

```
Input: two object big integer a and b.
Output: True nếu a >= b, False if otherwise.
Return Greater(a, b)||Equal(a, b). //Algorithm 2.14, 2.15.
```

Algorithm 2.17 (*Smaller - so sánh nhỏ hơn giữa hai số lớn a và b*).

```
Input: two object big integer a and b.
Output: True nếu a < b, False if otherwise.
Return !GreaterOrEqual(a, b).//Algorithm 2.16.
```

Algorithm 2.18 (*SmallerOrEqual - so sánh nhỏ hơn hoặc bằng giữa hai số lớn*).

```
Input: two object big integer a and b.
Output: True nếu a <= b, False if otherwise.
Return !Greater(a, b).
```

Từ các giải thuật so sánh hai số lớn *a* và *b*. Ta sử dụng phương pháp nạp chồng toán tử trong ngôn ngữ lập trình C# để định nghĩa các phép so sánh.

Function 2.8 (*So sánh lớn hơn* (>)).

```
Public static bool operator >(BigNumInteger a,BigNumInteger b)
 {
     Return Greater(a, b);
 };
```

Function 2.9 (*So sánh lớn hơn hoặc bằng* (>=)).

```
Public static bool operator>=(BigNumInteger a,BigNumInteger b)
 {
     Return GreaterOrEqual(a, b);
 };
```

Function 2.10 (*So sánh nhỏ hơn* (<)).

```
  Public static bool operator<(BigNumInteger a,BigNumInteger b)
  {
     Return Smaller(a, b);
  };
```

Function 2.11 (*So sánh nhỏ hơn hoặc bằng* (<=)).

```
  Public static bool operator<=(BigNumInteger a,BigNumInteger b)
  {
     Return SmallerOrEqual(a, b);
  };
```

Function 2.12 (*So sánh bằng* (==)).

```
  Public static bool operator==(BigNumInteger a,BigNumInteger b)
  {
     Return Equal(a, b);
  };
```

Function 2.13 (*So sánh không bằng* (!=)).

```
Public static bool operator!=(BigNumInteger a,BigNumInteger b)
 {
     Return !Equal(a, b);
 };
```

2.3. Khả năng của lớp thư viện số lớn

Trong luận văn này với cấu trúc dữ liệu của số nguyên lớn, cùng với các phép toán số học đã được tối ưu. Thực tế thư viện đã được cài đặt để thực hiện xử lý các phép toán số học như: +, -, *, /, %, thao tác trên dãy *bit*, *lũy thừa*, *lũy thừa modulo*, các phép *so sánh*... trên các số nguyên có độ dài hàng nghìn chữ số với thời gian thực hiện tương đương với các phép toán thực hiện trên kiểu số nguyên chuẩn cho cả số có dấu và số không dấu.

