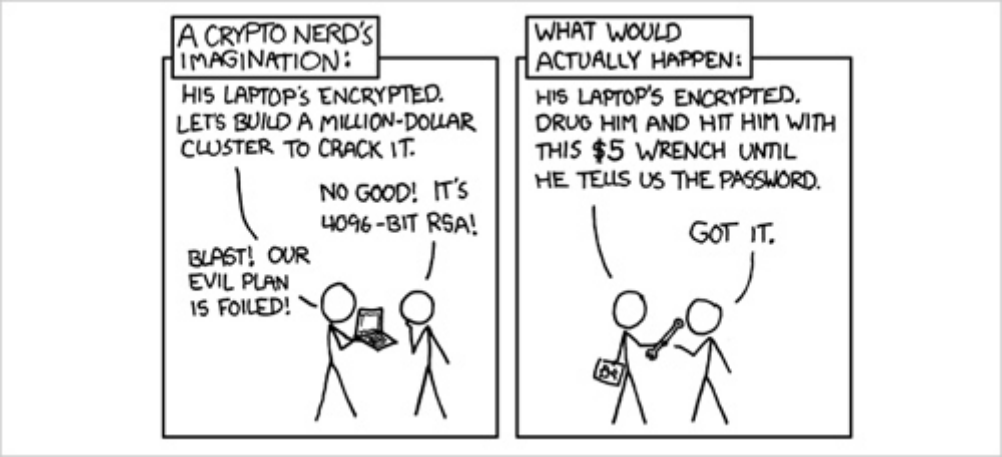
Mã hóa dữ liệu ổ cứng để tăng tính bảo mật trên Linux

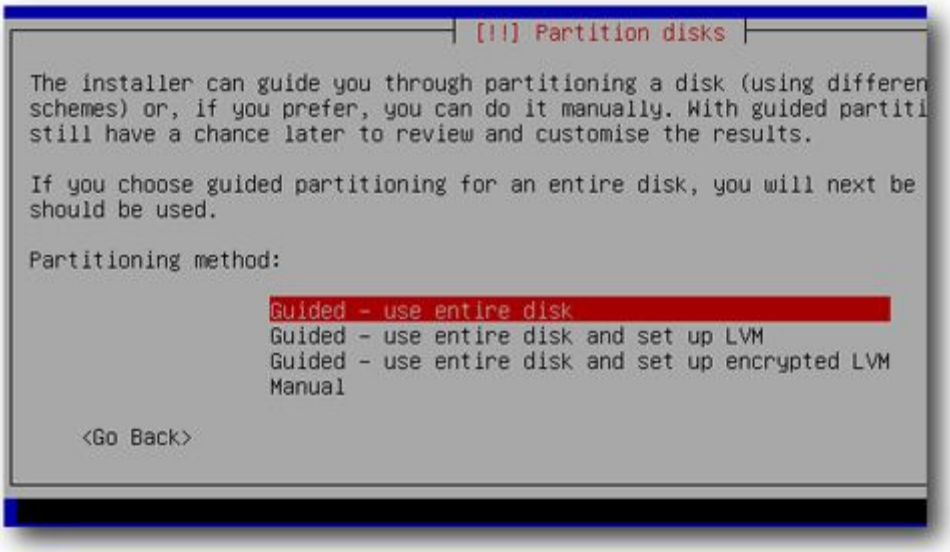
-Dữ liệu cá nhân trong ổ cứng là điều nhạy cảm và yêu cầu độ bảo mật tối đa, tùy vào nhu cầu và mục đích sử dụng trong công việc mà người dùng lựa chọn cách thức bảo vệ cho phù hợp. Trong bài viết sau, Quản Trị Mạng sẽ giới thiệu với các bạn cách mã hóa dữ liệu ổ cứng, cụ thể là từng phân vùng, thư mục trong hệ điều hành Linux với TrueCrypt và eCryptfs.



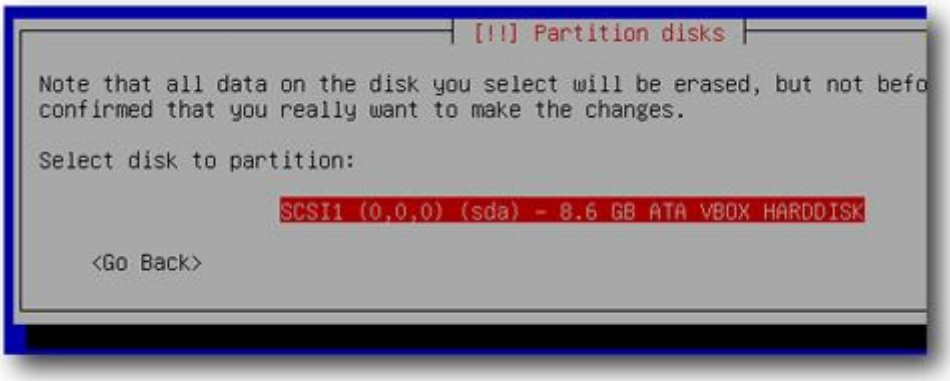
Mã hóa phân vùng ổ cứng

Phiên bản đĩa cài đặt thay thế của Ubuntu cung cấp thêm cho người dùng 1 sự lựa chọn nữa để mã hóa phân vùng cài đặt Ubuntu, do vậy các bạn chỉ cần tải file ISO về máy, ghi ra đĩa CD/DVD hoặc tạo USB boot và tiến hành cài đặt Ubuntu sau đó.

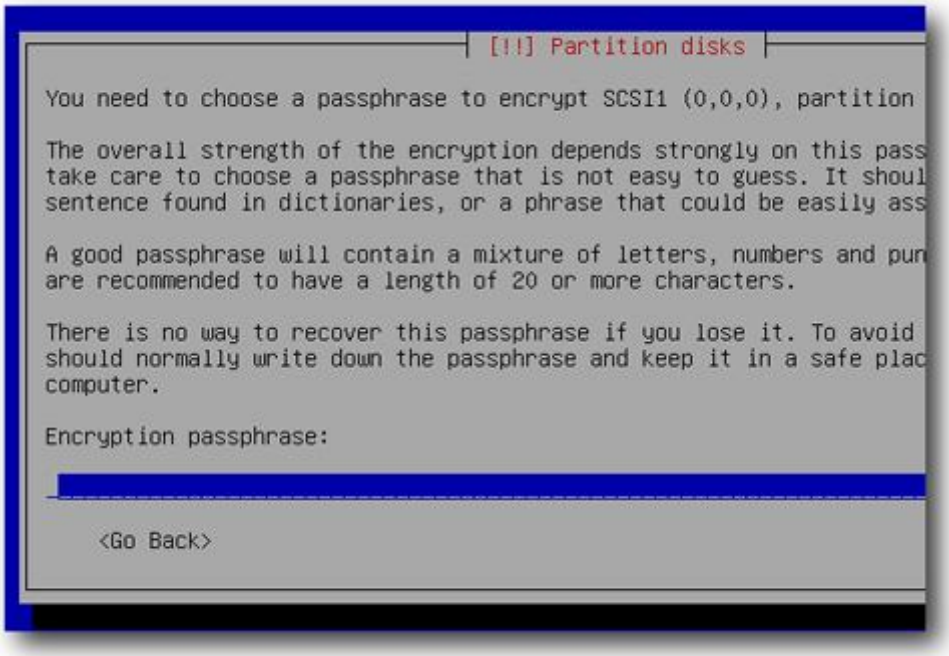
Với quy trình cài đặt khá giống với với Ubuntu nguyên bản, bước đầu người sử dụng sẽ phải chọn ngôn ngữ hiển thị, kiểu bàn phím, hệ thống network, và bước quan trọng nhất đương nhiên là phân vùng cài đặt với lựa chọn *Guided – use entire disk and set up encrypted LVM* để áp dụng trên toàn bộ ổ cứng:



Lưu ý rằng chúng ta cần khai báo hoặc khởi tạo phân vùng Master hoặc không phải là Slave để bắt đầu cài đặt:

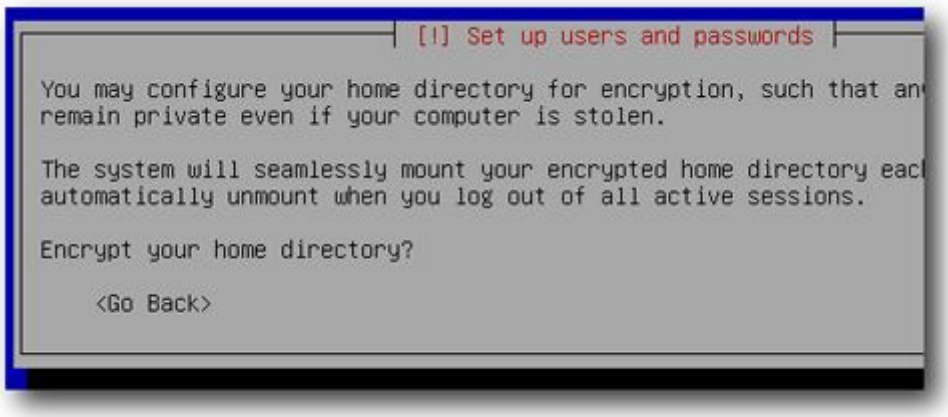


Khởi tạo khóa mật khẩu sử dụng để mã hóa ổ cứng khi đăng nhập vào Ubuntu:



Lựa chọn có muốn mã hóa thư mục gốc - home hay không, chỉ trong trường hợp chúng ta thay thế thư mục này bên ngoài phân

vùng cài đặt Ubuntu:



Như vậy là chúng ta đã hoàn tất các bước cơ bản, các bạn chỉ cần thực hiện các bước tiếp theo để hoàn tất quá trình này.

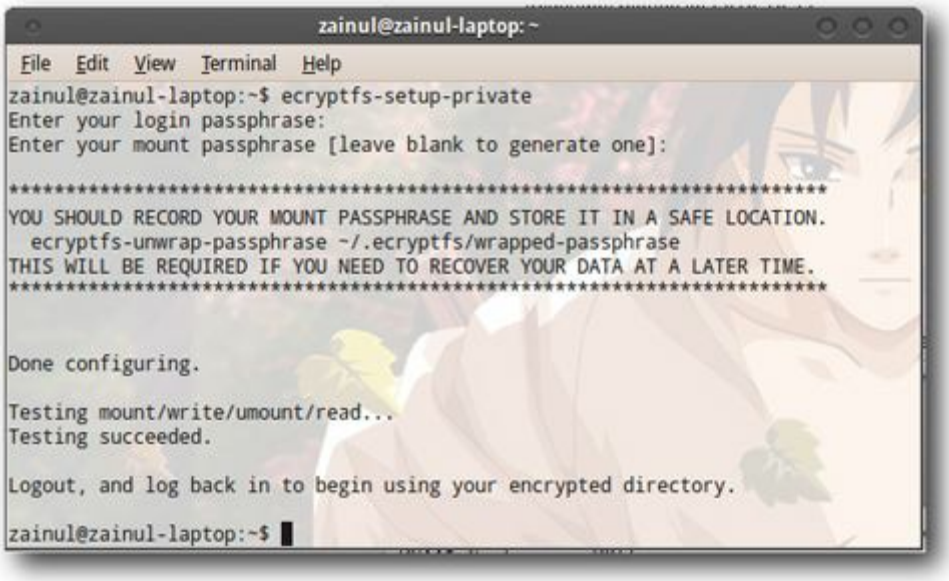
Mã hóa thư mục

eCryptfs là 1 hệ thống mã hóa file dựa trên chuẩn PGP được tạo ra bởi Philip Zimmerman vào năm 1991. Điểm độc đáo của eCryptfs so với các công cụ mã hóa khác, như TrueCrypt, là không cần xác định trước dung lượng phân vùng hoặc ổ cứng chúng ta cần áp dụng. Để cài đặt eCryptfs, các bạn hãy sử dụng lệnh sau:

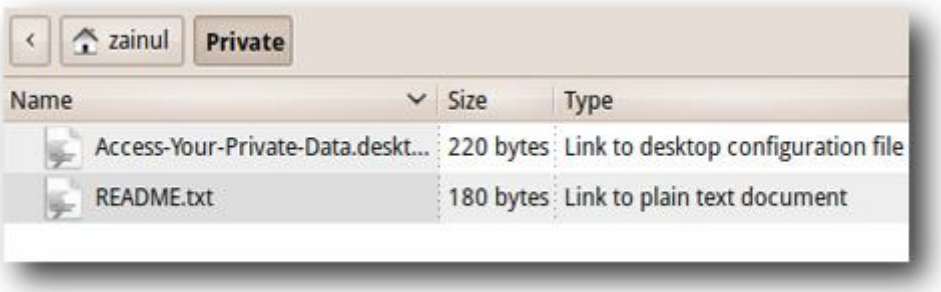
sudo aptitude install ecryptfs-utils

eCryptfs sẽ tạo ra 1 thư mục private trên ổ cứng nơi chương trình hoạt động và dùng để lưu trữ dữ liệu trong đó:

ecryptfs-setup-private



Lưu ý rằng quá trình này sẽ ẩn thư mục *~/.Private.* Chúng ta nên lưu trữ các dữ liệu nhạy cảm trong thư mục private này để đảm bảo không ai có thể truy cập và sử dụng, vì ecryptfs sẽ giấu toàn bộ dữ liệu trong thư mục đó:



1 điểm khác là thư mục private này sẽ tự động xuất hiện trong hệ thống khi bạn đăng nhập, đồng thời đây cũng là cơ hội cho người khác sử dụng và truy cập khi bạn không ở bên cạnh máy tính. Chúng ta có thể áp dụng cách làm sau để khắc phục vấn đề này là ngăn chặn ecryptfs mở khóa các thư mục khi người dùng đăng nhập bằng cách xóa bỏ các file rỗng lưu trữ trong thư mục ~/.ecryptfs/ và "cách ly" thư mục này khi không sử dụng máy tính:

ecryptfs-umount-private

Kiểm tra quá trình mã hóa email

Quản Trị Mạng - Phân tích giao thức POP3, IMAP và SMTP thông qua cơ chế bảo mật SSL

Để thuận lợi cho quá trình phân tích này, sẽ rất tốt khi "nói chuyện" trực tiếp với server SMTP hoặc IMAP của bạn. Nhưng mọi việc sẽ trở nên phức tạp khi tiến hành mã hóa dữ liệu đầu cuối, nhưng với các công cụ thích hợp, việc này sẽ không quá khó khăn.

Thông thường, hầu như tất cả hệ thống mail server đều yêu cầu lựa chọn cơ chế mã hóa kết nối. 2 phương thức sau được sử dụng – hoặc toàn bộ các địa chỉ gửi qua SSL hoặc 1 cơ chế khác là StartTLS sẽ được sử dụng để kích hoạt quá trình mã hóa sau khi nhận được yêu cầu kết nối.

Trước tiên hãy xem qua về dịch vụ SSL, thường được sử dụng với các yêu cầu chuyên dụng, đặc biệt qua cổng TCP. Sau đây là bảng tham khảo về các cổng quan trọng khác:

Service	Abbreviation	TCP port
HTTP over SSL	https	443
IMAP over SSL	imaps	993
IRC over SSL	ircs	994
POP3 over SSL	pop3s	995
SMTP over SSL	ssmtp	465

Dịch vụ này sẽ lắng nghe yêu cầu từ cổng TCP, đặc biệt là những kết nối trực tiếp qua SSL, ví dụ những hệ thống email client nào không hỗ trợ SSL sẽ không thể giao tiếp với server IMAPS qua cổng 993. Một khi các dữ liệu và thông số mã hóa đã được thực hiện, chúng sẽ được "cấp phép" và tạo ra 1 tunnel – đường hầm riêng biệt, thông qua đó, quá trình lưu chuyển dữ liệu được thực hiện trong thực tế. Dựa vào các sự kết hợp và các thành phần liên quan trong kết nối SSL, khi xảy ra bất kỳ sự cố nào, các công cụ hỗ trợ như telnet và netcat thường có xu hướng rút ngắn quá trình này lại.

Tiếp theo là 1 bước kiểm tra nho nhỏ với OpenSSL, có bao gồm 1 ví dụ SSL client nho nhỏ có thể được sử dụng để tạo kết nối tới dịch vụ SSL

như https://www.heise.de:

```
$ openssl s_client -host www.heise.de -port 443
CONNECTED(00000003)
[...]
---
Certificate chain
0 s:/C=DE/ST=Niedersachsen/L=Hannover/O=Heise Zeitschriften Verlag
GmbH Co KG/OU=Netzwerkadministration/OU=Terms of use at
www.verisign.com/rpa (c)05/CN=www.heise.de
i:/O=VeriSign Trust Network/OU=VeriSign, Inc./OU=VeriSign International
Server CA - Class 3/OU=www.verisign.com/CPS Incorp.by Ref. LIABILITY
LTD.(c)97 VeriSign
1 s:/O=VeriSign Trust Network/OU=VeriSign, Inc./OU=VeriSign
International Server CA - Class 3/OU=www.verisign.com/CPS Incorp.by Ref.
LIABILITY LTD.(c)97 VeriSign
i:/C=US/O=VeriSign, Inc./OU=Class 3 Public Primary Certification Authority
---
[...]
```

Các thông tin trên được cung cấp và chứng thực bởi openssl, cho phép chúng ta kiểm tra các chứng nhận khác đã được sử dụng. Nếu không làm như vậy, chẳng khác nào các nhà quản lý ở cửa sẵn và chờ đợi những cuộc tấn công theo kiểu man-in-the-middle. Về mặt kỹ thuật, những ai có thể sử dụng công nghệ ettercap hoàn toàn có thể lấy được mật khẩu quản trị 1 cách đơn giản.

Tham số mã hóa và giải mã tín hiệu SSL client hoàn toàn “vô hình” – transparent, vì vậy người sử dụng có thể liên lạc trực tiếp đến server:

```
GET / HTTP/1.1
Host: www.heise.de
<return>
HTTP/1.1 302 Found
Date: Wed, 16 Sep 2009 10:24:44 GMT
Server: Apache/1.3.34
Location: http://www.heise.de/
[...]
```

Đăng nhập vào IMAPS

Quá trình này chỉ phức tạp hơn 1 chút:

```
$ openssl s_client -host imap.irgendwo.de -port 993
[...]
* OK IMAP4 Ready 0.0.0.0 0001f994
1 Login user-ju secret
1 OK You are so in
2 LIST "" "*"
* LIST (\HasChildren) "." "INBOX"
* LIST (\HasNoChildren) "." "INBOX.AV"
[...]
2 OK Completed (0.130 secs 5171 calls)
3 logout
* BYE LOGOUT received
3 OK Completed
```

Khi thực hiện xong bước này, đừng quên sắp xếp lại các số thứ tự tương ứng với câu lệnh IMAP trước đó. Đối với giao thức POP3 cũng tương tự như vậy, chúng ta phải tự xác thực bên trong "đường hầm" SSL bằng câu lệnh USER và PASS POP3:

```
$ openssl s_client -host pop.irgendwo.de -port 995
[...]
+OK POP server ready H mimap3
USER user-ju
+OK password required for user "user-ju"
PASS secret
+OK mailbox "user-ju" has 0 messages (0 octets) H mimap3
quit
+OK POP server signing off
```

Đây có thể coi là sự lựa chọn và thay thế thích hợp dành cho công cụ telnet-ssl.

StartTLS

Những nhà cung cấp dịch vụ Internet đặc biệt thích sử dụng mô hình SSL, Transport Layer Security thông qua StartTLS. Mô hình này có lợi thế hơn với nhiều lựa chọn trong khi vẫn cho phép client không giao tiếp với server mà không được mã hóa. Mặt trái của điều này là các email client cần phải tương tác trực tiếp với server nếu muốn từ chối 1 kết nối TLS bất kỳ nào đó.

Lựa chọn mặc định của email client là "TLS, if available" đi kèm với sự mạo hiểm, các cuộc tấn công man-in-the-middle có thể “nhẹ nhàng” thay đổi câu lệnh StartTLS – với tính năng kích hoạt quá trình mã hóa, thành XtartTLS. Sau đó, server sẽ phản hồi lại rằng không thực hiện lệnh XtartTLS, và gây ra hiện tượng các email client khi gửi dữ liệu trong dạng chưa mã hóa vào 1 form không xác định ngược về phía người sử dụng. Do đó, khuyến cáo nên kiểm tra kỹ rằng máy chủ có thể xử lý lệnh StartTLS, và sau đó kích hoạt tính năng này. Nếu nhận được thông báo lỗi bất kỳ, rõ ràng là đã có vấn đề đâu đó trong hệ thống.

Các cổng mà dịch vụ TLS hoạt động trên đó phụ thuộc vào phía nhà cung cấp. Về nguyên tắc, các kiểu mã hóa này có thể nhúng 1 cách “vô hình” – transparent, vào trong hệ thống mà không yêu cầu bất kỳ hành động nào. Để tìm hiểu về hệ thống mail server có hỗ trợ tính năng này hay không:

```
$ nc smtp.irgendwo.de smtp
220 Mailserver ESMTP Exim 4.69 Wed, 16 Sep 2009 13:05:15 +0200
ehlo test
250-Mailserver Hello loki [10.1.2.73]
250-SIZE 78643200
250-PIPELINING
250-STARTTLS
250 HELP
quit
221 Mailserver closing connection
```

Danh sách này nên đi kèm với lệnh StartTLS, chức năng chính là kích hoạt quá trình mã hóa Transport Layer Security:

```
STARTTLS
220 TLS go ahead
```

Vào thời điểm này, Netcat sẽ gây ra 1 số phiền phức khó hiểu, nhưng OpenSSL lại có thể khắc phục điều này dễ dàng. Các nhà phát triển đã tạo ra hệ thống SSL client đủ thông minh để yêu cầu mã hóa TLS đối với các giao thức SMTP, POP3, IMAP và FTP, mặc dù không hoạt động với tất cả các server:

```
$ openssl s_client -host mail.irgendwo.de -port 25 -starttls smtp
```

CONNECTED(00000003)
[...]
250 HELP
ehlo test
250-Mailserver Hello loki [10.1.2.73]
250-SIZE 52428800
250-PIPELINING
250-AUTH PLAIN LOGIN
250 HELP

Cơ chế xác thực SMTP

Việc xác thực trong SMTP có 1 chút rắc rối hơn. Đối với hầu hết server, như trong ví dụ này, hỗ trợ phương thức AUTH PLAIN, nơi các dữ liệu phải đạt chuẩn Base64. Quá trình này được xử lý bởi câu lệnh Pearl sau:

$ perl -MMIME::Base64 -e 'print encode_base64("\000user-ju\000secret")'
AHVzZXItanUAc2VjcmV0

Kết quả thu được sẽ phải khớp với yêu cầu từ SMTP server:

AUTH PLAIN AHVzZXItanUAc2VjcmV0
235 Authentication succeeded

Những tín hiệu nhận được đã sẵn sàng với các các câu lệnh SMTP tiếp theo, đối với các địa chỉ và server không hỗ trợ OpenSSL, người sử dụng có thể dùng gnutls-cli có sẵn trong gói gnutls-bin. Đầu tiên, nó tạo ra 1 kết nối có dạng cleartext tới bất kỳ dịch vụ độc quyền TLS nào như:

$ gnutls-cli -s -p submission smtp.heise.de
Resolving 'smtp.heise.de'...
Connecting to '10.1.2.41:587'...

- Simple Client Mode:

220 taxis03.heise.de ESMTP Exim 4.69 Wed, 16 Sep 2009 18:03:01 +0200
ehlo test
250-taxis03.heise.de Hello loki.ct.heise.de [10.10.22.75]
250-SIZE 78643200
250-PIPELINING
250-STARTTLS

250 HELP
starttls
220 TLS go ahead

Tiếp theo, chuyển sang câu lệnh thứ 2 để xử lý ID của các công cụ và gửi trực tiếp tín hiệu SIGALARM tới đó:

$ ps aux | grep gnutls
ju 6103 pts/3 S+ 18:03 0:00 gnutls-cli [...]
$ kill -s SIGALRM 6103

Điều này sẽ khiến gnutls-cli dàn xếp với chuẩn TLS và tự động kết nối lại tham số stdin và stdout để tạo ra "đường hầm" mới. Đồng thời, cũng chỉ ra 1 số thông tin khá thú vị về kết nối TLS mới tạo ra:

*** Starting TLS handshake
- Certificate type: X.509
- Got a certificate list of 1 certificates.

- Certificate[0] info:
The hostname in the certificate matches 'smtp.heise.de'.
valid since: Thu Dec 14 14:08:41 CET 2006
expires at: Sun Dec 11 14:08:41 CET 2016
fingerprint: 28:8C:E0:29:B9:31:9B:96:F6:3D:B4:49:10:CD:06:80
Subject's DN: C=DE,ST=Niedersachsen,L=Hannover,O=Heise Zeitschriften Verlag GmbH Co KG,OU=Netzwerkadministration,CN=smtp.heise.de,EMAIL=admin@heise.de
Issuer's DN: C=DE,ST=Niedersachsen,L=Hannover,O=Verlag Heinz Heise GmbH & Co KG,OU=Netzwerkadministration,CN=admin@heise.de,EMAIL=admin@heise.de

- Peer's certificate issuer is unknown
- Peer's certificate is NOT trusted
- Version: TLS 1.0
- Key Exchange: DHE RSA
- Cipher: AES 256 CBC
- MAC: SHA
- Compression: NULL
quit

221 taxis03.heise.de closing connection
- Peer has closed the GNUTLS connection

Điều này cho phép người sử dụng kết nối trực tiếp đến thư viện lưu trữ các dịch vụ để kích hoạt TLS. Nếu người dùng muốn thử nghiệm thêm để chắc chắn rằng OpenSSL có hỗ trợ s_server để có thể thực thi các câu lệnh và gửi đến www server. Tính năng gnutls-serv đồng thời cũng cung cấp các chức năng tương đương ở trong gói gnutls-bin.

T.Anh (theo h-online)

Mã hoá trong SQL Server 2005

Mã hoá là một phương pháp quan trọng nhằm bảo mật dữ liệu. Những dữ liệu nhạy cảm như số CMT, số thẻ tín dụng, mật khẩu… cần phải được bảo vệ trước vô vàn mối nguy hiểm tấn công hiện nay. Trong SQL Server 2000 bạn có thể tự tạo các hàm của riêng mình hoặc sử dụng các DLL ngoài để mã hoá dữ liệu. Trong SQL Server 2005, các hàm và phương thức này được mặc định cho phép sẵn.

SQL Server 2005 cung cấp các kỹ thuật sau để mã hoá dữ liệu

- Mã hoá bằng mật khẩu
- Mã hoá khoá đối xứng
- Mã hoá khoá không đối xứng
- Mã hoá chứng nhận

Trong phần đầu của loạt bài này, chúng tôi sẽ giải thích cách sử dụng kỹ thuật mã hoá bằng mật khẩu và phương pháp giải mã nó.

SQL Server 2005 cung cấp 2 hàm cho việc mã hoá: một cho việc mã hoá và một cho việc giải mã.

“Mã hoá bằng mật khẩu” là phương pháp mã hoá dữ liệu cơ bản thông qua mật khẩu. Dữ liệu có thể được giải mã nếu nhập đúng mật khẩu đã sử dụng khi mã hoá. Chúng ta sẽ thử một ví dụ về việc mã hoá và giải mã dữ liệu bằng kỹ thuật mã hoá thông qua mật khẩu.

```
select EncryptedData = EncryptByPassPhrase('MAK', '123456789' )
```

Kết quả

EncryptedData
0x0100000000214F5A73054F3AB954DD23571154019F3EFC031ABFCCD2
58FD22ED69A48002

Giờ chúng ta sẽ thực thi 3 lần hàm Encryptbypassphrase trên theo ví dụ sau

```
declare @count int
declare @SocialSecurityNumber varchar(500)
declare @password varchar(12)
set @count =1
```

```
while @count<=3
begin
set @SocialSecurityNumber = '123456789'
set @Password = 'MAK'
select EncryptedData = EncryptByPassPhrase(@password,
@SocialSecurityNumber )
set @count=@count+1
end
```

Kết quả

```
EncryptedData
0x01000000CBB7EE45B5C1460D6996B149CE16B76C7F7CD598DC56364
D106B05D47B930093

(1 row(s) affected)

EncryptedData
0x010000005E884D30C8FF7E4723D4E70A03B0B07F877667BAF1DA9BE1
E116434842D11B99

(1 row(s) affected)

EncryptedData
0x01000000C508FB0C4FC7734B47B414D2602A71A338417DD6852291736
84D319334A084CD
```

Lưu ý:
“123456789” ở đây có thể là số thẻ tín dụng và “MAK” là mật khẩu

Kết quả của Encryptbypassphrase sau mỗi lần thực thi hàm là khác nhau. Tuy nhiên, khi bạn giải mã dữ liệu thì nó vẫn ra kết quả như ban đầu trước khi mã hoá.

Giờ chúng ta sẽ thử giải mã dữ liệu đã được mã hoá ở trên với hàm DecryptByPassPhrase

```
select convert(varchar(100),DecryptByPassPhrase('MAK',
0x01000000CBB7EE45B5C1460D6996B149CE16B76C7F7CD598DC56364
D106B05D47B930093))
```

select convert(varchar(100),DecryptByPassPhrase('MAK',
0x010000005E884D30C8FF7E4723D4E70A03B0B07F877667BAF1DA9BE1
E116434842D11B99))

select convert(varchar(100),DecryptByPassPhrase('MAK',
0x01000000C508FB0C4FC7734B47B414D2602A71A338417DD6852291736
84D319334A084CD))

Kết quả

123456789

(1 row(s) affected)

123456789

(1 row(s) affected)

123456789

(1 row(s) affected)

Thử giải mã dữ liệu đã được mã hoá với một mật khẩu khác. Thực thi theo câu lệnh sau

select convert(varchar(100),DecryptByPassPhrase('test',
0x01000000C508FB0C4FC7734B47B414D2602A71A338417DD6852291736
84D319334A084CD))

Kết quả

NULL

(1 row(s) affected)

Kết quả cho bạn thấy SQL Server trả lại giá trị NULL nếu mật khẩu sai.

Giờ chúng ta sẽ thử tạo một bảng chứa số thẻ tín dụng và số CMT, sau đó sẽ

mã hoá dữ liệu này thông qua phương pháp mã hoá mật khẩu.

```
USE [master]
GO
/****** Object: Database [admin] Script Date: 11/25/2007 10:50:47 ******/
IF EXISTS (SELECT name FROM sys.databases WHERE name =
N'Customer DB')
DROP DATABASE [Customer DB]
go

create database [Customer DB]
go
use [Customer DB]
go

create table [Customer data]
([customer id] int,
[Credit Card Number] bigint,
[Social Security Number] bigint)
go

insert into [Customer data] values (1, 1234567812345678, 123451234)
insert into [Customer data] values (2, 1234567812345378, 323451234)
insert into [Customer data] values (3, 1234567812335678, 133451234)
insert into [Customer data] values (4, 1234567813345678, 123351234)
insert into [Customer data] values (5, 1234563812345678, 123431234)
go
```

Tạo hai cột để lưu dữ liệu đã được mã hoá

```
use [Customer DB]
go
alter table [Customer Data] add
[Encrypted Credit Card Number] varbinary(MAX)
go
alter table [Customer Data] add
[Encrypted Social Security Number] varbinary(MAX)
go
```

Cập nhật dữ liệu đã được mã hoá vào hai cột vừa tạo

```
use [Customer DB]
go
update [Customer Data] set [Encrypted Credit Card Number] =
EncryptByPassPhrase('Credit Card', convert(varchar(100),[Credit Card
Number]) )
go
update [Customer Data] set [Encrypted Social Security Number] =
EncryptByPassPhrase('Social Security', convert(varchar(100),[Social Security
Number]) )
Go
```

Truy vấn bảng bằng các lệnh sau (hình 1)

```
use [Customer DB]
go
select * from [customer data]
go
```

Kết quả

Results | Messages

	customer id	Credit Card Number	Social Security Number	Encrypted Credit Card Number	Encrypted Social
1	1	1234567812345678	123451234	0x01000000E1A14715BF3457B608919B769FEA5ADF6999009...	0x010000003CE
2	2	1234567812345378	323451234	0x01000000FB54075BB2E8DFFC00F43BB27A541CB66EFD640...	0x010000001D1
3	3	1234567812335678	133451234	0x010000003C1F7E37D02C35D14F7647A76CD9207A0D5513...	0x01000000F0C
4	4	1234567813345678	123351234	0x0100000052B8C5BA8D683FF92721F5C15E83A05708F5C8E...	0x010000009A0
5	5	1234563812345678	123431234	0x010000000CA594AE030A1955DC1272E392543593CB01E85...	0x010000009296

Hình 1

Xoá bỏ cột chứa dữ liệu chưa được mã hoá

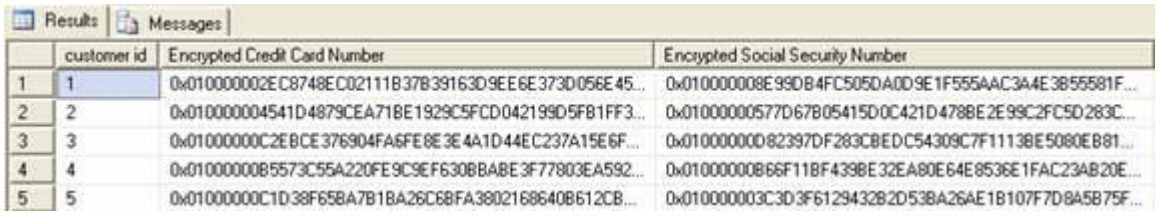
```
use [Customer DB]
go
alter table [Customer Data] drop column [Credit Card Number]
go
alter table [Customer Data] drop column [Social Security Number]
go
```

Truy vấn bảng theo các lệnh sau (hình 2)

```
use [Customer DB]
go
select * from [customer data]
```

go

Kết quả



Results | Messages

	customer id	Encrypted Credit Card Number	Encrypted Social Security Number
1	1	0x010000002EC8748EC02111B37B39163D9EE6E373D056E45...	0x010000008E99DB4FC505DA0D9E1F555AAC3A4E3B55581F...
2	2	0x010000004541D4879CEA71BE1929C5FCD042199D5FB1FF3...	0x01000000577D67B05415D0C421D478BE2E99C2FC5D283C...
3	3	0x01000000C2EBCE376904FA6FE8E3E4A1D44EC237A15E6F...	0x01000000D82397DF283CBEDC54309C7F1113BE5080EB81...
4	4	0x01000000B5573C55A220FE9C9EF630BBABE3F77803EA592...	0x01000000B66F11BF439BE32EA80E64E8536E1FAC23AB20E...
5	5	0x01000000C1D38F65BA7B1BA26C68FA3802168640B612CB...	0x010000003C3D3F6129432B2D53BA26AE1B107F7D8A5B75F...

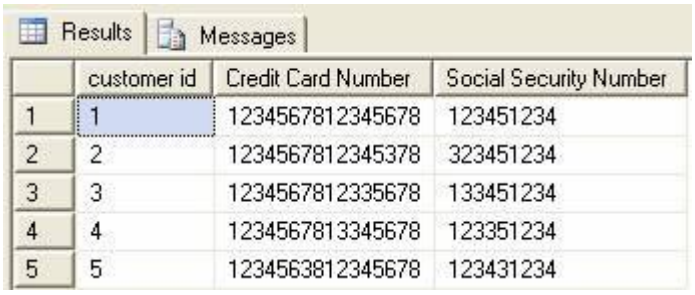
Hình 2

Giải mã dữ liệu trên bảng thông qua hàm Decryptbypassphrase như sau (hình 3)

```
use [Customer DB]
go
select
[customer id],
convert(bigint,convert(varchar(100),decryptbypassphrase('Credit
Card',[Encrypted Credit Card Number]) )) as
[Credit Card Number],
convert(bigint,convert(varchar(100),decryptbypassphrase('Social
Security',[Encrypted Social Security Number] ) )) as
[Social Security Number] from [customer data]
Go
```

Kết quả

customer id,Credit Card Number,Social Security Number
1, 1234567812345678, 123451234
2, 1234567812345378, 323451234
3, 1234567812335678, 133451234
4, 1234567813345678, 123351234
5, 1234563812345678, 123431234



Results | Messages

	customer id	Credit Card Number	Social Security Number
1	1	1234567812345678	123451234
2	2	1234567812345378	323451234
3	3	1234567812335678	133451234
4	4	1234567813345678	123351234
5	5	1234563812345678	123431234

Hình 3

Kết luận

Mã hoá dữ liệu thực sự rất quan trọng. Thông qua bài này chúng tôi đã giới thiệu đến các bạn một trong 4 kỹ thuật mã hoá sẵn có trong SQL Server 2005 – kỹ thuật mã hoá bằng mật khẩu – và phương pháp giải mã nó. Trong bài sau, chúng ta sẽ bàn luận về phương pháp hack/khôi phục dữ liệu đã được mã hoá bằng mật khẩu này.

Theo Databasejournal

Các phương pháp mã hóa và bảo mật thông tin

Lời cảm ơn

Em xin gửi lời cảm ơn chân thành tới các thầy cô giáo của khoa Công Nghệ Thông Tin, các anh chị trong công ty CSE, gia đình và các bạn bè, đã nhiệt tình giúp đỡ em trong suốt quá trình làm luận văn. Hơn nữa em xin trân trọng cảm ơn sự chỉ dẫn nhiệt tình của thầy giáo hướng dẫn Tiến Sĩ **Nguyễn Đình Công**, và sự trực tiếp chỉ bảo của anh Nguyễn Hà Chiến cùng với sự giúp đỡ nhiệt tình của thầy giáo phản biện Phó Tiến Sĩ **Trịnh Nhật Tiến** để em hoàn thành tốt cuốn luận văn tốt nghiệp.

Em xin chân thành cảm ơn.

Hà nội ngày 06 tháng 06 năm 1999.

Sinh viên

Đặng Văn Hanh

Mục Lục

Mở đầu

Thế kỷ XXI thế kỷ công nghệ thông tin, thông tin đã và đang tác động trực tiếp đến mọi mặt hoạt động kinh tế xã hội của hầu hết các quốc gia trên thế giới. Thông tin có một vai trò hết sức quan trọng, bởi vậy chúng ta phải làm sao đảm bảo được tính trong suốt của thông tin nghĩa là thông tin không bị sai lệch, bị thay đổi, bị lộ trong quá trình truyền từ nơi gửi đến nơi nhận.

Với sự phát triển rất nhanh của công nghệ mạng máy tính đặc biệt là mạng INTERNET thì khối lượng thông tin ngày càng chuyển tải nhiều hơn. Những tập đoàn công nghiệp, những công ty đa quốc gia, thị trường chứng khoán tiến hành xử lý và truyền nhận những thông tin đắt giá, những phiên giao dịch hay mua bán cổ phiếu, trái phiếu đều được tiến hành qua mạng. Giờ đây với sự tăng trưởng nhanh của các siêu thị điện tử, thương mại điện tử thì hàng ngày có một khối lượng tiền rất lớn được lưu chuyển trên mạng toàn cầu INTERNET, vấn đề khó khăn đặt ra là làm sao giữ được thông tin bí mật và giữ cho tiền đến đúng được địa chỉ cần đến.

Bạn sẽ ra sao nếu như bạn gửi thư cho một người bạn nhưng lại bị một kẻ lạ mặt nào đó xem trộm và sửa đổi nội dung bức thư trái với chủ ý của bạn, tệ hại hơn nữa là khi bạn ký một hợp đồng, gửi thông qua mạng và lại bị kẻ xấu sửa đổi những điều khoản trong đó, và sẽ còn nhiều điều tương tự như vậy nữa ... Hậu quả sẽ như thế nào nhỉ ? Bạn bị người khác hiểu nhầm vì nội dung bức thư bị thay đổi, còn hợp đồng bị phá vỡ bởi những điều khoản đã không còn nguyên vẹn. Như vậy là cả tình cảm, tiền bạc của bạn và nói rộng hơn là cả sự nghiệp của bạn đều bị đe dọa nếu như những thông tin mà bạn gửi đi không đảm bảo được tính nguyên vẹn của chúng. ***Mã hoá thông tin*** là một trong các phương pháp đảm bảo được tính trong suốt của thông tin. Nó có thể giải quyết các vấn rắc rối ở trên giúp bạn, một khi thông tin đã được mã hoá và gửi đi thì kẻ xấu rất khó hoặc không thể giải mã được.

Với mong muốn phục vụ những thông tin được truyền đi trên mạng được nguyên vẹn, trong cuốn luận văn này em nghiên cứu một số khái niệm cơ bản về mã hoá thông tin, phương pháp mã hoá thông tin RSA và xây dựng một thư viện các hàm mã hoá phục vụ trao đổi thông tin trong mô hình Client/Server. Những phần trình bày trong luận văn này bao gồm vấn đề chính sau :

Chương I Cơ sở toán học

Chương II Mật mã

Chương III Hệ mã hoá RSA.

Chương IV Mô hình Client/Server

Chương V Xây dựng hàm thư viện

Chương i Cơ sở toán học

Để có những thuật toán mã hoá tốt, chúng ta phải có những kiến thức cơ bản về toán học đáp ứng cho yêu cầu, chương này mô tả những khái niệm cơ bản về lý thuyết thông tin như Entropy, tốc độ của ngôn ngữ, hiểu biết về độ phức tạp của thuật toán, độ an toàn của thuật toán, cùng với những kiến thức toán học: modulo số học, số nguyên tố, định lý phần dư trung hoa, định lý Fermat . . . và các phương pháp kiểm tra xem một số có phải là nguyên tố hay không. Những vấn đề chính sẽ được trình bày trong chương này gồm :

- ☐ Lý thuyết thông tin
- ☐ Lý thuyết độ phức tạp
- ☐ Lý thuyết số học.

1.Lý thuyết thông tin

Mô hình lý thuyết thông tin được định nghĩa lần đầu tiên vào năm 1948 bởi Claude Elmwood Shannon. Trong phần này chúng ta chỉ đề cập tới một số chủ đề quan trọng của lý thuyết thông tin.

1.1 Entropy

Lý thuyết thông tin được định nghĩa là khối lượng thông tin trong một thông báo như là số bít nhỏ nhất cần thiết để mã hoá tất cả những nghĩa có thể của thông báo đó.

Ví dụ, trường ngay_thang trong một cơ sở dữ liệu chứa không quá 3 bít thông tin, bởi vì thông tin tại đây có thể mã hoá với 3 bít.

000 = Sunday

001 = Monday

010 = Tuesday

011 = Wednesday

100 = Thursday

101 = Friday

110 = Saturday

111 is unused

Nếu thông tin này được biểu diễn bởi chuỗi ký tự ASCII tương ứng, nó sẽ chiếm nhiều không gian nhớ hơn, nhưng cũng không chứa nhiều thông tin hơn. Tương tự như trường gioi_tinh của một cơ sở dữ liệu chứa chỉ 1 bít thông tin, nó có thể lưu trữ như một trong hai xâu ký tự ASCII : Nam, Nữ.

Khối lượng thông tin trong một thông báo M là đo bởi **Entropy** của thông báo đó, ký hiệu bởi H(M). Entropy của thông báo gioi_tinh chỉ ra là 1 bít, ký hiệu H(gioi_tinh) = 1, Entropy của thông báo số ngày trong tuần là nhỏ hơn 3bits.

Trong trường hợp tổng quát, **Entropy** của một thông báo là $\log_2 n$, với n là số khả năng có thể.

$$H(M) = \log_2 n$$

1.2 Tốc độ của ngôn ngữ. (Rate of Language)

Đối với một ngôn ngữ, tốc độ của ngôn ngữ là

$$r = H(M)/N$$

trong trường hợp này N là độ dài của thông báo. Tốc độ của tiếng Anh bình thường có một vài giá trị giữa 1.0 bits/chữ cái và 1.5 bits/chữ cái, áp dụng với giá trị N rất lớn.

Tốc độ tuyệt đối của ngôn ngữ là số bits lớn nhất, chúng có thể mã hoá trong mỗi ký tự. Nếu có L ký tự trong một ngôn ngữ, thì tốc độ tuyệt đối là :

$$R = \log_2 L$$

Đây là số Entropy lớn nhất của mỗi ký tự đơn lẻ. Đối với tiếng Anh gồm 26 chữ cái, tốc độ tuyệt đối là $\log_2 26 = 4.7$bits/chữ cái. Sẽ không có điều gì là

ngạc nhiên đối với tất cả mọi người rằng thực tế tốc độ của tiếng Anh nhỏ hơn nhiều so với tốc độ tuyệt đối.

1.3 An toàn của hệ thống mã hoá

Shannon định nghĩa rất rõ ràng, tỉ mỉ các mô hình toán học, điều đó có nghĩa là hệ thống mã hoá là an toàn. Mục đích của người phân tích là phát hiện ra khoá **k**, bản rõ **p**, hoặc cả hai thứ đó. Hơn nữa họ có thể hài lòng với một vài thông tin có khả năng về bản rõ **p** nếu đó là âm thanh số, nếu nó là văn bản tiếng Đức, nếu nó là bảng tính dữ liệu, v. v . . .

Trong hầu hết các lần phân tích mã, người phân tích có một vài thông tin có khả năng về bản rõ **p** trước khi bắt đầu phân tích. Họ có thể biết ngôn ngữ đã được mã hoá. Ngôn ngữ này chắc chắn có sự dư thừa kết hợp với chính ngôn ngữ đó. Nếu nó là một thông báo gửi tới Bob, nó có thể bắt đầu với "Dear Bob". Chắc chắn là "Dear Bob " sẽ là một khả năng có thể hơn là chuỗi không mang ý nghĩa gì chẳng hạn "tm*h&rf". Mục đích của việc thám mã là sửa những tập hợp khả năng có thể có của bản mã với mỗi khả năng có thể của bản rõ.

Có một điều giống như hệ thống mã hoá, chúng đạt được sự bí mật tuyệt đối. Hệ thống mã hoá này trong đó bản mã không mang lại thông tin có thể để tìm lại bản rõ. Shannon phát triển lý thuyết cho rằng, hệ thống mã hoá chỉ an toàn tuyệt đối nếu nếu số khoá có thể ít nhất là nhiều bằng số thông báo có thể. Hiểu theo một nghĩa khác, khoá tối thiểu dài bằng thông báo của chính nó.

Ngoại trừ an toàn tuyệt đối, bản mã mang lại một vài thông tin đúng với bản rõ, điều này là không thể tránh được. Một thuật toán mật mã tốt giữ cho thông tin ở mức nhỏ nhất, một người thám mã tốt khai thác những thông tin này để phát hiện ra bản rõ.

Người phân tích mã sử dụng sự dư thừa tự nhiên của ngôn ngữ để làm giảm số khả năng có thể của bản rõ. Nhiều thông tin dư thừa của ngôn ngữ, sẽ dễ dàng hơn cho sự phân tích mật mã. Chính vì lý do này mà nhiều sự thực hiện mã hoá sử dụng chương trình nén bản rõ để giảm kích thước văn bản trước khi mã hoá chúng. Bởi vậy quá trình nén làm giảm sự dư thừa của thông báo.

Entropy của hệ thống mã hoá là đo kích thước của không gian khoá (keyspace).

$$H(K) = \log_2(\text{number of keys})$$

1.4 Sự lộn xộn và sự rườm rà. (Confusion and Diffusion)

Theo nhà khoa học Shannon, có hai kỹ thuật cơ bản để che dấu sự dư thừa thông tin trong thông báo gốc đó là : sự lộn xộn và sự rườm rà.

Kỹ thuật lộn xộn (Confusion) che dấu mối quan hệ giữa bản rõ và bản gốc. Kỹ thuật này làm thất bại sự cố gắng nghiên cứu bản mã tìm kiếm thông tin dư thừa và thống kê mẫu. Phương pháp dễ nhất để thực hiện điều này là thông qua kỹ thuật thay thế. Một hệ mã hoá thay thế đơn giản, chẳng hạn hệ mã dịch vòng Caesar, dựa trên nền tảng của sự thay thế các chữ cái, nghĩa là chữ cái này được thay thế bằng chữ cái khác. Sự tồn tại của một chữ cái trong bản mã, là do việc dịch chuyển đi k vị trí của chữ cái trong bản rõ.

Kỹ thuật rườm rà (Diffusion) làm mất đi sự dư thừa của bản rõ bằng bề rộng của nó vượt quá bản mã (nghĩa là bản mã kích thước nhỏ hơn bản rõ). Một người phân tích tìm kiếm sự dư thừa đó sẽ có một thời gian rất khó khăn để tìm ra chúng. Cách đơn giản nhất tạo ra sự rườm rà là thông qua việc đổi chỗ (hay còn gọi là hoán vị).

2.Lý thuyết độ phức tạp.

Lý thuyết độ phức tạp cung cấp một phương pháp để phân tích độ phức tạp tính toán của thuật toán và các kỹ thuật mã hoá khác nhau. Nó so sánh các thuật toán mã hoá, kỹ thuật và phát hiện ra độ an toàn của các thuật toán đó. *Lý thuyết thông tin đã cho chúng ta biết rằng một thuật toán mã hoá có thể bị bại lộ. Còn lý thuyết độ phức tạp cho biết nếu liệu chúng có thể bị bại lộ trước khi vũ trụ xụp đổ hay không.*

Độ phức tạp thời gian của thuật toán là hàm số với độ dài đầu vào. Thuật toán có độ phức tạp thời gian f(n) đối với mọi n và độ dài đầu vào n, nghĩa là sự thực hiện của thuật toán lớn hơn f(n) bước.

Độ phức tạp thời gian thuật toán phụ thuộc vào mô hình của các thuật toán, số các bước nhỏ hơn nếu các hoạt động được tập chung nhiều trong một bước.

Các lớp của thuật toán, thời gian chạy được chỉ rõ như hàm số mũ của đầu vào là "không có khả năng thực hiện được". Các thuật toán có độ phức tạp giống nhau được phân loại vào trong các lớp tương đương. Ví dụ tất cả các thuật toán có độ phức tạp là n^3 được phân vào trong lớp n^3 và ký hiệu bởi $O(n^3)$. Có hai lớp tổng quát sẽ được chỉ dẫn là lớp P và lớp NP.

Các thuật toán thuộc lớp P có độ phức tạp là hàm đa thức của đầu vào. Nếu mỗi bước tiếp theo của thuật toán là duy nhất thì thuật toán gọi là đơn định. Tất cả thuật toán thuộc lớp P đơn định có thời gian giới hạn là P_time, điều này cho biết chúng sẽ thực hiện trong thời gian đa thức, tương đương với độ phức tạp đa thức trong độ dài đầu vào.

Thuật toán mà ở bước tiếp theo sự tính toán phải lựa chọn giải pháp từ những giới hạn giá trị của hoạt động gọi là không đơn định. Lý thuyết độ phức tạp sử dụng các máy đặc biệt mô tả đặc điểm bằng cách đưa ra kết luận bởi các chuẩn. Máy Turinglà một máy đặc biệt, máy hoạt động trong thời gian rời rạc, tại một thời điểm nó nằm trong khoảng trạng thái đầy đủ số của

tất cả các trạng thái có thể là hữu hạn. Chúng ta có thể định nghĩa hàm độ phức tạp thời gian kết hợp với máy Turing A.

$f_A(n) = \max\{m/A$ kết thúc sau m bước với đầu vào $w = n^3\}$

Chúng ta giả sử rằng A là trạng thái kết thúc đối với tất cả các đầu vào, vấn đề sẽ trở nên khó khăn hơn nếu các trạng thái không nằm trong P . Máy Turing không đơn định hoạt động trong thuật toán NP. Máy Turing không đơn định có thể có một vài trạng thái chính xác. S(w) là trạng thái đo sự thành công ngắn nhất của thuật toán, (Nghĩa là sự tính toán dẫn đến trạng thái cuối cùng)

Hàm số độ phức tạp thời gian của máy Turing không đơn định A được định nghĩa :

$f_A(n)=\max\{1,m/s(w)$ có m bước đối với $w/w=n\}$,

ở mỗi bước máy Turing không đơn định bố trí nhiều bản sao của chính nó như có một vài giải pháp và tính toán độc lập với mọi lời giải.

Các thuật toán thuộc lớp NP là không đơn định và có thể tính toán trên máy Turing không đơn định trong thời gian P.

3.Lý thuyết toán học.

3.1 Modular số học.

Về cơ bản $a \equiv b(\text{mod } n)$ nếu $a = b+kn$ trong đó k là một số nguyên. Nếu a và b dương và a nhỏ hơn n, bạn có thể nghĩ rằng a là phần dư của b khi chia cho n. Nói chung a và b đều là phần dư khi chia cho n. Đôi khi b gọi là thặng dư của a, modulo n, đôi khi a gọi là đồng dư của b, modulo n.

Tập hợp các số nguyên từ 0 đến n-1 còn được gọi là tập hợp thặng dư hoàn toàn modulo n. Điều này có nghĩa là, với mỗi số nguyên a, thì thặng dư modulo n là một số từ 0 đến n-1.

Modulo số học cũng giống như số học bình thường, bao gồm các phép giao hoán, kết hợp và phân phối. Mặt khác giảm mỗi giá trị trung gian trong suốt quá trình tính toán.

$(a+b) \bmod n = ((a \bmod n) + (b \bmod n)) \bmod n$

$(a- b) \bmod n = ((a \bmod n) - (b \bmod n)) \bmod n$

$(a \times b) \bmod n = ((a \bmod n) \times (b \bmod n)) \bmod n$

$(a \times (b + c)) \bmod n = (((a \times b) \bmod n) + ((a \times c) \bmod n)) \bmod n$

Hệ thống mã hoá sự dụng nhiều sự tính toán modulo n, bởi vì vấn đề này giống như tính toán logarithm rời rạc và diện tích hình vuông là khó khăn. Mặt khác nó làm việc dễ hơn, bởi vì nó bị giới hạn trong tất cả giá trị trung gian và kết quả. Ví dụ : a là một số k bits, n là kết quả trung gian của phép cộng, trừ, nhân sẽ không vượt quá 24 bits. Như vậy chúng ta có thể thực hiện hàm mũ trong modulo số học mà không cần sinh ra kết quả trung gian đồ sộ.

3.2 Số nguyên tố.

Số nguyên tố là một số lớn hơn 1, nhưng chỉ chia hết cho 1 và chính nó, ngoài ra không còn số nào nó có thể chia hết nữa. Số 2 là một số nguyên tố. Do vậy 7, 17, 53, 73, 2521, 2365347734339 cũng là số nguyên tố. Số lượng số nguyên tố là vô tận. Hệ mật mã thường sử dụng số nguyên tố lớn cỡ 512 bits và thậm chí lớn hơn như vậy.

3.3 Ước số chung lớn nhất.

Hai số gọi là cặp số nguyên tố khi mà chúng không có thừa số chung nào khác 1, hay nói một cách khác, nếu ước số chung lớn nhất của a và n là bằng 1. Chúng ta có thể viết như sau :

$\gcd(a,n)=1$

Số 15 và 28 là một cặp số nguyên tố, nhưng 15 và 27 thì không phải cặp số nguyên tố do có ước số chung là 1 và 3, dễ dàng thấy 13 và 500 cũng là một

cặp số nguyên tố. Một số nguyên tố là một cặp số nguyên tố với tất cả những số khác loại trừ những số là bội số.

Một cách dễ nhất để tính toán ra ước số chung lớn nhất của hai số là nhờ vào thuật toán Euclid. Knuth mô tả thuật toán và một vài mô hình của thuật toán đã được sửa đổi.

Dưới đây là đoạn mã nguồn trong ngôn ngữ C.

```
/* Thuật toán tìm ước số chung lớn nhất của x và y, giả sử x,y>0 */
int  gcd(int x, int y)
{
     int  g;
     if(x<0)
          x=-x;
     if(y<0)
          y=-y ;
     g=y;
     while(x>0){
          g=x;
          x=y%x;
          y=g;
     }
     return g;
}
```

Thuật toán sau đây có thể sinh ra và trả lại ước số chung lớn nhất của một mảng m số.

```
int multiple gcd ( int m, int *x)
{
     size  t, i ;
     int  g;
     if(m<1)
          return(0);
```

```
        g = x[0];
        for(i=1;i<m;++i){
            g=gcd(g,x[i]);
            if(g==1)
                return 1;
        }
        return g;
}
```

3.4 Số nghịch đảo Modulo.

Số nghịch đảo của 10 là 1/10, bởi vì 10 × 1/10=1. Trong số học modulo thì vấn đề nghịch đảo phức tạp hơn.

$4 \times x \equiv 1 \bmod 7$

Phương trình trên tương đương với tìm x và k sao cho

$4x = 7k+1$

với điều kiện là cả x và k đều là số nguyên.

Vấn đề chung đặt ra tại đây là tìm x sao cho

$1 = (a \times x) \bmod n$

có thể viết lại như sau :

$a^{-1} \equiv x(\bmod n)$

Sự thu nhỏ vấn đề Modulo là rất khó giải quyết. Đôi khi nó là một vấn đề, nhưng đôi khi lại không phải vậy.

Ví dụ : nghịch đảo của 5 modulo 14 là 3 bởi

$5 \times 3 = 15 \equiv 1 \pmod{14}$.

Trong trường hợp chung $a^{-1} \equiv x \pmod n$ chỉ có duy nhất một giải pháp nếu a và n là một cặp số nguyên tố. Nếu a và n không phải là cặp số nguyên tố, thì $a^{-1} \equiv x \pmod n$ không có giải pháp nào. Thuật toán Euclid có thể tính ra được số nghịch đảo của số Modulo n, đôi khi thuật toán này còn gọi là thuật toán Euclid mở rộng. Sau đây thuật toán được mô tả trong ngôn ngữ C.

```
static void Update(int *un,int *vn, int q)
{
     int tn;

     tn = *un-vn*q;
     *un = *vn;
     *vn = tn;
}

int extended euclidian(int u,int v,int u1_out,int u2_out)
{
     int u1=1;
     int u3=u;
     int v1=0;
     int v3=v;
     int q;

     while(v3>0){
          q=u3/v3;
          Update(&u1,&v1,q);
          Update(&u3,&v,q);
     }
     *u1_out=u1;
     *u2_out=(u3-u1*u)/v;
     return u3;
}
```

3.5 Ký hiệu La grăng (Legendre Symboy)

Ký hệu L(a,p) được định nghĩa khi a là một số nguyên và p là mộ t số nguyên tố lớn hơn 2. Nó nhận ba giá trị 0, 1, -1 :

L(a,p) = 0 nếu a chia hết cho p.

L(a,p) = 1 nếu a là thặng dư bậc 2 mod p.

L(a,p) = -1 nếu a không thặng dư mod p.

Một phương pháp dễ dàng để tính toán ra L(a,p) là :

$L(a,p) = a^{(p-1)/2} \bmod p$

3.6 Ký hiệu Jacobi (Jacobi Symboy)

Ký hiệu Jacobi được viết J(a,n), nó là sự khái quát hoá của ký hiệu Lagrăng, nó định nghĩa cho bất kỳ cặp số nguyên a và n. Ký hiệu Jacobi là một chức năng trên tập hợp số thặng dư thấp của ước số n và có thể tính toán theo công thức sau:

- Nếu n là số nguyên tố, thì J(a,n) = 1 với điều kiện a là thặng dư bậc hai modulo n .
- Nếu n là số nguyên tố, thì J(a,n) = -1 với điều kiện a không là thặng dư bậc hai modulo n .
- Nếu n không phải là số nguyên tố thì Jacobi

  $J(a,n)=J(h,p_1) \times J(h,p_2) \times \ldots \times J(h,p_m)$

  với $p_1,p_2 \ldots,p_m$ là các thừa số lớn nhất của n.

Thuật toán này tính ra số Jacobi tuần hoàn theo công thức sau :

1. J(1,k) = 1
2. $J(a \times b,k) = J(a,k) \times J(b,k)$
3. J(2,k) =1 Nếu $(k^2-1)/8$ là chia hết

   J(2,k) =-1 trong các trường hợp khác.
4. J(b,a) = J((b mod a),a)
5. Nếu GCD(a,b)=1 :
   a. $J(a,b) \times J(b,a) = 1$ nếu (a-1)(b-1)/4 là chia hết.
   b. $J(a,b) \times J(b,a) = -1$ nếu (a-1)(b-1)/4 là còn dư.

Sau đây là thuật toán trong ngôn ngữ C :

```
int jacobi(int a,int b)
{
     int a1,a2;
     if(a>=b)
          a%=b;
     if(a==0)
          return 0;
     if(a==1)
          return 1;
     if(a==2)
          if(((b*b-1)/8)%2==0)
               return 1;
          else
               return -1;
     if(a&b&1)   (cả a và b đều là số dư)
          if(((a-1)*(b-1)/4)%2==0)
               return +jacobi(b,a);
          else
               return -jacobi(b,a);
     if(gcd(a,b)==1)
          if(((a-1)*(b-1)/4)%2==0)
          return +jacobi(b,a);
          else
               return -jacobi(b,a);
     factor2(a,&a1,&a2);
     return jacobi(a1,b) * jacobi(a2,b);
}
```

Nếu p là số nguyên tố có cách tốt hơn để tính số Jacobi như dưới đây :

1. Nếu a=1 thì J(a/p)=1
2. Nếu a là số chai hết, thì $J(a,p)=J(a/2,p) \times (-1)^{(p\wedge 2-1)/8}$
3. Nếu a là số dư khác 1 thì $J(a,p)=J(p \bmod a, a) \times (-1)^{(a-1)\times(p-1)/4}$

3.7 Định lý phần dư trung hoa.

Nếu bạn biết cách tìm thừa số nguyên tố của một số n, thì bạn có thể đã sử dụng, một số điều gọi là định lý phần dư trung hoa để giải quyết trong suốt hệ phương trình. Bản dịch cơ bản của đinh lý này được khám phá bởi toán học Trung Hoa vào thế kỷ thứ nhất.

Giả sử, sự phân tích thừa số của $n=p_1 \times p_2 \times . . . \times p_t$ thì hệ phương trình

$$(X \bmod p_i) = a_i \text{ , với } i=1,2,. . .t$$

có duy nhất một cách giải, tại đó x nhỏ hơn n.

Bởi vậy, với a,b tuỳ ý sao cho a < p và b < q (p,q là số nguyên tố) thì tồn tại duy nhất a,x ,khi x nhỏ hơn $p \times q$ thì

$$x \equiv a \pmod p, \text{ và } x \equiv b \pmod q$$

Để tìm ra x đầu tiên sử dụng thuật toán Euclid để tìm u, ví dụ :

$$u \times q \equiv 1 \pmod p$$

Khi đó cần tính toán :

$$x=(((a-b) \times u) \bmod p) \times q + b$$

Dưới đây là đoạn mã định lý phần dư trung hoa trong ngôn ngữ C :

```
Int chinese remainder(size t r, int *m, int *u)
{
     size t i;
     int modulus;
     int n;
     modulus = 1;
     for ( i=0; i<r:++i )
          modulus *=m[i];
          n=0;
     for ( i=0; i<r:++i )
     {
     n+=u[i]*modexp(modulus/m[i],totient(m[i]),m[i]);
```

```
            n%=modulus;
        }
        return n;
}
```

3.8 Định lý Fermat.

Nếu m là số nguyên tố, và a không phải là bội số của m thì định lý Fermat phát biểu :

$$a^{m-1} \equiv 1 (\text{mod } m)$$

4. Các phép kiểm tra số nguyên tố.

Hàm một phía là một khái niệm cơ bản của mã hoá công khai, việc nhân hai số nguyên tố được phỏng đoán như là hàm một phía, nó rất dễ dàng nhân các số để tạo ra một số lớn, nhưng rất khó khăn để phân tích số lớn đó ra thành các thừa số là hai số nguyên tố lớn.

Thuật toán mã hoá công khai cần thiết tới những số nguyên tố. Bất kỳ mạng kích thước thế nào cũng cần một số lượng lớn số nguyên tố. Có một vài phương pháp để sinh ra số nguyên tố. Tuy nhiên có một số vấn đề được đặt ra đối với số nguyên tố như sau :

- Nếu mọi người cần đến những số nguyên tố khác nhau, chúng ta sẽ không đạt được điều đó đúng không. Không đúng, bởi vì trong thực tế có tới 10^{150} số nguyên tố có độ dài 512 bits hoặc nhỏ hơn.
- Điều gì sẽ xảy ra nếu có hai người ngẫu nhiên chọn cùng một số nguyên tố?. Với sự chọn lựa từ số lượng 10^{150} số nguyên tố, điều kỳ quặc này xảy ra là xác xuất nhỏ hơn so với sự tự bốc cháy của máy tính. Vậy nó không có gì là đáng lo ngại cho bạn hết.

4.1 Soloway-Strassen

Soloway và Strassen đã phát triển thuật toán có thể kiểm tra số nguyên tố. Thuật toán này sử dụng hàm Jacobi.

Thuật toán kiểm tra số p là số nguyên tố :

1. Chọn ngẫu nhiên một số a nhỏ hơn p.
2. Nếu ước số chung lớn nhất gcd(a,p) ≠ 1 thì p là hợp số.
3. Tính $j = a^{(p-1)/2} \bmod p$.
4. Tính số Jacobi J(a,p).
5. Nếu j ≠ J(a,p), thì p không phải là số nguyên tố.
6. Nếu j = J(a,p) thì nói p có thể là số nguyên tố với chắc chắn 50%.

Lặp lại các bước này n lần, với những n là giá trị ngẫu nhiên khác nhau của a. Phần dư của hợp số với n phép thử là không quá 2^n.

Thực tế khi thực hiện chương trình, thuật toán chạy với tốc độ nhanh.

4.2 Rabin-Miller

Thuật toán này được phát triển bởi Rabin, dựa trên một phần ý tưởng của Miller. Thực tế những phiên bản của thuật toán đã được giới thiệu tại NIST. (National Institute of Standards and Technology).

Đầu tiên là chọn ngẫu nhiên một số p để kiểm tra. Tính b, với b là số mũ của 2 chia cho p-1. Tiếp theo tính m tương tự như $n = 1+2^b m$.

Sau đây là thuật toán :

1. Chọn một sô ngẫu nhiên a, và giả sử a nhỏ hơn p.
2. Đặt j=0 và $z=a^m \bmod p$.
3. Nếu z=1, hoặc z=p-1 thì pđã qua bước kiểm tra và có thể là số nguyên tố.
4. Nếu j > 0 và z=1 thì p không phải là số nguyên tố.
5. Đặt j = j+1. Nếu j < b và z ≠ p-1 thì đặt $z=z^2 \bmod p$ và trở lại bước 4.
6. Nếu j = b và z ≠ p-1, thì p không phải là số nguyên tố.

4.3 Lehmann.

Một phương pháp đơn giản hơn kiểm tra số nguyên tố được phát triển độc lập bởi Lehmann. Sau đây là thuật toán với số bước lặp là 100.

1. Chọn ngẫu nhiên một số n để kiểm tra.
2. Chắc chắn rằng n không chia hết cho các số nguyên tố nhỏ như 2,3,5,7 và 11.
3. Chọn ngẫu nhiên 100 số $a_1, a_2, \ldots, a_{100}$ giữa 1 và n-1.
4. Tính $a_i^{(n-1)/2}$ (mod n) cho tất cả $a_i = a_1 \ldots a_{100}$. Dừng lại nếu bạn tìm thấy a_i sao cho phép kiểm tra là sai.
5. Nếu $a_i(n-1)/2 = 1$ (mod n) với mọi i, thì n có thể là hợp số.

   Nếu $a_i(n-1)/2 \neq 1$ hoặc -1 (mod n) với i bất kỳ, thì n là hợp số.

   Nếu $a_i(n-1)/2 = 1$ hoặc -1 (mod n) với mọi $i \neq 1$, thì n là số nguyên tố.

4.4 Strong Primes.

Strong Primes thường được sử dụng cho hai số p và q, chúng là hai số nguyên tố với các thuộc tính chắc chắn rằng có thể tìm được thừa số bằng phương pháp phân tích thừa số. Trong số các thuộc tính đạt được bao gồm

+ Ước số chung lớn nhất của p-1 và q-1 là nhỏ.
+ Hai số p-1 và q-1 nên có thừa số nguyên tố lớn, đạo hàm riêng p' và q'
+ Hai số p'-1 và q'-1 nên có thừa số nguyên tố lớn, đạo hàm riêng p" và q"
+ Cả (p-1)/2 và (q-1)/2 nên là số nguyên tố.

Trong bất cứ trường hợp nào Strong Primes rất cần thiết là đối tượng trong các buổi tranh luận. Những thuộc tính đã được thiết kế cản trở một vài thuật toán phân tích thừa số. Hơn nữa, những thuật toán phân tích thừa số nhanh nhất có cơ hội tốt để đạt các tiêu chuẩn.

Chương II **Mật mã**

Trong chương trước chúng ta đã nêu ra các khái niệm cơ bản về lý thuyết thông tin, về độ phức tạp của thuật toán, và những khái niệm cơ bản về toán học cần thiết. Chương này sẽ mô tả một cách tổng quan về mã hoá, bao gồm những khái niệm về mã hoá thông tin, một hệ thống mã hoá bao gồm những thành phần nào, khái niệm protocol, các loại protocol. Mã hoá dòng là gì, mã hoá khối là gì, thế nào là hệ thống mã hoá cổ điển, thế nào là hệ thống mã hoá công khai. Và cuối cùng là bằng những cách nào kẻ địch tấn công hệ thống mã hoá. Những vấn đề sẽ được đề cập trong chương này:

- ☐ Khái niệm cơ bản của mã hoá.
- ☐ Protocol
- ☐ Mã dòng , mã khối (CFB, CBC)
- ☐ Các hệ mật mã đối xứng và công khai
- ☐ Các cách thám mã

1. Khái niệm cơ bản.

-Bản rõ (plaintext or cleartext)

Chứa các xâu ký tự gốc, thông tin trong bản rõ là thông tin cần mã hoá để giữ bí mật.

-Bản mã (ciphertext)

Chứa các ký tự sau khi đã được mã hoá, mà nội dung được giữ bí mật.

-Mật mã học (Crytography)

Là nghệ thuật và khoa học để giữ thông tin được an toàn.

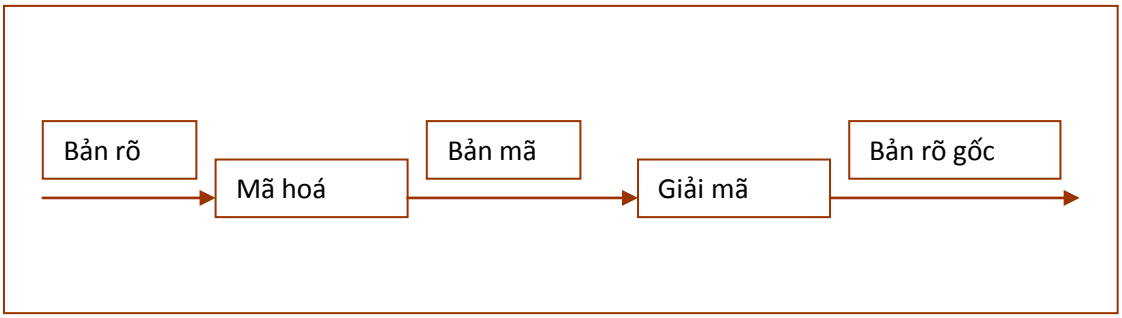
-Sự mã hoá (Encryption)

Quá trình che dấu thông tin bằng phương pháp nào đó để làm ẩn nội dung bên trong gọi là sự mã hoá.

-Sự giải mã (Decryption)

Quá trình biến đổi trả lại bản mã bản thành bản rõ gọi là giải mã.

Quá trình mã hoá và giải mã được thể hiện trong sơ đồ sau:



-Hệ mật mã : là một hệ bao gồm 5 thành phần (P, C, K, E, D) thoả mãn các tính chất sau

P (Plaintext) là tập hợp hữu hạn các bản rõ có thể.

C (Ciphertext) là tập hợp hữu hạn các bản mã có thể.

K (Key) là tập hợp các bản khoá có thể.

E (Encrytion) là tập hợp các qui tắc mã hoá có thể.

D (Decrytion) là tập hợp các qui tắc giải mã có thể.

Chúng ta đã biết một thông báo thường được tổ chức dưới dạng bản rõ. Người gửi sẽ làm nhiệm vụ mã hoá bản rõ, kết quả thu được gọi là bản mã. Bản mã này được gửi đi trên một đường truyền tới người nhận sau khi nhận được bản mã người nhận giải mã nó để tìm hiểu nội dung.

Dễ dàng thấy được công việc trên khi sử dụng định nghĩa hệ mật mã :

$$E_K(P) = C \text{ và } D_K(C) = P$$

2. Protocol

2.1 Giới thiệu Protocol

Trong suốt cả quá trình của hệ thống mật mã là giải quyết các vấn đề, những vấn đề của hệ bao gồm: giải quyết công việc xung quanh sự bí mật, tính

không tin cậy và những kẻ bất lương. Bạn có thể học mọi điều về thuật toán cũng như các kỹ thuật, nhưng có một điều rất đáng quan tâm đó là Protocol. ***Protocol là một loạt các bước, bao gồm hai hoặc nhiều người, thiết kế để hoàn thành nhiệm vụ*** . "Một loạt các bước" nghĩa là Protocol thực hiện theo một tuần tự, từ khi bắt đầu cho tới lúc kết thúc. Mỗi bước phải được thực hiện tuần tự và không có bước nào được thực hiện trước khi bước trước đó đã hoàn thành. "Bao gồm hai hay nhiều người" nghĩa là cần ít nhất hai người hoàn thành protocol, một người không thể tạo ra được một Protocol. Và chắc chắn rằng một người có thể thực hiện một loạt các bước để hoàn thành nhiệm vụ, nhưng đó không phải là Protocol. Cuối cùng "thiết kế để hoàn thành nhiệm vụ" nghĩa là mỗi Protocol phải làm một vài điều gì đó.

Protocol có một vài thuộc tính khác như sau :

1. Mọi người cần phải trong một Protocol, phải biết protocol đó và tuân theo tất cả mọi bước trong sự phát triển.
2. Mọi người cần phải trong một Protocol, và phải đồng ý tuân theo nó.
3. Một Protocol phải rõ ràng, mỗi bước phải được định nghĩa tốt và phải không có cơ hội hiểu nhầm.
4. Protocol phải được hoàn thành, phải có những hành động chỉ rõ cho mỗi trường hợp có thể.

2.2 Protocol mật mã.

Protocol mật mã là protocol sử dụng cho hệ thống mật mã. Một nhóm có thể gồm những người bạn bè và những người hoàn toàn tin cậy khác hoặc họ có thể là địch thủ hoặc những người không tin cậy một chút nào hết. Một điều hiển nhiên là protocol mã hoá phải bao gồm một số thuật toán mã hoá,

nhưng mục đích chung của protocol là một điều gì đó xa hơn là điều bí mật đơn giản.

2.3 Mục đích của Protocol.

Trong cuộc sống hàng ngày, có rất nhiều nghi thức thân mật cho hầu hết tất cả mọi điều như gọi điện thoại, chơi bài, bầu cử. Không có gì trong số chúng lại không có protocol, chúng tiến triển theo thời gian, mọi người đều biết sử dụng chúng như thế nào và làm việc với chúng.

Hơn nữa bây giờ mọi người giao tiếp với nhau qua mạng máy tính thay cho sự gặp mặt thông thường. Máy tính cần thiết một nghi thức chuẩn để làm những việc giống nhau như con người không phải suy nghĩ. Nếu bạn đi từ một địa điểm này tới địa điểm khác, thậm chí từ quốc gia này tới quốc gia khác, bạn thấy một trạm điện thoại công cộng khác hoàn toàn so với cái bạn đã sử dụng, bạn dễ dàng đáp ứng. Nhưng máy tính thì không mềm dẻo như vậy.

Thật ngây thơ khi bạn tin rằng mọi người trên mạng máy tính là chân thật, và cũng thật ngây thơ khi tin tưởng rằng người quản trị mạng, người thiết kế mạng là chân thật. Hầu hết sẽ là chân thật, nhưng nó sẽ là không chân khi bạn cần đến sự an toàn tiếp theo. ***Bằng những protocol chính thức, chúng ta có thể nghiên cứu những cách mà những kẻ không trung thực có thể lừa đảo và phát triển protocol để đánh bại những kẻ lừa đảo đó.*** Protocol rất hữa ích bởi vì họ trừu tượng hoá tiến trình hoàn thành nhiệm vụ từ kỹ thuật, như vậy nhiệm vụ đã được hoàn thành.

Sự giao tiếp giữa hai máy tính giống như một máy tính là IBM PC, máy kia là VAX hoặc loại máy tương tự. Khái niệm trừu tượng này cho phép chúng ta nghiên cứu những đặc tính tốt của protocol mà không bị xa lầy vào sự thực hiện chi tiết. Khi chúng ta tin rằng chúng ta có một protocol tốt, thì

chúng ta có thể thực hiện nó trong mọi điều từ một máy tính đến điện thoại, hay đến một lò nướng bánh thông minh.

2.4 Truyền thông sử dụng hệ mật mã đối xứng.

Hai máy thực hiện việc truyền thông an toàn như thế nào ? Chúng sẽ mã hoá sự truyền thông đó, đương nhiên rồi. Để hoàn thành một protocol là phức tạp hơn việc truyền thông. Chúng ta hãy cùng xem xétđiều gì sẽ xảy ra nếu máy Client muốn gửi thông báo mã hoá tới cho Server.

1. Client và Server đồng ý sử dụng một hệ mã hóa.
2. Client và Server thống nhất khoá với nhau.
3. Client lấy bản rõ và mã hoá sử dụng thuật toán mã hoá và khoá. Sau đó bản mã đã được tạo ra.
4. Client gửi bản mã tới cho Server.
5. Server giải mã bản mã đó với cùng một thuật toán và khoá, sau đó đọc được bản rõ.

Điều gì sẽ xảy ra đối với kẻ nghe trộm cuộc truyền thông giữa Client và Server trong protocol trên. Nếu như kẻ nghe trộm chỉ nghe được sự truyền đi bản mã trong bước 4, chúng sẽ cố gắng phân tích bản mã. Những kẻ nghe trộm chúng không ngu rốt, chúng biết rằng nếu có thể nghe trộm từ bước 1 đến bước 4 thì chắc chắn sẽ thành công. Chúng sẽ biết được thuật toán và khoá như vậy chúng sẽ biết được nhiều như Server. Khi mà thông báo được truyền đi trên kênh truyền thông trong bước thứ 4, thì kẻ nghe trộm sẽ giải mã bằng chính những điều đã biết.

Đây là lý do tại sao quản lý khoá lại là vấn đề quan trọng trong hệ thống mã hoá. Một hệ thống mã hoá tốt là mọi sự an toàn phụ thuộc vào khoá và không phụ thuộc vào thuật toán. Với thuật toán đối xứng, Client và Server có thể thực hiện bước 1 là công khai, nhưng phải thực hiện bước 2 bí mật.

Khoá phải được giữ bí mật trước, trong khi, và sau protocol, mặt khác thông báo sẽ không giữ an toàn trong thời gian dài.

Tóm lại, hệ mật mã đối xứng có một vài vấn đề như sau :

- Nếu khoá bị tổn thương (do đánh cắp, dự đoán ra, khám phá, lỗi lộ) thì đối thủ là người có khoá, anh ta có thể giải mã tất cả thông báo với khoá đó. Một điều rất quan trọng là thay đổi khoá tuần tự để giảm thiểu vấn đề này.
- Những khoá phải được thảo luận bí mật. Chúng có thể có giá trị hơn bất kỳ thông báo nào đã được mã hoá, từ sự hiểu biết về khoá có nghĩa là hiểu biết về thông báo.
- Sử dụng khoá riêng biệt cho mỗi cặp người dùng trên mạng vậy thì tổng số khoá tăng lên rất nhanh giống như sự tăng lên của số người dùng. Điều này có thể giải quyết bằng cách giữ số người dùng ở mức nhỏ, nhưng điều này không phải là luôn luôn có thể.

2.5 Truyền thông sử dụng hệ mật mã công khai.

- Hàm một phía (one way function)

Khái niệm hàm một phía là trung tâm của hệ mã hoá công khai. Không có một Protocol cho chính nó, hàm một phía là khối xây dựng cơ bản cho hầu hết các mô tả protocol.

Một hàm một phía là hàm mà dễ dàng tính toán ra quan hệ một chiều nhưng rất khó để tính ngược lại. Ví như : biết giả thiết x thì có thể dễ dàng tính ra f(x), nhưng nếu biết f(x) thì rất khó tính ra được x. Trong trường hợp này "khó" có nghĩa là để tính ra được kết quả thì phải mất hàng triệu năm để tính toán, thậm chí tất cả máy tính trên thế giới này đều tính toán công việc đó.