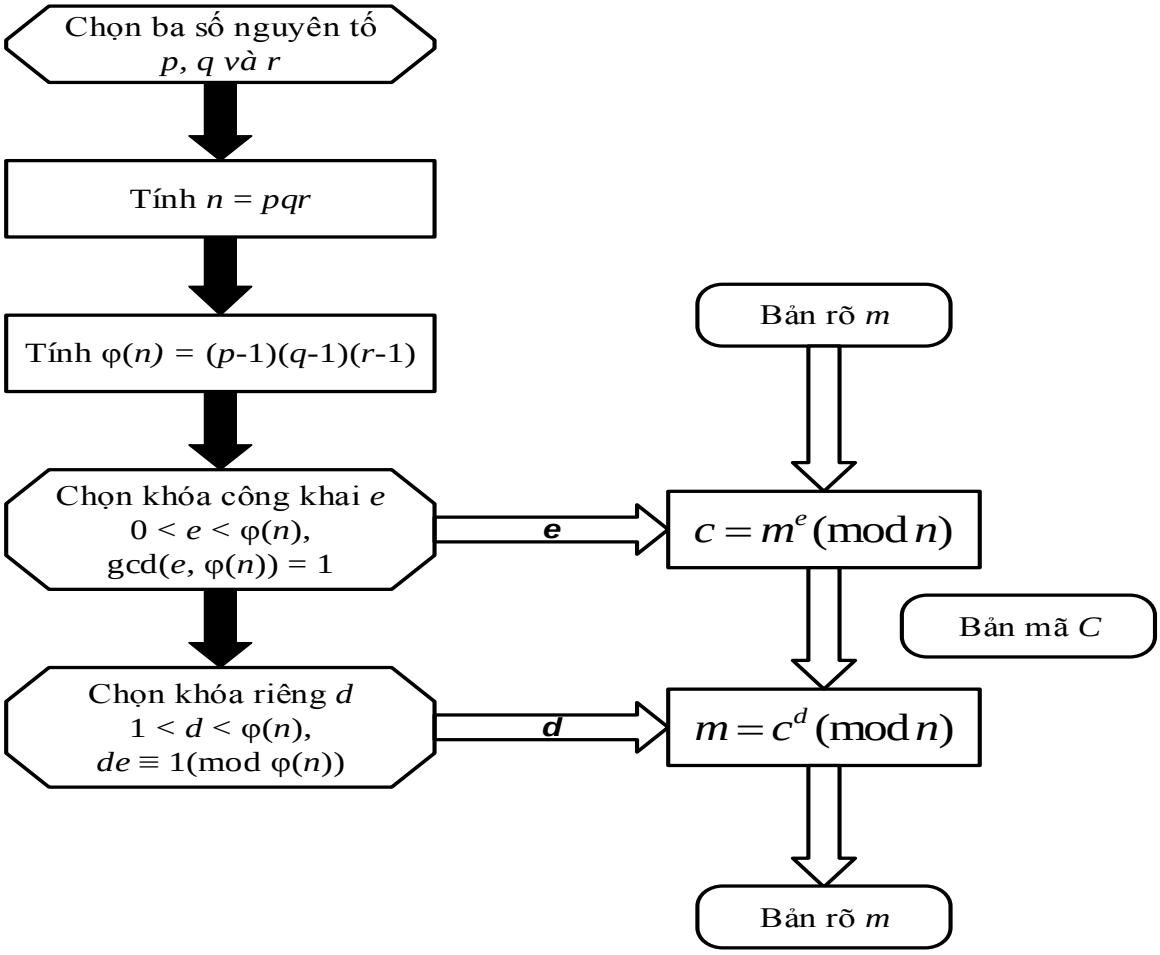
Ngoài ra, thư viện có khả năng mở rộng bổ sung thêm các phép toán trên số lớn phức tạp hơn từ các phép toán số học cơ bản đã xây dựng.

Chi tiết đánh giá hiệu năng của các phép tính toán được mô tả chi tiết ở chương tiếp theo sau đây.

CHƯƠNG 3
ỨNG DỤNG LỚP THƯ VIỆN SỐ NGUYÊN LỚN CHO HỆ RSA MỞ RỘNG

3.1. Phân tích các phép toán học sử dụng trong hệ mật mã multi-prime RSA

Như cơ chế hoạt động của RSA mở rộng với ba số nguyên tố *p, q* và *r* đã được trình bày tại Mục 1.8. Ta mô hình hóa cơ chế *multi-prime* RSA như sau:



Hình 3.1. Cơ chế hoạt động của hệ mật mã multi-prime RSA

Qua lưu đồ trên ta phân tích các phép xử lý toán học mà *multi-prime* RSA sử dụng

3.1.1. Chọn ba số nguyên tố đủ lớn phân biệt *p, q* và *r*

Ta sẽ dùng giải thuật Miller-Rabin đã trình bày tại *Algorithm 1.6* luận văn này để kiểm tra và sinh ba số nguyên tố lớn.

3.1.2. Tính *n = pqr*

Toán tử nhân ba số lớn *p, q, r* đã được sử dụng qua phương thức nạp chồng toán tử. Hàm $n = pqr$ là hàm một chiều, nếu *p, q, r* là các số lớn và có kích cỡ tương đương nhau cỡ 1024 *bit* trở lên thì *n* rất khó để phân tích ngược ra các số *p, q* và *r*.

3.1.3. Tính phi hàm $\varphi(n) = (p-1)(q-1)(r-1)$

Tương tự ở Bước 2, ở bước này tiếp tục sử dụng toán tử nhân và toán tử trừ. Có được $\varphi(n) = (p-1)(q-1)(r-1)$ là do ta sử dụng tính chất của *Phi hàm Euler* trong *Chương 1* luận văn này.

3.1.4. Tính khóa công khai *e*

Khóa công khai *e* được chọn sao cho $0 < e < \varphi(n)$ và $\gcd(e, \varphi(n)) = 1$. Tức *e* là nguyên tố cùng nhau với $\varphi(n)$. Chi tiết về cách tính *e* sẽ được trình bày ở mục dưới đây.

3.1.5. Tính khóa bí mật *d*

Ta tính khóa bí mật *d* sao cho $de \equiv 1 (\mathrm{mod}\, \varphi(n))$. Tức là *d* là phần tử nghịch đảo của *e* trong *modulo* $\varphi(n)$ hay $d = \frac{x.\varphi(n)+1}{e}$. Áp dụng thuật toán Euclid mở rộng để tìm *d* đã được trình bày theo *Algorithm 1.8*. Chi tiết về cách tính *d* được trình bày ở mục dưới đây của chương này.