Vậy thì hàm một phía tốt ở những gì ? Chúng ta không thể sử dụng chúng cho sự mã hoá. Một thông báo mã hoá với hàm một phía là không hữu ích,

bất kỳ ai cũng không giải mã được. Đối với mã hoá chúng ta cần một vài điều gọi là cửa sập hàm một phía.

Cửa sập hàm một phía là một kiểu đặc biệt của hàm một phía với cửa sập bí mật. Nó dễ dàng tính toán từ một điều kiện này nhưng khó khăn để tính toán từ một điều kiện khác. Nhưng nếu bạn biết điều bí mật, bạn có thể dễ dàng tính toán ra hàm từ điều kiện khác. Ví dụ : tính f(x) dễ dàng từ x, rất khó khăn để tính toán x ra f(x). Hơn nữa có một vài thông tin bí mật, y giống như f(x) và y nó có thể tính toán dễ dàng ra x. Như vậy vấn đề có thể đã được giải quyết.

Hộp thư là một ví dụ rất tuyệt về cửa sập hàm một phía. Bất kỳ ai cũng có thể bỏ thư vào thùng. Bỏ thư vào thùng là một hành động công cộng. Mở thùng thư không phải là hành động công cộng. Nó là khó khăn, bạn sẽ cần đến mỏ hàn để phá hoặc những công cụ khác. Hơn nữa nếu bạn có điều bí mật (chìa khoá), nó thật dễ dàng mở hộp thư. Hệ mã hoá công khai có rất nhiều điều giống như vậy.

☐ Hàm băm một phía.

Hàm băm một phía là một khối xây dựng khác cho nhiều loại protocol. Hàm băm một phía đã từng được sử dụng cho khoa học tính toán trong một thời gian dài. Hàm băm là một hàm toán học hoặc loại khác, nó lấy chuỗi đầu vào và chuyển đổi thành kích thước cố định cho chuỗi đầu ra.

Hàm băm một phía là một hàm băm nó sử dụng hàm một phía. Nó rất dễ dàng tính toán giá trị băm từ xâu ký tự vào, nhưng rất khó tính ra một chuỗi từ giá trị đơn lẻ đưa vào.

Có hai kiểu chính của hàm băm một phía, hàm băm với khoá và không khoá. Hàm băm một phía không khoá có thể tính toán bởi mọi người giá trị băm là hàm chỉ có đơn độc chuỗi đưa vào. Hàm băm một phía với khoá là hàm cả

hai thứ chuỗi vào và khoá, chỉ một vài người có khoá mới có thể tính toán giá trị băm.

☐ Hệ mã hoá sử dụng khoá công khai.

Với những sự mô tả ở trên có thể nghĩ rằng thuật toán đối xứng là an toàn. Khoá là sự kết hợp, một vài người nào đó với sự kết hợp có thể mở sự an toàn này, đưa thêm tài liệu vào, và đóng nó lại. Một người nào đó khác với sự kết hợp có thể mở được và lấy đi tài liệu đó.

Năm 1976 Whitfied và Martin Hellman đã thay đổi vĩnh viễn mô hình của hệ thống mã hoá. Chúng được mô tả là hệ mã hoá sử dụng khoá công khai. Thay cho một khoá như trước, hệ bao gồm hai khoá khác nhau, một khoá là công khai và một khoá kia là khoá bí mật. Bất kỳ ai với khoá công khai cũng có thể mã hoá thông báo nhưng không thể giải mã nó. Chỉ một người với khoá bí mật mới có thể giải mã được.

Trên cơ sở toán học, tiến trình này phụ thuộc vào cửa sập hàm một phía đã được trình bày ở trên. Sự mã hoá là chỉ thị dễ dàng. Lời chỉ dẫn cho sự mã hoá là khoá công khai, bất kỳ ai cũng có thể mã hoá. Sự giải mã là một chỉ thị khó khăn. Nó tạo ra khó khăn đủ để một người sử dụng máy tính Cray phải mất hàng ngàn năm mới có thể giải mã. Sự bí mật hay cửa sập chính là khoá riêng. Với sự bí mật, sự giải mã sẽ dễ dàng như sự mã hoá.

Chúng ta hãy cùng xem xét khi máy Client gửi thông báo tới Server sử dụng hệ mã hoá công khai.

1. Client và Server nhất trí sử dụng hệ mã hóa công khai.
2. Server gửi cho Client khoá công khai của Server.
3. Client lấy bản rõ và mã hoá sử dụng khoá công khai của Server. Sau đó gửi bản mã tới cho Server.
4. Server giải mã bản mã đó sử dụng khoá riêng của mình.

Chú ý rằng hệ thống mã hoá công khai giải quyết vấn đề chính của hệ mã hoá đối xứng, bằng cách phân phối khoá. Với hệ thống mã hoá đối xứng đã qui ước, Client và Server phải nhất trí với cùng một khoá. Client có thể chọn ngẫu nhiên một khoá, nhưng nó vẫn phải thông báo khoá đó tới Server, điều này gây lãng phí thời gian. Đối với hệ thống mã hoá công khai, thì đây không phải là vấn đề.

3. Khoá

3.1 Độ dài khoá.

Độ an toàn của thuật toán mã hoá cổ điển phụ thuộc vào hai điều đó là độ dài của thuật toán và độ dài của khoá. Nhưng độ dài của khoá dễ bị lộ hơn.

Giả sử rằng độ dài của thuật toán là lý tưởng, khó khăn lớn lao này có thể đạt được trong thực hành. Hoàn toàn có nghĩa là không có cách nào bẻ gãy được hệ thống mã hoá trừ khi cố gắng thử với mỗi khoá. Nếu khoá dài 8 bits thì có $2^8 = 256$ khoá có thể. Nếu khoá dài 56 bits, thì có 2^{56} khoá có thể. Giả sử rằng siêu máy tính có thể thực hiện 1 triệu phép tính một giây, nó cũng sẽ cần tới 2000 năm để tìm ra khoá thích hợp. Nếu khoá dài 64 bits, thì với máy tính tương tự cũng cần tới xấp xỉ 600,000 năm để tìm ra khoá trong số 2^{64} khoá có thể. Nếu khoá dài 128 bits, nó cần tới 10^{25} năm , trong khi vũ trụ của chúng ta chỉ tồn tại cỡ 10^{10} năm. Như vậy với 10^{25} năm có thể là đủ dài.

Trước khi bạn gửi đi phát minh hệ mã hoá với 8 Kbyte độ dài khoá, bạn nên nhớ rằng một nửa khác cũng không kém phần quan trọng đó là thuật toán phải an toàn nghĩa là không có cách nào bẻ gãy trừ khi tìm được khoá thích hợp. Điều này không dễ dàng nhìn thấy được, hệ thống mã hoá nó như một nghệ thuật huyền ảo.

Một điểm quan trọng khác là độ an toàn của hệ thống mã hoá nên phụ thuộc vào khoá, không nên phụ thuộc vào chi tiết của thuật toán. Nếu độ dài của hệ thống mã hoá mới tin rằng trong thực tế kẻ tấn công không thể biết nội dung

bên trong của thuật toán. Nếu bạn tin rằng giữ bí mật nội dung của thuật toán, tận dụng độ an toàn của hệ thống hơn là phân tích những lý thuyết sở hữu chung thì bạn đã nhầm. Và thật ngây thơ hơn khi nghĩ rằng một ai đó không thể gỡ tung mã nguồn của bạn hoặc đảo ngược lại thuật toán.

Giả sử rằng một vài kẻ thám mã có thể biết hết tất cả chi tiết về thuật toán của bạn. Giả sử rằng họ có rất nhiều bản mã, như họ mong muốn. Giả sử họ có một khối lượng bản rõ tấn công với rất nhiều dữ liệu cần thiết. Thậm chí giả sử rằng họ có thể lựa chọn bản rõ tấn công Nếu như hệ thống mã hoá của có thể dư thừa độ an toàn trong tất cả mọi mặt, thì bạn đã có đủ độ an toàn bạn cần.

Tóm lại câu hỏi đặt ra trong mục này là : ***Khoá nên dài bao nhiêu.***

Trả lời câu hỏi này phụ thuộc vào chính những ứng dụng cụ thể của bạn. Dữ liệu cần an toàn của bạn dài bao nhiêu ? Dữ liệu của bạn trị giá bao nhiêu ? ... Thậm chí bạn có thể chỉ chỉ rõ những an toàn cần thiết theo cách sau.

> Độ dài khoá phải là một trong 2^{32} khoá để tương ứng với nó là kẻ tấn công phải trả 100.000.000 $ để bẻ gãy hệ thống.

3.2 Quản lý khoá công khai.

Trong thực tế, quản lý khoá là vấn đề khó nhất của an toàn hệ mã hoá. Để thiết kế an toàn thuật toán mã hoá và protocol là một việc là không phải là dễ dàng nhưng để tạo và lưu trữ khoá bí mật là một điều khó hơn. Kẻ thám mã thường tấn công cả hai hệ mã hoá đối xứng và công khai thông qua hệ quản lý khoá của chúng.

Đối với hệ mã hoá công khai việc quản lý khoá dễ hơn đối với hệ mã hoá đối xứng, nhưng nó có một vấn đề riêng duy nhất. Mỗi người chỉ có một khoá công khai, bất kể số ngư ời ở trên mạng là bao nhiêu. Nếu Eva muốn gửi thông báo đến cho Bob, thì cô ấy cần có khoá công khai của Bob. Có một vài phương pháp mà Eva có thể lấy khoá công khai của Bob :

- ☐ Eva có thể lấy nó từ Bob.
- ☐ Eva có thể lấy từ trung tâm cơ sở dữ liệu.
- ☐ Eva có thể lấy từ cơ sở dữ liệu riêng của cô ấy.

Chứng nhận khoá công khai :

Chứng nhận khoá công khai là xác định khoá thuộc về một ai đó, được quản lý bởi một người đáng tin cậy. Chứng nhận để sử dụng vào việc cản trở sự cống gắng thay thế một khoá này bằng một khoá khác. Chứng nhận của Bob, trong sơ sở dữ liệu khoá công khai, lưu trữ nhiều thông tin hơn chứ không chỉ là khoá công khai. Nó lưu trữ thông tin về Bob như tên, địa chỉ, ... và nó được viết bởi ai đó mà Eva tin tưởng, người đó thường gọi là CA(certifying authority). Bằng cách xác nhận cả khoá và thông tin về Bob. CA xác nhận thông tin về Bob là đúng và khoá công khai thuộc quyền sở hữu của Bob. Eva kiểm tra lại các dấu hiệu và sau đó cô ấy có thể sử dụng khoá công khai, sự an toàn cho Bob và không một ai khác biết. Chứng nhận đóng một vai trò rất quan trọng trong protocol của khoá công khai.

Quản lý khoá phân phối :

Trong một vài trường hợp, trung tâm quản lý khoá có thể không làm việc. Có lẽ không có một CA (certifying authority) nào mà Eva và Bob tin tưởng. Có lẽ họ chỉ tin tưởng bạn bè thân thiết hoặc họ không tin tưởng bất cứ ai. Quản lý khoá phân phối, sử dụng trong những chương trình miền công khai, giải quyết vấn đề này với người giới thiệu (introducers). Người giới thiệu là một trong những người dùng khác của hệ thống anh ta là người nhận ra khoá công khai của bạn anh ta.

Ví dụ :

Khi Bob sinh ra khoá công khai, anh ta đưa bản copy cho bạn anh ấy là Bin và Dave. Họ đều biết Bob, vì vậy họ có khoá của Bob và đưa cho các dấu hiệu của anh ta. Bây giờ Bob đưa ra khoá công khai của anh ta cho người lạ,

giả sử đó là Eva, Bob đưa ra khoá cùng với các dấu hiệu của hai người giới thiệu. Mặt khác nếu Eva đã biết Bin hoặc Dave, khi đó cô ta có lý do tin rằng khoá của Bob là đúng. Nếu Eva không biết Bin hoặc Dave thì cô ấy không có lý do tin tưởng khoá của Bob là đúng.

Theo thời gian, Bob sẽ tập hợp được nhiều người giới thiệu như vậy khoá của anh ta sẽ được biết đến rộng rãi hơn. Lợi ích của kỹ thuật này là không cần tới trung tâm phân phối khoá, mọi người đều có sự tín nhiệm, khi mà Eva nhận khoá công khai của Bob, sẽ không có sự bảo đảm nào rằng cô ấy sẽ biết bất kỳ điều gì của người giới thiệu và hơn nữa không có sự đảm bảo nào là cô ấy sẽ tin vào sự đúng đắn của khoá.

4. Mã dòng, mã khối (CFB, CBC)

4.1 Mô hình mã hoá khối.

Mã hoá sử dụng các thuật toán khối gọi đó là mã hoá khối, thông thường kích thước của khối là 64 bits. Một số thuật toán mã hoá khối sẽ được trình bày sau đây.

4.1.1 Mô hình dây truyền khối mã hoá.

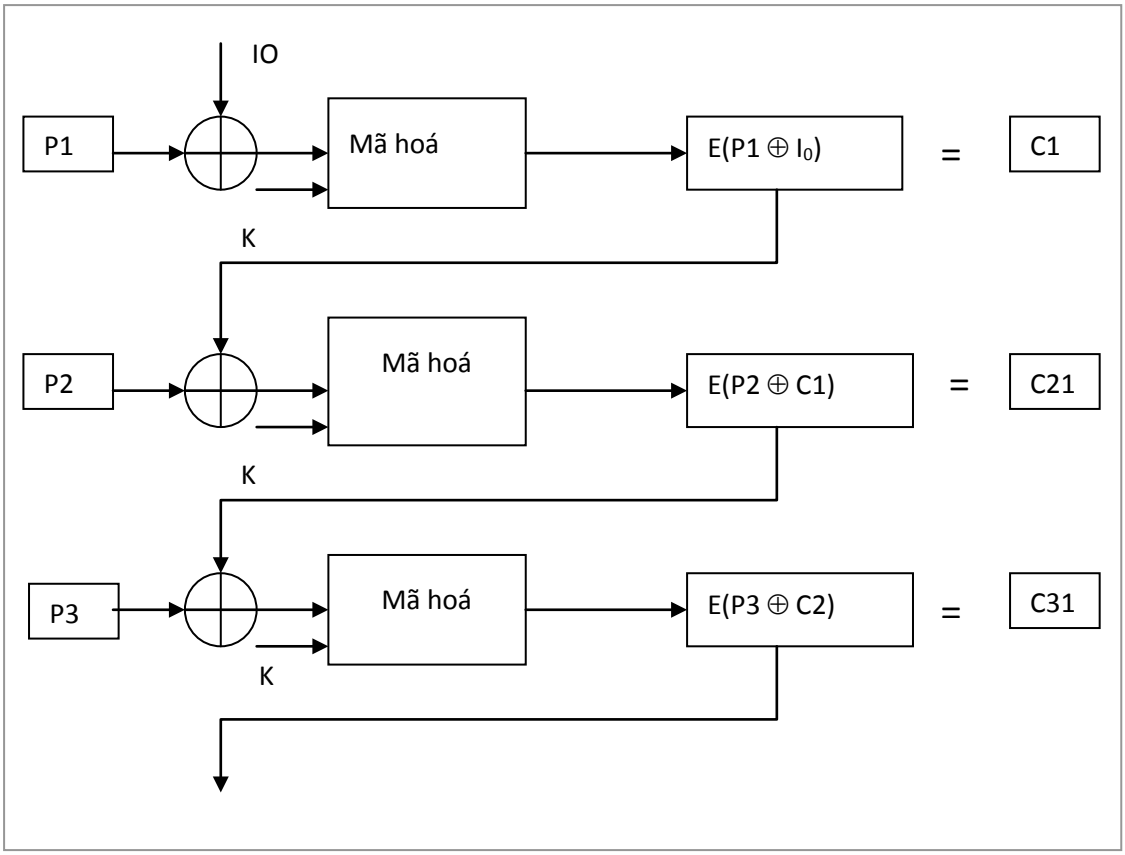
Dây truyền sử dụng kỹ thuật thông tin phản hồi, bởi vì kết quả của khối mã hoá trước lại đưa vào khối mã hoá hiện thời. Nói một cách khác khối trước đó sử dụng để sửa đổi sự mã hoá của khối tiếp theo. Mỗi khối mã hoá không phụ thuộc hoàn toàn vào khối của bản rõ.

Trong dây truyền khối mã hoá (Cipher Block Chaining Mode), bản rõ đã được XOR với khối mã hoá kế trước đó trước khi nó được mã hoá. Hình

4.1.1 thể hiện các bước trong dây truyền khối mã hoá.

Sau khi khối bản rõ được mã hoá, kết quả của sự mã hoá được lưu trữ trong thanh ghi thông tin phản hồi. Trước khi khối tiếp theo của bản rõ được mã hoá, nó sẽ XOR với thanh ghi thông tin phản hồi để trở thành đầu vào cho tuyến mã hoá tiếp theo. Kết quả của sự mã hoá tiếp tục được lưu trữ trong

thanh ghi thông tin phản hồi, và tiếp tục XOR với khối bản rõ tiếp theo, tiếp tục như vậy cho tới kết thúc thông báo. Sự mã hoá của mỗi khối phụ thuộc vào tất cả các khối trước đó.



Hình 4.1.1 Sơ đồ mô hình dây chuyền khối mã hoá .

Sự giải mã là cân đối rõ ràng. Một khối mã hoá giải mã bình thường và mặt khác được cất giữ trong thanh ghi thông tin phản hồi. Sau khi khối tiếp theo được giải mã nó XOR với kết quả của thanh ghi phản hồi. Như vậy khối mã hoá tiếp theo được lưa trữ trong thanh ghi thông tin phản hồi, tiếp tục như vậy cho tới khi kết thúc thông báo.

Công thức toán học của quá trình trên như sau :

$C_i = E_K(P_i \text{ XOR } C_{i-1})$

$P_i = C_{i-1} \text{ XOR } D_K(C_i)$

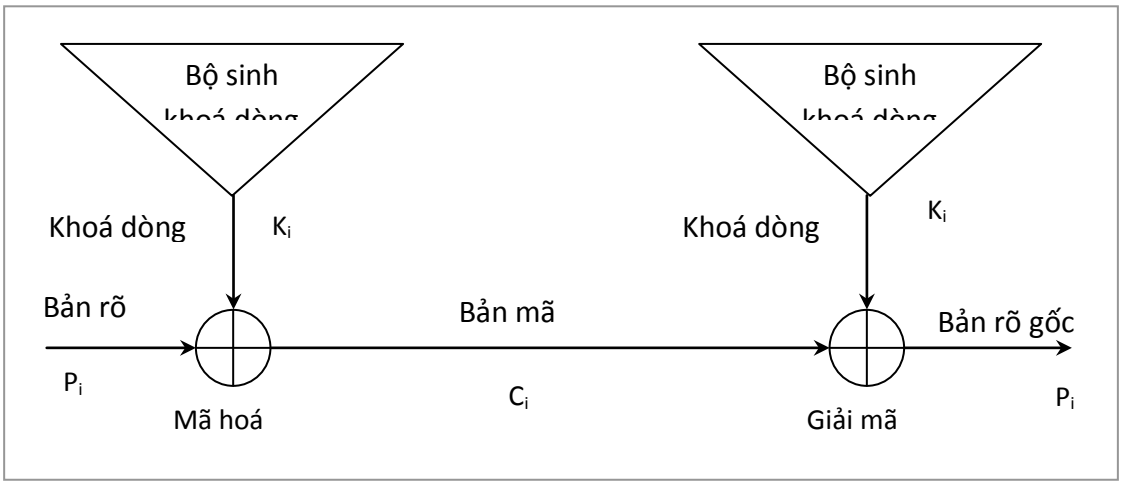
4.1.2 Mô hình mã hoá với thông tin phản hồi.

Trong mô hình dây truyền khối mã hoá(CBC_Cipher Block Chaining Mode), sự mã hóa không thể bắt đầu cho tới khi hoàn thành nhận được một khối dữ liệu. Đây thực sự là vấn đề trong một vài mạng ứng dụng. Ví dụ, trong môi trường mạng an toàn, một thiết bị đầu cuối phải truyền mỗi ký tự tới máy trạm như nó đã được đưa vào. Khi dữ liệu phải xử lý như một khúc kích thước byte, thì mô hình dây truyền khối mã hoá là không thoả đáng.

Tại mô hình CFB dữ liệu là được mã hóa trong một đơn vị nhỏ hơn là kích thước của khối. Ví dụ sẽ mã hoá một ký tự ASCII tại một thời điểm (còn gọi là mô hình 8 bits CFB) nhưng không có gì là bất khả kháng về số 8. Bạn có thể mã hoá 1 bit dữ liệu tại một thời điểm, sử dụng thuật toán 1 bit CFB.

4.2 Mô hình mã hoá dòng.

Mã hóa dòng là thật toán, chuyển đổi bản rõ sang bản mã là 1 bit tại mỗi thời điểm. Sự thực hiện đơn giản nhất của mã hoá dòng được thể hiện trong hình 4.2



Hình 4.2 Mã hoá dòng.

Bộ sinh khoá dòng là đầu ra một dòng các bits : k_1, k_2, k_3, . . . k_i. Đây là khoá dòng đã được XOR với một dòng bits của bản rõ, p_1, p_2, p_3, . . p_i, để đưa ra dòng bits mã hoá.

$$c_i = p_i \text{ XOR } k_i$$

Tại điểm kết thúc của sự giải mã, các bits mã hoá được XOR với khoá dòng để trả lại các bits bản rõ.

$$p_i = c_i \text{ XOR } k_i$$

Từ lúc p_i XOR k_i XOR k_i = p_i là một công việc tỉ mỉ.

Độ an toàn của hệ thống phụ thuộc hoàn toàn vào bên trong bộ sinh khoá dòng. Nếu đầu ra bộ sinh khoá dòng vô tận bằng 0, thì khi đó bản rõ bằng bản mã và cả quá trình hoạt động sẽ là vô dụng. Nếu bộ sinh khoá dòng sinh ra sự lặp lại 16 bits mẫu, thì thuật toán sẽ là đơn giản với độ an toàn không đáng kể.

Nếu bộ sinh khoá dòng là vô tận của dòng ngẫu nhiên các bits, bạn sẽ có một vùng đệm (one time-pad) và độ an toàn tuyệt đối.

Thực tế mã hoá dòng nó nằm đâu đó giữa XOR đơn giản và một vùng đệm. Bộ sinh khoá dòng sinh ra một dòng bits ngẫu nhiên, thực tế điều này quyết định thuật toán có thể hoàn thiện tại thời điểm giải mã. Đầu ra của bộ sinh khoá dòng là ngẫu nhiên, như vậy người phân tích mã sẽ khó khăn hơn khi

bẻ gãy khoá. Như bạn đã đoán ra được rằng, tạo một bộ sinh khoá dòng mà sản phẩm đầu ra ngẫu nhiên là một vấn đề không dễ dàng.

5. Các hệ mật mã đối xứng và công khai

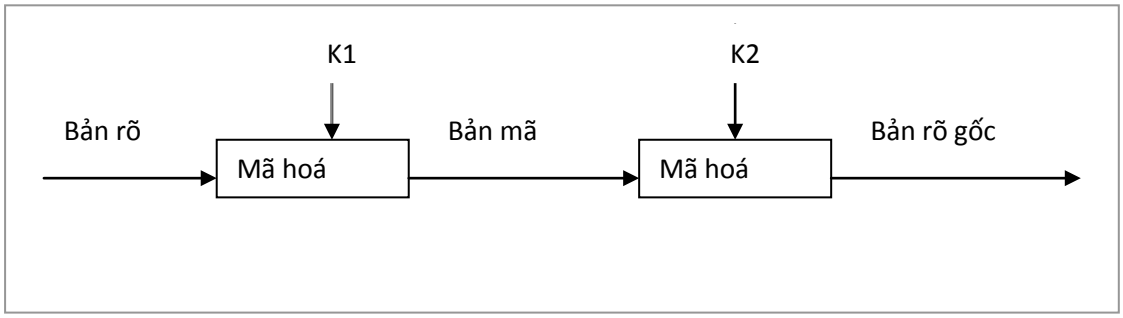
5.1 Hệ mật mã đối xứng

Thuật toán đối xứng hay còn gọi thuật toán mã hoá cổ điển là thuật toán mà tại đó khoá mã hoá có thể tính toán ra được từ khoá giải mã. Trong rất nhiều trường hợp, khoá mã hoá và khoá giải mã là giống nhau. Thuật toán này còn có nhiều tên gọi khác như thuật toán khoá bí mật, thuật toán khoá đơn giản, thuật toán một khoá. Thuật toán này yêu cầu người gửi và người nhận phải thoả thuận một khoá trước khi thông báo được gửi đi, và khoá này phải được cất giữ bí mật. Độ an toàn của thuật toán này vẫn phụ thuộc và khoá, nếu để lộ ra khoá này nghĩa là bất kỳ người nào cũng có thể mã hoá và giải mã thông báo trong hệ thống mã hoá.

Sự mã hoá và giải mã của thuật toán đối xứng biểu thị bởi :

$$E_K(P) = C$$

$$D_K(C) = P$$



Hình 5.1 Mã hoá và giải mã với khoá đối xứng .

Trong hình vẽ trên thì :

K1có thể trùng K2, hoặc

K1 có thể tính toán từ K2, hoặc

K2 có thể tính toán từ K1.

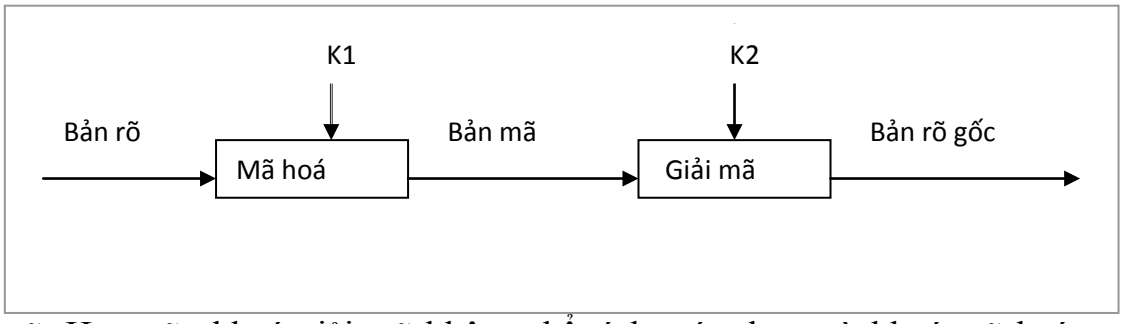
Một số nhược điểm của hệ mã hoá cổ điển

- Các phương mã hoá cổ điển đòi hỏi người mã hoá và người giải mã phải cùng chung một khoá. Khi đó khoá phải được giữ bí mật tuyệt đối, do vậy ta dễ dàng xác định một khoá nếu biết khoá kia.
- Hệ mã hoá đối xứng không bảo vệ được sự an toàn nếu có xác suất cao khoá người gửi bị lộ. Trong hệ khoá phải được gửi đi trên kênh an toàn nếu kẻ địch tấn công trên kênh này có thể phát hiện ra khoá.
- Vấn đề quản lý và phân phối khoá là khó khăn và phức tạp khi sử dụng hệ mã hoá cổ điển. Người gửi và người nhận luôn luôn thông nhất với nhau về vấn đề khoá. Việc thay đổi khoá là rất khó và dễ bị lộ.
- Khuynh hướng cung cấp khoá dài mà nó phải được thay đổi thường xuyên cho mọi người trong khi vẫn duy trì cả tính an toàn lẫn hiệu quả chi phí sẽ cản trở rất nhiều tới việc phát triển hệ mật mã cổ điển.

5.2 Hệ mật mã công khai

Vào những năm 1970 Diffie và Hellman đã phát minh ra một hệ mã hoá mới được gọi là **hệ mã hoá công khai** hay **hệ mã hoá phi đối xứng.**

Thuật toán mã hoá công khai là khác biệt so với thuật toán đối xứng. Chúng được thiết kế sao cho **khoá** sử dụng vào việc mã hoá là khác so với **khoá**



giải mã. Hơn nữa khoá giải mã không thể tính toán được từ khoá mã hoá. Chúng được gọi với tên hệ thống mã hoá công khai bởi vì khoá để mã hoá có thể công khai, một người bất kỳ có thể sử dụng khoá công khai để mã hoá thông báo, nhưng chỉ một vài người có đúng khoá giải mã thì mới có khả năng giải mã. Trong nhiều hệ thống, khoá mã hoá gọi là khoá công khai (public key), khoá giải mã thường được gọi là khoá riêng (private key).

Hình 5.2 Mã hoá và giải mã với hai khoá .

Trong hình vẽ trên thì :

K1 không thể trùng K2, hoặc

K2 không thể tính toán từ K1.

Đặc trưng nổi bật của hệ mã hoá công khai là cả khoá công khai(public key) và bản tin mã hoá (ciphertext) đều có thể gửi đi trên một kênh thông tin không an toàn.

Diffie và Hellman đã xác định rõ các điều kiện của một hệ mã hoá công khai như sau :

1. Việc tính toán ra cặp khoá công khai K_B và bí mật k_B dựa trên cơ sở các điều kiện ban đầu phải được thực hiện một cách dễ dàng, nghĩa là thực hiện trong thời gian đa thức.

2. Người gửi A có được khoá công khai của người nhận B và có bản tin P cần gửi đi thì có thể dễ dàng tạo ra được bản mã C.

$$C = EKB (P) = EB (P)$$

Công việc này cũng trong thời gian đa thức.

3. Người nhận B khi nhận được bản tin mã hóa C với khoá bí mật k_B thì có thể giải mã bản tin trong thời gian đa thức.

$$P = DkB (C) = DB[EB(M)]$$

4. Nếu kẻ địch biết khoá công khai K_B cố gắng tính toán khoá bí mật thì khi đó chúng phải đương đầu với trường hợp nan giải, trường hợp này đòi hỏi nhiều yêu cầu không khả thi về thời gian.
5. Nếu kẻ địch biết được cặp (K_B,C) và cố gắng tính toán ra bản rõ P thì giải quyết bài toán khó với số phép thử là vô cùng lớn, do đó không khả thi.

6. Các cách thám mã

Có sáu phương pháp chung để phân tích tấn công, dưới đây là danh sách theo thứ tự khả năng của từng phương pháp. Mỗi phương pháp trong số chúng giả sử rằng kẻ thám mã hoàn toàn có hiểu biết về thuật toán mã hoá được sử dụng.

1. **Chỉ có bản mã.** Trong trường hợp này, người phân tích chỉ có một vài bản tin của bản mã, tất cả trong số chúng đều đã được mã hoá và cùng sử dụng chung một thuật toán. Công việc của người phân tích là tìm lại được bản rõ của nhiều bản mã có thể hoặc tốt hơn nữa là suy luận ra được khoá sử dụng mã hoá, và sử dụng để giải mã những bản mã khác với cùng khoá này.

   Giả thiết : $C_1 = E_k(P_1)$, $C_2 = E_k(P_2)$, . . .$C_i = E_k(P_i)$

   Suy luận : Mỗi P_1,P_2, . . P_i, k hoặc thuật toán kết luận P_{i+1} từ

$C_{i+1} = E_k(P_{i+1})$

2. **Biết bản rõ.** Người phân tích không chỉ truy cập được một vài bản mã mặt khác còn biết được bản rõ. Công việc là suy luận ra khoá để sử dụng giải mã hoặc thuật toán giải mã để giải mã cho bất kỳ bản mã nào khác với cùng khoá như vậy.

   Giả thiết : P_1, $C_1 = E_k(P_1)$, P2, C2= $E_k(P_2)$, . . . Pi, $C_i = E_k(P_i)$

   Suy luận : Mỗi k hoặc thuật toán kết luận P_{i+1} từ $C_{i+1} = E_k(P_{i+1})$

3. **Lựa chọn bản rõ.** Người phân tích không chỉ truy cập được bản mã và kết hợp bản rõ cho một vài bản tin, nhưng mặt khác lựa chọn bản rõ đã mã hoá. Phương pháp này tỏ ra có khả năng hơn phương pháp **biết bản rõ** bởi vì người phân tích có thể chọn cụ thể khối bản rõ cho mã hoá, một điều khác có thể là sản lượng thông tin về khoá nhiều hơn.

   Giả thiết : P_1, $C_1 = E_k(P_1)$, P2, C2= $E_k(P_2)$, . . . Pi, $C_i = E_k(P_i)$ tại đây người phân tích chọn P_1, P_2,. . . P_i

   Suy luận : Mỗi k hoặc thuật toán kết luận P_{i+1} từ $C_{i+1} = E_k(P_{i+1})$

4. **Mô phỏng lựa chọn bản rõ.** Đây là trường hợp đặc biệt của lựa chọn bản rõ. Không chỉ có thể lựa chọn bản rõ đã mã hoá, nhưng họ còn có thể sửa đổi sự lựa chọn cơ bản kết quả của sự mã hoá lần trước. Trong trường lựa chọn bản mã người phân tích có thể đã chọn một khối lớn bản rõ đã mã hoá, nhưng trong trường hợp này có thể chọn một khối nhỏ hơn và chọn căn cứ khác trên kết quả của lần đầu tiên.

5. **Lựa chọn bản mã.** Người phân tích có thể chọn bản mã khác nhau đã được mã hoá và truy cập bản rõ đã giải mã. Trong ví dụ khi một người phân tích có một hộp chứng cớ xáo chộn không thể tự động giải mã, công việc là suy luận ra khoá.

Giả thiết : C_1, $P_1 = D_k(C_1)$, C2, P2= $D_k(C_2)$, . . . Ci, $P_i = D_k(C_i)$

tại Suy luận : k

6. **Lựa chọn khoá.** Đây không phải là một cách tấn công khi mà bạn đã có khoá. Nó không phải là thực hành thám mã mà chỉ là sự giải mã thông thường, bạn chỉ cần lựa chọn khoá cho phù hợp với bản mã.

Một điểm đáng chú ý khác là đa số các kỹ thuật thám mã đều dùng phương pháp thống kê tần suất xuất hiện của các từ, các ký tự trong bản mã. Sau đó thực hiện việc thử thay thế với các chữ cái có tần suất xuất hiện tương đồng trong ngôn ngữ tự nhiên. Tại đây chúng ta chỉ xem xét đối với ngôn ngữ thông dụng nhất hiện nay đó là tiếng Anh. Việc thống kê tần suất xuất hiện của các ký tự trong trường hợp này được tiến hành dựa trên các bài báo, sách, tạp chí và các văn bản cùng với một số loại khác ...

Sau đây là bảng thống kê tần suất xuất hiện của 26 chữ cái trong bảng chữ cái tiếng Anh theo tài liệu của Beker và Piper.

Ký tự	Xác Suất	Ký tự	Xác suất	Ký tự	Xác suất
A	0.082	J	0.002	S	0.063
B	0.015	K	0.008	T	0.091
C	0.028	L	0.040	U	0.028
D	0.043	M	0.024	V	0.010
E	0.127	N	0.067	W	0.023
F	0.022	O	0.075	X	0.001
G	0.020	P	0.019	Y	0.020
H	0.061	Q	0.001	Z	0.001
I	0.070	R	0.060		

Cùng với việc thống kê các tần xuất của các ký tự trong tiếng Anh, việc thống kê tần suất xuất hiện thường xuyên của các dãy gồm 2 hoặc 3 ký tự liên tiếp nhau cũng có một vai trò quan trọng trong công việc thám mã. Sysu Deck đưa ra 30 bộ đôi xuất hiện thường xuyên của tiếng Anh được sắp theo thứ tự giảm dần như sau :

Tính hữu dụng của các phép thống kê ký tự và các dãy ký tự được người phân tích mã khai thác triệt để trong những lần thám mã. Khi thực hiện việc thám mã người phân tích thống kê các ký tự trong bản mã, từ đó so sánh với bản thống kê mẫu và đưa ra các ký tự phỏng đoán tương tự. Phương pháp này được sử dụng thường xuyên và đem lại hiệu quả khá cao.

Cặp chữ	Tần suất	Cặp chữ	Tần suất	Cặp chữ	Tần suất
TH	10.00	ED	4.12	OF	3.38
HE	9.50	TE	4.04	IT	3.26
IN	7.17	TI	4.00	AL	3.15
ER	6.65	OR	3.98	AS	3.00
RE	5.92	ST	3.81	HA	3.00
ON	5.70	AR	3.54	NG	2.92
AN	5.63	ND	3.52	CO	2.80
EN	4.76	TO	3.50	SE	2.75
AT	4.72	NT	3.44	ME	2.65
ES	4.24	IS	3.43	DE	2.65

Chương III **Hệ mã hoá RSA.**

Với đề tài xây dựng thư viện các hàm mã hoá dùng cho việc bảo mật thông tin trao đổi trong mô hình Client/Server, thì cần thiết một phương pháp mã hoá để áp dụng, thuật toán mã hoá công khai RSA đã được lựa chọn cho giải pháp này. Phương pháp này có những ưu điểm, nhược điểm, đặc tính gì đó là phần sẽ trình bày trong chương này

- ☐ Khái niệm hệ mật mã RSA
- ☐ Phân phối khoá công kkai trong RSA
- ☐ Độ an toàn của hệ RSA
- ☐ Một số tính chất của hệ RSA

1. Khái niệm hệ mật mã RSA

Khái niệm hệ mật mã RSA đã được ra đời năm 1976 bởi các tác giả R.Rivets, A.Shamir, và L.Adleman. Hệ mã hoá này dựa trên cơ sở của hai bài toán :

+ Bài toán Logarithm rời rạc (Discrete logarith)
+ Bài toán phân tích thành thừa số.

Trong hệ mã hoá RSA các bản rõ, các bản mã và các khoá (public key và private key) là thuộc tập số nguyên $Z_N = \{1, \ldots, N-1\}$. Trong đó tập Z_N với $N=p\times q$ là các số nguyên tố khác nhau cùng với phép cộng và phép nhân Modulo N tạo ra modulo số học N.

Khoá mã hoá E_{KB} là cặp số nguyên (N,K_B) và khoá giải mã D_{kb} là cặp số nguyên (N,k_B), các số là rất lớn, số N có thể lên tới hàng trăm chữ số.

Các phương pháp mã hoá và giải mã là rất dễ dàng.

Công việc mã hoá là sự biến đổi bản rõ P (Plaintext) thành bản mã C (Ciphertext) dựa trên cặp khoá công khai K_B và bản rõ P theo công thức sau đây :

$$C = E_{KB}(P) = E_B(P) = P^{KB} \pmod N . \quad (1)$$

Công việc giải mã là sự biến đổi ngược lại bản mã C thành bản rõ P dựa trên cặp khoá bí mật k_B, modulo N theo công thức sau :

$$P = D_{kB}(C) = D_B(C) = C^{kB} \pmod N . \quad (2)$$

Dễ thấy rằng, bản rõ ban đầu cần được biến đổi một cách thích hợp thành bản mã, sau đó để có thể tái tạo lại bản rõ ban đầu từ chính bản mã đó :

$$P = D_B(E_B(P)) \quad (3)$$

Thay thế (1) vào (2) ta có :

$$(P^{KB})^{kB} = P \pmod N \quad (4)$$

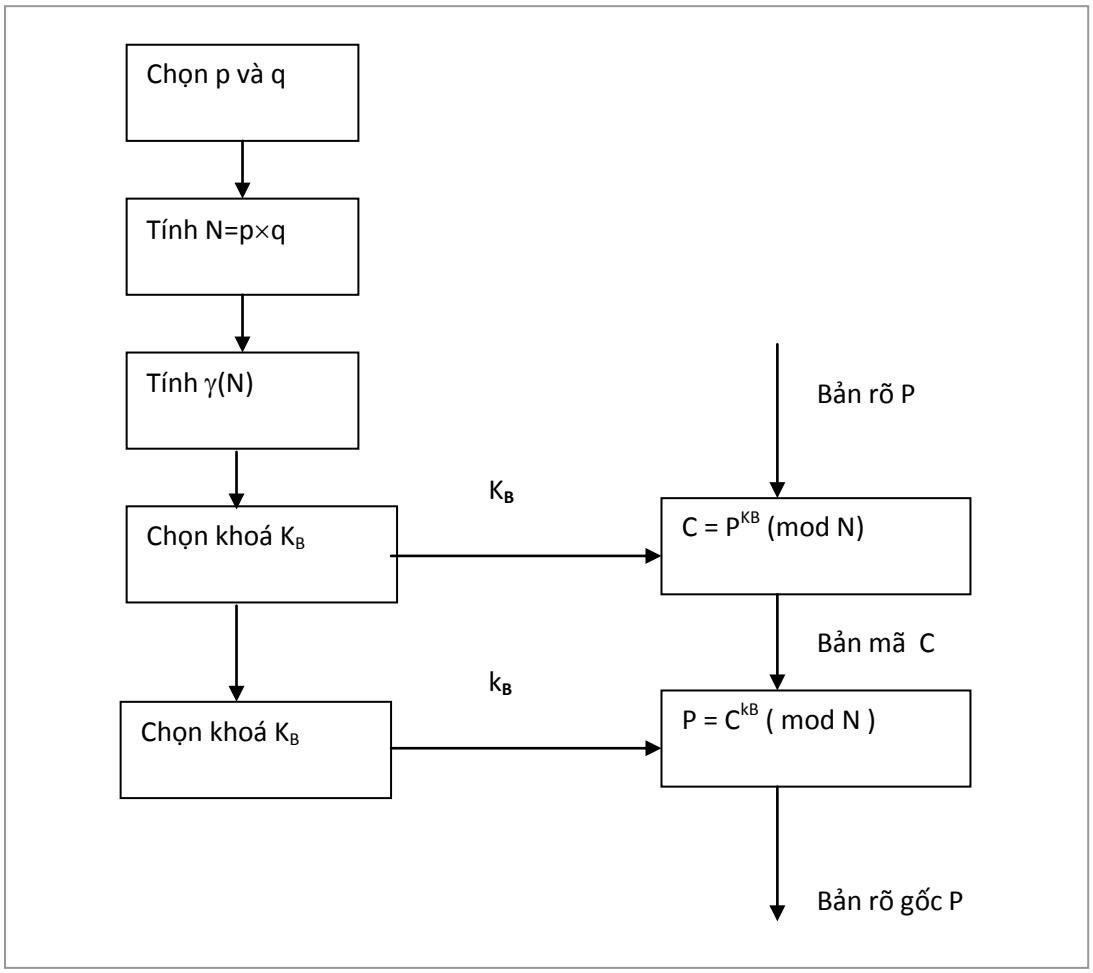
Trong toán học đã chứng minh được rằng, nếu N là số nguyên tố thì công thức (4) sẽ có lời giải khi và chỉ khi $K_B.k_B = 1 \pmod{N-1}$, áp dụng thuật toán ta thấy $N=p\times q$ với p, q là số nguyên tố, do vậy (4) sẽ có lời giải khi và chỉ khi :

$$K_B.k_B \equiv 1 \pmod{\gamma(N)} \quad (5)$$

trong đó $\gamma(N) = LCM(p-1,q-1)$.

LCM (Lest Common Multiple) là bội số chung nhỏ nhất.

Nói một cách khác, đầu tiên người nhận B lựa chọn một khoá công khai K_B một cách ngẫu nhiên. Khi đó khoá bí mật k_B được tính ra bằng công thức (5). Điều này hoàn toàn tính được vì khi B biết được cặp số nguyên tố (p,q) thì sẽ tính được $\gamma(N)$.



Hình 1.1 Sơ đồ các bước thực hiện mã hoá theo thuật toán RSA.

2. Độ an toàn của hệ RSA

Một nhận định chung là tất cả các cuộc tấn công giải mã đều mang mục đích không ốt. Trong phần độ an toàn của hệ mã hoá RSA sẽ đề cập đến một vài phương thức tấn công điển hình của kẻ địch nhằm giải mã trong thuật toán này.

Chúng ta xét đến trường hợp khi kẻ địch nào đó biết được modulo N, khoá công khai K_B và bản tin mã hoá C, khi đó kẻ địch sẽ tìm ra bản tin gốc (Plaintext) như thế nào. Để làm được điều đó kẻ địch thường tấn vào hệ thống mật mã bằng hai phương thức sau đây:

□ Phương thức thứ nhất :

Trước tiên dựa vào phân tích thừa số modulo N. Tiếp theo sau chúng sẽ tìm cách tính toán ra hai số nguyên tố p và q, và có khả năng thành công khi đó sẽ tính được λ(N) và khoá bí mật k_B. Ta thấy N cần phải là tích của hai số nguyên tố, vì nếu N là tích của hai số nguyên tố thì thuật toán phân tích thừa số đơn giản cần tối đa $\sqrt{N}$ bước, bởi vì có một số nguyên tố nhỏ hơn $\sqrt{N}$. Mặt khác, nếu N là tích của n số nguyên tố, thì thuật toán phân tích thừa số đơn giản cần tối đa $N^{1/n}$ bước.

Một thuật toán phân tích thừa số có thể thành phức tạp hơn, cho phép phân tích một số N ra thành thừa số trong $O(\sqrt{P})$ bước, trong đó p là số chia nhỏ nhất của N, việc chọn hai số nguyên tố là cho thuật toán tăng hiệu quả.

□ Phương thức thứ hai :

Phương thức tấn công thứ hai vào hệ mã hoá RSA là có thể khởi đầu bằng cách giải quyết trường hợp thích hợp của bài toán logarit rời rạc. Trường hợp này kẻ địch đã có trong tay bản mã C và khoá công khai K_B tức là có cặp (K_B,C)

Cả hai phương thức tấn công đều cần một số bước cơ bản, đó là :

$O(\exp \sqrt{\ln N \ln(\ln N)})$, trong đó N là số modulo.

3. Một số tính chất của hệ RSA

□ ***Trong các hệ mật mã RSA, một bản tin có thể được mã hoá trong thời gian tuyến tính.***

Đối với các bản tin dài, độ dài của các số được dùng cho các khoá có thể được coi như là lằng. Tương tự như vậy, nâng một số lên luỹ thừa được thực hiện trong thời gian hằng, các số không được phép dài hơn một độ dài hằng. Thực ra tham số này che dấu nhiều chi tiết cài đặt có liên quan đến việc tính toán với các con số dài, chi phí của các phép toán thực sự là một yếu tố ngăn cản sự phổ biến ứng dụng của phương pháp này. Phần quan

trọng nhất của việc tính toán có liên quan đến việc mã hoá bản tin. Nhưng chắc chắn là sẽ không có hệ mã hoá nào hết nếu không tính ra được các khoá của chúng là các số lớn.

□ ***Các khoá cho hệ mã hoá RSA có thể được tạo ra mà không phải tính toán quá nhiều.***

Một lần nữa, ta lại nói đến các phương pháp kiểm tra số nguyên tố. Mỗi số nguyên tố lớn có thể được phát sinh bằng cách đầu tiên tạo ra một số ngẫu nhiên lớn, sau đó kiểm tra các số kế tiếp cho tới khi tìm được một số nguyên tố. Một phương pháp đơn giản thực hiện một phép tính trên một con số ngẫu nhiên, với xác suất 1/2 sẽ chứng minh rằng số được kiểm tra không phải nguyên tố. Bước cuối cùng là tính p dựa vào thuật toán Euclid.

Như phần trên đã trình bày trong hệ mã hoá công khai thì khoá giải mã (private key) k_B và các thừa số p,q là được giữ bí mật và sự thành công của phương pháp là tuỳ thuộc vào kẻ địch có khả năng tìm ra được giá trị của k_B hay không nếu cho trước N và K_B. Rất khó có thể tìm ra được k_B từ K_B cần biết về p và q, như vậy cần phân tích N ra thành thừa số để tính p và q. Nhưng việc phân tích ra thừa số là một việc làm tốn rất nhiều thời gian, với kỹ thuật hiện đại ngày nay thì cần tới hàng triệu năm để phân tích một số có 200 chữ số ra thừa số.

Độ an toàn của thuật toán RSA dựa trên cơ sở những khó khăn của việc xác định các thừa số nguyên tố của một số lớn. Bảng dưới đây cho biết các thời gian dự đoán, giả sử rằng mỗi phép toán thực hiện trong một micro giây.

Số các chữ số trong số được phân tích	Thời gian phân tích
50	4 giờ
75	104 giờ
100	74 năm
200	4.000.000 năm
300	5×10^{15} năm
500	4×10^{25} năm

Chương IV Mô hình Client/Server

Trong thực tế, mô hình Client/Server đã trở nên rất phổ biến trong hệ thống mạng điểm tới điểm, và chúng được áp dụng hầu hết cho những máy tính truyền thông ngày nay. Kiến trúc mô hình Client/Server và khi nào cần mã hoá thông tin truyền trong Client/Server là chủ đề sẽ được trình bày trong chương này.

1.Mô hình Client/Server

Nói chung, một ứng dụng khởi tạo truyền thông từ điểm tới điểm được gọi là client. Người dùng cuối thường xuyên gọi phần mềm client khi họ cần tới những dịch vụ trên mạng. Mô hình Client/Server cố gắng tổ chức lại các máy PC, trên mạng cục bộ, để thích hợp với các máy tính lớn mainframe, tăng tính thích ứng, tính hiệu quả của hệ thống. Mặc dù có sự thay đổi rất lớn các quan điểm về mô hình Client/Server, nhưng chúng có một vài đặc tính dưới đây.

- Máy Client là các máy PC hay là các workstations, tập trung vào mạng và sử dụng các tài nguyên trên mạng.
- Giao diện người sử dụng với Client, nói chung sử dụng giao diện người dùng đồ hoạ (GUI), ví như Microsoft Windowns
- Trong hệ thống Client/Server có một vài Client, với mỗi Client sử dụng giao diện riêng của mình. Các Client sử dụng các tài nguyên được chia sẻ bởi Server.
- Server có thể là một workstation lớn, như mainframe, minicomputer, hoặc các thiết bị mạng LAN.
- Client có thể gửi các truy vấn hoặc các lệnh tới Server, nhưng thực hiện tiến trình này không phải là Client.
- Server trả lại kết quả trên màn hình của Client.

- Các loại Server thông thường là : database server, file server, print server, image-processing server, computing server và communication server.
- Server không thể khởi tạo bất kỳ công việc nào, nhưng nó thực hiện các yêu cầu to lớn của Client.
- Nhiệm vụ chia là hai phần : phần mặt trước thực hiện bởi client, và phần mặt sau thực hiện bởi Server.
- Server thực hiện việc chia sẻ File, lưu trữ và tìm ra các thông tin, mạng và quản lý tài liệu, quản lý thư điện tử, bảng thông báo và văn bản video.

2. Mã hoá trong mô hình Client/Server.

Trong mô hình Client/Server việc trao đổi thông tin diễn ra thường xuyên nên rất dễ bị kẻ xấu lợi dụng, bởi vậy bảo vệ thông tin trên đường truyền là vô cùng quan trọng, chúng đảm bảo thông tin trên đường truyền là đúng đắn. Tại mô hình này mỗi khi những yêu cầu được gửi từ Client đến Server hoặc khi Server gửi trả lại kết quả cho Client thì những thông tin này đều được mã hoá trong khi truyền.

Chương V Xây dựng hàm thư viện

Xu hướng trên thế giới hiện nay là phần mềm được bán và phân phối ở dạng các modul phần mềm. Các hình thức của modul phụ thuộc vào các gói phần mềm cụ thể và các ngôn ngữ mà người sử dụng dùng. Ví dụ bạn có thể tạo các thư viện tĩnh với các file có phần mở rộng .LIB hoặc bạn có thể tạo một điều khiển ActiveX với phần mở rộng OCX, hoặc hơn nữa bạn có thể tạo các thư viện liên kết động với các file .DLL .

Các ngôn ngữ lập trình hiện nay có tính modul độc lập rất cao, nghĩa là bạn có thể tạo ra các ứng dụng bằng cách kết hợp nhiều modul phần mềm độc lập nhau thành một ứng dụng cụ thể. Thông thường khi thiết kế một phần mềm ứng dụng thuộc loại phức tạp, bạn sẽ tìm kiếm các modul có thể sử dụng được để giảm chi phí, giảm thời gian thiết kế và tập chung nhiều hơn cho những phần ứng dụng tự bạn viết ra.

Một câu hỏi đặt ra tại đây là vì sao chúng ta lại không tạo ra các hàm thực hiện các công việc chuyên biệt và phân phối nó cho người sử dụng, có một vài lý do sau đây không cho phép thực hiện điều này :

- Người dùng có thể vô tình thay đổi làm xáo trộn các lệnh trong chương trình.
- Bạn không muốn người dùng biết "bí quyết" của bạn mà chỉ muốn họ sử dụng kết quả bạn tạo ra.

Trong chương này của cuốn luận văn trình bày thư viện liên kết động là gì, và chúng thực hiện như thế nào. Thư viện liên kết động DLL (Dynamic Link Library) là một tập tin thư viện chứa các hàm. Người lập trình có thể gọi một tập tin DLL vào trong chương trình của họ và sử dụng các hàm trong DLL đó.

DLL là một thư viện liên kết động với các chương trình sử dụng nó, nghĩa là khi bạn tạo ra tập tin EXE của chương trình mà không cần liên kết tập tin DLL với chương trình của bạn. Tập tin DLL sẽ được liên kết động với

chương trình trong thời gian thi hành chương trình. Bởi vậy khi viết một ứng dụng có sử dụng DLL, bạn phải phân phối tập tin DLL cùng với tập tin EXE của chương trình bạn viết.

1.Xây dựng thư viện liên kết động CRYPTO.DLL

Thư viện **crypto.dll** được xây dựng dới đây cung cấp cho các bạn các hàm cần thiết phục vụ cho việc mã hoá thông tin, chúng bao gồm

int enciph(char *, char *) : hàm mã hoá.

int deciph(char *, char *) : hàm gải mã.

□ Hàm Enciph.c

Các bạn có thể sử dụng hàm này để thực hiện các thao tác mã hoá với xâu kí tự, bằng cách đưa vào một xâu ký tự (bản rõ) ở đầu ra bạn sẽ nhận được một xâu ký tự đã được mã hoá (bản mã). Với bản mã này các bạn có thể yên tâm về nội dụng thông tin sẽ rất khó bị lộ. Hàm thực hiện có sử dụng khoá công khai lấy vào từ File PUBLIC.KEY.

```
//============================
// Ham Enciph.c
#include <stdio.h>
#include <conio.h>
#include <miracl.h>
#include <stdlib.h>
#include <string.h>

/*
#define RSA
*/
int enciph(char  *sin,char *sout)
{  /*  encipher using public key  */
    big x,ke;
    FILE *ifile;
```

```
    int ch,i,leng;
    long seed;
    miracl *mip=mirsys(100,0);
    x=mirvar(0);
    ke=mirvar(0);
    mip->IOBASE=60;

    if ((ifile=fopen("public.key","r"))==NULL)
    {
         return 1;
    }
    cinnum(ke,ifile);
    fclose(ifile);
    seed=123456789;
    irand(seed);
    bigrand(ke,x);
    leng=strlen(sin);
    for(i=0; i <= (leng-1); i++)
    { /* encipher character by character */
#ifdef RSA
        power(x,3,ke,x);
#else
        mad(x,x,x,ke,ke,x);
#endif
        ch=*(sin+i);
        ch^=x[1];          /* XOR with last byte of x */
        sout[i]=ch;
     }
    return 0;
}
//============================
miracl *mirsys(int nd,mr_small nb)
{  /*  Initialize MIRACL system to   *
```

```
    *  use numbers to base nb, and   *
    *  nd digits or (-nd) bytes long */
    int i;
    mr_small b;
    mr_mip=(miracl *)mr_alloc(1,sizeof(miracl));
    mr_mip->depth=0;
    mr_mip->trace[0]=0;
    mr_mip->depth++;
    mr_mip->trace[mr_mip->depth]=25;
if (MIRACL>=MR_IBITS) mr_mip->TOOBIG =(1<<(MR_IBITS-2));
    else
    mr_mip->TOOBIG =(1<<(MIRACL-1));

#ifdef MR_FLASH
    mr_mip->BTS=MIRACL/2;
    if (mr_mip->BTS==MR_IBITS) mr_mip->MSK=(-1);
    else  mr_mip->MSK=(1<<(mr_mip->BTS))-1;
    #endif

#ifdef MR_NO_STANDARD_IO
    mr_mip->ERCON=TRUE;
#else
    mr_mip->ERCON=FALSE;
#endif
    mr_mip->N=0;
    mr_mip->MSBIT=((mr_small)1<<(MIRACL-1));
    mr_mip->OBITS=mr_mip->MSBIT-1;
    mr_mip->user=NULL;
    mr_set_align(0);

#ifdef MR_NOFULLWIDTH
    if (nb==0)
    {
```

```
            mr_berror(MR_ERR_BAD_BASE);
            mr_mip->depth--;
            return mr_mip;
        }
#endif
        if (nb==1 || nb>MAXBASE)
        {
            mr_berror(MR_ERR_BAD_BASE);
            mr_mip->depth--;
            return mr_mip;
        }
        mr_setbase(nb);
        b=mr_mip->base;
        mr_mip->lg2b=0;
        mr_mip->base2=1;
        if (b==0)
        {
            mr_mip->lg2b=MIRACL;
            mr_mip->base2=0;
        }
        else while (b>1)
        {
            b/=2;
            mr_mip->lg2b++;
            mr_mip->base2*=2;
        }
        if (nd>0)
            mr_mip->nib=(nd-1)/mr_mip->pack+1;
        else
            mr_mip->nib=(mr_mip->lg2b-8*nd-1)/mr_mip->lg2b;
        if (mr_mip->nib<2) mr_mip->nib=2;
#ifdef MR_FLASH
        mr_mip->workprec=mr_mip->nib;
```

```
    mr_mip->stprec=mr_mip->nib;
while(mr_mip->stprec>2 && mr_mip->stprec> MR_FLASH/
      mr_mip->lg2b)
    mr_mip->stprec=(mr_mip->stprec+1)/2;
    if (mr_mip->stprec<2) mr_mip->stprec=2;
    mr_mip->pi=NULL;
#endif
    mr_mip->check=ON;
    mr_mip->IOBASE=10; mr_mip->ERNUM=0;
    mr_mip->RPOINT=OFF;
    mr_mip->NTRY=6;
    mr_mip->EXACT=TRUE;
    mr_mip->TRACER=OFF;
    mr_mip->INPLEN=0;
    mr_mip->PRIMES=NULL;
    mr_mip->IOBUFF=mr_alloc(MR_IOBSIZ+1,1);
for (i=0;i<NK;i++) mr_mip->ira[i]=0L;
    irand(0L);
mr_mip->nib=2*mr_mip->nib+1;
#ifdef MR_FLASH
    if (mr_mip->nib!=(mr_mip->nib&(mr_mip->MSK)) || mr_mip-
>nib > mr_mip->TOOBIG)
#else
    if(mr_mip->nib!=(mr_mip->nib&(mr_mip->OBITS)) ||
       mr_mip->nib>mr_mip->TOOBIG)
#endif
    {
        mr_berror(MR_ERR_TOO_BIG);
        mr_mip->nib=(mr_mip->nib-1)/2;
        mr_mip->depth--;
        return mr_mip;
    }
    mr_mip->modulus=NULL;
```

```
    mr_mip->A=NULL;
    mr_mip->B=NULL;
    mr_mip->fin=FALSE;
    mr_mip->fout=FALSE;
    mr_mip->active=ON;
    mr_mip->w0=mirvar(0); /* w0 is double length  */
    mr_mip->nib=(mr_mip->nib-1)/2;
#ifdef MR_KCM
    mr_mip->big_ndash=NULL;
    mr_mip->ws=mirvar(0);
#endif
    mr_mip->w1=mirvar(0); /* initialize workspace */
    mr_mip->w2=mirvar(0);
    mr_mip->w3=mirvar(0);
    mr_mip->w4=mirvar(0);
    mr_mip->nib=2*mr_mip->nib+1;
    mr_mip->w5=mirvar(0);
    mr_mip->w6=mirvar(0);
    mr_mip->w7=mirvar(0);
    mr_mip->nib=(mr_mip->nib-1)/2;
    mr_mip->w5d=&(mr_mip->w5[mr_mip->nib+1]);
    mr_mip->w6d=&(mr_mip->w6[mr_mip->nib+1]);
    mr_mip->w7d=&(mr_mip->w7[mr_mip->nib+1]);

    mr_mip->w8=mirvar(0);
    mr_mip->w9=mirvar(0);
    mr_mip->w10=mirvar(0);
    mr_mip->w11=mirvar(0);
    mr_mip->w12=mirvar(0);
    mr_mip->w13=mirvar(0);
    mr_mip->w14=mirvar(0);
    mr_mip->w15=mirvar(0);
    mr_mip->depth--;
```

```
    return mr_mip;
}
//============================
flash mirvar(int iv)
{ /* initialize big/flash number */
    flash x;
    if (mr_mip->ERNUM) return NULL;
    mr_mip->depth++;
    mr_mip->trace[mr_mip->depth]=23;
    if (mr_mip->TRACER) mr_track();
    if (!(mr_mip->active))
    {
        mr_berror(MR_ERR_NO_MIRSYS);
        mr_mip->depth--;
        return NULL;
    }
    x=(mr_small *)mr_alloc(mr_mip->nib+1,sizeof(mr_small));
    if (x==NULL)
    {
        mr_berror(MR_ERR_OUT_OF_MEMORY);
        mr_mip->depth--;
        return x;
    }
    convert(iv,x);
    mr_mip->depth--;
    return x;
}
//============================
int cinnum(flash x,FILE *filep)
{ /* convert from string to flash x */
    int n;
    if (mr_mip->ERNUM) return 0;
    mr_mip->depth++;
```

```
    mr_mip->trace[mr_mip->depth]=14;
    if (mr_mip->TRACER) mr_track();
    mr_mip->infile=filep;
    mr_mip->fin=TRUE;
    n=cinstr(x,NULL);
    mr_mip->fin=FALSE;
    mr_mip->depth--;
    return n;
}
//============================
void power(flash x,int n,flash w)
{
    copy(x,mr_mip->w8);
    zero(w);
    if (mr_mip->ERNUM || size(mr_mip->w8)==0) return;
    convert(1,w);
    if (n==0) return;
    mr_mip->depth++;
    mr_mip->trace[mr_mip->depth]=51;
    if (mr_mip->TRACER) mr_track();
    if (n<0)
    {
        n=(-n);
        frecip(mr_mip->w8,mr_mip->w8);
    }
    if (n==1)
    {
        copy(mr_mip->w8,w);
        mr_mip->depth--;
        return;
    }
    forever
    {
```

```
        if (n%2!=0) fmul(w,mr_mip->w8,w);
        n/=2;
        if (mr_mip->ERNUM || n==0) break;
        fmul(mr_mip->w8,mr_mip->w8,mr_mip->w8);
    }
    mr_mip->depth--;
}
//============================
void mad(big x,big y,big z,big w,big q,big r)
{
    if (mr_mip->ERNUM) return;
    mr_mip->depth++;
    mr_mip->trace[mr_mip->depth]=24;
    if (mr_mip->TRACER) mr_track();
    mr_mip->check=OFF;
    if (w==r)
    {
        mr_berror(MR_ERR_BAD_PARAMETERS);
        mr_mip->depth--;
        return;
    }
    multiply(x,y,mr_mip->w0);
    if (x!=z && y!=z)add(mr_mip->w0,z,mr_mip->w0);

    divide(mr_mip->w0,w,q);
    if (q!=r) copy(mr_mip->w0,r);
    mr_mip->check=ON;
    mr_mip->depth--;
}
//============================
```

☐ Hàm Deciph.c

Hàm sử dụng để thực hiện các thao tác giải mã hoá với xâu kí tự đã được mã hoá bằng hàm enciph.c ở trên, bằng cách đa vào một xâu ký tự đã mã hoá (bản mã) ở đầu ra bạn sẽ nhận lại một xâu ký tự ban đầu (bản rõ gốc). Hàm thực hiện có sử dụng khoá bí mật lấy vào từ File PRIVATE.KEY. Hai File PUBLIC.KEY và PRIVATE.KEY chúng cùng được sinh ra do chương trình genkey, chúng có quan hệ mật thiết với nhau và không thể tách rời, nếu có khoá công khai mà không có khoá bí mật thì cũng không thể giải mã được, còn nếu có khoá bí mật mà không có khoá công khai thì cũng chẳng ích lợi gì.

```
//============================
//Deciph.c
#include <stdio.h>
#include <miracl.h>
#include <stdlib.h>
#include <string.h>

int deciph(char *strinputde, char *stroutputde)
{
     /*  decipher using private key  */
    big x,y,ke,p,q,n,a,b,alpha,beta,t;
    FILE *ifile;
    int ch,i,leng;
    long ipt;
    miracl *mip=mirsys(100,0);
    x=mirvar(0);
    ke=mirvar(0);
    p=mirvar(0);
    q=mirvar(0);
    n=mirvar(0);
    y=mirvar(0);
```

```
    alpha=mirvar(0);
    beta=mirvar(0);
    a=mirvar(0);
    b=mirvar(0);
    t=mirvar(0);
    mip->IOBASE=60;
    if ((ifile=fopen("private.key","r"))==NULL)
    {
          return 1;
    }
    cinnum(p,ifile);
    cinnum(q,ifile);
    fclose(ifile);
    multiply(p,q,ke);
    leng=strlen(strinputde);
    cinstr(x,strinputde);
    xgcd(p,q,a,b,t);
    lgconv(leng,n);    /* first recover "one-time pad" */

#ifdef RSA
    decr(p,1,alpha);
    premult(alpha,2,alpha);
    incr(alpha,1,alpha);
    subdiv(alpha,3,alpha);
#else
    incr(p,1,alpha);
    subdiv(alpha,4,alpha);
#endif
    decr(p,1,y);
    powmod(alpha,n,y,alpha);
#ifdef RSA
    decr(q,1,beta);
    premult(beta,2,beta);
```

```
    incr(beta,1,beta);
    subdiv(beta,3,beta);
#else
    incr(q,1,beta);
    subdiv(beta,4,beta);
#endif
    decr(q,1,y);
    powmod(beta,n,y,beta);
    copy(x,y);
    divide(x,p,p);
    divide(y,q,q);
    powmod(x,alpha,p,x);
    powmod(y,beta,q,y);
    mad(x,q,q,ke,ke,t);
    mad(t,b,b,ke,ke,t);
    mad(y,p,p,ke,ke,x);
    mad(x,a,a,ke,ke,x);
    add(x,t,x);
    divide(x,ke,ke);
    if (size(x)<0) add(x,ke,x);

for (i=0;i<leng;i++)
    { /* decipher character by character */
        ch=*(strinputde+i);
        ch^=x[1];   /* XOR with last byte of x */
        stroutputde[i]=ch;
#ifdef RSA
        power(x,3,ke,x);
#else
        mad(x,x,x,ke,ke,x);
#endif
    }
   return 0;
```

```
}
//============================
void multiply(big x,big y,big z)
{  /*  multiply two big numbers: z=x.y  */
    int i,xl,yl,j,ti;
    mr_small carry,sz;
    big w0;
#ifdef MR_NOASM
    mr_large dble;
#endif
    if (mr_mip->ERNUM) return;
    if (y[0]==0 || x[0]==0)
    {
        zero(z);
        return;
    }
    w0=mr_mip->w0;    /* local pointer */
    mr_mip->depth++;
    mr_mip->trace[mr_mip->depth]=5;
    if (mr_mip->TRACER) mr_track();
#ifdef MR_FLASH
    if (mr_notint(x) || mr_notint(y))
    {
        mr_berror(MR_ERR_INT_OP);
        mr_mip->depth--;
        return;
    }
#endif
    sz=((x[0]&mr_mip->MSBIT)^(y[0]&mr_mip->MSBIT));
    xl=(int)(x[0]&mr_mip->OBITS);
    yl=(int)(y[0]&mr_mip->OBITS);
    zero(w0);
    if (mr_mip->check && xl+yl>mr_mip->nib)
```

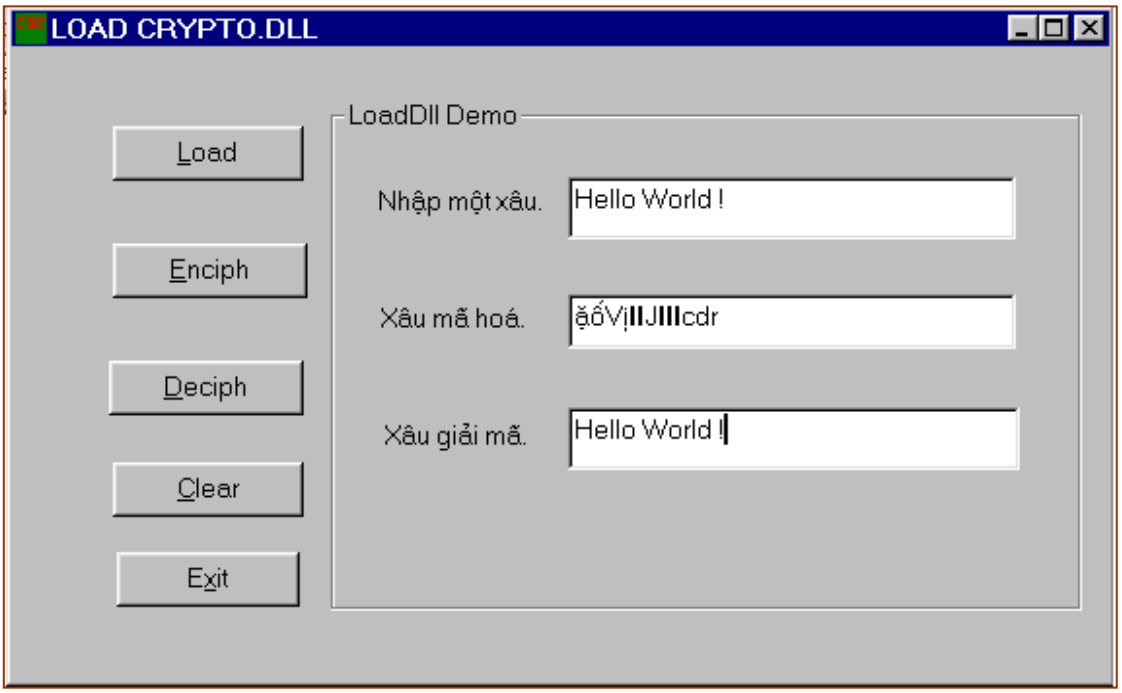
```
    {
        mr_berror(MR_ERR_OVERFLOW);
        mr_mip->depth--;
        return;
    }

//============================
void mad(big x,big y,big z,big w,big q,big r)
{
    if (mr_mip->ERNUM) return;
    mr_mip->depth++;
    mr_mip->trace[mr_mip->depth]=24;
    if (mr_mip->TRACER) mr_track();
    mr_mip->check=OFF;
    if (w==r)
    {
        mr_berror(MR_ERR_BAD_PARAMETERS);
        mr_mip->depth--;
        return;
    }
    multiply(x,y,mr_mip->w0);
    if (x!=z && y!=z)add(mr_mip->w0,z,mr_mip->w0);

    divide(mr_mip->w0,w,q);
    if (q!=r) copy(mr_mip->w0,r);
    mr_mip->check=ON;
    mr_mip->depth--;
}
//============================
int cinstr(flash x,unsigned char *string)
{  /*  input big number in base IOBASE  */
    mr_small newb,oldb,b,lx;
    int ipt;
```

```
    if (mr_mip->ERNUM) return 0;
    mr_mip->depth++;
    mr_mip->trace[mr_mip->depth]=78;
    if (mr_mip->TRACER) mr_track();
    newb=mr_mip->IOBASE;
    oldb=mr_mip->apbase;
    mr_setbase(newb); /* temporarily change base ... */
    b=mr_mip->base;
    mr_mip->check=OFF;
    ipt=instr(mr_mip->w5,string); /* ... and get number  */
    mr_mip->check=ON;
    lx=(mr_mip->w5[0]&mr_mip->OBITS);
#ifdef MR_FLASH
    if         ((int)(lx&mr_mip->MSK)>mr_mip->nib         ||
(int)((lx>>mr_mip->BTS)&mr_mip->MSK)>mr_mip->nib)
#else
    if ((int)lx>mr_mip->nib)
#endif
    { /* numerator or denominator too big */
        mr_berror(MR_ERR_OVERFLOW);
        mr_mip->depth--;
        return 0;
    }
    mr_setbase(oldb);      /* restore original base */
    cbase(mr_mip->w5,b,x);
    mr_mip->depth--;
    return ipt;
}
//============================
void incr(big x,int n,big z)
{  /* add int to big number: z=x+n */
    if (mr_mip->ERNUM) return;
    mr_mip->depth++;
```

```
    mr_mip->trace[mr_mip->depth]=7;
    if (mr_mip->TRACER) mr_track();
    convert(n,mr_mip->w0);
    select(x,PLUS,mr_mip->w0,z);
    mr_mip->depth--;
}
//============================
void decr(big x,int n,big z)
{  /* subtract int from big number: z=x-n */
    if (mr_mip->ERNUM) return;
    mr_mip->depth++;
    mr_mip->trace[mr_mip->depth]=8;
    if (mr_mip->TRACER) mr_track();
    convert(n,mr_mip->w0);
    select(x,MINUS,mr_mip->w0,z);
    mr_mip->depth--;
}
```

2.Chương trình Demo thư viện CRYPTO.DLL

Phần này xây dựng một ứng dụng đơn giản để Demo thư viện CRYPTO.DLL, chương trình xây dựng nhập vào một xâu rồi mã hoá, giải mã và trả lại kết quả ban đầu.



kết luận.

Qua quá trình làm luận văn, em đã hiểu biết thêm kiến thức về sự an toàn của thông tin trên mạng, một số thuật toán và phương pháp mã hoá. Để so sánh, đánh giá một thuật toán mã hoá cần dựa vào một số yếu tố cơ bản như độ phức tạp thuật toán, thời gian mã hoá và vấn đề phân phối khoá trong môi trường nhiều người sử dụng.

Dễ nhận thấy rằng các phương pháp mã hoá cổ điển như phương pháp đổi chỗ và thay thế là đơn giản và dễ thực hiện, tuy nhiên độ an toàn không cao do không đạt được độ phức tạp cần thiết, đồng thời khoá cũng rất dễ bị lộ do khoá của người gửi và người nhận là giống nhau. Đối với các thuật toán mã khoá công khai đã khắc phục được vấn đề phân phối khoá, khoá mã hoá có thể công khai và bất kỳ người nào có khoá công khai đều có thể mã hoá bản tin của mình, nhưng chỉ duy nhất người có khoá bí mật mới có thể giải mã được.

Phương pháp mã hoá công khai sử dụng thuật toán RSA khá chậm chạp do yêu cầu những số nguyên tố lớn để sinh ra khoá công khai và khoá bí mật nhưng mặt khác nó rất hữu ích vì cho tới nay chưa có thuật toán nào phân tích nhanh một số lớn thành các thừa số là các số nguyên tố.

Với đề tài "Xây dựng thư viện các hàm mã hoá phục vụ bảo mật thông tin trong mô hình Client/Server" em đã hoàn thành xây dựng thư viện động CRYPTO.DLL với hai hàm mã hoá và hàm giải mã sử dụng thuật toán RSA, bên cạnh đó chưa hoàn thành phần việc xây dựng một ứng dụng để Demo thư viện trên mô hình Client/Server. Tuy nhiên do quĩ thời gian hạn hẹp, trình độ còn hạn chế nên không tránh khỏi thiếu xót, rất mong được sự chỉ bảo, góp ý nhiệt tình của các thầy.

Trong tương lai nếu điều kiện thời gian và kỹ thuật không bị hạn chế em sẽ xây dựng thư viện với các hàm đầy đủ hơn như, hàm kiểm tra một số có phải nguyên tố không, hàm sinh khoá, hàm tính giai thừa . . .

Em xin chân thành cảm ơn !

Hà Nội, Ngày 06 tháng 06 năm 1999.

Người thực hiện.

Đặng Văn Hanh

Tài liệu tham khảo :

BRASSARD, **Modern Cryptology. Lecture Notes in Computer Science**, Vol. 325. Springer-Verlag 1988.

BRUCE SCHNEIER, **APPLIED CRYPTOGRAPHY, Protocol, Algorithms, and Source Code in C**, John Wiley & Sons 1994

COMBA, **Exponentiation Cryptosystems on the IBM PC. IBM**

Phạm Văn ất, **Kỹ thuật lập trình C, cơ sở và nâng cao**
Nhà xuất bản giáo dục 1997.

Xuân Nguyệt và Phùng Kim Hoàng, **học Visual C++ 5 trong 21 ngày.**
Nhà xuất bản Mũi cà mau 1998.

SQL Server 2005 – Hack dữ liệu đã mã hoá bởi mật khẩu

Trong phần 1 của loạt bài này, chúng tôi đã giới thiệu phương pháp mã hoá và giải mã bằng mật khẩu. Phần 2 này sẽ đi vào cách hack lại dữ liệu đó.

Như bạn đã biết, mã hoá bằng mật khẩu là một phương pháp mã hoá dữ liệu cơ bản chỉ sử dụng đến mật khẩu và có thể giải mã với cùng mật khẩu đó. Giờ hãy giả dụ chúng ta quên mất mật khẩu đã đặt và cần phải khôi phục lại dữ liệu như ban đầu.

Bước 1

Mã hoá dữ liệu theo phương pháp mã hoá bằng mật khẩu

```
select EncryptedData = EncryptByPassPhrase('MAK', '123456789' )
```

Kết quả

```
EncryptedData
------------------------------------------------------------------------
0x01000000F75D553409C74570F6DDBCADA53FD489DDD52D9277010050565ADF30F244F8CC
```

Bước 2

Tạo thủ tục sử dụng để khôi phục lại dữ liệu đã má hoá. Thủ tục này sẽ sử dụng hàm DecryptByPassPhrase để giải mã dữ liệu và hiển thị lên mật khẩu.

```
USE [Master]
GO

/****** Object:  StoredProcedure [dbo].[hack_encryption] Script Date: 12/18/2007 18:18:36 ******/
IF EXISTS (SELECT * FROM sys.objects WHERE object_id = OBJECT_ID(N'[dbo].[hack_encryption]')
                                AND type in (N'P', N'PC'))
DROP PROCEDURE [dbo].[hack_encryption]
GO
set nocount on
SET CONCAT_NULL_YIELDS_NULL OFF
go
```

```
USE [Master]
GO

/****** Object:  StoredProcedure [dbo].[hack_encryption] Script Date:
12/18/2007 18:18:55 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE procedure [dbo].[hack_encryption] @encryptedtext varbinary(max)
as
declare @password varchar(6)
declare @i int
declare @j int
declare @k int
declare @l int
declare @m int
declare @n int


set @i=-1
set @j=-1
set @k=-1
set @l=-1
set @m=-1
set @n=-1
set @password =''

while @i<255
begin
  while @j<255
  begin
     while @k<255
     begin
            while @l<255
            begin
              while @m<255
              begin
```

```
                while @n<=255
                begin
                set @password=isnull(char(@i),'')
                      + isnull(char(@j),'')
                      +isnull(char(@k),'')+ isnull(char(@l),'')
                      +isnull(char(@m),'') + isnull(char(@n),'')
                if convert(varchar(100),
DecryptByPassPhrase(ltrim(rtrim(@password)),
                                                    @encryptedtext)) is not null
                begin
                print 'This is the Encrypted text:' +@password
                set @i=256;set @j=256;set @k=256;set @l=256;set
@m=256;set @n=256;
                print 'The actual data is :' +convert(varchar(100),

DecryptByPassPhrase(ltrim(rtrim(@password)), @encryptedtext))
                end
                --print 'A'+ltrim(rtrim(@password))+'B'
                --print convert(varchar(100),
DecryptByPassPhrase(ltrim(rtrim(@password)),@encryptedtext))
                set @n=@n+1
                end
              set @n=0
              set @m=@m+1
              end
        set @m=0
        set @l=@l+1
        end
      set @l=0
      set @k=@k+1
      end
  set @k=0
  set @j=@j+1
  end
set @j=0
set @i=@i+1
end


GO
```

Bước 3

Giả sử rằng bạn đã quên mật khẩu dùng để mã hoá dữ liệu thành “0x01000000F75D553409C74570F6DDBCADA53FD489DDD52D9277010050565ADF30F244F8CC”. Chúng ta có thể truy tìm lại được mật khẩu và dữ liệu đã bị mã hoá bằng thủ tục sau

```
use master
go
select getdate() as StartingTime
go
declare @myencryptedtext varbinary(max)
set
@myencryptedtext=0x01000000F75D553409C74570F6DDBCADA53FD489
DDD52D9277010050565ADF30F244F8CC
print @myencryptedtext
exec hack_encryption @encryptedtext=@myencryptedtext
go
select getdate() as EndingTime
go
```

Kết quả

```
StartingTime
-----------------------
2007-12-18 18:24:10.843

0x01000000F75D553409C74570F6DDBCADA53FD489DDD52D927701005
0565ADF30F244F8CC
This is the Encrypted text: MAK
The actual data is :123456789

EndingTime
-----------------------
2007-12-18 18:26:36.080
```

Hình 1

Như bạn thấy trong kết quả (hình 1), nó chỉ cần đến 2 phút để tìm lại được dữ liệu và mật khẩu. Về cơ bản, thủ tục này lặp lại tất cả khả năng hợp lý có thể xảy ra của các ký tự ASCII có độ dài trên 6 ký tự để tìm ra mật khẩu và sử dụng nó để giải mã dữ liệu.