3.1.6. Mã hóa và giải mã

Theo lưu đồ trên việc mã hóa và giải mã thông qua khóa *e* và *d*. Kết quả của việc mã hóa và giải là việc tính lũy thừa theo *modulo* (Modular Exponentiation). Việc tính lũy thừa theo *modulo* sẽ được trình bày ngay sau đây.

Ngoài ra, còn có việc thực hiện chuyển đổi bản rõ ban đầu *M* thành *m* để thực hiện mã hóa *c* và việc chuyển đổi từ bản mã *c* thành *m* rồi chuyển thành bản rõ ban đầu *M*.

3.2. Ứng dụng thư viện số lớn xây dựng hệ mật RSA (mở rộng)

Việc thực hiện xây dựng hệ mật RSA để mã hóa hoặc kết hợp với hàm băm để xác thực thông tin thì phải giải quyết được các phép toán đã phân tích như trên ngoài các phép toán xử lý số học đã được trình bày trong luận văn này.

3.2.1. Hàm kiểm tra và sinh số nguyên tố sát sau một số nguyên cho trước

Trong luận văn này sẽ sử dụng thuật toán Miller-Rabin để kiểm tra xác xuất tính chính xác một số nguyên lớn cho trước có phải là nguyên tố hay không?

Function 3.1 (*Hàm kiểm tra tính nguyên tố của số lớn theo Miller-Rabin*)

```
Input: Số lớn n cần kiểm tra và k là số lần thử.
Output: True nếu n là nguyên tố và False khi ngược lại.
Tên hàm: IsPrimeNumber(n, int k)
 1. If (n < 5) If ( n == 2 || n == 3) return True else False;
```

```
2. If (n.odd() = false) return false;   //Nếu n là số chẵn.
3. q, x, s, r;
4. //Phân tích (n-1) thành dạng 2^s.q với q là số lẻ
   While ((n-1) % (Pow(2,s+1)) == 0) s++; //Pow- Lũy thừa
5. q = (n-1)/(Pow(2, s));
6. For i = 1 to k step 1  //Kiểm tra k lần test.
   {
     BigNumInteger a = GetRandomInteger(2, n-1);
     x = ModPow(a, q ,n).//x=a^q(mod n)
     if (x != 1 && x!= (n-1)
       {
         For r = 1 to s -1 && x!(n-1) step 1
          {
             x = ModPow(x, 2, n); if (x == 1) return false;
          }
          If (x!( n-1 )) return false;
       }
     Return true; //Nếu đã vượt qua k lần test.
   };
```

Function 3.2 (*Hàm sinh số sát sau số nguyên lớn cho trước*).

```
Input: Số lớn n cần kiểm tra ban đầu và số lần kiểm tra k
Output: Số lớn n là nguyên tố liền sau số ban đầu.
Tên hàm: NextPrimeNumber(BigNum, int k)
//Nếu n chẵn thì tăng lên 1 => thành số lẻ
//Nểu n ban đầu lẻ thì tăng lên 2 để thanh số lẻ tiếp theo
  1. If (n.odd() = false) {n++;} else n = n + 2;//Nếu n chẵn
  2. While (!IsPrimeNumber(n, k))  n = n + 2;
  3. Return(n).
```

3.2.2. Tính khóa công khai *e* (public key)

Số *e* là khóa công khai và được chọn sao cho $0 < e < \varphi(n)$ và $gcd(e, \varphi(n)) = 1$ (tức là *e* là số nguyên tố cùng nhau với $\varphi(n)$. Do các số lớn được biểu diễn thành các chuỗi 32 *bit* liên tiếp nên ta cài đặt như sau để chọn *e*.

Function 3.3 (*Cài đặt hàm chọn khóa công khai e*).

```
Input: Số lớn φ(n) = (p − 1) (q − 1) (r − 1), chiều dài bit của khóa
e cần tạo, một đối tượng random (rand).
Output: Số lớn e (public key)
Tên hàm: GetPublickey(φ(n), bit, rand)
  1. BigNuminteger r, g; bool done = false.
  2. While (!done)
     {
       r.GenRandomBits(bit, rand);
       g = gcd(r, φ(n)); //Ước chung lớn nhất
       if (g.LengthData == 1 && g.Data[0] == 1) done = true;
       Return (r).
     };
```

3.2.3. Tính khóa bí mật *d* (private key)

Từ $de \equiv 1 \pmod{\varphi(n)}$ thì *d* chính là phần tử nghịch đảo của *e* trong *modulo* $\varphi(n)$. Theo tính chất nếu *e* có nghịch đảo *d* theo *modulo* $\varphi(n)$ khi và chỉ khi $gcd(e, \varphi(n)) = 1$. Ta dùng giải thuật được mô tả tại *Algorithm 1.8* để tìm *d*.