Tạo ra một thủ tục sẽ không giúp gì nhiều khi dữ liệu đã được mã hoá nằm trong một bảng. Vì vậy chúng ta phải thay đổi thủ thục này thành một hàm vô hướng như hướng dẫn dưới đây

Bước 1

Tạo thủ tục như sau

```
USE [master]
GO

/****** Object:  UserDefinedFunction [dbo].[hack_encryption_password]
Script Date: 12/18/2007 18:36:29 ******/
IF  EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[dbo].[hack_encryption_password]')
                                AND type in (N'FN', N'IF', N'TF', N'FS', N'FT'))
DROP FUNCTION [dbo].[hack_encryption_password]
GO
use [Master]
```

```
go

CREATE function [dbo].[hack_encryption_password] (@encryptedtext
varbinary(max))
returns varchar(6)
with execute as caller
as
begin
declare @password varchar(6)
declare @i int
declare @j int
declare @k int
declare @l int
declare @m int
declare @n int


set @i=-1
set @j=-1
set @k=-1
set @l=-1
set @m=-1
set @n=-1
set @password =''

while @i<255
begin
   while @j<255
   begin
      while @k<255
      begin
         while @l<255
         begin
            while @m<255
            begin
               while @n<=255
               begin
               set @password=isnull(char(@i),'') + isnull(char(@j),'')
                        +isnull(char(@k),'')+ isnull(char(@l),'')
                        +isnull(char(@m),'') + isnull(char(@n),'')
```

```
                if convert(varchar(100),
DecryptByPassPhrase(ltrim(rtrim(@password)),
                                                @encryptedtext)) is not null
                begin
                --print 'This is the Encrypted text:' +@password
                set @i=256;set @j=256;set @k=256;set @l=256;set @m=256;set
@n=256;
                --print 'The actual data is :' +convert(varchar(100),

DecryptByPassPhrase(ltrim(rtrim(@password)),@encryptedtext))
                end
                --print 'A'+ltrim(rtrim(@password))+'B'
                --print
convert(varchar(100),DecryptByPassPhrase(ltrim(rtrim(@password)),@encryp
tedtext))
                set @n=@n+1
                end
             set @n=0
             set @m=@m+1
             end
          set @m=0
          set @l=@l+1
          end
       set @l=0
       set @k=@k+1
       end
    set @k=0
    set @j=@j+1
    end
set @j=0
set @i=@i+1
end

return @password
END
```

Bước 2

Tạo một bảng với dữ liệu được mã hoá

```
USE [tempdb]
GO
/****** Object: Table [dbo].[MyTable] Script Date: 12/18/2007 18:44:40
******/
IF EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[dbo].[MyTable]') AND type in (N'U'))
DROP TABLE [dbo].[MyTable]
GO
create table MyTable(id int, encrypteddata varbinary(max))
go
insert into MyTable select 1, EncryptByPassPhrase('Do', '1112228333')
insert into MyTable select 2, EncryptByPassPhrase('Re', '1212223833')
insert into MyTable select 3, EncryptByPassPhrase('Me', '1132223393')
insert into MyTable select 4, EncryptByPassPhrase('Fa', '1114223383')
insert into MyTable select 5, EncryptByPassPhrase('So', '1112523333')
insert into MyTable select 6, EncryptByPassPhrase('La', '1112263373')
insert into MyTable select 7, EncryptByPassPhrase('Si', '1112227338')
go
```

Bước 3

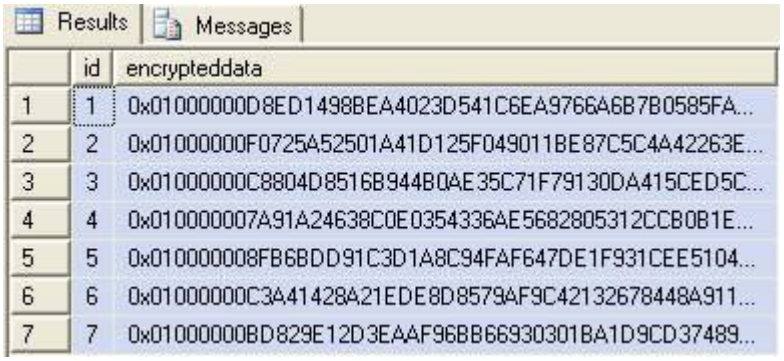
Truy vấn dữ liệu sử dụng câu lệnh SQL sau

```
Select * from MyTable
```

Bạn sẽ thấy dữ liệu hiển thị như sau (hình 2)

```
1
0x01000000D8ED1498BEA4023D541C6EA9766A6B7B0585FAE91B942C8
8C23677550C6FD7FA
2
0x01000000F0725A52501A41D125F049011BE87C5C4A42263E7538B837B
8278ADEE5FC2678
3
0x01000000C8804D8516B944B0AE35C71F79130DA415CED5CCF58E5226
92AC749115EEF0D9
4
0x010000007A91A24638C0E0354336AE5682805312CCB0B1E6BBACB6D
9E65DC5D9DA73906E
5
```

0x010000008FB6BDD91C3D1A8C94FAF647DE1F931CEE5104045BD03D
E4E809565E74604DF3
6
0x01000000C3A41428A21EDE8D8579AF9C42132678448A9113A31A8692
76A7631A58A32BE3
7
0x01000000BD829E12D3EAAF96BB66930301BA1D9CD3748946F3543019
22A03AE49047FE00


Results | Messages

	id	encrypteddata
1	1	0x01000000D8ED1498BEA4023D541C6EA9766A6B7B0585FA...
2	2	0x01000000F0725A52501A41D125F049011BE87C5C4A42263E...
3	3	0x01000000C8804D8516B944B0AE35C71F79130DA415CED5C...
4	4	0x010000007A91A24638C0E0354336AE5682805312CCB0B1E...
5	5	0x010000008FB6BDD91C3D1A8C94FAF647DE1F931CEE5104...
6	6	0x01000000C3A41428A21EDE8D8579AF9C42132678448A911...
7	7	0x01000000BD829E12D3EAAF96BB66930301BA1D9CD37489...

Hình 2

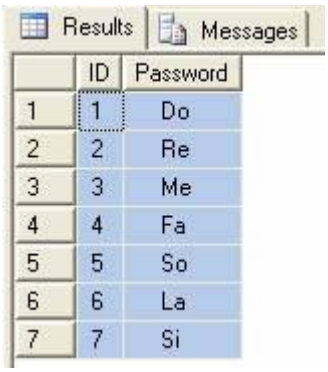
Bước 4

Sử dụng hàm hack_encryption_password để khôi phục tất cả các mật khẩu từ dưc liệu đã được mã hoá trong bảng MyTable. Thực thi câu lệnh SQL sau

select ID ,master.[dbo].[hack_encryption_password] (encrypteddata) as Password from MyTable

Bạn sẽ thấy kết quả như sau (Hình 3)

1 Do
2 Re
3 Me
4 Fa
5 So
6 La
7 Si



Results | Messages

	ID	Password
1	1	Do
2	2	Re
3	3	Me
4	4	Fa
5	5	So
6	6	La
7	7	Si

Hình 3

Hàm trên có thê được chỉnh sửa để trả về cả dữ liệu đã được mã hoá, thực hiện như sau

Bước 1

Tạo hàm sau

```
USE [master]
GO

/****** Object:  UserDefinedFunction [dbo].[hack_encryption_password]
Script Date: 12/18/2007 18:36:29 ******/
IF  EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[dbo].[hack_encryption_data]')
                                AND type in (N'FN', N'IF', N'TF', N'FS', N'FT'))
DROP FUNCTION [dbo].[hack_encryption_data]
GO
use [Master]
go

CREATE function [dbo].[hack_encryption_data] (@encryptedtext
varbinary(max))
returns varchar(8000)
with execute as caller
as
begin
declare @data varchar(8000)
declare @password varchar(6)
declare @i int
declare @j int
```

```
declare @k int
declare @l int
declare @m int
declare @n int


set @i=-1
set @j=-1
set @k=-1
set @l=-1
set @m=-1
set @n=-1
set @password =''

while @i<255
begin
   while @j<255
   begin
      while @k<255
      begin
         while @l<255
         begin
            while @m<255
            begin
               while @n<=255
               begin
               set @password=isnull(char(@i),'') +
isnull(char(@j),'')+isnull(char(@k),'')
                        + isnull(char(@l),'')+isnull(char(@m),'') +
isnull(char(@n),'')
               if
convert(varchar(100),DecryptByPassPhrase(ltrim(rtrim(@password)),
                                                @encryptedtext)) is not null
               begin
               --print 'This is the Encrypted text:' +@password
               set @i=256;set @j=256;set @k=256;set @l=256;set @m=256;set
@n=256;
               set @data = convert(varchar(100),
DecryptByPassPhrase(ltrim(rtrim(@password)),@encryptedtext))
               end
```

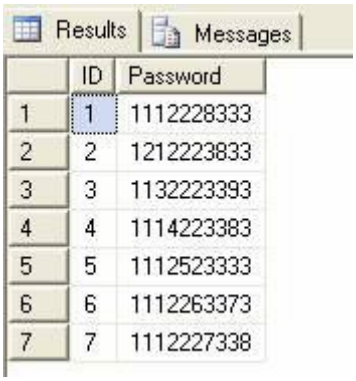
```
                --print 'A'+ltrim(rtrim(@password))+'B'
                --print convert(varchar(100),
DecryptByPassPhrase(ltrim(rtrim(@password)),@encryptedtext))
                set @n=@n+1
                end
            set @n=0
            set @m=@m+1
            end
        set @m=0
        set @l=@l+1
        end
    set @l=0
    set @k=@k+1
    end
  set @k=0
  set @j=@j+1
  end
set @j=0
set @i=@i+1
end

return @data
END
```

Bước 2

Giải mã dữ liệu sử dụng hàm đã tạo

```
select ID ,master.[dbo].[hack_encryption_data] (encrypteddata) as Password
from MyTable
```

Kết quả như hình 4



Hình 4

Lưu ý:

- Thủ thục và các hàm chỉ có thể hack đối với mật khẩu dài 6 ký tự.
- Thủ tục và hàm này có thể chiếm rất nhiều CPU để lấy lại dữ liệu và tìm lại mật khẩu

Theo Databasejournal

THUẬT TOÁN MÃ HÓA VÀ ỨNG DỤNG

Lời giới thiệu

Mật mã (Cryptography) là ngành khoa học là ngành nghiên cứu các kỹ thuật toán học nhằm cung cấp các dịch vụ bảo vệ thông tin [44]. Đây là ngành khoa học quan trọng, có nhiều ứng dụng trong đời sống – xã hội.

Khoa học mật mã đã ra đời từ hàng nghìn năm. Tuy nhiên, trong suốt nhiều thế kỷ, các kết quả của lĩnh vực này hầu như không được ứng dụng trong các lĩnh vực dân sự thông thường của đời sống – xã hội mà chủ yếu được sử dụng trong lĩnh vực quân sự, chính trị, ngoại giao... Ngày nay, các ứng dụng mã hóa và bảo mật thông tin đang được sử dụng ngày càng phổ biến trong các lĩnh vực khác nhau trên thế giới, từ các lĩnh vực an ninh, quân sự, quốc phòng…, cho đến các lĩnh vực dân sự như thương mại điện tử, ngân hàng…

Với sự phát triển ngày càng nhanh chóng của Internet và các ứng dụng giao dịch điện tử trên mạng, nhu cầu bảo vệ thông tin trong các hệ thống và ứng dụng điện tử ngày càng được quan tâm và có ý nghĩa hết sức quan trọng. Các kết quả của khoa học mật mã ngày càng được triển khai trong nhiều lĩnh vực khác nhau của đời sống – xã hội, trong đó phải kể đến rất nhiều những ứng dụng đa dạng trong lĩnh vực dân sự, thương mại...Các ứng dụng mã hóa thông tin cá nhân, trao đổi thông tin kinh doanh, thực hiện các giao dịch điện tử qua mạng... đã trở nên gần gũi và quen thuộc với mọi người.

Cùng với sự phát triển của khoa học máy tính và Internet, các nghiên cứu và ứng dụng của mật mã học ngày càng trở nên đa dạng hơn, mở ra nhiều hướng nghiên cứu chuyên sâu vào từng lĩnh vực ứng dụng đặc thù với những đặc trưng riêng. Ứng dụng của khoa học mật mã không chỉ đơn thuần là mã hóa và giải mã thông tin mà còn bao gồm nhiều vấn đề khác nhau cần được nghiên cứu và giải quyết, ví dụ như chứng thực nguồn gốc

nội dung thông tin (kỹ thuật chữ ký điện tử), chứng nhận tính xác thực về người sở hữu mã khóa (chứng nhận khóa công cộng), các quy trình giúp trao đổi thông tin và thực hiện giao dịch điện tử an toàn trên mạng...

Các ứng dụng của mật mã học và khoa học bảo vệ thông tin rất đa dạng và phong phú; tùy vào tính đặc thù của mỗi hệ thống bảo vệ thông tin mà ứng dụng sẽ có các tính năng với đặc trưng riêng. Trong đó, chúng ta có thể kể ra một số tính năng chính của hệ thống bảo vệ thông tin:

- Tính bảo mật thông tin: hệ thống đảm bảo thông tin được giữ bí mật. Thông tin có thể bị phát hiện, ví dụ như trong quá trình truyền nhận, nhưng người tấn công không thể hiểu được nội dung thông tin bị đánh cắp này.
- Tính toàn vẹn thông tin: hệ thống bảo đảm tính toàn vẹn thông tin trong liên lạc hoặc giúp phát hiện rằng thông tin đã bị sửa đổi.
- Xác thực các đối tác trong liên lạc và xác thực nội dung thông tin trong liên lạc.
- Chống lại sự thoái thác trách nhiệm: hệ thống đảm bảo một đối tác bất kỳ trong hệ thống không thể từ chối trách nhiệm về hành động mà mình đã thực hiện

Những kết quả nghiên cứu về mật mã cũng đã được đưa vào trong các hệ thống phức tạp hơn, kết hợp với những kỹ thuật khác để đáp ứng yêu cầu đa dạng của các hệ thống ứng dụng khác nhau trong thực tế, ví dụ như hệ thống bỏ phiếu bầu cử qua mạng, hệ thống đào tạo từ xa, hệ thống quản lý an ninh của các đơn vị với hướng tiếp cận sinh trắc học, hệ thống cung cấp dịch vụ đa phương tiện trên mạng với yêu cầu cung cấp dịch vụ và bảo vệ bản quyền sở hữu trí tuệ đối với thông tin số...

Khi biên soạn tập sách này, nhóm tác giả chúng tôi mong muốn giới thiệu với quý độc giả những kiến thức tổng quan về mã hóa và ứng dụng, đồng thời trình bày và phân tích một số phương pháp mã hóa và quy trình bảo vệ thông tin an toàn và hiệu quả trong thực tế.

Bên cạnh các phương pháp mã hóa kinh điển nổi tiếng đã được sử dụng rộng rãi trong nhiều thập niên qua như DES, RSA, MD5…, chúng tôi cũng giới thiệu với bạn đọc các phương pháp mới, có độ an toàn cao như chuẩn mã hóa AES, phương pháp ECC, chuẩn hàm băm mật mã SHA224/256/384/512… Các mô hình và quy trình chứng nhận khóa công cộng cũng được trình bày trong tập sách này.

Nội dung của sách gồm 10 chương. Sau phần giới thiệu tổng quan về mật mã học và khái niệm về hệ thống mã hóa ở chương 1, từ chương 2 đến chương 5, chúng ta sẽ đi sâu vào tìm hiểu hệ thống mã hóa quy ước, từ các khái niệm cơ bản, các phương pháp đơn giản, đến các phương pháp mới như Rijndael và các thuật toán ứng cử viên AES. Nội dung của chương 6 giới thiệu hệ thống mã hóa khóa công cộng và phương pháp RSA. Chương 7 sẽ trình bày về khái niệm chữ ký điện tử cùng với một số phương pháp phổ biến như RSA, DSS, ElGamal. Các kết quả nghiên cứu ứng dụng lý thuyết đường cong elliptic trên trường hữu hạn vào mật mã học được trình bày trong chương 8. Chương 9 giới thiệu về các hàm băm mật mã hiện đang được sử dụng phổ biến như MD5, SHS cùng với các phương pháp mới được công bố trong thời gian gần đây như SHA-256/384/512. Trong chương 10, chúng ta sẽ tìm hiểu về hệ thống chứng nhận khóa công cộng, từ các mô hình đến quy trình trong thực tế của hệ thống chứng nhận khóa công cộng, cùng với một ví dụ về việc kết hợp hệ thống mã hóa quy ước, hệ thống mã hóa khóa công cộng và chứng nhận khóa công cộng để xây dựng hệ thống thư điện tử an toàn.

Với bố cục và nội dung nêu trên, chúng tôi hi vọng các kiến thức trình bày trong tập sách này sẽ là nguồn tham khảo hữu ích cho quý độc giả quan tâm đến lĩnh vực mã hóa và ứng dụng.

Mặc dù đã cố gắng hoàn thành sách với tất cả sự nỗ lực nhưng chắc chắn chúng tôi vẫn còn những thiếu sót nhất định. Kính mong sự cảm thông và sự góp ý của quý độc giả.

NHÓM TÁC GIẢ: TS. Dương Anh Đức - ThS. Trần Minh Triết
cùng với sự đóng góp của các sinh viên Khoa Công nghệ Thông tin, Trường Đại học Khoa học Tự nhiên, Đại học Quốc gia thành phố Hồ Chí Minh.

Văn Đức Phương Hồng
Phan Thị Minh Đức
Nguyễn Minh Huy
Lương Vĩ Minh
Nguyễn Ngọc Tùng

Thành phố Hồ Chí Minh, tháng 01 năm 2005

Mục lục

Danh sách hình

Danh sách bảng

Chương 1
Tổng quan

✍ *Nội dung của chương 1 giới thiệu tổng quan các khái niệm cơ bản về mật mã học và hệ thống mã hóa, đồng thời giới thiệu sơ lược về hệ thống mã hóa quy ước và hệ thống mã hóa khóa công cộng.*

1.1 Mật mã học

Mật mã học là ngành khoa học ứng dụng toán học vào việc biến đổi thông tin thành một dạng khác với mục đích che dấu nội dung, ý nghĩa thông tin cần mã hóa. Đây là một ngành quan trọng và có nhiều ứng dụng trong đời sống xã hội. Ngày nay, các ứng dụng mã hóa và bảo mật thông tin đang được sử dụng ngày càng phổ biến hơn trong các lĩnh vực khác nhau trên thế giới, từ các lĩnh vực an ninh, quân sự, quốc phòng…, cho đến các lĩnh vực dân sự như thương mại điện tử, ngân hàng…

Cùng với sự phát triển của khoa học máy tính và Internet, các nghiên cứu và ứng dụng của khoa học mật mã ngày càng trở nên đa dạng hơn, mở ra nhiều hướng nghiên cứu chuyên sâu vào từng lĩnh vực ứng dụng đặc thù với những đặc trưng

riêng. Ứng dụng của khoa học mật mã không chỉ đơn thuần là mã hóa và giải mã thông tin mà còn bao gồm nhiều vấn đề khác nhau cần được nghiên cứu và giải quyết: chứng thực nguồn gốc nội dung thông tin (kỹ thuật chữ ký điện tử), chứng nhận tính xác thực về người sở hữu mã khóa (chứng nhận khóa công cộng), các quy trình giúp trao đổi thông tin và thực hiện giao dịch điện tử an toàn trên mạng... Những kết quả nghiên cứu về mật mã cũng đã được đưa vào trong các hệ thống phức tạp hơn, kết hợp với những kỹ thuật khác để đáp ứng yêu cầu đa dạng của các hệ thống ứng dụng khác nhau trong thực tế, ví dụ như hệ thống bỏ phiếu bầu cử qua mạng, hệ thống đào tạo từ xa, hệ thống quản lý an ninh của các đơn vị với hướng tiếp cận sinh trắc học, hệ thống cung cấp dịch vụ multimedia trên mạng với yêu cầu cung cấp dịch vụ và bảo vệ bản quyền sở hữu trí tuệ đối với thông tin số...

1.2 Hệ thống mã hóa (cryptosystem)

Định nghĩa 1.1: *Hệ thống mã hóa (cryptosystem) là một bộ năm (P, C, K, E, D) thỏa mãn các điều kiện sau:*

1. *Tập nguồn P là tập hữu hạn tất cả các mẩu tin nguồn cần mã hóa có thể có*
2. *Tập đích C là tập hữu hạn tất cả các mẩu tin có thể có sau khi mã hóa*
3. *Tập khóa K là tập hữu hạn các khóa có thể được sử dụng*
4. *E và D lần lượt là tập luật mã hóa và giải mã. Với mỗi khóa* $k \in K$ *, tồn tại luật mã hóa* $e_k \in E$ *và luật giải mã* $d_k \in D$ *tương ứng. Luật mã hóa* $e_k : P \to C$ *và luật giải mã* $e_k : C \to P$ *là hai ánh xạ thỏa mãn* $d_k(e_k(x)) = x, \forall x \in P$

Tính chất 4 là tính chất chính và quan trọng của một hệ thống mã hóa. Tính chất này bảo đảm một mẩu tin $x \in P$ được mã hóa bằng luật mã hóa $e_k \in E$ có thể được giải mã chính xác bằng luật $d_k \in D$.

Định nghĩa 1.2: $\mathbb{Z}_m$ *được định nghĩa là tập hợp* $\{0,1,...,m-1\}$, *được trang bị phép cộng (ký hiệu +) và phép nhân (ký hiệu là ×). Phép cộng và phép nhân trong* $\mathbb{Z}_m$ *được thực hiện tương tự như trong* $\mathbb{Z}$, *ngoại trừ kết quả tính theo modulo m*.

❒ Ví dụ: Giả sử ta cần tính giá trị 11×13 trong $\mathbb{Z}_{16}$. Trong $\mathbb{Z}$, ta có kết quả của phép nhân $11 \times 13 = 143$. Do $143 \equiv 15 \pmod{16}$ nên $11 \times 13 = 15$ trong $\mathbb{Z}_{16}$.

Một số tính chất của $\mathbb{Z}_m$

1. Phép cộng đóng trong $\mathbb{Z}_m$, $\forall a,b \in \mathbb{Z}_m$, $a+b \in \mathbb{Z}_m$
2. Tính giao hoán của phép cộng trong $\mathbb{Z}_m$, $\forall a,b \in \mathbb{Z}_m$, $a+b=b+a$
3. Tính kết hợp của phép cộng trong $\mathbb{Z}_m$, $\forall a,b,c \in \mathbb{Z}_m$, $(a+b)+c=a+(b+c)$
4. $\mathbb{Z}_m$ có phần tử trung hòa là 0, $\forall a,b \in \mathbb{Z}_m$, $a+0=0+a=a$
5. Mọi phần tử *a* trong $\mathbb{Z}_m$ đều có phần tử đối là $m-a$
6. Phép nhân đóng trong $\mathbb{Z}_m$, $\forall a,b \in \mathbb{Z}_m$, $a \times b \in \mathbb{Z}_m$
7. Tính giao hoán của phép nhân trong $\mathbb{Z}_m$, $\forall a,b \in \mathbb{Z}_m$, $a \times b = b \times a$
8. Tính kết hợp của phép nhân trong $\mathbb{Z}_m$, $\forall a,b,c \in \mathbb{Z}_m$, $(a \times b) \times c = a \times (b \times c)$

9. $\mathbb{Z}_m$ có phần tử đơn vị là 1, $\forall a,b \in \mathbb{Z}_m$, $a \times 1 = 1 \times a = a$

10. Tính phân phối của phép nhân đối với phép cộng, $\forall a,b,c \in \mathbb{Z}_m$, $(a+b) \times c = a \times c + b \times c$

$\mathbb{Z}_m$ có các tính chất 1, 3 – 5 nên tạo thành một nhóm. Do $\mathbb{Z}_m$ có tính chất 2 nên tạo thành nhóm Abel. $\mathbb{Z}_m$ có các tính chất (1) – (10) nên tạo thành một vành.

1.3 Hệ thống mã hóa quy ước (mã hóa đối xứng)

Trong hệ thống mã hóa quy ước, quá trình mã hóa và giải mã một thông điệp sử dụng cùng một mã khóa gọi là *khóa bí mật* (secret key) hay *khóa đối xứng* (symmetric key). Do đó, vấn đề bảo mật thông tin đã mã hóa hoàn toàn phụ thuộc vào việc giữ bí mật nội dung của mã khóa đã được sử dụng.

Với tốc độ và khả năng xử lý ngày càng được nâng cao của các bộ vi xử lý hiện nay, phương pháp mã hóa chuẩn (Data Encryption Standard – DES) đã trở nên không an toàn trong bảo mật thông tin. Do đó, Viện Tiêu chuẩn và Công nghệ Quốc gia Hoa Kỳ (*National Institute of Standards and Technology* – NIST) đã quyết định chọn một chuẩn mã hóa mới với độ an toàn cao nhằm phục vụ nhu cầu bảo mật thông tin liên lạc của chính phủ Hoa Kỳ cũng như trong các ứng dụng dân sự. Thuật toán Rijndael do Vincent Rijmen và Joan Daeman đã được chính thức chọn trở thành chuẩn mã hóa nâng cao (Advanced Encryption Standard – AES) từ 02 tháng 10 năm 2000.

1.4 Hệ thống mã hóa khóa công cộng (mã hóa bất đối xứng)

Nếu như vấn đề khó khăn đặt ra đối với các phương pháp mã hóa quy ước chính là bài toán trao đổi mã khóa thì ngược lại, các phương pháp mã hóa khóa công cộng giúp cho việc trao đổi mã khóa trở nên dễ dàng hơn. Nội dung của *khóa công cộng* (public key) không cần phải giữ bí mật như đối với khóa bí mật trong các phương pháp mã hóa quy ước. Sử dụng khóa công cộng, chúng ta có thể thiết lập một quy trình an toàn để truy đổi khóa bí mật được sử dụng trong hệ thống mã hóa quy ước.

Trong những năm gần đây, các phương pháp mã hóa khóa công cộng, đặc biệt là phương pháp RSA [45], được sử dụng ngày càng nhiều trong các ứng dụng mã hóa trên thế giới và có thể xem như đây là phương pháp chuẩn được sử dụng phổ biến nhất trên Internet, ứng dụng trong việc bảo mật thông tin liên lạc cũng như trong lĩnh vực thương mại điện tử.

1.5 Kết hợp mã hóa quy ước và mã hóa khóa công cộng

Các phương pháp mã hóa quy ước có ưu điểm xử lý rất nhanh và khả năng bảo mật cao so với các phương pháp mã hóa khóa công cộng nhưng lại gặp phải vấn đề khó khăn trong việc trao đổi mã khóa. Ngược lại, các phương pháp mã hóa khóa công cộng tuy xử lý thông tin chậm hơn nhưng lại cho phép người sử dụng trao đổi mã khóa dễ dàng hơn. Do đó, trong các ứng dụng thực tế, chúng ta cần phối hợp được ưu điểm của mỗi phương pháp mã hóa để xây dựng hệ thống mã hóa và bảo mật thông tin hiệu quả và an toàn.

Chương 2
Một số phương pháp mã hóa quy ước

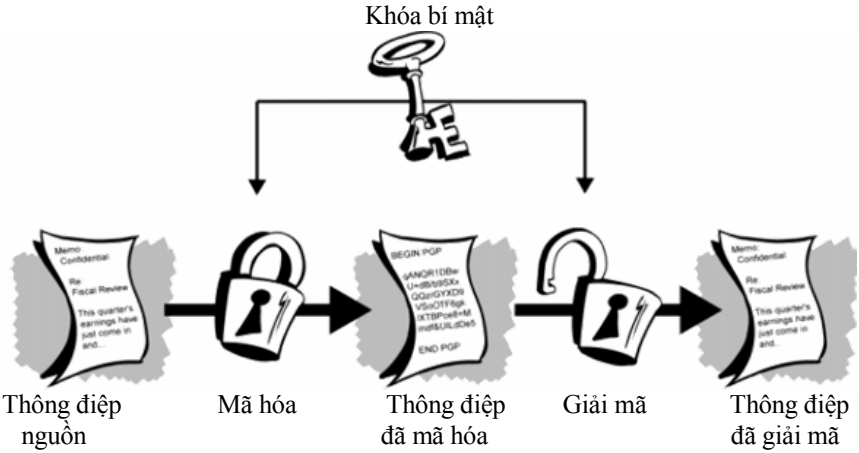
✍ *Trong chương 1, chúng ta đã tìm hiểu tổng quan về mật mã học và hệ thống mã hóa. Nội dung của chương 2 sẽ giới thiệu chi tiết hơn về hệ thống mã hóa quy ước (hay còn gọi là hệ thống mã hóa đối xứng). Một số phương pháp mã hóa quy ước kinh điển như phương pháp dịch chuyển, phương pháp thay thế… cùng với các phương pháp mã hóa theo khối được sử dụng phổ biến trong những thập niên gần đây như DES, Tripple DES, AES cũng được giới thiệu trong chương này.*

2.1 Hệ thống mã hóa quy ước

Hệ thống mã hóa quy ước là hệ thống mã hóa trong đó quy trình mã hóa và giải mã đều sử dụng chung một khoá - *khóa bí mật*. Việc bảo mật thông tin phụ thuộc vào việc bảo mật khóa.

Trong hệ thống mã hóa quy ước, thông điệp nguồn được mã hóa với mã khóa *k* được thống nhất trước giữa người gửi A và người nhận B. Người A sẽ sử dụng

mã khóa *k* để mã hóa thông điệp *x* thành thông điệp *y* và gửi *y* cho người B; người B sẽ sử dụng mã khóa *k* để giải mã thông điệp *y* này. Vấn đề an toàn bảo mật thông tin được mã hóa phụ thuộc vào việc giữ bí mật nội dung mã khóa *k*. Nếu người C biết được mã khóa *k* thì C có thể "mở khóa" thông điệp đã được mã hóa mà người A gửi cho người B.



Hình 2.1. *Mô hình hệ thống mã hóa quy ước*

2.2 Phương pháp mã hóa dịch chuyển

Phương pháp mã hóa dịch chuyển là một trong những phương pháp lâu đời nhất được sử dụng để mã hóa. Thông điệp được mã hóa bằng cách dịch chuyển xoay vòng từng ký tự đi *k* vị trí trong bảng chữ cái.

Trong trường hợp đặc biệt $k = 3$, phương pháp mã hóa bằng dịch chuyển được gọi là phương pháp *mã hóa Caesar*.

Thuật toán 2.1. *Phương pháp mã hóa dịch chuyển*

Cho $P = C = K = \mathbb{Z}_n$

Với mỗi khóa $k \in K$, định nghĩa:

$e_k(x) = (x + k) \bmod n$ và $d_k(y) = (y - k) \bmod n$ với $x, y \in \mathbb{Z}_n$

$E = \{e_k, k \in K\}$ và $D = \{d_k, k \in K\}$

Mã hóa dịch chuyển là một phương pháp mã hóa đơn giản, thao tác xử lý mã hóa và giải mã được thực hiện nhanh chóng. Tuy nhiên, trên thực tế, phương pháp này có thể dễ dàng bị phá vỡ bằng cách thử mọi khả năng khóa $k \in K$. Điều này hoàn toàn có thể thực hiện được do không gian khóa *K* chỉ có *n* phần tử để chọn lựa.

❒ Ví dụ: Để mã hóa một thông điệp được biểu diễn bằng các chữ cái từ A đến Z (26 chữ cái), ta sử dụng $P = C = K = \mathbb{Z}_{26}$. Khi đó, thông điệp được mã hóa sẽ không an toàn và có thể dễ dàng bị giải mã bằng cách thử lần lượt 26 giá trị khóa $k \in K$. Tính trung bình, thông điệp đã được mã hóa có thể bị giải mã sau khoảng $n/2$ lần thử khóa $k \in K$.

2.3 Phương pháp mã hóa thay thế

Phương pháp mã hóa thay thế (Substitution Cipher) là một trong những phương pháp mã hóa nổi tiếng và đã được sử dụng từ hàng trăm năm nay. Phương pháp này thực hiện việc mã hóa thông điệp bằng cách hoán vị các phần tử trong bảng chữ cái hay tổng quát hơn là hoán vị các phần tử trong tập nguồn *P*.

Thuật toán 2.2. *Phương pháp mã hóa bằng thay thế*

Cho $P = C = Z_n$

K là tập hợp tất cả các hoán vị của n phần tử $0,1,...,n-1$. Như vậy, mỗi khóa $\pi \in K$ là một hoán vị của n phần tử $0,1,...,n-1$.

Với mỗi khóa $\pi \in K$, định nghĩa:

$e_\pi(x) = \pi(x)$ và $d_\pi(y) = \pi^{-1}(y)$ với $x, y \in \mathbb{Z}_n$

$E = \{e_\pi, \pi \in K\}$ và $D = \{D_\pi, \pi \in K\}$

Đây là một phương pháp đơn giản, thao tác mã hóa và giải mã được thực hiện nhanh chóng. Phương pháp này khắc phục điểm hạn chế của phương pháp mã hóa bằng dịch chuyển là có không gian khóa K nhỏ nên dễ dàng bị giải mã bằng cách thử nghiệm lần lượt n giá trị khóa $k \in K$. Trong phương pháp mã hóa thay thế có không gian khóa K rất lớn với $n!$ phần tử nên không thể bị giải mã bằng cách "vét cạn" mọi trường hợp khóa k. Tuy nhiên, trên thực tế thông điệp được mã hóa bằng phương pháp này vẫn có thể bị giải mã nếu như có thể thiết lập được bảng tần số xuất hiện của các ký tự trong thông điệp hay nắm được một số từ, ngữ trong thông điệp nguồn ban đầu!

2.4 Phương pháp Affine

Nếu như phương pháp mã hóa bằng dịch chuyển là một trường hợp đặc biệt của phương pháp mã hóa bằng thay thế, trong đó chỉ sử dụng n giá trị khóa k trong số $n!$ phần tử, thì phương pháp Affine lại là một trường hợp đặc biệt khác của mã hóa bằng thay thế.

Thuật toán 2.3. *Phương pháp Affine*

Cho $P = C = Z_n$

$K = \{(a,b) \in \mathbb{Z}_n \times \mathbb{Z}_n : \gcd(a,n) = 1\}$

Với mỗi khóa $k = (a,b) \in K$, định nghĩa:

$e_k(x) = (ax+b) \bmod n$ và $d_k(x) = (a^{-1}(y-b)) \bmod n$ với $x, y \in \mathbb{Z}_n$

$E = \{e_k, k \in K\}$ và $D = \{D_k, k \in K\}$

Để có thể giải mã chính xác thông tin đã được mã hóa bằng hàm $e_k \in E$ thì e_k phải là một song ánh. Như vậy, với mỗi giá trị $y \in \mathbb{Z}_n$, phương trình $ax + b \equiv y (\bmod n)$ phải có nghiệm duy nhất $x \in \mathbb{Z}_n$.

Phương trình $ax + b \equiv y (\bmod n)$ tương đương với $ax \equiv (y-b) (\bmod n)$. Vậy, ta chỉ cần khảo sát phương trình $ax \equiv (y-b) (\bmod n)$.

Định lý 2.1: *Phương trình* $ax + b \equiv y (\bmod n)$ *có nghiệm duy nhất* $x \in \mathbb{Z}_n$ *với mỗi giá trị* $b \in \mathbb{Z}_n$ *khi và chỉ khi a và n nguyên tố cùng nhau*.

Vậy, điều kiện *a* và *n* nguyên tố cùng nhau bảo đảm thông tin được mã hóa bằng hàm e_k có thể được giải mã và giải mã một cách chính xác.

Gọi $\phi(n)$ là số lượng phần tử thuộc $\mathbb{Z}_n$ và nguyên tố cùng nhau với *n*.

Định lý 2.2: *Nếu* $n = \prod_{i=1}^{m} p_i^{e_i}$ *với* p_i *là các số nguyên tố khác nhau và* $e_i \in \mathbb{Z}^+$, $1 \le i \le m$ *thì* $\phi(n) = \prod_{i=1}^{m} \left(p_i^{e_i} - p_i^{e_i-1}\right)$.

Trong phương pháp mã hóa Affine, ta có n khả năng chọn giá trị b, $\phi(n)$ khả năng chọn giá trị a. Vậy, không gian khóa K có tất cả $n\phi(n)$ phần tử.

Vấn đề đặt ra cho phương pháp mã hóa Affine là để có thể giải mã được thông tin đã được mã hóa cần phải tính giá trị phần tử nghịch đảo $a^{-1} \in \mathbb{Z}_n$. Thuật toán Euclide mở rộng có thể giải quyết trọn vẹn vấn đề này [45].

Trước tiên, cần khảo sát thuật toán Euclide (ở dạng cơ bản) sử dụng trong việc tìm ước số chung lớn nhất của hai số nguyên dương r_0 và r_1 với $r_0 > r_1$. Thuật toán Euclide bao gồm một dãy các phép chia:

$$\begin{aligned}
&r_0 = q_1 r_1 + r_2, \ 0 < r_2 < r_1 \\
&r_1 = q_2 r_2 + r_3, \ 0 < r_3 < r_2 \\
&\dots \\
&r_{m-2} = q_{m-1} r_{m-1} + r_m, \ 0 < r_m < r_{m-1} \\
&r_{m-1} = q_m r_m
\end{aligned} \tag{2.1}$$

Dễ dàng nhận thấy rằng: $\gcd(r_0, r_1) = \gcd(r_1, r_2) = \dots = \gcd(r_{m-1}, r_m) = r_m$. Như vậy, ước số chung lớn nhất của r_0 và r_1 là r_m.

Xây dựng dãy số $t_0, t_1, ..., t_m$ theo công thức truy hồi sau:

$$t_0 = 0$$

$$t_1 = 1$$

$$t_j = (t_{j-2} - q_{j-1}t_{j-1}) \bmod r_0 \text{ với } j \geq 2 \quad (2.2)$$

Định lý 2.3: *Với mọi j,* $0 \leq j \leq m$, *ta có* $r_j \equiv t_j r_1 (\bmod r_0)$, *với* q_j *và* r_j *được xác định theo thuật toán Euclide và* t_j *được xác định theo công thức truy hồi nêu trên.*

Định lý 2.4: *Nếu* r_0 *và* r_1 *nguyên tố cùng nhau (với* $r_0 > r_1$*) thì* t_m *là phần tử nghịch đảo của* r_1 *trong* $\mathbb{Z}_{r_0}$.

$$\gcd(r_0, r_1) = 1 \Rightarrow t_m = r_1^{-1} \bmod r_0 \quad (2.3)$$

Trong thuật toán Euclide, dãy số $\{t_j\}$ có thể được tính đồng thời với dãy số $\{q_j\}$ và $\{r_j\}$. Thuật toán Euclide mở rộng dưới đây được sử dụng để xác định phần tử nghịch đảo (nếu có) của một số nguyên dương *a* (modulo *n*). Trong thuật toán không cần sử dụng đến cấu trúc dữ liệu mảng để lưu giá trị của dãy số $\{t_j\}$, $\{q_j\}$ hay $\{r_j\}$ vì tại mỗi thời điểm, ta chỉ cần quan tâm đến giá trị của hai phần tử cuối cùng của mỗi dãy tại thời điểm đang xét.

Thuật toán 2.4. *Thuật toán Euclide mở rộng*

xác định phần tử nghịch đảo của a (modulo n)

$n_0 = n$

$a_0 = a$

$t_0 = 0$

$t = 1$

$q = \left\lfloor \frac{n_0}{a_0} \right\rfloor$

$r = n_0 - qa_0$

while $r > 0$ **do**

    $temp = t_0 - qt$

    if $temp \geq 0$ **then**

        $temp = temp \bmod n$

    end if

    if $temp < 0$ **then**

        $temp = n - ((-temp) \bmod n)$

    end if

    $t_0 = t$

    $t = temp$

    $n_0 = a_0$

    $a_0 = r$

    $q = \left\lfloor \frac{n_0}{a_0} \right\rfloor$

    $r = n_0 - qa_0$

end while

if $a_0 \neq 1$ **then**

    a không có phần tử nghịch đảo modulo n

else

    $a^{-1} = t \bmod n$

end if

2.5 Phương pháp Vigenere

Trong phương pháp mã hóa bằng thay thế cũng như các trường hợp đặc biệt của phương pháp này (mã hóa bằng dịch chuyển, mã hóa Affine,…), ứng với một khóa *k* được chọn, mỗi phần tử $x \in P$ được ánh xạ vào duy nhất một phần tử $y \in C$. Nói cách khác, ứng với mỗi khóa $k \in K$, một song ánh được thiết lập từ *P* vào *C*.

Khác với hướng tiếp cận này, phương pháp Vigenere sử dụng một từ khóa có độ dài *m*. Có thể xem như phương pháp mã hóa Vigenere Cipher bao gồm *m* phép mã hóa bằng dịch chuyển được áp dụng luân phiên nhau theo chu kỳ.

Không gian khóa *K* của phương pháp Vigenere Cipher có số phần tử là n^m, lớn hơn hẳn phương pháp số lượng phần tử của không gian khóa *K* trong phương pháp mã hóa bằng dịch chuyển. Do đó, việc tìm ra mã khóa *k* để giải mã thông điệp đã được mã hóa sẽ khó khăn hơn đối với phương pháp mã hóa bằng dịch chuyển.

Thuật toán 2.5. *Phương pháp mã hóa Vigenere*

Chọn số nguyên dương *m*. Định nghĩa $P = C = K = (\mathbb{Z}_n)^m$

$K = \left\{(k_0, k_1, ..., k_{r-1}) \in (\mathbb{Z}_n)^r\right\}$

Với mỗi khóa $k = (k_0, k_1, ..., k_{r-1}) \in K$, định nghĩa:

$e_k(x_1, x_2, ..., x_m) = ((x_1 + k_1) \bmod n, (x_2 + k_2) \bmod n, ..., (x_m + k_m) \bmod n)$

$d_k(y_1, y_2, ..., y_m) = ((y_1 - k_1) \bmod n, (y_2 - k_2) \bmod n, ..., (y_m - k_m) \bmod n)$

với $x, y \in (\mathbb{Z}_n)^m$.

2.6 Phương pháp Hill

Phương pháp Hill được Lester S. Hill công bố năm 1929: Cho số nguyên dương *m*, định nghĩa $P = C = (\mathbb{Z}_n)^m$. Mỗi phần tử $x \in P$ là một bộ *m* thành phần, mỗi thành phần thuộc $\mathbb{Z}_n$. Ý tưởng chính của phương pháp này là sử dụng *m* tổ hợp tuyến tính của *m* thành phần trong mỗi phần tử $x \in P$ để phát sinh ra *m* thành phần tạo thành phần tử $y \in C$.

Thuật toán 2.6. *Phương pháp mã hóa Hill*

Chọn số nguyên dương *m*. Định nghĩa:

$P = C = (\mathbb{Z}_n)^m$ và *K* là tập hợp các ma trận $m \times m$ khả nghịch

Với mỗi khóa $k = \begin{pmatrix} k_{1,1} & k_{1,2} & \cdots & k_{1,m} \\ k_{2,1} & \cdots & \cdots & k_{2,m} \\ \vdots & \vdots & & \vdots \\ k_{m,1} & k_{m,2} & \cdots & k_{m,m} \end{pmatrix} \in K$, định nghĩa:

$$e_k(x) = xk = (x_1, x_2, ..., x_m) \begin{pmatrix} k_{1,1} & k_{1,2} & \cdots & k_{1,m} \\ k_{2,1} & \cdots & \cdots & k_{2,m} \\ \vdots & \vdots & & \vdots \\ k_{m,1} & k_{m,2} & \cdots & k_{m,m} \end{pmatrix} \text{ với } x = (x_1, x_2, ..., x_m) \in P$$

và $d_k(y) = yk^{-1}$ với $y \in C$.

Mọi phép toán số học đều được thực hiện trên $\mathbb{Z}_n$.

2.7 Phương pháp mã hóa hoán vị

Những phương pháp mã hóa nêu trên đều dựa trên ý tưởng chung: thay thế mỗi ký tự trong thông điệp nguồn bằng một ký tự khác để tạo thành thông điệp đã được mã hóa. Ý tưởng chính của phương pháp mã hóa hoán vị (Permutation Cipher) là vẫn giữ nguyên các ký tự trong thông điệp nguồn mà chỉ thay đổi vị trí các ký tự; nói cách khác thông điệp nguồn được mã hóa bằng cách sắp xếp lại các ký tự trong đó.

Thuật toán 2.7. *Phương pháp mã hóa bằng hoán vị*

Chọn số nguyên dương *m*. Định nghĩa:

$P = C = (\mathbb{Z}_n)^m$ và K là tập hợp các hoán vị của m phần tử $\{1, 2, ..., m\}$

Với mỗi khóa $\pi \in K$, định nghĩa:

$e_\pi(x_1, x_2, ..., x_m) = (x_{\pi(1)}, x_{\pi(2)}, ..., x_{\pi(m)})$ và

$d_\pi(y_1, y_2, ..., y_m) = (y_{\pi^{-1}(1)}, y_{\pi^{-1}(2)}, ..., y_{\pi^{-1}(m)})$

với π^{-1} hoán vị ngược của π

Phương pháp mã hóa bằng hoán vị chính là một trường hợp đặc biệt của phương pháp Hill. Với mỗi hoán vị π của tập hợp {1, 2, ..., m} , ta xác định ma trận $k_\pi = (k_{i,j})$ theo công thức sau:

$$k_{i,j} = \begin{cases} 1, & \text{nếu } i = \pi(j) \\ 0, & \text{trong trường hợp ngược lại} \end{cases} \quad (2.4)$$

Ma trận k_π là ma trận mà mỗi dòng và mỗi cột có đúng một phần tử mang giá trị 1, các phần tử còn lại trong ma trận đều bằng 0. Ma trận này có thể thu được bằng cách hoán vị các hàng hay các cột của ma trận đơn vị I_m nên k_π là ma trận khả nghịch. Rõ ràng, mã hóa bằng phương pháp Hill với ma trận k_π hoàn toàn tương đương với mã hóa bằng phương pháp hoán vị với hoán vị π.

2.8 Phương pháp mã hóa bằng phép nhân

2.8.1 Phương pháp mã hóa bằng phép nhân

Thuật toán 2.8. *Phương pháp mã hóa bằng phép nhân*

Cho $P = C = (\mathbb{Z}_n)^m$, $K = \{k \in \mathbb{Z}_n : \gcd(k, n) = 1\}$

Với mỗi khóa $k \in \mathbb{Z}_n$, định nghĩa:

$e_k(x) = k_x \bmod n$ và $d_k(y) = k^{-1} y \bmod n$ với $x, y \in \mathbb{Z}_n$

Phương pháp mã hóa bằng phép nhân (Multiplicative Cipher) là một phương pháp mã hóa đơn giản. Không gian khóa *K* có tất cả $\phi(n)$ phần tử. Tuy nhiên, việc chọn khóa $k = 1 \in K$ sẽ không có ý nghĩa trong việc mã hóa thông nên số lượng phần tử thật sự được sử dụng trong *K* là $\phi(n) - 1$.

Vấn đề được đặt ra ở đây là độ an toàn của phương pháp này phụ thuộc vào số lượng phần tử trong tập khóa *K*. Nếu giá trị $\phi(n) - 1$ không đủ lớn thì thông tin được mã hóa có thể bị giải mã bằng cách thử toàn bộ các khóa $k \in K$. Để nâng

cao độ an toàn của phương pháp này, giá trị *n* được sử dụng phải có $\phi(n)$ đủ lớn hay chính giá trị *n* phải đủ lớn. Khi đó, một vấn đề mới được đặt ra là làm thế nào thực hiện được một cách nhanh chóng các phép toán trên số nguyên lớn.

2.8.2 Xử lý số học

Trong phương pháp mã hóa này, nhu cầu tính giá trị của biểu thức $z = (a \times b) \bmod n$ được đặt ra trong cả thao tác mã hóa và giải mã. Nếu thực hiện việc tính giá trị theo cách thông thường thì rõ ràng là không hiệu quả do thời gian xử lý quá lớn.

Sử dụng thuật toán phép nhân Ấn Độ, ta có thể được sử dụng để tính giá trị biểu thức $z = (a \times b) \bmod n$ một cách nhanh chóng và hiệu quả.

Thuật toán 2.9. *Thuật toán phép nhân Ấn Độ để tính giá trị* $z = (a \times b) \bmod n$

$z = 0$
$a = a \bmod n$
$b = b \bmod n$
Biểu diễn *b* dưới dạng nhị phân $b_{l-1}, b_{l-2}, ..., b_2, b_1$, $b_i \in \{0,1\}$, $0 \le i < l$
for i = 0 **to** $l - 1$
    if $b_i = 1$ **then**
        $z = (z + a) \bmod n$
    end if
    $a = (2a) \bmod n$
end for
$z = (z + a) \bmod n$

2.9 Phương pháp DES (Data Encryption Standard)

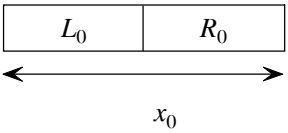
2.9.1 Phương pháp DES

Khoảng những năm 1970, tiến sĩ Horst Feistel đã đặt nền móng đầu tiên cho chuẩn mã hóa dữ liệu DES với phương pháp mã hóa Feistel Cipher. Vào năm 1976 Cơ quan Bảo mật Quốc gia Hoa Kỳ (NSA) đã công nhận DES dựa trên phương pháp Feistel là chuẩn mã hóa dữ liệu [25]. Kích thước khóa của DES ban đầu là 128 bit nhưng tại bản công bố FIPS kích thước khóa được rút xuống còn 56 bit.

Trong phương pháp DES, kích thước khối là 64 bit. DES thực hiện mã hóa dữ liệu qua 16 vòng lặp mã hóa, mỗi vòng sử dụng một khóa chu kỳ 48 bit được tạo ra từ khóa ban đầu có độ dài 56 bit. DES sử dụng 8 bảng hằng số S-box để thao tác.

Quá trình mã hóa của DES có thể được tóm tắt như sau: Biểu diễn thông điệp nguồn $x \in P$ bằng dãy 64bit. Khóa k có 56 bit. Thực hiện mã hóa theo ba giai đoạn:

1. Tạo dãy 64 bit x_0 bằng cách hoán vị x theo hoán vị IP (Initial Permutation). Biểu diễn $x_0 = IP(x) = L_0R_0$, L_0 gồm 32 bit bên trái của x_0, R_0 gồm 32 bit bên phải của x_0.



Hình 2.2. *Biểu diễn dãy 64 bit x thành 2 thành phần L và R*

2. Thực hiện 16 vòng lặp từ 64 bit thu được và 56 bit của khoá *k* (chỉ sử dụng 48 bit của khoá k trong mỗi vòng lặp). 64 bit kết quả thu được qua mỗi vòng lặp sẽ là đầu vào cho vòng lặp sau. Các cặp từ 32 bit L_i, R_i (với $1 \le i \le 16$) được xác định theo quy tắc sau:

$$L_i = R_{i-1}$$

$$R_i = L_{i-1} \oplus f(R_{i-1}, K_i) \tag{2.5}$$

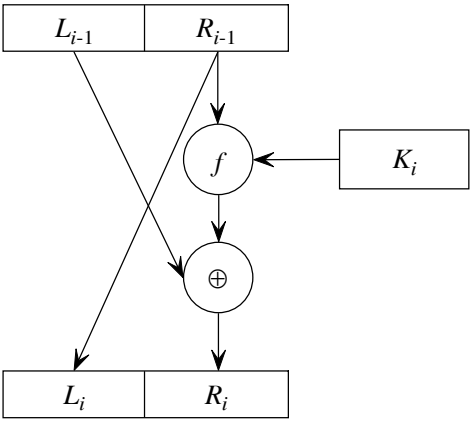
với $\oplus$ biểu diễn phép toán XOR trên hai dãy bit, K_1, K_2, ..., K_{16} là các dãy 48 bit phát sinh từ khóa *K* cho trước (Trên thực tế, mỗi khóa K_i được phát sinh bằng cách hoán vị các bit trong khóa *K* cho trước).

3. Áp dụng hoán vị ngược IP^{-1} đối với dãy bit $R_{16}L_{16}$, thu được từ *y* gồm 64 bit. Như vậy, $y = IP^{-1}(R_{16}L_{16})$.

Hàm *f* được sử dụng ở bước 2 là hàm có gồm hai tham số: Tham số thứ nhất *A* là một dãy 32 bit, tham số thứ hai *J* là một dãy 48 bit. Kết quả của hàm *f* là một dãy 32 bit. Các bước xử lý của hàm $f(A, J)$ như sau:

Tham số thứ nhất *A* (32 bit) được mở rộng thành dãy 48 bit bằng hàm mở rộng *E*. Kết quả của hàm $E(A)$ là một dãy 48 bit được phát sinh từ *A* bằng cách hoán vị

theo một thứ tự nhất định 32 bit của A, trong đó có 16 bit của A được lặp lại hai lần trong $E(A)$.



Hình 2.3. *Quy trình phát sinh dãy L_iR_i từ dãy $L_{i-1}R_{i-1}$ và khóa K_i*

Thực hiện phép toán XOR cho hai dãy 48 bit $E(A)$ và J, ta thu được một dãy 48 bit B. Biểu diễn B thành từng nhóm 6 bit như sau: $B = B_1B_2B_3B_4B_5B_6B_7B_8$.

Sử dụng tám ma trận $S_1, S_2, ..., S_8$, mỗi ma trận S_i có kích thước 4×16 và mỗi dòng của ma trận nhận đủ 16 giá trị từ 0 đến 15. Xét dãy gồm 6 bit $B_j = b_1b_2b_3b_4b_5b_6$, $S_j(B_j)$ được xác định bằng giá trị của phần tử tại dòng r cột c của S_j, trong đó, chỉ số dòng r có biểu diễn nhị phân là b_1b_6, chỉ số cột c có biểu diễn nhị phân là $b_2b_3b_4b_5$. Bằng cách này, ta xác định được các dãy 4 bit $C_j = S_j(B_j)$, $1 \le j \le 8$.

Tập hợp các dãy 4 bit C_j lại, ta có được dãy 32 bit $C = C_1C_2C_3C_4C_5C_6C_7C_8$. Dãy 32 bit thu được bằng cách hoán vị C theo một quy luật P nhất định chính là kết quả của hàm $F(A, J)$.

Quá trình giải mã chính là thực hiện theo thứ tự đảo ngược các thao tác của quá trình mã hóa.

2.9.2 Nhận xét

Do tốc độ tính toán của máy tính ngày càng tăng cao và DES đã được sự quan tâm chú ý của các nhà khoa học lẫn những người phá mã (cryptanalyst) nên DES nhanh chóng trở nên không an toàn. Năm 1997, một dự án đã tiến hành bẻ khóa DES chưa đến 3 ngày với chi phí thấp hơn 250.000 dollars. Và vào năm 1999, một mạng máy tính gồm 100.000 máy có thể giải mã một thư tín mã hóa DES chưa đầy 24 giờ.

Trong quá trình tìm kiếm các thuật toán mới an toàn hơn DES, Tripple DES ra đời như một biến thể của DES. Tripple DES thực hiện ba lần thuật toán DES với 3 khoá khác nhau và với trình tự khác nhau. Trình tự thực hiện phổ biến là EDE (Encrypt – Decrypt – Encrypt), thực hiện xen kẽ mã hóa với giải mã (lưu ý là khóa trong từng giai đoạn thực hiện khác nhau).

2.10 Phương pháp chuẩn mã hóa nâng cao AES

Để tìm kiếm một phương pháp mã hóa quy ước mới với độ an toàn cao hơn DES, NIST đã công bố một chuẩn mã hóa mới, thay thế cho chuẩn DES. Thuật toán đại diện cho chuẩn mã hóa nâng cao AES (Advanced Encryption Standard) sẽ là thuật toán mã hóa khóa quy ước, sử dụng miễn phí trên toàn thế giới. Chuẩn AES bao gồm các yêu cầu sau [23]:

- Thuật toán mã hóa theo khối 128 bit.
- Chiều dài khóa 128 bit, 192 bit và 256 bit.
- Không có khóa yếu.
- Hiệu quả trên hệ thống Intel Pentium Pro và trên các nền phần cứng và phần mềm khác.
- Thiết kế dễ dàng (hỗ trợ chiều dài khóa linh hoạt, có thể triển khai ứng dụng rộng rãi trên các nền và các ứng dụng khác nhau).
- Thiết kế đơn giản: phân tích đánh giá và cài đặt dễ dàng.
- Chấp nhận bất kỳ chiều dài khóa lên đến 256 bit.
- Mã hóa dữ liệu thấp hơn 500 chu kỳ đồng hồ cho mỗi khối trên Intel Pentium, Pentium Pro và Pentium II đối với phiên bản tối ưu của thuật toán.
- Có khả năng thiết lập khóa 128 bit (cho tốc độ mã hóa tối ưu) nhỏ hơn thời gian đòi hỏi để mã hóa các khối 32 bit trên Pentium, Pentium Pro và Pentium II.
- Không chứa bất kỳ phép toán nào làm nó giảm khả năng trên các bộ vi xử lý 8 bit, 16 bit, 32 bit và 64 bit.
- Không bao hàm bất kỳ phần tử nào làm nó giảm khả năng của phần cứng.
- Thời gian mã hóa dữ liệu rất thấp dưới 10/1000 giây trên bộ vi xử lý 8 bit.
- Có thể thực hiện trên bộ vi xử lý 8 bit với 64 byte bộ nhớ RAM.

Sau khi thực hiện hai lần tuyển chọn, có năm thuật toán được vào vòng chung kết, gồm có: MARS, RC6, SERPENT, TWOFISH và RIJNDAEL. Các thuật toán này đều đạt các yêu cầu của AES nên được gọi chung là các thuật toán ứng viên AES. Các thuật toán ứng viên AES có độ an toàn cao, chi phí thực hiện thấp. Chi tiết về các thuật toán này được trình bày trong Chương 3 - Phương pháp mã hóa Rijndael và Chương 5 - Các thuật toán ứng cử viên AES.

Chương 3
Phương pháp mã hóa Rijndael

Nội dung của chương 3 trình bày chi tiết về phương pháp mã hóa Rijndael của hai tác giả Vincent Rijmen và Joan Daeman. Đây là giải thuật được Viện Tiêu chuẩn và Công nghệ Hoa Kỳ (NIST) chính thức chọn làm chuẩn mã hóa nâng cao (AES) từ ngày 02 tháng 10 năm 2000.

3.1 Giới thiệu

Với tốc độ và khả năng xử lý ngày càng được nâng cao của các bộ vi xử lý hiện nay, phương pháp mã hóa chuẩn (Data Encryption Standard – DES) trở nên không an toàn trong bảo mật thông tin. Do đó, Viện Tiêu chuẩn và Công nghệ Hoa Kỳ (National Institute of Standards and Technology – NIST) đã quyết định chọn một chuẩn mã hóa mới với độ an toàn cao nhằm phục vụ nhu cầu bảo mật thông tin liên lạc của Chính phủ Hoa Kỳ cũng như trong các ứng dụng dân sự. Thuật toán Rijndael do Vincent Rijmen và Joan Daeman đã được chính thức chọn trở thành chuẩn mã hóa nâng cao AES (Advanced Encryption Standard) từ ngày 02 tháng 10 năm 2000.

Phương pháp mã hóa Rijndael là phương pháp mã hóa theo khối (block cipher) có kích thước khối và mã khóa thay đổi linh hoạt với các giá trị 128, 192 hay 256 bit. Phương pháp này thích hợp ứng dụng trên nhiều hệ thống khác nhau từ các thẻ thông minh cho đến các máy tính cá nhân.

3.2 Tham số, ký hiệu, thuật ngữ và hàm

AddRoundKey	Phép biến đổi sử dụng trong mã hóa và giải mã, thực hiện việc cộng mã khóa của chu kỳ vào trạng thái hiện hành. Độ dài của mã khóa của chu kỳ bằng với kích thước của trạng thái.
SubBytes	Phép biến đổi sử dụng trong mã hóa, thực hành việc thay thế phi tuyến từng byte trong trạng thái hiện hành thông qua bảng thay thế (S-box).
InvSubBytes	Phép biến đổi sử dụng trong giải mã. Đây là phép biến đổi ngược của phép biến đổi SubBytes.
MixColumns	Phép biến đổi sử dụng trong mã hóa, thực hiện thao tác trộn thông tin của từng cột trong trạng thái hiện hành. Mỗi cột được xử lý độc lập.
InvMixColumns	Phép biến đổi sử dụng trong giải mã. Đây là phép biến đổi ngược của phép biến đổi MixColumns.

ShiftRows	Phép biến đổi sử dụng trong mã hóa, thực hiện việc dịch chuyển xoay vòng từng dòng của trạng thái hiện hành với di số tương ứng khác nhau
InvShiftRows	Phép biến đổi sử dụng trong giải mã. Đây là phép biến đổi ngược của phép biến đổi ShiftRows.
Nw	Số lượng byte trong một đơn vị dữ liệu "từ". Trong thuật toán Rijndael, thuật toán mở rộng 256/384/512 bit và thuật toán mở rộng 512/768/1024 bit, giá trị *Nw* lần lượt là 4, 8 và 16
K	Khóa chính.
Nb	Số lượng cột (số lượng các từ 8×*Nw* bit) trong trạng thái. Giá trị *Nb* = 4, 6, hay 8. Chuẩn AES giới hạn lại giá trị của *Nb* = 4.
Nk	Số lượng các từ (8×*Nw* bit) trong khóa chính. Giá trị *Nk* = 4, 6, hay 8.
Nr	Số lượng chu kỳ, phụ thuộc vào giá trị *Nk* and *Nb* theo công thức: *Nr* = max (*Nb*, *Nk*)+6.

`RotWord`	Hàm được sử dụng trong quá trình mở rộng mã khóa, thực hiện thao tác dịch chuyển xoay vòng *Nw* byte thành phần của một từ.
`SubWord`	Hàm được sử dụng trong quá trình mở rộng mã khóa. Nhận vào một từ (*Nw* byte), áp dụng phép thay thế dựa vào S-box đối với từng byte thành phần và trả về từ gồm *Nw* byte thành phần đã được thay thế.
XOR	Phép toán Exclusive-OR.
$\oplus$	Phép toán Exclusive-OR.
$\otimes$	Phép nhân hai đa thức (mỗi đa thức có bậc < *Nw*) modulo cho đa thức $x^{Nw} + 1$.
$\bullet$	Phép nhân trên trường hữu hạn.

3.3 Một số khái niệm toán học

Đơn vị thông tin được xử lý trong thuật toán Rijndael là byte. Mỗi byte xem như một phần tử của trường Galois GF(2^8) được trang bị phép cộng (ký hiệu $\oplus$) và phép nhân (ký hiệu $\bullet$). Mỗi byte có thể được biểu diễn bằng nhiều cách khác

nhau: dạng nhị phân ($\{b_7b_6b_5b_4b_3b_2b_1b_0\}$), dạng thập lục phân ($\{h_1h_0\}$) hay dạng đa thức có các hệ số nhị phân $\sum_{i=0}^{7} b_i x^i$

3.3.1 Phép cộng

Phép cộng hai phần tử trên GF(2^8) được thực hiện bằng cách "cộng" (thực chất là phép toán XOR, ký hiệu $\oplus$) các hệ số của các đơn thức đồng dạng của hai đa thức tương ứng với hai toán hạng đang xét. Như vậy, phép cộng và phép trừ hai phần tử bất kỳ trên GF(2^8) là hoàn toàn tương đương nhau.

Nếu biểu diễn lại các phần tử thuộc GF(2^8) dưới hình thức nhị phân thì phép cộng giữa $\{a_7a_6a_5a_4a_3a_2a_1a_0\}$ với $\{b_7b_6b_5b_4b_3b_2b_1b_0\}$ là $\{c_7c_6c_5c_4c_3c_2c_1c_0\}$ với $c_i = a_i \oplus b_j$, $0 \le i \le 7$.

3.3.2 Phép nhân

Khi xét trong biểu diễn đa thức, phép nhân trên GF(2^8) (ký hiệu $\bullet$) tương ứng với phép nhân thông thường của hai đa thức đem chia lấy dư (modulo) cho một *đa thức tối giản* (irreducible polynomial) bậc 8. Đa thức được gọi là tối giản khi và chỉ khi đa thức này chỉ chia hết cho 1 và chính mình. Trong thuật toán Rijndael, đa thức *tối giản* được chọn là

$$m(x) = x^8 + x^4 + x^3 + x + 1 \qquad (3.1)$$

hay `1{1b}` trong biểu diễn dạng thập lục phân.

Kết quả nhận được là một đa thức bậc nhỏ hơn 8 nên có thể được biểu diễn dưới dạng 1 byte. Phép nhân trên GF(2^8) không thể được biểu diễn bằng một phép toán đơn giản ở mức độ byte.

Phép nhân được định nghĩa trên đây có tính kết hợp, tính phân phối đối với phép cộng và có phần tử đơn vị là `{01}`. Với mọi đa thức $b(x)$ có hệ số nhị phân với bậc nhỏ hơn 8 tồn tại phần tử nghịch đảo của $b(x)$, ký hiệu $b^{-1}(x)$ (được thực hiện bằng cách sử dụng thuật toán Euclide mở rộng [45]).

Nhận xét: Tập hợp 256 giá trị từ 0 đến 255 được trang bị phép toán cộng (được định nghĩa là phép toán XOR) và phép nhân định nghĩa như trên tạo thành trường hữu hạn GF(2^8).

3.3.2.1 Phép nhân với x

Phép nhân (thông thường) đa thức

$$b(x) = b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0 = \sum_{i=0}^{7} b_i x^i \qquad (3.2)$$

với đa thức x cho kết quả là đa thức

$$b_7x^8 + b_6x^7 + b_5x^6 + b_4x^5 + b_3x^4 + b_2x^3 + b_1x^2 + b_0x \qquad (3.3)$$

Kết quả $x \bullet b(x)$ được xác định bằng cách modulo kết quả này cho đa thức $m(x)$.

1. Trường hợp $b_7 = 0$

$$x \bullet b(x) = b_6x^7 + b_5x^6 + b_4x^5 + b_3x^4 + b_2x^3 + b_1x^2 + b_0x \qquad (3.4)$$

2. Trường hợp $b_7 = 1$

$$x \bullet b(x) = \left(b_7 x^8 + b_6 x^7 + b_5 x^6 + b_4 x^5 + b_3 x^4 + b_2 x^3 + b_1 x^2 + b_0 x\right) \bmod m(x)$$
$$= \left(b_7 x^8 + b_6 x^7 + b_5 x^6 + b_4 x^5 + b_3 x^4 + b_2 x^3 + b_1 x^2 + b_0 x\right) - m(x) \quad (3.5)$$

Như vậy, phép nhân với đa thức *x* (hay phần tử {00000010} ∈ GF(2^8)) có thể được thực hiện ở mức độ byte bằng một phép shift trái và sau đó thực hiện tiếp phép toán XOR với giá trị {1b} nếu $b_7 = 1$. Thao tác này được ký hiệu là xtime(). Phép nhân với các lũy thừa của *x* có thể được thực hiện bằng cách áp dụng nhiều lần thao tác xtime(). Kết quả của phép nhân với một giá trị bất kỳ được xác định bằng cách cộng ($\oplus$) các kết quả trung gian này lại với nhau.

Khi đó, việc thực hiện phép nhân giữa hai phần tử *a*, *b* bất kỳ thuộc GF(2^8) có thể được tiến hành theo các bước sau:

1. Phân tích một phần tử (giả sử là *a*) ra thành tổng của các lũy thừa của 2.

2. Tính tổng các kết quả trung gian của phép nhân giữa phần tử còn lại (là *b*) với các thành phần là lũy thừa của 2 được phân tích từ *a*.

❒ Ví dụ:

{57} • {13} = {fe} vì

{57} • {02} = xtime({57}) = {ae}

{57} • {04} = xtime({ae}) = {47}

{57} • {08} = xtime({47}) = {8e}

{57} • {10} = xtime({8e}) = {07},

Như vậy:

$$\begin{aligned} \{57\} \bullet \{13\} &= \{57\} \bullet (\{01\} \oplus \{02\} \oplus \{10\}) \\ &= \{57\} \oplus \{ae\} \oplus \{07\} \\ &= \{fe\} \end{aligned}$$

3.3.3 Đa thức với hệ số trên GF(2^8)

Xét đa thức $a(x)$ và $b(x)$ bậc 4 với các hệ số thuộc GF(2^8):

$$a(x) = \sum_{i=0}^{3} a_i x^i \quad \text{và} \quad b(x) = \sum_{i=0}^{3} b_i x^i \tag{3.6}$$

Hai đa thức này có thể được biểu diễn lại dưới dạng từ gồm 4 byte $[a_0, a_1, a_2, a_3]$ và $[b_0, b_1, b_2, b_3]$. Phép cộng đa thức được thực hiện bằng cách cộng (chính là phép toán XOR trên byte) các hệ số của các đơn thức đồng dạng với nhau:

$$a(x) + b(x) = \sum_{i=0}^{3} (a_i \oplus b_i) x^i \tag{3.7}$$

Phép nhân giữa $a(x)$ với $b(x)$ được thực hiện thông qua hai bước. Trước tiên, thực hiện phép nhân thông thường $c(x) = a(x)b(x)$.

$$c(x) = c_6 x^6 + c_5 x^5 + c_4 x^4 + c_3 x^3 + c_2 x^2 + c_1 x + c_0 \tag{3.8}$$

với

$$\begin{aligned} c_0 &= a_0 \bullet b_0 & c_4 &= a_3 \bullet b_1 \oplus a_2 \bullet b_2 \oplus a_1 \bullet b_3 \\ c_1 &= a_1 \bullet b_0 \oplus a_0 \bullet b_1 & c_5 &= a_3 \bullet b_2 \oplus a_2 \bullet b_3 \\ c_2 &= a_2 \bullet b_0 \oplus a_1 \bullet b_1 \oplus a_0 \bullet b_2 & c_6 &= a_3 \bullet b_3 \\ c_3 &= a_3 \bullet b_0 \oplus a_2 \bullet b_1 \oplus a_1 \bullet b_2 \oplus a_0 \bullet b_3 . \end{aligned} \tag{3.9}$$

Rõ ràng là $c(x)$ không thể được biểu diễn bằng một từ gồm 4 byte. Đa thức $c(x)$ có thể được đưa về một đa thức có bậc nhỏ hơn 4 bằng cách lấy $c(x)$ modulo cho một đa thức bậc 4. Trong thuật toán Rijndael, đa thức bậc 4 được chọn là $M(x) = x^4 + 1$.

Do $x^j \bmod (x^4 + 1) = x^{j \bmod 4}$ nên kết quả $d(x) = a(x) \otimes b(x)$ được xác định bằng

$$d(x) = d_3 x^3 + d_2 x^2 + d_1 x + d_0 \tag{3.10}$$

với

$$\begin{aligned} d_0 &= a_0 \bullet b_0 \oplus a_3 \bullet b_1 \oplus a_2 \bullet b_2 \oplus a_1 \bullet b_3 \\ d_1 &= a_1 \bullet b_0 \oplus a_0 \bullet b_1 \oplus a_3 \bullet b_2 \oplus a_2 \bullet b_3 \\ d_2 &= a_2 \bullet b_0 \oplus a_1 \bullet b_1 \oplus a_0 \bullet b_2 \oplus a_3 \bullet b_3 \\ d_3 &= a_3 \bullet b_0 \oplus a_2 \bullet b_1 \oplus a_1 \bullet b_2 \oplus a_0 \bullet b_3 \end{aligned} \tag{3.11}$$

Trong trường hợp đa thức $a(x)$ cố định, phép nhân $d(x) = a(x) \otimes b(x)$ có thể được biểu diễn dưới dạng ma trận như sau

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} \tag{3.12}$$

Do $x^4 + 1$ không phải là một đa thức tối giản trên GF(2^8) nên phép nhân với một đa thức $a(x)$ cố định được chọn bất kỳ không đảm bảo tính khả nghịch. Vì vậy, trong phương pháp Rijndael đã chọn đa thức $a(x)$ có phần tử nghịch đảo (modulo $M(x)$)

$$a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\} \tag{3.13}$$

$$a^{-1}(x) = \{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\} \tag{3.14}$$

3.3.3.1 Phép nhân với x

Xét đa thức

$$b(x) = b_3x^3 + b_2x^2 + b_1x + b_0 \tag{3.15}$$

Kết quả của phép nhân $c(x) = b(x) \otimes x$ được xác định bằng

$$c(x) = b_2x^3 + b_1x^2 + b_0x + b_3 \tag{3.16}$$

Phép nhân với x tương đương với phép nhân ở dạng ma trận như đã trình bày ở phần trên với các giá trị $a_0 = a_2 = a_3 =$ `{00}` và $a_1 =$ `{01}`.

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} 00 & 00 & 00 & 01 \\ 01 & 00 & 00 & 00 \\ 00 & 01 & 00 & 00 \\ 00 & 00 & 01 & 00 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} \tag{3.17}$$

Như vậy, phép nhân với x hay các lũy thừa của x sẽ tương ứng với phép dịch chuyển xoay vòng các byte thành phần trong một từ.

Trong thuật toán Rijndael cần sử dụng đến đa thức x^3 ($a_0 = a_1 = a_2 =$ `{00}` và $a_3 =$ `{01}`)trong hàm `RotWord` nhằm xoay vòng 4 byte thành phần của một từ được đưa vào. Như vậy, nếu đưa vào từ gồm 4 byte [b_0, b_1, b_2, b_3] thì kết quả nhận được là từ gồm 4 byte [b_1, b_2, b_3, b_0].

3.4 Phương pháp Rijndael

Phương pháp mã hóa Rijndael bao gồm nhiều bước biến đổi được thực hiện tuần tự, kết quả đầu ra của bước biến đổi trước là đầu vào của bước biến đổi tiếp theo. Kết quả trung gian giữa các bước biến đổi được gọi là *trạng thái* (state).

Một trạng thái có thể được biểu diễn dưới dạng một ma trận gồm 4 dòng và *Nb* cột với *Nb* bằng với độ dài của khối chia cho 32. Mã khóa chính (Cipher Key) cũng được biểu diễn dưới dạng một ma trận gồm 4 dòng và *Nk* cột với *Nk* bằng với độ dài của khóa chia cho 32. Trong một số tình huống, ma trận biểu diễn một trạng thái hay mã khóa có thể được khảo sát như mảng một chiều chứa các phần tử có độ dài 4 byte, mỗi phần tử tương ứng với một cột của ma trận.

Số lượng chu kỳ, ký hiệu là *Nr*, phụ thuộc vào giá trị của *Nb* và *Nk* theo công thức: $Nr = \max\{Nb, Nk\} + 6$

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$	$a_{0,4}$	$a_{0,5}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	$a_{2,5}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$	$a_{3,5}$

$k_{0,0}$	$k_{0,1}$	$k_{0,2}$	$k_{0,3}$
$k_{1,0}$	$k_{1,1}$	$k_{1,2}$	$k_{1,3}$
$k_{2,0}$	$k_{2,1}$	$k_{2,2}$	$k_{2,3}$
$k_{3,0}$	$k_{3,1}$	$k_{3,2}$	$k_{3,3}$

Hình 3.1. *Biểu diễn dạng ma trận của trạng thái (Nb = 6) và mã khóa (Nk = 4)*

3.4.1 Quy trình mã hóa

Quy trình mã hóa Rijndael sử dụng bốn phép biến đổi chính:

1. AddRoundKey: cộng (⊕) mã khóa của chu kỳ vào trạng thái hiện hành. Độ dài của mã khóa của chu kỳ bằng với kích thước của trạng thái.

2. SubBytes: thay thế phi tuyến mỗi byte trong trạng thái hiện hành thông qua bảng thay thế (S-box).

3. MixColumns: trộn thông tin của từng cột trong trạng thái hiện hành. Mỗi cột được xử lý độc lập.

4. ShiftRows: dịch chuyển xoay vòng từng dòng của trạng thái hiện hành với di số khác nhau.

Mỗi phép biến đổi thao tác trên trạng thái hiện hành *S*. Kết quả *S'* của mỗi phép biến đổi sẽ trở thành đầu vào của phép biến đổi kế tiếp trong quy trình mã hóa.

Trước tiên, toàn bộ dữ liệu đầu vào được chép vào mảng trạng thái hiện hành. Sau khi thực hiện thao tác cộng mã khóa đầu tiên, mảng trạng thái sẽ được trải qua $Nr = 10$, 12 hay 14 chu kỳ biến đổi (tùy thuộc vào độ dài của mã khóa chính cũng như độ dài của khối được xử lý). $Nr - 1$ chu kỳ đầu tiên là các chu kỳ biến đổi bình thường và hoàn toàn tương tự nhau, riêng chu kỳ biến đổi cuối cùng có sự khác biệt so với $Nr - 1$ chu kỳ trước đó. Cuối cùng, nội dung của mảng trạng thái sẽ được chép lại vào mảng chứa dữ liệu đầu ra.

Quy trình mã hóa Rijndael được tóm tắt lại như sau:

1. Thực hiện thao tác AddRoundKey đầu tiên trước khi thực hiện các chu kỳ mã hóa.
2. *Nr* – 1 chu kỳ mã hóa bình thường: mỗi chu kỳ bao gồm bốn bước biến đổi liên tiếp nhau: SubBytes, ShiftRows, MixColumns, và AddRoundKey.
3. Thực hiện `chu` kỳ mã hóa cuối cùng: trong chu kỳ này thao tác MixColumns được bỏ qua.

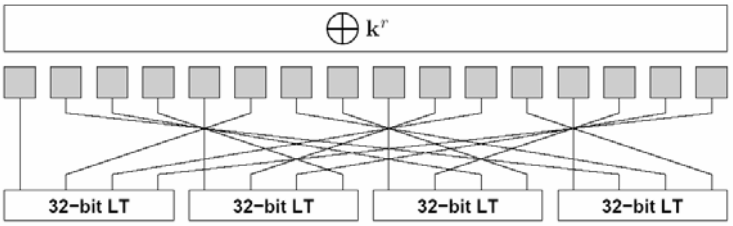
Trong thuật toán dưới đây, mảng `w[]` chứa bảng mã khóa mở rộng; mảng `in[]` và `out[]` lần lượt chứa dữ liệu vào và kết quả ra của thuật toán mã hóa.

```
Cipher( byte in[4 * Nb],
        byte out[4 * Nb],
        word w[Nb * (Nr + 1)])
begin
   byte   state[4,Nb]
   state =   in
   AddRoundKey(state, w)                    // Xem phần 3.4.6
   for round = 1 to Nr - 1
      SubBytes(state)                       // Xem phần 3.4.2
      ShiftRows(state)                      // Xem phần 3.4.4
      MixColumns(state)                     // Xem phần 3.4.5
      AddRoundKey(state, w + round * Nb)
   end for
   SubBytes(state)
   ShiftRows(state)
   AddRoundKey(state, w + Nr * Nb)
   out = state
end
```

3.4.2 Kiến trúc của thuật toán Rijndael

Thuật toán Rijndael được xây dựng theo kiến trúc SPN sử dụng 16 s-box (kích thước 8 × 8) để thay thế. Trong toàn bộ quy trình mã hóa, thuật toán sử dụng chung bảng thay thế s-box cố định. Phép biến đổi tuyến tính bao gồm 2 bước: hoán vị byte và áp dụng song song bốn khối biến đổi tuyến tính (32 bit) có khả năng khuếch tán cao. Hình 3.2 thể hiện một chu kỳ mã hóa của phương pháp Rijndael.

Trên thực tế, trong mỗi chu kỳ mã hóa, khóa của chu kỳ được cộng (XOR) sau thao tác biến đổi tuyến tính. Do chúng ta có thực hiện thao tác cộng khóa trước khi thực hiện chu kỳ đầu tiên nên có thể xem thuật toán Rijndael thỏa cấu trúc SPN [29].



Hình 3.2. *Một chu kỳ mã hóa của phương pháp Rijndael (với Nb = 4)*

3.4.3 Phép biến đổi SubBytes

Thao tác biến đổi SubBytes là phép thay thế các byte phi tuyến và tác động một cách độc lập lên từng byte trong trạng thái hiện hành. Bảng thay thế (S-box) có tính khả nghịch và quá trình thay thế 1 byte x dựa vào S-box bao gồm hai bước:

1. Xác định phần tử nghịch đảo $x^{-1} \in GF(2^8)$. Quy ước $\{00\}^{-1} = \{00\}$.

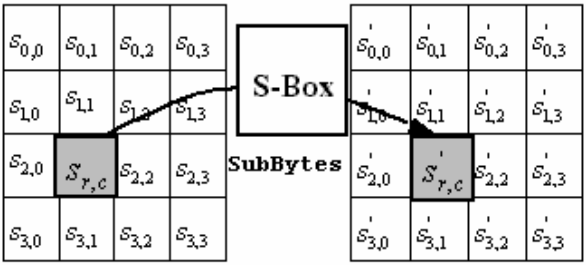
2. Áp dụng phép biến đổi affine (trên GF(2)) đối với x^{-1} (giả sử x^{-1} có biểu diễn nhị phân là $\{x_7x_6x_5x_4x_3x_2x_1x_0\}$):

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \tag{3.18}$$

hay

$$y_i = x_i \oplus x_{(i+4) \bmod 8} \oplus x_{(i+5) \bmod 8} \oplus x_{(i+6) \bmod 8} \oplus x_{(i+7) \bmod 8} \oplus c_i \tag{3.19}$$

với c_i là bit thứ i của $\{63\}$, $0 \le i \le 7$.



Hình 3.3. *Thao tác SubBytes tác động trên từng byte của trạng thái*

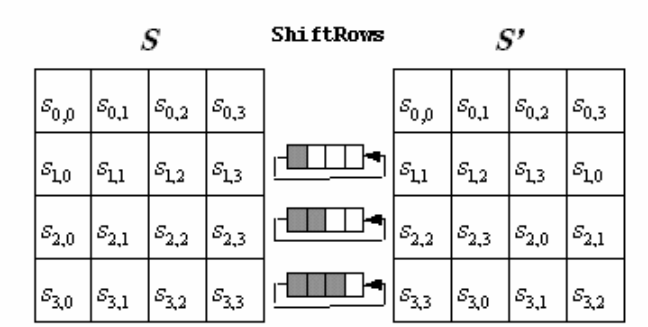
Bảng D.1 thể hiện bảng thay thế S-box được sử dụng trong phép biến đổi SubBytes ở dạng thập lục phân.

❒ Ví dụ: nếu giá trị `{xy}` cần thay thế là `{53}` thì giá trị thay thế S-box (`{xy}`) được xác định bằng cách lấy giá trị tại dòng 5 cột 3 của Bảng D.1. Như vậy, S-box (`{xy}`) = `{ed}`.

Phép biến đổi SubBytes được thể hiện dưới dạng mã giả:

```
SubBytes(byte state[4,Nb])
begin
    for r = 0 to 3
        for c = 0 to Nb - 1
            state[r,c] = Sbox[state[r,c]]
        end for
    end for
end
```

3.4.4 Phép biến đổi ShiftRows



Hình 3.4. *Thao tác ShiftRows tác động trên từng dòng của trạng thái*

Trong thao tác biến đổi ShiftRows, mỗi dòng của trạng thái hiện hành được dịch chuyển xoay vòng đi một số vị trí.

Byte $S_{r,c}$ tại dòng *r* cột *c* sẽ dịch chuyển đến cột (*c* - *shift*(*r*, *Nb*)) mod *Nb* hay:

$$S'_{r,c} = S_{r,(c+shift(r,Nb))\bmod Nb} \quad \text{với } 0 < r < 8 \quad \text{và } 0 \le c < Nb \tag{3.20}$$

Giá trị di số *shift*(*r*, *Nb*) phụ thuộc vào chỉ số dòng *r* và kích thước `Nb` của khối dữ liệu.

Bảng 3.1. *Giá trị di số shift(r, Nb)*

***shift*(r, *Nb*)**		*r*		
		1	2	3
Nb	4	1	2	3
	6	1	2	3
	8	1	3	4

Phép biến đổi ShiftRows được thể hiện dưới dạng mã giả:

```
ShiftRows(byte state[4,Nb])
begin
   byte t[Nb]
   for r = 1 to 3
      for c = 0 to Nb - 1
         t[c] = state[r, (c + h[r,Nb]) mod Nb]
      end for
      for c = 0 to Nb - 1
         state[r,c] = t[c]
      end for
   end for
end
```

3.4.5 Phép biến đổi MixColumns

Trong thao tác biến đổi MixColumns, mỗi cột của trạng thái hiện hành được biểu diễn dưới dạng đa thức $s(x)$ có các hệ số trên GF(2^8). Thực hiện phép nhân

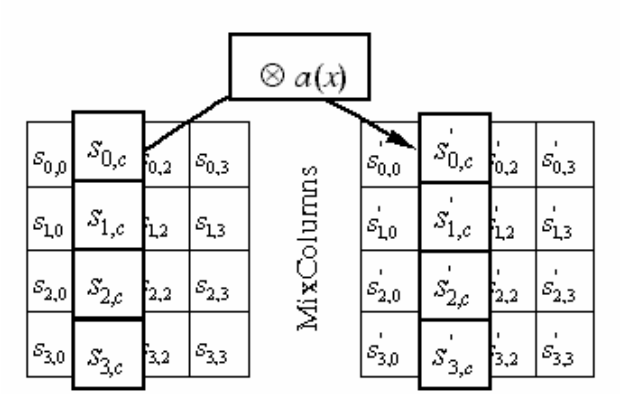
$$s'(x) = a(x) \otimes s(x) \tag{3.21}$$

với

$$a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\} \tag{3.22}$$

Thao tác này được thể hiện ở dạng ma trận như sau:

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \tag{3.23}$$



Hình 3.5. *Thao tác MixColumns tác động lên mỗi cột của trạng thái*

Trong đoạn mã chương trình dưới đây, hàm `FFmul(x, y)` thực hiện phép nhân (trên trường GF(2^8)) hai phần tử *x* và *y* với nhau

```
MixColumns(byte state[4,Nb])
begin
    byte t[4]
    for c = 0 to Nb - 1
        for r = 0 to 3
            t[r] = state[r,c]
        end for
        for r = 0 to 3
            state[r,c] =
                FFmul(0x02, t[r])              xor
                FFmul(0x03, t[(r + 1) mod 4])  xor
                t[(r + 2) mod 4]               xor
                t[(r + 3) mod 4]
        end for
    end for
end
```

3.4.6 Thao tác AddRoundKey

Phương pháp Rijndael bao gồm nhiều chu kỳ mã hóa liên tiếp nhau, mỗi chu kỳ có một mã khóa riêng (Round Key) có cùng kích thước với khối dữ liệu đang được xử lý và được phát sinh từ mã khóa chính (Cipher Key) cho trước ban đầu. Mã khóa của chu kỳ cũng được biểu diễn bằng một ma trận gồm 4 dòng và *Nb* cột. Mỗi cột của trạng thái hiện hành được XOR với cột tương ứng của mã khóa của chu kỳ đang xét:

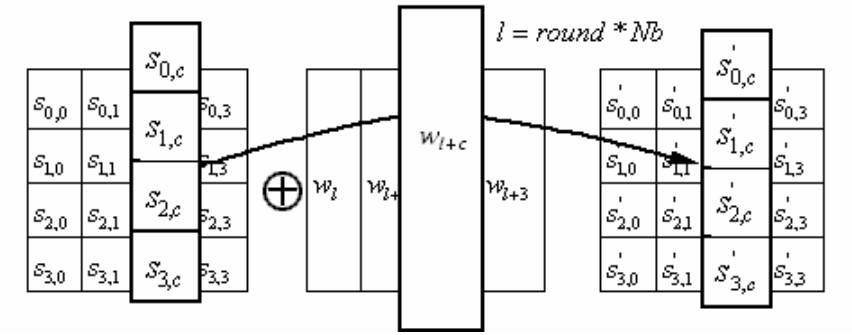
$$[s'_{0,c}, s'_{1,c}, s'_{2,c}, s'_{3,c}] = [s_{0,c}, s_{1,c}, s_{2,c}, s_{3,c}] \oplus [w_{round*Nb+c}], \quad (3.24)$$

với $0 \le c < Nb$.

Thao tác biến đổi ngược của AddRoundKey cũng chính là thao tác AddRoundKey.

Trong đoạn chương trình dưới đây, hàm `xbyte(r, w)` thực hiện việc lấy byte thứ *r* trong từ *w*.

```
AddRoundKey(byte state[4,Nb], word rk[])
// rk = w + round * Nb
begin
    for c = 0 to Nb - 1
        for r = 0 to 3
            state[r,c] = state[r,c] xor xbyte(r, rk[c])
        end for
    end for
end
```



Hình 3.6. *Thao tác AddRoundKey tác động lên mỗi cột của trạng thái*

3.5 Phát sinh khóa của mỗi chu kỳ

Các khóa của mỗi chu kỳ (RoundKey) được phát sinh từ khóa chính. Quy trình phát sinh khóa cho mỗi chu kỳ gồm 2 giai đoạn::

1. Mở rộng khóa chính thành bảng khóa mở rộng,

2. Chọn khóa cho mỗi chu kỳ từ bảng khóa mở rộng.

3.5.1 Xây dựng bảng khóa mở rộng

Bảng khóa mở rộng là mảng 1 chiều chứa các từ (có độ dài 4 byte), được ký hiệu là $w[Nb*(Nr + 1)]$. Hàm phát sinh bảng khóa mở rộng phụ thuộc vào giá trị *Nk*, tức là phụ thuộc vào độ dài của mã khóa chính.

Hàm `SubWord(W)` thực hiện việc thay thế (sử dụng `S-box`) từng byte thành phần của từ 4 byte được đưa vào và trả kết quả về là một từ bao gồm 4 byte kết quả sau khi thực hiệc việc thay thế.

Hàm `RotWord(W)` thực hiện việc dịch chuyển xoay vòng 4 byte thành phần (*a*, *b*, *c*, *d*) của từ được đưa vào. Kết quả trả về của hàm `RotWord` là một từ gồm 4 byte thành phần là (*b*, *c*, *d*, *a*).

```
KeyExpansion(byte key[4 * Nk], word w[Nb * (Nr + 1)], Nk)
begin
    i=0
   while (i < Nk)
        w[i] = word[key[4*i],key[4*i+1],
                    key[4*i+2],key[4*i+3]]
        i = i + 1
    end while
    i = Nk
   while (i < Nb * (Nr + 1))
        word temp = w[i - 1]
        if (i mod Nk = 0) then
            temp = SubWord(RotWord(temp))  xor Rcon[i / Nk]
        else
            if (Nk = 8) and (i mod Nk = 4) then
                temp = SubWord(temp)
            end if
     w[i] = w[i - Nk] xor temp
    i = i + 1
    end while
end
```

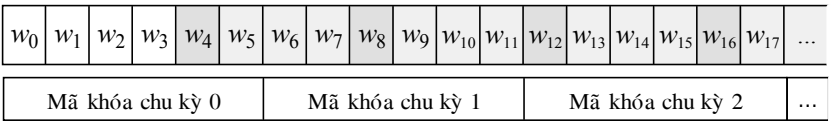
Các hằng số của mỗi chu kỳ hoàn toàn độc lập với giá trị *Nk* và được xác định bằng Rcon[*i*] = (RC[*i*], {00}, {00}, {00}) với RC[*i*] ∈ GF(2^8) và thỏa:

$$RC[1]=1\ (\{01\})$$

$$RC[i] = x\ (\{02\}) \bullet (RC[i\text{-}1]) = x^{(i-1)} \qquad (3.25)$$

3.5.2 Xác định khóa của chu kỳ

Khóa của chu kỳ thứ *i* được xác định bao gồm các từ (4 byte) có chỉ số từ $Nb * i$ đến $Nb * (i+1) - 1$ của bảng mã khóa mở rộng. Như vậy, mã khóa của chu kỳ thứ *i* bao gồm các phần tử $w[Nb * i]$, $w[Nb * i + 1]$,…, $w[Nb * (i+1) - 1]$.



Hình 3.7. *Bảng mã khóa mở rộng và cách xác định mã khóa của chu kỳ (Nb = 6 và Nk = 4)*

Việc phát sinh mã khóa cho các chu kỳ có thể được thực hiện mà không nhất thiết phải sử dụng đến mảng $w[Nb * (Nr + 1)]$. Trong trường hợp dung lượng bộ nhớ hạn chế như ở các thẻ thông minh, các mã khóa cho từng chu kỳ có thể được xác định khi cần thiết ngay trong quá trình xử lý mà chỉ cần sử dụng $\max(Nk, Nb) * 4$ byte trong bộ nhớ.

Bảng khóa mở rộng luôn được tự động phát sinh từ khóa chính mà không cần phải được xác định trực tiếp từ người dùng hay chương trình ứng dụng. Việc

chọn lựa khóa chính (Cipher Key) là hoàn toàn tự do và không có một điều kiện ràng buộc hay hạn chế nào.

3.6 Quy trình giải mã

Quy trình giải mã được thực hiện qua các giai đoạn sau:

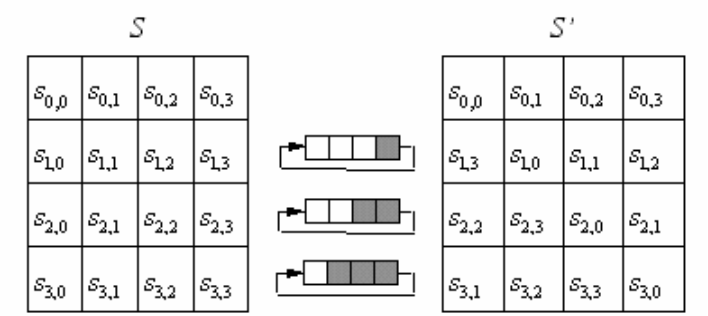
1. Thực hiện thao tác AddRoundKey đầu tiên trước khi thực hiện các chu kỳ giải mã.
2. $Nr-1$ chu kỳ giải mã bình thường: mỗi chu kỳ bao gồm bốn bước biến đổi liên tiếp nhau: InvShiftRows, InvSubBytes, AddRoundKey, InvMixColumns.
3. Thực hiện chu kỳ giải mã cuối cùng. Trong chu kỳ này, thao tác InvMixColumns được bỏ qua.

Dưới đây là mã giả của quy trình giải mã:

```
InvCipher(byte in[4 * Nb],
          byte out[4 * Nb],
          word w[Nb * (Nr + 1)])
begin
   byte   state[4,Nb]
   state = in
   AddRoundKey(state, w + Nr * Nb)          // Xem phần 3.4.6
   for round = Nr - 1 downto 1
      InvShiftRows(state)                   // Xem phần 3.6.1
      InvSubBytes(state)                    // Xem phần 3.6.2
      AddRoundKey(state, w + round * Nb)
      InvMixColumns(state)                  // Xem phần 3.6.3
   end for
```

```
    InvShiftRows(state)
    InvSubBytes(state)
    AddRoundKey(state, w)
    out = state
end
```

3.6.1 Phép biến đổi InvShiftRows



Hình 3.8. *Thao tác InvShiftRows tác động lên từng dòng của trạng thái hiện hành*

InvShiftRows chính là phép biến đổi ngược của phép biến đổi ShiftRows. Dòng đầu tiên của trạng thái sẽ vẫn được giữ nguyên trong khác ba dòng cuối của trạng thái sẽ được dịch chuyển xoay vòng theo chiều ngược với phép biến đổi ShiftRows với các di số *Nb–shift* (*r*, *Nb*) khác nhau. Các byte ở cuối dòng được đưa vòng lên đầu dòng trong khi các byte còn lại có khuynh hướng di chuyển về cuối dòng.

$$s'_{r,(c+shift(r,Nb)) \bmod Nb} = s_{r,c} \text{ với } 0 < r < 4 \text{ và } 0 \le c < Nb \qquad (3.26)$$

Giá trị của di số *shift(r,Nb)* phụ thuộc vào chỉ số dòng *r* và kích thước *Nb* của khối và được thể hiện trong Bảng 3.1.

```
InvShiftRows(byte state[4,Nb])
begin
   byte t[Nb]
   for r = 1 to 3
      for c = 0 to Nb - 1
         t[(c + h[r,Nb]) mod Nb] = state[r,c]
      end for
      for c = 0 to Nb - 1
         state[r,c] = t[c]
      end for
   end for
end
```

3.6.2 Phép biến đổi InvSubBytes

Phép biến đổi ngược của thao tác `SubBytes`, ký hiệu là InvSubBytes, sự dụng bảng thay thế nghịch đảo của `S-box` trên GF(2^8), ký hiệu là `S-box`$^{-1}$. Quá trình thay thế 1 byte *y* dựa vào `S-box`$^{-1}$ bao gồm hai bước sau:

1. Áp dụng phép biến đổi affine (trên GF(2)) sau đối với *y* (có biểu diễn nhị phân là $\{y_7y_6y_5y_4y_3y_2y_1y_0\}$):

$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad (3.27)$$

hay

$$x_i = y_{(i+2) \bmod 8} \oplus y_{(i+5) \bmod 8} \oplus y_{(i+7) \bmod 8} \oplus d_i,$$

với d_i là bit thứ i của giá trị `{05}`, $0 \le i \le 7$. (3.28)

Rõ ràng đây chính là phép biến đổi affine ngược của phép biến đổi affine ở bước 1 của `S-box`.

2. Gọi x là phần tử thuộc GF(2^8) có biểu diễn nhị phân là $\{x_7 x_6 x_5 x_4 x_3 x_2 x_1 x_0\}$.

   Xác định phần tử nghịch đảo $x^{-1} \in$ GF(2^8) với quy ước `{00}`$^{-1}$ = `{00}`

```
InvSubBytes(byte state[4,Nb])
begin
    for r = 0 to 3
        for c = 0 to Nb - 1
            state[r,c] = InvSbox[state[r,c]]
        end for
    end for
end
```

Bảng D.2 thể hiện bảng thay thế nghịch đảo được sử dụng trong phép biến đổi InvSubBytes

3.6.3 Phép biến đổi InvMixColumns

InvMixColumns là biến đổi ngược của phép biến đổi MixColumns. Mỗi cột của trạng thái hiện hành được xem như đa thức $s(x)$ bậc 4 có các hệ số thuộc GF(2^8) và được nhân với đa thức $a^{-1}(x)$ là nghịch đảo của đa thức $a(x)$ (modulo $M(x)$) được sử dụng trong phép biến đổi MixColumns.

$$a^{-1}(x) = \{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\} \tag{3.29}$$

Phép nhân $s'(x) = a^{-1}(x) \otimes s(x)$ có thể được biểu diễn dưới dạng ma trận:

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \text{ với } 0 \le c < Nb \tag{3.30}$$

Trong đoạn mã chương trình dưới đây, hàm `FFmul(x, y)` thực hiện phép nhân (trên trường GF(2^8)) hai phần tử x và y với nhau.

```
InvMixColumns(byte block[4,Nb])
begin
   byte t[4]
   for c = 0 to Nb - 1
      for r = 0 to 3
         t[r] = block[r,c]
      end for
      for r = 0 to 3
```

```
          block[r,c] =
              FFmul(0x0e, t[r])              xor
              FFmul(0x0b, t[(r + 1) mod 4])  xor
              FFmul(0x0d, t[(r + 2) mod 4])  xor
              FFmul(0x09, t[(r + 3) mod 4])
      end for
   end for
end
```

3.6.4 Quy trình giải mã tương đương

Nhận xét:

1. Phép biến đổi InvSubBytes thao tác trên giá trị của từng byte riêng biệt của trạng thái hiện hành, trong khi phép biến đổi InvShiftRows chỉ thực hiện thao tác di chuyển các byte mà không làm thay đổi giá trị của chúng. Do đó, thứ tự của hai phép biến đổi này trong quy trình mã hóa có thể được đảo ngược.

2. Với phép biến đổi tuyến tính A bất kỳ, ta có $A(x+k) = A(x) + A(k)$. Từ đó, suy ra

```
InvMixColumns(state XOR Round Key)=
       InvMixColumns(state) XOR InvMixColumns(Round Key)
```

Như vậy, thứ tự của phép biến đổi InvMixColumns và AddRoundKey trong quy trình giải mã có thể được đảo ngược với điều kiện mỗi từ (4 byte) trong bảng mã khóa mở rộng sử dụng trong giải mã phải được biến đổi bởi InvMixColumns. Do trong chu kỳ mã hóa cuối cùng không thực hiện thao tác MixColumns nên không

cần thực hiện thao tác InvMixColumns đối với mã khóa của chu kỳ giải mã đầu tiên cũng như chu kỳ giải mã cuối cùng.

Vậy, quy trình giải mã Rijndael có thể được thực hiện theo với trình tự các phép biến đổi ngược *hoàn toàn tương đương* với quy trình mã hóa.

```
EqInvCipher(byte in[4*Nb], byte out[4*Nb],
            word dw[Nb*(Nr+1)])
begin
   byte   state[4,Nb]
   state = in
   AddRoundKey(state, dw + Nr * Nb)
   for round = Nr - 1 downto 1
      InvSubBytes(state)
      InvShiftRows(state)
      InvMixColumns(state)
      AddRoundKey(state, dw + round * Nb)
   end for
   InvSubBytes(state)
   InvShiftRows(state)
   AddRoundKey(state, dw)
   out = state
end
```

Trong quy trình trên, bảng mã khóa mở rộng *dw* được xây dựng từ bảng mã khóa *w* bằng cách áp dụng phép biến đổi InvMixColumns lên từng từ (4 byte) trong *w*, ngoại trừ *Nb* từ đầu tiên và cuối cùng của *w*.

```
for i = 0 to (Nr + 1) * Nb - 1
    dw[i] = w[i]
end for
for rnd = 1 to Nr - 1
    InvMixColumns(dw + rnd * Nb)
end for
```

3.7 Các vấn đề cài đặt thuật toán

Gọi *a* là trạng thái khi bắt đầu chu kỳ mã hóa. Gọi *b*, *c*, *d*, *e* lần lượt là trạng thái kết quả đầu ra sau khi thực hiện các phép biến đổi SubBytes, ShiftRows, MixColumns và AddRoundKey trong chu kỳ đang xét. Quy ước: trong trạng thái *s* ($s = a, b, c, d, e$), cột thứ *j* được kí hiệu s_j, phần tử tại dòng *i* cột *j* kí hiệu là $s_{i,j}$.

Sau biến đổi SubBytes**:**

$$\begin{bmatrix} b_{0,j} \\ b_{1,j} \\ b_{2,j} \\ b_{3,j} \end{bmatrix} = \begin{bmatrix} S[a_{0,j}] \\ S[a_{1,j}] \\ S[a_{2,j}] \\ S[a_{3,j}] \end{bmatrix} \tag{3.31}$$

Sau biến đổi ShiftRows:

$$\begin{bmatrix} c_{0,j} \\ c_{1,j} \\ c_{2,j} \\ c_{3,j} \end{bmatrix} = \begin{bmatrix} b_{0,j} \\ b_{1,(j+shift(1,Nb))\bmod Nb} \\ b_{2,(j+shift(2,Nb))\bmod Nb} \\ b_{3,(j+shift(3,Nb))\bmod Nb} \end{bmatrix} \tag{3.32}$$

Sau biến đổi MixColumns**:**

$$\begin{bmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} c_{0,j} \\ c_{1,j} \\ c_{2,j} \\ c_{3,j} \end{bmatrix} \tag{3.33}$$

Sau biến đổi AddRoundKey**:**

$$\begin{bmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{bmatrix} = \begin{bmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{bmatrix} \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix} \tag{3.34}$$

Kết hợp các kết quả trung gian của mỗi phép biến đổi trong cùng chu kỳ với nhau, ta có:

$$\begin{bmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} S[a_{0,j}] \\ S\left[a_{1,(j+shift(1,Nb))\bmod Nb}\right] \\ S\left[a_{2,(j+shift(2,Nb))\bmod Nb}\right] \\ S\left[a_{3,(j+shift(3,Nb))\bmod Nb}\right] \end{bmatrix} \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix} \tag{3.35}$$

Ký hiệu $j[r] = (j + shift(r, Nb)) \bmod Nb$, biểu thức (3.35) có thể viết lại như sau:

$$\begin{bmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} S[a_{0,j[0]}] \\ S\left[a_{1,j[1]}\right] \\ S\left[a_{2,j[2]}\right] \\ S\left[a_{3,j[3]}\right] \end{bmatrix} \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix} \tag{3.36}$$

Khai triển phép nhân ma trận, ta có:

$$\begin{bmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{bmatrix} = S\left[a_{0,j[0]}\right] \begin{bmatrix} 02 \\ 01 \\ 01 \\ 03 \end{bmatrix} \oplus S\left[a_{1,j[1]}\right] \begin{bmatrix} 03 \\ 02 \\ 01 \\ 01 \end{bmatrix} \oplus S\left[a_{2,j[2]}\right] \begin{bmatrix} 01 \\ 03 \\ 02 \\ 01 \end{bmatrix} \oplus S\left[a_{3,j[3]}\right] \begin{bmatrix} 01 \\ 01 \\ 03 \\ 02 \end{bmatrix} \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix} \tag{3.37}$$

Định nghĩa các bảng tra cứu T_0, T_1, T_2, T_3 như sau:

$$T_0[a]=\begin{bmatrix} S[a]\bullet 02 \\ S[a] \\ S[a] \\ S[a]\bullet 03 \end{bmatrix}, \mathrm{T}_1[a]=\begin{bmatrix} \mathrm{S}[a]\bullet 03 \\ \mathrm{S}[a]\bullet 02 \\ \mathrm{S}[a] \\ \mathrm{S}[a] \end{bmatrix},$$

$$T_2[a]=\begin{bmatrix} S[a] \\ S[a]\bullet 03 \\ S[a]\bullet 02 \\ S[a] \end{bmatrix}, T_3[a]=\begin{bmatrix} S[a] \\ S[a] \\ S[a]\bullet 03 \\ S[a]\bullet 02 \end{bmatrix} \quad (3.38)$$

Khi đó, biểu thức (3.38) được viết lại như sau:

$$e_j=\left(\bigoplus_{i=0}^{3} T_i\left[a_{i,j[i]}\right]\right)\oplus w_{round*Nb+j} \quad (3.39)$$

với *round* là số thứ tự của chu kỳ đang xét.

Như vậy, mỗi cột e_j của trạng thái kết quả sau khi thực hiện một chu kỳ mã hóa có thể được xác định bằng bốn phép toán XOR trên các số nguyên 32 bit sử dụng bốn bảng tra cứu T_0, T_1, T_2 và T_3.

Công thức (3.39) chỉ áp dụng được cho *Nr*-1 chu kì đầu. Do chu kỳ cuối cùng không thực hiện phép biến đổi MixColumns nên cần xây dựng 4 bảng tra cứu riêng cho chu kì này:

$$U_0[a]=\begin{bmatrix} S[a] \\ 0 \\ 0 \\ 0 \end{bmatrix}, U_1[a]=\begin{bmatrix} 0 \\ S[a] \\ 0 \\ 0 \end{bmatrix}, U_2[a]=\begin{bmatrix} 0 \\ 0 \\ S[a] \\ 0 \end{bmatrix}, U_3[a]=\begin{bmatrix} 0 \\ 0 \\ 0 \\ S[a] \end{bmatrix} \quad (3.40)$$

3.7.1 Nhận xét

Kỹ thuật sử dụng bảng tra cứu giúp cải thiện tốc độ mã hóa và giải mã một cách đáng kể. Ngoài ra, kỹ thuật này còn giúp chống lại các phương pháp phá mã dựa trên thời gian mã hóa do khi sử dụng bảng tra cứu, thời gian mã hóa dữ liệu bất kỳ đều như nhau.

Kỹ thuật này có thể được sử dụng trong quy trình mã hóa và quy trình giải mã tương đương do sự tương ứng giữa các bước thực hiện của hai quy trình này. Khi đó, chúng ta có thể dùng chung một quy trình cho việc mã hóa và giải mã nhưng sử dụng bảng tra khác nhau.

Trên thực tế, các bảng tra cứu có thể được lưu trữ sẵn hoặc được xây dựng trực tiếp dựa trên bảng thay thế S-Box cùng với thông tin về các khuôn dạng tương ứng.

Trên các bộ vi xử lý 32-bit, những thao tác biến đổi sử dụng trong quy trình mã hóa có thể được tối ưu hóa bằng cách sử dụng bốn bảng tra cứu, mỗi bảng có 256 phần tử với kích thước mỗi phần tử là 4 byte. Với mỗi phần tử $a \in GF(2^8)$, đặt:

$$T_0[a] = \begin{bmatrix} S[a] \bullet 02 \\ S[a] \\ S[a] \\ S[a] \bullet 03 \end{bmatrix}, \; T_1[a] = \begin{bmatrix} S[a] \bullet 03 \\ S[a] \bullet 02 \\ S[a] \\ S[a] \end{bmatrix},$$

$$T_2[a] = \begin{bmatrix} S[a] \\ S[a] \bullet 03 \\ S[a] \bullet 02 \\ S[a] \end{bmatrix}, \; T_3[a] = \begin{bmatrix} S[a] \\ S[a] \\ S[a] \bullet 03 \\ S[a] \bullet 02 \end{bmatrix} \tag{3.41}$$

Nhận xét: $T_i[a]$ = `RotWord`($T_{i-1}[a]$) với $i = 1, 2, 3$. Ký hiệu RotWord^i là hàm xử lý gồm i lần thực hiện hàm RotWord, ta có:

$$T_i[a] = \text{RotWord}^i(T_0[a]) \tag{3.42}$$

Như vậy, thay vì dùng 4 kilobyte để lưu trữ sẵn cả bốn bảng, chỉ cần tốn 1 kilobyte để lưu bảng đầu tiên, các bảng còn lại có thể được phát sinh lại khi sử dụng. Các hạn chế về bộ nhớ thường không được đặt ra, trừ một số ít trường hợp như đối với các applet hay servlet. Khi đó, thay vì lưu trữ sẵn bảng tra cứu, chỉ cần lưu đoạn mã xử lý phát sinh lại các bảng này. Lúc đó, công thức (3.39) sẽ trở thành:

$$e_j = k_j \bigoplus_{i=0}^{3} T_i[a_{i,j[i]}] = k_j \bigoplus_{i=0}^{3} \text{RotWord}^i\left(T_0[a_{i,j[i]}]\right) \tag{3.43}$$

3.8 Kết quả thử nghiệm

Bảng 3.2. *Tốc độ xử lý của phương pháp Rijndael*

		Tốc độ xử lý (Mbit/giây)							
Kích thước (bit)		Pentium 200 MHz		Pentium II 400 MHz		Pentium III 733 MHz		Pentium IV 2.4 GHz	
Khóa	Khối	C++	C	C++	C	C++	C	C++	C
128	128	69.4	70.5	138.0	141.5	252.9	259.2	863.0	884.7
192	128	58.0	59.8	116.2	119.7	212.9	219.3	726.5	748.3
256	128	50.1	51.3	101.2	101.5	185.5	186.1	633.5	634.9

Kết quả thử nghiệm thuật toán Rijndael được ghi nhận trên máy Pentium 200 MHz (sử dụng hệ điều hành Microsoft Windows 98), máy Pentium II 400 MHz, Pentium III 733 MHz (sử dụng hệ điều hành Microsoft Windows 2000 Professional), Pentium IV 2,4GHz (sử dụng hệ điều hành Microsoft Windows XP Service Pack 2).

3.9 Kết luận

3.9.1 Khả năng an toàn

Việc sử dụng các hằng số khác nhau ứng với mỗi chu kỳ giúp hạn chế khả năng tính đối xứng trong thuật toán. Sự khác nhau trong cấu trúc của việc mã hóa và giải mã đã hạn chế được các khóa "yếu" (weak key) như trong phương pháp DES (xem phần 4.5.1). Ngoài ra, thông thường những điểm yếu liên quan đến mã khóa đều xuất phát từ sự phụ thuộc vào giá trị cụ thể của mã khóa của các thao tác phi tuyến như trong phương pháp IDEA (International Data Encryption Algorithm). Trong các phiên bản mở rộng, các khóa được sử dụng thông qua thao tác XOR và tất cả những thao tác phi tuyến đều được cố định sẵn trong S-box mà không phụ thuộc vào giá trị cụ thể của mã khóa (xem phần 4.5.4). Tính chất phi tuyến cùng khả năng khuếch tán thông tin (diffusion) trong việc tạo bảng mã khóa mở rộng làm cho việc phân tích mật mã dựa vào các khóa tương đương hay các khóa có liên quan trở nên không khả thi (xem phần 4.5.5). Đối với phương pháp vi phân rút gọn, việc phân tích chủ yếu khai thác đặc tính tập trung thành vùng (cluster) của các vết vi phân trong một số phương pháp mã hóa. Trong trường hợp thuật toán Rijndael với số lượng chu kỳ lớn hơn 6, không tồn tại phương pháp công phá mật mã nào hiệu quả hơn phương pháp thử và sai (xem phần 4.5.2). Tính chất phức tạp của biểu thức S-box trên GF(2^8) cùng với hiệu ứng khuếch tán giúp cho thuật toán không thể bị phân tích bằng phương pháp nội suy (xem phần 4.5.3).

3.9.2 Đánh giá

Phương pháp Rijndael thích hợp cho việc triển khai trên nhiều hệ thống khác nhau, không chỉ trên các máy tính cá nhân mà điển hình là sử dụng các chip Pentium, mà cả trên các hệ thống thẻ thông minh. Trên các máy tính cá nhân, thuật toán AES thực hiện việc xử lý rất nhanh so với các phương pháp mã hóa khác. Trên các hệ thống thẻ thông minh, phương pháp này càng phát huy ưu điểm không chỉ nhờ vào tốc độ xử lý cao mà còn nhờ vào mã chương trình ngắn gọn, thao tác xử lý sử dụng ít bộ nhớ. Ngoài ra, tất cả các bước xử lý của việc mã hóa và giải mã đều được thiết kế thích hợp với cơ chế xử lý song song nên phương pháp Rijndael càng chứng tỏ thế mạnh của mình trên các hệ thống thiết bị mới. Do đặc tính của việc xử lý thao tác trên từng byte dữ liệu nên không có sự khác biệt nào được đặt ra khi triển khai trên hệ thống big-endian hay little-endian.

Xuyên suốt phương pháp AES, yêu cầu đơn giản trong việc thiết kế cùng tính linh hoạt trong xử lý luôn được đặt ra và đã được đáp ứng. Độ lớn của khối dữ liệu cũng như của mã khóa chính có thể tùy biến linh hoạt từ 128 đến 256-bit với điều kiện là chia hết cho 32. Số lượng chu kỳ có thể được thay đổi tùy thuộc vào yêu cầu riêng được đặt ra cho từng ứng dụng và hệ thống cụ thể.

Tuy nhiên, vẫn tồn tại một số hạn chế mà hầu hết liên quan đến quá trình giải mã. Mã chương trình cũng như thời gian xử lý của việc giải mã tương đối lớn hơn việc mã hóa, mặc dù thời gian này vẫn nhanh hơn đáng kể so với một số phương pháp khác. Khi cài đặt bằng chương trình, do quá trình mã hóa và giải mã không giống nhau nên không thể tận dụng lại toàn bộ đoạn chương trình mã hóa cũng như các bảng tra cứu cho việc giải mã. Khi cài đặt trên phần cứng, việc giải mã

chỉ sử dụng lại một phần các mạch điện tử sử dụng trong việc mã hóa và với trình tự sử dụng khác nhau.

Phương pháp Rijndael với mức độ an toàn rất cao cùng các ưu điểm đáng chú ý khác chắc chắn sẽ nhanh chóng được áp dụng rộng rãi trong nhiều ứng dụng trên các hệ thống khác nhau.

Chương 4
Phương pháp Rijndael mở rộng

Trong chương 3, chúng ta đã tìm hiểu về phương pháp mã hóa Rijndael. Nội dung của chương 4 sẽ trình bày một số phiên bản mở rộng của chuẩn mã hóa Rijndael. Một số kết quả thử nghiệm cùng với phần phân tích và chứng minh khả năng an toàn của phương pháp Rijndael và các phiên bản mở rộng này cũng được trình bày trong chương 4.

4.1 Nhu cầu mở rộng phương pháp mã hóa Rijndael

Vào thập niên 1970-1980, phương pháp DES vốn được xem là rất an toàn và chưa thể công phá bằng các công nghệ thời bấy giờ. Tuy nhiên, hiện nay phương pháp này có thể bị phá vỡ và trở nên không còn đủ an toàn để bảo vệ các thông tin quan trọng. Đây chính là một trong những lý do mà NIST quyết định chọn một thuật toán mã hóa mới để thay thế DES nhằm phục vụ nhu cầu bảo mật thông tin của Chính phủ Hoa Kỳ cũng như trong một số ứng dụng dân sự khác. Phương pháp mã hóa Rijndael được đánh giá có độ an toàn rất cao và phương pháp vét cạn vẫn là cách hiệu quả nhất để công phá thuật toán này. Với khả năng

hiện nay của các hệ thống máy tính trên Thế giới thì giải pháp vét cạn vẫn là không khả thi. Tuy nhiên, với sự phát triển ngày càng nhanh của công nghệ thông tin, các thế hệ máy tính mới ra đời với năng lực và tốc độ xử lý ngày càng cao, thuật toán Rijndael sẽ có thể bị công phá trong tương lai. Khi đó, những thông tin quan trọng vốn đã được bảo mật bằng phương pháp Rijndael cần phải được mã hóa lại bằng một phương pháp mã hóa mới an toàn hơn. Vấn đề tái tổ chức dữ liệu quan trọng được tích lũy sau nhiều thập niên là hoàn toàn không đơn giản. Điều này đã dẫn đến yêu cầu mở rộng để nâng cao độ an toàn của thuật toán, chẳng hạn như tăng kích thước khóa và kích thước khối được xử lý. Các phiên bản mở rộng 256/384/512-bit và phiên bản mở rộng 512/768/1024-bit của thuật toán Rijndael được trình bày dưới đây được chúng tôi xây dựng trên cùng cơ sở lý thuyết của thuật toán nguyên thủy và có khả năng xử lý các khóa và khối dữ liệu lớn hơn nhiều lần so với phiên bản gốc.

4.2 Phiên bản mở rộng 256/384/512-bit

Trong thuật toán mở rộng 256/384/512-bit của phương pháp Rijndael, mỗi từ gồm có *Nw*=8 byte. Mỗi trạng thái có thể được biểu diễn dưới dạng một ma trận gồm 8 dòng và *Nb* cột với *Nb* bằng với độ dài của khối chia cho 64. Khóa chính cũng được biểu diễn dưới dạng một ma trận gồm 8 dòng và *Nk* cột với *Nk* bằng với độ dài của khóa chia cho 64. Ma trận biểu diễn 1 trạng thái hay khóa có thể được khảo sát dưới dạng mảng 1 chiều các từ (Nw byte), mỗi phần tử tương ứng với 1 cột của ma trận.

Số lượng chu kỳ, ký hiệu là *Nr*, có giá trị là

$$Nr = \max\{Nb, Nk\} + 6 \qquad (4.1)$$

4.2.1 Quy trình mã hóa

Trong quy trình mã hóa vẫn sử dụng 4 phép biến đổi chính như đã trình bày trong thuật toán mã hóa Rijndael cơ bản:

1. AddRoundKey: cộng ($\oplus$) mã khóa của chu kỳ vào trạng thái hiện hành. Độ dài của mã khóa của chu kỳ bằng với kích thước của trạng thái.

2. SubBytes: thay thế phi tuyến mỗi byte trong trạng thái hiện hành thông qua bảng thay thế (S-box).

3. MixColumns: trộn thông tin của từng cột trong trạng thái hiện hành. Mỗi cột được xử lý độc lập.

4. ShiftRows: dịch chuyển xoay vòng từng dòng của trạng thái hiện hành với di số khác nhau.

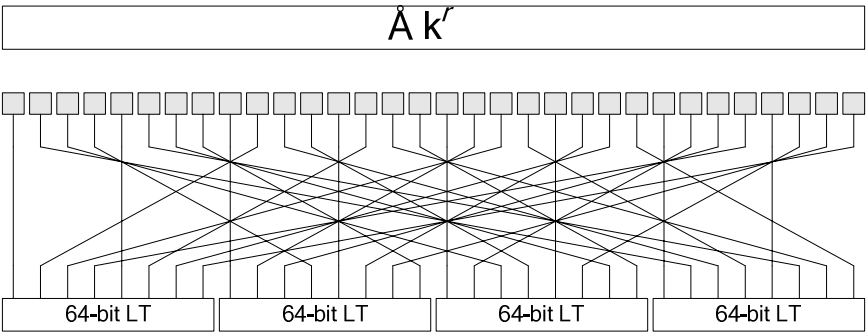
Mỗi phép biến đổi thao tác trên trạng thái hiện hành *S*. Kết quả *S'* của mỗi phép biến đổi sẽ trở thành đầu vào của phép biến đổi kế tiếp trong quy trình mã hóa.

Trước tiên, toàn bộ dữ liệu đầu vào được chép vào mảng trạng thái hiện hành. Sau khi thực hiện thao tác cộng mã khóa đầu tiên, mảng trạng thái sẽ được trải qua Nr = 10, 12 hay 14 chu kỳ biến đổi (tùy thuộc vào độ dài của mã khóa chính cũng như độ dài của khối được xử lý). $Nr-1$ chu kỳ đầu tiên là các chu kỳ biến đổi bình thường và hoàn toàn tương tự nhau, riêng chu kỳ biến đổi cuối cùng có sự khác biệt so với $Nr-1$ chu kỳ trước đó. Cuối cùng, nội dung của mảng trạng thái sẽ được chép lại vào mảng chứa dữ liệu đầu ra.

Hình 4.1 thể hiện kiến trúc của một chu kỳ biến đổi trong thuật toán Rijndael mở rộng 256/384/512-bit với *Nb* = 4.

Quy trình mã hóa Rijndael mở rộng được tóm tắt lại như sau:

1. Thực hiện `thao` tác AddRoundKey đầu tiên trước khi thực hiện các chu kỳ mã hóa.
2. `Nr`−1 chu kỳ mã hóa bình thường: mỗi chu kỳ bao gồm 4 bước biến đổi liên tiếp nhau: SubBytes, ShiftRows, MixColumns, và AddRoundKey.
3. Thực hiện `chu` kỳ mã hóa cuối cùng: trong chu kỳ này thao tác MixColumns được bỏ qua.



Hình 4.1. *Kiến trúc một chu kỳ biến đổi của thuật toán Rijndael mở rộng 256/384/512-bit với Nb = 4*

Trong thuật toán dưới đây, mảng `w[]` chứa bảng mã khóa mở rộng; mảng `in[]` và `out[]` lần lượt chứa dữ liệu vào và kết quả ra của thuật toán mã hóa.

```
Cipher(byte in[8 * Nb],
       byte out[8 * Nb],
       word w[Nb * (Nr + 1)])
begin
   byte   state[8,Nb]
   state =   in
   AddRoundKey(state, w)                    // Xem phần 4.2.1.4
   for round = 1 to Nr - 1
       SubBytes(state)                      // Xem phần 4.2.1.1
       ShiftRows(state)                     // Xem phần 4.2.1.2
       MixColumns(state)                    // Xem phần 4.2.1.3
       AddRoundKey(state, w + round * Nb)
   end for
   SubBytes(state)
   ShiftRows(state)
   AddRoundKey(state, w + Nr * Nb)
   out = state
end
```

4.2.1.1 Phép biến đổi SubBytes

Thao tác biến đổi SubBytes là phép thay thế các byte phi tuyến và tác động một cách độc lập lên từng byte trong trạng thái hiện hành. Bảng thay thế (`S-box`) có tính khả nghịch và quá trình thay thế 1 byte x dựa vào `S-box` bao gồm hai bước:

1. Xác định phần tử nghịch đảo $x^{-1} \in GF(2^8)$. Quy ước $\{00\}^{-1} = \{00\}$

2. Áp dụng phép biến đổi affine (trên GF(2)) đối với x^{-1} (giả sử x^{-1} có biểu diễn nhị phân là $\{x_7x_6x_5x_4x_3x_2x_1x_0\}$):

$$y_i = x_i \oplus x_{(i+4) \bmod 8} \oplus x_{(i+5) \bmod 8} \oplus x_{(i+6) \bmod 8} \oplus x_{(i+7) \bmod 8} \oplus c_i \quad (4.2)$$

với c_i là bit thứ i của {63}, $0 \le i \le 7$.

Phép biến đổi SubBytes được thể hiện dưới dạng mã giả:

```
SubBytes(byte state[8,Nb])
begin
   for r = 0 to 7
      for c = 0 to Nb - 1
         state[r,c] = Sbox[state[r,c]]
      end for
   end for
end
```

Bảng D.2 thể hiện bảng thay thế nghịch đảo được sử dụng trong phép biến đổi SubBytes.

4.2.1.2 Phép biến đổi ShiftRows

Trong thao tác biến đổi ShiftRows, mỗi dòng của trạng thái hiện hành được dịch chuyển xoay vòng với độ dời khác nhau. Byte $S_{r,c}$ tại dòng r cột c sẽ dịch chuyển đến cột (c - *shift*(r, Nb)) mod Nb hay:

$$s'_{r,c} = s_{r,(c+shift(r,Nb)) \bmod Nb} \text{ với } 0 < r < 8 \text{ và } 0 \le c < Nb \quad (4.3)$$

với

$$shift(r, Nb) = r \bmod Nb \quad (4.4)$$

Phép biến đổi ShiftRows được thể hiện dưới dạng mã giả:

```
ShiftRows(byte state[8,Nb])
begin
    byte t[Nb]
    for r = 1 to 7
        for c = 0 to Nb - 1
            t[c] = state[r, (c + shift[r,Nb]) mod Nb]
        end for
        for c = 0 to Nb - 1
            state[r,c] = t[c]
        end for
    end for
end
```

4.2.1.3 Phép biến đổi MixColumns

Trong thao tác biến đổi MixColumns, mỗi cột của trạng thái hiện hành được biểu diễn dưới dạng đa thức $s(x)$ có các hệ số trên GF(2^8). Thực hiện phép nhân:

$$s'(x) = a(x) \otimes s(x) \text{ với } a(x) = \sum_{i=0}^{7} a_i x^i \ , \ a_i \in \text{GF}(2^8) \tag{4.5}$$

Đặt $$M_a = \begin{bmatrix} \alpha_0 & \alpha_7 & \alpha_6 & \alpha_5 & \alpha_4 & \alpha_3 & \alpha_2 & \alpha_1 \\ \alpha_1 & \alpha_0 & \alpha_7 & \alpha_6 & \alpha_5 & \alpha_4 & \alpha_3 & \alpha_2 \\ \alpha_2 & \alpha_1 & \alpha_0 & \alpha_7 & \alpha_6 & \alpha_5 & \alpha_4 & \alpha_3 \\ \alpha_3 & \alpha_2 & \alpha_1 & \alpha_0 & \alpha_7 & \alpha_6 & \alpha_5 & \alpha_4 \\ \alpha_4 & \alpha_3 & \alpha_2 & \alpha_1 & \alpha_0 & \alpha_7 & \alpha_6 & \alpha_5 \\ \alpha_5 & \alpha_4 & \alpha_3 & \alpha_2 & \alpha_1 & \alpha_0 & \alpha_7 & \alpha_6 \\ \alpha_6 & \alpha_5 & \alpha_4 & \alpha_3 & \alpha_2 & \alpha_1 & \alpha_0 & \alpha_7 \\ \alpha_7 & \alpha_6 & \alpha_5 & \alpha_4 & \alpha_3 & \alpha_2 & \alpha_1 & \alpha_0 \end{bmatrix} \tag{4.6}$$

Ta có:
$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \\ s'_{4,c} \\ s'_{5,c} \\ s'_{6,c} \\ s'_{7,c} \end{bmatrix} = M_a \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \\ s_{4,c} \\ s_{5,c} \\ s_{6,c} \\ s_{7,c} \end{bmatrix}, 0 \le c \le Nb \quad (4.7)$$

Chúng ta có nhiều khả năng chọn lựa đa thức $a(x)$ khác nhau mà vẫn đảm bảo tính hiệu quả và độ an toàn của thuật toán. Để đảm bảo các tính chất an toàn của mình, các hệ số của ma trận này phải thỏa các tính chất sau:

1. Khả nghịch.
2. Tuyến tính trên GF(2).
3. Các phần tử ma trận (các hệ số) có giá trị càng nhỏ càng tốt.
4. Khả năng chống lại các tấn công của thuật toán (xem 4.4 - Phân tích mật mã vi phân và phân tích mật mã tuyến tính)

Đoạn mã chương trình dưới đây thể hiện thao tác biến đổi MixColumns với đa thức được trình bày trong công thức (2.6). Trong đoạn chương trình này, hàm `FFmul(x,y)` thực hiện phép nhân (trên trường $GF(2^8)$) hai phần tử x và y với nhau.

```
MixColumns(byte state[8, Nb])
begin
   byte t[8]
   for c = 0 to Nb - 1
      for r = 0 to 7
         t[r] = state[r,c]
      end for
      for r = 0 to 7
         state[r,c] =
            FFmul(0x01, t[r]) xor
            FFmul(0x05, t[(r + 1) mod 8]) xor
            FFmul(0x03, t[(r + 2) mod 8]) xor
            FFmul(0x05, t[(r + 3) mod 8]) xor
            FFmul(0x04, t[(r + 4) mod 8]) xor
            FFmul(0x03, t[(r + 5) mod 8]) xor
            FFmul(0x02, t[(r + 6) mod 8]) xor
            FFmul(0x02, t[(r + 7) mod 8]) xor
      end for
   end for
end
```

4.2.1.4 Thao tác AddRoundKey

Mã khóa của chu kỳ được biểu diễn bằng 1 ma trận gồm 8 dòng và *Nb* cột. Mỗi cột của trạng thái hiện hành được XOR với cột tương ứng của mã khóa của chu kỳ đang xét:

$$[s'_{0,c}, s'_{1,c}, s'_{2,c}, s'_{3,c}, s'_{4,c}, s'_{5,c}, s'_{6,c}, s'_{7,c}] = [s_{0,c}, s_{1,c}, s_{2,c}, s_{3,c}, s_{4,c}, s_{5,c}, s_{6,c}, s_{7,c}] \oplus [w_{round*Nb+c}] \quad \text{với } 0 \le c < Nb, \qquad (4.8)$$

❖ *Nhận xét*: Thao tác biến đổi ngược của AddRoundKey cũng chính là thao tác AddRoundKey.

Trong đoạn chương trình dưới đây, hàm `xbyte(r, w)` thực hiện việc lấy byte thứ *r* trong từ *w*.

```
AddRoundKey(byte state[8,Nb], word rk[])
// rk = w + round * Nb
begin
    for c = 0 to Nb – 1
        for r = 0 to 7
            state[r,c] = state[r,c] xor xbyte(r, rk[c])
        end for
    end for
end
```

4.2.2 Phát sinh khóa của mỗi chu kỳ

Quy trình phát sinh khóa cho mỗi chu kỳ bao gồm hai giai đoạn:

1. Mở rộng khóa chính thành bảng mã khóa mở rộng,
2. Chọn khóa cho mỗi chu kỳ từ bảng mã khóa mở rộng.

4.2.2.1 Xây dựng bảng khóa mở rộng

Bảng khóa mở rộng là mảng 1 chiều chứa các từ (có độ dài 8 byte), được ký hiệu là $w[Nb*(Nr + 1)]$. Hàm phát sinh bảng khóa mở rộng phụ thuộc vào giá trị *Nk*, tức là phụ thuộc vào độ dài của mã khóa chính.

Hàm `SubWord(W)` thay thế (sử dụng `S-box`) từng byte thành phần của một từ (có độ dài 8 byte).

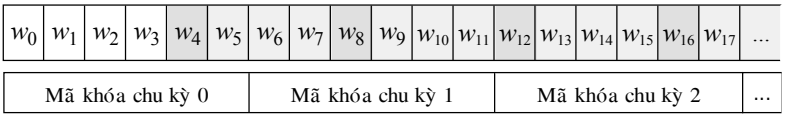
Hàm `RotWord(W)` thực hiện việc dịch chuyển xoay vòng 8 byte thành phần (b_0, b_1, b_2, b_3, b_4, b_5, b_6, b_7) của từ được đưa vào. Kết quả trả về của hàm `RotWord` là 1 từ gồm 8 byte thành phần là (b_1, b_2, b_3, b_4, b_5, b_6, b_7, b_0).

```
KeyExpansion(byte key[8 * Nk], word w[Nb * (Nr + 1)], Nk)
begin
    i = 0
    while (i < Nk)
        w[i]=word[key[8*i]   ,    key[8*i+1],
                  key[8*i+2], key[8*i+3],
                  key[8*i+4], key[8*i+5],
                  key[8*i+6], key[8*i+7]]
        i = i + 1
    end while
    i = Nk
    while (i < Nb * (Nr + 1))
        word temp = w[i - 1]
        if (i mod Nk = 0) then
            temp = SubWord(RotWord(temp)) xor Rcon[i / Nk]
        else
            if ((Nk = 8) and (i mod Nk = 4)) then
                temp = SubWord(temp)
            end if
        end if
        w[i] = w[i - Nk] xor temp
        i = i + 1
    end while
end
```

Các hằng số của mỗi chu kỳ hoàn toàn độc lập với giá trị *Nk* và được xác định bằng Rcon[i] = (x^{i-1}, 0, 0, 0, 0, 0, 0, 0), $i \geq 1$

4.2.2.2 Xác định khóa của chu kỳ

Mã khóa của chu kỳ thứ *i* được xác định bao gồm các từ (8 byte) có chỉ số từ $Nb*i$ đến $Nb*(i+1)-1$ của bảng mã khóa mở rộng. Như vậy, mã khóa của chu kỳ thứ *i* bao gồm các phần tử $w[Nb*i]$, $w[Nb*i+1]$, …, $w[Nb*(i+1)-1]$.



Hình 4.2. *Bảng mã khóa mở rộng và cách xác định mã khóa của chu kỳ (với Nb = 6 và Nk = 4)*

4.2.3 Quy trình giải mã

Quy trình giải mã được thực hiện qua các giai đoạn sau:

1. Thực hiện thao tác AddRoundKey đầu tiên trước khi thực hiện các chu kỳ giải mã.

2. *Nr* – 1 chu kỳ giải mã bình thường: mỗi chu kỳ bao gồm bốn bước biến đổi liên tiếp nhau: InvShiftRows, InvSubBytes, AddRoundKey, InvMixColumns.

3. Thực hiện chu kỳ giải mã cuối cùng. Trong chu kỳ này, thao tác InvMixColumns được bỏ qua.

```
InvCipher(byte in[8 * Nb],
          byte out[8 * Nb],
          word w[Nb * (Nr + 1)])
begin
   byte   state[8,Nb]
   state = in
   AddRoundKey(state, w + Nr * Nb)        // Xem phần 0
   for round = Nr - 1 downto 1
      InvShiftRows(state)                 // Xem phần 4.2.3.1
      InvSubBytes(state)                  // Xem phần 0
      AddRoundKey(state, w + round * Nb)
      InvMixColumns(state)                // Xem phần 0
   end for
   InvShiftRows(state)
   InvSubBytes(state)
   AddRoundKey(state, w)
   out = state
end
```

4.2.3.1 Phép biến đổi InvShiftRows

InvShiftRows là biến đổi ngược của biến đổi ShiftRows. Mỗi dòng của trạng thái được dịch chuyển xoay vòng theo chiều ngược với biến đổi ShiftRows với độ dời *Nb–shift* (*r*, *Nb*) khác nhau. Các byte ở cuối dòng được đưa vòng lên đầu dòng trong khi các byte còn lại có khuynh hướng di chuyển về cuối dòng.

$$s'_{r,(c+shift(r,Nb)) \bmod Nb} = s_{r,c} \quad \text{với } 0 < r < 8 \text{ và } 0 \le c < Nb \tag{4.9}$$

```
InvShiftRows(byte state[8,Nb])
begin
   byte t[Nb]
   for r = 1 to 7
      for c = 0 to Nb - 1
         t[(c + shift[r,Nb]) mod Nb] = state[r,c]
      end for
      for c = 0 to Nb - 1
         state[r,c] = t[c]
      end for
   end for
end
```

4.2.3.2 Phép biến đổi InvSubBytes

Phép biến đổi ngược của thao tác SubBytes, ký hiệu là InvSubBytes, sử dụng bảng thay thế nghịch đảo của S-box trên GF(2^8) được ký hiệu là S-box^{-1}. Quá trình thay thế 1 byte *y* dựa vào S-box^{-1} bao gồm hai bước sau:

1. Áp dụng phép biến đổi affine (trên GF(2)) sau đối với *y* (có biểu diễn nhị phân là $\{y_7 y_6 y_5 y_4 y_3 y_2 y_1 y_0\}$):

   $$x_i = y_{(i+2) \bmod 8} \oplus y_{(i+5) \bmod 8} \oplus y_{(i+7) \bmod 8} \oplus d_i,$$

   với d_i là bit thứ *i* của giá trị {05}, $0 \le i \le 7$. (4.10)

   Đây chính là phép biến đổi affine ngược của phép biến đổi affine ở bước 1 của S-box.

2. Gọi x là phần tử thuộc GF(2^8) có biểu diễn nhị phân là $\{x_7x_6x_5x_4x_3x_2x_1x_0\}$. Xác định phần tử nghịch đảo $x^{-1} \in$ GF(2^8) với quy ước {00}$^{-1}$ = {00}

Bảng D.2 thể hiện bảng thay thế nghịch đảo được sử dụng trong phép biến đổi InvSubBytes

```
InvSubBytes(byte state[8,Nb])
begin
    for r = 0 to 7
        for c = 0 to Nb - 1
            state[r,c] = InvSbox[state[r,c]]
        end for
    end for
end
```

4.2.3.3 Phép biến đổi InvMixColumns

InvMixColumns là biến đổi ngược của phép biến đổi MixColumns. Mỗi cột của trạng thái hiện hành được xem như đa thức $s(x)$ bậc 8 có các hệ số thuộc GF(2^8) và được nhân với đa thức $a^{-1}(x)$ là nghịch đảo của đa thức $a(x)$ (modulo $M(x) = x^8 + 1$) được sử dụng trong phép biến đổi MixColumns.

Với

$$a(x) = \{05\}x^7 + \{03\}x^6 + \{05\}x^5 + \{04\}x^4 + \{03\}x^3 + \{02\}x^2 + \{02\}x + \{01\} \quad (4.11)$$

ta có:

$$a^{-1}(x) = \{b3\}x^7 + \{39\}x^6 + \{9a\}x^5 + \{a1\}x^4 + \{db\}x^3 + \{54\}x^2 + \{46\}x + \{2a\} \quad (4.12)$$

Phép nhân $s'(x) = a^{-1}(x) \otimes s(x)$ được biểu diễn dưới dạng ma trận như sau:

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \\ s'_{4,c} \\ s'_{5,c} \\ s'_{6,c} \\ s'_{7,c} \end{bmatrix} = M_{a^{-1}} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \\ s_{4,c} \\ s_{5,c} \\ s_{6,c} \\ s_{7,c} \end{bmatrix}, 0 \le c \le Nb \qquad (4.13)$$

Đoạn chương trình sau thể hiện thao tác InvMixColumns sử dụng đa thức $a^{-1}(x)$ trong công thức (4.12).

```
InvMixColumns(byte block[8,Nb])
begin
   byte t[8]
   for c = 0 to Nb - 1
      for r = 0 to 7
         t[r] = block[r,c]
      end for
      for r = 0 to 7
         block[r,c] =
            FFmul(0x2a, t[r])             xor
            FFmul(0xb3, t[(r + 1) mod 8]) xor
            FFmul(0x39, t[(r + 2) mod 8]) xor
            FFmul(0x9a, t[(r + 3) mod 8]) xor
            FFmul(0xa1, t[(r + 4) mod 8]) xor
            FFmul(0xdb, t[(r + 5) mod 8]) xor
            FFmul(0x54, t[(r + 6) mod 8]) xor
```

```
            FFmul(0x46, t[(r + 7) mod 8])
        end for
    end for
end
```

4.2.4 Quy trình giải mã tương đương

Quy trình giải mã Rijndael có thể được thực hiện theo với trình tự các phép biến đổi ngược *hoàn toàn tương đương* với quy trình mã hóa (xem chứng minh trong phần 3.6.4-Quy trình giải mã tương đương).

```
EqInvCipher(byte in[8*Nb], byte out[8*Nb], word dw[Nb*(Nr + 1)])
begin
    byte   state[8,Nb]
    state = in
    AddRoundKey(state, dw + Nr * Nb)
    for round = Nr - 1 downto 1
        InvSubBytes(state)
        InvShiftRows(state)
        InvMixColumns(state)
        AddRoundKey(state, dw + round * Nb)
    end for
    InvSubBytes(state)
    InvShiftRows(state)
    AddRoundKey(state, dw)
    out = state
end
```

Bảng mã khóa mở rộng *dw* được xây dựng từ bảng mã khóa *w* bằng cách áp dụng phép biến đổi InvMixColumns lên từng từ (8 byte) trong *w*, ngoại trừ *Nb* từ đầu tiên và cuối cùng của *w*.

```
for i = 0 to (Nr + 1) * Nb – 1
    dw[i] = w[i]
end for
for rnd = 1 to Nr – 1
    InvMixColumns(dw + rnd * Nb)
end for
```

4.3 Phiên bản mở rộng 512/768/1024-bit

Thuật toán mở rộng 512/768/1024-bit dựa trên phương pháp Rijndael được xây dựng tương tự như thuật toán mở rộng 256/384/512-bit:

- Trong thuật toán 512/768/1024 bit, mỗi từ có kích thước *Nw*=16 byte.
- Đa thức được chọn trong thao tác MixColumns có bậc 15 và phải có hệ số Branch Number là 17. Chúng ta có thể chọn đa thức sau để minh họa:

$$a(x) = \{07\}x^{15} + \{09\}x^{14} + \{04\}x^{13} + \{09\}x^{12} + \{08\}x^{11} + \{03\}x^{10} + \{02\}x^{9} + \{08\}x^{8} + \{06\}x^{7} + \{04\}x^{6} + \{04\}x^{5} + \{01\}x^{4} + \{08\}x^{3} + \{03\}x^{2} + \{06\}x + \{05\} \quad (4.14)$$

Và đa thức nghịch đảo $a^{-1}(x)$ tương ứng là

$$a^{-1}(x) = \{1e\}x^{15} + \{bc\}x^{14} + \{55\}x^{13} + \{8d\}x^{12} + \{1a\}x^{11} + \{37\}x^{10} + \{97\}x^{9} + \{10\}x^{8} + \{f0\}x^{7} + \{d5\}x^{6} + \{01\}x^{5} + \{ad\}x^{4} + \{59\}x^{3} + \{82\}x^{2} + \{59\}x + \{3a\} \quad (4.15)$$

Chi tiết về thuật toán được trình bày trong [12], [16].

4.4 Phân tích mật mã vi phân và phân tích mật mã tuyến tính

4.4.1 Phân tích mật mã vi phân

Phương pháp phân tích mật mã vi phân (Differential Cryptanalysis) được Eli Biham và Adi Shamir trình bày trong [3].

Phương pháp vi phân chỉ có thể được áp dụng nếu có thể dự đoán được sự lan truyền những khác biệt trong các mẫu đầu vào qua hầu hết các chu kỳ biến đổi với số truyền (prop ratio [10]) lớn hơn đáng kể so với giá trị 2^{1-n} với n là độ dài khối (tính bằng bit).

Như vậy, để đảm bảo an toàn cho một phương pháp mã hóa, điều kiện cần thiết là không tồn tại vết vi phân (differential trail) lan truyền qua hầu hết các chu kỳ có số truyền lớn hơn đáng kể so với giá trị 2^{1-n}.

Đối với phương pháp Rijndael, các tác giả đã chứng minh không tồn tại vết vi phân lan truyền qua bốn chu kỳ có số truyền lớn hơn $2^{-30(Nb+1)}$ [8] với $Nb = n/Nw = n/32$. Như vậy, không tồn tại vết vi phân lan truyền qua tám chu kỳ có số truyền lớn hơn $2^{-60(Nb+1)}$. Điều này đủ để đảm bảo tính an toàn cho thuật toán Rijndael.

Phần chứng minh được trình bày trong 4.4.5-Trọng số vết vi phân và vết tuyến tính cho chúng ta các kết luận sau:

- Đối với thuật toán mở rộng 256/384/512-bit, không tồn tại vết vi phân lan truyền qua bốn chu kỳ có số truyền lớn hơn $2^{-54(Nb+1)}$ với $Nb = n/Nw = n/64$. Như vậy, không tồn tại vết vi phân lan truyền qua tám chu kỳ có số truyền lớn hơn $2^{-108(Nb+1)}$.
- Đối với thuật toán mở rộng 512/768/1024-bit, không tồn tại vết vi phân lan truyền qua bốn chu kỳ có số truyền lớn hơn $2^{-102(Nb+1)}$ với $Nb = n/Nw = n/128$. Như vậy, không tồn tại vết vi phân lan truyền qua tám chu kỳ có số truyền lớn hơn $2^{-204(Nb+1)}$.

Các kết luận trên đảm bảo tính an toàn cho thuật toán mở rộng 256/384/512 bit và 512/768/1024-bit đối với phương pháp phân tích mật mã vi phân.

4.4.2 Phân tích mật mã tuyến tính

Phương pháp phân tích mật mã tuyến tính (Linear Cryptanalysis) được Mitsuru Matsui trình bày trong [32].

Phương pháp tuyến tính chỉ có thể được áp dụng nếu sự tương quan giữa đầu ra với đầu vào của thuật toán qua hầu hết các chu kỳ có giá trị rất lớn so với $2^{-n/2}$.

Như vậy, để đảm bảo an toàn cho một phương pháp mã hóa, điều kiện cần thiết là không tồn tại vết tuyến tính (linear trail [10]) lan truyền qua hầu hết các chu kỳ có số truyền lớn hơn đáng kể so với giá trị $2^{-n/2}$.

Đối với phương pháp Rijndael, các tác giả đã chứng minh được rằng không tồn tại vết tuyến tính nào lan truyền qua bốn chu kỳ với độ tương quan lớn hơn $2^{-15(Nb+1)}$ [8]. Như vậy, không tồn tại vết tuyến tính nào lan truyền qua tám chu kỳ với độ tương quan lớn hơn $2^{-39(Nb+1)}$. Điều này đủ để đảm bảo tính an toàn cho thuật toán Rijndael.

Phần chứng minh được trình bày trong 4.4.4-Sự lan truyền mẫu cho chúng ta các kết luận sau:

- Đối với thuật toán mở rộng 256/384/512-bit, không tồn tại vết tuyến tính lan truyền qua bốn chu kỳ với độ tương quan lớn hơn $2^{-27(Nb+1)}$. Như vậy, không tồn tại vết tuyến tính nào lan truyền qua tám chu kỳ với độ tương quan lớn hơn $2^{-54(Nb+1)}$.
- Đối với thuật toán mở rộng 512/768/1024-bit, không tồn tại vết tuyến tính lan truyền qua bốn chu kỳ với độ tương quan lớn hơn $2^{-51(Nb+1)}$. Như vậy, không tồn tại vết tuyến tính nào lan truyền qua tám chu kỳ với độ tương quan lớn hơn $2^{-102(Nb+1)}$.

Các kết luận trên đảm bảo tính an toàn cho thuật toán mở rộng 256/384/512 bit và 512/768/1024-bit đối với phương pháp phân tích mật mã tuyến tính.

4.4.3 Branch Number

Xét phép biến đổi tuyến tính *F* trên vector các byte. Một byte khác 0 được gọi là *byte hoạt động* (active). Trọng số byte của một vector *a*, ký hiệu là *W*(*a*), là số lượng byte hoạt động trong vector này.

Định nghĩa 4.1: *Branch Number B của phép biến đổi tuyến tính F là độ đo khả năng khuếch tán của F, được định nghĩa như sau:*

$$B(F) = \min_{a \neq 0} (W(a) + W(F(a))) \tag{4.16}$$

- *Nhận xét***:** Branch Number càng lớn thì khả năng khuếch tán thông tin của *F* càng mạnh, giúp cho hệ thống SPN càng trở nên an toàn hơn.

Trong phép biến đổi MixColumns, nếu trạng thái ban đầu có 1 byte hoạt động thì trạng thái kết quả nhận được sau khi áp dụng MixColumns có tối đa *Nw* byte hoạt động. Do đó, ta có:

$$B(\text{MixColumns}) \leq Nw + 1 \tag{4.17}$$

với *Nw* lần lượt nhận giá trị là 4, 8 và 16 trong thuật toán Rijndael, thuật toán mở rộng 256/384/512 bit và thuật toán mở rộng 512/768/1024 bit.

Như vậy, để đạt được mức độ khuếch tán thông tin cao nhất, chúng ta cần phải chọn phép biến đổi MixColumns sao cho hệ số Branch Number đạt được giá trị cực đại là $Nw + 1$. Nói cách khác, Branch Number của MixColumns trong thuật toán Rijndael, thuật toán mở rộng 256/384/512 bit và thuật toán mở rộng 512/768/1024 bit phải đạt được giá trị lần lượt là 5, 9 và 17. Khi đó, quan hệ tuyến tính giữa các bit trong trạng thái đầu vào và đầu ra của MixColumns liên quan đến các $Nw + 1$ byte khác nhau trên cùng một cột.

4.4.4 Sự lan truyền mẫu

Trong phương pháp vi phân, số lượng S-box hoạt động được xác định bằng số lượng byte khác 0 trong trạng thái đầu vào của chu kỳ. Gọi *mẫu (vi phân) hoạt động* (difference activity pattern) là mẫu xác định vị trí các byte khác 0 trong trạng thái và gọi *trọng số byte* là số lượng byte khác 0 trong mẫu.

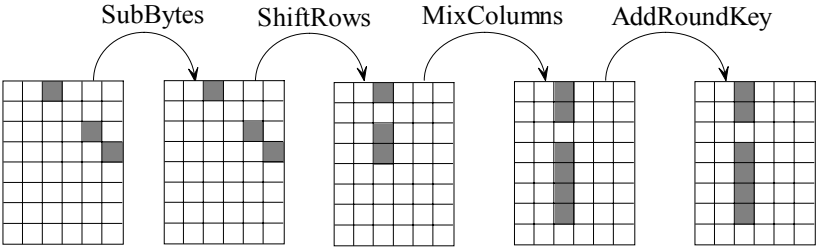
Trong phương pháp tuyến tính, số lượng S-box hoạt động được xác định bằng số lượng byte khác 0 trong các vector được chọn ở trạng thái bắt đầu của chu kỳ [10]. Gọi *mẫu (tương quan) hoạt động* (correlation activity pattern) là mẫu xác định vị trí các byte khác 0 trong trạng thái và gọi *trọng số byte* là số lượng byte khác 0 trong mẫu.

Mỗi cột trong trạng thái có ít nhất một byte thành phần là byte hoạt động được gọi *cột hoạt động*. *Trọng số cột* của trạng thái a, ký hiệu là $W_c(a)$, được định nghĩa là số lượng cột hoạt động trong mẫu. *Trọng số byte* của cột j của trạng thái a , ký hiệu là $W(a)\big|_j$, được định nghĩa là số lượng byte hoạt động trong cột này.

Trọng số của một vết lan truyền qua các chu kỳ được tính bằng tổng tất cả các trọng số của các mẫu hoạt động ở đầu vào của mỗi chu kỳ thành phần.

Trong các hình minh họa dưới đây, cột hoạt động được tô màu xám còn các byte hoạt động được tô màu đen.

Hình 4.3 minh họa sự lan truyền các mẫu hoạt động (bao gồm cả mẫu vi phân và mẫu tương quan) qua từng phép biến đổi trong các chu kỳ mã hóa của thuật toán mở rộng 256/384/512-bit của phương pháp Rijndael với *Nb* = 6.



Hình 4.3. *Sự lan truyền mẫu hoạt động qua từng phép biến đổi trong thuật toán mở rộng 256/384/512-bit của phương pháp Rijndael với Nb = 6*

Mỗi phép biến đổi thành phần trong phương pháp mã hóa Rijndael có tác động khác nhau đối với các mẫu hoạt động và các trọng số:

1. SubBytes và AddRoundKey không làm thay đổi các mẫu hoạt động cũng như giá trị trọng số cột và trọng số byte của mẫu.

2. ShiftRows làm thay đổi mẫu hoạt động và trọng số cột. Do phép biến đổi ShiftRows tác động lên từng byte của trạng thái một cách độc lập, không có sự tương tác giữa các byte thành phần trong trạng thái đang xét nên không làm thay đổi trọng số byte.

3. MixColumns làm thay đổi mẫu hoạt động và trọng số byte. Do phép biến đổi MixColumns tác động lên từng cột của trạng thái một cách độc lập, không có sự tương tác giữa các cột thành phần trong trạng thái đang xét nên không làm thay đổi trọng số cột.

Bảng 4.1 tóm tắt ảnh hưởng của các phép biến đổi lên mẫu hoạt động.

Bảng 4.1. *Ảnh hưởng của các phép biến đổi lên mẫu hoạt động*

		Sự ảnh hưởng		
STT	Phép biến đổi	Mẫu hoạt động	Trọng số cột	Trọng số byte
1	SubBytes	Không	Không	Không
2	ShiftRows	Có	Có	Không
3	MixColumns	Có	Không	Có
4	AddRoundKey	Không	Không	Không

Như vậy, phép biến đổi SubBytes và AddRoundKey không ảnh hưởng đến sự lan truyền các mẫu hoạt động trong vết nên chúng ta có thể bỏ qua các phép biến đổi này trong quá trình khảo sát các vết vi phân và vết tuyến tính dưới đây.

Trong phép biến đổi MixColumns, với mỗi cột hoạt động trong mẫu đầu vào (hoặc mẫu đầu ra) của một chu kỳ, tổng trọng số byte của cột này trong mẫu đầu vào và đầu ra bị chặn dưới bởi Branch Number.

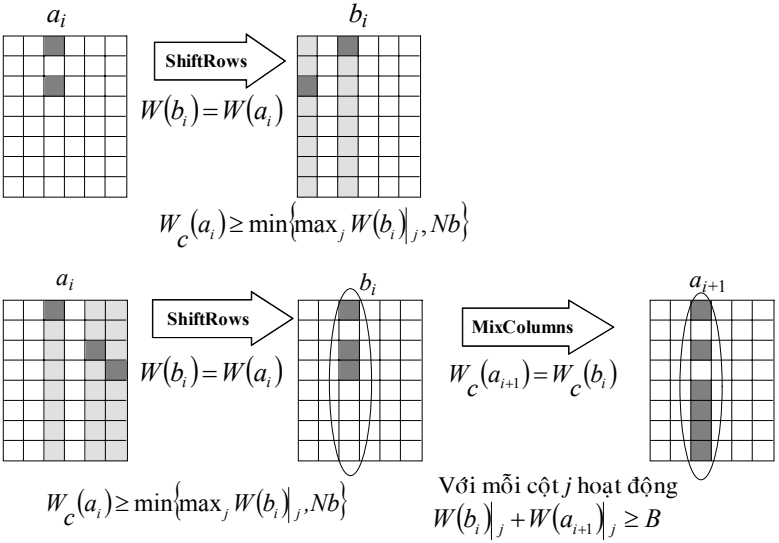
Do ShiftRows thực hiện việc dịch chuyển tất cả các byte thành phần trong một cột của mẫu đến các cột khác nhau nên phép biến đổi ShiftRows có các tính chất đặc biệt sau:

1. Trọng số cột của mẫu đầu ra bị chặn dưới bởi giá trị tối đa của trọng số byte của mỗi cột trong mẫu đầu vào.

2. Trọng số cột của mẫu đầu vào bị chặn dưới bởi giá trị tối đa của trọng số byte của mỗi cột trong mẫu đầu ra.

Dĩ nhiên cũng cần lưu ý là trọng số cột của một mẫu bất kỳ bị chặn dưới bởi số lượng cột (*Nb*) có trong mẫu.

Trong phần dưới đây, mẫu hoạt động ở đầu vào của chu kỳ mã hóa được ký hiệu là $a_{i\text{-}1}$, mẫu hoạt động kết quả sau khi thực hiện phép biến đổi ShiftRows được ký hiệu là $b_{i\text{-}1}$, Các chu kỳ biến đổi được đánh số tăng dần bắt đầu từ 1. Như vậy, a_0 chính là mẫu hoạt động ở đầu vào của chu kỳ mã hóa đầu tiên. Dễ dàng nhận thấy rằng mẫu a_i và b_i có cùng trọng số byte, mẫu $b_{j\text{-}1}$ và a_j có cùng trọng số cột. Trọng số của một vết lan truyền qua m chu kỳ được xác định bằng tổng trọng số của các mẫu a_0, a_1, ..., $a_{m\text{-}1}$. Trong các hình minh họa dưới đây, cột hoạt động được tô màu xám còn các byte hoạt động được tô màu đen. Hình 4.4 minh họa sự lan truyền mẫu trong một chu kỳ của thuật toán 256/384/512-bit của phương pháp Rijndael.



Hình 4.4. *Sự lan truyền mẫu hoạt động (thuật toán mở rộng 256/384/512-bit)*

Định lý 4.1: *Trọng số của vết lan truyền qua hai chu kỳ có Q cột hoạt động ở đầu vào của chu kỳ 2 bị chặn dưới bởi B*Q với B là Branch Number của phép biến đổi MixColumns.*

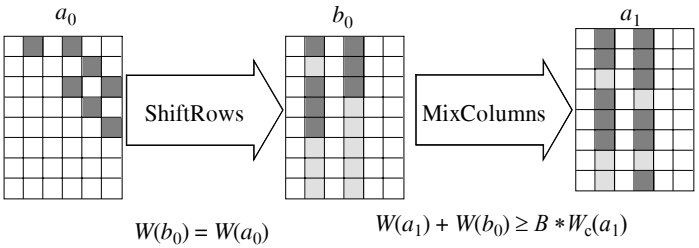
$$W_c(a_1) = Q \Rightarrow W(a_0) + W(a_1) \geq B * Q \qquad (4.18)$$

với $B = \text{BranchNumber}(\texttt{MixColumns})$

✍**Chứng minh**: Gọi B là Branch Number của phép biến đổi MixColumns. Tổng trọng số byte của mỗi cột tương ứng hoạt động trong mẫu b_0 và a_1 bị chặn dưới bởi B. Nếu trọng số cột của a_1 là Q thì tổng trọng số byte của b_0 và a_1 bị chặn dưới bởi $B*Q$. Do a_0 và b_0 có cùng trọng số byte nên tổng trọng số byte của a_0 và a_1 bị chặn dưới bởi $B*Q$.

Như vậy, bất kỳ một vết lan truyền qua hai chu kỳ đều có ít nhất $B*Q$ phần tử hoạt động.