Function 3.4 (*Hàm tìm khóa bí mật d*).

```
Input: Số lớn φ(n) = (p − 1) (q − 1) (r − 1), khóa công khai e.
Output: Số lớn d (private key)
Tên hàm: GetPrivatekey(e, φ(n))
 1. BigNum[] p = {0, 1}, BigNumInteger[] q = BigNum[2],
    BigNum[] r = {0, 0}, step = 0, a = φ(n), b = e;
 2. While (b.LengthData > 1 || (b.LengthData == 1 &&
    b.Data[0] !=0 ))
     {
       BigNum quotient, remainder;
       If (step > 1)
        {
            BigNum pval = (p[0] - (p[1].q[0] )) % φ(n),
            p[0] = p[1], p[1] = pval;
        }
```

```
        Div(a, b, quotient, remainder);
        q[0]=q[1],r[0]= r[1];
        q[1] = quotient, r[1] = remainder;
        a = b, b = remainder, step++;
      }
3. If (r[0].LengthData > 1 || (r[0].LengthData ==1 &&
   r[0].Data[0]!= 1)) throw ("Không tồn tại nghịch đảo");
4. BigNum k = ((p[0] – (p[1].q[0]))% φ(n).
   If ((k.Data[MaxLength - 1] & 0x80000000)!= 0)
   k+=φ(n);
5. Return(k).
```

3.2.4. Tính lũy thừa theo modulo (Modular Exponentiation)

Để tính lũy thừa theo *modulo* ($a^{\exp} \pmod n$), Ta sử dụng phương pháp *bình phương và nhân* rút gọn theo *modulo n* đã được trình bày tại *Algorithm 2.6*.

Function 3.5 (*Hàm tính lũy thừa nhanh theo modulo*).

```
Input: Các số lớn a, exp, n.
Output: Số lớn a^exp mod n.
Tên hàm: ModPow(a, exp, n).
  1. r = 1, pos = 0, temp = a % n, count = 0;
  2. Totalbits = exp.bitcount(); //Hàm đếm số bit của exp.
  3. For pos = 0  to exp.LengthData – 1 step 1
     {
       mask = 0x01, index = 0;
       for index = 0 to 31 step 1
       {
          If ((exp.Data[pos] & mask)!= 0 )r = (r.temp) % n;
          mask <<= 1;
          temp =(temp.temp) % n;
          count++;
          if (count == totalbits) break;
       }
     }
  4. Return (r).
```

Hàm ModPow trên được sử dụng chung để tiến hành mã hóa và giải mã cho hệ mật RSA mở rộng. Trong thực tế nếu dữ liệu *m* ban đầu mà lớn thì mã hóa và giải mã sẽ chậm.

Hiện nay, RSA được sử dụng trong lĩnh vực xác thực (chữ ký số) thông tin. Nó hiệu quả khi kết hợp với hàm băm mà đã trình bày tại *Mục 1.9 của Chương I*.

Thí dụ 3.1. Dùng hàm SHA-1 để băm chuỗi *M* ="CONG HOA XA HOI CHU NGHIA VIET NAM". Ta giả sử cho hàm băm tên: M_1 = HashFunction(*M*).

Khi đó giá trị băm M_1 = "*KGopBdo2cM3J7jjF0mOeUEmkrU4*". Sau đó ta có thể dùng RSA để mã hóa chuỗi *M1* này (hình thành lên chữ ký số).

Trong phạm vi của luận văn này luận văn không trình bày sâu về chữ ký số. Luận văn chỉ muốn nhấn mạnh tính hiệu quả khi RSA sử dụng kết hợp với hàm băm để xác thực tính toàn vẹn của thông tin dữ liệu ban đầu.

3.3. Giao diện demo thực hiện các phép toán số học và multi – prime RSA

Chương trình demo được cài đặt trên máy laptop có cấu hình như sau:

- *Phần cứng: CPU* Core i7, 3630QM, 2.4Ghz, RAM: 12GB, Ổ cứng 100GB.
- *Phần mềm:* HĐH Windows 10 pro, .Net Framework 4.5. Microft Studio 2013.