Hình 4.5 minh họa Định lý 4.1 đối với thuật toán mở rộng 256/384/512-bit (Q=2)



Hình 4.5. *Minh họa Định lý 4.1 với Q = 2 (th-toán mở rộng 256/384/512-bit)*

Định lý 4.2: *Với mỗi vết lan truyền qua hai chu kỳ, tổng số cột hoạt động trong mẫu đầu vào và mẫu đầu ra tối thiểu là* $Nb+1$ *với Nb là số lượng cột trong trạng thái.*

$$W_c(a_0)+W_c(a_2)\geq Nb+1 \tag{4.19}$$

✍**Chứng minh**: Trong một vết bất kỳ tồn tại ít nhất một cột hoạt động trong mẫu a_1 (hoặc b_0). Gọi cột hoạt động này là cột *g*. Gọi *B* là Branch Number của phép biến đổi MixColumns. Tổng trọng số byte của cột *g* trong mẫu b_0 và mẫu a_1 bị chặn dưới bởi *B*.

$$W(b_0)\big|_g+W(a_1)\big|_g\geq B \tag{4.20}$$

Phép biến đổi ShiftRows di chuyển tất cả các byte thành phần trong một cột bất kỳ thuộc a_i đến các cột khác nhau thuộc b_i và ngược lại, mỗi cột thuộc b_i lại chứa các byte thành phần của các cột khác nhau thuộc a_i. Trọng số cột hay số lượng cột hoạt động của a_i bị chặn dưới bởi trọng số byte của mỗi cột thuộc b_i và trọng số cột của b_i bị chặn dưới bởi trọng số byte của mỗi cột thuộc a_i. Dĩ nhiên là trọng số cột của a_i hay b_i đều bị chặn dưới bởi số lượng cột *Nb* của trạng thái.

$$W_c(a_i)\geq \min\left\{Nb,\max_j W(b_i)\big|_j\right\} \tag{4.21}$$

$$W_c(b_i)\geq \min\left\{Nb,\max_j W(a_i)\big|_j\right\} \tag{4.22}$$

$$\Rightarrow W_c(a_0)+W_c(b_1)\geq \min\left\{Nb,\max_j W(b_0)\big|_j\right\}+\min\left\{Nb,\max_j W(a_1)\big|_j\right\} \tag{4.23}$$

$$\Rightarrow W_c(a_0)+W_c(b_1)\geq \min\left\{Nb,W(b_0)\big|_g\right\}+\min\left\{Nb,W(a_1)\big|_g\right\} \tag{4.24}$$

1. Trường hợp 1: Nếu $W(b_0)\big|_g \geq Nb$ hay $W(a_1)\big|_g \geq Nb$ thì

$$W_c(a_0) + W_c(b_1) \geq Nb + 1 \tag{4.25}$$

2. Trường hợp 2: Nếu $W(b_0)\big|_g < Nb$ và $W(a_1)\big|_g < Nb$ thì

$$W_c(a_0) + W_c(b_1) \geq W(b_0)\big|_g + W(a_1)\big|_g \geq B \tag{4.26}$$

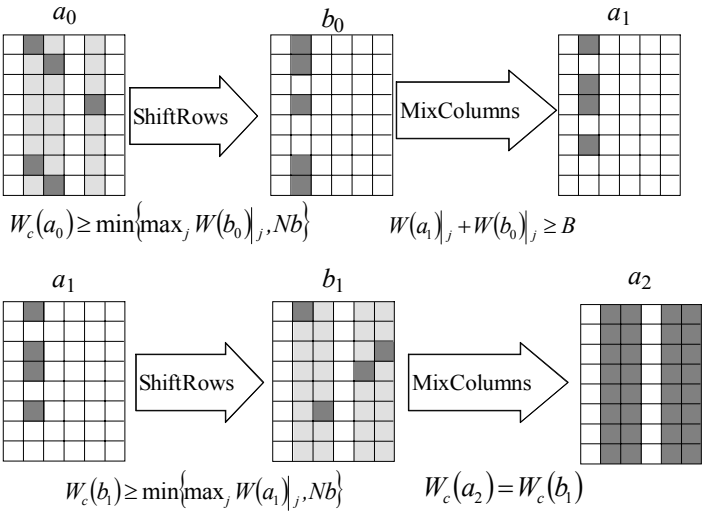
Do Nb chỉ nhận một trong ba giá trị 4, 6, hay 8 và B chỉ nhận một trong ba giá trị là 5, 9 hay 17 (tương ứng với thuật toán gốc, thuật toán mở rộng 256/384/512-bit hay 512/768/1024-bit). Vậy:

$$W_c(a_0) + W_c(b_1) \geq B \geq Nb + 1 \tag{4.27}$$

Do a_2 và b_1 có cùng trọng số cột nên suy ra

$$W_c(a_0) + W_c(b_2) \geq Nb + 1 \tag{4.28}$$

Hình 4.6 minh họa Định lý 4.2 đối với thuật toán mở rộng 256/384/512-bit.



Hình 4.6. *Minh họa Định lý 4.2 với* $W_c(a_1) = 1$

(thuật toán mở rộng 256/384/512-bit)

Định lý 4.3: *Mọi vết lan truyền qua 4 chu kỳ đều có tối thiểu* $B*(Nw+1)$ *byte hoạt động với B là Branch Number của phép biến đổi MixColumns.*

✍**Chứng minh**: Áp dụng Định lý 4.1 cho hai chu kỳ đầu (chu kỳ 1 và 2) và hai chu kỳ sau (chu kỳ 3 và 4), ta có:

$$\begin{cases} W(a_0)+W(a_1)\geq BW_c(a_1) \\ W(a_2)+W(a_3)\geq BW_c(a_3) \end{cases} \tag{4.29}$$

$$\Rightarrow \sum_{i=0}^{3} W(a_i)\geq B(W_c(a_1)+W_c(a_3)) \tag{4.30}$$

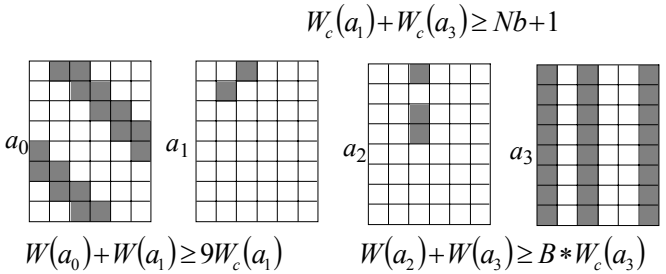
Như vậy, trọng số byte của vết bị chặn dưới bởi $B(W_c(a_1) + W_c(a_3))$

Theo Định lý 4.2, tổng trọng số cột của a_1 và a_3 bị chặn dưới bởi $Nb+1$.

$$W_c(a_1)+W_c(a_3)\geq Nb+1 \tag{4.31}$$

Vậy, trọng số byte của vết lan truyền qua bốn chu kỳ bị chặn bởi $B(Nb+1)$ hay vết lan truyền qua bốn chu kỳ có ít nhất $B(Nb+1)$ byte hoạt động.

Hình 4.7 minh họa Định lý 4.3 đối với thuật toán mở rộng 256/384/512-bit.



Hình 4.7. *Minh họa Định lý 4.3 (thuật toán mở rộng 256/384/512-bit)*

4.4.5 Trọng số vết vi phân và vết tuyến tính

Trong [10], J. Daemen đã chứng minh rằng:

1. Số truyền của vết vi phân có thể được xấp xỉ bằng tích số của các S-box hoạt động
2. Độ tương quan của vết tuyến tính có thể được xấp xỉ bằng tích số của độ tương quan giữa đầu ra-đầu vào của các S-box hoạt động.

Trong chiến lược thiết kế thuật toán Rijndael, S-box được chọn sao cho giá trị lớn nhất của số truyền và giá trị lớn nhất của độ tương quan càng nhỏ càng tốt. Bảng thay thế S-box được chọn có giá trị lớn nhất của số truyền và giá trị lớn nhất của độ tương quan lần lượt là 2^{-6} và 2^{-3}.

Ngoài ra, số lượng S-box hoạt động trong vết vi phân hay vết tuyến tính lan truyền qua bốn chu kỳ mã hóa của thuật toán nguyên thủy, phiên bản 256/384/512-bit và phiên bản 512/768/1024-bit lần lượt là 5(*Nb*+1), 9(*Nb*+1) và

17(*Nb*+1) với *Nb* là số cột trong một trạng thái (phần chứng minh được trình bày trong 4.4.4-Sự lan truyền mẫu). Như vậy, có thể kết luận rằng:

1. Mọi vết vi phân lan truyền qua bốn chu kỳ của thuật toán Rijndael có số truyền tối đa là $2^{-30(Nb+1)}$
2. Mọi vết vi phân lan truyền qua bốn chu kỳ của thuật toán mở rộng 256/384/512-bit có số truyền tối đa là $2^{-54(Nb+1)}$
3. Mọi vết vi phân lan truyền qua bốn chu kỳ của thuật toán mở rộng 512/768/1024-bit có số truyền tối đa là $2^{-102(Nb+1)}$.
4. Mọi vết tuyến tính lan truyền qua bốn chu kỳ của thuật toán Rijndael nguyên thủy có độ tương quan tối đa là $2^{-15(Nb+1)}$.
5. Mọi vết tuyến tính lan truyền qua bốn chu kỳ của thuật toán mở rộng 256/384/512-bit có độ tương quan tối đa là $2^{-27(Nb+1)}$.
6. Mọi vết tuyến tính lan truyền qua bốn chu kỳ của thuật toán mở rộng 512/768/1024-bit có độ tương quan tối đa là $2^{-51(Nb+1)}$.

4.5 Khảo sát tính an toàn đối với các phương pháp tấn công khác

4.5.1 Tính đối xứng và các khóa yếu của DES

Việc sử dụng các hằng số `Rcon` khác nhau cho mỗi chu kỳ giúp hạn chế tính đối xứng trong thuật toán. Sự khác nhau trong cấu trúc của việc mã hóa và giải mã đã hạn chế được các khóa "yếu" như trong phương pháp DES. Tính chất phi tuyến của quá trình phát sinh bảng mã khóa mở rộng giúp hạn chế các phương pháp phân tích dựa vào khóa tương đương.

4.5.2 Phương pháp tấn công Square

Phương pháp mã hóa Square được J. Daemen, L.R. Knudsen và V. Rijmen giới thiệu vào năm 1997 [9]. Trong bài viết này, các tác giả đã trình bày phương pháp tấn công đặc biệt đối với thuật toán mã hóa Square. Do phương pháp Rijndael kế thừa nhiều đặc tính của phương pháp Square nên phương pháp tấn công này cũng có thể được áp dụng đối với thuật toán Rijndael.

Trong [8], J. Daeman và V. Rijmen đã trình bày cách áp dụng phương pháp tấn công Square cho thuật toán Rijndael có tối đa 6 chu kỳ mã hóa. Đối với thuật toán Rijndael có dưới 6 chu kỳ mã hóa, phương pháp tấn công Square tỏ ra hiệu quả hơn phương pháp vét cạn để tìm mã khóa mặc dù với kỹ thuật hiện nay, phương pháp tấn công Square vẫn không thể thực hiện được. Với các thuật toán Rijndael có trên 6 chu kỳ mã hóa (có từ 7 chu kỳ mã hóa trở lên), phương pháp vét cạn để tìm mã khóa vẫn là phương pháp hiệu quả nhất.

4.5.3 Phương pháp nội suy

Phương pháp nội suy sử dụng trong phân tích mật mã áp dụng trên các thuật toán mã hóa theo khối được Jokobsen và Knudsen trình bày trong [28] vào năm 1997. Phương pháp này chỉ áp dụng được khi các thành phần sử dụng trong quy trình mã hóa có thể biểu diễn bằng các biểu thức đại số. Yêu cầu chính của phương pháp này là xây dựng được các đa thức (hay biểu thức chuẩn hóa) dựa vào các cặp dữ liệu trước và sau khi mã hóa. Nếu các đa thức này có bậc tương đối nhỏ thì chỉ cần sử dụng một vài cặp dữ liệu trước và sau khi mã hóa để xác định được các hệ số (độc lập với mã khóa) của đa thức này.

Bảng thay thế `S-box` có công thức trên GF(2^8) là:

$$S(x) = \{63\}+\{8f\}x^{127}+\{b5\}x^{191}+\{01\}x^{223}+\{f4\}x^{239}+ \{25\}x^{247}+\{f9\}x^{251}+\{09\}x^{253}+\{05\}x^{254} \quad (4.32)$$

Do tính chất phức tạp của biểu thức này cùng với hiệu ứng khuếch tán trong thuật toán nên không thể sử dụng phương pháp nội suy để tấn công phương pháp Rijndael.

4.5.4 Các khóa yếu trong IDEA

Trong một số phương pháp mã hóa, ví dụ như phương pháp IDEA (International Data Encryption Algorithm), việc chọn lựa mã khóa gặp phải một số hạn chế. Trong các phương pháp này, một số mã khóa dù hợp lệ nhưng khi sử dụng chúng để mã hóa dữ liệu sẽ dễ dàng bị phân tích và thông tin cần mã hóa sẽ không an toàn [10]. Thông thường những điểm yếu liên quan đến mã khóa đều xuất phát từ sự phụ thuộc vào giá trị cụ thể của mã khóa trong các thao tác phi tuyến. Trong phương pháp Rijndael cũng như các thuật toán mở rộng, các khóa được sử dụng thông qua thao tác XOR và tất cả những thao tác phi tuyến đều được cố định sẵn trong bảng thay thế S-box mà không phụ thuộc vào giá trị cụ thể của mã khóa nên không có bất kỳ một hạn chế nào trong việc chọn mã khóa chính.

4.5.5 Phương pháp tấn công khóa liên quan

Vào năm 1993, Eli Biham đã giới thiệu một phương pháp tấn công mật mã sử dụng các mã khóa liên quan [4]. Sau đó, phương pháp này được John Kelsey, Bruce Schneier và David Wagner nghiên cứu và áp dụng thử trên một số thuật toán mã hóa [30] vào năm 1996.

Trong phương pháp tấn công khóa liên quan, người phân tích thực hiện việc mã hóa sử dụng các khóa phân biệt có liên quan với nhau. Đối với phương pháp Rijndael cũng như các thuật toán mở rộng, tính chất phi tuyến cùng khả năng khuếch tán thông tin trong việc tạo bảng khóa mở rộng làm cho việc phân tích mật mã dựa vào các khóa liên quan trở nên không khả thi.

4.6 Kết quả thử nghiệm

Nhờ áp dụng kỹ thuật bảng tra cứu trong việc cài đặt các phiên bản mở rộng của thuật toán Rijndael nên thời gian thực hiện việc mã hóa và thời gian thực hiện việc giải mã là tương đương với nhau. Các thử nghiệm được tiến hành và ghi nhận trên máy Pentium 200 MHz (sử dụng hệ điều hành Microsoft Windows 98), máy Pentium II 400 MHz, Pentium III 733 MHz (sử dụng hệ điều hành Microsoft Windows 2000 Professional), Pentium IV 2.4GHz (sử dụng hệ điều hành Microsoft Windows XP Service Pack 2).

Bảng 4.2. *Tốc độ xử lý phiên bản 256/384/512-bit trên máy Pentium IV 2.4GHz*

Pentium IV 2.4 GHz		C++		C	
Khóa (bit)	Khối (bit)	#Nhịp	Tốc độ (Mbit/giây)	#Nhịp	Tốc độ (Mbit/giây)
256	256	1763	343.9	1721	353.3
384	256	2091	290.4	2052	297.8
512	256	2456	257.4	2396	263.1

Bảng 4.3. *Tốc độ xử lý phiên bản 512/768/1024-bit trên máy Pentium IV 2.4 GHz*

Pentium IV 2.4 GHz		C++		C	
Khóa (bit)	Khối (bit)	#Nhịp	Tốc độ (Mbit/giây)	#Nhịp	Tốc độ (Mbit/giây)
512	512	8360	153.4	8160	157.4
768	512	9910	130.1	9730	132.3
1024	512	11645	110.7	11364	113.7

Bảng 4.2 và Bảng 4.3 thể hiện tốc độ xử lý của phiên bản 256/384/512-bit và phiên bản 512/768/1024-bit trên máy Pentium IV 2.4 GHz. Kết quả được tính theo đơn vị Mbit/giây và đơn vị nhịp dao động.

Bảng 4.4. *Bảng so sánh tốc độ xử lý của phiên bản 256/384/512-bit*

		Tốc độ xử lý (Mbit/giây)							
Kích thước (bit)		Pentium 200 MHz		Pentium II 400 MHz		Pentium III 733 MHz		Pentium IV 2.4 GHz	
Khóa	Khối	C++	C	C++	C	C++	C	C++	C
256	256	26.9	27.4	55.0	56.4	100.8	103.4	343.9	353.3
384	256	22.7	23.3	46.4	47.5	85.0	87.1	290.4	297.8
512	256	19.5	20.2	41.1	42.0	75.3	76.9	257.4	263.1

Bảng 4.5. *Bảng so sánh tốc độ xử lý của phiên bản 512/768/1024-bit*

		Tốc độ xử lý (Mbit/giây)							
Kích thước (bit)		Pentium 200 MHz		Pentium II 400 MHz		Pentium III 733 MHz		Pentium IV 2.4 GHz	
Khóa	Khối	C++	C	C++	C	C++	C	C++	C
512	512	12.0	12.4	24.4	25.1	44.7	45.9	153.4	157.4
768	512	10.6	11.0	20.7	21.6	37.9	38.6	130.1	132.3
1024	512	8.9	9.2	17.6	18.1	32.3	33.1	110.7	113.7

Kết quả so sánh tốc độ xử lý trên máy Pentium 200 MHz (sử dụng hệ điều hành Microsoft Windows 98), máy Pentium II 400 MHz, Pentium III 733 MHz (sử dụng hệ điều hành Microsoft Windows 2000 Professional), Pentium IV 2.4GHz (sử dụng hệ điều hành Microsoft Windows XP Service Pack 2) của phiên bản 256/384/512-bit và phiên bản 512/768/1024-bit được thể hiện trong Bảng 4.4 và Bảng 4.5.

4.7 Kết luận

Đối với phiên bản nguyên thủy của thuật toán mã hóa Rijndael, phương pháp hiệu quả nhất để phân tích mật mã vẫn là phương pháp vét cạn để tìm ra mã khóa chính đã được sử dụng. Như vậy, nếu sử dụng mã khóa chính có 128/192/256 bit thì không gian mã khóa K cần khảo sát lần lượt có 2^{128}, 2^{192}, 2^{256} phần tử.

Một cách tương tự, đối với các phiên bản mở rộng của thuật toán Rijndael, phương pháp vét cạn để tìm ra mã khóa vẫn là phương pháp khả thi hơn so với các phương pháp khác.

Đối với phiên bản mở rộng 256/384/512-bit của thuật toán mã hóa Rijndael, không gian mã khóa K cần khảo sát có 2^{256}, 2^{384}, 2^{512} phần tử tùy thuộc vào độ dài của mã khóa chính được sử dụng là 256, 384 hay 512 bit.

Đối với phiên bản mở rộng 512/768/1024-bit của thuật toán mã hóa Rijndael, không gian mã khóa K cần khảo sát có 2^{512}, 2^{768}, 2^{1024} phần tử tùy thuộc vào độ dài của mã khóa chính được sử dụng là 512, 768 hay 1024 bit.

Dựa vào các số liệu thống kê trong Bảng 3.2, Bảng 4.4 và Bảng 4.5, chúng ta có thể nhận thấy rằng khi tăng gấp đôi kích thước khối được xử lý thì thời gian mã

hóa một khối dữ liệu tăng lên hơn 4 lần và do đó tốc độ mã hóa sẽ giảm đi hơn hai lần. Tuy nhiên, điều này hoàn toàn có thể chấp nhận được do việc tăng kích thước mã khóa và kích thước khối xử lý sẽ làm không gian mã khóa tăng lên đáng kể và thông tin được mã hóa sẽ càng an toàn hơn.

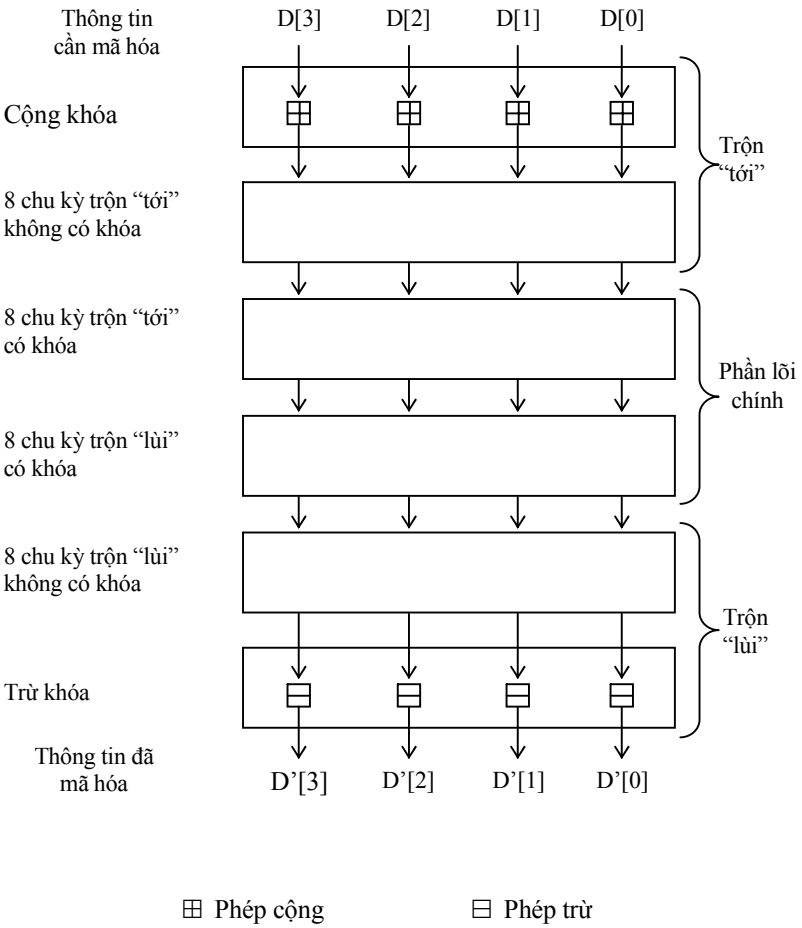
Chương 5
Các thuật toán ứng cử viên AES

Trong chương 3, chúng ta đã khảo sát phương pháp mã hóa Rijndael. Cùng với phương pháp này, còn có bốn phương pháp mã hóa khác được chọn vào vòng chung kết các ứng cử viên của chuẩn mã hóa AES, bao gồm phương pháp MARS, RC6, Serpent và TwoFish. Trong nội dung của chương này sẽ lần lượt giới thiệu về bốn phương pháp mã hóa ứng cử viên AES này.

5.1 Phương pháp mã hóa MARS

MARS là thuật toán mã hóa khóa đối xứng hỗ trợ kích thước khối dữ liệu 128 bit và cho phép sử dụng mã khóa có kích thước thay đổi được. Thuật toán được thiết kế trên cơ sở khai thác các thế mạnh của việc thực hiện các phép toán trên các thế hệ máy tính hiện nay nhằm tăng hiệu quả của thuật toán so với các thuật toán mã hóa quy ước trước đây.

5.1.1 Quy trình mã hóa



Hình 5.1. *Quy trình mã hóa MARS*

Hình 5.1 thể hiện mô hình chung của quy trình mã hóa MARS. Dữ liệu đầu vào và kết quả của quá trình mã hóa đều là từ có độ dài 32 bit. Tất cả các phép toán trong quy trình mã hóa và giải mã đều thực hiện trên các từ 32 bit. Trong trường hợp khảo sát dữ liệu mã hóa dưới dạng mảng gồm 4 byte, các tác giả quy ước sử dụng thứ tự lưu trữ little–endian.

5.1.2 S–box

Trong quá trình thiết kế S–box, các phần tử trong S-box được chọn sao cho S–box có các đặc tính tuyến tính và vi phân an toàn chống lại các phương pháp tấn công. Phụ lục A trình bày chi tiết nội dung của S-box được sử dụng trong thuật toán MARS.

Các `S-box` được phát sinh bằng cách cho $i = 0$ đến 102, $j = 0$ đến 4,

$$S[5i + j] = \text{SHA} - 1(5i \,|\, c1 \,|\, c2 \,|\, c3)_j \tag{5.1}$$

(ở đây SHA – $1(.)_j$ là từ thứ j trong kết quả của SHA – 1). Xem i như một số nguyên không dấu 32 bit và c_1, c_2, c_3 là các hằng số cố định. Trong khi thực hiện ta đặt $c_1 = 0\text{xb}7415162$, $c_2 = 0\text{x}283\text{f}6\text{a}88$ (là phần khai triển nhị phân của các phân số e, π tương ứng) và biến đổi c_3 cho đến khi tìm được một `S-box` có những đặc tính tốt. Xem SHA–1 như một phép toán trên các dòng byte và sử dụng quy ước little–endian để chuyển đổi giữa các từ và các byte.

`S-box` được xây dựng như sau: Đầu tiên biến đổi các giá trị có thể có của c_3 theo thứ tự tăng dần, bắt đầu với $c_3 = 0$. Đối với mỗi giá trị, phát sinh `S-box` và sau đó cố định nó bằng cách biến đổi toàn bộ các cặp (i, j) của các mục trong S0, S1

theo thứ tự từ điển và kiểm tra xem S[i] ⊕ S[j] có chênh lệch 2 hoặc nhiều byte zero. Bất kỳ lúc nào tìm được sự chênh lệch 2 hoặc nhiều byte zero thì thay thế S[i] với 3·S[i] và di chuyển đến i kế tiếp. Sau khi dừng lại, thử nghiệm `S-box` lại để kiểm tra xem nó có thỏa mãn hết các điều kiện 1–8 ở trên và tính single bit correlation (điều kiện 9). Giá trị của c_3 giảm single bit correlation là $c_3 = 0x02917d59$. S–box này có parity bias 2^{-7}, single bit bias đạt cao nhất là 1/30, Two consecutive bit bias đạt cao nhất 1/32 và single bit correlation bias nhỏ hơn 1/22.

5.1.3 Khởi tạo và phân bố khóa

Thủ tục Key–Expansion thực hiện việc mở rộng mảng khóa k[] bao gồm n từ 32 bit (với n là số bất kỳ trong khoảng từ 4 đến 14) thành một mảng K[] gồm 40 từ. Cần lưu ý là không cần có bất kỳ yêu cầu đặc biệt gì về cấu trúc của khóa gốc k[] (ví dụ như khóa không cần sử dụng các bit parity). Ngoài ra, thủ tục Key–Expansion cũng đảm bảo rằng mỗi từ trong khóa được sử dụng cho phép nhân trong thủ tục mã hóa có các đặc tính sau đây:

1. Hai bit thấp nhất của một từ trong khóa được sử dụng trong phép nhân có giá trị 1.

2. Không có từ nào trong khóa chứa liên tiếp 10 bit 0 hoặc 10 bit 1.

5.1.3.1 Thủ tục Key–Expansion

Thủ tục Key–Expansion bao gồm các bước sau:

1. Ban đầu, nội dung khóa gốc được chép vào một mảng tạm T[] (có độ dài là 15 từ), tiếp theo là số n và cuối cùng là các số 0. Nghĩa là:

$$T[0..n-1] = k[0..n-1], T[n] = n, T[n+1..14] = 0 \quad (5.2)$$

2. Sau đó, các bước dưới đây được thực hiện lặp lại bốn lần. Mỗi lần lặp sẽ tính giá trị của 10 từ kế tiếp trong khóa mở rộng:

   a) Mảng T[] được biến đổi sử dụng công thức tuyến tính sau:

   for i = 0 to 14

   $$T[i] = T[i] \oplus ((T[i-7 \bmod 15] \oplus T[i-2 \bmod 15]) <<< 3) \oplus (4i + j)$$

   với j là số thứ tự của lần lặp (j = 0, 1,…)

   b) Kế đến, mảng T[] sẽ được biến đổi qua bốn chu kỳ của mạng Feistel loại 1:

   $$T[i]=(T[i] + S[\text{9 bit thấp của } T[i-1 \bmod 15]]) <<< 9$$

   với i = 0, 1, …, 14.

   c) Sau đó, lấy 10 từ trong mảng T[], sắp xếp lại rồi đưa vào thành 10 từ kế tiếp của mảng khóa mở rộng K[].

   $$K[10j + i] = T[4i \bmod 15],\ i = 0,1,\ldots,9$$

   với j là số thứ tự của lần lặp, j = 0,1,…

3. Cuối cùng, xét 16 từ dùng cho phép nhân trong mã hóa (bao gồm các từ $K[5]$, $K[7]$, …, $K[35]$) và biến đổi chúng để có hai đặc tính nêu trên. Cần lưu ý là khả năng từ được chọn lựa ngẫu nhiên không thỏa đặc tính thứ hai (tức là từ có 10 bit liên tiếp bằng 0 hoặc bằng 1) là khoảng 1/41. Mỗi từ $K[5]$, $K[7]$, …, $K[35]$ được xử lý như sau:

   a) Ghi nhận hai bit thấp nhất của $K[i]$ bằng cách đặt $j = K[i] \wedge 3$. Sau đó, xây dựng từ w dựa trên $K[i]$ bằng cách thay thế hai bit thấp nhất của $K[i]$ bằng giá trị 1, tức là $w = K[i] \vee 3$.

   b) Xây dựng một mặt nạ M của các bit trong w thuộc một dãy gồm 10 (hoặc nhiều hơn) bit 0 hoặc 1 liên tiếp. Ta có $M_\ell = 1$ nếu và chỉ nếu w_ℓ thuộc một dãy 10 bit 0 hoặc 1 liên tục. Sau đó đặt lại 0 cho các bit 1 trong M tương ứng với điểm cuối của đường chạy các bit 0 hoặc 1 liên tục trong w, cũng làm như vậy đối với 2 bit thấp nhất và 1 bit cao nhất của M. Như vậy, bit thứ i của M được đặt lại giá trị 0 nếu $i < 2$, hoặc $i = 31$, hoặc nếu bit thứ i của w khác bit thứ $(i+1)$ hoặc bit thứ $(i-1)$.

      ❒ Ví dụ, giả sử ta có $w = 0^3 1^{13} 0^{12} 1011$ (ở đây 0^i, 1^i biểu diễn i bit 0 hoặc 1 liên tục). Trong trường hợp này, đầu tiên đặt $M = 0^3 1^{25} 0^4$, kế đến, gán lại giá trị 1 ở cho các bit ở vị trí 4, 15, 16 và 28 để có $M = 0^4 1^{11} 001^{10} 0^5$.

   c) Tiếp theo, sử dụng một bảng B (gồm bốn từ) cố định để "sửa w". Bốn phần tử trong B được chọn sao cho mỗi phần tử (cũng như các giá trị xoay chu kỳ khác được xây dựng từ phần tử này) không chứa bảy bit 0 hoặc mười bit 1 liên tiếp nhau. Cụ thể, các tác giả sử dụng bảng

B[] = {0xa4a8d57b, 0x5b5d193b, 0xc8a8309b, 0x73f9a978}, (đây là các phần tử thứ 265 đến 268 trong `S-box`). Lý do chọn các phần tử này là chỉ có 14 mẫu 8 bit xuất hiện hai lần trong các phần tử này và không có mẫu nào xuất hiện nhiều hơn hai lần.

Sử dụng hai bit j (ở bước (a)) để chọn một phần tử trong B và sử dụng năm bit thấp nhất của $K[i-1]$ để quay giá trị của phần tử được chọn này, tức là:

$p = B[j] <<<$ (5 bit thấp nhất của $K[i-1]$)

d) Cuối cùng, thực hiện XOR mẫu p với w sử dụng mặt nạ M và lưu kết quả trong $K[i]$.

$$K[i] = w \oplus (p \wedge M)$$

Do hai bit thấp nhất của M là 0 nên hai bit thấp nhất của K[i] sẽ là 1 (do những bit này trong w là 1). Ngoài ra, việc chọn giá trị của mảng B bảo đảm rằng $K[i]$ không chứa dãy mười bit 0 hoặc 1 liên tục.

Lưu ý rằng thủ tục này không chỉ đảm bảo rằng các từ $K[5]$, $K[7]$,…, $K[35]$ có hai đặc tính nêu trên mà còn giữ được tính chất "ngẫu nhiên" của các từ này, tức là không có bất kỳ một giá trị của từ đơn nào có xác suất lớn hơn trong sự phân bố đồng. Sử dụng phương pháp vét cạn, có thể kiểm chứng được rằng không có mẫu 20 bit nào xuất hiện trong các từ này với xác xuất lớn hơn 1.23×2^{-20}. Tương tự, không có mẫu 10 bit nào xuất hiện với xác suất lớn hơn 1.06×2^{-10}. Các yếu tố này được sử dụng trong việc phân tích thuật toán.

Dưới đây là mã giả cho thủ tục Key–Expansion

```
Key-Expansion(input: k[], n; output: K[])
// n là số lượng từ trong mảng khóa k[], (4 ≤ n ≤ 14)
// K[] là mảng chứa khóa mở rộng, bao gồm 40 từ
// T[] là mảng tạm, bao gồm 15 từ
// B[] là mảng cố định gồm 4 từ

// Khởi tạo mảng B[]
B[] = {0xa4a8d57b, 0x5b5d193b, 0xc8a8309b, 0x73f9a978}

// Khởi tạo mảng T với giá trị của mảng khóa k[]
T[0…n–1] = k[0…n–1], T[n] = n, T[n+1… 14] = 0

// Lặp 4 lần, mỗi lần tính giá trị 10 từ trong mảng K[]
for j = 0 to 3
    for i = 0 to 14                          // Biến đổi tuyến tính
        T[i] = T[i]⊕((T[i–7 mod 15] ⊕ T[i–2 mod 15]) <<< 3) ⊕ (4i+j)
    repeat 4 lần                             // 4 chu kỳ biến đổi
        for i = 0 to 14
            T[i] = (T[i] + S[9 bit thấp của T[i–1 mod 15]]) <<< 9
    end repeat
    for i = 0 to 9                           // Lưu kết quả vào 10 từ kế tiếp của K[]
        K[10j + i] = T[4i mod 15]
end for

// Sửa đổi các giá trị khóa sẽ sử dụng trong phép nhân
```

for $i = 5, 7, \ldots 35$

j = 2 bit thấp nhất của $K[i]$

$w = K[i]$ với 2 bit thấp nhất đặt lại là 1

// Phát sinh mặt nạ M

$M_\ell = 1$ khi vào chỉ khi w_ℓ thuộc về dãy 10 bit 0 hay 1 liên tiếp trong w

và $2 \le \ell \le 30$ và $w_{\ell-1} = w_\ell = w_{\ell+1}$

// Chọn 1 mẫu trong mảng B, quay giá trị phần tử được chọn

r = 5 bit thấp của $K[i-1]$ // số lượng bit quay

$p = B[j] <<< r$

// Thay đổi $K[i]$ sử dụng giá trị p và mặt nạ M

$K[i] = w \oplus (p \wedge M)$

end for

5.1.4 Quy trình mã hóa

Cấu trúc chung của việc mã hóa được mô tả trong Hình 5.1 gồm ba giai đoạn: trộn "tới" (Forward mixing), phần lõi chính (Cryptographic core) và trộn "lùi" (Backward mixing). Việc mã hóa chính nằm ở phần lõi bao gồm các phép biến đổi có khóa.

Một số ký hiệu sử dụng trong quy trình mã hóa:

1. *D*[] là một mảng bốn từ dữ liệu 32 bit. Ban đầu *D* chứa các từ của văn bản ban đầu (thông tin cần mã hóa). Khi kết thúc quá trình mã hóa, *D* chứa các từ của thông tin đã được mã hóa.

2. *K*[] là mảng khóa mở rộng, bao gồm 40 từ 32 bit.

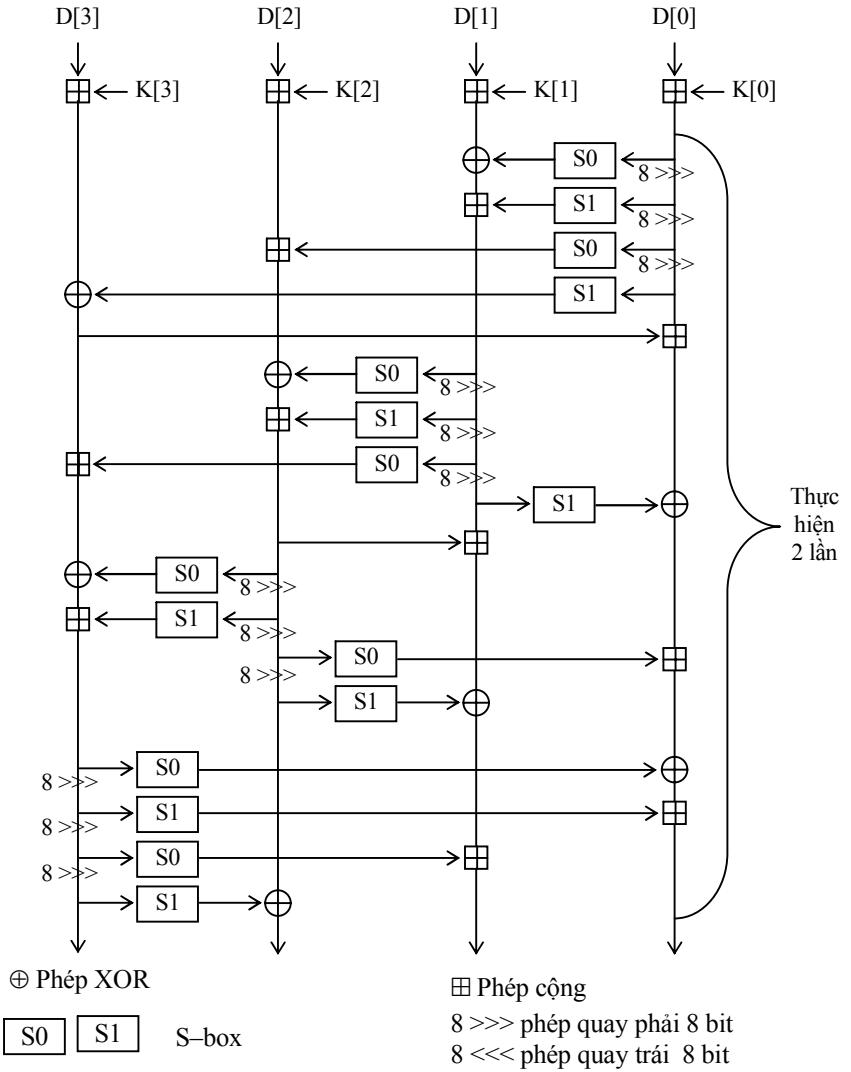
3. *S*[] là một S–box, bao gồm 512 từ 32 bit, được chia thành hai mảng: S0 gồm 256 từ đầu tiên trong S–box và S1 gồm 256 từ còn lại.

Tất cả các mảng sử dụng có chỉ số mảng bắt đầu từ 0.

5.1.4.1 Giai đoạn 1: Trộn "tới"

Nếu ký hiệu 4 byte của các từ nguồn bằng b_0, b_1, b_2, b_3 (ở đây b_0 là byte thấp nhất và b_3 là byte cao nhất), sau đó dùng b_0, b_2 làm chỉ số trong `S-box` S0 và b_1, b_3 làm chỉ số trong `S-box` S1. Đầu tiên XOR S0[b_0] với từ đích thứ nhất, sau đó cộng S1[b_1] cũng với từ đích thứ nhất. Kế đến cộng S0[b_2] với từ đích thứ hai và xor S1[b_3] với từ đích thứ 3. Cuối cùng, quay từ nguồn 24 bit về bên phải.

Đối với chu kỳ kế tiếp, quay bốn từ về bên phải một từ để từ đích thứ nhất hiện tại trở thành từ nguồn kế tiếp, từ đích thứ hai hiện tại trở thành từ đích thứ nhất tiếp theo, từ đích thứ ba hiện tại trở thành từ đích thứ hai tiếp theo và từ nguồn hiện tại trở thành từ đích thứ ba tiếp theo.



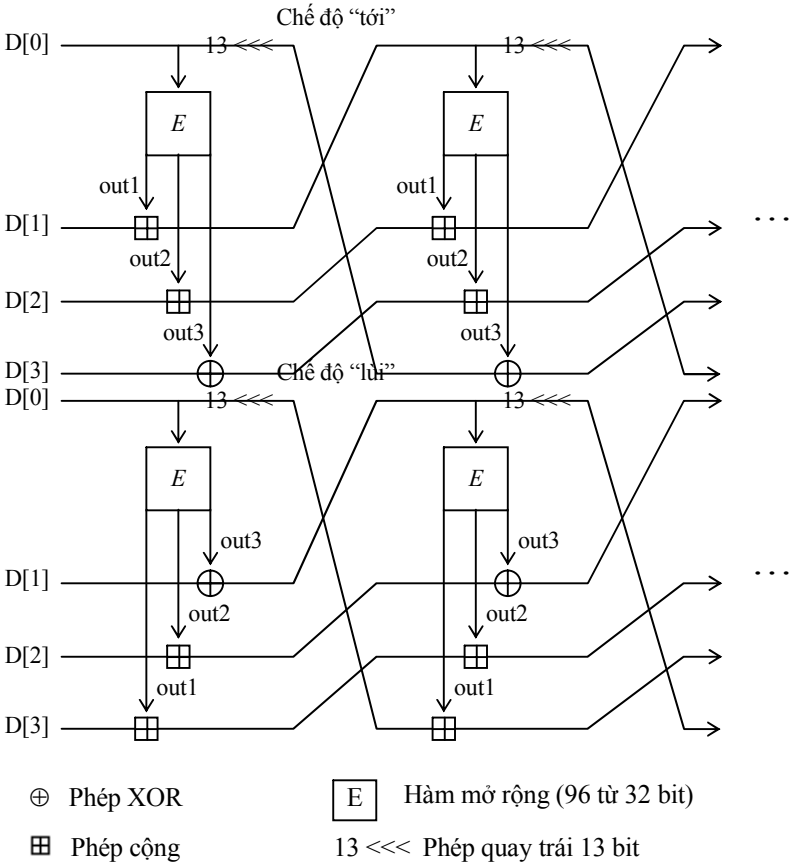
Hình 5.2. *Cấu trúc giai đoạn "Trộn tới"*

Hơn nữa, sau mỗi 4 chu kỳ riêng biệt cộng một từ trong các từ đích với từ nguồn. Cụ thể, sau chu kỳ thứ nhất và chu kỳ thứ năm cộng từ đích thứ 3 với từ nguồn và sau chu kỳ thứ hai và chu kỳ thứ sáu cộng từ đích thứ nhất với từ nguồn. Lý do để thực hiện thêm những phép trộn lẫn thêm vào này là để loại trừ một vài phương pháp tấn công vi phân chống lại giai đoạn này.

5.1.4.2 Giai đoạn 2: phần lõi chính của giai đoạn mã hóa

Phần lõi chính của quy trình mã hóa MARS là một hệ thống Feistel loại 3 bao gồm 16 chu kỳ. Trong mỗi chu kỳ sử dụng một hàm *E* được xây dựng dựa trên một tổ hợp của các phép nhân, phép quay phụ thuộc dữ liệu và S–box. Hàm này nhận vào một từ dữ liệu và trả ra ba từ dữ liệu. Cấu trúc của hệ thống Feistel được thể hiện trong Hình 5.3 và hàm *E* được mô tả trong Hình 5.4. Trong mỗi chu kỳ sử dụng một từ dữ liệu đưa vào *E* và cho ra ba từ dữ liệu được cộng hoặc XOR với ba từ dữ liệu khác. Sau khi thực hiện xong hàm E từ nguồn được quay 13 bit về bên trái.

Để đảm bảo rằng việc mã hóa có sức chống chọi các phương pháp xâm nhập văn bản mã hóa, ba từ dữ liệu cho ra từ hàm *E* được dùng với một thứ tự khác hơn trong 8 chu kỳ đầu so với 8 chu kỳ sau. Nghĩa là, trong 8 chu kỳ đầu cộng từ thứ nhất và từ thứ hai từ kết quả hàm *E* với từ đích thứ nhất và thứ hai, và XOR từ thứ ba từ kết quả hàm *E* với từ đích thứ ba. Trong 8 chu kỳ cuối, cộng từ thứ nhất và từ thứ hai từ kết quả hàm *E* với từ đích thứ ba và thứ hai, và XOR từ thứ ba từ kết quả hàm *E* với từ đích thứ nhất.

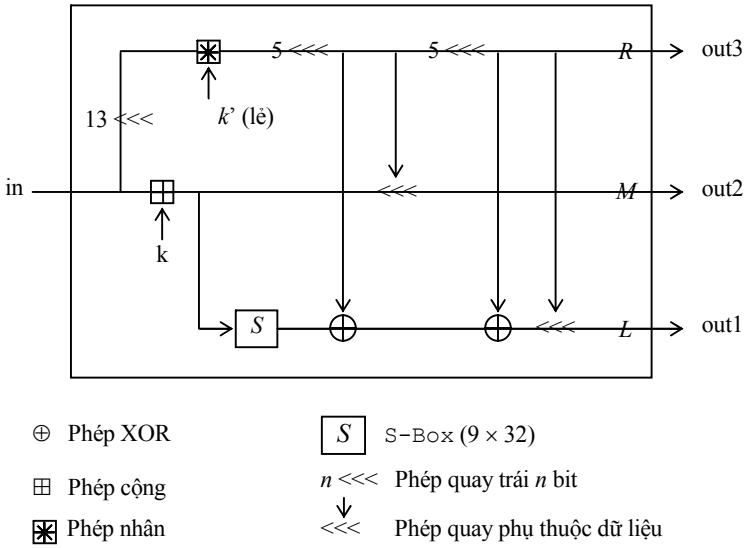


Hình 5.3. *Hệ thống Feistel loại 3*

5.1.4.3 Hàm E

Hàm *E* nhận vào một từ dữ liệu và sử dụng hai từ khóa nữa để sinh ra ba từ. Trong hàm này dùng ba biến tạm *L*, *M* và *R* (tương ứng với trái, giữa và phải).

Đầu tiên, *R* giữ giá trị của từ nguồn được quay 13 bit về bên trái và *M* giữ giá trị tổng của từ nguồn và từ khóa thứ nhất. Sau đó xem 9 bit thấp nhất của *M* như một chỉ số của `S-box` S 512–entry (thu được bằng cách kết hợp S0 và S1 từ giai đoạn trộn) và *L* giữ giá trị của một mục tương ứng trong `S-box`.



Hình 5.4. *Hàm E*

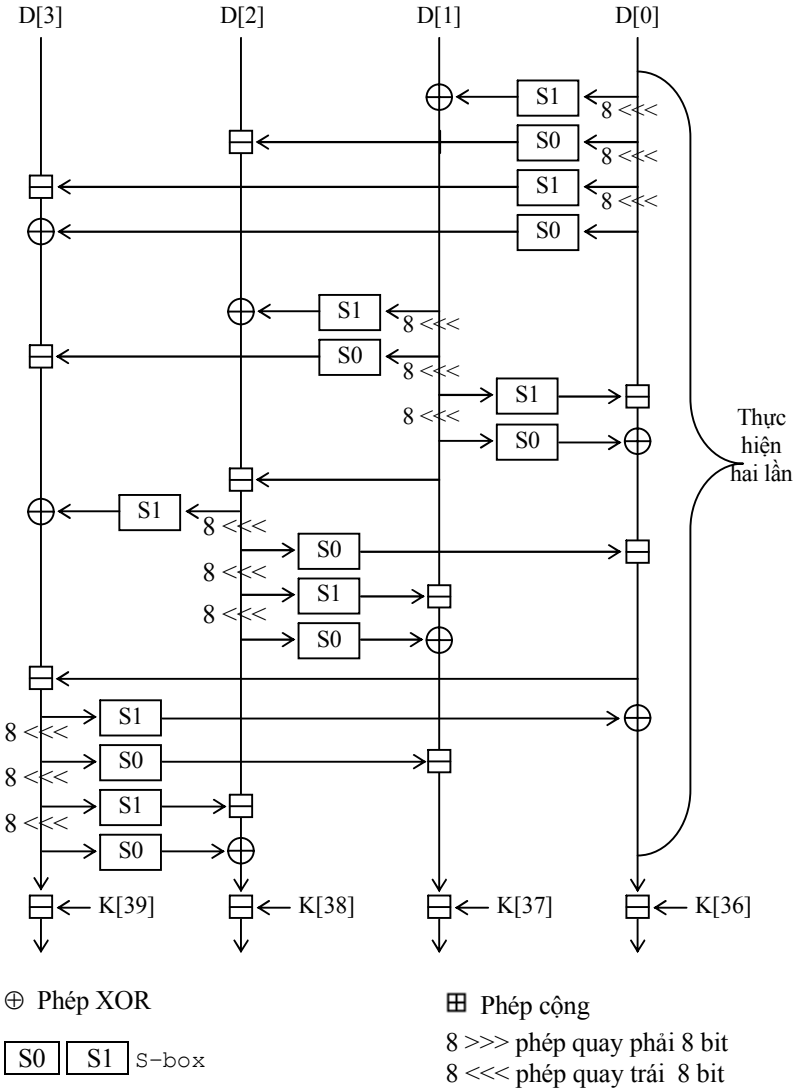
Tiếp theo nhân từ khóa thứ hai (phải chứa một số nguyên lẻ) với *R* và quay *R* 5 bit về bên trái (do đó 5 bit cao nhất của tích số trở thành 5 bit thấp nhất của *R* sau khi quay). Kế đến xor *R* và *L*, và cũng xem 5 bit thấp nhất của *R* như một số bit quay trong khoảng 0 và 31, và quay *M* về bên trái với số bit quay này. Tiếp theo, quay *R* 5 bit nữa về bên trái và XOR với *L*. Cuối cùng, lại xem 5 bit thấp nhất của *R* như một số bit quay và quay *L* về bên trái với số bit quay này. Từ kết quả thứ nhất của hàm *E* là *L*, thứ hai là *M* và thứ ba là *R*.

Dưới đây là đoạn mã giả cho hàm *E*

```
E-function(input: in, key1, key2)
//Sử dụng 3 biến tạm L, M, R
    M = in + key1          //cộng từ đầu tiên của khóa
    R =(in <<< 13)×key2    //nhân với từ thứ 2 của khóa (số lẻ)
    m = 9 bit thấp của M
    L = S[m]               //Bảng tra S–box
    R = R <<< 5
    R = 5 bit thấp của R   //xác định số bit cần quay
    M = M <<< r            //phép quay phụ thuộc dữ liệu lần 1
    L = L ⊕ R
    R = R <<< 5
    L = L ⊕ R
    r = 5 bit thấp của R   //xác định số bit cần quay
    L = L <<< r            //phép quay phụ thuộc dữ liệu lần 2
    output(L, M, R)
```

5.1.4.4 Giai đoạn 3: Trộn lùi

Giai đoạn trộn lùi giống giai đoạn trộn tới của quy trình mã hóa, ngoại trừ các từ dữ liệu được xử lý theo thứ tự khác. Nghĩa là, nếu đưa kết quả từ giai đoạn trộn tới không dùng khóa vào giai đoạn trộn lùi không dùng khóa theo thứ tự đảo lại (tức là dữ liệu kết quả D[3] đưa vào dữ liệu vào D[0], dữ liệu kết quả D[2] đưa vào dữ liệu vào D[1], …) sau đó hai giai đoạn này sẽ khử lẫn nhau. Hình 5.5 thể hiện giai đoạn trộn lùi.



Hình 5.5. *Cấu trúc giai đoạn "Trộn lùi"*

Như ở giai đoạn trộn tới, ở đây cũng vậy trong mỗi chu kỳ sử dụng một từ nguồn để thay đổi ba từ đích khác. Bốn byte của từ nguồn được biểu diễn bằng b_0, b_1, b_2, b_3. Với b_0 và b_2 được sử dụng làm chỉ số cho `S-box` S1; b_1 và b_3 làm chỉ số cho `S-box` S0. XOR S1[b_0] với từ đích thứ nhất, trừ S0[b_3] với từ dữ liệu thứ hai, trừ S1[b_2] với từ đích thứ ba và sau đó XOR S0[b_1] với từ đích thứ ba. Cuối cùng, quay từ nguồn 24 bit về bên trái.

Đối với chu kỳ kế tiếp quay bốn từ về bên phải một từ để từ đích thứ nhất hiện tại trở thành từ nguồn kế tiếp, từ đích thứ hai hiện tại trở thành từ đích thứ nhất kế tiếp, từ đích thứ ba hiện tại trở thành từ đích thứ hai kế tiếp và từ nguồn hiện tại trở thành từ đích thứ ba kế tiếp.

Cũng như vậy, trước mỗi bốn chu kỳ riêng biệt trừ một từ trong số các từ đích với từ nguồn: trước chu kỳ thứ tư và chu kỳ thứ tám trừ từ đích thứ nhất với từ nguồn và trước chu kỳ thứ ba và chu kỳ thứ bảy trừ từ đích thứ ba với từ nguồn.

5.1.4.5 Quy trình mã hóa MARS

Trong đoạn mã giả mô tả quy trình mã hóa của phương pháp MARS sử dụng các kí hiệu và quy ước sau:

1. Các phép toán sử dụng trong mã hóa được thực hiện trên các từ 32 bit (được xem là số nguyên không dấu). Các bit được đánh số từ 0 đến 31, bit 0 là bit thấp nhất và bit 31 là bit cao nhất.

2. Chúng ta biểu diễn:
   $a \oplus b$ là phép XOR của a và b,

$a \vee b$ và $a \wedge b$ là phép OR và AND của a và b.

a + b biểu diễn phép cộng modulo 2^{32}.

$a - b$ biểu diễn phép trừ modulo 2^{32}.

$a \times b$ biểu diễn phép nhân modulo 2^{32}.

a <<< b và a >>> b biểu diễn phép quay của từ 32 bit a sang phải hoặc sang trái b bit.

(D[3], D[2], D[1], D[0]) ← (D[0], D[3], D[2], D[1]) biểu diễn phép quay một mảng bốn từ sang phải một từ.

```
MARS-Encrypt(input: D[], K[])
Pha (I): Trộn "tới"
//Trước tiên, cộng các subkey vào dữ liệu
for i = 0 to 3
    D[i] = D[i] = K[i]

//Sau đó thực hiện 8 chu kỳ trộn "tới"
for i = 0 to 7     //Dùng D[0] để thay đổi D[1], D[2], D[3]
    //Tra bảng S–box
    D[1] = D[1] ⊕ S0[byte thứ 1 của D[0]]
    D[1] = D[1] + S1[byte thứ 2 của D[0]]
    D[2] = D[2] + S0[byte thứ 3 của D[0]]
    D[3] = D[3] ⊕ S1[byte thứ 4 của D[0]]

    //thực hiện phép quay phải từ nguồn (source word)
    D[0] = D[0] >>> 24
```

```
    //Thao tác trộn bổ sung
    if i = 1 or 4 then
        D[0] = D[0] + D[3]    //Cộng D[3] vào từ nguồn
    end if
    if i = 1 or 5 then
        D[0] = D[0] + D[1]    //Cộng D[1] vào từ nguồn
    end if

    //Quay D[] sang phải 1 từ để chuẩn bị cho chu kỳ tiếp theo
    (D[3], D[2], D[1], D[0]) ← (D[0], D[3], D[2], D[1])
end for
```

Pha (II) Biến đổi sử dụng khóa

```
//Thực hiện 16 chu kỳ biến đổi có khóa
for i = 0 to 15
    (out1,out2,out3) = E-function(D[0], K[2i + 4], K[2i + 5])
    D[0] = D[0] <<< 13
    D[2] = D[2] + out2

    if i < 8 then      //8 chu kỳ đầu – chế độ "tới"
        D[1] = D[1] + out1
        D[3] = D[3] ⊕ out3
    else               //8 chu kỳ sau – chế độ "lùi"
        D[3] = D[3] + out1
        D[1] = D[1] ⊕ out3
    end if

    //Quay D[] sang phải 1 từ để chuẩn bị cho chu kỳ tiếp theo
    (D[3], D[2], D[1], D[0]) ← (D[0], D[3], D[2], D[1])
end for
```

Pha (III): Trộn "lùi"

```
//Thực hiện 8 chu kỳ trộn "lùi"
for i = 0 to 7
    //Thao tác trộn bổ sung
    if i = 2 or 6 then
        D[0] = D[0] – D[3]    //trừ từ nguồn cho D[3]
    if i = 3 or 7 then
        D[0] = D[0] – D[1]    //trừ từ nguồn cho D[1]

    //Tra bảng S–box
    D[1] = D[1] ⊕ S1[byte thứ 1 của D[0]]
    D[2] = D[2] – S0[byte thứ 4 của D[0]]
    D[3] = D[3] – S1[byte thứ 3 của D[0]]
    D[4] = D[4] ⊗ S0[byte thứ 2 của D[0]]

    //Quay từ nguồn sang trái
    D[0] = D[0] <<< 24

    //Quay D[] sang phải 1 từ để chuẩn bị cho chu kỳ tiếp theo
    (D[3], D[2], D[1], D[0]) ← (D[0], D[3], D[2], D[1])
end for

//Trừ dữ liệu cho subkey
for i = 0 to 3
    D[i] = D[i] – K[36 + i]
end for
```

5.1.5 Quy trình giải mã

Quy trình giải mã là nghịch đảo của quy trình mã hóa. Mã giả cho quy trình giải mã của thuật toán MARS tương tự với mã giả của quy trình mã hóa của thuật toán

```
MARS-Decrypt(input: D[], K[])
Pha (I): Trộn "tới"
// Cộng các subkey vào dữ liệu
for i = 0 to 3
    D[i] = D[i] + K[36 + i]

//Thực hiện 8 chu kỳ trộn "tới"
for i = 7 downto 0
    //Quay D[] sang trái 1 từ để bắt đầu xử lý trong chu kỳ này
    (D[3], D[2], D[1], D[0]) ← (D[2], D[1], D[0], D[3])

    //Quay từ nguồn sang phải
    D[0] = D[0] >>> 24

    //Tra bảng S–box
    D[4] = D[4] ⊕ S0[byte thứ 2 của D[0]]
    D[3] = D[3] + S1[byte thứ 3 của D[0]]
    D[2] = D[2] + S0[byte thứ 4 của D[0]]
    D[1] = D[1] ⊕ S1[byte thứ 1 của D[0]]

    //Thao tác trộn bổ sung
    if i = 2 or 6 then
        D[0] = D[0] + D[3]    //Cộng D[3] vào từ nguồn
```

```
    if i = 3 or 7 then
        D[0] = D[0] + D[1]    //Cộng D[1] vào từ nguồn
end for
```

Pha (II): Biến đổi sử dụng khóa

```
//Thực hiện 16 chu kỳ biến đổi có khóa
for i = 15 downto 0
    //Quay D[] sang trái 1 từ để bắt đầu chu kỳ này
    (D[3], D[2], D[1], D[0]) ← (D[2], D[1], D[0], D[3])
    D[0] = D[0] >>> 13
    (out1, out2, out3)=E-function(D[0], K[2i + 4], K[2i + 5])
    D[2] = D[2] – out2

    if i < 8 then //8 chu kỳ cuối – chế độ "tới"
        D[1] = D[1] – out1
        D[3] = D[3] ⊕ out3
    else          //8 chu kỳ đầu – chế độ "lùi"
        D[3] = D[3] – out1
        D[1] = D[1] ⊕ out3
    end if
end for
```

Pha (III): Trộn "lùi"

```
//Thực hiện 8 chu kỳ trộn "lùi"
for i = 7 downto 0
    //Quay D[] sang trái 1 từ để bắt đầu chu kỳ này
    (D[3], D[2], D[1], D[0]) ← (D[2], D[1], D[0], D[3])

    //Thao tác trộn bổ sung
    if i = 0 or 4 then
```

```
        D[0]=D[0] – D[3] //Trừ từ nguồn cho D[3]
    if i = 1 or 5 then
        D[0] = D[0] – D[1]   //Trừ từ nguồn cho D[1]

    //Quay từ nguồn sang trái
    D[0] = D[0] <<< 24

    //Tra bảng S–box
    D[3] = D[3] ⊕ S1[byte thứ 4 của D[0]]
    D[2] = D[2] – S0[byte thứ 3 của D[0]]
    D[1] = D[1] – S1[byte thứ 2 của D[0]]
    D[1] = D[1] ⊕ S0[byte thứ 1 của D[0]]
end for

//Trừ dữ liệu cho các subkey
for i = 0 to 3
    D[i] = D[i] – K[i]
end for
```

5.2 Phương pháp mã hóa RC6

Thuật toán RC6 tương ứng với các tham số *w*/*r*/*b*, trong đó kích thước từ là *w* bit, quy trình mã hóa bao gồm *r* chu kỳ và tham số *b* xác định chiều dài mã khóa tính bằng byte. Để đáp ứng yêu cầu khi tham gia vào việc chọn lựa chuẩn mã hóa AES, RC6 phải đạt được kích thước khóa *b* là 16, 24 và 32–byte (tương ứng với 128/192/256 bit).

RC6–*w*/*r*/*b* thực hiện trên các đơn vị bốn từ *w* bit sử dụng sáu phép toán cơ bản và Logarit cơ số 2 của *w*, ký hiệu bằng lg*w*.

$a + b$	phép cộng số nguyên modulo 2^w
$a - b$	phép trừ số nguyên modulo 2^w
$a \oplus b$	phép XOR
$a \times b$	phép nhân số nguyên modulo 2^w
$a <<< b$	quay chu kỳ tròn bên trái b bit
$a >>> b$	quay chu kỳ tròn bên phải b bit

5.2.1 Khởi tạo và phân bố khóa

RC6 lấy các từ từ khóa người sử dụng cung cấp để sử dụng trong suốt quá trình mã hóa và giải mã. Người sử dụng cung cấp một khóa có chiều dài *b* byte ($0 \le b \le 255$), thêm các byte zero vào để chiều dài khóa bằng với một số nguyên ($2r + 4$) của các từ, sau đó những byte khóa này được nạp vào tạo thành một dãy *c* từ *w* bit $L[0], \ldots, L[c-1]$. Như vậy byte đầu tiên của khóa sẽ lưu vào vị trí byte thấp của $L[0]$, … và $L[c-1]$ sẽ được thêm vào các byte zero ở vị trí cao nếu cần. (Để ý rằng nếu $b = 0$ thì $c = 1$ và $L[0] = 0$). Số từ *w* bit được phát sinh để bổ sung vào các khóa thực hiện một chu kỳ là $2r + 4$ và các khóa này được giữ lại trong mảng $S[0, \ldots, 2r + 3]$.

Hằng số $P_{32} = 0xB7E15163$ và $Q_{32} = 0x9E3779B9$ giống như "hằng số huyền bí" trong việc phân bố khóa. Giá trị P_{32} phát sinh từ việc khai triển nhị phân của $e - 2$ (e là cơ số của hàm logarit). Giá trị Q_{32} phát sinh từ việc khai triển nhị phân của $\phi - 1$ (ϕ là tỷ số vàng).

Dưới đây là đoạn mã giả cho việc khởi tạo và phân bố khóa

Key schedule của RC6–*w*/*r*/*b*

Input:

Khóa (gồm *b* byte) do người dùng cung cấp được đưa vào mảng $L[0,\ldots, c-1]$ (gồm *c*–từ)

r là số lượng chu kỳ

Output: Các khóa chu kỳ *w* bit $S[0, \ldots, 2r + 3]$

Begin

$S[0] = P_w$

for $i = 1$ **to** $2r + 3$

$S[i] = S[i - 1] + Q_w$

$A = B = i = j = 0$

$v = 3 \times \max\{c; 2r + 4\}$

for $s = 1$ **to** v

$A = S[i] = (S[i] + A + B) <<< 3$

$B = L[j] = (L[j] + A + B) <<< (A + B)$

$i = (i + 1) \bmod (2r + 4)$

$j = (j + 1) \bmod c$

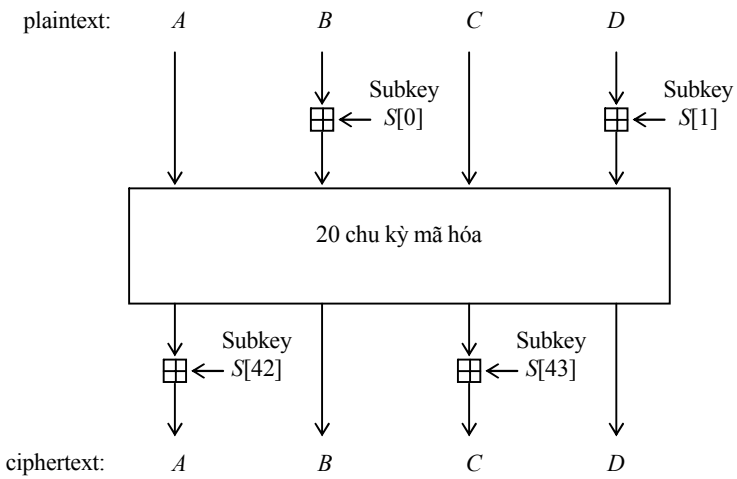
end for

End

5.2.2 Quy trình mã hóa

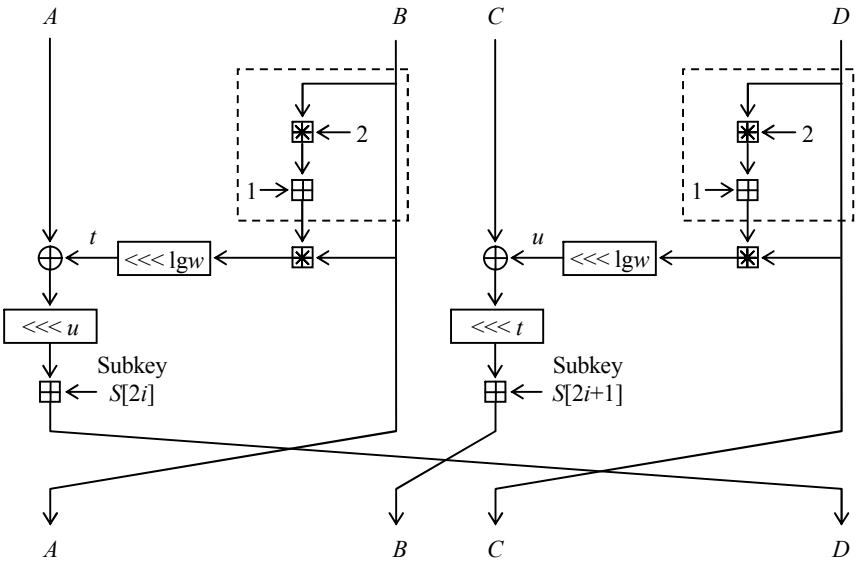
RC6 làm việc với bốn từ *w* bit *A*, *B*, *C*, *D* chứa các dữ liệu đưa vào ban đầu cũng như dữ liệu mã hóa đưa ra cuối quy trình mã hóa. Byte đầu tiên của văn bản ban

đầu và văn bản mã hóa được đặt vào vị trí byte thấp nhất của *A*; byte cuối cùng của văn bản ban đầu và văn bản mã hóa được đặt vào byte cao nhất của *D*.



Hình 5.6. *Cấu trúc mã hóa RC6*

Đầu tiên, từ *B* cộng thêm vào từ khóa thứ nhất và từ *D* cộng thêm vào từ khóa thứ hai. Tiếp theo thực hiện 20 chu kỳ liên tục. Trong mỗi chu kỳ, trước tiên quay $f(b) = b \times (2b + 1)$ sang trái lg*w* (= 5 cho kích thước từ = 32 bit) vị trí và lưu vào biến *t*. Tương tự, quay $f(d) = d \times (2d + 1)$ sang trái lg*w* vị trí và lưu vào biến *u*. Kế đến XOR từ *A* với *t* rồi quay sang trái *u* vị trí và cộng thêm vào *A* từ khóa thứ 2*i* (chu kỳ thứ *i*), tương tự XOR từ *C* với *u* rồi quay sang trái *t* vị trí và cộng thêm vào *C* từ khóa thứ 2*i* + 1.



⊕ phép XOR ⊠ phép nhân

⊞ phép cộng <<< *n* phép quay trái n bit

Hình 5.7. *Chu kỳ thứ i của quy trình mã hóa RC6*

Đối với chu kỳ kế tiếp quay bốn từ về bên phải 1 vị trí $(A,B,C,D) \Rightarrow (B,C,D,A)$. Do đó bốn từ nguồn cho chu kỳ thực hiện kế tiếp là (*B*, *C*, *D*, *A*) ứng với đầu vào là (*A*, *B*, *C*, *D*).

Sau khi thực hiện xong 20 chu kỳ, từ A cộng thêm vào từ khóa thứ $2r + 2$ (ở đây r là số chu kỳ = 20, từ khóa thứ 42) và từ C cộng thêm vào từ khóa thứ $2r + 3$ (từ khóa thứ 43).

Mã giả quy trình mã hóa RC6–*w*/*r*/*b*:

Encryption RC6–w/r/b
Input:
Dữ liệu cần mã hóa được lưu trữ trong bốn thanh ghi *w* bit *A, B, C, D*
r: số lượng chu kỳ
Các khóa chu kỳ (*w* bit) $S[0, \ldots, 2r + 3]$
Output: Thông tin đã mã hóa được lưu trữ trong bốn thanh ghi *A, B, C, D*
Begin
$B = B + S[0]$
$D = D + S[1]$
for $i = 1$ **to** r
$t = (B \times (2B + 1)) <<< \lg w$
u = $(D \times (2D + 1)) <<< \lg w$
$A = ((A \oplus t) <<< u) + S[2i]$
$C = ((C \oplus u) <<< t) + S[2i+ 1]$
$(A, B, C, D) = (B, C, D, A)$
end for
$A = A + S[2r + 2]$
$C = C + S[2r + 3]$
End

5.2.3 Quy trình giải mã

Quy trình giải mã của RC6 là nghịch đảo của quy trình mã hóa. Dưới đây là đoạn mã giả cho quy trình giải mã RC6–*w*/*r*/*b*:

Input:

Thông tin đã mã hóa cần được giải mã được lưu trữ trong bốn thanh ghi *w* bit *A*, *B*, *C*, *D*

r: số lượng chu kỳ

Các khóa chu kỳ (*w* bit) *S*[0, …, 2*r* + 3]

Output:

Dữ liệu được giải mã được lưu trữ trong 4 thanh ghi *A*, *B*, *C*, *D*

begin

$C = C - S[2r + 3]$

$A = A - S[2r + 2]$

for $i = r$ **downto** 1

$(A, B, C, D) = (D, A, B, C)$

$u = (D \times (2D + 1)) <<< \lg w$

$t = (B \times (2B + 1)) <<< \lg w$

$C = ((C - S[2i + 1]) >>> t) \oplus u$

$A = ((A - S[2i]) >>> u) \oplus t$

end for

$D = D - S[1]$

$B = B - S[0]$

end

5.3 Phương pháp mã hóa Serpent

5.3.1 Thuật toán SERPENT

Serpent là một hệ thống 32 chu kỳ thực hiện trên 4 từ 32 bit, do đó nó đưa ra kích thước khối là 128 bit. Tất cả các giá trị dùng trong việc mã hóa được xem như các dòng bit. Ứng với mỗi từ 32 bit, chỉ số bit được đánh từ 0 đến 31, các khối 128 bit có chỉ số từ 0 đến 127 và các khóa 256 bit có chỉ số từ 0 đến 255… Đối với các phép tính bên trong, tất cả các giá trị đặt trong little–endian, ở đó từ đầu tiên (từ có chỉ số 0) là từ thấp nhất, từ cuối cùng là từ cao nhất và bit 0 của từ 0 là bit thấp nhất. Ở ngoài, ta viết mỗi khối dưới dạng số hexa 128 bit.

Serpent mã hóa một văn bản ban đầu P 128 bit thành một văn bản mã hóa C 128 bit qua 32 chu kỳ với sự điều khiển của 33 subkey 128 bit ($K\hat{A}_0$, …, $K\hat{A}_{32}$). Chiều dài khóa người dùng là biến số (nếu ta cố định chiều dài khóa là 128, 192 hoặc 256 bit thì khi người sử dụng đưa vào chiều dài khóa ngắn hơn, ta đặt một bit 1 vào cuối MSB, còn lại điền các bit 0).

5.3.2 Khởi tạo và phân bố khóa

Việc mã hóa đòi hỏi 132 từ 32 bit của toàn bộ khóa. Đầu tiên từ khóa người sử dụng cung cấp (nếu cần ta biến đổi theo chiều dài khóa đã định như đã trình bày ở trên). Sau đó ta mở rộng thành 33 subkey 128 bit (K_0, …, K_{32}) bằng cách ghi khóa K thành 8 từ 32 bit (w_{-8}, …, w_{-1}) và mở rộng các từ này thành khóa trung gian w_0, …, w_{131} bằng công thức sau:

$$w_i = (w_{i-8} \oplus w_{i-5} \oplus w_{i-3} \oplus w_{i-1} \oplus \phi \otimes i) <<< 11 \qquad (5.3)$$

ở đây ϕ là phần phân số của tỉ số vàng $(\sqrt{5}+1)/2$ hoặc số hexa 0x9e3779b9. Đa thức cơ sở $x^8+x^7+x^5+x^3+1$ cùng với phép cộng của chỉ số chu kỳ được chọn đảm bảo một sự phân bố đều đặn các bit khóa qua các chu kỳ, loại các khóa yếu và các khóa buộc.

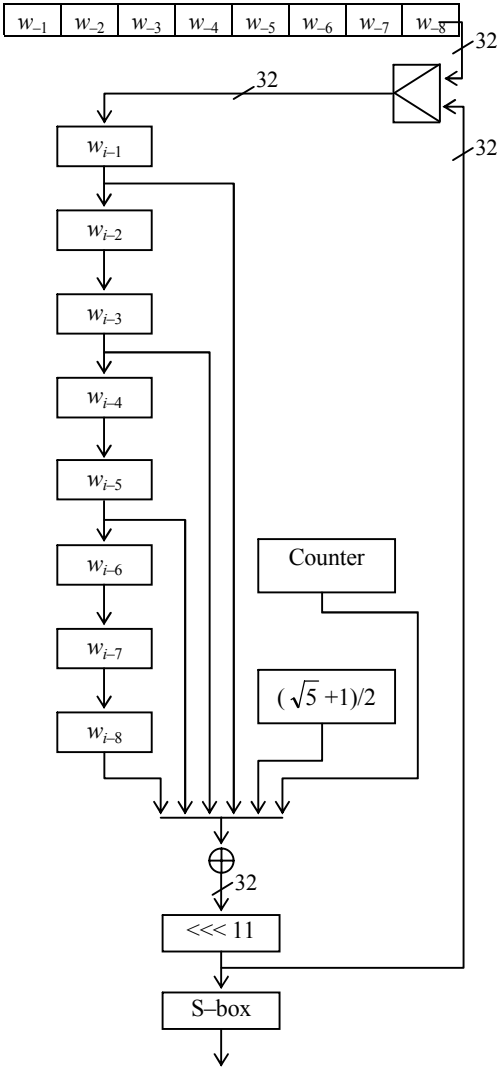
Những khóa thực hiện một chu kỳ được suy ra từ các khóa trước khi sử dụng các `S-box`. Sử dụng `S-box` để biến đổi các khóa w_i thành các từ k_i của khóa chu kỳ theo cách sau:

$$\begin{aligned}
\{k_0, k_1, k_2, k_3\} &= S_3(w_0, w_1, w_2, w_3)\\
\{k_4, k_5, k_6, k_7\} &= S_2(w_4, w_5, w_6, w_7)\\
\{k_8, k_9, k_{10}, k_{11}\} &= S_1(w_8, w_9, w_{10}, w_{11})\\
\{k_{12}, k_{13}, k_{14}, k_{15}\} &= S_0(w_{12}, w_{13}, w_{14}, w_{15})\\
\{k_{16}, k_{17}, k_{18}, k_{19}\} &= S_7(w_{16}, w_{17}, w_{18}, w_{19})\\
&\ldots\\
\{k_{124}, k_{125}, k_{126}, k_{127}\} &= S_4(w_{124}, w_{125}, w_{126}, w_{127})\\
\{k_{128}, k_{129}, k_{130}, k_{131}\} &= S_3(w_{128}, w_{129}, w_{130}, w_{131})
\end{aligned} \tag{5.4}$$

Ta đánh số lại các giá trị 32 bit k_j giống các subkey 128 bit K_i (cho $i \in 0, \ldots, r$) như sau:

$$K_i = \{k_{4i}, k_{4i+1}, k_{4i+2}, k_{4i+3}\} \tag{5.5}$$

Kế đến áp dụng phép hoán vị đầu (IP) vào khóa thực hiện một chu kỳ để định vị các bit khóa vào đúng vị trí (cột).



Hình 5.8. *Mô hình phát sinh khóa*

5.3.3 S–box

S–box của Serpent là phép hoán vị 4 bit. S–box được phát sinh theo cách sau: sử dụng một ma trận gồm 32 dãy, mỗi dãy 16 phần tử. Ma trận được khởi gán với 32 hàng của S–box DES và được biến đổi bằng cách hoán đổi các phần tử trong dãy *r* tùy thuộc vào giá trị của các phần tử trong dãy (*r* + 1) và chuỗi ban đầu đại diện cho một khóa. Nếu dãy kết quả có các đặc tính như mong muốn (vi phân và tuyến tính), ta lưu dãy này như một Serpent S–box. Lặp đi lặp lại thủ tục này đến khi 8 S–box được phát sinh.

Chính xác hơn, cho serpent[·] là một dãy chứa 4 bit thấp nhất (thấp nhất) của mỗi 16 kí tự ASCII "sboxesforserpent". Cho sbox[·][·] là một dãy (32 x 16) chứa 32 hàng của 8 S–box DES, ở đây sbox[r][·] là hàng *r*. Hàm `swapentries`(·, ·) dùng để hoán vị hai phần tử.

Dưới đây là đoạn mã giả phát sinh S–box

```
index = 0
repeat
    currentsbox = index mod 32;
    for i = 0 to 15
        j = sbox[(currentsbox+1) mod 32][serpent[i]];
        swapentries (sbox[currentsbox][i], sbox[currentsbox][j]);
    end for
    if sbox[currentsbox][.] có tính chất theo yêu cầu then lưu lại;
    index = index + 1;
until 8 S–boxes đã được phát sinh xong
```

Phụ lục C trình bày nội dung chi tiết `S-box` và `S-box` nghịch đảo được sử dụng trong thuật toán Serpent.

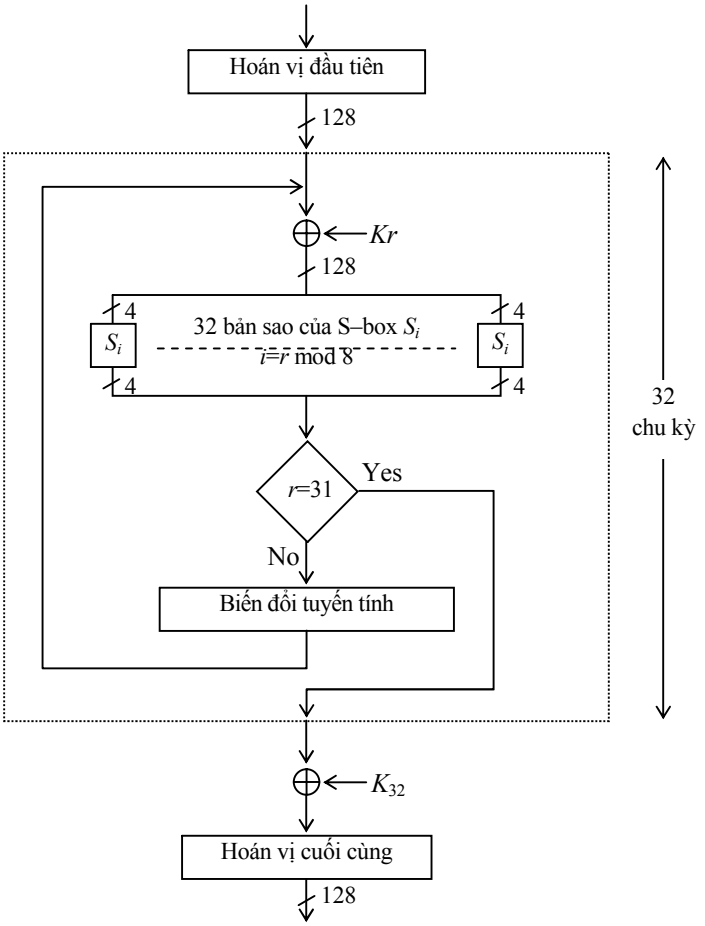
5.3.4 Quy trình mã hóa

Việc mã hóa bao gồm:

1. Phép hoán vị đầu IP (initial permutation);
2. 32 chu kỳ, mỗi chu kỳ bao gồm một phép trộn khóa, một lượt duyệt qua các S–box và một phép biến đổi tuyến tính (cho tất cả các chu kỳ trừ chu kỳ cuối). Ở chu kỳ cuối cùng, phép biến đổi tuyến tính này thay thế bằng một phép trộn khóa.
3. Phép hoán vị cuối FP (final permutation).

Phép hoán vị đầu và hoán vị cuối được trình bày chi tiết trong Phụ lục B - Các hoán vị sử dụng trong thuật toán Serpent.

Ta sử dụng các ký hiệu như sau: Phép hoán vị đầu IP áp dụng vào văn bản ban đầu P cho ra $B\hat{A}_0$ là dữ liệu vào chu kỳ thứ nhất (các chu kỳ đánh số từ 0 đến 31). Dữ liệu ra của chu kỳ thứ nhất là $B\hat{A}_1$, dữ liệu ra của chu kỳ thứ hai là $B\hat{A}_2$, dữ liệu ra của chu kỳ thứ i là $B\hat{A}_{i+1}$… cho đến chu kỳ cuối cùng. Phép biến đổi tuyến tính ở chu kỳ cuối cùng thay thế bằng phép trộn khóa được ký hiệu $B\hat{A}_{32}$. Phép hoán vị cuối FP áp dụng vào $B\hat{A}_{32}$ cho ra văn bản mã hóa C.



Hình 5.9. *Cấu trúc mã hóa*

Cho K_i là subkey 128 bit chu kỳ thứ i và `S-box` S_i được sử dụng ở chu kỳ thứ i. Cho L là phép biến đổi tuyến tính. Khi đó hàm thực hiện một chu kỳ được định nghĩa như sau:

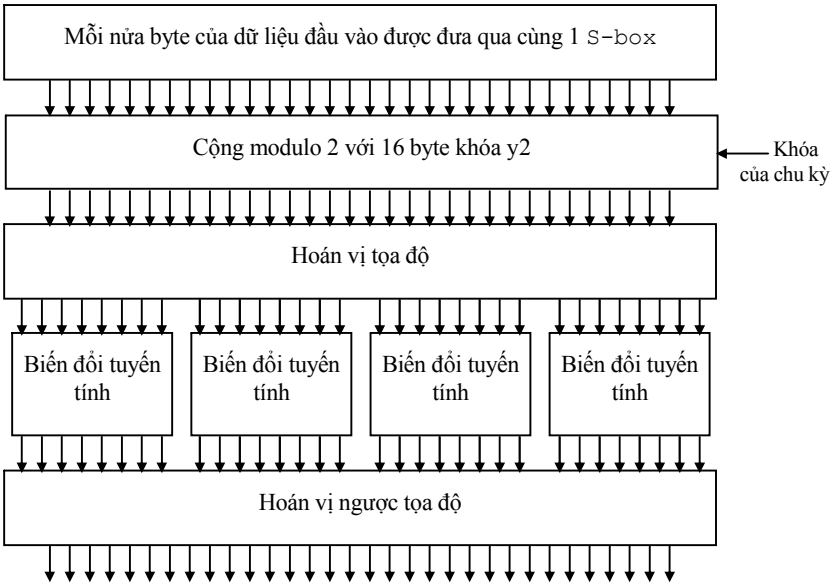
$$X_i \leftarrow B_i \oplus K_i$$

$$Y_i \leftarrow S_i(X_i)$$

$$B_{i-1} \leftarrow L(Y_i),\ i = 0, \ldots, 30$$

$$B_{i-1} \leftarrow Y_i \oplus K_{i-1},\ i = 31 \qquad (5.6)$$

Hình 5.8 thể hiện các bước thực hiện trong chu kỳ thứ i (i = 0, …, 30) của quy trình mã hóa Serpent. Riêng chu kỳ thứ 31, phép biến đổi tuyến tính được thay bằng phép cộng modulo 2 với round key.



Hình 5.10. *Chu kỳ thứ i (i = 0, ..., 30) của quy trình mã hóa Serpent*

Ở mỗi chu kỳ hàm R_i ($i \in \{0, \ldots, 31\}$) chỉ sử dụng một bản sao `S-box`. Ví dụ: R_0 sử dụng bản sao S_0, 32 bản sao của S_0 được thực hiện song song. Do đó bản sao thứ nhất của S_0 chọn các bit 0, 1, 2 và 3 của $B\hat{A}_0 \oplus K\hat{A}_0$ làm dữ liệu vào và trả ra 4 bit đầu của vector trung gian, bản sao kế tiếp của S_0 chọn các bit từ 4 đến 7 của $B\hat{A}_0 \oplus K\hat{A}_0$ làm dữ liệu vào và trả ra 4 bit kế tiếp của vector trung gian… Sau đó sử dụng phép biến đổi tuyến tính để biến đổi vector trung gian này, kết quả cho ra $B\hat{A}_1$. Tương tự R_1 sử dụng 32 bản sao của S_1 thực hiện song song trên $B\hat{A}_1 \oplus K\hat{A}_1$ và sử dụng phép biến đổi tuyến tính để biến đổi dữ liệu ra, kết quả cho ra $B\hat{A}_2$.

Xét một `S-box` S_i ứng dụng vào khối X_i 128 bit. Đầu tiên tách X_i thành 4 từ 32 bit x_0, x_1, x_2 và x_3. Ứng với mỗi vị trí của 32 bit, xây dựng một bộ 4 bit từ mỗi từ và bit ở vị trí x_3 là bit cao nhất. Sau đó áp dụng `S-box` S_i vào để xây dựng 4 bit và lưu kết quả vào các bit tương ứng của $Y_i = (y_0, y_1, y_2, y_3)$.

Phép biến đổi tuyến tính L trên $Y_i = (y_0, y_1, y_2, y_3)$ định nghĩa như sau:

$$
\begin{aligned}
&y_0 \leftarrow y_0 <<< 13\\
&y_2 \leftarrow y_2 <<< 3\\
&y_1 \leftarrow y_0 \oplus y_1 \oplus y_2\\
&y_3 \leftarrow y_2 \oplus y_3 \oplus (y_0 << 3)\\
&y_1 \leftarrow y_1 <<< 1\\
&y_3 \leftarrow y_3 <<< 7\\
&y_0 \leftarrow y_0 \oplus y_1 \oplus y_3\\
&y_2 \leftarrow y_2 \oplus y_3 \oplus (y_1 << 7)\\
&y_0 \leftarrow y_0 <<< 5\\
&y_2 \leftarrow y_2 <<< 22\\
&B_{i+1} \leftarrow (y_0, y_1, y_2, y_3)
\end{aligned}
\qquad (5.7)
$$

Trong các biểu thức trên đây, ký hiệu <<< là phép quay trái và << là phép dịch trái.

Bộ tám `S-box` ($S_0 \ldots S_7$) được sử dụng 4 lần. Do đó sau khi sử dụng S_7 ở chu kỳ 7, S_0 lại tiếp tục được sử dụng ở chu kỳ 8, S_1 ở chu kỳ 9… Ở chu kỳ cuối cùng hàm R_{31} hơi khác so với các hàm còn lại: áp dụng S_7 vào $B\hat{A}_{31} \oplus K\hat{A}_{31}$ và XOR kết quả thu được với $K\hat{A}_{32}$. Sau đó kết quả $B\hat{A}_{32}$ được hoán vị bằng FP cho ra văn bản mã hóa.

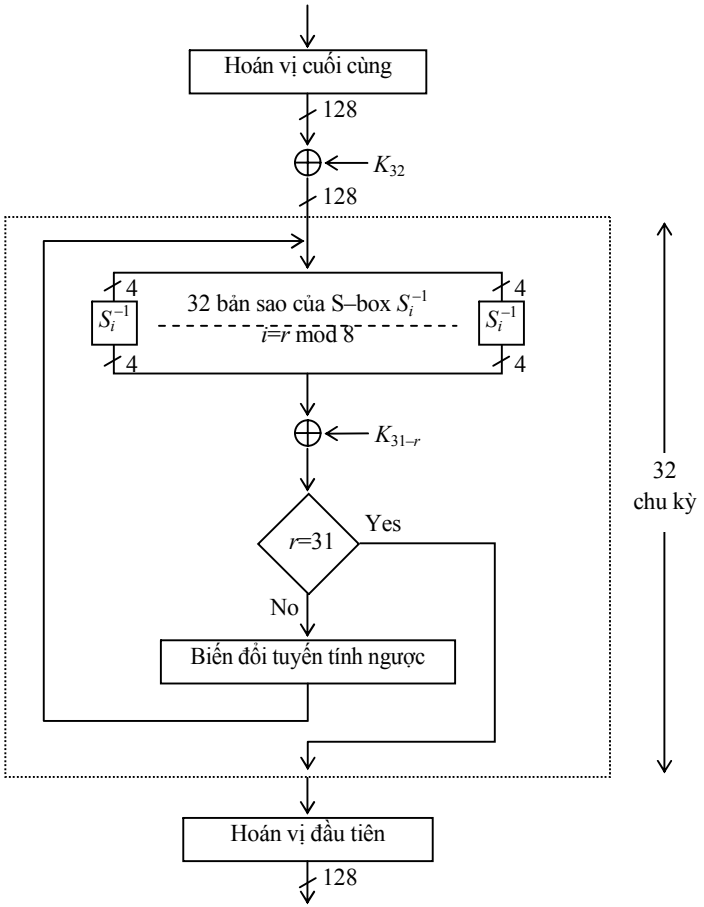
Vậy 32 chu kỳ sử dụng 8 `S-box` khác nhau, mỗi S–box ánh xạ 4 bit vào thành 4 bit ra. Mỗi S–box sử dụng 4 chu kỳ riêng biệt và trong mỗi chu kỳ S–box được sử dụng 32 lần song song.

Phép hoán vị cuối là nghịch đảo của phép hoán vị đầu. Do đó việc mã hóa có thể mô tả bằng công thức sau:

$$
\begin{aligned}
&B\hat{A}_0 = \mathrm{IP}(P)\\
&B\hat{A}_{i+1} = R_i(B\hat{A}_i)\\
&C = \mathrm{FP}(B\hat{A}_{32})\\
&R_i(X) = L(S\hat{A}_i(X \oplus K\hat{A}_i)),\ i = 0, \ldots, 30\\
&R_i(X) = S\hat{A}_i(X \oplus K\hat{A}_i) \oplus K\hat{A}_{32},\ i = 31
\end{aligned}
\qquad (5.8)
$$

ở đây $S\hat{A}_i$ là kết quả khi áp dụng S–box $S_{i \bmod 8}$ 32 lần song song và L là phép biến đổi tuyến tính.

5.3.5 Quy trình giải mã



Hình 5.11. *Cấu trúc giải mã*

Quy trình giải mã có khác với quy trình mã hóa. Cụ thể là nghịch đảo các S–box (S–box $^{-1}$) phải được sử dụng theo thứ tự ngược lại, cũng như nghịch đảo của biến đổi tuyến tính và nghịch đảo thứ tự các subkey.

5.4 Phương pháp mã hóa TwoFish

5.4.1 Khởi tạo và phân bố khóa

Giai đoạn tạo khóa phát sinh ra 40 từ khóa mở rộng K_0, …, K_{39} và bốn `S-box` phụ thuộc khóa để sử dụng trong hàm *g*. Thuật toán Twofish được xây dựng đối với chiều dài khóa $N = 128$, $N = 192$ và $N = 256$ bit. Các khóa có chiều dài bất kỳ ngắn hơn 256 có thể được biến đổi thành khóa 256 bit bằng cách điền các số 0 vào cho đến khi đủ chiều dài.

Ta định nghĩa $k = N/64$. Khóa *M* bao gồm $8k$ byte m_0, ..., m_{8k-1}. Các byte này được biến đổi thành $2k$ từ 32 bit.

$$M_i = \sum_{j=0}^{3} m_{(4i+j)} . 2^{8j} \text{ , } I = 0, ..., 2k-1 \tag{5.9}$$

sau đó biến đổi thành hai từ vector có chiều dài *k*

$$M_e = (M_0, M_2, \ldots, M_{2k-2})$$

$$M_o = (M_1, M_3, \ldots, M_{2k-1}) \tag{5.10}$$

Một vector gồm *k* từ 32 bit thứ 3 cũng được suy ra từ khóa bằng cách lấy ra từng nhóm gồm 8 byte trong khóa, xem nhóm các byte này là một vector trên GF(2^8) và nhân vector này với ma trận 4×8 (thu được từ Reed–Solomon code). Sau đó

mỗi kết quả 4 byte được xem như một từ 32 bit. Những từ này kết hợp lại tạo thành vector thứ ba.

$$\begin{pmatrix} s_{i,0} \\ s_{i,1} \\ s_{i,2} \\ s_{i,3} \end{pmatrix} = \begin{pmatrix} . & \cdots & . \\ \vdots & RS & \vdots \\ . & \cdots & . \end{pmatrix} . \begin{pmatrix} m_{8i} \\ m_{8i+1} \\ m_{8i+2} \\ m_{8i+3} \\ m_{8i+4} \\ m_{8i+5} \\ m_{8i+6} \\ m_{8i+7} \end{pmatrix} \tag{5.11}$$

$$S_i = \sum_{j=0}^{3} s_{i,j} . 2^{8j} \tag{5.12}$$

với $i = 0, \ldots, k-1$ và $S = (S_{k-1}, S_{k-2}, \ldots, S_0)$

Cần lưu ý rằng thứ tự các từ trong danh sách S bị đảo ngược. Đối với ma trận nhân RS, GF(2^8) được biểu diễn bằng GF(2)[x]/$w(x)$, với $w(x) = x^8 + x^6 + x^3 + x^2 + 1$ là một đa thức tối giản bậc 8 trên GF(2). Phép ánh xạ giữa các giá trị byte và các phần tử của GF(2^8) thực hiện tương tự như đối với phép nhân ma trận MDS.

Ma trận RS được định nghĩa như sau:

$$\text{RS} = \begin{pmatrix} 01 & A4 & 55 & 87 & 5A & 58 & DB & 9E \\ A4 & 56 & 82 & F3 & 1E & C6 & 68 & E5 \\ 02 & A1 & FC & C1 & 47 & AE & 3D & 19 \\ A4 & 55 & 87 & 5A & 58 & DB & 9E & 03 \end{pmatrix} \tag{5.13}$$

5.4.1.1 Mở rộng đối với các chiều dài khóa

Twofish chấp nhận bất kỳ chiều dài khóa lên đến 256 bit. Đối với kích thước khóa không xác định (≠ 128, 192, 256), các khóa này được thêm vào các số 0 cho đủ chiều dài xác định. Ví dụ: một khóa 80 bit m_0, ..., m_9 sẽ mở rộng bằng các đặt $m_i = 0$ với $i = 10, ..., 15$ và xem nó như khóa 128 bit.

5.4.1.2 Hàm h

Hình 5.12 thể hiện tổng quan về hàm *h*. Hàm này đưa hai dữ liệu vào, một là từ 32 bit *X* và một là danh sách $L = (L_0, ..., L_{k-1})$ của các từ 32 bit, kết quả trả ra là một từ. Hàm này thực hiện k giai đoạn. Trong mỗi giai đoạn, 4 byte, mỗi byte thực hiện qua một S–box cố định và XOR với một byte trong danh sách. Cuối cùng, một lần nữa các byte này lại được thực hiện qua một S–box cố định và 4 byte nhân với ma trận MDS như trong hàm *g*. Đúng hơn, ta chia các từ thành các byte

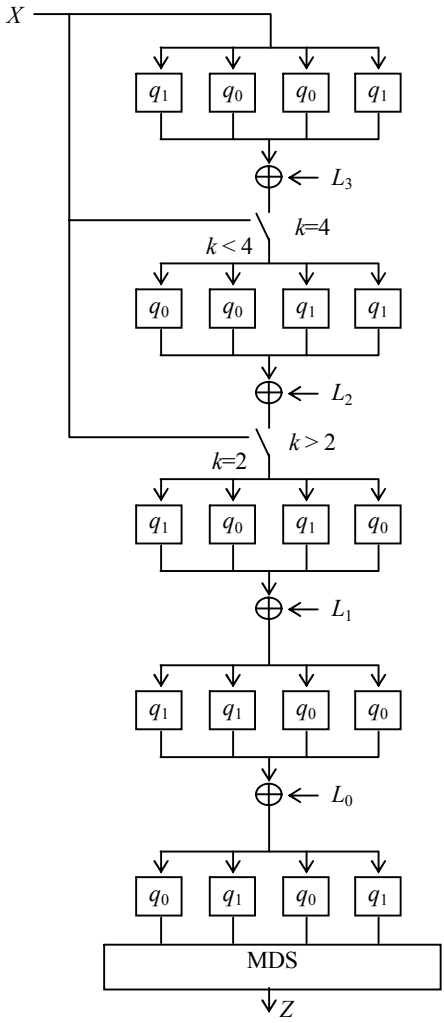
$$l_{i,j} = \left\lfloor L_i / 2^{8j} \right\rfloor \bmod 2^8$$
$$x_j = \left\lfloor X / 2^{8j} \right\rfloor \bmod 2^8 \qquad (5.14)$$

với $i = 0, ..., k - 1$ và $j = 0, ..., 3$. Sau đó lần lượt thay thế và áp dụng phép XOR.

$$y_{k,j} = x_j, j = 0,...,3 \qquad (5.15)$$

Nếu $k = 4$, ta có:

$$y_{3,0} = q_1[y_{4,0}] \oplus l_{3,0}$$
$$y_{3,1} = q_0[y_{4,1}] \oplus l_{3,1}$$
$$y_{3,2} = q_0[y_{4,2}] \oplus l_{3,2}$$
$$y_{3,3} = q_1[y_{4,3}] \oplus l_{3,3} \qquad (5.16)$$



Hình 5.12. *Hàm h*

Nếu $k \geq 3$, ta có:

$$\begin{aligned} y_{2,0} &= q_1[y_{3,0}] \oplus l_{2,0} \\ y_{2,1} &= q_0[y_{3,1}] \oplus l_{2,1} \\ y_{2,2} &= q_0[y_{3,2}] \oplus l_{2,2} \\ y_{2,3} &= q_1[y_{3,3}] \oplus l_{2,3} \end{aligned} \tag{5.17}$$

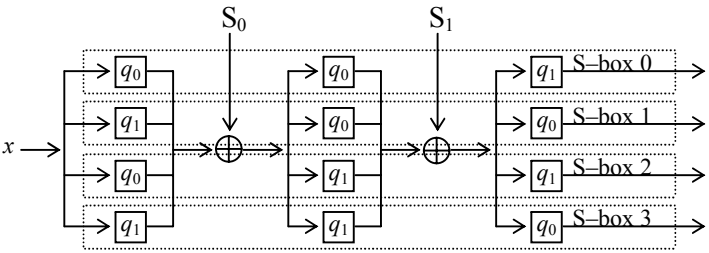
Trong mọi trường hợp ta có

$$\begin{aligned} y_0 &= q_1[q_0[q_0]y_{2,0}] \oplus l_{1,0}] \oplus l_{0,0}] \\ y_1 &= q_0[q_0[q_1]y_{2,1}] \oplus l_{1,1}] \oplus l_{0,1}] \\ y_2 &= q_1[q_1[q_0]y_{2,2}] \oplus l_{1,2}] \oplus l_{0,2}] \\ y_3 &= q_0[q_1[q_1]y_{2,3}] \oplus l_{1,3}] \oplus l_{0,3}] \end{aligned} \tag{5.18}$$

5.4.1.3 S–box phụ thuộc khóa

Mỗi S–box được định nghĩa với 2, 3 hoặc 4 byte của dữ liệu đầu vào của khóa tùy thuộc vào kích thước khóa. Điều này thực hiện như sau cho các khóa 128 bit:

$$\begin{aligned} s_0(x) &= q_1[q_0[q_0[x] \oplus s_{0,0}] \oplus s_{1,0}] \\ s_1(x) &= q_0[q_0[q_1[x] \oplus s_{0,1}] \oplus s_{1,1}] \\ s_2(x) &= q_1[q_1[q_0[x] \oplus s_{0,2}] \oplus s_{1,2}] \\ s_3(x) &= q_0[q_1[q_1[x] \oplus s_{0,3}] \oplus s_{1,3}] \end{aligned} \tag{5.19}$$



Hình 5.13. *Mô hình phát sinh các* `S-box` *phụ thuộc khóa*

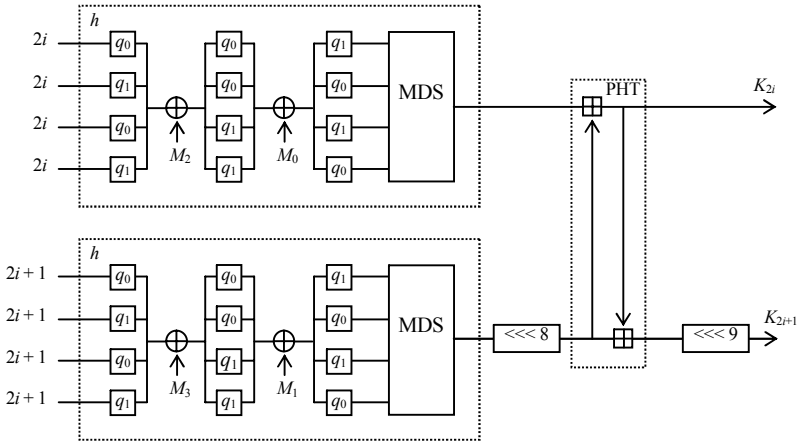
Ở đây $s_{i,j}$ là các byte lấy từ các byte khóa sử dụng ma trận RS. Để ý rằng với các byte khóa bằng nhau sẽ không có cặp `S-box` bằng nhau. Khi mọi $s_{i,j} = 0$ thì $s_0(x) = q_1[s_1^{-1}(x)]$.

Đối với khóa 128 bit, mỗi khóa $N/8$ bit dùng để xác định các kết quả hoán vị 1 byte trong một phép hoán vị riêng biệt. Ví dụ: trường hợp khóa 128 bit, S–box s_0 sử dụng 16 bit của key material. Mỗi phép hoán vị s_0 trong 2^{16} phép hoán vị được xác định riêng biệt, với s_1, s_2, s_3 cũng giống vậy.

5.4.1.4 Các từ khóa mở rộng K_j

Các từ khóa mở rộng được định nghĩa bằng cách sử dụng hàm h.

$$\begin{aligned} \rho &= 2^{24} + 2^{16} + 2^8 + 2^0 \\ A_i &= h(2i\rho, M_e) \\ B_i &= \mathrm{ROL}(h((2i+1)\rho, M_o), 8) \\ K_{2i} &= (A_i + B_i) \bmod 2^{32} \\ K_{2i+1} &= \mathrm{ROL}((A_i + 2B_i) \bmod 2^{32}, 9) \end{aligned} \qquad (5.20)$$



Hình 5.14. *Mô hình phát sinh subkey K_j*

Hằng số ρ sử dụng để nhân đôi các byte, $i \in 0, ..., 255$, $i\rho$ gồm 4 byte bằng nhau, mỗi byte ứng với giá trị i. Áp dụng hàm h lên các từ theo dạng này. Đối với A_i các giá trị byte là $2i$ và đối số thứ hai của h là M_e. Tương tự B_i được tính toán, sử dụng $2i + 1$ như giá trị byte và M_o như đối số thứ hai với một phép quay thêm trên 8 bit. Các giá trị A_i và B_i tổ hợp thành một PHT (Pseudo–Hadamard Transform). Một trong hai kết quả này quay 9 bit nữa. Hai kết quả này tạo thành hai từ khóa mở rộng.

5.4.1.5 Các phép hoán vị q_0 và q_1

Các phép hoán vị q_0 và q_1 là các phép hoán vị cố định trên các giá trị 8 bit. Chúng được xây dựng từ 4 phép hoán vị 4 bit khác nhau. Đối với giá trị dữ liệu vào x, ta xác định được giá trị dữ liệu ra y tương ứng như sau:

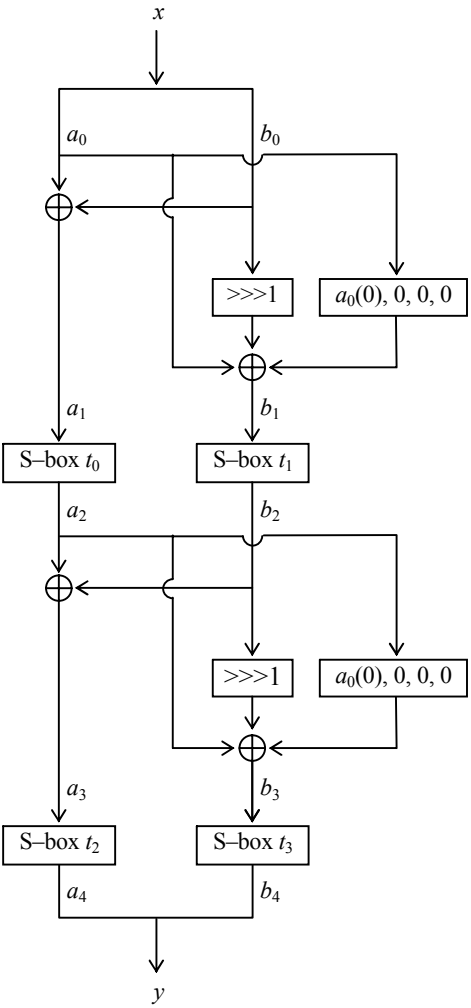
$$
\begin{aligned}
a_0, b_0 &= [x/16], x \bmod 16 \\
a_1 &= a_0 \oplus b_0 \\
b_1 &= a_0 \oplus \mathrm{ROR}_4(b_0, 1) \oplus 8a_0 \bmod 16 \\
a_2, b_2 &= t_0[a_1], t_1[b_1] \\
a_3 &= a_2 \oplus b_2 \\
b_3 &= a_2 \oplus \mathrm{ROR}_4(b_2, 1) \oplus 8a_2 \bmod 16 \\
a_4, b_4 &= t_2[a_3], t_3[b_3] \\
y &= 16b_4 + a_4
\end{aligned} \tag{5.21}
$$

Ở đây ROR_4 là hàm quay phải các giá trị 4 bit. Trước tiên, 1 byte được chia thành hai nhóm gồm 4 bit. Hai nhóm 4 bit này được kết hợp vào trong một bước trộn objective. Sau đó, mỗi 4 bit thực hiện thông qua `S-box` 4 bit cố định của chính nó ($a_1 \rightarrow t_0$, $b_1 \rightarrow t_1$). Tiếp theo tương tự cho ($a_3 \rightarrow t_2$, $b_3 \rightarrow t_3$). Cuối cùng, hai 4 bit tái kết hợp lại thành 1 byte. Đối với phép hoán vị q_0, các `S-box` 4 bit được cho như sau:

$$
\begin{aligned}
t_0 &= [\ 8\ 1\ 7\ D\ 6\ F\ 3\ 2\ 0\ B\ 5\ 9\ E\ C\ A\ 4\ ] \\
t_1 &= [\ E\ C\ B\ 8\ 1\ 2\ 3\ 5\ F\ 4\ A\ 6\ 7\ 0\ 9\ D\ ] \\
t_2 &= [\ B\ A\ 5\ E\ 6\ D\ 9\ 0\ C\ 8\ F\ 3\ 2\ 4\ 7\ 1\ ] \\
t_3 &= [\ D\ 7\ F\ 4\ 1\ 2\ 6\ E\ 9\ B\ 3\ 0\ 8\ 5\ C\ A\ ]
\end{aligned} \tag{5.22}
$$

Ở đây mỗi `S-box` 4 bit được mô tả bằng một danh sách các mục sử dụng ký hiệu *hexa* (các mục của dữ liệu vào là danh sách có thứ tự từ 0, 1, ..., 15). Tương tự, đối với q_1 các `S-box` 4 bit được cho như sau:

$$
\begin{aligned}
t_0 &= [\ 2\ 8\ B\ D\ F\ 7\ 6\ E\ 3\ 1\ 9\ 4\ 0\ A\ C\ 5\ ] \\
t_1 &= [\ 1\ E\ 2\ B\ 4\ C\ 3\ 7\ 6\ D\ A\ 5\ F\ 9\ 0\ 8\ ] \\
t_2 &= [\ 4\ C\ 7\ 5\ 1\ 6\ 9\ A\ 0\ E\ D\ 8\ 2\ B\ 3\ F\ ] \\
t_3 &= [\ B\ 9\ 5\ 1\ C\ 3\ D\ E\ 6\ 4\ 7\ F\ 2\ 0\ 8\ A\ ]
\end{aligned} \tag{5.23}
$$



Hình 5.15. *Phép hoán vị q*

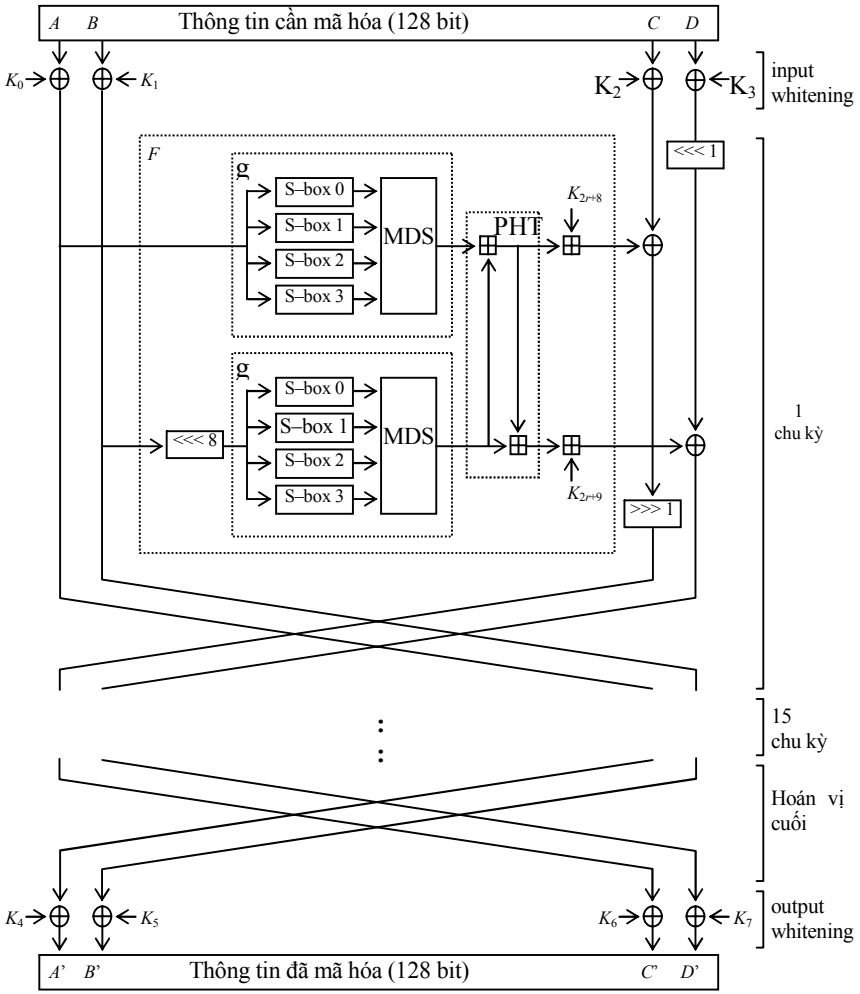
5.4.2 Quy trình mã hóa

Hình 5.16 thể hiện tổng quan về quy trình mã hóa Twofish. Twofish sử dụng một cấu trúc tựa Feistel gồm 16 chu kỳ với bộ whitening được thêm vào ở giai đoạn trước khi dữ liệu vào và ra. Chỉ các phần tử phi-Feistel là quay 1 bit. Các phép quay có thể được đưa vào trong hàm F để tạo ra một cấu trúc Feistel thuần túy.

Văn bản ban đầu đưa vào là bốn từ 32 bit *A*, *B*, *C*, *D*. Trong bước whitening dữ liệu vào, bốn từ này XOR với bốn từ khóa $K_{0..3}$. Kế đến thực hiện tiếp 16 chu kỳ. Trong mỗi chu kỳ, hai từ *A*, *B* là dữ liệu vào của hàm *g* (đầu tiên từ B được quay trái 8 bit). Hàm *g* bao gồm bốn S–box (mỗi S–box là một byte) phụ thuộc khóa, theo sau là bước trộn tuyến tính dựa trên ma trận MDS. Kết hợp kết quả trả ra của hai hàm *g* thông qua biến đổi tựa Hadamard (PHT) rồi cộng thêm vào hai từ khóa (K_{2r+8} cho *A* và K_{2r+9} cho *B* ở chu kỳ *r*). Sau đó hai kết quả này XOR với hai từ *C* và *D* (trước khi xor từ *D* với *B*, từ *D* được quay trái 1 bit và sau khi XOR từ *C* với *A*, từ *C* được quay phải 1 bit). Kế đến hai từ *A* và *C*, *B* và *D* hoán đổi cho nhau để thực hiện chu kỳ kế tiếp. Sau khi thực hiện xong 16 chu kỳ, hoán chuyển trở lại hai từ *A* và *C*, *B* và *D*, cuối cùng thực hiện phép XOR bốn từ *A*, *B*, *C*, *D* với bốn từ khóa $K_{4...7}$ cho ra bốn từ *A*', *B*', *C*', *D*' đã được mã hóa.

Chính xác hơn, đầu tiên 16 byte của văn bản ban đầu P_0, ..., P_{15} chia thành bốn từ P_0, ..., P_3 32 bit sử dụng quy ước little–endian.

$$P_i = \sum_{j=0}^{3} p_{(4i+j)} \cdot 2^{8j} \quad , \; i = 0, ..., 3 \tag{5.24}$$



Hình 5.16. *Cấu trúc mã hóa*

Trong bước whitening của dữ liệu vào, các từ này XOR với bốn từ của khóa mở rộng:

$$R_{0,i} = P_i \oplus K_i, \ \ i = 0, ..., 3 \tag{5.25}$$

Với mỗi chu kỳ trong 16 chu kỳ, hai từ A, B và chỉ số chu kỳ được sử dụng làm dữ liệu vào của hàm F. Từ C XOR với từ kết quả thứ nhất của hàm F và quay phải 1 bit. Từ thứ D quay trái 1 bit và XOR với từ kết quả thứ hai của hàm F. Cuối cùng, hai từ A và C, B và D hoán đổi cho nhau. Do đó:

$$\begin{aligned}
(F_{r,0}, F_{r,1}) &= F(R_{r,0}, R_{r,1}, r) \\
R_{r+1,0} &= \mathrm{ROR}(\mathrm{Rr}_{,2} \oplus F_{r,0}, 1) \\
R_{r+1,1} &= \mathrm{ROL}(R_{r,3}, 1) \oplus F_{r,1} \\
R_{r+1,2} &= R_{r,0} \\
R_{r+1,3} &= R_{r,1}
\end{aligned} \tag{5.26}$$

$r \in (0, ..., 15)$, ROR và ROL là hai hàm quay phải và trái với đối số thứ nhất là từ 32 bit được quay, đối số thứ hai là số bit cần quay.

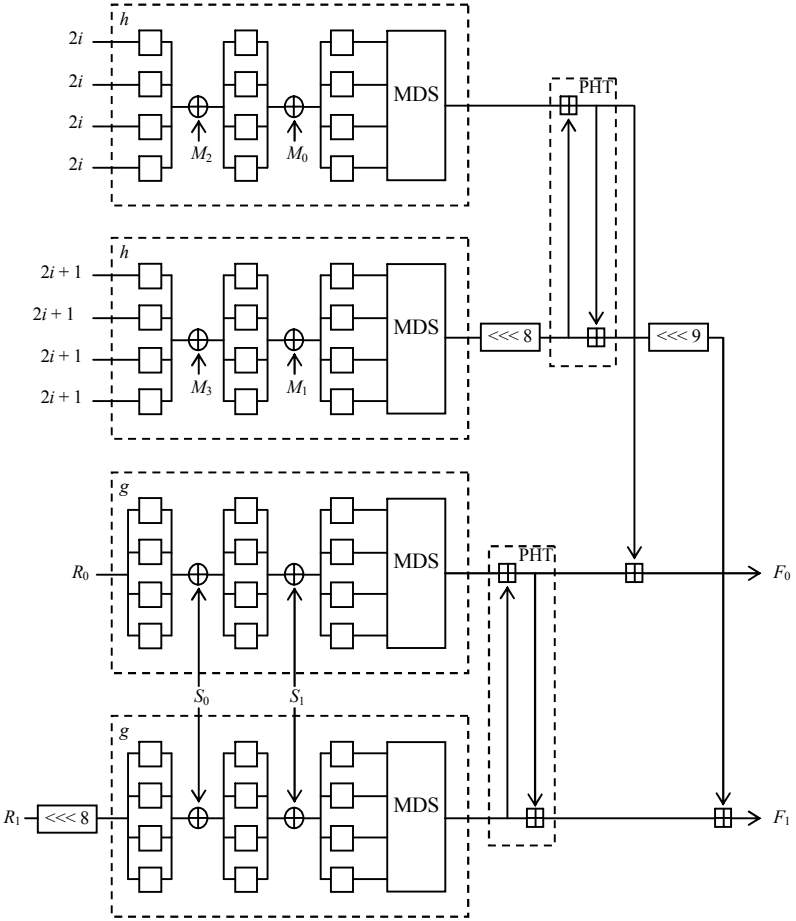
Bước whitening dữ liệu ra không thực hiện thao tác hoán chuyển ở chu kỳ cuối mà nó thực hiện phép XOR các từ dữ liệu với bốn từ khóa mở rộng.

$$C_i = R_{16,(i+2) \bmod 4} \oplus K_{i+4}, \ i = 0, ..., 3 \tag{5.27}$$

Sau đó, bốn từ của văn bản mã hóa được ghi ra thành 16 byte $c_0, ..., c_{15}$ sử dụng quy ước little–endian như đã áp dụng với văn bản ban đầu.

$$c_i = \left\lfloor \frac{C_{[i/4]}}{2^{8(i \bmod 4)}} \right\rfloor \bmod 2^8, \ i = 0, ..., 15 \tag{5.28}$$

5.4.2.1 Hàm F



Hình 5.17. *Hàm F (khóa 128 bit)*

Hàm F là phép hoán vị phụ thuộc khóa trên các giá trị 64 bit. Hàm F nhận vào ba đối số gồm hai từ dữ liệu vào R_0 và R_1, và số thứ tự r của chu kỳ dùng để lựa chọn các subkey thích hợp. R_0 được đưa qua hàm g để tạo ra T_0. R_1 được quay trái 8 bit, sau đó được đưa qua hàm g để sinh ra T_1. Kế đến, kết quả T_0 và T_1 được kết hợp sử dụng PHT và cộng thêm hai từ trong bảng khóa mở rộng.

$$\begin{aligned} T_0 &= g(R_0) \\ T_1 &= g(\mathrm{ROL}(R_1, 8)) \\ F_0 &= (T_0 + T_1 + K_{2r+8}) \bmod 2^{32} \\ F_1 &= (T_0 + 2T_1 + K_{2r+9}) \bmod 2^{32}, (F_0, F_1) \text{ là kết quả của } F. \end{aligned} \quad (5.29)$$

5.4.2.2 Hàm g

Hàm g là trung tâm của thuật toán Twofish. Từ dữ liệu vào X được chia thành 4 byte. Mỗi byte thực hiện thông qua `S-box` phụ thuộc khóa của chính mình. Mỗi `S-box` đưa 8 bit dữ liệu vào và đưa ra 8 bit kết quả. 4 byte kết quả được xem như một vector có chiều dài bằng 4 trên GF(2^8) và vector này nhân với ma trận MDS 4×4 (sử dụng vùng GF(2^8) cho việc tính toán). Vector kết quả được xem như một từ 32 bit và nó cũng là kết quả của hàm g.

$$\begin{aligned} x_i &= [X/2^{8i}] \bmod 2^8, \; i = 0, \ldots, 3 \\ y_i &= s_i[x_i], \; i = 0, \ldots, 3 \\ \begin{pmatrix} z_0 \\ z_1 \\ z_2 \\ z_3 \end{pmatrix} &= \begin{pmatrix} . & \cdots & . \\ \vdots & \mathrm{MDS} & \vdots \\ . & \cdots & . \end{pmatrix} . \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix} \\ Z &= \sum_{i=0}^{3} z_i . 2^{8i} \end{aligned} \quad (5.30)$$

với s_i là S-box phụ thuộc khóa và Z là kết quả của g. Để làm rõ vấn đề này, ta cần xác định rõ mối quan hệ giữa giá trị của mỗi byte với các phần tử của GF(2^8). Ta biểu diễn GF(2^8) dưới dạng GF(2)[x]/$v(x)$ với $v(x) = x^8 + x^6 + x^5 + x^3 + 1$ là đa thức cơ sở (primitive) bậc 8 trên GF(2). Phần tử $a = \sum_{i=0}^{7} a_i x^i$ với $a_i \in$ GF(2) ($i = 0, \ldots, k\text{-}1$) đồng nhất với giá trị byte $\sum_{i=0}^{7} a_i 2^i$.

Ta có ma trận MDS cho như sau:

$$\text{MDS} = \begin{pmatrix} 01 & EF & 5B & 5B \\ 5B & EF & EF & 01 \\ EF & 5B & 01 & EF \\ EF & 01 & EF & 5B \end{pmatrix} \tag{5.31}$$

ở đây các phần tử được viết dưới dạng giá trị byte hexa.

Ma trận này nhân một giá trị dữ liệu vào 32 bit với các hằng số 8 bit, tất cả các phép nhân này đều thực hiện trên GF(2^8). Đa thức $x^8 + x^6 + x^5 + x^3 + 1$ là đa thức cơ sở bậc 8 trên GF(2). Chỉ có 3 phép nhân khác nhau được sử dụng trong ma trận MDS là:

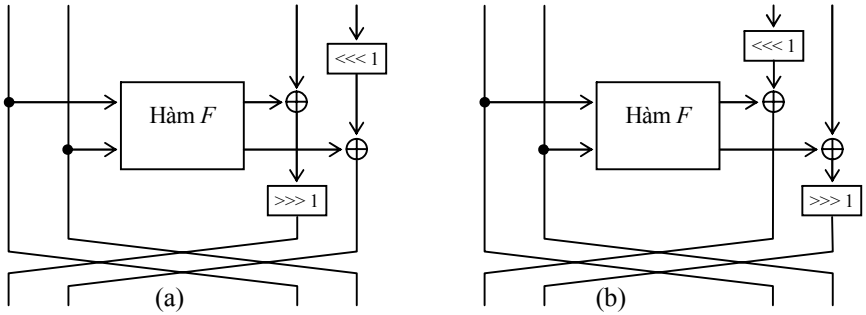
1. $5B_{16} = 0101\ 1011_2$ (thể hiện trên GF(2^8) bằng đa thức $x^6 + x^4 + x^3 + x + 1$

2. $EF_{16} = 1110\ 1111_2$ (thể hiện trên GF(2^8) bằng đa thức $x^7 + x^6 + x^5 + x^3 + x^2 + x + 1$

3. $01_{16} = 0000\ 0001_2$ (tương đương với phần tử trong GF(2^8) bằng 1)

5.4.3 Quy trình giải mã

Quy trình mã hóa và giải mã của thuật toán Twofish tương tự như nhau. Tuy nhiên, quy trình giải mã đòi hỏi áp dụng các subkey theo thứ tự đảo ngược và một số thay đổi nhỏ trong cấu trúc mã hóa (Xem Hình 5.18)



Hình 5.18. *So sánh quy trình mã hóa (a) và giải mã (b)*

5.5 Kết luận

Với bốn thuật toán trên quy trình mã hóa được thực hiện qua các giai đoạn chính: khởi tạo, phân bố khóa và mã hóa. Tương tự đối với giải mã cũng thực hiện qua các giai đoạn chính: khởi tạo, phân bố khóa và giải mã.

Quy trình khởi tạo và phân bố khóa được thực hiện dựa trên khóa người sử dụng cung cấp để phát sinh bộ subkey phục vụ cho việc mã hóa và giải mã.

Quy trình mã hóa được thực hiện đối với:

- MARS gồm ba giai đoạn: trộn tới (Forward mixing), Phần lõi chính (Cryptographic core) và trộn lùi (Backward mixing).
    - Giai đoạn trộn tới gồm phép toán cộng khóa và 8 chu kỳ trộn tới không dùng khóa.
    - Giai đoạn cốt lõi chính gồm 8 chu kỳ biến đổi tới có khóa và 8 chu kỳ biến đổi lùi có khóa.
    - Giai đoạn trộn lùi gồm 8 chu kỳ trộn lùi không dùng khóa và phép toán trừ khóa.
- RC6 gồm:
    - Phép cộng khóa đầu.
    - 20 chu kỳ.
    - Phép cộng khóa cuối.

- SERPENT gồm:
    - Phép hoán vị đầu IP (initial permutation).
    - 32 chu kỳ.
    - Phép hoán vị cuối FP (final permutation).

- TWOFISH gồm:
    - Input whitening.
    - 16 chu kỳ.
    - Output whitening.

Dữ liệu vào và ra quy trình mã hóa cũng như giải mã là khối dữ liệu 128 bit.

Tương quan giữa quy trình mã hóa và giải mã:

- Trong phương pháp MARS và RC6, hai quy trình này thực hiện tương tự nhau (theo thứ tự đảo ngược)
- Trong SERPENT, hai quy trình này khác nhau.
- Trong phương pháp TWOFISH, hai quy trình này gần như giống hệt nhau.

Chương 6
Một số hệ thống mã hóa khóa công cộng

✍ *Nội dung của chương 6 sẽ giới thiệu khái niệm về hệ thống mã hóa khóa công cộng. Phương pháp RSA nổi tiếng cũng được trình bày chi tiết trong chương này. Ở cuối chương là phần so sánh giữa hệ thống mã hóa quy ước và hệ thống mã hóa khóa công cộng cùng với mô hình kết hợp giữa hai hệ thống này.*

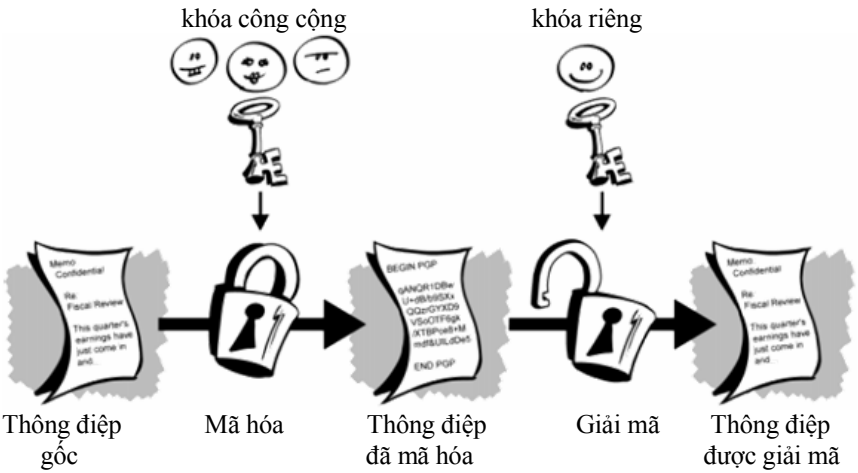
6.1 Hệ thống mã hóa khóa công cộng

Vấn đề phát sinh trong các hệ thống mã hóa quy ước là việc quy ước chung mã khóa *k* giữa người gửi A và người nhận B. Trên thực tế, nhu cầu thay đổi nội dung của mã khóa *k* là cần thiết, do đó, cần có sự trao đổi thông tin về mã khóa *k* giữa A và B. Để bảo mật mã khóa *k*, A và B phải trao đổi với nhau trên một kênh liên lạc thật sự an toàn và bí mật. Tuy nhiên, rất khó có thể bảo đảm được sự an toàn của kênh liên lạc nên mã khóa *k* vẫn có thể bị phát hiện bởi người C!

Ý tưởng về hệ thống mã hóa khóa công cộng được Martin Hellman, Ralph Merkle và Whitfield Diffie tại Đại học Stanford giới thiệu vào năm 1976. Sau đó,

phương pháp Diffie-Hellman của Martin Hellman và Whitfield Diffie đã được công bố [45]. Năm 1977, trên báo "*The Scientific American*", nhóm tác giả Ronald Rivest, Adi Shamir và Leonard Adleman đã công bố phương pháp RSA, phương pháp mã hóa khóa công cộng nổi tiếng và được sử dụng rất nhiều hiện nay trong các ứng dụng mã hóa và bảo vệ thông tin [39]. RSA nhanh chóng trở thành chuẩn mã hóa khóa công cộng trên toàn thế giới do tính an toàn và khả năng ứng dụng của nó.

Một hệ thống khóa công cộng sử dụng hai loại khóa trong cùng một cặp khóa: khóa công cộng (public key) được công bố rộng rãi và được sử dụng trong mã hóa thông tin, khóa riêng (private key) chỉ do một người nắm giữ và được sử dụng để giải mã thông tin đã được mã hóa bằng khóa công cộng. Các phương pháp mã hóa này 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. Chỉ khi biết được mã khóa riêng thì mới có thể thực hiện được ánh xạ ngược f^{-1}.



Hình 6.1. *Mô hình hệ thống mã hóa với khóa công cộng*

Khi áp dụng hệ thống mã hóa khóa công cộng, người A sử dụng mã khóa công cộng để mã hóa thông điệp và gửi cho người B. Do biết được mã khóa riêng nên B mới có thể giải mã thông điệp mà A đã mã hóa. Người C nếu phát hiện được thông điệp mà A gửi cho B, kết hợp với thông tin về mã khóa công cộng đã được công bố, cũng rất khó có khả năng giải mã được thông điệp này do không nắm được mã khóa riêng của B.

6.2 Phương pháp RSA

6.2.1 Phương pháp RSA

Năm 1978, R.L.Rivest, A.Shamir và L.Adleman đã đề xuất hệ thống mã hóa khóa công cộng RSA (hay còn được gọi là "hệ thống MIT"). Trong phương pháp này, tất cả các phép tính đều được thực hiện trên Z_n với n là tích của hai số nguyên tố lẻ p và q khác nhau. Khi đó, ta có $\phi(n) = (p-1)(q-1)$

Thuật toán 6.1. *Phương pháp mã hóa RSA*

$n = pq$ với p và q là hai số nguyên tố lẻ phân biệt.
Cho $P = C = \mathbb{Z}_n$ và định nghĩa:
$K = \{((n, p, q, a, b): n = pq, p, q$ là số nguyên tố, $ab \equiv 1 \pmod{\phi(n)}\}$
Với mỗi $k = (n, p, q, a, b) \in K$, định nghĩa:
$e_k(x) = x^b \bmod n$ và $d_k(y) = y^a \bmod n$, với $x, y \in \mathbb{Z}_n$
Giá trị n và b được công bố, trong khi giá trị p, q, a được giữ bí mật

Dựa trên định nghĩa phương pháp mã hóa RSA, việc áp dụng vào thực tế được tiến hành theo các bước sau:

Thuật toán 6.2. *Sử dụng phương pháp RSA*

Phát sinh hai số nguyên tố có giá trị lớn p và q Tính $n = pq$ và $\phi(n) = (p - 1)(q - 1)$ Chọn ngẫu nhiên một số nguyên b ($1 < b < \phi(n)$) thỏa $\gcd(b, \phi(n)) = 1$ Tính giá trị $a = b^{-1} \bmod \phi(n)$ (bằng thuật toán Euclide mở rộng) Giá trị n và b được công bố (khóa công cộng), trong khi giá trị p, q, a được giữ bí mật (khóa riêng)

6.2.2 Một số phương pháp tấn công giải thuật RSA

Tính chất an toàn của phương pháp RSA dựa trên cơ sở chi phí cho việc giải mã bất hợp lệ thông tin đã được mã hóa sẽ quá lớn nên xem như không thể thực hiện được.

Vì khóa là công cộng nên việc tấn công bẻ khóa phương pháp RSA thường dựa vào khóa công cộng để xác định được khóa riêng tương ứng. Điều quan trọng là dựa vào n để tính p, q của n, từ đó tính được d.

6.2.2.1 Phương pháp sử dụng $\phi(n)$

Giả sử người tấn công biết được giá trị $\phi(n)$. Khi đó việc xác định giá trị p, q được đưa về việc giải hai phương trình sau:

$$n = p \cdot q$$

$$\phi(n) = (p-1)(q-1) \tag{6.1}$$

Thay $q = n/p$, ta được phương trình bậc hai:

$$p^2 - (n - \phi(n) + 1)p + n = 0 \tag{6.2}$$

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

6.2.2.2 Thuật toán phân tích ra thừa số p-1

Thuật toán 6.3. *Thuật toán phân tích ra thừa số p-1*

Nhập n và B
1. $a = 2$
2. **for** $j = 2$ **to** B **do**
   $a = a^j \bmod n$
3. $d = \gcd(a - 1, n)$
4. **if** $1 < d < n$ **then**
   d là thừa số nguyên tố của n (thành công)
   else
   không xác định được thừa số nguyên tố của n (thất bại)

Thuật toán Pollard p-1 (1974) là một trong những thuật toán đơn giản hiệu quả dùng để phân tích ra thừa số nguyên tố các số nguyên lớn. Tham số đầu vào của thuật toán là số nguyên (lẻ) n cần được phân tích ra thừa số nguyên tố và giá trị giới hạn B.

Giả sử $n = p.q$ (p, q chưa biết) và B là một số nguyên đủ lớn, với mỗi thừa số nguyên tố k, $k \le B \wedge k \left| (p-1) \right. \Rightarrow (p-1) \left| B! \right.$

Ở cuối vòng lặp (bước 2), ta có

$$a \equiv 2^{B!} \pmod{n} \tag{6.3}$$

Suy ra

$$a \equiv 2^{B!} \pmod{p} \tag{6.4}$$

Do $p|n$ nên theo định lý Fermat, ta có :

$$2^{p-1} \equiv 1 \pmod{p} \tag{6.5}$$

Do $(p\text{-}1)|B!$, nên ở bước 3 của thuật toán, ta có:

$$a \equiv 1 \pmod{p}. \tag{6.6}$$

Vì thế, ở bước 4:

$$p|(a-1) \text{ và } p|n, \tag{6.7}$$

nên nếu $d = \gcd(a-1, n)$ thì $d = p$.

❒ Ví dụ: Giả sử n = 15770708441. Áp dụng thuật toán p – 1 với B = 180, chúng ta xác định được a = 11620221425 ở bước 3 của thuật toán và xác định được giá trị d = 135979. Trong trường hợp này, việc phân tích ra thừa số nguyên tố thành công do giá trị 135978 chỉ có các thừa số nguyên tố nhỏ khi phân tích ra thừa số nguyên tố:

$$135978 = 2 \times 3 \times 131 \times 173$$

Do đó, khi chọn $B \geq 173$ sẽ đảm bảo điều kiện $135978 \mid B!$

Trong thuật toán $p - 1$ có $B - 1$ phép tính lũy thừa modulo, mỗi phép đòi hỏi tối đa $2\log_2 B$ phép nhân modulo sử dụng thuật toán bình phương và nhân (xem 6.2.6 - Xử lý số học). Việc tính USCLN sử dụng thuật toán Euclide có độ phức tạp $O((\log n)^3)$. Như vậy, độ phức tạp của thuật toán là $O\left(B \log B (\log n)^2 + (\log n)^3\right)$

Tuy nhiên xác suất chọn giá trị B tương đối nhỏ và thỏa điều kiện $(p-1) \mid B!$ là rất thấp. Ngược lại, khi tăng giá trị B (chẳng hạn như $B \approx \sqrt{n}$) thì giải thuật sẽ thành công, nhưng thuật toán này sẽ không nhanh hơn giải thuật chia dần như trình bày trên.

Giải thuật này chỉ hiệu quả khi tấn công phương pháp RSA trong trường hợp n có thừa số nguyên tố p mà $(p - 1)$ chỉ có các ước số nguyên tố rất nhỏ. Do đó, chúng ta có thể dễ dàng xây dựng một hệ thống mã hóa khóa công cộng RSA an toàn đối với giải thuật tấn công $p - 1$. Cách đơn giản nhất là tìm một số nguyên tố p_1 lớn, mà $p = 2p_1 + 1$ cũng là số nguyên tố, tương tự tìm q_1 nguyên tố lớn và $q = 2q_1 + 1$ nguyên tố.

6.2.2.3 Bẻ khóa khi biết số mũ d của hàm giải mã

Việc tính ra được giá trị d không dễ dàng, bởi vì đây là khóa riêng nên nếu biết nó thì có thể giải mã được mọi đoạn tin tương ứng. Tuy nhiên giải thuật này mang ý nghĩa về mặt lý thuyết, nó cho chúng ta biết rằng nếu có d thì ta có thể tính các

thừa số của n. Nếu điều này xảy ra thì người sở hữu khóa này không thể thay đổi khóa công cộng, mà phải thay luôn số n.

Nhắc lại: phương trình $x^2 \equiv 1 \pmod{p}$ có hai nghiệm (modulo p) là $x = \pm 1 \bmod p$. Tương tự, phương trình $x^2 \equiv 1 \pmod{q}$ có hai nghiệm (modulo q) là $x = \pm 1 \bmod q$.

Do

$$x^2 \equiv 1 \pmod{n} \Leftrightarrow x^2 \equiv 1 \pmod{p} \wedge x^2 \equiv 1 \pmod{q} \tag{6.8}$$

nên ta có

$$x^2 \equiv 1 \pmod{n} \Leftrightarrow x = \pm 1 \pmod{p} \wedge x = \pm 1 \pmod{q} \tag{6.9}$$

Sử dụng lý thuyết số dư Trung Hoa, chúng ta có thể xác định được bốn căn bậc hai của 1 modulo n..

Nếu chọn được w là bội số của p hay q thì ở bước 2 của thuật toán, chúng ta có thể phân tích được n ra thừa số nguyên tố ngay. Nếu w nguyên tố cùng nhau với n, chúng ta tính $w^r, w^{2r}, w^{4r}, \ldots$ cho đến khi tồn tại t sao cho:

$$w^{2^t r} \equiv 1 \left(\bmod n\right) \tag{6.10}$$

Do $ab - 1 = 2^s r \equiv 0 \pmod{\phi(n)}$ nên $w^{2^s r} \equiv 1 \left(\bmod n\right)$. Vậy, vòng lặp *while* ở bước 8 của thuật toán thực hiện tối đa s lần lặp.

Sau khi thực hiện xong vòng lặp while, chúng ta tìm được giá trị v_0 thỏa $v_0^2 \equiv 1 \pmod{n}$ hay $v_0 \equiv \pm 1 \pmod{n}$. Nếu $v_0 \equiv -1 \pmod{n}$ thì thuật toán thất bại;

ngược lại, v_0 là căn bậc 2 không tầm thường của 1 modulo n và chúng ta có thể phân tích n ra thừa số nguyên tố.

Thuật toán 6.4. *Thuật toán phân tích ra thừa số nguyên tố, biết trước giá trị số mũ giải mã a*

Chọn ngẫu nhiên w thỏa $1 \le w \le n-1$
Tính $x = \gcd(w, n)$
if $1 < x < n$ **then**
    Chấm dứt thuật toán (thành công với $x = q$ hay $x = p$)
end if
Tính $a = A(b)$
Đặt $ab - 1 = 2^s r$ với r lẻ
Tính $v = w^r \bmod n$
if $v \equiv 1 \pmod n$ **then**
    Chấm dứt thuật toán (thất bại).
end if
while $v <> 1 \pmod n$ **do**
    $v_0 = v$
    $v = v^2 \bmod n$
    if $v_0 \equiv -1 \pmod n$ **then**
        Chấm dứt thuật toán (thất bại).
    else
        Tính $x = \gcd(v_0+1, n)$
        Chấm dứt thuật toán (thành công với $x = q$ hay $x = p$).
    end if
end while

6.2.2.4 Bẻ khóa dựa trên các tấn công lặp lại

Siimons và Norris đã chỉ ra rằng hệ thống RSA có thể bị tổn thương khi sử dụng tấn công lặp liên tiếp. Đó là khi đối thủ biết cặp khóa công cộng $\{n, b\}$ và từ khóa C thì anh ta có thể tính chuỗi các từ khóa sau:

$$C_1 = C^e \pmod{n}$$

$$C_2 = C_1^e \pmod{n}$$

$$\cdots$$

$$C_i = C_{i-1}^e \pmod{n} \qquad (6.11)$$

Nếu có một phần tử C_j trong chuỗi $C_1, C_2, C_{3,\ldots,} C_i$ sao cho $C_j = C$ thì khi đó anh ta sẽ tìm được $M = C_{j-1}$ bởi vì:

$$C_j = C_{j-1}^e \pmod{n}$$

$$C = M^e \pmod{n} \qquad (6.12)$$

❒ Ví dụ: Giả sử anh ta biết $\{n, b, C\}=\{35, 17, 3\}$,anh ta sẽ tính:

$$C_1 = C^e \pmod{n} = 3^{17} \pmod{35} = 33$$

$$C_2 = C_1^e \pmod{n} = 33^{17} \pmod{35} = 3$$

Vì $C_2 = C$ nên $M = C_1 = 33$

6.2.3 Sự che dấu thông tin trong hệ thống RSA

Hệ thống RSA có đặc điểm là thông tin không phải luôn được che dấu. Giả sử người gởi có $e = 17$, $n = 35$. Nếu anh ta muốn gởi bất cứ dữ liệu nào thuộc tập sau:

{1, 6, 7, 8, 13, 14, 15, 20, 21, 22, 27, 28, 29, 34}

thì kết quả của việc mã hóa lại chính là dữ liệu ban đầu. Nghĩa là, $M = M^e \bmod n$.

Còn khi $p = 109$, $q = 97$, $e = 865$ thì hệ thống hoàn toàn không có sự che dấu thông tin, bởi vì:

$$\forall M,\ M = M^{865} \bmod (109*97),$$

Với mỗi giá trị n, có ít nhất 9 trường hợp kết quả mã hóa chính là dữ liệu nguồn ban đầu. Thật vậy,

$$M = M^e \bmod n \tag{6.1}$$

hay:

$$M = M^e \bmod p \text{ và } M = M^e \bmod q \tag{6.2}$$

Với mỗi e, (6.2) có ít nhất ba giải pháp thuộc tập {0, 1, -1}. Để xác định chính xác số thông điệp không được che dấu (không bị thay đổi sau khi mã hóa) ta sử dụng định lý sau: "Nếu các thông điệp được mã hóa trong hệ thống RSA được xác định bởi số modulus $n = p.q$ (p,q là số nguyên tố) và khóa công cộng e thì có:

$m = [1+\gcd(e-1, p-1)][1+\gcd(e-1), q-1]$ thông điệp không bị che dấu.

Mấu chốt để có thể giải mã được thông tin là có được giá trị p và q tạo nên giá trị n. Khi có được hai giá trị này, ta có thể dễ dàng tính ra được $\phi(n) = (p - 1)(q - 1)$ và giá trị $a = b^{-1} \bmod \phi(n)$ theo thuật toán Euclide mở rộng. Nếu số nguyên n có thể được phân tích ra thừa số nguyên tố, tức là giá trị p và q có thể được xác định thì xem như tính an toàn của phương pháp RSA không còn được bảo đảm nữa. Như vậy, tính an toàn của phương pháp RSA dựa trên cơ sở các máy tính tại thời điểm hiện tại chưa đủ khả năng giải quyết việc phân tích các số nguyên rất lớn ra thừa số nguyên tố. Tuy nhiên, với sự phát triển ngày càng nhanh chóng của máy tính cũng như những bước đột phá trong lĩnh vực toán học, phương pháp RSA sẽ gặp phải những khó khăn trong việc bảo mật thông tin. Năm 1994, Peter Shor, một nhà khoa học tại phòng thí nghiệm AT&T, đã đưa ra một thuật toán có thể phân tích một cách hiệu quả các số nguyên rất lớn trên máy tính lượng tử. Mặc dù máy tính lượng tử hiện chưa thể chế tạo được nhưng rõ ràng phương pháp RSA sẽ gặp phải nhiều thách thức lớn trong tương lai.

6.2.4 Vấn đề số nguyên tố

Để bảo đảm an toàn cho hệ thống mã hóa RSA, số nguyên $n = pq$ phải đủ lớn để không thể dễ dàng tiến hành việc phân tích n ra thừa số nguyên tố. Hiện tại, các thuật toán phân tích thừa số nguyên tố đã có thể giải quyết được các số nguyên có trên 130 chữ số (thập phân). Để an toàn, số nguyên tố p và q cần phải đủ lớn, ví dụ như trên 100 chữ số. Vấn đề đặt ra ở đây là giải quyết bài toán: làm thế nào để kiểm tra một cách nhanh chóng và chính xác một số nguyên dương n là số nguyên tố hay hợp số?

Theo định nghĩa, một số nguyên dương n là số nguyên tố khi và chỉ khi n chỉ chia hết cho 1 và n (ở đây chỉ xét các số nguyên dương). Từ đó suy ra, n là số nguyên

tố khi và chỉ khi *n* không có ước số dương nào thuộc đoạn $\left[2,...,\left[\sqrt{n}\right]\right]$. Như vậy, ta có: *n* là số nguyên tố $\Leftrightarrow \forall i \in \left[2,...,\left[\sqrt{n}\right]\right], \neg\left(n \equiv 0\left(\bmod i\right)\right)$

Việc kiểm tra một số nguyên dương *n* là số nguyên tố theo phương pháp trên sẽ đưa ra kết quả hoàn toàn chính xác. Tuy nhiên, thời gian xử lý của thuật toán rõ ràng là rất lớn, hoặc thậm chí không thể thực hiện được, trong trường hợp *n* tương đối lớn.

6.2.5 Thuật toán Miller-Rabin

Trên thực tế, việc kiểm tra một số nguyên dương *n* là số nguyên tố thường áp dụng các phương pháp thuộc nhóm thuật toán Monte Carlo, ví dụ như thuật toán Solovay-Strassen hay thuật toán Miller-Robin; trong đó, thuật toán Miller-Robin thường được sử dụng phổ biến hơn. Các thuật toán này đều có ưu điểm là xử lý nhanh chóng (số nguyên dương *n* có thể được kiểm tra trong thời gian tỉ lệ với $\log_2 n$, tức là số lượng các bit trong biểu diễn nhị phân của *n*) nhưng vẫn có khả năng là kết luận của thuật toán không hoàn toàn chính xác, nghĩa là có khả năng một hợp số *n* lại được kết luận là số nguyên tố, mặc dù xác suất xảy ra kết luận không chính xác là không cao. Tuy nhiên, vấn đề này có thể được khắc phục bằng cách thực hiện thuật toán một số lần đủ lớn, ta có thể làm giảm khả năng xảy ra kết luận sai xuống dưới một ngưỡng cho phép và khi đó, xem như kết luận có độ tin cậy rất cao.

Định nghĩa 6.1: *Thuật toán thuộc nhóm Monte Carlo được sử dụng trong việc khẳng định hay phủ định một vấn đề nào đó. Thuật toán luôn đưa ra câu trả lời và câu trả lời thu được chỉ có khả năng hoặc là "Có" (yes) hoặc là "Không" (no).*

Định nghĩa 6.2: *Thuật toán "yes-biased Monte Carlo" là thuật toán Monte Carlo, trong đó, câu trả lời "Có" (Yes) luôn chính xác nhưng câu trả lời "Không" (No) có thể không chính xác.*

Thuật toán 6.5. *Thuật toán Miller-Rabin*

Phân tích số nguyên dương p dưới dạng $n = 2^k m + 1$ với m lẻ
Chọn ngẫu nhiên số nguyên dương $a \in \{1, 2, ..., n\text{-}1\}$
Tính $b = a^m \bmod p$
if $b \equiv 1 \pmod p$ **then**
    Kết luận "p là số nguyên tố" và dừng thuật toán
end if
for $i = 0$ **to** $k - 1$
    if $b \equiv p - 1 \pmod p$ **then**
        Kết luận "p là số nguyên tố" và dừng thuật toán
    else
        $b = b^2 \bmod p$
    end if
end for
Kết luận "p là hợp số"

Thuật toán Miller-Rabin là thuật toán "yes-biased Monte Carlo" đối với vị từ "số nguyên dương n là hợp số". Xác suất xảy ra kết luận sai, nghĩa là thuật toán đưa ra kết luận "n là số nguyên tố" khi n thật sự là hợp số, chỉ tối đa là 25%. Nếu áp dụng thuật toán k lần với các giá trị a khác nhau mà ta vẫn thu được kết luận "n là số nguyên tố" thì xác suất chính xác của kết luận này là $1 - \frac{1}{4^k} \to 1$, với k đủ lớn.

6.2.6 Xử lý số học

Trong phương pháp mã hóa RSA, nhu cầu tính giá trị của biểu thức $z = x^b \bmod n$ được đặt ra trong cả thao tác mã hóa và giải mã. Nếu thực hiện việc tính giá trị theo cách thông thường thì rõ ràng là không hiệu quả do thời gian xử lý quá lớn.

Thuật toán "bình phương và nhân" (square-and-multiply) có thể được sử dụng để tính giá trị biểu thức $z = x^b \bmod n$ một cách nhanh chóng và hiệu quả

Thuật toán 6.6. *Thuật toán "bình phương và nhân" để tính giá trị*

$$z = x^b \bmod n$$

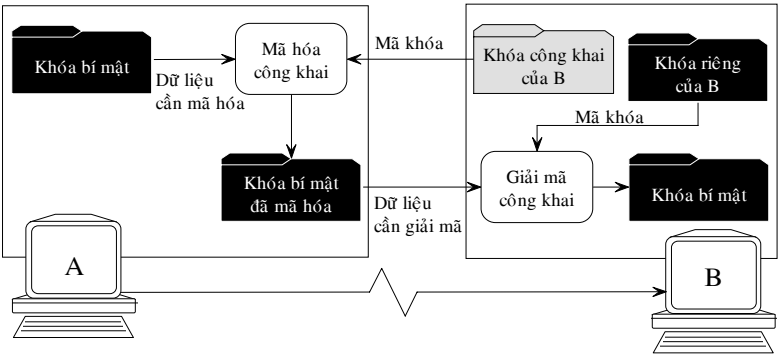
Biểu diễn b dưới dạng nhị phân $b_{l-1}b_{l-2}...b_1b_0$, $b_i \in \{0, 1\}$, $0 \le i < l$
$z = 1$
$x = x \bmod n$
for $i = l$-1 **downto** 0
    $z = z^2 \bmod n$
    if $b_i = 1$ **then**
        $z = z \times x \bmod n$
    end if
end for

6.3 Mã hóa quy ước và mã hóa khóa công cộng

Các phương pháp mã hóa quy ước có ưu điểm xử lý rất nhanh so với các phương pháp mã hóa khóa công cộng. Do khóa dùng để mã hóa cũng được dùng để giải mã nên cần phải giữ bí mật nội dung của khóa và mã khóa được gọi là khóa bí

mật (secret key). Ngay cả trong trường hợp khóa được trao đổi trực tiếp thì mã khóa này vẫn có khả năng bị phát hiện. Vấn đề khó khăn đặt ra đối với các phương pháp mã hóa này chính là bài toán trao đổi mã khóa.

Ngược lại, các phương pháp mã hóa khóa công cộng giúp cho việc trao đổi mã khóa trở nên dễ dàng hơn. Nội dung của khóa công cộng (public key) không cần phải giữ bí mật như đối với khóa bí mật trong các phương pháp mã hóa quy ước. Sử dụng khóa công cộng, mã khóa bí mật có thể được trao đổi an toàn theo quy trình trong Hình 6.2.



Hình 6.2. *Quy trình trao đổi khóa bí mật sử dụng khóa công cộng*

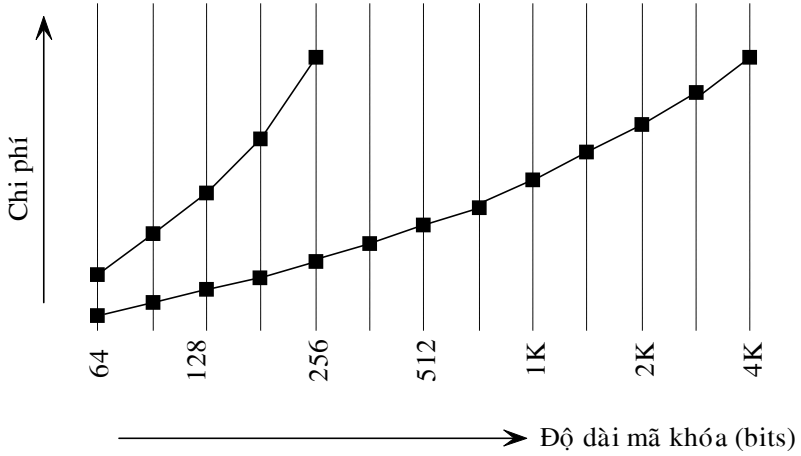
Vấn đề còn lại đối với khóa công cộng là làm cách nào xác nhận được chính xác người chủ thật sự của một khóa công cộng (xem Chương 10).

Dựa vào Bảng 6.1, chúng ta có thể nhận thấy rằng để có được mức độ an toàn tương đương với một phương pháp mã hóa quy ước, một phương pháp mã hóa

khóa công cộng phải sử dụng mã khóa có độ dài lớn hơn nhiều lần mã khóa bí mật được sử dụng trong mã hóa quy ước. Điều này được thể hiện rõ hơn qua đồ thị so sánh chi phí cần thiết để công phá khóa bí mật và khóa công cộng trong Hình 6.3. Kích thước mã khóa được tính dựa trên mô hình đánh giá, ước lượng chi phí phân tích mật mã do Hội đồng Nghiên cứu Quốc gia Hoa Kỳ (National Research Council) đề nghị [43].

Bảng 6.1. *So sánh độ an toàn giữa khóa bí mật và khóa công cộng*

Phương pháp mã hóa quy ước		Phương pháp mã hóa khóa công cộng	
Kích thước mã khóa (bit)	Thuật toán	Kích thước mã khóa (bit)	Ứng dụng
56	DES	256	
70		384	Phiên bản PGP cũ (kích thước tối thiểu)
80	SKIPJACK	512	Short DSS, PGP "low grade"
96		768	PGP "high grade"
112	3DES với 2 khóa	1024	Long DSS, PGP "military grade"
128	IDEA, AES	1440	
150		2047	PGP "alien grade"
168	3DES với 3 khóa	2880	
192	AES	3000	
256	AES	4096	



Hình 6.3. *Đồ thị so sánh chi phí công phá khóa bí mật và khóa công cộng*

Trên thực tế, khóa công cộng dễ bị tấn công hơn khóa bí mật. Để tìm ra được khóa bí mật, người giải mã cần phải có thêm một số thông tin liên quan đến các đặc tính của văn bản nguồn trước khi mã hóa để tìm ra manh mối giải mã thay vì phải sử dụng phương pháp vét cạn mã khóa. Ngoài ra, việc xác định xem thông điệp sau khi giải mã có đúng là thông điệp ban đầu trước khi mã hóa hay không lại là một vấn đề khó khăn. Ngược lại, đối với các khóa công cộng, việc công phá hoàn toàn có thể thực hiện được với điều kiện có đủ tài nguyên và thời gian xử lý. Ngoài ra, để có thể giải mã một thông điệp sử dụng phương pháp mã hóa khóa công cộng, người giải mã cũng không cần phải vét cạn toàn bộ không gian mã khóa mà chỉ cần khảo sát trên tập con của không gian này.

Bên cạnh đó, khóa công cộng còn là mục tiêu tấn công đáng giá đối với những người giải mã hơn các khóa bí mật. Khóa công cộng thường dùng để mã hóa các khóa bí mật khi thực hiện việc trao đổi mã khóa bí mật. Nếu khóa công cộng bị phá thì các thông điệp sau đó sử dụng mã khóa này cũng bị giải mã. Trong khi đó, nếu chỉ phát hiện được một mã khóa bí mật thì chỉ có thông điệp sử dụng mã khóa này mới bị giải mã. Trên thực tế, mã khóa bí mật thường chỉ được sử dụng một lần nên ít có giá trị hơn so với khóa công cộng. Tóm lại, mặc dù khóa công cộng được dùng để mã hóa các thông tin ngắn nhưng đây lại là các thông tin quan trọng.

Chương 7
Chữ ký điện tử

✍ *Nội dung của chương 7 sẽ giới thiệu khái niệm về chữ ký điện tử cùng với một số phương pháp chữ ký điện tử phổ biến hiện nay như RSA, ElGamal và DSS*

7.1 Giới thiệu

Chữ ký điện tử không được sử dụng nhằm bảo mật thông tin mà nhằm bảo vệ thông tin không bị người khác cố tình thay đổi để tạo ra thông tin sai lệch. Nói cách khác, chữ ký điện tử giúp xác định được người đã tạo ra hay chịu trách nhiệm đối với một thông điệp.

Một phương pháp chữ ký điện tử bao gồm hai thành phần chính: thuật toán dùng để tạo ra chữ ký điện tử và thuật toán tương ứng để xác nhận chữ ký điện tử.

Định nghĩa 7.1: *Một phương pháp chữ ký điện tử được định nghĩa là một bộ-năm (P, A, K, S, V) thỏa các điều kiện sau:*

1. *P là tập hợp hữu hạn các thông điệp.*

2. *A là tập hợp hữu hạn các chữ ký có thể được sử dụng.*

3. *Không gian khóa K là tập hợp hữu hạn các khóa có thể sử dụng.*

4. *Với mỗi khóa $k \in K$, tồn tại thuật toán chữ ký $sig_k \in S$ và thuật toán xác nhận chữ ký tương ứng $ver_k \in V$. Mỗi thuật toán $sig_k : \mathrm{P} \to \mathrm{A}$ và $ver_k : \mathrm{P} \times \mathrm{A} \to \{true, false\}$ là các hàm thỏa điều kiện:*

$$\forall x \in P, \forall y \in A : ver(x, y) = \begin{cases} true & \text{nếu } y = sig(x) \\ false & \text{nếu } y \neq sig(x) \end{cases} \tag{7.1}$$

7.2 Phương pháp chữ ký điện tử RSA

Phương pháp chữ ký điện tử RSA được xây dựng dựa theo phương pháp mã hóa khóa công cộng RSA.

Thuật toán 7.1. *Phương pháp chữ ký điện tử RSA*

$n = pq$ với p và q là hai số nguyên tố lẻ phân biệt.

Cho $P = C = \mathbb{Z}_n$ và định nghĩa:

$K = \{(n, p, q, a, b): n = pq, p, q \text{ là số nguyên tố}, ab \equiv 1 \pmod{\phi(n)}\}$

Giá trị n và b được công bố, trong khi giá trị p, q, a được giữ bí mật.

Với mỗi $K = (n, p, q, a, b) \in K$, định nghĩa:

$$sig_K(x) = x^a \bmod n$$

và

$$ver_K(x, y) = \text{true} \Leftrightarrow x \equiv y^b \pmod{n}, \text{ với } x, y \in \mathbb{Z}_n$$

7.3 Phương pháp chữ ký điện tử ElGamal

Phương pháp chữ ký điện tử ElGamal được giới thiệu vào năm 1985. Sau đó, Viện Tiêu chuẩn và Công nghệ Quốc gia Hoa Kỳ (NIST) đã sửa đổi bổ sung phương pháp này thành chuẩn chữ ký điện tử (Digital Signature Standard– DSS). Khác với phương pháp RSA có thể áp dụng trong mã hóa khóa công cộng và chữ ký điện tử, phương pháp ElGamal được xây dựng chỉ nhằm giải quyết bài toán chữ ký điện tử.

7.3.1 Bài toán logarit rời rạc

Phát biểu bài toán logarit rời rạc: Cho số nguyên tố p, gọi $\alpha \in Z_p$ là phần tử sinh (generator) và $\beta \in Z_p$*. Cần xác định số nguyên dương $a \in Z_{p-1}$ sao cho

$$\alpha^a \equiv \beta \,(\mathrm{mod}\, p) \tag{7.2}$$

Khi đó, a được ký hiệu là $\log_\alpha \beta$.

Trên thực tế, bài toán logarit rời rạc thuộc nhóm *NP* hay nói cách khác, chưa có thuật toán có thời gian đa thức nào có thể giải quyết được vấn đề này. Với p có tối thiểu 150 chữ số và $p - 1$ có thừa số nguyên tố đủ lớn, phép toán lũy thừa modulo p có thể xem như là hàm 1 chiều hay việc giải bài toán logarit rời rạc trên Z_p xem như không thể thực hiện được.

7.3.2 Phương pháp ElGamal

Trong phương pháp ElGamal, một thông điệp bất kỳ có thể có nhiều chữ ký hợp lệ khác nhau.

Thuật toán 7.2. *Phương pháp chữ ký điện tử ElGamal*

Cho p là số nguyên tố sao cho việc giải bài toán logarit rời rạc trên Z_p xem như không thể thực hiện được. Cho $\alpha \in Z_p^*$ là phần tử sinh.

Cho $P = Z_p^*$, $A = Z_p^* \times Z_{p-1}$ và định nghĩa

$$K = \{ (p, \alpha, a, \beta): \beta \equiv \alpha^a \pmod p \}$$

Giá trị p, α và β được công bố, trong khi giá trị a được giữ bí mật.

Với mỗi $K = (p, \alpha, a, \beta) \in K$ và một số ngẫu nhiên (được giữ bí mật) $k \in Z_{p-1}^*$, định nghĩa:

$$sig_K(x,k) = (\gamma, \delta)$$

với

$$\gamma = \alpha^k \bmod p$$

và

$$\delta = (x - a\gamma)\, k^{-1} \bmod (p-1)$$

Với x, $\gamma \in Z_p^*$ và $\delta \in Z_{p-1}$, định nghĩa

$$ver_K(x, \gamma, \delta) = \text{true} \Leftrightarrow \beta^\gamma \gamma^\delta \equiv \alpha^x \pmod p$$

7.4 Phương pháp Digital Signature Standard

Phương pháp *Digital Signature Standard* (DSS) là sự cải tiến của phương pháp ElGamal. Phương pháp này được công bố trên Federal Register vào ngày 19

tháng 5 năm 1994 và chính thức trở thành phương pháp chuẩn từ ngày 1 tháng 12 năm 1994.