Một số giao diện chương trình demo ứng dụng lớp thư viện số lớn được cài đặt theo các bước trên cho các phép toán số học và mã hóa thông tin của hệ mật RSA được trình bày tại ***Phụ lục - Một số giao diện chương trình thực nghiệm Multi –prime RSA.***

3.4. Đánh giá kết quả thực nghiệm

3.4.1. Đánh giá các phép toán số học với các số nguyên lớn

- Luận văn đã trình bày được cấu trúc dữ liệu và biểu diễn số nguyên lớn (có dấu và không dấu). Minh họa các thuật toán số học đã được tối ưu như: *cộng, trừ, nhân, chia, lũy thừa nhanh, lũy thừa nhanh theo modulo,...*
- Thể hiện thông số về thời gian khi kiểm thử thực nghiệm các phép toán đặc biệt là *phép nhân, chia và lũy thừa*. Đồng thời cũng đã kiểm nghiệm được tính đúng đắn của kết quả các phép tính.

3.4.2. Đánh giá hiệu suất thực hiện các phép toán

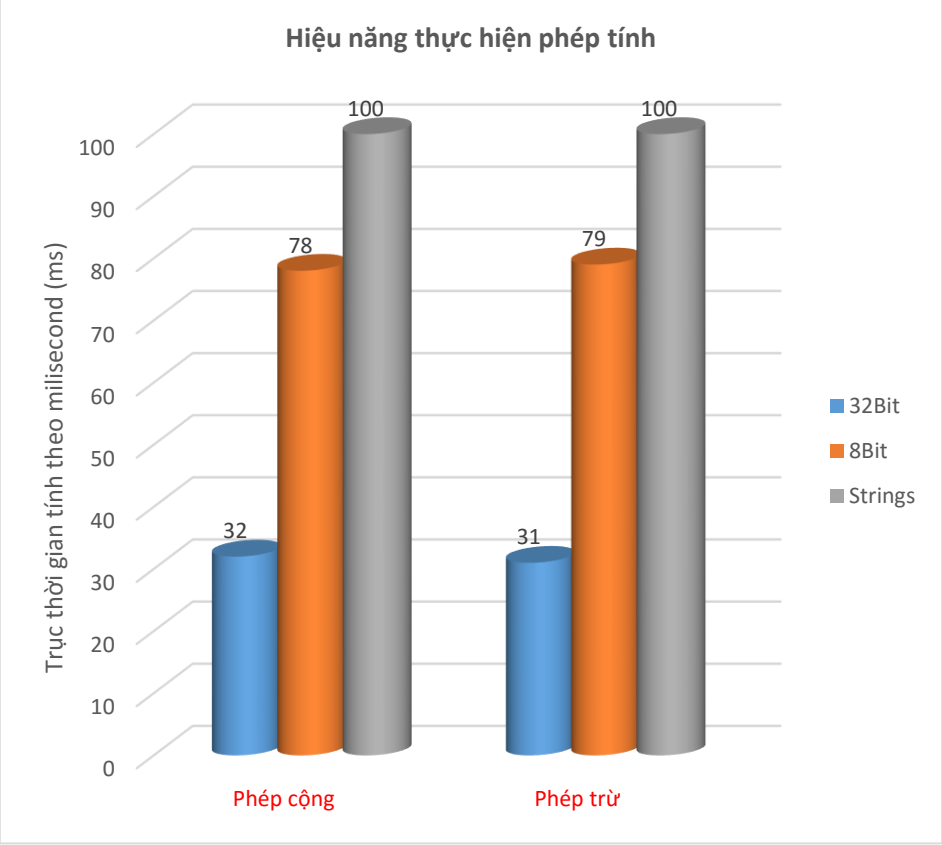
Luận văn này đã đánh giá việc thực hiện các phép toán số học như: *cộng, trừ, nhân, chia, so sánh, lũy thừa, lũy thừa theo modulo, ước chung lớn nhất,...* được xử lý trên các phép thao tác *bit* khi dùng

8 bit, 32 bit để biểu diễn và phương pháp biểu diễn số lớn dưới dạng chuỗi các ký tự số với các phép toán thông thường (*String*).

Thời gian tính từ Input số đầu ngẫu nhiên vào đến khi thực hiện xong phép toán và hiển thị ra ngoài.