Thuật toán 7.3. *Phương pháp Digital Sinature Standard*

Cho p là số nguyên tố 512-bit sao cho việc giải bài toán logarit rời rạc trên Z_p xem như không thể thực hiện được và q là số nguyên tố 160-bit là ước số của $p - 1$. Cho $\alpha \in Z_p^*$ là căn bậc q của 1 modulo p.

Cho $P = Z_q^*$, $A = Z_q \times Z_q$ và định nghĩa

$$K = \{ (p, q, \alpha, a, \beta): \beta \equiv \alpha^a \pmod p \}$$

Giá trị p, q, α và β được công bố, trong khi giá trị a được giữ bí mật.

Với mỗi $K = (p, \alpha, a, \beta) \in K$ và một số ngẫu nhiên (được giữ bí mật) $k \in Z_q^*$, định nghĩa:

$$sig_K(x,k) = (\gamma, \delta)$$

với

$$\gamma = (\alpha^k \bmod p) \bmod q$$

và

$$\delta = (x + a\gamma)\, k^{-1} \bmod q$$

Với $x \in Z_q^*$ và $\gamma, \delta \in Z_q$, định nghĩa

$$ver_K(x,\gamma,\delta) = true \Leftrightarrow \left(\alpha^{e_1} \beta^{e_2} \bmod p\right) \bmod q = \gamma$$

với $$e_1 = x\delta^{-1} \bmod q \text{ và } e_2 = \gamma\delta^{-1} \bmod q$$

Một văn bản điện tử, ví dụ như các hợp đồng kinh tế hay di chúc thừa kế, có thể cần được kiểm tra để xác nhận chữ ký nhiều lần sau một khoảng thời gian dài nên vấn đề an toàn đối với chữ ký điện tử cần phải được quan tâm nhiều hơn. Do mức độ an toàn của phương pháp ElGamal phụ thuộc vào độ phức tạp của việc tìm lời

giải cho bài toán logarit rời rạc nên cần thiết phải sử dụng số nguyên tố p đủ lớn (tối thiểu là 512-bit [43]). Nếu sử dụng số nguyên tố p có 512 bit thì chữ ký điện tử được tạo ra sẽ có độ dài 1024-bit và không phù hợp với các ứng dụng sử dụng thẻ thông minh vốn có nhu cầu sử dụng chữ ký ngắn hơn. Phương pháp DSS đã giải quyết vấn đề này bằng cách dùng chữ ký điện tử 320-bit trên văn bản 160-bit với các phép tính toán đều được thực hiện trên tập con có 2^{160} phần tử của Z_p* với p là số nguyên tố 512-bit.

Chương 8
Phương pháp ECC

✍ *Trong chương 6 và 7, chúng ta đã tìm hiểu về về khái niệm và một số phương pháp cụ thể phổ biến trong hệ thống mã hóa khóa công cộng và chữ ký điện tử. Trong chương này, chúng ta sẽ tìm hiểu về việc ứng dụng lý thuyết toán học đường cong elliptic (elliptic curve) trên trường hữu hạn vào hệ thống mã hóa khóa công cộng.*

8.1 Lý thuyết đường cong elliptic

Hệ thống mã hóa khóa công cộng dựa trên việc sử dụng các bài toán khó giải quyết. Vấn đề khó ở đây chính là việc số lượng phép tính cần thiết để tìm ra một lời giải cho bài toán là rất lớn. Trong lịch sử 20 năm của ngành mã hóa bất đối xứng đã có nhiều đề xuất khác nhau cho dạng bài toán như vậy, tuy nhiên chỉ có hai trong số các đề xuất đó còn tồn tại vững đến ngày này. Hai bài toán đó bao gồm: bài toán logarit rời rạc (discrete logarithm problem) và bài toán phân tích thừa số của số nguyên.

Cho đến năm 1985, hai nhà khoa học Neal Koblitz và Victor S. Miller đã độc lập nghiên cứu và đưa ra đề xuất ứng dụng lý thuyết toán học đường cong elliptic (elliptic curve) trên trường hữu hạn [35].

Đường cong elliptic – cũng như đại số hình học – được nghiên cứu rộng rãi trong vòng 150 năm trở lại đây và đã đạt được một số kết quả lý thuyết có giá trị. Đường cong elliptic được phát hiện lần đầu vào thế kỷ 17 dưới dạng công thức Diophantine: $y^2 - x^3 = c$ với $c \in \mathbb{Z}$.

Tính bảo mật của hệ thống mã hóa sử dụng đường cong elliptic dựa trên điểm mấu chốt là độ phức tạp của bài toán logarit rời rạc trong hệ thống đại số. Trong suốt 10 năm gần đây, bài toán này nhận được sự quan tâm chú ý rộng rãi của các nhà toán học hàng đầu trên thế giới. Không giống như bài toán logarit rời rạc trên trường hữu hạn hoặc bài toán phân tích thừa số của số nguyên, bài toán logarit rời rạc trên đường cong elliptic chưa có thuật toán nào có thời gian thực hiện nhỏ hơn cấp lũy thừa. Thuật toán tốt nhất được biết cho đến hôm nay tốn thời gian thực hiện cấp lũy thừa [27].

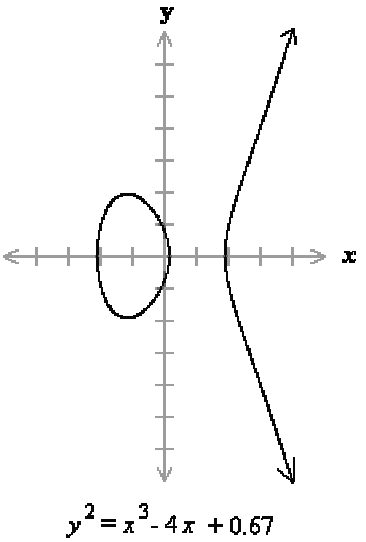
8.1.1 Công thức Weierstrasse và đường cong elliptic

Gọi K là một trường hữu hạn hoặc vô hạn. Một đường cong elliptic được định nghĩa trên trường K bằng công thức Weierstrass:

$$y^2 + a_1 xy + a_3 y = x^3 + a_2 x^2 + a_4 x + a_6 \tag{8.1}$$

trong đó $a_1, a_2, a_3, a_4, a_5, a_6 \in K$.

Đường cong elliptic trên trường K được ký hiệu $E(K)$. Số lượng các điểm nguyên trên E ký hiệu là $\#E(K)$, có khi chỉ đơn giản là $\#E$. Đối với từng trường khác nhau, công thức Weierstrass có thể được biến đổi và đơn giản hóa thành các dạng khác nhau. Một đường cong elliptic là tập hợp các điểm thỏa công thức trên.



Hình 8.1. *Một ví dụ về đường cong elliptic*

8.1.2 Đường cong elliptic trên trường R^2

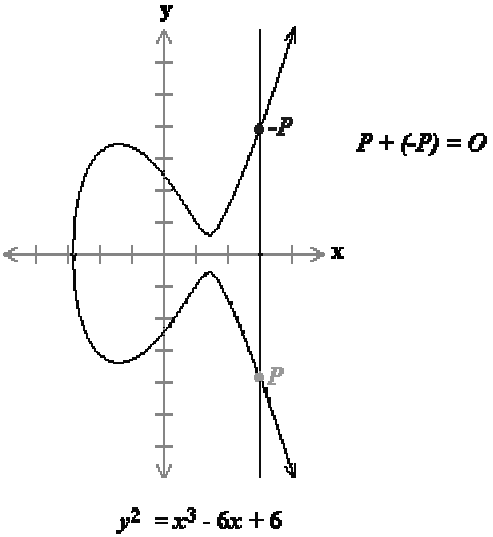
Đường cong elliptic E trên trường số thực R là tập hợp các điểm (x, y) thoả mãn công thức:

$$y^2 = x^3 + a_4x + a_6 \text{ với } a_4, a_6 \in R \tag{8.2}$$

cùng với một điểm đặc biệt O được gọi là điểm tại vô cực (cũng là phần tử identity). Cặp giá trị (x, y) đại diện cho một điểm trên đường cong elliptic và tạo

nên mặt phẳng tọa độ hai chiều (affine) $R \times R$. Đường cong elliptic E trên R^2 được gọi là định nghĩa trên R, ký hiệu là $E(R)$. Đường cong elliptic trên số thực có thể dùng để thể hiện một nhóm $(E(R), +)$ bao gồm tập hợp các điểm $(x, y) \in R \times R$ với phép cộng + trên $E(R)$.

8.1.2.1 Phép cộng



Hình 8.2. *Điểm ở vô cực*

Phép cộng điểm (ESUM) được định nghĩa trên tập $E(R)$ của các điểm (x, y). Điểm tại vô cực O là điểm cộng với bất kỳ điểm nào cũng sẽ ra chính điểm đó.

Như vậy,. $\forall P(x,y) \in E(R)$, $P + O = O + P = P$:

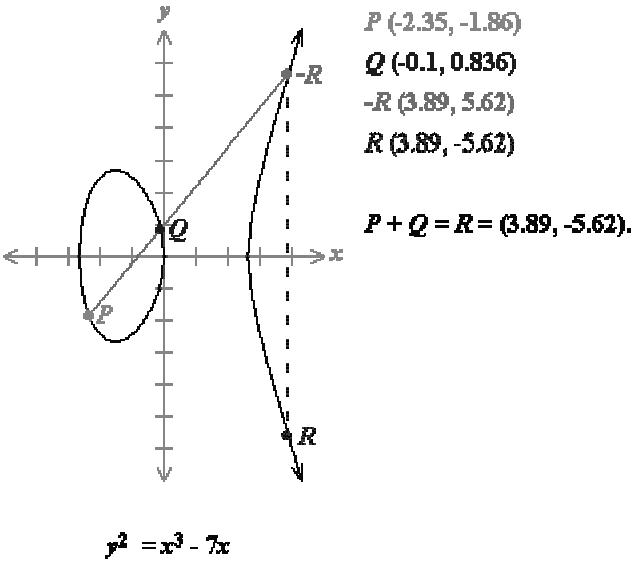
$$\forall P(x,y) \in E(R): \pm y = \sqrt{x^3 + a_4 x + a_6} \tag{8.3}$$

Như vậy, tương ứng với một giá trị x ta sẽ có hai giá trị tọa độ y.

Điểm $(x, –y)$ ký hiệu là $–P \in E(R)$, được gọi là điểm đối của P với:

$$P + (–P) = (x, y) + (x, –y) = O \tag{8.4}$$

Phép cộng trên $E(R)$ được định nghĩa theo phương diện hình học. Giả sử có hai điểm phân biệt P và Q, $P, Q \in E(R)$. Phép cộng trên nhóm đường cong elliptic là $P + Q = R, R \in E(R)$.



Hình 8.3. *Phép cộng trên đường cong elliptic*

Để tìm điểm R, ta nối P và Q bằng đường thẳng L. Đường thẳng L sẽ cắt E tại ba điểm P, Q và $–R(x, y)$. Điểm $R(x, –y)$ sẽ có tung độ là giá trị đối của y.

Thể hiện phép cộng đường cong elliptic dưới dạng đại số, ta có:

$$P = (x_1, y_1)$$

$$Q = (x_2, y_2) \tag{8.5}$$

$$R = P + Q = (x_3, y_3)$$

trong đó $P, Q, R \in E(R)$ và:

$$x_3 = \theta^2 - x_1 - x_2$$

$$y_3 = \theta(x_1 + x_3) - y_1 \tag{8.6}$$

$$\theta = \frac{y_2 - y_1}{x_2 - x_1} \text{ nếu } P \neq Q \tag{8.7}$$

hoặc $$\theta = \frac{3x_1^2 + a_4}{2y_1} \text{ nếu } P = Q \tag{8.8}$$

Thuật toán cộng trên đường cong elliptic được thể hiện như sau:

Thuật toán 8.1: *Thuật toán cộng điểm trên đường cong elliptic*

Input:

Đường cong elliptic $E(R)$với các tham số $a_4, a_6 \in E(R)$,

Điểm $P = (x_1, y_1) \in E(R)$ và $Q = (x_2, y_2) \in E(R)$

Output: $R = P + Q$, $R = (x_3, y_3) \in E(R)$

If $P = O$ **then** $R \leftarrow Q$ và trả về giá trị R

If $Q = O$ **then** $R \leftarrow P$ và trả về giá trị R

If $x_1 = x_2$ **then**

If $y_1 = y_2$ **then**

$$\theta \leftarrow \frac{3x_1^2 + a_4}{2y_1}$$

else if $y_1 = -y_2$ **then**

$R \leftarrow O$ và trả về R,

else

$$\theta \leftarrow \frac{y_2 - y_1}{x_2 - x_1}$$

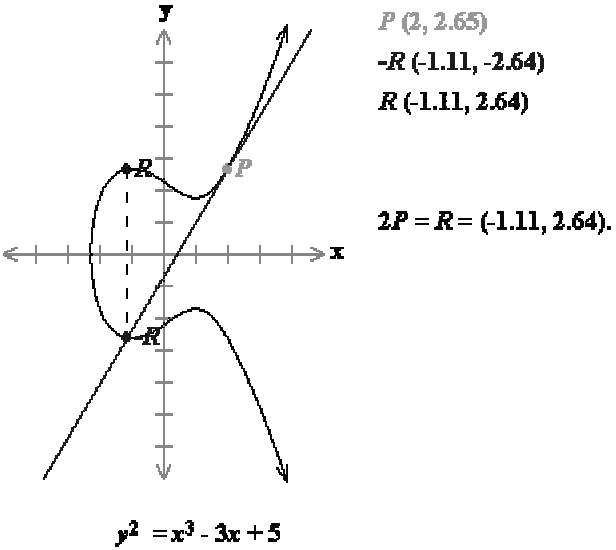
end if

$x_3 = \theta^2 - x_1 - x_2$

$y_3 = \theta(x_1 + x_3) - y_1$

Trả về $(x_3, y_3) = R$

8.1.2.2 Phép nhân đôi



Hình 8.4. *Phép nhân đôi trên đường cong elliptic*

Xét phép nhân đôi (EDBL): nếu cộng hai điểm $P, Q \in E(R)$ với $P = Q$ thì đường thẳng L sẽ là tiếp tuyến của đường cong elliptic tại điểm P. Trường hợp này điểm $-R$ sẽ là giao điểm còn lại của L với E. Lúc đó $R = 2P$.

8.1.3 Đường cong elliptic trên trường hữu hạn

Đường cong elliptic được xây dựng trên các trường hữu hạn. Có hai trường hữu hạn thường được sử dụng: trường hữu hạn F_q với q là số nguyên tố hoặc q là 2^m (m là số nguyên).

Tùy thuộc vào trường hữu hạn F_q, với mỗi bậc của q, tồn tại nhiều đường cong elliptic. Do đó, với một trường hữu hạn cố định có q phần tử và q lớn, có nhiều sự lựa chọn nhóm đường cong elliptic.

8.1.3.1 Đường cong elliptic trên trường F_p (p là số nguyên tố)

Cho p là số nguyên tố ($p > 3$), Cho $a, b \in F_p$ sao cho $4a^3 + 27b^2 \neq 0$ trong trường F_p. Một đường cong elliptic $E(F_p)$ trên F_p (được định nghĩa bởi các tham số a và b) là một tập hợp các cặp giá trị (x, y) $(x, y \in F_p)$ thỏa công thức

$$y^2 = x^3 + ax + b \tag{8.9}$$

cùng với một điểm O – gọi là điểm tại vô cực. Số lượng điểm của $E(F_p)$ là $\#E(F_p)$ thỏa định lý Hasse:

$$p + 1 - 2\sqrt{p} \leq \#E(F_p) \leq p + 1 + 2\sqrt{p} \tag{8.10}$$

Các phép toán của đường cong elliptic trên F_p cũng tương tự với $E(R)$. Tập hợp các điểm trên $E(F_p)$ tạo thành một nhóm thỏa các tính chất sau:

- Tính đóng: $\forall\ a, b \in G, a + b \in G$.
- Tính kết hợp: Các phép toán trong nhóm có tính kết hợp. Do đó, $(a + b) + c = a + (b + c)$.
- Phần tử trung hòa: có một giá trị $0 \in G$ sao cho $a + 0 = 0 + a = a$, $\forall\ a \in G$.
- Phần tử đối: $\forall a \in G, \exists -a \in G$ gọi là số đối của a, sao cho $-a + a = a + (-a) = 0$.

Bậc của một điểm A trên $E(F_p)$ là một số nguyên dương r sao cho:

$$\underbrace{A + A + ... + A}_{r} = O \tag{8.11}$$

8.1.3.2 Đường cong elliptic trên trường F_{2^m}

Một đường cong elliptic $E(F_{2^m})$ trên F_{2^m} được định nghĩa bởi các tham số $a, b \in F_{2^m}$ (với $b \neq 0$) là tập các điểm (x, y) với $x \in F_{2^m}$, $y \in F_{2^m}$ thỏa công thức:

$$y^2 + xy = x^3 + ax^2 + b \tag{8.12}$$

cùng với điểm O là điểm tại vô cực. Số lượng các điểm thuộc $E(F_{2^m})$ ký hiệu $\#E(F_{2^m})$ thoả định lý Hasse:

$$q+1-2\sqrt{q} \le \#E(F_{2^m}) \le q+1+2\sqrt{q} \tag{8.13}$$

trong đó $q = 2^m$. Ngoài ra, $\#E(F_{2^m})$ là số chẵn.

Tập hợp các điểm thuộc $E(F_{2^m})$ tạo thành một nhóm thỏa các tính chất sau:

- $O + O = O$
- $(x, y) + O = (x, y)$, $\forall (x, y) \in E(F_{2^m})$
- $(x, y) + (x, x + y) = O$, $\forall (x, y) \in E(F_{2^m})$. Khi đó, $(x, x + y)$ là điểm đối của (x, y) trên $E(F_{2^m})$)

Việc xử lý được thực hiện trên hai hệ tọa độ khác nhau: hệ tọa độ affine và hệ tọa độ quy chiếu. Với các hệ tọa độ khác nhau, việc tính toán trên đường cong cũng khác nhau.

❖ **Các phép toán trên đường cong elliptic trong hệ tọa độ affine**

Hệ mã hóa đường cong elliptic dựa trên bài toán logarit rời rạc trên $E(F_{2^m})$ và các tính toán cơ bản trên đường cong elliptic. Phép nhân được thể hiện là một dãy các phép cộng và phép nhân đôi các điểm của đường cong elliptic. Giống như các phép tính trên đường cong elliptic trên số thực, phép cộng và phép nhân đôi được định nghĩa trên hệ tọa độ.

Xét đường cong elliptic E trên F_{2^m} trong hệ tọa độ affine. Cho $P = (x_1, y_1)$, $Q = (x_2, y_2)$ là hai điểm trên đường cong elliptic $E(F_{2^m})$. Điểm đối của P là $-P = (x_1, y_1 + x_1) \in E(F_{2^m})$.

Nếu $Q \neq -P$ thì $P + Q = R = (x_3, y_3) \in E(F_{2^m})$.

Nếu $P \neq Q$ thì
$$\begin{cases} \theta = \dfrac{y_1 + y_2}{x_1 + x_2} \\ x_3 = \theta^2 + \theta + x_1 + x_2 + a_2 \\ y_3 = (x_1 + x_3)\theta + x_3 + y_1 \end{cases} \qquad (8.14)$$

Nếu $P = Q$ thì
$$\begin{cases} \theta = \dfrac{y_1}{x_1} + x_1 \\ x_3 = \theta^2 + \theta + a_2 \\ y_3 = x_1^2 + (\theta + 1)x_3 \end{cases} \qquad (8.15)$$

Thuật toán 8.2: *Thuật toán cộng điểm trong hệ tọa độ affine*

Input:

Đường cong elliptic $E(F_{2^m})$ với các tham số $a_2, a_6 \in F_{2^m}$,

Điểm $P = (x_1, y_1) \in E(F_{2^m})$ và $Q = (x_2, y_2) \in E(F_{2^m})$

Output: $R = P + Q$, $R = (x_3, y_3) \in E(F_{2^m})$

If $P = O$ **then** $R \leftarrow Q$ và trả về giá trị R

If $Q = O$ **then** $R \leftarrow P$ và trả về giá trị R

If $x_1 = x_2$ **then**

If $y_1 = y_2$ **then**

$\theta \leftarrow \frac{y_1}{x_1} + x_1$ và $x_3 \leftarrow \theta^2 + \theta + a_2$

Else If $y_2 = x_1 + y_1$ **then**

$R \leftarrow O$ và trả về R,

End If

$\theta \leftarrow \frac{y_1 + y_2}{x_1 + x_2}$

End If

$x_3 \leftarrow \theta^2 + \theta + x_1 + x_2 + a_2$

$y_3 \leftarrow (x_1 + x_3)\theta + x_3 + y_1$

Trả về $(x_3, y_3) = R$

❖ **Các phép toán đường cong elliptic trong hệ tọa độ chiếu**

Đường cong $E(F_{2^m})$ có thể được xem là tương đương với tập hợp các điểm $E'(F_{2^m})$ trên mặt phẳng chiếu $P^2(F_{2^m})$ thỏa mãn công thức:

$$y^2z + xyz = x^3 + a^2x^2z^2 + a^6z^3 \tag{8.16}$$

Sử dụng hệ tọa độ chiếu, thao tác tính nghịch đảo cần cho phép cộng và phép nhân đôi điểm trong hệ affine có thể được loại bỏ.

❖ **Chuyển đổi giữa hệ tọa độ affine và hệ tọa độ chiếu**

Mọi điểm $(a, b) \in E(F_{2^m})$ trong hệ tọa độ affine có thể được xem là bộ ba (x, y, z) trong $E'(F_{2^m})$ trong hệ tọa độ chiếu với $x = a, y = b, z = 1$. Hơn nữa, một điểm (t_x, t_y, t_z) trong hệ tọa độ chiếu với $t \neq 0$ được xem như trùng với điểm (x, y, z). Như vậy, chuyển đổi giữa hệ affine và hệ tọa độ chiếu như sau:

$$M(a, b) = M'(a, b, 1) \tag{8.17}$$

$$N(p, q, r) = N'(\frac{p}{r}, \frac{q}{r}, 1) = N(\frac{p}{r}, \frac{q}{r}) \tag{8.18}$$

❖ **Các phép toán đường cong trong hệ tọa độ chiếu**

Phương pháp trình bày công thức của phép cộng và nhân đôi trong hệ tọa độ chiếu tương tự với hệ tọa độ affine.

Cho $P' = (x_1 : y_1 : z_1) \in E'(F_{2^m})$, $Q' = (x_2 : y_2 : z_2) \in E'(F_{2^m})$ và $P' \neq -Q'$ trong đó P', Q' thuộc hệ tọa độ quy chiếu.

Do $P' = (x_1/z_1 : y_1/z_1 : 1)$, ta có thể áp dụng công thức cộng và nhân cho điểm $P(x_1/z_1, y_1/z_1)$ và Q (x_2, y_2) cho $E(F_{2^m})$ trong hệ affine để tìm $P' + Q' = R'$ $(x'_3 : y'_3 : 1)$.

Từ đó ta có:

$$x'_3 = \frac{B^2}{A^2} + \frac{B}{A} + \frac{A}{z_1} + a_2$$
$$y'_3 = \frac{B}{A}\left(\frac{x_1}{z_1} + x'_3\right) + x'_3 + \frac{y_1}{z_1} \tag{8.19}$$

Trong đó $A = (x_2z_1 + x_1)$ và $B = (y_2z_1 + y_1)$. Đặt $z_3 = A^3z_1$ và $x_3 = x'_3z_3$, $y_3 = y'_3z_3$, nếu $P + Q = (x_3: y_3: z_3)$ thì:

$$x_3 = AD,$$
$$y_3 = CD + A^2(Bx_1 + Ay_1) \tag{8.20}$$
$$z_3 = A^3z_1$$

với $C = A + B$ và $D = A^2(A + a_2z_1) + z_1BC$.

Tương tự $2P = (x_3 : y_3 : z_3)$ với

$$x_3 = AB,$$
$$y_3 = x_1^4A + B(x_1^2 + y_1z_1 + A) \tag{8.21}$$
$$z_3 = A^3$$

Trong đó $A = x_1z_1$ và $B = a_6z_1^4 + x_1^4$. Điểm kết quả có thể được chuyển trở lại sang hệ affine bằng cách nhân với z_3^{-1}. Như vậy sẽ không có thao tác tính nghịch đảo trong hệ tọa độ chiếu. Do đó, chỉ cần 1 phép nghịch đảo sau một dãy các phép cộng và nhân đôi để chuyển sang hệ affine.

Bảng 8.1. *So sánh số lượng các thao tác đối với các phép toán trên đường cong elliptic trong hệ tọa độ Affine và hệ tọa độ chiếu*

Thao tác	Tọa độ affine		Tọa độ chiếu	
	ESUM	EDBL	ESUM	EDBL
Nhân	2	2	13	7
Nghịch đảo	1	1	0	0

8.1.3.3 Phép nhân đường cong

Thuật toán 8.3: *Thuật toán nhân điểm trong hệ tọa độ affine*

Input: $P \in E(F_{2^m})$ và $c \in F_{2^m}$

Output: $Q = c \times P$

$c = \sum_{i=0}^{n} b_i 2^i$, $b_i \in \{0, 1\}$, $b_n = 1$

$Q \leftarrow P$

for $i = n$-1 **downto** 0

  Gán $Q \leftarrow Q + Q$ (Affine EDBL)

  if $b_i = 1$ **then**

    Gán $Q \leftarrow Q + P$ (Affine ESUM)

  end if

end for

Trả về Q

Phép nhân được định nghĩa như một dãy các phép cộng.

$$Q = c \times P = \underbrace{P + P + ... + P}_{c} \tag{8.22}$$

Thuật toán 8.4: *Thuật toán nhân điểm trong hệ tọa độ chiếu*

Input: $P \in E(F_{2^m})$ and $c \in F_{2^m}$
Output: $Q = c \times P$

$c = \sum_{i=0}^{n} b_i 2^i$, $b_i \in \{0, 1\}$, $b_n = 1$

Biểu diễn P trong hệ tọa độ chiếu: P'

Gán $Q' \leftarrow P'$

for $i = n$-1 **downto** 0

    $Q' \leftarrow Q' + Q'$ (Projective EDBL)

    if $b_i = 1$ **then**

        $Q' \leftarrow Q' + P'$ (Projective ESUM)

    end if

end for

Biểu diễn Q' trong hệ tọa độ affine, ta được Q

Trả về Q

8.1.4 Bài toán logarit rời rạc trên đường cong elliptic

Bài toán logarit rời rạc trên đường cong elliptic (ECDLP): *Cho E là một đường cong elliptic và $P \in E$ là một điểm có bậc n. Cho điểm $Q \in E$, tìm số nguyên dương m ($2 \le m \le n - 2$) thỏa mãn công thức $Q = m \times P$.*

Hiện nay chưa có thuật toán nào được xem là hiệu quả để giải quyết bài toán này. Để giải bài toán logarit rời rạc trên đường cong ellipse, cần phải kiểm tra tất cả các giá trị $m \in [2..n-2]$. Nếu điểm P được chọn lựa cẩn thận với n rất lớn thì việc giải bài toán ECDLP xem như không khả thi. Việc giải bài toán ECDLP khó

khăn hơn việc giải quyết bài toán logarit rời rạc trên trường số nguyên thông thường [2].

8.1.5 Áp dụng lý thuyết đường cong elliptic vào mã hóa

Các lý thuyết toán học nền tảng của đường cong elliptic được các nhà khoa học áp dụng khá hiệu quả vào lĩnh vực mã hóa, bảo mật (Elliptic Curve Cryptography - ECC). Các kết quả nghiên cứu về đường cong elliptic đã được sử dụng trong quy trình mã hóa dữ liệu, trao đổi khóa và ký nhận điện tử .

8.2 Mã hóa dữ liệu

Mô hình mã hóa dữ liệu sử dụng đường cong elliptic (Elliptic Curve Encryption Scheme - ECES) bao gồm 2 thao tác: mã hóa và giải mã.

Trước khi thực hiện việc mã hóa dữ liệu với Elliptic Curve, người gởi và người nhận cần phải sở hữu một cặp khóa công cộng – khóa riêng. Các giá trị sau được quy ước chung giữa người gởi và người nhận, gọi là các tham số chung của hệ thống mã hóa:

- Đường cong elliptic curve E.
- Điểm P, $P \in E$. Điểm P có bậc n ($n \times P = O$).

Quá trình tạo khóa được thực hiện như sau:

- Chọn một số nguyên bất kỳ d, $d \in [2, n-2]$. Đây chính là khóa riêng.
- Tính giá trị của điểm $Q = d \times P \in E$. Đây chính là khóa công cộng.

8.2.1 Thao tác mã hóa

Thao tác mã hóa sẽ mã hóa một thông điệp bằng khóa công cộng của người nhận và các tham số đường cong đã được quy ước thống nhất chung giữa người gởi (B) và người nhận (A).

Trình tự mã hóa được thực hiện như sau:

- B sử dụng khóa công cộng của A (Q_A).
- B chọn một số nguyên bất kỳ $k \in [2, n\text{-}2]$.
- B tính giá trị của điểm $(x_1, y_1) = k \times P$.
- B tính giá trị của điểm $(x_2, y_2) = k \times Q_A$. x_2 là giá trị bí mật sẽ được sử dụng để tạo khóa mã hóa thông điệp.
- B tạo mặt nạ (mask) Y từ giá trị bí mật x_2. Giá trị của Y được tạo thành từ một hàm mask generation. Tùy theo việc cài đặt hàm mask generation mà Y sẽ có giá trị khác nhau. Y chính là khóa quy ước để mã hóa thông điệp.
- B tính giá trị $C = \Phi(Y, M)$. C chính là thông điệp đã được mã hóa. Thông thường, $\Phi(Y, M) = Y \oplus M$.
- B gởi cho A thông điệp đã mã hóa C cùng với giá trị (x_1, y_1).

Giá trị k và (x_1, y_1) được tạo ra không phải khóa riêng và khóa công cộng để giao dịch của B. Đây là cặp khóa công cộng – khóa riêng được phát sinh nhất thời (one-time key pair) nhằm mã hóa thông điệp. Mỗi một thông điệp mã hóa nên sử dụng một cặp khóa công cộng – khóa riêng được phát sinh ngẫu nhiên.

8.2.2 Kết hợp ECES với thuật toán Rijndael và các thuật toán mở rộng

Trong ECES, thông thường hàm mã hóa Φ thực hiện thao tác XOR khóa với thông điệp. Trên thực tế, để tăng độ an toàn của thuật toán mã hóa, các hệ thống mã hóa bằng đường cong ellipse thay thế thao tác XOR thông điệp với khóa bằng cách kết hợp với một thuật toán mã hóa đối xứng hiệu quả hơn. Trong [27] trình bày phương pháp ECAES chính là sự kết hợp ECES với AES. Chúng ta cũng có thể sử dụng các thuật toán mở rộng 256/384/512-bit và 512/768/1024-bit trong quá trình mã hóa của ECES để tạo ra một hệ thống mã có độ an toàn rất cao.

8.2.3 Thao tác giải mã

Bằng việc sử dụng các tham số quy ước kết hợp với khóa bí mật của người nhận (A) và giá trị (x_1, y_1), A thực hiện giải mã thông điệp được mã hóa bằng ECES (C) theo trình tự sau:

Trình tự giải mã:

- A nhận giá trị (x_1, y_1).
- A tính giá trị của điểm $(x_2, y_2) = d \times (x_1, y_1)$. x_2 là giá trị bí mật sẽ được sử dụng để tạo khóa giải mã thông điệp.

- Sử dụng cùng một hàm tạo mặt nạ (mask function) như đã sử dụng ở giai đoạn mã hóa, A tạo mặt nạ Y từ giá trị bí mật x_2. Y chính là khóa bí mật để giải mã.
- A giải mã thông điệp C để lấy thông điệp M ban đầu bằng cách tính giá trị $M = \Phi^{-1}(C, Y)$. Thông thường, $\Phi^{-1}(C, Y) = C \oplus Y$.

8.3 Trao đổi khóa theo phương pháp Diffie - Hellman sử dụng lý thuyết đường cong elliptic (ECDH)

8.3.1 Mô hình trao đổi khóa Diffie-Hellman

Năm 1976, Whitfield Diffie và Martin Hellman đã đưa ra một giao thức để trao đổi các giá trị khóa quy ước giữa các đối tác trên đường truyền có độ bảo mật trung bình. Sự ra đời của giao thức trao đổi khóa Diffie-Hellman được xem là bước mở đầu cho lĩnh vực mã hóa khóa công cộng.

Giao thức này dựa trên nguyên lý của bài toán logarit rời rạc trên trường số nguyên hữu hạn. Các thao tác thực hiện trao đổi khóa Diffie-Hellman giữa hai đối tác A và B như sau:

- A và B thống nhất các giá trị g và số nguyên tố $p < g$
- A chọn một số ngẫu nhiên m. A tính giá trị $Q_A = g^m$ và gởi Q_A cho B
- B chọn một số ngẫu nhiên n. B tính giá trị $Q_B = g^n$ và gởi Q_B cho A
- A nhận được Q_B và tính giá trị $k = (Q_B)^m = g^{n \times m}$
- B nhận được Q_A và tính giá trị $k = (Q_A)^n = g^{m \times n}$

k chính là giá trị bí mật được quy ước chung.

8.3.2 Mô hình trao đổi khóa Elliptic Curve Diffie - Hellman

Mô hình trao đổi khóa Elliptic curve Diffie-Hellman tương tự mô hình trao đổi khóa Diffie-Hellman. ECDH cũng dựa vào nguyên lý của bài toán logarit rời rạc nhưng áp dụng trên đường elliptic curve. Mô hình này dùng để thiết lập một hoặc nhiều khóa quy ước chung giữa hai đối tác A và B.

Các thao tác để trao đổi khóa bằng ECDH được thực hiện như sau:

- A và B thống nhất các tham số sẽ sử dụng như: đường elliptic curve E, và điểm $P(x, y)$
- A chọn một giá trị m ngẫu nhiên. A tính giá trị điểm $Q_A = m \times P$ và gởi Q_A cho B
- B chọn một giá trị n ngẫu nhiên. B tính giá trị điểm $Q_B = n \times P$ và gởi Q_B cho A
- A nhận được Q_B và tính giá trị $G = m \times Q_B = m \times n \times P$
- B nhận được Q_A và tính giá trị $G = n \times Q_A = n \times m \times P$

Giá trị $G = m \times n \times P$ chính là giá trị bí mật được quy ước chung.

Giả sử có một người C tấn công vào đường truyền và lấy được các giá trị Q_A, Q_B, E, P, C cần lấy được m hoặc n để tìm $G = m \times n \times P$. Điều đó chính là C phải giải bài toán logarit rời rạc trên đường cong elliptic. Giải bài toán này đòi hỏi chi phí tính toán tương đương với sử dụng thuật toán vét cạn trên đường cong elliptic.

8.4 Kết luận

Hệ thống mã hóa khóa công cộng ra đời đã giải quyết các hạn chế của mã hóa quy ước. Mã hóa khóa công cộng sử dụng một cặp khóa, một khóa (thông thường là khóa riêng) dùng để mã hóa và một khóa (khóa riêng) dùng để giải mã. Mã hóa khóa công cộng giúp tránh bị tấn công khi trao đổi khóa do khóa để giải mã (khóa riêng) không cần phải truyền hoặc chia sẻ với người khác. Ngoài ra, mỗi người chỉ cần sở hữu một cặp khóa công cộng – khóa riêng và người gởi thông tin chỉ cần giữ khóa công cộng của người nhận do đó số lượng khóa cần phải quản lý giảm khá nhiều. Mỗi người chỉ cần lưu trữ bảo mật một khóa riêng của chính mình.

Tuy nhiên, do nhu cầu mã hóa và giải mã bằng hai khóa khác nhau trong cùng một cặp khóa nên để đảm bảo bảo mật, kích thước khóa công cộng – khóa riêng lớn hơn rất nhiều so với khóa công cộng. Do đó tốc độ mã hóa khóa công cộng chậm hơn tốc độ mã hóa khóa quy ước. Tốc độ mã hóa bằng phần mềm của thuật toán DES nhanh hơn khoảng 100 lần so với mã hóa RSA với cùng mức độ bảo mật.

Bảng 8.2. *So sánh kích thước khóa giữa mã hóa quy ước và mã hóa khóa công cộng với cùng mức độ bảo mật*

	Kích thước khóa (tính bằng bit)					
Khóa quy ước	56	80	112	128	192	256
RSA/DSA	512	1K	2K	3K	7.5K	15K
ECC	160		224	256	384	512

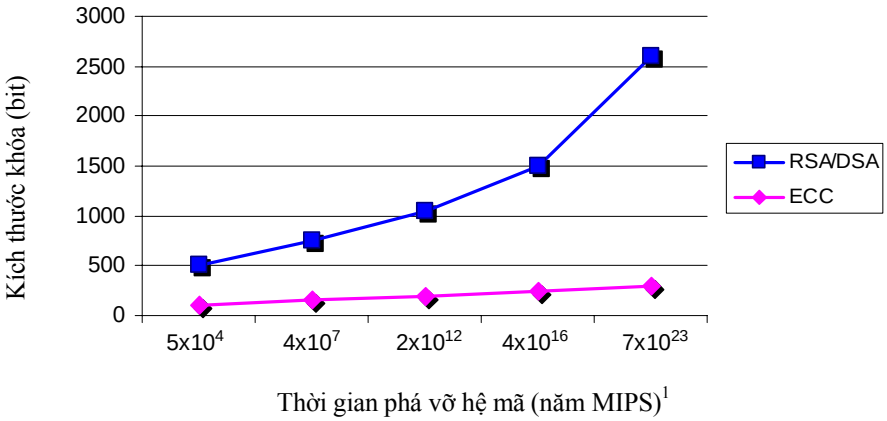
So sánh giữa các phương pháp mã hóa khóa công cộng

Mã hóa khóa công cộng dựa trên hai vấn đề lớn của toán học là bài toán logarit rời rạc và bài toán phân tích thừa số của số nguyên. Phương pháp RSA dựa trên bài toán phân tích thừa số của số nguyên tố và đã được đưa ra từ cuối thập niên 70. Phương pháp ECC dựa trên bài toán logarit rời rạc trên trường số của đường elliptic curve (ECDLP) chỉ mới được đưa ra từ năm 1985.

Một ưu điểm của ECC là khả năng bảo mật cao với kích thước khóa nhỏ dựa vào mức độ khó giải quyết của vấn đề ECDLP. Đây chính là một tính chất rất hữu ích đối với xu hướng ngày nay là tìm ra phương pháp tăng độ bảo mật của mã hóa khóa công cộng với kích thước khóa được rút gọn. Kích thước khóa nhỏ hơn giúp thu gọn được kích thước của chứng nhận giao dịch trên mạng và giảm kích thước tham số của hệ thống mã hóa. Kích thước khóa nhỏ giúp các hệ thống bảo mật dựa trên ECC giảm thời gian tạo khóa. Thời gian tạo khóa thường rất lớn ở các hệ thống RSA.

Bảng 8.3. *So sánh kích thước khóa RSA và ECC với cùng mức độ an toàn*

Thời gian cần để tấn công vào khóa (đơn vị: năm)	Kích thước khóa		Tỉ lệ kích thước khóa RSA : ECC
	RSA / DSA	ECC	
10^4	512	106	5:1
10^8	768	132	6:1
10^{11}	1024	160	7:1
10^{20}	2048	210	10:1
10^{78}	21000	600	35:1



Hình 8.5: *So sánh mức độ bảo mật giữa ECC với RSA / DSA*

Do có kích thước khóa nhỏ và khả năng phát sinh khóa rất nhanh nên ECC rất được quan tâm để áp dụng cho các ứng dụng trên môi trường giới hạn về thông lượng truyền dữ liệu, giới hạn về khả năng tính toán, khả năng lưu trữ. ECC thích hợp với các thiết bị di động kỹ thuật số như handheld, PDA, điện thoại di động và thẻ thông minh (smart card).

Các hệ thống ECC đã và đang được một số công ty lớn về viễn thông và bảo mật trên thế giới quan tâm phát triển. Nổi bật trong số đó là Certicom (Canada) kết hợp với Đại học Waterloo đã nghiên cứu và xem ECC như là chiến lược phát

[1] Nguồn: Certicom Corp. http://www.certicom.com

triển bảo mật chính của công ty. Certicom cung cấp dịch vụ bảo mật dựa trên ECC. Ngoài ra, một số công ty khác như Siemens (Đức), Matsushita (Nhật), Thompson (Pháp) cũng nghiên cứu phát triển ECC. Mới đây, RSA Security Laboratory – phòng thí nghiệm chính của RSA – đã bắt đầu nghiên cứu và đưa ECC vào sản phẩm của mình.

Tuy nhiên, ECC vẫn có một số hạn chế nhất định. Hạn chế lớn nhất hiện nay là việc chọn sử dụng các tham số đường cong và điểm quy ước chung như thế nào để thật sự đạt được độ bảo mật cần thiết. Hầu hết các đường cong được đưa ra đều thất bại khi áp dụng vào thực tiễn. Do đó hiện nay số lượng đường cong thật sự được sử dụng không được phong phú. NIST đề xuất một số đường cong elliptic curve đã được kiểm định là an toàn để đưa vào sử dụng thực tế trong tài liệu FIPS 186-2. Ngoài ra, đối với các tham số mang giá trị nhỏ, mức độ bảo mật của ECC không bằng RSA (khi $e = 3$). Đối với một số trường hợp RSA vẫn là lựa chọn tốt do RSA đã chứng minh được tính ổn định trong một khoảng thời gian khá dài.

ECC vẫn còn non trẻ và cần được kiểm định trong tương lai tuy nhiên ECC cung cấp khả năng ứng dụng rất lớn trong lĩnh vực mã hóa khóa công cộng trên các thiết bị di động và smart card. Tương lai ECC sẽ được nghiên cứu đưa vào thực tiễn phổ biến hơn.

Chương 9
Hàm băm mật mã

✍ *Nội dung của chương 7 đã trình bày về chữ ký điện tử. Để có thể sử dụng chữ ký điện tử vào các ứng dụng thực tế, chúng ta cần sử dụng các hàm băm mật mã. Nội dung của chương 9 sẽ trình bày về hàm băm mật mã. Bên cạnh các phương pháp phổ biến như MD5, SHS, các phương pháp mới như SHA-224, SHA-256/384/512 cũng được giới thiệu trong chương này.*

9.1 Giới thiệu

9.1.1 Đặt vấn đề

Trên thực tế, các thông điệp sử dụng chữ ký điện tử có độ dài bất kỳ, thậm chí lên đến vài Megabyte. Trong khi đó, thuật toán chữ ký điện tử được trình bày trên đây lại áp dụng trên các thông điệp có độ dài cố định và thường tương đối ngắn, chẳng hạn như phương pháp DSS sử dụng chữ ký 320 bit trên thông điệp 160 bit. Để giải quyết vấn đề này, chúng ta có thể chia nhỏ thông điệp cần ký thành các

đoạn nhỏ có độ dài thích hợp và ký trên từng mảnh thông điệp này. Tuy nhiên, giải pháp này lại có nhiều khuyết điểm và không thích hợp áp dụng trong thực tế:

- Nếu văn bản cần được ký quá dài thì số lượng chữ ký được tạo ra sẽ rất nhiều và kết quả nhận được là một thông điệp có kích thước rất lớn. Chẳng hạn như khi sử dụng phương pháp DSS thì thông điệp sau khi được ký sẽ có độ dài gấp đôi văn bản nguyên thủy ban đầu!

- Hầu hết các phương pháp chữ ký điện tử có độ an toàn cao đều đòi hỏi chi phí tính toán cao và do đó, tốc độ xử lý rất chậm. Việc áp dụng thuật toán tạo chữ ký điện tử nhiều lần trên một văn bản sẽ thực hiện rất lâu.

- Từng đoạn văn bản sau khi được ký có thể dễ dàng bị thay đổi thứ tự hay bỏ bớt đi mà không làm mất đi tính hợp lệ của văn bản. Việc chia nhỏ văn bản sẽ không thể bảo đảm được tính toàn vẹn của thông tin ban đầu cần được ký.

9.1.2 Hàm băm mật mã

Hàm băm mật mã là hàm toán học chuyển đổi một thông điệp có độ dài bất kỳ thành một dãy bit có độ dài cố định (tùy thuộc vào thuật toán băm). Dãy bit này được gọi là thông điệp rút gọn (message digest) hay giá trị băm (hash value), đại diện cho thông điệp ban đầu.

Dễ dàng nhận thấy rằng hàm băm h không phải là một song ánh. Do đó, với thông điệp x bất kỳ, tồn tại thông điệp $x' \neq x$ sao cho $h(x)= h(x')$. Lúc này, ta nói rằng "có sự đụng độ xảy ra".

Một hàm băm *h* được gọi là an toàn (hay "ít bị đụng độ") khi không thể xác định được (bằng cách tính toán) cặp thông điệp *x* và *x'* thỏa mãn $x \neq x'$ và $h(x) = h(x')$. Trên thực tế, các thuật toán băm là hàm một chiều, do đó, rất khó để xây dựng lại thông điệp ban đầu từ thông điệp rút gọn.

Hàm băm giúp xác định được tính toàn vẹn dữ liệu của thông tin: mọi thay đổi, dù là rất nhỏ, trên thông điệp cho trước, ví dụ như đổi giá trị 1 bit, đều làm thay đổi thông điệp rút gọn tương ứng. Tính chất này hữu ích trong việc phát sinh, kiểm tra chữ ký điện tử, các đoạn mã chứng nhận thông điệp, phát sinh số ngẫu nhiên, tạo ra khóa cho quá trình mã hóa…

Hàm băm là nền tảng cho nhiều ứng dụng mã hóa. Có nhiều thuật toán để thực hiện hàm băm, trong số đó, phương pháp SHA-1 và MD5 thường được sử dụng khá phổ biến từ thập niên 1990 đến nay.

1. Hàm băm MD4 (Message Digest 4) và MD5 (Message Digest 5):
   - Hàm băm MD4 được Giáo sư Ron Rivest đề nghị vào năm 1990. Vào năm 1992, phiên bản cải tiến MD5 của thuật toán này ra đời.
   - Thông điệp rút gọn có độ dài 128 bit.
   - Năm 1995, Hans Dobbertin đã chỉ ra sự đụng độ ngay chính trong bản thân hàm nén của giải thuật (mặc dù chưa thật sự phá vỡ được giải thuật).
   - Năm 2004, nhóm tác giả Xiaoyun Wang, Dengguo Feng, Xuejia Lai và Hongbo Yu đã công bố kết quả về việc phá vỡ thuật toán MD4 và MD5 bằng phương pháp tấn công đụng độ[2] [49].

[2] Trong tài liệu [49], nhóm tác giả không chỉ trình bày kết quả tấn công bằng đụng độ đối với phương pháp MD4, MD5 mà còn cả thuật toán HAVAL-128 và RIPEMD

2. Phương pháp Secure Hash Standard (SHS):
   - Phương pháp Secure Hash Standard (SHS) do NIST và NSA xây dựng được công bố trên Federal Register vào ngày 31 tháng 1 năm 1992 và sau đó chính thức trở thành phương pháp chuẩn từ ngày 13 tháng 5 năm 1993.
   - Thông điệp rút gọn có độ dài 160 bit.

Ngày 26/08/2002, Viện Tiêu chuẩn và Công nghệ quốc gia của Hoa Kỳ (National Institute of Standard and Technology - NIST) đã đề xuất hệ thống chuẩn hàm băm an toàn (Secure Hash Standard) gồm 4 thuật toán hàm băm SHA-1, SHA-256, SHA-384, SHA-512. Đến 25/03/2004, NIST đã chấp nhận thêm thuật toán hàm băm SHA-224 vào hệ thống chuẩn hàm băm. Các thuật toán hàm băm do NIST đề xuất được đặc tả trong tài liệu FIPS180-2 [24].

9.1.3 Cấu trúc của hàm băm

Hầu hết các hàm băm mật mã đều có cấu trúc giải thuật như sau:

- Cho trước một thông điệp M có độ dài bất kỳ. Tùy theo thuật toán được sử dụng, chúng ta có thể cần bổ sung một số bit vào thông điệp này để nhận được thông điệp có độ dài là bội số của một hằng số cho trước. Chia nhỏ thông điệp thành từng khối có kích thước bằng nhau: $M_1, M_2, \ldots M_s$
- Gọi H là trạng thái có kích thước n bit, f là "hàm nén" thực hiện thao tác trộn khối dữ liệu với trạng thái hiện hành
  - ✓ Khởi gán H_0 bằng một vector khởi tạo nào đó
  - ✓ $H_i = f(H_{i-1}, M_i)$ với $i = 1, 2, 3, \ldots, s$
- H_s chính là thông điệp rút gọn của thông điệp M ban đầu

9.1.4 Tính an toàn của hàm băm đối với hiện tượng đụng độ

Hàm băm được xem là an toàn đối với hiện tượng đụng độ khi rất khó tìm được hai thông điệp có cùng giá trị băm.

Nhận xét: Trong một tập hợp mà các phần tử mang một trong N giá trị cho trước với xác suất bằng nhau, chúng ta cần khoảng $\sqrt{N}$ phép thử ngẫu nhiên để tìm ra một cặp phần tử có cùng giá trị.

Như vậy, phương pháp hàm băm được xem là an toàn đối với hiện tượng đụng độ nếu chưa có phương pháp tấn công nào có thể tìm ra cặp thông điệp có cùng giá trị hàm băm với số lượng tính toán ít hơn đáng kể so với ngưỡng $2^{n/2}$, với n là kích thước (tính bằng bit) của giá trị băm.

Phương pháp tấn công dựa vào đụng độ:

- Tìm ra 2 thông điệp có nội dung khác nhau nhưng cùng giá trị băm.
- Ký trên một thông điệp, sau đó, người ký sẽ không thừa nhận đây là chữ ký của mình mà nói rằng mình đã ký trên một thông điệp khác.
- Như vậy, cần phải chọn 2 thông điệp "đụng độ" với nhau trước khi ký.

9.1.5 Tính một chiều

Hàm băm được xem là hàm một chiều khi cho trước giá trị băm, không thể tái tạo lại thông điệp ban đầu, hay còn gọi là "tiền ảnh" ("pre-image"). Như vậy, trong

trường hợp lý tưởng, cần phải thực hiện hàm băm cho khoảng 2^n thông điệp để tìm ra được "tiền ảnh" tương ứng với một giá trị băm.

Nếu tìm ra được một phương pháp tấn công cho phép xác định được "tiền ảnh" tương ứng với một giá trị băm cho trước thì thuật toán băm sẽ không còn an toàn nữa.

Cách tấn công nhằm tạo ra một thông điệp khác với thông điệp ban đầu nhưng có cùng giá trị băm gọi là tấn công "tiền ảnh thứ hai" (second pre-image attack).

9.2 Hàm băm MD5

9.2.1 Giới thiệu MD5

Hàm băm MD4 (Message Digest 4) được Giáo sư Rivest đề nghị vào năm 1990. Vào năm sau, phiên bản cải tiến MD5 của thuật toán này ra đời. Cùng với phương pháp SHS, đây là ba phương pháp có ưu điểm tốc độ xử lý rất nhanh nên thích hợp áp dụng trong thực tế đối với các thông điệp dài.

Thông điệp ban đầu x sẽ được mở rộng thành dãy bit X có độ dài là bội số của 512. Một bit 1 được thêm vào sau dãy bit x, tiếp đến là dãy gồm d bit 0 và cuối cùng là dãy 64 bit l biểu diễn độ dài của thông điệp x. Dãy gồm d bit 0 được thêm vào sao cho dãy X có độ dài là bội số 512. Quy trình này được thể hiện trong Thuật toán 9.1.

Thuật toán 9.1 *Thuật toán xây dựng dãy bit X từ dãy bit x*

$d = (447 - |x|) \bmod 512$

Gọi dãy 64 bit l là biểu diễn nhị phân của giá trị $|x| \bmod 2^{64}$.

$X = x \,||\, 1 \,||\, 0^d \,||\, l$

Đơn vị xử lý trong MD5 là các từ 32-bit nên dãy X sẽ được biểu diễn thành dãy các từ $X[i]$ 32 bit: $X = X[0]\ X[1]\ ...\ X[N–1]$ với N là bội số của 16.

Thuật toán 9.2 *Hàm băm MD5*

```
A = 0x67452301;
B = 0xefcdab89;
C = 0x98badcfe;
D = 0x10325476;
for i = 0 to N/16 -1
    for j = 0 to 15
        M[j] = X[16i-j]
    end for
    AA = A
    BB = B
    CC = C
    DD = D
    Round1
    Round2
    Round3
    Round4
    A = A+AA
    B = B+BB
    C = C+CC
    D = D+DD
end for
```

Đầu tiên, bốn biến A, B, C, D được khởi tạo. Những biến này được gọi là *chaining variables.*

Bốn chu kỳ biến đổi trong MD5 hoàn toàn khác nhau và lần lượt sử dụng các hàm *F*, *G*, *H* và *I*. Mỗi hàm có tham số *X*, *Y*, *Z* là các từ 32 bit và kết quả là một từ 32 bit.

$F(X, Y, Z) = (X \wedge Y) \vee ((\neg X) \wedge Z)$

$G(X, Y, Z) = (X \wedge Z) \vee (Y \wedge (\neg Z))$

$H(X, Y, Z) = X \oplus Y \oplus Z$

$I(X, Y, Z) = Y \oplus (X \vee (\neg Z))$ (9.1)

với quy ước:

$X \wedge Y$	Phép toán AND trên bit giữa *X* và *Y*
$X \vee Y$	Phép toán OR trên bit giữa *X* và *Y*
$X \oplus Y$	Phép toán XOR trên bit giữa *X* và *Y*
$\neg X$	Phép toán NOT trên bit của *X*
$X + Y$	Phép cộng (modulo 2^{32})
$X <<< s$	Các bit của *X* được dịch chuyển xoay vòng sang trái *s* vị trí ($0 \le s < 32$)

Định nghĩa các hàm:

```
FF(a,b,c,d,Mj,s,ti):
       a = b + ((a + F(b,c,d) + Mj + ti) <<< s)
GG(a,b,c,d,Mj,s,ti):
       a = b + ((a + G(b,c,d) + Mj + ti) <<< s)
HH(a,b,c,d,Mj,s,ti):
       a = b + ((a + H(b,c,d) + Mj + ti) <<< s)
II(a,b,c,d,Mj,s,ti):
       a = b + ((a + I(b,c,d) + Mj + ti) <<< s)
```

với `Mj` là *M*[*j*] và hằng số `ti` xác định theo công thức:

$$\texttt{ti} = \lfloor 2^{32} \left| \sin(i) \right| \rfloor, \; i \text{ tính bằng radian.}$$

Bảng 9.1 thể hiện chi tiết bốn chu kỳ biến đổi sử dụng trong MD5.

Bảng 9.1. *Chu kỳ biến đổi trong MD5*

Chu kỳ 1	Chu kỳ 2
FF(a,b,c,d,M0 , 7,0xd76aa478)	GG(a,b,c,d,M1 , 5,0xf61e2562)
FF(d,a,b,c,M1 ,12,0xe8c7b756)	GG(d,a,b,c,M6 , 9,0xc040b340)
FF(c,d,a,b,M2 ,17,0x242070db)	GG(c,d,a,b,M11,14,0x265e5a51)
FF(b,c,d,a,M3 ,22,0xclbdceee)	GG(b,c,d,a,M0 ,20,0xe9b6c7aa)
FF(a,b,c,d,M4 , 7,0xf57c0faf)	GG(a,b,c,d,M5 , 5,0xd62fl05d)
FF(d,a,b,c,M5 ,12,0x4787c62a)	GG(d,a,b,c,M10, 9,0x02441453)
FF(c,d,a,b,M6 ,17,0xa8304613)	GG(c,d,a,b,M15,14,0xd8ale681)
FF(b,c,d,a,M7 ,22,0xfd469501)	GG(b,c,d,a,M4 ,20,0xeid3fbc8)
FF(a,b,c,d,M8 , 7,0x698098d8)	GG(a,b,c,d,M9 , 5,0x21elcde6)
FF(d,a,b,c,M9 ,12,0x8b44f7af)	GG(d,a,b,c,M14, 9,0xc33707d6)
FF(c,d,a,b,M10,17,0xffff5bbl)	GG(c,d,a,b,M3 ,14,0xf4d50d87)
FF(b,c,d,a,M11,22,0x895cd7be)	GG(b,c,d,a,M8 ,20,0x455al4ed)
FF(a,b,c,d,M12, 7,0x6b901122)	GG(a,b,c,d,M13, 5,0xa9e3e905)
FF(d,a,b,c,M13,12,0xfd987193)	GG(d,a,b,c,M2 , 9,0xfcefa3f8)
FF(c,d,a,b,M14,17,0xa679438e)	GG(c,d,a,b,M7 ,14,0x676f02d9)
FF(b,c,d,a,M15,22,0x49b40821)	GG(b,c,d,a,M12,20,0x8d2a4c8a)

Chu kỳ 3	Chu kỳ 4
HH(a,b,c,d,M5 , 4,0xfffa3942)	II(a,b,c,d,M0 , 6,0xf4292244)
HH(d,a,b,c,M8 ,11,0x8771f6811	II(d,a,b,c,M7 ,10,0x432aff97)
HH(c,d,a,b,M11,16,0x6d9d6122)	II(c,d,a,b,M14,15,0xab9423a7)
HH(b,c,d,a,M14,23,0xfde5380c)	II(b,c,d,a,M5 ,21,0xfc93a039)
HH(a,b,c,d,M1 , 4,0xa4beea44)	II(a,b,c,d,M12, 6,0x655b59c3)
HH(d,a,b,c,M4 ,11,0x4bdecfa9)	II(d,a,b,c,M3 ,10,0x8f0ccc92)
HH(c,d,a,b,M7 ,16,0xf6bb4b60)	II(c,d,a,b,M10,15,0xffeff47d)
HH(b,c,d,a,M10,23,0xbebfbc70)	II(b,c,d,a,M1 ,21,0x85845ddl)
HH(a,b,c,d,M13, 4,0x289biec6)	II(a,b,c,d,M8 , 6,0x6fa87e4f)
HH(d,a,b,c,M0 ,11,0xeaal27fa)	II(d,a,b,c,M15,10,0xfe2ce6e0)
HH(c,d,a,b,M3 ,16,0xd4ef3085)	II(c,d,a,b,M6 ,15,0xa3014314)
HH(b,c,d,a,M6 ,23,0x04881d05)	II(b,c,d,a,M13,21,0x4e0811al)
HH(a,b,c,d,M9 , 4,0xd9d4d039)	II(a,b,c,d,M4 , 6,0xf7537e82)
HH(d,a,b,c,M12,11,0xe6db99e5)	II(d,a,b,c,M11,10,0xbd3af235)
HH(c,d,a,b,M15,16,0xlfa27cf8)	II(c,d,a,b,M2 ,15,0x2ad7d2bb)
HH(b,c,d,a,M2 ,23,0xc4ac5665)	II(b,c,d,a,M9 ,21,0xeb86d391)

9.2.2 Nhận xét

Phương pháp MD5 có những ưu điểm cải tiến so với phương pháp MD4 [45]:

- MD4 chỉ có ba chu kỳ biến đổi trong khi MD5 được bổ sung thêm chu kỳ thứ tư giúp tăng mức độ an toàn.

- Mỗi thao tác trong từng chu kỳ biến đổi của MD5 sử dụng các hằng số ti phân biệt trong khi MD4 sử dụng hằng số chung cho mọi thao tác trong cùng

chu kỳ biến đổi (Trong MD4, hằng số ti sử dụng trong mỗi chu kỳ lần lượt là 0, 0x5a827999, 0x6ed9eba1).

- Hàm G ở chu kỳ hai của MD4: $G(X, Y, Z) = ((X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z))$ được thay thế bằng $((X \wedge Z) \vee (Y \wedge Z))$ nhằm giảm tính đối xứng.

- Mỗi bước biến đổi trong từng chu kỳ chịu ảnh hưởng kết quả của bước biến đổi trước đó nhằm tăng nhanh tốc độ của hiệu ứng lan truyền (avalanche).

- Các hệ số dịch chuyển xoay vòng trong mỗi chu kỳ được tối ưu hóa nhằm tăng tốc độ hiệu ứng lan truyền. Ngoài ra, mỗi chu kỳ sử dụng bốn hệ số dịch chuyển khác nhau.

9.3 Phương pháp Secure Hash Standard (SHS)

Phương pháp Secure Hash Standard (SHS) do NIST và NSA xây dựng được công bố trên Federal Register vào ngày 31 tháng 1 năm 1992 và sau đó chính thức trở thành phương pháp chuẩn từ ngày 13 tháng 5 năm 1993.

Nhìn chung, SHS được xây dựng trên cùng cơ sở với phương pháp MD4 và MD5. Tuy nhiên, phương pháp SHS lại áp dụng trên hệ thống big-endian thay vì little-endian như phương pháp MD4 và MD5. Ngoài ra, thông điệp rút gọn kết quả của hàm băm SHS có độ dài 160 bit (nên phương pháp này thường được sử dụng kết hợp với thuật toán DSS).

Tương tự MD5, thông điệp nguồn *x* sẽ được chuyển thành một dãy bit có độ dài là bội số của 512. Từng nhóm gồm 16 từ-32 bit *X*[0], *X*[1],..., *X*[15] sẽ được mở rộng thành 80 từ-32 bit *W*[0], *W*[1], ..., *W*[79] theo công thức:

$$W[t] = \begin{cases} X[t], & 0 \le t \le 15 \\ X[j-3] \oplus X[j-8] \oplus X[j-14] \oplus X[j-16], & 16 \le t \le 79 \end{cases} \quad (9.2)$$

Trong phiên bản cải tiến của SHS, công thức trên được thay bằng:

$$W[t] = \begin{cases} X[t], & 0 \le t \le 15 \\ (X[j-3] \oplus X[j-8] \oplus X[j-14] \oplus X[j-16]) <<< 1, & 16 \le t \le 79 \end{cases} \quad (9.3)$$

Tương tự MD5, phương pháp SHS sử dụng bốn chu kỳ biến đổi, trong đó, mỗi chu kỳ gồm 20 bước biến đổi liên tiếp nhau. Chúng ta có thể xem như SHS bao gồm 80 bước biến đổi liên tiếp nhau. Trong đoạn mã chương trình dưới đây, hàm *f*[*t*] và hằng số *K*[*t*] được định nghĩa như sau:

$$f[t](X,Y,Z) = \begin{cases} (X \wedge Y) \vee ((\neg X) \wedge Z), & 0 \le t \le 19 \\ X \oplus Y \oplus Z, & 20 \le t \le 39 \\ (X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z), & 40 \le t \le 59 \\ X \oplus Y \oplus Z, & 60 \le t \le 79 \end{cases} \quad (9.4)$$

$$K[t] = \begin{cases} \texttt{0x5a827999}, & 0 \le t \le 19 \\ \texttt{0x6ed9eba1}, & 20 \le t \le 39 \\ \texttt{0x8f1bbcdc}, & 40 \le t \le 59 \\ \texttt{0xca62c1d6}, & 60 \le t \le 79 \end{cases} \quad (9.5)$$

```
A = 0x67452301;
B = 0xefcdab89;
C = 0x98badcfe;
D = 0x10325476;
```

```
E = 0xc3d2e1f0;
for i=0 to N/16 -1
    for t=0 to 15 do
        W[t] = X[16*t-j]
    end for
    for t=16 to 79
        W[t] =(W[t-3] xor W[t-8] xor W[t-14] xor W[t-16])<<<1
    a = A
    b = B
    c = C
    d = D
    e = E
    for t=0 to 79
        TEMP = (a<<<5)+f[t](b,c,d)+e+W[t]+K[t]
        e    = d
        d    = c
        c    = b <<< 30
        b    = a
        a    = TEMP
    end for
    A = A+a
    B = B+b
    C = C+c
    D = D+d
    E = E+e
end for
```

9.3.1 Nhận xét

Phương pháp SHS rất giống với MD4 nhưng thông điệp rút gọn được tạo ra có độ dài 160-bit. Cả 2 phương pháp này đều là sự cải tiến từ MD4. Dưới đây là một số đặc điểm so sánh giữa MD5 và SHS:

- Tương tự như MD5, phương pháp SHS cũng bổ sung thêm chu kỳ biến đổi thứ tư để tăng mức độ an toàn. Tuy nhiên, chu kỳ thứ tư của SHS sử dụng lại hàm f của chu kỳ thứ 2.

- 20 bước biến đổi trong cùng chu kỳ của phương pháp SHS sử dụng hằng số chung $K[t]$ trong khi mỗi bước biến đổi của phương pháp MD5 lại dùng các hằng số khác nhau.

- Trong phương pháp MD5, hàm G ở chu kỳ thứ hai của MD4: $G(X,Y,Z) = ((X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z))$ được thay thế bằng $((X \wedge Z) \vee (Y \wedge Z))$ nhằm giảm tính đối xứng. Phương pháp SHS vẫn sử dụng hàm G như trong MD4.

- Trong MD5 và SHS, mỗi bước biến đổi chịu ảnh hưởng bởi kết quả của bước biến đổi trước đó để tăng nhanh hiệu ứng lan truyền.

Hiện tại vẫn chưa có phương pháp tấn công nào có thể áp dụng được đối với phương pháp SHS. Ngoài ra, do thông điệp rút gọn của phương pháp SHS có độ dài 160 bit nên có độ an toàn cao hơn đối với phương pháp tấn công brute-force (kể cả phương pháp birthday attack) so với phương pháp MD5.

9.4 Hệ thống chuẩn hàm băm mật mã SHA

9.4.1 Ý tưởng của các thuật toán hàm băm SHA

Các thuật toán hàm băm SHA gồm 2 bước: tiền xử lý và tính toán giá trị băm.

- Bước tiền xử lý bao gồm các thao tác:
  - Mở rộng thông điệp
  - Phân tích thông điệp đã mở rộng thành các khối *m* bit
  - Khởi tạo giá trị băm ban đầu
- Bước tính toán giá trị băm bao gồm các thao tác:
  - Làm *N* lần các công việc sau:
    - Tạo bảng phân bố thông điệp (message schedule) từ khối thứ *i*.
    - Dùng bảng phân bố thông điệp cùng với các hàm, hằng số, các thao tác trên từ để tạo ra giá trị băm *i*.
  - Sử dụng giá trị băm cuối cùng để tạo thông điệp rút gọn.

Thông điệp *M* được mở rộng trước khi thực hiện băm. Mục đích của việc mở rộng này nhằm đảm bảo thông điệp mở rộng có độ dài là bội số của 512 hoặc 1024 bit tùy thuộc vào thuật toán.

Sau khi thông điệp đã mở rộng, thông điệp cần được phân tích thành *N* khối *m*-bit trước khi thực hiện băm.

Đối với SHA-1 và SHA-256, thông điệp mở rộng được phân tích thành N khối 512-bit $M^{(1)}$, $M^{(2)}$,..., $M^{(N)}$. Do đó 512 bit của khối dữ liệu đầu vào có thể được thể hiện bằng 16 từ 32-bit, $M_0^{(i)}$ chứa 32 bit đầu của khối thông điệp i, $M_1^{(i)}$ chứa 32 bit kế tiếp...

Đối với SHA-384 và SHA-512, thông điệp mở rộng được phân tích thành N khối 1024-bit $M^{(1)}$, $M^{(2)}$,..., $M^{(N)}$. Do đó 1024 bit của khối dữ liệu đầu vào có thể được thể hiện bằng 16 từ 64-bit, $M_0^{(i)}$ chứa 64 bit đầu của khối thông điệp i, $M_1^{(i)}$ chứa 64 bit kế tiếp...

Trước khi thực hiện băm, với mỗi thuật toán băm an toàn, giá trị băm ban đầu $H^{(0)}$ phải được thiết lập. Kích thước và số lượng từ trong $H^{(0)}$ tùy thuộc vào kích thước thông điệp rút gọn. Các giá trị băm ban đầu của các thuật toán SHA được trình bày trong phần Phụ lục E .

Các cặp thuật toán SHA-224 và SHA-256; SHA-384 và SHA-512 có các thao tác thực hiện giống nhau, chỉ khác nhau về số lượng bit kết quả của thông điệp rút gọn. Nói cách khác, SHA-224 sử dụng 224 bit đầu tiên trong kết quả thông điệp rút gọn sau khi áp dụng thuật toán SHA256. Tương tự SHA-384 sử dụng 384 bit đầu tiên trong kết quả thông điệp rút gọn sau khi áp dụng thuật toán SHA-512.

9.4.2 Khung thuật toán chung của các hàm băm SHA

Trong các hàm băm SHA, chúng ta cần sử dụng thao tác quay phải một từ, ký hiệu là ROTR, và thao tác dịch phải một từ, ký hiệu là SHR.

Hình 9.1 thể hiện khung thuật toán chung cho các hàm băm SHA

Hình 9.1. *Khung thuật toán chung cho các hàm băm SHA*

for $i = 1$ **to** N

  for $t = 0$ **to** 15

    $W_t = M_t^{(i)}$

  end for

  for $t = 16$ **to** *scheduleRound*

    $W_t = \sigma_1(W_{t-2}) + W_{t-7} + \sigma_0(W_{t-15}) + W_{t-16}$

  end for

  $a = H_0^{(i-1)}$

  $b = H_1^{(i-1)}$

  $c = H_2^{(i-1)}$

  $d = H_3^{(i-1)}$

  $e = H_4^{(i-1)}$

  $f = H_5^{(i-1)}$

  $g = H_6^{(i-1)}$

  $h = H_7^{(i-1)}$

  for $t = 0$ **to** 63

    $T_1 = h + \sum_1(e) + \text{Ch}(e, f, g) + K_t + W_t$

    $T_2 = \sum_0(a) + \text{Maj}(a, b, c)$

    $h = g$

    $g = f$

    $f = e$

    $e = d + T_1$

    $d = c$

    $c = b$

$b = a$

$a = T_1 + T_2$

end for

$H_0^{(i)} = a + H_0^{(i-1)}$

$H_1^{(i)} = b + H_1^{(i-1)}$

$H_2^{(i)} = c + H_2^{(i-1)}$

$H_3^{(i)} = d + H_3^{(i-1)}$

$H_4^{(i)} = e + H_4^{(i-1)}$

$H_5^{(i)} = f + H_5^{(i-1)}$

$H_6^{(i)} = g + H_6^{(i-1)}$

$H_7^{(i)} = h + H_7^{(i-1)}$

end for

Mỗi thuật toán có bảng hằng số phân bố thông điệp tương ứng. Kích thước bảng hằng số thông điệp (scheduleRound) của SHA-224 và SHA-256 là 64, kích thước bảng hằng số thông điệp của SHA-384 và SHA-512 là 80. Chi tiết của từng bảng hằng số được trình bày trong Phụ lục E .

Trong phương pháp SHA-224 và SHA-256, chúng ta cần sử dụng các hàm sau:

$$\begin{aligned}
&\mathrm{Ch}(x,y,z) = (x \wedge y) \oplus (\neg x \wedge z) \\
&\mathrm{Maj}(x,y,z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) \\
&\sum\nolimits_0 (x) = \mathrm{ROTR}^2(x) \oplus \mathrm{ROTR}^{13}(x) \oplus \mathrm{ROTR}^{22}(x) \\
&\sum\nolimits_1 (x) = \mathrm{ROTR}^6(x) \oplus \mathrm{ROTR}^{11}(x) \oplus \mathrm{ROTR}^{25}(x) \\
&\sigma_0(x) = \mathrm{ROTR}^7(x) \oplus \mathrm{ROTR}^{18}(x) \oplus \mathrm{SHR}^3(x) \\
&\sigma_1(x) = \mathrm{ROTR}^{17}(x) \oplus \mathrm{ROTR}^{19}(x) \oplus \mathrm{SHR}^{10}(x)
\end{aligned} \tag{9.6}$$

Trong phương pháp SHA-384 và SHA-512, chúng ta cần sử dụng các hàm sau:

$$
\begin{aligned}
&\mathrm{Ch}(x,y,z) = (x \wedge y) \oplus (\neg x \wedge z) \\
&\mathrm{Maj}(x,y,z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) \\
&\textstyle\sum_0(x) = \mathrm{ROTR}^{28}(x) \oplus \mathrm{ROTR}^{34}(x) \oplus \mathrm{ROTR}^{29}(x) \\
&\textstyle\sum_1(x) = \mathrm{ROTR}^{14}(x) \oplus \mathrm{ROTR}^{18}(x) \oplus \mathrm{ROTR}^{41}(x) \\
&\sigma_0(x) = \mathrm{ROTR}^{1}(x) \oplus \mathrm{ROTR}^{8}(x) \oplus \mathrm{SHR}^{7}(x) \\
&\sigma_1(x) = \mathrm{ROTR}^{19}(x) \oplus \mathrm{ROTR}^{61}(x) \oplus \mathrm{SHR}^{6}(x)
\end{aligned}
\tag{9.7}
$$

9.4.3 Nhận xét

Chuẩn SHS đặc tả 5 thuật toán băm an toàn SHA-1, SHA-224[3], SHA-256, SHA-384 và SHA-512. Bảng 9.2 thể hiện các tính chất cơ bản của bốn thuật toán băm an toàn.

Sự khác biệt chính của các thuật toán là số lượng bit bảo mật của dữ liệu được băm – điều này có ảnh hưởng trực tiếp đến chiều dài của thông điệp rút gọn. Khi một thuật toán băm được sử dụng kết hợp với thuật toán khác đòi hỏi phải cho kết quả số lượng bit tương ứng. Ví dụ, nếu một thông điệp được ký với thuật toán chữ ký điện tử cung cấp 128 bit thì thuật toán chữ ký đó có thể đòi hỏi sử dụng một thuật toán băm an toàn cung cấp 128 bit như SHA-256.

Ngoài ra, các thuật toán khác nhau về kích thước khối và kích thước từ được sử dụng.

[3] Đây là thuật toán hàm băm vừa được NIST công nhận thành chuẩn hàm băm an toàn vào 02/2004.

Bảng 9.2. *Các tính chất của các thuật toán băm an toàn*

Thuật toán	Kích thước (bit)				Độ an toàn[4] (đơn vị: bit)
	Thông điệp	Khối	Từ	Thông điệp rút gọn	
SHA-1	$< 2^{64}$	512	32	160	80
SHA-224	$< 2^{64}$	512	32	224	112
SHA-256	$< 2^{64}$	512	32	256	128
SHA-384	$< 2^{128}$	1024	64	384	192
SHA-512	$< 2^{128}$	1024	64	512	256

9.5 Kiến trúc hàm băm Davies-Mayer và ứng dụng của thuật toán Rijndael và các phiên bản mở rộng vào hàm băm

9.5.1 Kiến trúc hàm băm Davies-Mayer

Hàm băm Davies-Mayer [36] là một kiến trúc hàm băm dựa trên việc mã hóa theo khối trong đó độ dài của thông điệp rút gọn (tính theo bit) bằng với kích thước khối thông điệp ứng với thuật toán mã hóa được sử dụng.

Gọi n, k lần lượt là kích thước khối và kích thước khóa của thuật toán được sử dụng. Trong hàm băm Davies-Mayer không cần sử dụng khóa. Khóa ban đầu được thiết lập mặc định, có giá trị là 2^k-1 với k là kích thước khóa (tính bằng bit) của thuật toán. Hàm mã hóa E sử dụng khóa K được ký hiệu là E_K.

[4] "Độ an toàn" là việc sử dụng phương pháp tấn công vào thông điệp rút gọn kích thước n, đòi hỏi xử lý xấp xỉ $2^{n/2}$

Thông điệp ban đầu được chia thành *m* khối có kích thước *n* bit. Davies-Mayer hash chính là thực hiện lần lượt *m* lần thao tác sau:

$$H_i = E_{X_i}(H_{i-1}) \oplus X_i \tag{9.8}$$

H_m chính là thông điệp rút gọn của thông điệp ban đầu.

9.5.2 Hàm AES-Hash

Các thuật toán mã hóa được sử dụng chủ yếu với chức năng chính là để mã hóa và giải mã dữ liệu, tuy nhiên các thuật toán này còn có một khả năng ứng dụng khác ít được đề cập đến đó là được sử dụng như một hàm băm. Bram Cohen đề xuất việc sử dụng thuật toán thuộc chuẩn AES để làm hàm băm (AES-Hash) vào tháng 05 năm 2001.

Theo Bram Cohen[6], AES-Hash đảm bảo các tính chất của một hàm băm: nhận vào thông điệp ban đầu là một chuỗi bit có độ dài bất kỳ và trả về một chuỗi bit có độ dài cố định là 256 bit. Mọi sự thay đổi dù nhỏ nhất của thông điệp ban đầu sẽ làm giá trị băm thay đổi. Việc tìm kiếm hai thông điệp ban đầu có cùng giá trị băm 256 bit đòi hỏi phải thực hiện 2^{128} phép toán, và cần 2^{256} phép toán để tìm "tiền ảnh" của giá trị băm 256 bit.

AES-Hash được mô tả dựa trên kiến trúc hàm băm Davies-Mayer, sử dụng thuật toán Rijndael với kích thước khối và khóa đều là 256 bit.

Quá trình thực hiện AES-Hash gồm các bước:

- Mở rộng thông điệp.

  Thông điệp được mở rộng để có kích thước bằng một bội số chẵn nhỏ nhất (lớn hơn kích thước thông điệp) của kích thước khối. Việc này được thực hiện bằng cách thêm vào các bit zero vào cuối thông điệp sao cho kích thước đạt được là một bội số lẻ nhỏ nhất (lớn hơn kích thước thông điệp) của 128 bit. Sau đó thêm 128 bit chứa giá trị chiều dài ban đầu của thông điệp.

❑**Ví dụ:** Thông điệp ban đầu (40 bit):

1110 1011 0010 0110 0011 0110 0111 1011 1001 1001

Thông điệp mở rộng sẽ có độ dài: 40 bit ban đầu + (128 – 40) bit 0 mở rộng + 128 bit thể hiện giá trị 101000_2

Thông điệp mở rộng:

$$\underbrace{1110\ 1011\ 0010\ 0110\ 0011\ 0110\ 0111\ 1011\ 1001\ 1001}_{40\text{bit}}\ \underbrace{000.....000}_{88\text{bit}}\ \underbrace{0.....00101000}_{128\text{bit}}$$

- Chia thông điệp mở rộng thành n khối x_1, ... x_n, mỗi khối kích thước 256 bit.
- Áp dụng Davies-Mayer Hash bằng thuật toán Rijndael n lần cho n khối.

$$H_i = E_{X_i}(H_{i-1}) \oplus X_i \tag{9.9}$$

- Áp dụng thao tác bổ sung cuối để thu được giá trị băm.

$$H_{n+1} = E_{H_n}(H_n) \oplus H_n \tag{9.10}$$

H_{n+1} chính là giá trị băm của thông điệp ban đầu.

9.5.3 Hàm băm Davies-Mayer và AES-Hash

Hàm băm Davies-Mayer được chứng minh rằng để tìm thông điệp ban đầu thứ 2 có cùng kết quả giá trị băm (độ dài *n* bit) với thông điệp ban đầu cho trước ("tiền ảnh thứ hai") cần phải thực hiện 2^n thao tác, để tìm cặp thông điệp có cùng giá trị băm cần thực hiện $2^{n/2}$ thao tác [36]. Do đó, để đạt được mức độ bảo mật có thể chấp nhận được thì kích thước khối đòi hỏi phải lớn. Vào thời điểm hiện tại, kích thước khối phải lớn hơn 80 bit để tránh tấn công "tiền ảnh thứ hai" và lớn hơn 160 bit để tránh tấn công đụng độ. Điều này có nghĩa không thể sử dụng các thuật toán mã hóa có kích thước khối 64 bit (ví dụ như DES [25], IDEA...) để thực hiện Davies-Mayer Hash. Một điều lưu ý khác là hàm băm Davies-Mayer được xem là không an toàn khi sử dụng các thuật toán DES-X (ví dụ như 3DES).