3.4.2.1. Đánh giá hiệu năng thực hiện phép cộng, phép trừ

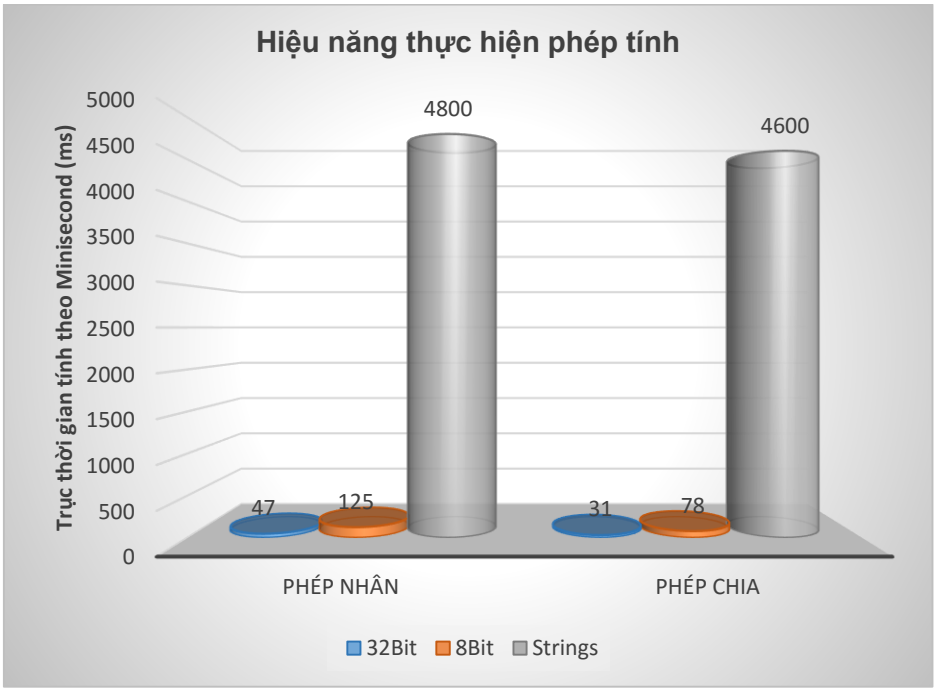
Với hai số *a* = 4096 bit, *b* = 4096 *bit* (cỡ 1.233 số chữ số). Chiều dài lớn nhất khi khai báo mảng *Data* chứa cỡ 16.000 *bit*, tính trung bình của 1.000.000 lần gọi với các số ngẫu nhiên, ta có biểu đồ đánh giá, so sánh như sau



Hình 3.2. Biểu đồ đánh giá hiệu năng phép cộng, trừ hai số lớn.

3.4.2.2. Đánh giá hiệu năng thực hiện phép nhân và chia

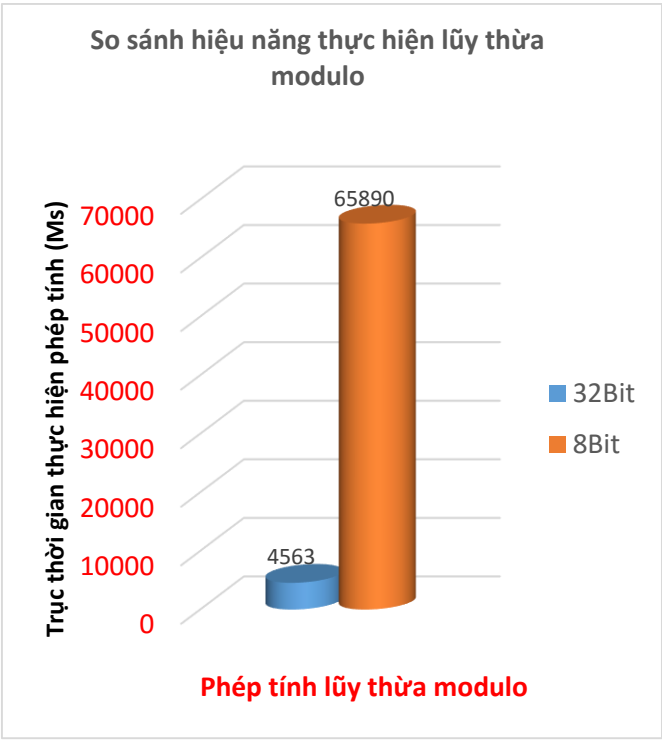
Chiều dài lớn nhất của mảng *Data* cỡ 16.000 *bit*, trung bình của 1.000.000 lần gọi. Với phép nhân số *a* = 4096 *bit* và số *b* = 4096 *bit*. Với phép chia số *a* = 4096 *bit* và số *b* là số lớn lấy ngẫu nhiên từ 2 đến 4096 bit thực hiện lặp 1.000.000 lần



Hình 3.3. Biểu đồ đánh giá hiệu năng phép nhân, chia hai số lớn

3.4.2.3. *Đánh giá hiệu năng thực hiện phép lũy thừa theo* modulo

Giả sử cần tính kết quả của $a^e \pmod n$. Với *a*=2048 *bit*, *e*=2048 *bit* và *n* =4096 *bit*. Thực hiện test với 100 lần khác nhau của *a*, *e* và *n* với cùng giải thuật mảng *Data* có chiều dài 16.000 *bit* biểu diễn số lớn 8 *bit* và 32 *bit*.



Hình 3.4. Biểu đồ đánh giá hiệu năng phép lũy thừa modulo

3.4.3. Đánh giá chương trình mô phỏng hệ mật mã multi – prime RSA

Qua quá trình thử nghiệm bằng chương trình mô phỏng hệ thống RSA bao gồm:

- *Xác định ba số lớn nguyên tố p, q,r (bằng giải thuật Miller – Rabin ở trên).*
- *Xác định cặp khóa công khai e (e, n), khóa bí mật (d, n).*
- *Thực hiện mã hóa, giải mã.*

Chương trình mô phỏng RSA cho ra kết quả chính xác và tốc độ.

CHƯƠNG 4
KẾT LUẬN

Nội dung của luận văn này với mục đích nghiên cứu cơ sở về lý thuyết toán học và cài đặt lớp thư viện số lớn với các phép toán số học đã được tối ưu áp dụng cho hệ mật mã hóa khóa công khai – RSA chuẩn và *multi-prime* RSA.

Hệ mã hóa khóa công khai RSA an toàn, bảo mật dựa vào độ khó của việc phân tích một nguyên *n* thành các thừa số nguyên tố và bài toán RSA (bài toán *logarit rời rạc* trong *modulo* hợp số). Đặc biệt, khi các số nguyên tố tham gia vào RSA càng lớn thì việc thực hiện phân tích càng đòi hỏi chi phí thời gian thực hiện rất lớn. Do đó, cần có các phép toán xử lý số học hiệu quả. Trong luận văn này đã mô tả cấu trúc dữ liệu, cách biểu diễn, các phép toán cùng các giải thuật tối ưu xử lý trên số nguyên lớn và đã cài đặt được thư viện số nguyên lớn theo mục đích ban đầu đề ra.

Qua quá trình thử nghiệm thư viện số lớn có khả năng lưu trữ và xử lý các số nguyên có độ dài hàng ngàn chữ số với tốc độ thực hiện không chậm hơn các thư viện đã có. Đồng thời thư viện số lớn có khả năng mở rộng linh hoạt, dễ tùy biến, dễ dàng tích hợp bổ sung thêm các phép toán số học áp dụng cho từng bài toán cụ thể giúp tiết kiệm bộ nhớ và thời gian tính toán.

4.1. Một số kết quả đã đạt được

4.1.1. Về nội dung

- Trình bày tổng quan về mã hóa, hệ mã hóa và cơ chế hoạt động của RSA.
- Trình bày về kết quả nghiên cứu *multi – prime* cho RSA, mở rộng cho hệ mã hóa công khai RSA. Cơ sở toán học của lý thuyết mật mã.
- Trình bày về số nguyên tố, tầm quan trọng và các phương pháp kiểm tra tính nguyên tố của một số nguyên lớn.
- Xây dựng được cấu trúc tổ chức dữ liệu, cách biểu diễn số nguyên lớn.
- Trình bày và áp dụng các giải thuật toán học tối ưu áp dụng cho các phép xử lý số học với số nguyên lớn.
- Đánh giá được độ phức tạp tính toán của các phép toán thấp hơn nhiều so với các phép toán thông thường hiện nay.
- Đánh giá được hiệu năng thực hiện các phép toán xử lý số học với số lớn.

4.1.2. Về xây dựng chương trình

- Đã áp dụng được cấu trúc, biểu diễn số lớn, các phép toán xử lý số học và cài đặt thành công trong thư viện số lớn bằng ngôn ngữ C#.
- Áp dụng thư viện số lớn để xây dựng chương trình kiểm tra và sinh số nguyên tố. Xây dựng chương trình sinh khóa sử dụng cho quá trình mã hóa và giả mã với hệ mã hóa khóa công khai *multi-prime* RSA.

4.1.3. Về áp dụng thực tế

Chương trình sinh khóa và Engine mã hóa và giải mã từ thư viện số lớn cỡ 4096 *bit* đã được áp dụng thực tế vào việc xác thực người dùng sử dụng hệ thống Internet Banking của Ngân hàng Xây dựng (CBBank) và giữa CBBank với đối tác cung cấp dịch vụ SMS Banking.

4.1.4. Về kết quả mới thực hiện được

- Xây dựng được cấu trúc dữ liệu, biểu diễn số nguyên lớn dưới dạng mảng liên tiếp 32 *bit* và các phép toán xử lý số học trên số lớn này.
- Mở rộng hệ mật RSA nhằm tăng cường độ an toàn bằng cách sử dụng ba số nguyên tố lớn $n = pqr$ thay vì sử dụng hai số là p và q. Có thể mở rộng dùng tới m số nguyên tố. Đồng thời đã chứng minh tính đúng đắn và cài đặt thử nghiệm thành công điều này.

4.4.2. Một số vấn đề còn tồn tại

- Chưa cài đặt thử nghiệm được thuật toán AKS để kiểm tra chính xác một số nguyên có phải là nguyên tố hay không. Luận văn vẫn sử dụng thuật toán Miller – Rabin, một thuật toán xác xuất để kiểm tra tính nguyên tố.
- Luận văn mới dùng lại ở cấu trúc dữ liệu và cách biểu diễn với số nguyên nhưng chưa áp dụng được cho số thực.
- Kỹ thuật cài đặt thư viện số lớn vẫn còn những điểm chưa được tối ưu nhất.
- Chưa đưa được kết quả đánh giá về độ an toàn của RSA mở rộng đã trình bày bằng toán học khi dùng ba số nguyên tố p, q và r.
- Chương trình sử dụng thực tế tại CBBank chưa có một đơn vị độc lập có thẩm quyền kiểm định chất lượng, tính ổn định và an toàn.

4.3. Hướng phát triển trong tương lai

- Tiếp tục thử nghiệm và đánh giá các thuật toán tối ưu phép tính số học áp dụng để nâng cấp thư viện số lớn trình bày trong luận văn này.
- Nghiên cứu phép nhân nhanh bằng phép biến đổi nhanh Fourier (Fast Fourier Transform).
- Đánh giá về độ an toàn của hệ RSA mở rộng bằng toán học.
- Kết hợp mã hóa công khai RSA với các nội dung bảo mật an toàn thông tin khác như giấu tin...
- Phát triển cấu trúc dữ liệu và các giải thuật số học xử lý cho số thực.
- Tiếp tục nâng cấp, phát triển để áp dụng vào nhiều hệ thống bảo mật, thực hiện tạo và xác thực chữ ký số hơn nữa.

TÀI LIỆU THAM KHẢO

[1] Hà Huy Khoái (1998). *Số học thuật toán*. NXB Giáo dục, Hà Nội.

[2] Nguyễn Xuân Huy (2008). *Sáng tạo trong thuật toán và lập trình*. NXB Khoa học tự nhiên và Công nghệ Quốc gia.

[3] Lưu Hồng Dũng (2012). 'Nghiên cứu phát triển thuật toán chữ ký số RSA'. *Tạp chí khoa học và kỹ thuật – Học viện Kỹ thuật Quân sự,* số 150.

[4] Nguyễn Khanh Văn (2012). Hệ thống mật mã khóa công khai. In: Nguyễn Khanh Văn & Trần Đức Khánh. *Mật mã và an toàn thông tin*. Đại học Bách khoa, Hà Nội.

[5] Phan Đình Diệu (2002). *Lý thuyết mật mã và an toàn thông tin*. NXB Đại học Quốc Gia Hà Nội.

[6] Bùi Doãn Khanh, Nguyễn Đình Thúc, Trần Đan Thư (2007). *Cơ sở lý thuyết số trong an toàn, bảo mật thông tin*. NXB Giáo dục.

[7] Nguyễn Bình (2006). *Lý thuyết thông tin*. NXB Bưu Điện, Hà Nội.

[8] Nguyễn Vũ Lương. (eds) (2006). *Các bài giảng về số học*. NXB Trường Đại học Khoa học Tự nhiên, Đại học QG HN.

[9] Nguyễn Viết Đông, Trần Ngọc Hội (2005). *Đại số đại cương*. NXB Đại học Quốc Gia TPHCM.

[10] Thomas H. Cormen, Charles. Leiserson, Ronald. Rivest, Clifford Stein (eds) (2009). *Introduction To Algorithms*, The MIT Press. Cambridge, Massachusetts London, England.

[11] Rosen, Kenneth H (ed) (2005). *Elementary Number Theory and Its Applications*. Addison – Wesley, NewYork.

[12] A. Menezes, P. van Oorschot, and S. Vanstone (1996). Efficient Implementation. In. A. Menezes, P. van Oorschot, and S. Vanstone (eds). *Handbook of Applied Cryptography*. CRC Press, Boca Raton. PP. 591 - 630

[13] M. Jason Hinek (2006). On the Security of Multi-prime RSA. *David R. Cheriton School of Computer Science, University of Waterloo Waterloo, Ontario, Canada*. July 13. From <http://cacr.uwaterloo.ca/techreports/2006/>

[14] Shoup Victor (2005). *A Computational Introduction to Number Theory and Algebra*. Cambridge University Press, Theorem 2.4, p.15.