AES-Hash áp dụng Davies-Mayer Hash, sử dụng thuật toán Rijndael 256 bit nên đảm bảo được độ an toàn đối với tấn công "tiền ảnh thứ hai" và tấn công "đụng độ". Ngoài ra, AES-Hash còn thực hiện thao tác bổ sung cuối để tăng chi phí khi tấn công hàm băm. Do đó, mức độ an toàn bảo mật của hàm băm AES-Hash sẽ được tăng đáng kể.

Hiện tại, thuật toán AES-Hash chưa được NIST bổ sung vào danh sách các chuẩn hàm băm an toàn vì AES-Hash sử dụng thuật toán Rijndael với kích thước khối 256 bit, trong khi NIST chỉ mới quy định kích thước khối trong chuẩn AES là 128 bit. Tuy nhiên, NIST đã đưa AES-Hash vào danh sách đề nghị chuẩn hàm băm an toàn[5].

[5] Computer Security Objects Register (CSOR): http://csrc.nist.gov/csor/

9.6 Xây dựng các hàm băm sử dụng các thuật toán mở rộng dựa trên thuật toán Rijndael

Một trong những ứng dụng của hàm băm là biến đổi chuỗi mật khẩu có độ dài bất kỳ của người dùng thành mảng các byte có kích thước cố định để sử dụng làm khóa của các thuật toán mã hóa đối xứng. Đối với các thuật toán mở rộng dựa trên thuật toán Rijndael, bao gồm thuật toán mở rộng 256/384/512-bit và thuật toán mở rộng 512/768/1024-bit, chúng ta cần sử dụng mã khóa có kích thước là 256, 384, 512, 768 hoặc 1024 bit. Nếu sử dụng các hàm băm thông thường (như nhóm các hàm băm SHA hoặc AES-HASH) thì chưa đáp ứng được tất cả các trường hợp kích thước mã khóa của các thuật toán mở rộng này. Việc ghép nối hay biến đổi giá trị băm của các hàm băm thông thường để kéo dài chuỗi bit nhận được ra đủ độ dài đòi hỏi của khóa không phải là giải pháp tối ưu. Do đó, giải pháp được đề nghị là sử dụng chính các thuật toán mở rộng để xây dựng các hàm băm có không gian giá trị băm rộng hơn, đồng thời có khả năng phục vụ cho việc tạo khóa cho chính các thuật toán này từ chuỗi mật khẩu của người dùng.

Quá trình thực hiện nhóm hàm băm này hoàn toàn tương tự như AES-Hash, chỉ thay đổi độ dài của khối và thao tác mã hóa thông tin được sử dụng trong thuật toán.

Chương 10
Chứng nhận khóa công cộng

✍ *Nội dung của chương 10 trình bày các vấn đề về chứng nhận khóa công cộng, bao gồm các loại giấy chứng nhận khóa công cộng, các thành phần của một cơ sở hạ tầng khóa công cộng (PKI), các quy trình quản lý giấy chứng nhận và các mô hình chứng nhận khóa công cộng. Phần cuối chương này trình bày ứng dụng kết hợp giữa hệ thống mã hóa quy ước và hệ thống mã hóa khóa công cộng có sử dụng chứng nhận khóa công cộng để xây dựng hệ thống thư điện tử an toàn.*

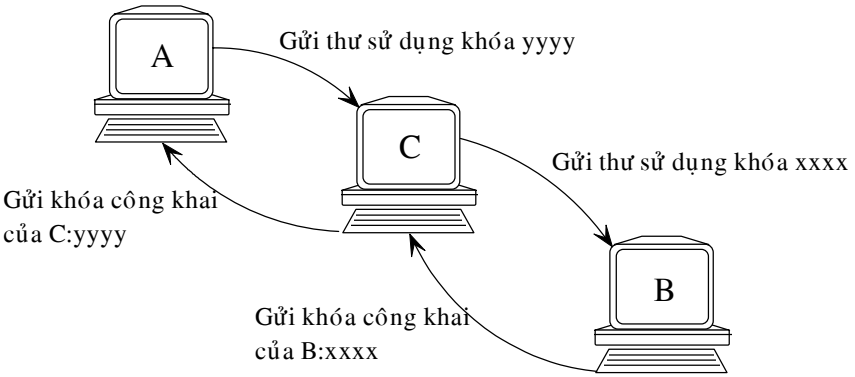
10.1 Giới thiệu

Không giống như các mã khóa bí mật, mã khóa công cộng vẫn có thể đảm bảo được an toàn thông tin ngay cả khi được công bố rộng rãi. Điều này giúp cho vấn đề trao đổi mã khóa trở nên dễ dàng hơn. Tuy nhiên, vẫn còn tồn tại một số vấn đề liên quan đến việc trao đổi mã khóa công cộng, đặc biệt là vấn đề làm thế nào xác định được ai thật sự là chủ của một mã khóa.

Một hệ thống sử dụng khóa công cộng chỉ thật sự an toàn khi xác định được chính xác người chủ sở hữu của mã khóa. Dưới đây là một trường hợp không an toàn trong

việc sử dụng khóa công cộng mà không thể xác định chính xác được người chủ của mã khóa.

❒ Ví dụ: Giả sử C có thể nhận được tất cả thông tin trao đổi giữa A và B. Khi B gửi mã khóa công cộng xxxx của mình cho A, C sẽ nhận lấy thông điệp này và gửi cho A mã khóa công cộng yyyy của mình. Như vậy, A sẽ cho rằng yyyy chính là khóa công cộng của B và dùng mã khóa này để mã hóa thư gửi cho B. Lúc này, C lại giải mã bức thư của A và mã hóa một thông điệp khác bằng khóa công cộng xxxx của B rồi gửi cho B. Như vậy, B sẽ nhận được một thông điệp từ C thay vì từ A.



Hình 10.1. *Vấn đề chủ sở hữu khóa công cộng*

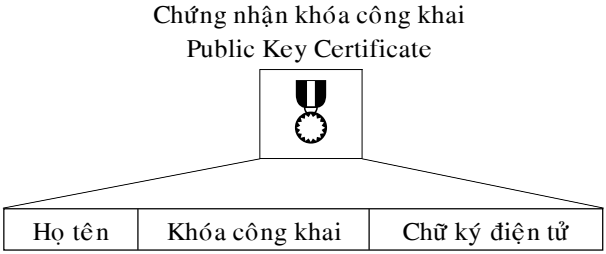
Trên thực tế, vấn đề này được giải quyết theo hai cách:

- Chứng nhận khóa công cộng: Khóa công cộng được phân phối gồm ba thành phần chính: họ tên hoặc định danh của người sở hữu thật sự của khóa,

khóa công cộng và chữ ký điện tử giúp xác nhận được tính hợp lệ của hai thành phần này (Hình 10.2).

- Hệ thống phân phối khóa tin cậy: sử dụng hệ thống trao đổi thông tin đáng tin cậy để chuyển mã khóa công cộng đến người nhận. Quá trình trao đổi này dễ dàng hơn so với quá trình trao đổi mã khóa bí mật vì ở đây không đặt ra vấn đề bảo mật mà chỉ cần đảm bảo được nội dung chính xác của mã khóa cần trao đổi. Giải pháp này thường áp dụng đối với khóa công cộng sẽ được cùng để kiểm tra chữ ký điện tử trên chứng nhận của các khóa công cộng khác.

Các chứng nhận khóa công cộng được ký bởi một tổ chức trung gian có uy tín được gọi là CA (Certification Authority). Khóa công cộng của CA sẽ được cung cấp cho người sử dụng thông qua hệ thống phân phối khóa tin cậy để họ có thể kiểm tra được các chứng nhận khóa công cộng khác do tổ chức này ký.

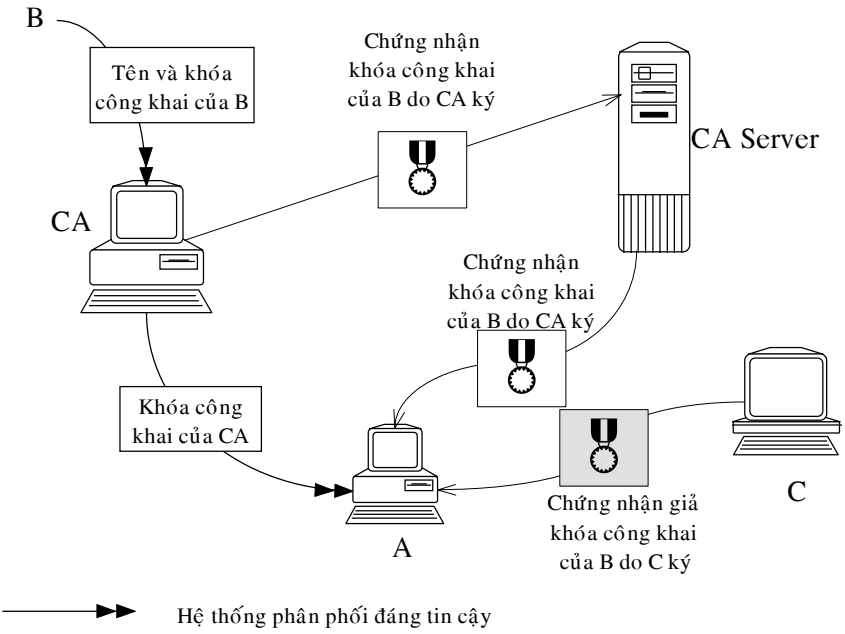


Hình 10.2. *Các thành phần của một chứng nhận khóa công cộng*

Hình 10.3 minh họa hệ thống sử dụng chứng nhận khóa công cộng. Giả sử A cần có khóa công cộng của B. Khi đó, A sẽ nhận xác nhận khóa công cộng của B từ CA Server và sử dụng khóa công cộng của CA để kiểm tra xem đây có thật sự là khóa

công cộng của B hay không. A sẽ dễ dàng phát hiện được xác nhận khóa công cộng giả của B do C tạo ra nhờ vào khóa công cộng của CA.

Mã hóa khóa công cộng có thể gặp phải vấn đề trong việc phân phối khóa nhưng vấn đề này không nghiêm trọng như trong việc phân phối khóa của mã hóa đối xứng. Sự chứng thực của khóa công cộng có thể được thực hiện bởi một tổ chức trung gian thứ ba đáng tin cậy. Sự bảo đảm về tính xác thực của người sở hữu khóa công cộng được gọi là sự chứng nhận khóa công cộng. Người hay tổ chức chứng nhận khóa công cộng được gọi là tổ chức chứng nhận (CA – Certification Authority).



Hình 10.3. *Mô hình Certification Authority đơn giản*

10.2 Các loại giấy chứng nhận khóa công cộng

Để khóa công cộng của mình được chứng nhận, bên đối tác phải tạo ra một cặp khóa bất đối xứng và gửi cặp khóa này cho tổ chức CA. Bên đối tác phải gửi kèm các thông tin về bản thân như tên hoặc địa chỉ. Khi tổ chức CA đã kiểm tra tính xác thực các thông tin của bên đối tác, nó sẽ phát hành một giấy chứng nhận khóa công cộng cho bên đối tác. Giấy chứng nhận là một tập tin nhị phân có thể dễ dàng chuyển đổi qua mạng máy tính.

Tổ chức CA áp dụng chữ ký điện tử của nó cho giấy chứng nhận khóa công cộng mà nó phát hành. Một tổ chức CA chứng nhận khóa công cộng bằng cách ký nhận nó. Nếu phía đối tác bên kia tin tưởng vào tổ chức CA thì họ có thể tin vào chữ ký của nó.

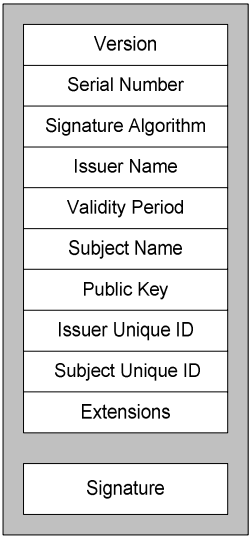
Sau đây là một số loại giấy chứng nhận khóa công cộng.

10.2.1 Chứng nhận X.509

Chứng nhận X.509 là chứng nhận khóa công cộng phổ biến nhất. Hiệp hội viễn thông quốc tê (International Telecommunications Union – ITU) đã chỉ định chuẩn X.509 vào năm 1988 [2] Đây là định dạng phiên bản 1 của chuẩn X.509. Vào năm 1993, phiên bản 2 của chuẩn X.509 được phát hành với 2 trường tên nhận dạng duy nhất được bổ sung. Phiên bản 3 của chuẩn X.509 được bổ sung thêm trường mở rộng đã phát hành vào năm 1997.

Một chứng nhận khóa công cộng kết buộc một khóa công cộng với sự nhận diện của một người (hoặc một thiết bị). Khóa công cộng và tên thực thể sở hữu khóa này là hai mục quan trọng trong một chứng nhận. Hầu hết các trường khác trong chứng

nhận X.509 phiên bản 3 đều đã được chứng tỏ là có ích. Sau đây là thông tin về các trường trong chứng nhận X.509 phiên bản 3 [2]:



Hình 10.4. *Phiên bản 3 của chuẩn chứng nhận X.509*

- *Version*: Chỉ định phiên bản của chứng nhận X.509.
- *Serial Number*: Số loạt phát hành được gán bởi CA. Mỗi CA nên gán một mã số loạt duy nhất cho mỗi giấy chứng nhận mà nó phát hành.
- *Signature Algorithm*: Thuật toán chữ ký chỉ rõ thuật toán mã hóa được CA sử dụng để ký giấy chứng nhận. Trong chứng nhận X.509 thường là sự kết hợp giữa thuật toán băm (chẳng hạn như MD5) và thuật toán khóa công cộng (chẳng hạn như RSA).
- *Issuer Name*: Tên tổ chức CA phát hành giấy chứng nhận, đây là một tên phân biệt theo chuẩn X.500 (X.500 Distinguised Name – X.500 DN). Hai CA không được sử dụng cùng một tên phát hành.
- *Validity Period*: Trường này bao gồm hai giá trị chỉ định khoảng thời gian mà giấy chứng nhận có hiệu lực. Hai phần của trường này là not-before và not-after. Not-before chỉ định thời gian mà chứng nhận này bắt đầu có hiệu lực, Not-after chỉ định thời gian mà chứng nhận hết hiệu lực. Các giá trị thời gian này được đo theo chuẩn thời gian Quốc tế, chính xác đến từng giây.

- *Subject Name*: là một X.500 DN, xác định đối tượng sở hữu giấy chứng nhận mà cũng là sở hữu của khóa công cộng. Một CA không thể phát hành 2 giấy chứng nhận có cùng một Subject Name.

- *Public key*: Xác định thuật toán của khóa công cộng (như RSA) và chứa khóa công cộng được định dạng tuỳ vào kiểu của nó.

- *Issuer Unique ID* và *Subject Unique ID*: Hai trường này được giới thiệu trong X.509 phiên bản 2, được dùng để xác định hai tổ chức CA hoặc hai chủ thể khi chúng có cùng DN. RFC 2459 đề nghị không nên sử dụng hai trường này.

- *Extensions*: Chứa các thông tin bổ sung cần thiết mà người thao tác CA muốn đặt vào chứng nhận. Trường này được giới thiệu trong X.509 phiên bản 3.

- *Signature*: Đây là chữ ký điện tử được tổ chức CA áp dụng. Tổ chức CA sử dụng khóa bí mật có kiểu quy định trong trường thuật toán chữ ký. Chữ ký bao gồm tất cả các phần khác trong giấy chứng nhận. Do đó, tổ chức CA chứng nhận cho tất cả các thông tin khác trong giấy chứng nhận chứ không chỉ cho tên chủ thể và khóa công cộng.

10.2.2 Chứng nhận chất lượng

Đặc điểm chính của các giấy chứng nhận chất lượng là chúng quan tâm quan tới đối tượng mà chúng được phát hành đến. Thực thể cuối sở hữu giấy chứng nhận X.509 hoặc RFC 2459 có thể là một người hoặc một máy. Tuy nhiên, các giấy chứng nhận chất lượng chỉ có thể được phát hành cho con người.

Giấy chứng nhận chất lượng RFC 3039 cung cấp các yêu cầu chi tiết dựa trên nội dung của nhiều trường trong chứng nhận X.509. Các trường tên nhà xuất bản, tên

chủ thể, phần mở rộng đều được cung cấp các yêu cầu nội dung cụ thể. Tên nhà xuất bản của giấy chứng nhận chất lượng phải xác định được tổ chức chịu trách nhiệm phát hành giấy chứng nhận đó. Tên chủ thể của giấy chứng nhận chất lượng phải xác định một con người thật.

10.2.3 Chứng nhận PGP

Đơn giản hơn chứng nhận X.509, giấy chứng nhận PGP không hỗ trợ phần mở rộng.

Giấy chứng nhận X.509 được ký bởi tổ chức CA. Trong khi đó, giấy chứng nhận PGP có thể được ký bởi nhiều cá nhân. Do đó mô hình tin cậy của giấy chứng nhận PGP đòi hỏi bạn phải tin tưởng vào những người ký giấy chứng nhận PGP mà bạn muốn dùng chứ không chỉ tin tưởng vào tổ chức CA phát hành chứng nhận X.509.

10.2.4 Chứng nhận thuộc tính

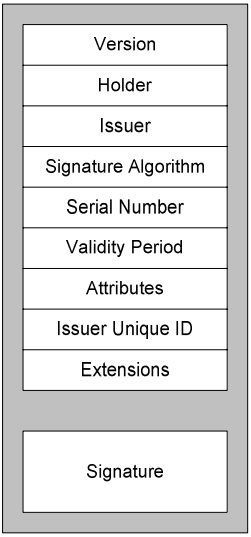
Các giấy chứng nhận thuộc tính (Attribute Certificates – AC [2]) là các giấy chứng nhận điện tử không chứa khóa công cộng. Thay vì thao tác chứng nhận khóa công cộng, ACs chỉ thao tác chứng nhận một tập hợp các thuộc tính.

Các thuộc tính trong một AC được dùng để chuyển các thông tin giấy phép liên quan đến người giữ giấy chứng nhận.

Các chứng nhận thuộc tính phân quyền cho người giữ chúng.

Hệ thống phát hành, sử dụng và hủy ACs là Privilege Management Infrastructure (PMI). Trong PMI, tổ chức chứng nhận thuộc tính Attribute Authority (AA) phát hành ACs. Một AA có thể không giống như một CA.

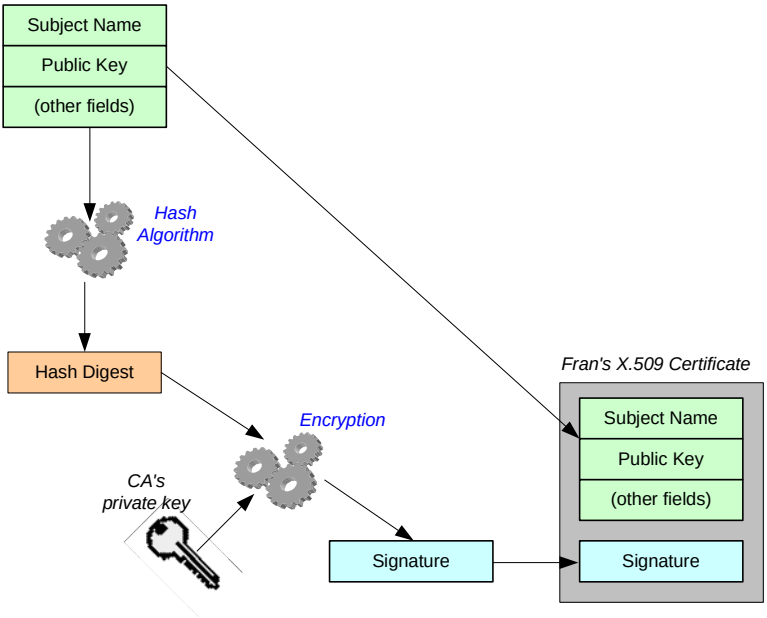
Động cơ chính cho việc sử dụng ACs là để cấp phép. Vì một người dùng có thể chỉ giữ một vai trò nào đó trong tổ chức trong một thời gian ngắn, nên khác với giấy chứng nhận khóa công cộng, AC chỉ có giá trị trong một vài ngày hoặc ngắn hơn.



Hình 10.5. *Phiên bản 2 của cấu trúc chứng nhận thuộc tính*

10.3 Sự chứng nhận và kiểm tra chữ ký

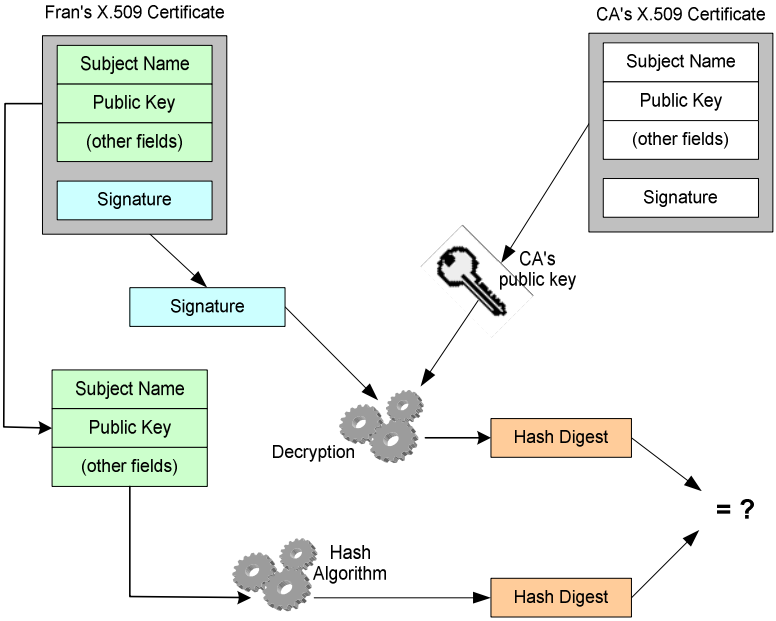
Quá trình chứng nhận chữ ký diễn ra theo hai bước. Đầu tiên, các trường của chứng nhận được ký và nén bởi thuật toán trộn cho trước. Sau đó, kết quả xuất của hàm trộn, được gọi là hash digest, được mã hóa với khóa bí mật của tổ chức CA đã phát hành chứng nhận này.



Hình 10.6. *Quá trình ký chứng nhận*

Chứng nhận của CA phải được ký bởi khóa bí mật. Khóa bí mật này phải thuộc quyền sở hữu của CA, và thông qua việc ký chứng nhận của đối tác A, tổ chức CA này chứng nhận sự hiện hữu của đối tác A.

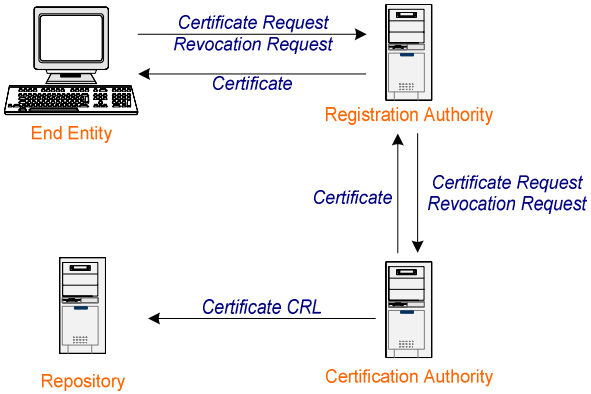
Để có một chứng nhận, một tổ chức CA chỉ cần tạo ra và ký giấy chứng nhận cho chính nó, chứ không cần áp dụng cho một CA khác để chứng nhận. Điều này được hiểu như sự tự chứng nhận (self-certification), và một giấy chứng như thế được gọi là giấy chứng nhận tự ký (self-signed certificate)



Hình 10.7. *Quá trình kiểm tra chứng nhận*

Tổ chức CA sử dụng khóa bí mật của nó để ký giấy chứng nhận của đối tác A và dùng cùng khóa bí mật đó để ký giấy chứng nhận cho chính nó. Một đối tác B có thể kiểm tra cả chữ ký trên giấy chứng nhận của đối tác A và chữ ký trên giấy chứng nhận của tổ chức CA thông qua việc dùng khóa công cộng trong giấy chứng nhận của CA. Cả hai giấy chứng nhận của đối tác A và tổ chức CA tạo nên một chuỗi chứng nhận. Quá trình kiểm tra chứng nhận thường yêu cầu sự kiểm tra của chuỗi chứng nhận. Sự kiểm tra kết thúc khi một giấy chứng nhận tự ký được kiểm tra ở cuối chuỗi [2].

10.4 Các thành phần của một cở sở hạ tầng khóa công cộng



Hình 10.8. *Mô hình PKI cơ bản*

10.4.1 Tổ chức chứng nhận – Certificate Authority (CA)

Tổ chức CA là một thực thể quan trọng duy nhất trong X.509 PKI. (Public key Infrastructure).

Tổ chức CA có nhiệm vụ phát hành, quản lý và hủy bỏ các giấy chứng nhận.

Để thực hiện nhiệm vụ phát hành giấy chứng nhận của mình, CA nhận yêu cầu chứng nhận từ khách hàng. Nó chứng nhận sự tồn tại của khách hàng và kiểm tra nội dung yêu cầu chứng nhận của khách hàng. Sau đó, tổ chức CA tạo ra nội dung chứng nhận mới cho khách hàng và ký nhận cho chứng nhận đó.

Nếu CA có sử dụng nơi lưu trữ chứng nhận thì nó sẽ lưu giấy chứng nhận mới được tạo ra này ở đó. Tổ chức CA cũng phân phối chứng nhận tới khách hàng thông qua email hoặc địa chỉ URL, nơi mà khách hàng có thể lấy chứng nhận.

Khi một giấy chứng nhận cần bị hủy bỏ, tổ chức CA sẽ tạo và quản lý thông tin hủy bỏ cho chứng nhận. Khi hủy bỏ một giấy chứng nhận, CA có thể xóa chứng nhận khỏi nơi lưu trữ hoặc đánh dấu xóa. Tổ chức CA luôn thông báo cho khách hàng rằng chứng nhận của họ đã bị hủy, đồng thời cũng sẽ thêm số loạt của chứng nhận bị hủy vào danh sách các chứng nhận đã bị hủy – Certificate Revocation List (CRL) [2].

10.4.2 Tổ chức đăng ký chứng nhận – Registration Authority (RA)

Một RA là một thực thể tùy chọn được thiết kế để chia sẻ bớt công việc trên CA. Một RA không thể thực hiện bất kỳ một dịch vụ nào mà tổ chức CA của nó không thực hiện được [2].

Các nhiệm vụ chính của RA có thể được chia thành các loại: các dịch vụ chứng nhận và các dịch vụ kiểm tra. Một RA sẽ chứng nhận các yêu cầu khác nhau của các dịch vụ được trực tiếp gửi đến tổ chức CA của nó. Một RA có thể được xác lập để xử lý các yêu cầu chứng nhận, các yêu cầu hủy bỏ chứng nhận thay cho một CA. Sau khi xác minh một yêu cầu, tức là xác định yêu cầu đó đến từ thực thể thích hợp, một RA sẽ kiểm tra tính hợp lệ của nội dung yêu cầu .

Một RA hoạt động như là một xử lý ngoại vi của CA. Một RA chỉ nên phục vụ cho một CA. Trong khi đó, một CA có thể được hỗ trợ bởi nhiều RA.

Một CA có thể còn chịu trách nhiệm trong sự tương tác với nơi lưu trữ chứng nhận và có thể ký CLRs cũng như ký các giấy chứng nhận. Thông qua việc chia sẻ bớt nhiều nhiệm vụ cho các RA, về thực chất một CA có thể làm tăng thời gian trả lời của nó cho các yêu cầu của thực thể cuối.

10.4.3 Kho lưu trữ chứng nhận – Certificate Repository (CR)

Một kho chứng nhận là một cơ sở dữ liệu chứa các chứng nhận được phát hành bởi một CA. Kho có thể được tất cả các người dùng của PKI dùng như nguồn trung tâm các chứng nhận, và do đó là nguồn các khóa công cộng. Một kho cũng có thể được dùng như vị trí trung tâm của các danh sách CRL [2].

10.5 Chu trình quản lý giấy chứng nhận

10.5.1 Khởi tạo

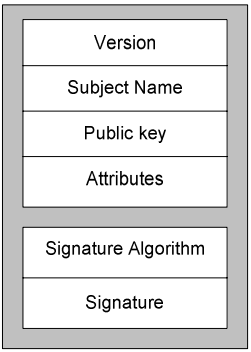
Trước khi yêu cầu một chứng nhận, đối tác phải tìm hiểu về PKI mà mình muốn tham gia. Đối tác phải có địa chỉ của tổ chức CA, của RA và kho lưu trữ nếu chúng tồn tại. Đối tác cũng cần phải có giấy chứng nhận của tổ chức CA, và có thể cả chứng nhận của RA. Cuối cùng, đối tác cần phải có cách tạo ra cặp khóa bất đối xứng và lựa chọn các thuộc tính cho tên phân biệt (Distinguised name- DN [2]) của mình.

10.5.2 Yêu cầu về giấy chứng nhận

Đối tác có thể yêu cầu một chứng nhận từ CA thông qua nhiều kĩ thuật. Trong trường hợp phát sinh lại, đối tác không cần yêu cầu, tổ chức CA sẽ tạo ra một giấy chứng nhận thay cho đối tác. Kĩ thuật này yêu cầu tổ chức CA cũng phải phát sinh cặp khóa bất đối xứng để có được khóa công cộng được kèm theo trong chứng nhận.

Hầu hết các CA sử dụng một trong hai phương thức tiêu chuẩn của yêu cầu chứng nhận : PKCS #10 và CRMF.

Yêu cầu chứng nhận theo chuẩn PKCS #10 [2]:



Hình 10.9. *Mẫu yêu cầu chứng nhận theo chuẩn PKCS#10*

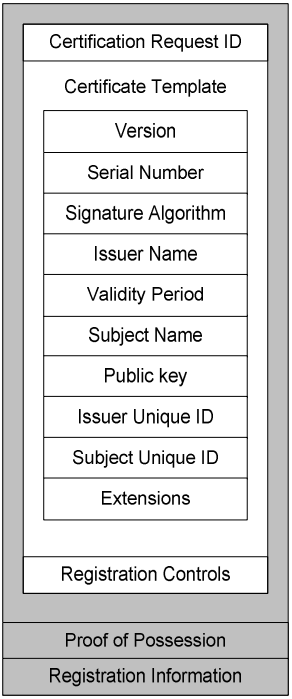
- *Version*: phiên bản của định dạng yêu cầu chứng nhận.
- *Subject Name*: là một X.500 DN, xác định thực thể cuối yêu cầu giấy chứng nhận, người sở hữu khóa công cộng.
- *Public Key*: chỉ ra thuật toán của khóa công cộng, chứa khóa công cộng có định dạng tùy thuộc vào loại của nó.
- *Attributes*: bao gồm các thông tin bổ sung dùng để xác định thực thể cuối.
- *Signature Algorithm*: chỉ ra thuật toán mã hóa được dùng bởi thực thể cuối để ký yêu cầu chứng nhận.
- *Signature*: chữ ký điện tử được áp dụng bởi thực thể cuối yêu cầu chứng nhận.

Yêu cầu chứng nhận theo chuẩn của CRMF [2]:

- *Request ID*: số được sử dụng bởi đối tác và tổ chức CA để liên kết yêu cầu với trả lời chứa chứng nhận được yêu cầu.

- *Certificate Template* : trong yêu cầu PKCS #10, đối tác chỉ có thể chỉ định tên và thông tin khóa công cộng bao gồm trong giấy chứng nhận. Trong CRMF, đối tác có thể bao gồm bất cứ trường nào của chứng nhận X.509 như là một mẫu chứng nhận trong yêu cầu của họ.

- *Controls* : cung cấp cách thức mà đối tác gửi các chi tiết giám sát liên quan tới yêu cầu của họ tới tổ chức CA. Trường này có thể được dùng tương tự như trường thuộc tính trong PKCS #10.



Hình 10.10. *Định dạng thông điệp yêu cầu chứng nhận theo RFC 2511*

- *Proof of Possesion* : CRMF hỗ trợ bốn phương thức để đối tác chứng minh rằng họ sở hữu khóa bí mật tương ứng với khóa công cộng trong yêu cầu. Mỗi phương thức được sử dụng tùy thuộc vào mục đích sử dụng khóa.

- *Registration Information* : là trường tùy chọn chứa các dữ liệu liên quan đến yêu cầu chứng nhận được định dạng trước hoặc được thay thế.

10.5.3 Tạo lại chứng nhận

Đối tác có thể muốn tạo mới lại chứng nhận của mình vì nhiều lý do: giấy chứng nhận hết hạn, thêm thông tin mới vào chứng nhận, xác nhận lại khóa công cộng hiện có, hoặc xác nhận khóa mới. Khi tổ chức CA đáp ứng yêu cầu tạo mới lại này, nó sẽ phát hành cho đối tác một giấy chứng nhận mới và có thể xuất bản giấy chứng nhận mới này vào kho lưu trữ.

Yêu cầu tạo lại thì đơn giản hơn rất nhiều so với yêu cầu chứng nhận nguyên thủy. Khi CA nhận yêu cầu chứng nhận, nó phải xác minh sự tồn tại của đối tác. Nhưng khi đối tác gửi yêu cầu tạo lại, họ có thể bao gồm giấy chứng nhận hiện có và chữ ký sử dụng khóa bí mật tương ứng với chứng nhận đó. Điều đó có thể xem như sự chứng nhận tồn tại của đối tác. Do đó, việc tạo lại chứng nhận thì dễ cho CA đáp ứng hơn.

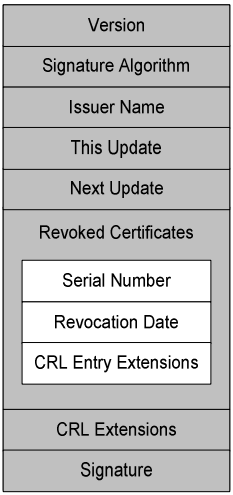
10.5.4 Hủy bỏ chứng nhận

Tất cả các chứng nhận đều có thời hạn sử dụng của nó và chúng cuối cùng sẽ bị hết hạn. Tuy nhiên, cần phải hủy bỏ một chứng nhận trước khi nó bị hết hạn. Lý do chung nhất để hủy một chứng nhận là do sự nhận diện được xác nhận bởi CA đã thay đổi.

Certificate Revocation List (CRL) là cách đầu tiên và thông dụng nhất để phổ biến thông tin hủy bỏ. CRL chứa thông tin thời gian nhằm xác định thời điểm tổ chức CA phát hành nó. CA ký CRL với cùng khóa bí mật được dùng để ký các chứng nhận. Các CRL thường được chứa trong cùng kho với các chứng nhận nhằm dễ dàng cho việc rút trích.

Các CA phát hành các CRL theo định kì, thường là hàng giờ hoặc hàng ngày.

- *Version* : phiên bản định dạng CRL
- *Signature Algorithm* : xác định thuật toán mã hóa được dùng để ký CRL.
- *Issuer Name* : một X.500 DN, xác định tên tổ chức ký CRL.
- *This-Update* : thời điểm CRL được tạo ra.
- *Next-Update* : thời điểm CA tạo ra CRL kế tiếp.



Hình 10.11. *Phiên bản 2 của định dạng danh sách chứng nhận bị hủy*

- *Revoked Certificates* : danh sách các chứng nhận bị hủy bỏ. Mỗi chứng nhận bị hủy có một mục CRL, chứa các thông tin sau:
  - *Serial Number* : mã số chứng nhận
  - *Revocation Date* : ngày hủy bỏ
  - *CRL Entry Extension* : các thông tin bổ sung
- *CRL Extensions* : các thông tin bổ sung hỗ trợ cho việc dùng và quản lý các CRL.
- *Signature* : chữ ký của tổ chức phát hành CRL.

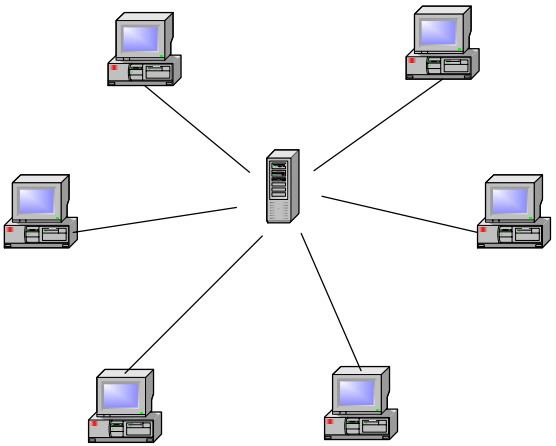
10.5.5 Lưu trữ và khôi phục khóa

Lưu trữ khóa là một dịch vụ được cung cấp bởi nhiều tổ chức CA.

Thông qua việc lưu trữ khóa mã hóa bí mật, khách hàng có thể tránh được trường hợp không giải mã được dữ liệu khi bị mất khóa. Để lưu trữ khóa, khách hàng phải gửi khóa bí mật tới nơi lưu trữ. Bởi vì các yêu cầu lưu trữ hay khôi phục khóa đều phải được xác minh nên các người dùng không thể thao tác trực tiếp đến nơi lưu trữ mà phải thông qua RA hoặc CA.

10.6 Các mô hình CA

10.6.1 Mô hình tập trung



Hình 10.12. *Mô hình CA tập trung*

Tất cả mọi chứng nhận khóa công cộng đều được ký tập trung bởi tổ chức CA và có thể được xác nhận bằng khóa công cộng của CA. Khóa công cộng này được phân phối trực tiếp đến người sử dụng dưới dạng đính kèm trong một chương trình kiểm tra chứng nhận khóa công cộng do tổ chức này cung cấp.

Đây là hướng tiếp cận truyền thống, được sử dụng trong các phiên bản đầu của Netscape Navigator.

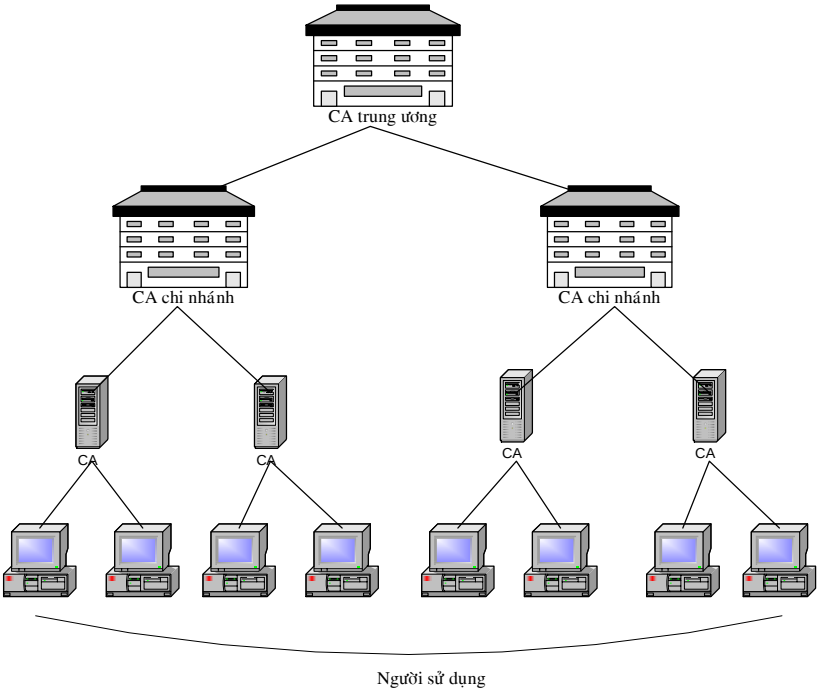
Khuyết điểm chính của mô hình này là hiện tượng "nút cổ chai" tại trung tâm [2].

10.6.2 Mô hình phân cấp

Tổ chức CA được phân ra thành nhiều cấp, tổ chức CA ở cấp cao hơn sẽ ký vào chứng nhận khóa công cộng của các tổ chức CA con trực tiếp của mình. Một chứng nhận khóa công cộng của người sử dụng sẽ được ký bởi một tổ chức CA cục bộ.

Khi một người sử dụng muốn kiểm tra một chứng nhận khóa công cộng, họ cần kiểm tra chứng nhận khóa công cộng của tổ chức CA cục bộ đã ký trên chứng nhận này. Để làm được điều này, cần phải kiểm tra chứng nhận khóa công cộng của tổ chức CA cấp cao hơn đã ký trên chứng nhận khóa công cộng của tổ chức CA cục bộ, … Việc kiểm tra cứ lan truyền lên các cấp cao hơn của tổ chức CA cho đến khi có thể kiểm tra được bằng chứng nhận khóa công cộng của tổ chức CA bằng khóa công cộng được cung cấp trực tiếp cho người sử dụng.

Hệ thống PEM (Privacy Enhanced Mail) và hệ thống DMS (Defense Message System) của Bộ Quốc phòng Hoa Kỳ sử dụng mô hình này.



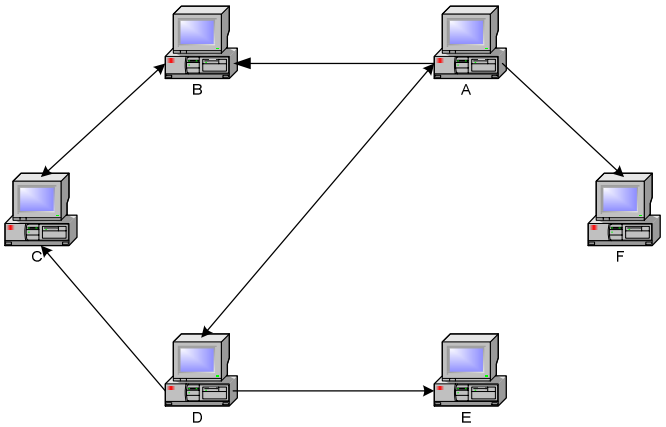
Hình 10.13. *Mô hình CA phân cấp*

10.6.3 Mô hình "Web of Trust"

Bất cứ ai có được chứng nhận khóa công cộng có thể ký vào chứng nhận khóa công cộng của người khác. Đây là hướng tiếp cận trong hệ thống Pertty Good Privacy (PGP) của CA.

Mỗi thành viên tham gia vào hệ thống này có thể đóng vai trò của CA để ký vào chứng nhận khóa công cộng của một thành viên khác. Để có thể tin một chứng nhận khóa công cộng là hợp lệ, ta cần phải có được khóa công cộng của người đã ký trên

chứng nhận này và cần phải đảm bảo rằng người này chỉ ký trên những chứng nhận hợp lệ.



Hình 10.14. *Mô hình "Web of trust"*

❒ Ví dụ: Trong hình sau, A ký vào chứng nhận khóa công cộng của B, D, F; D ký vào chứng nhận khóa công cộng của A, C, E; B và C ký vào chứng nhận khóa công cộng của nhau.

Để đảm bảo an toàn cho hệ thống, mỗi thành viên tham gia vào mô hình này có trách nhiệm đối với chữ ký của mình trên chứng nhận khóa công cộng của các thành viên khác. Để thực hiện điều này, thông thường:

- Tiếp xúc trực tiếp: Các thành viên có thể gặp nhau trực tiếp để trao đổi khóa công cộng của mình và khi đó họ có thể ký vào chứng nhận khóa công cộng của nhau.

- Kỹ thuật "Dấu vân tay" (Fingerprinting): "Dấu vân tay" là chuỗi gồm 128-bits kết quả khi sử dụng hàm băm MD5 đối với mã khóa công cộng.
    - "Dấu vân tay" của một người A sẽ được công bố rộng rãi theo nhiều cách khác nhau, chẳng hạn như trên card visit hay trên trang web của A…
    - Nếu người B chưa tin vào các chữ ký trên chứng nhận khóa công cộng của A thì B co thể sử dụng hàm băm MD5 để kiểm tra lại mã khóa này có phù hợp với "dấu vân tay" của A đã được công bố hay không.
    - Nhờ vào mức độ an toàn của phương pháp MD5, nên việc tìm một mã khóa công cộng khác có cùng giá trị dấu vân tay với một mã khóa cho trước là không khả thi.

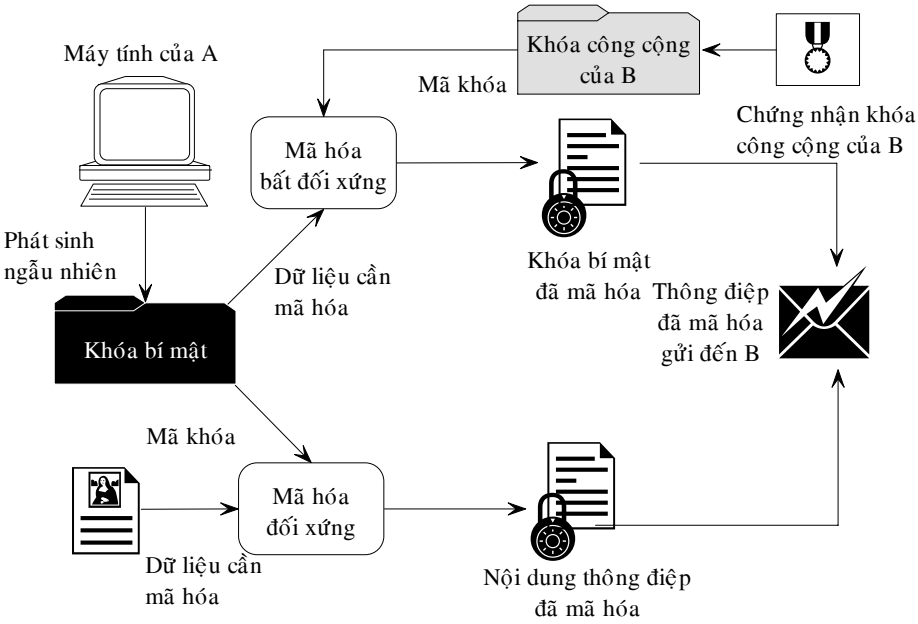
10.7 Ứng dụng "Hệ thống bảo vệ thư điện tử"

10.7.1 Đặt vấn đề

Thư tín điện tử đang ngày càng được sử dụng rộng rãi trong các lĩnh vực đời sống xã hội. Hệ thống thư điện tử cho phép thực hiện các giao dịch thương mại một cách nhanh chóng, hiệu quả, giúp các cơ quan, đơn vị có thể liên lạc dễ dàng với nhau, hỗ trợ việc triển khai các đề án đồng thời tại nhiều địa điểm...

Do tầm quan trọng chiến lược của nội dung chứa đựng bên trong thư điện tử nên yêu cầu đặt ra là phải bảo vệ được tính bí mật và an toàn của các bức thông điệp điện tử này. Quy trình mã hóa và giải mã thư điện tử dưới đây là một trong các giải pháp khả thi nhằm giải quyết bài toán bảo vệ thư tín điện tử ([20], [15]).

10.7.2 Quy trình mã hóa thư điện tử



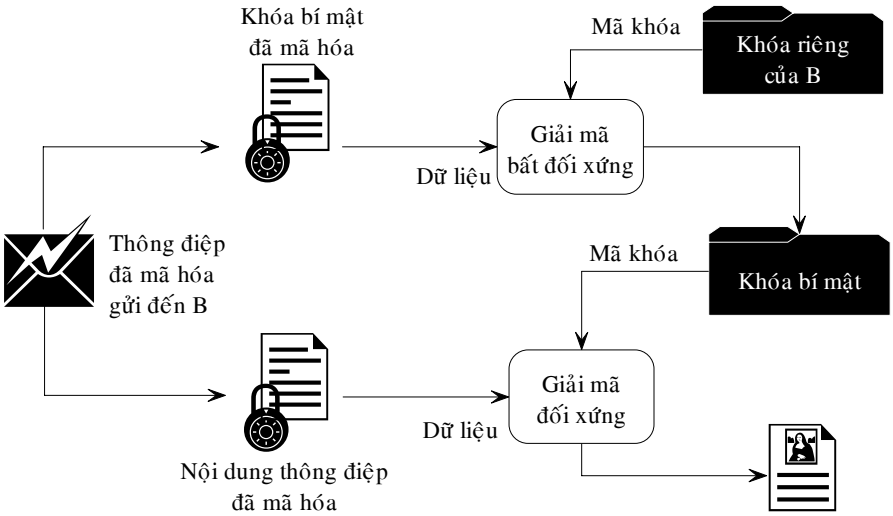
Hình 10.15. *Quy trình mã hóa thư điện tử*

Hình 10.15 thể hiện quy trình mã hóa thư điện tử. Giả sử A muốn gửi một thông điệp điện tử bí mật cho B và giả sử A đã có được khóa công cộng của B (có thể do B trao đổi trực tiếp cho A hay thông qua chứng nhận khóa công cộng của B).

- Giai đoạn 1 – Mã hóa thông điệp bằng một phương pháp mã hóa đối xứng an toàn: Máy tính của A sẽ phát sinh ngẫu nhiên khóa bí mật *K* được sử dụng để mã hóa toàn bộ thông điệp cần gửi đến cho B bằng phương pháp mã hóa đối xứng an toàn được chọn.

- Giai đoạn 2 – Mã hóa khóa bí mật *K* bằng một phương pháp mã hóa bất đối xứng sử dụng khóa công cộng của B.
- Nội dung thông điệp sau khi mã hóa ở giai đoạn 1 cùng với khóa bí mật *K* được mã hóa ở giai đoạn 2 sẽ được gửi cho B dưới dạng một bức thư điện tử.

10.7.3 Quy trình giải mã thư điện tử



Hình 10.16. *Quy trình giải mã thư điện tử*

Hình 10.16 thể hiện quy trình giải mã thư điện tử.

- Giai đoạn 1 – Giải mã khóa bí mật *K*: B sử dụng khóa riêng của mình để giải mã khóa bí mật *K* bằng phương pháp mã hóa bất đối xứng mà A đã dùng để mã hóa khóa *K*.

- Giai đoạn 2 – Giải mã thông điệp của A: B sử dụng khóa bí mật *K* để giải mã toàn bộ thông điệp của A bằng phương pháp mã hóa đối xứng mà A đã dùng.

10.7.4 Nhận xét – Đánh giá

Sử dụng kỹ thuật trên đây, người gửi thư có thể yên tâm rằng bức thư của mình chỉ có thể được giải mã bởi người nhận hợp lệ, bởi vì chỉ có người này mới có được mã khóa riêng để giải mã được khóa bí mật *K* và từ đó giải mã được nội dung của thông điệp.

Phụ lục A S-box của thuật toán MARS

```
WORD Sbox[ ] = {
       0x09d0c479, 0x28c8ffe0, 0x84aa6c39, 0x9dad7287,
       0x7dff9be3, 0xd4268361, 0xc96da1d4, 0x7974cc93,
       0x85d0582e, 0x2a4b5705, 0x1ca16a62, 0xc3bd279d,
       0x0f1f25e5, 0x5160372f, 0xc695c1fb, 0x4d7ff1e4,
       0xae5f6bf4, 0x0d72ee46, 0xff23de8a, 0xb1cf8e83,
       0xf14902e2, 0x3e981e42, 0x8bf53eb6, 0x7f4bf8ac,
       0x83631f83, 0x25970205, 0x76afe784, 0x3a7931d4,
       0x4f846450, 0x5c64c3f6, 0x210a5f18, 0xc6986a26,
       0x28f4e826, 0x3a60a81c, 0xd340a664, 0x7ea820c4,
       0x526687c5, 0x7eddd12b, 0x32a11d1d, 0x9c9ef086,
       0x80f6e831, 0xab6f04ad, 0x56fb9b53, 0x8b2e095c,
       0xb68556ae, 0xd2250b0d, 0x294a7721, 0xe21fb253,
       0xae136749, 0xe82aae86, 0x93365104, 0x99404a66,
       0x78a784dc, 0xb69ba84b, 0x04046793, 0x23db5c1e,
       0x46cae1d6, 0x2fe28134, 0x5a223942, 0x1863cd5b,
       0xc190c6e3, 0x07dfb846, 0x6eb88816, 0x2d0dcc4a,
       0xa4ccae59, 0x3798670d, 0xcbfa9493, 0x4f481d45,
       0xeafc8ca8, 0xdb1129d6, 0xb0449e20, 0x0f5407fb,
       0x6167d9a8, 0xd1f45763, 0x4daa96c3, 0x3bec5958,
       0xababa014, 0xb6ccd201, 0x38d6279f, 0x02682215,
       0x8f376cd5, 0x092c237e, 0xbfc56593, 0x32889d2c,
       0x854b3e95, 0x05bb9b43, 0x7dcd5dcd, 0xa02e926c,
       0xfae527e5, 0x36a1c330, 0x3412e1ae, 0xf257f462,
       0x3c4f1d71, 0x30a2e809, 0x68e5f551, 0x9c61ba44,
       0x5ded0ab8, 0x75ce09c8, 0x9654f93e, 0x698c0cca,
       0x243cb3e4, 0x2b062b97, 0x0f3b8d9e, 0x00e050df,
       0xfc5d6166, 0xe35f9288, 0xc079550d, 0x0591aee8,
       0x8e531e74, 0x75fe3578, 0x2f6d829a, 0xf60b21ae,
       0x95e8eb8d, 0x6699486b, 0x901d7d9b, 0xfd6d6e31,
       0x1090acef, 0xe0670dd8, 0xdab2e692, 0xcd6d4365,
       0xe5393514, 0x3af345f0, 0x6241fc4d, 0x460da3a3,
       0x7bcf3729, 0x8bf1d1e0, 0x14aac070, 0x1587ed55,
       0x3afd7d3e, 0xd2f29e01, 0x29a9d1f6, 0xefb10c53,
       0xcf3b870f, 0xb414935c, 0x664465ed, 0x024acac7,
       0x59a744c1, 0x1d2936a7, 0xdc580aa6, 0xcf574ca8,
       0x040a7a10, 0x6cd81807, 0x8a98be4c, 0xaccea063,
       0xc33e92b5, 0xd1e0e03d, 0xb322517e, 0x2092bd13,
       0x386b2c4a, 0x52e8dd58, 0x58656dfb, 0x50820371,
       0x41811896, 0xe337ef7e, 0xd39fb119, 0xc97f0df6,
       0x68fea01b, 0xa150a6e5, 0x55258962, 0xeb6ff41b,
       0xd7c9cd7a, 0xa619cd9e, 0xbcf09576, 0x2672c073,
       0xf003fb3c, 0x4ab7a50b, 0x1484126a, 0x487ba9b1,
       0xa64fc9c6, 0xf6957d49, 0x38b06a75, 0xdd805fcd,
```

```
0x63d094cf, 0xf51c999e, 0x1aa4d343, 0xb8495294,
0xce9f8e99, 0xbffcd770, 0xc7c275cc, 0x378453a7,
0x7b21be33, 0x397f41bd, 0x4e94d131, 0x92cc1f98,
0x5915ea51, 0x99f861b7, 0xc9980a88, 0x1d74fd5f,
0xb0a495f8, 0x614deed0, 0xb5778eea, 0x5941792d,
0xfa90c1f8, 0x33f824b4, 0xc4965372, 0x3ff6d550,
0x4ca5fec0, 0x8630e964, 0x5b3fbbd6, 0x7da26a48,
0xb203231a, 0x04297514, 0x2d639306, 0x2eb13149,
0x16a45272, 0x532459a0, 0x8e5f4872, 0xf966c7d9,
0x07128dc0, 0x0d44db62, 0xafc8d52d, 0x06316131,
0xd838e7ce, 0x1bc41d00, 0x3a2e8c0f, 0xea83837e,
0xb984737d, 0x13ba4891, 0xc4f8b949, 0xa6d6acb3,
0xa215cdce, 0x8359838b, 0x6bd1aa31, 0xf579dd52,
0x21b93f93, 0xf5176781, 0x187dfdde, 0xe94aeb76,
0x2b38fd54, 0x431de1da, 0xab394825, 0x9ad3048f,
0xdfea32aa, 0x659473e3, 0x623f7863, 0xf3346c59,
0xab3ab685, 0x3346a90b, 0x6b56443e, 0xc6de01f8,
0x8d421fc0, 0x9b0ed10c, 0x88f1a1e9, 0x54c1f029,
0x7dead57b, 0x8d7ba426, 0x4cf5178a, 0x551a7cca,
0x1a9a5f08, 0xfcd651b9, 0x25605182, 0xe11fc6c3,
0xb6fd9676, 0x337b3027, 0xb7c8eb14, 0x9e5fd030,
0x6b57e354, 0xad913cf7, 0x7e16688d, 0x58872a69,
0x2c2fc7df, 0xe389ccc6, 0x30738df1, 0x0824a734,
0xe1797a8b, 0xa4a8d57b, 0x5b5d193b, 0xc8a8309b,
0x73f9a978, 0x73398d32, 0x0f59573e, 0xe9df2b03,
0xe8a5b6c8, 0x848d0704, 0x98df93c2, 0x720a1dc3,
0x684f259a, 0x943ba848, 0xa6370152, 0x863b5ea3,
0xd17b978b, 0x6d9b58ef, 0x0a700dd4, 0xa73d36bf,
0x8e6a0829, 0x8695bc14, 0xe35b3447, 0x933ac568,
0x8894b022, 0x2f511c27, 0xddfbcc3c, 0x006662b6,
0x117c83fe, 0x4e12b414, 0xc2bca766, 0x3a2fec10,
0xf4562420, 0x55792e2a, 0x46f5d857, 0xceda25ce,
0xc3601d3b, 0x6c00ab46, 0xefac9c28, 0xb3c35047,
0x611dfee3, 0x257c3207, 0xfdd58482, 0x3b14d84f,
0x23becb64, 0xa075f3a3, 0x088f8ead, 0x07adf158,
0x7796943c, 0xfacabf3d, 0xc09730cd, 0xf7679969,
0xda44e9ed, 0x2c854c12, 0x35935fa3, 0x2f057d9f,
0x690624f8, 0x1cb0bafd, 0x7b0dbdc6, 0x810f23bb,
0xfa929a1a, 0x6d969a17, 0x6742979b, 0x74ac7d05,
0x010e65c4, 0x86a3d963, 0xf907b5a0, 0xd0042bd3,
0x158d7d03, 0x287a8255, 0xbba8366f, 0x096edc33,
0x21916a7b, 0x77b56b86, 0x951622f9, 0xa6c5e650,
0x8cea17d1, 0xcd8c62bc, 0xa3d63433, 0x358a68fd,
0x0f9b9d3c, 0xd6aa295b, 0xfe33384a, 0xc000738e,
0xcd67eb2f, 0xe2eb6dc2, 0x97338b02, 0x06c9f246,
0x419cf1ad, 0x2b83c045, 0x3723f18a, 0xcb5b3089,
0x160bead7, 0x5d494656, 0x35f8a74b, 0x1e4e6c9e,
```

```
        0x000399bd, 0x67466880, 0xb4174831, 0xacf423b2,
        0xca815ab3, 0x5a6395e7, 0x302a67c5, 0x8bdb446b,
        0x108f8fa4, 0x10223eda, 0x92b8b48b, 0x7f38d0ee,
        0xab2701d4, 0x0262d415, 0xaf224a30, 0xb3d88aba,
        0xf8b2c3af, 0xdaf7ef70, 0xcc97d3b7, 0xe9614b6c,
        0x2baebff4, 0x70f687cf, 0x386c9156, 0xce092ee5,
        0x01e87da6, 0x6ce91e6a, 0xbb7bcc84, 0xc7922c20,
        0x9d3b71fd, 0x060e41c6, 0xd7590f15, 0x4e03bb47,
        0x183c198e, 0x63eeb240, 0x2ddbf49a, 0x6d5cba54,
        0x923750af, 0xf9e14236, 0x7838162b, 0x59726c72,
        0x81b66760, 0xbb2926c1, 0x48a0ce0d, 0xa6c0496d,
        0xad43507b, 0x718d496a, 0x9df057af, 0x44b1bde6,
        0x054356dc, 0xde7ced35, 0xd51a138b, 0x62088cc9,
        0x35830311, 0xc96efca2, 0x686f86ec, 0x8e77cb68,
        0x63e1d6b8, 0xc80f9778, 0x79c491fd, 0x1b4c67f2,
        0x72698d7d, 0x5e368c31, 0xf7d95e2e, 0xa1d3493f,
        0xdcd9433e, 0x896f1552, 0x4bc4ca7a, 0xa6d1baf4,
        0xa5a96dcc, 0x0bef8b46, 0xa169fda7, 0x74df40b7,
        0x4e208804, 0x9a756607, 0x038e87c8, 0x20211e44,
        0x8b7ad4bf, 0xc6403f35, 0x1848e36d, 0x80bdb038,
        0x1e62891c, 0x643d2107, 0xbf04d6f8, 0x21092c8c,
        0xf644f389, 0x0778404e, 0x7b78adb8, 0xa2c52d53,
        0x42157abe, 0xa2253e2e, 0x7bf3f4ae, 0x80f594f9,
        0x953194e7, 0x77eb92ed, 0xb3816930, 0xda8d9336,
        0xbf447469, 0xf26d9483, 0xee6faed5, 0x71371235,
        0xde425f73, 0xb4e59f43, 0x7dbe2d4e, 0x2d37b185,
        0x49dc9a63, 0x98c39d98, 0x1301c9a2, 0x389b1bbf,
        0x0c18588d, 0xa421c1ba, 0x7aa3865c, 0x71e08558,
        0x3c5cfcaa, 0x7d239ca4, 0x0297d9dd, 0xd7dc2830,
        0x4b37802b, 0x7428ab54, 0xaeee0347, 0x4b3fbb85,
        0x692f2f08, 0x134e578e, 0x36d9e0bf, 0xae8b5fcf,
        0xedb93ecf, 0x2b27248e, 0x170eb1ef, 0x7dc57fd6,
        0x1e760f16, 0xb1136601, 0x864e1b9b, 0xd7ea7319,
        0x3ab871bd, 0xcfa4d76f, 0xe31bd782, 0x0dbeb469,
        0xabb96061, 0x5370f85d, 0xffb07e37, 0xda30d0fb,
        0xebc977b6, 0x0b98b40f, 0x3a4d0fe6, 0xdf4fc26b,
        0x159cf22a, 0xc298d6e2, 0x2b78ef6a, 0x61a94ac0,
        0xab561187, 0x14eea0f0, 0xdf0d4164, 0x19af70ee
};
```

Phụ lục B Các hoán vị sử dụng trong thuật toán Serpent

Hoán vị đầu tiên (Initial Permutation – IP)

0	32	64	96	1	33	65	97	2	34	66	98	3	35	67	99
4	36	68	100	5	37	69	101	6	38	70	102	7	39	71	103
8	40	72	104	9	41	73	105	10	42	74	106	11	43	75	107
12	44	76	108	13	45	77	109	14	46	78	110	15	47	79	111
16	48	80	112	17	49	81	113	18	50	82	114	19	51	83	115
20	52	84	116	21	53	85	117	22	54	86	118	23	55	87	119
24	56	88	120	25	57	89	121	26	58	90	122	27	59	91	123
28	60	92	124	29	61	93	125	30	62	94	126	31	63	95	127

Hoán vị cuối cùng (Final Permutation – FP)

0	4	8	12	16	20	24	28	32	36	40	44	48	52	56	60
64	68	72	76	80	84	88	92	96	100	104	108	112	116	120	124
1	5	9	13	17	21	25	29	33	37	41	45	49	53	57	61
65	69	73	77	81	85	89	93	97	101	105	109	113	117	121	125
2	6	10	14	18	22	26	30	34	38	42	46	50	54	58	62
66	70	74	78	82	86	90	94	98	102	106	110	114	118	122	126
3	7	11	15	19	23	27	31	35	39	43	47	51	55	59	63
67	71	75	79	83	87	91	95	99	103	107	111	115	119	123	127

Phụ lục C S-box sử dụng trong thuật toán Serpent

S-box sử dụng trong thuật toán Serpent

S0	3	8	15	1	10	6	5	11	14	13	4	2	7	0	9	12
S1	15	12	2	7	9	0	5	10	1	11	14	8	6	13	3	4
S2	8	6	7	9	3	12	10	15	13	1	14	4	0	11	5	2
S3	0	15	11	8	12	9	6	3	13	1	2	4	10	7	5	14
S4	1	15	8	3	12	0	11	6	2	5	4	10	9	14	7	13
S5	15	5	2	11	4	10	9	12	0	3	14	8	13	6	7	1
S6	7	2	12	5	8	4	6	11	14	9	1	15	13	3	10	0
S7	1	13	15	0	14	8	2	11	7	4	12	10	9	3	5	6

S-box nghịch đảo sử dụng trong thuật toán Serpent

InvS0	13	3	11	0	10	6	5	12	1	14	4	7	15	9	8	2
InvS1	5	8	2	14	15	6	12	3	11	4	7	9	1	13	10	0
InvS2	12	9	15	4	11	14	1	2	0	3	6	13	5	8	10	7
InvS3	0	9	10	7	11	14	6	13	3	5	12	2	4	8	15	1
InvS4	5	0	8	3	10	9	7	14	2	12	11	6	4	15	13	1
InvS5	8	15	2	9	4	1	13	14	11	6	5	3	7	12	10	0
InvS6	15	10	1	13	5	3	6	0	4	9	14	7	2	12	8	11
InvS7	3	0	6	13	9	14	15	8	5	12	11	7	10	1	4	2

Phụ lục D S-box của thuật toán Rijndael

Bảng D.1. *Bảng thay thế S-box cho giá trị {xy} ở dạng thập lục phân.*

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	B5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	Df
	f	8c	a1	89	0d	Bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Bảng D.2. *Bảng thay thế nghịch đảo cho giá trị {xy} ở dạng thập lục phân.*

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
	1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
	2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
	3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
	4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
	5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
	6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
	7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
	8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
	9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
	a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
	b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
	c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
	d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
	e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
	f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

Phụ lục E Hằng số và giá trị khởi tạo của SHA

E.1 Hằng số sử dụng trong SHA

E.1.1 Hằng số của SHA-1

SHA-1 sử dụng dãy 80 từ 32 bit là hằng số K_0, K_1, ..., K_{79}

$$\mathrm{K}_t = \begin{cases} \texttt{5a82799} & 0 \le t \le 19 \\ \texttt{6ed9eba1} & 20 \le t \le 39 \\ \texttt{8f1bbcdc} & 40 \le t \le 59 \\ \texttt{ca62c1d6} & 60 \le t \le 79 \end{cases}$$

E.1.2 Hằng số của SHA-224 và SHA-256

SHA-224 và SHA-256 sử dụng dãy 64 từ 32 bit là hằng số $K_0^{\{256\}}, K_1^{\{256\}}, ..., K_{63}^{\{256\}}$. Những từ này biểu diễn 32 bit đầu tiên của phần phân số của căn bậc ba của 64 số nguyên tố đầu tiên. Các hằng số bao gồm (theo thứ tự từ trái sang phải)

```
428a2f98 71374491 b5c0fbcf e9b5dba5
3956c25b 59f111f1 923f82a4 ab1c5ed5
d807aa98 12835b01 243185be 550c7dc3
72be5d74 80deb1fe 9bdc06a7 c19bf174
e49b69c1 efbe4786 0fc19dc6 240ca1cc
2de92c6f 4a7484aa 5cb0a9dc 76f988da
27b70a85 2e1b2138 4d2c6dfc 53380d13
650a7354 766a0abb 81c2c62e 92722c85
a2bfe8a1 a81a664b c24b8b70 c76c51a3
d192e819 d6990624 f4083585 106aa070
19a4c116 18376c08 2748774c 34b0bcb5
391c0cb3 4ed8aa4a 5b9cca4f 682e6ff3
748f82ee 78a5636f 84c87814 8cc70208
90befffa a4506ceb bef9a3f7 c67178f2
```

E.1.3 Hằng số của SHA-384 và SHA-512

SHA-384 và SHA-512 sử dụng cùng dãy 80 từ 64 bit là hằng số $K_0^{\{512\}}, K_1^{\{512\}}, ..., K_{79}^{\{512\}}$. Những từ này biểu diễn 64 bit đầu tiên của phần phân số của căn bậc ba của 80 số nguyên tố đầu tiên. Các hằng số bao gồm (theo thứ tự từ trái sang phải)

428a2f98d728ae22	7137449123ef65cd
b5c0fbcfec4d3b2f	e9b5dba58189dbbc
3956c25bf348b538	59f111f1b605d019
923f82a4af194f9b	ab1c5ed5da6d8118
d807aa98a3030242	12835b0145706fbe
243185be4ee4b28c	550c7dc3d5ffb4e2
72be5d74f27b896f	80deb1fe3b1696b1
9bdc06a725c71235	c19bf174cf692694
e49b69c19ef14ad2	efbe4786384f25e3
0fc19dc68b8cd5b5	240ca1cc77ac9c65
2de92c6f592b0275	4a7484aa6ea6e483
5cb0a9dcbd41fbd4	76f988da831153b5
983e5152ee66dfab	a831c66d2db43210
b00327c898fb213f	bf597fc7beef0ee4
c6e00bf33da88fc2	d5a79147930aa725
06ca6351e003826f	142929670a0e6e70
27b70a8546d22ffc	2e1b21385c26c926
4d2c6dfc5ac42aed	53380d139d95b3df
650a73548baf63de	766a0abb3c77b2a8
81c2c92e47edaee6	92722c851482353b
a2bfe8a14cf10364	a81a664bbc423001
c24b8b70d0f89791	c76c51a30654be30
d192e819d6ef5218	d69906245565a910

f40e35855771202a	106aa07032bbd1b8
19a4c116b8d2d0c8	1e376c085141ab53
2748774cdf8eeb99	34b0bcb5e19b48a8
391c0cb3c5c95a63	4ed8aa4ae3418acb
5b9cca4f7763e373	682e6ff3d6b2b8a3
748f82ee5defb2fc	78a5636f43172f60
84c87814a1f0ab72	8cc702081a6439ec
90befffa23631e28	a4506cebde82bde9
bef9a3f7b2c67915	c67178f2e372532b
ca273eceea26619c	d186b8c721c0c207
eada7dd6cde0eb1e	f57d4f7fee6ed178
06f067aa72176fba	0a637dc5a2c898a6
113f9804bef90dae	1b710b35131c471b
28db77f523047d84	32caab7b40c72493
3c9ebe0a15c9bebc	431d67c49c100d4c
4cc5d4becb3e42b6	597f299cfc657e2a
5fcb6fab3ad6faec	6c44198c4a475817

E.2 Giá trị khởi tạo trong SHA

SHA – 1:

$H_0^{(0)} = 67452301$

$H_1^{(0)} = efcdab89$

$H_2^{(0)} = 98badcfe$

$H_3^{(0)} = 10325476$

$H_4^{(0)} = c3d2e1f0$

SHA – 224:

$H_0^{(0)} = \texttt{c1059ed8}$

$H_1^{(0)} = \texttt{367cd507}$

$H_2^{(0)} = \texttt{3070dd17}$

$H_3^{(0)} = \texttt{f70e5939}$

$H_4^{(0)} = \texttt{ffc00b31}$

$H_5^{(0)} = \texttt{68581511}$

$H_6^{(0)} = \texttt{64f98fa7}$

$H_7^{(0)} = \texttt{befa4fa4}$

SHA – 256:

$H_0^{(0)} = \texttt{6a09e667}$

$H_1^{(0)} = \texttt{bb67ae85}$

$H_2^{(0)} = \texttt{3c6ef372}$

$H_3^{(0)} = \texttt{a54ff53a}$

$H_4^{(0)} = \texttt{510e527f}$

$H_5^{(0)} = \texttt{9b05688c}$

$H_6^{(0)} = \texttt{1f83d9ab}$

$H_7^{(0)} = \texttt{5be0cd19}$

SHA-384:

$H_0^{(0)} = \texttt{cbbb9d5dc1059ed8}$

$H_1^{(0)} = \texttt{629a292a367cd507}$

$H_2^{(0)} = \texttt{9159015a3070dd17}$

$H_3^{(0)} = \texttt{152fecd8f70e5939}$

$H_4^{(0)} = \texttt{67332667ffc00b31}$

$H_5^{(0)} = \texttt{8eb44a8768581511}$

$H_6^{(0)} = \texttt{db0c2e0d64f98fa7}$

$H_7^{(0)} = \texttt{47b5481dbefa4fa4}$

SHA – 512:

$H_0^{(0)}$ = 6a09e667f3 bcc908

$H_1^{(0)}$ = bb67ae8584 faa73b

$H_2^{(0)}$ = 3c6ef372fe 94f82b

$H_3^{(0)}$ = a54ff53a5f 1d36f1

$H_4^{(0)}$ = 510e527fad e682d1

$H_5^{(0)}$ = 9b05688c2b 3e6c1f

$H_6^{(0)}$ = 1f83d9abfb 41bd6b

$H_7^{(0)}$ = 5be0cd1913 7e2179

Tài liệu tham khảo

[1] Ross Anderson, Eli Biham, Lars Knudsen (1999)**,** *Serpent: A Proposal for the Advanced Encryption Standard.*

[2] Mohan Atreya, Ben Hammond, Stephen Paine, Paul Starrett, Stephen Wu (2002), *Digital Signatures*, RSA.

[3] E. Biham, A. Shamir (1991), *Differential cryptanalysis of DES-like cryptosystems*, Journal of Cryptology, Vol. 4, No. 1, pp. 3-72.

[4] E. Biham (1993), *New types of cryptanalytic attacks using related keys*, Advances in Cryptology, Proceedings Eurocrypt'93, LNCS 765, T. Helleseth, Ed., Springer-Verlag, pp. 398-409.

[5] Carolynn Burwick, Don Coppersmith, Edward D'Avignon, Rosario Gennaro, Shai Halevi, Charanjit Jutla, Stephen M.Matyas Jr., Luke O'Connor, Mohammad Peyravian, David Safford, Nevenko Zunic (1999), *MARS – a candidate cipher for AES,* IBM Corporation.

[6] Bram Cohen (2001), *AES-Hash*.

[7] Nicolas Courtois, Josef Pieprzyk(2002), *Cryptanalysis of Block Ciphers with Overdefined Systems of Equations*, ASIACRYPT 2002, pp267–287

[8] J. Daemen, V. Rijmen (1999), *AES Proposal: Rijndael, AES Algorithm Submission*.

[9] J. Daemen, L.R. Knudsen, V. Rijmen (1997), *The block cipher Square*, Fast Software Encryption, LNCS 1267, E. Biham, Ed., Springer-Verlag, tr. 149-165.

[10] J. Daemen (1995), *Cipher and hash function design strategies based on linear and differential cryptanalysis*, Doctoral Dissertation, K.U.Leuven.

[11] Dương Anh Đức, Trần Minh Triết, Lương Hán Cơ (2001), *The 256/384/512-bit version of the Rijndael Block Cipher*, Tạp chí Tin học và Điều khiển, Việt Nam, tập 17, số 4, tr. 45-56.

[12] Duong Anh Duc, Tran Minh Triet, Luong Han Co (2002), *The extended Rijndael-like Block Ciphers*, International Conference on Information Technology: Coding and Computing – 2002, The Orleans, Las Vegas, Nevada, USA, pp. 183-188.

[13] Duong Anh Duc, Tran Minh Triet, Luong Han Co (2002), *The Advanced Encryption Standard And Its Application in the examination security in Vietnam*, International Conference on Information Technology: Coding and Computing – 2002, The Orleans, Las Vegas, Nevada, USA, pp. 171-176.

[14] Duong Anh Duc, Tran Minh Triet, Luong Han Co (2001), *The extended versions of the Advanced Encryption Standard*, Workshop on Applied Cryptology: Coding Theory and Data Integrity, Singapore.

[15] Duong Anh Duc, Tran Minh Triet, Luong Han Co (2001), *Applying the Advanced Encryption Standard and its variants in Secured Electronic-Mail System In Vietnam*, Workshop on Applied Cryptology: Coding Theory and Data Integrity, Singapore.

[16] Duong Anh Duc, Tran Minh Triet, Luong Han Co (2001), *The extended version of the Rijndael Block Cipher*, Journal of Institute of Mathematics and Computer Sciences), India, Vol. 12, No. 2, pp. 201-218.

[17] Duong Anh Duc, Hoang Van Kiem, Tran Minh Triet, Luong Han Co (2002), *The Advanced Encryption Standard and Its Applications in the Examination Security Process in Vietnam*, International Conference on Computational Mathematics and Modelling CMM 2002, Thailand.

[18] Dương Anh Đức, Trần Minh Triết, Đặng Tuân, Hồ Ngọc Lâm (2002), *Watermarking - Tổng quan và ứng dụng trong các hệ thống quản lý và bảo vệ sản phẩm trí tuệ*, kỷ yếu Hội nghị khoa học (lần 3) trường Đại học Khoa Học Tự Nhiên, Đại học Quốc gia Thành phố Hồ Chí Minh, tr. 130-140

[19] Dương Anh Đức, Nguyễn Thanh Sơn, Trần Minh Triết (2004), *Bảo mật dữ liệu với kỹ thuật AES-DCT watermarking*, tạp chí Khoa học Công nghệ ĐHQG, số 4-5, tập 7, tr. 77-82.

[20] Dương Anh Đức, Trần Minh Triết, Lương Hán Cơ (2001), *Ứng dụng chuẩn mã hóa AES và các phiên bản mở rộng vào Hệ thống Thư điện tử an toàn tại Việt Nam*, Hội nghị khoa học kỷ niệm 25 năm Viện Công Nghệ Thông Tin, Hà Nội, Việt Nam, tr. 46-53.

[21] H. Feistel (1973), *Cryptography and computer privacy,* Scientific American, Vol. 228, No. 5, pp. 15-23.

[22] H. Feistel, W.A. Notz, J.L. Smith (1975), *Some cryptographic techniques for machine to machine data communications,* Proceedings of the IEEE, Vol. 63, No. 11, pp. 1545-1554.

[23] FIPS (2001), *Announcing the Advanced Encryption Standard (AES)*

[24] FIPS (2004), *Announcing the Secure Hash Standard*.

[25] FIPS (1993), *Data Encryption Standard (DES).*

[26] FIPS (2000), *Announcing the Digital Signature Standard (DSS)*

[27] IEEE-P1363 (1999), *Standard Specifications for Public Key Cryptography*.

[28] T. Jakobsen, L.R. Knudsen (1997), *The interpolation attack on block ciphers*, Fast Software Encryption, LNCS 1267, E. Biham, Ed., Springer-Verlag, pp. 28-40.

[29] Liam Keliher (2003), *Linear Cryptanalysis of Substitution-Permutation Networks*, PhD. Thesis, Queen's University, Kingston, Ontario, Canada.

[30] J. Kelsey, B. Schneier, D. Wagner (1996), *Key-schedule cryptanalysis of IDEA, GDES, GOST, SAFER, and Triple-DES*, Advances in Cryptology, pp. 237-252.

[31] J. Kelsey, B. Schneier, D. Wagner, Chris Hall (1998), *Cryptanalytic attacks on pseudorandom number generators*, Fast Software Encryption, LNCS 1372, S. Vaudenay, Ed., Springer-Verlag, pp. 168-188.

[32] M. Matsui (1994), *Linear cryptanalysis method for DES cipher*, Advances in Cryptology, Proceedings Eurocrypt'93, LNCS 765, T. Helleseth, Ed., Springer-Verlag, tr. 386-397.

[33] Alfred Menezes (2000), *Comparing the Security of ECC and RSA,* University of Waterloo.

[34] NIST (1999), *Recommended elliptic curves for federal government use.*

[35] Henna Pietilainen (2000), *Elliptic curve cryptography on smart card*, Helsinki University of Technology.

[36] Bart Preneel (2004), *The Davies-Mayer Hash Function*, K.U. Leuven.

[37] Eric Rescorla (2001), *SSL&TLS Designing and Building Secure Systems*.

[38] Ronald L.Rivest**,** M.J.B. Robshaw**,** R. Sidney**,** Y. L. Yin (1998)**,** *The RC6 Block Cipher: A simple fast secure AES proposal***.**

[39] RSA Data Security Inc (1997), "*RSA Laboratories FAQ on Cryptography*," "*RSA Laboratories Technical Reports*," "*RSA Laboratories Security Bulletins*," và "*CrytoBytes Newsletter*".

[40] Bruce Schneier (1995), *Applied Cryptography: Protocols, Algorithms, and Source Code in C, 2nd Edition*, John Wiley & Sons, Inc.

[41] C.E. Shannon (1949), *Communication theory of secrecy systems,* Bell System Technical Journal, Vol. 28, no. 4, pp. 656-715.

[42] Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, Niels Ferguson (1998)**,** *Twofish: A 128-Bit Block Cipher*.

[43] Richard E. Smith (1997), *Internet Cryptography*, Addison-Wesley.

[44] W. Stallings (2003), *Cryptography and Network Security: Principles and Practice, Third Edition,* Prentice Hall.

[45] Douglas R. Stison (1995), *Cryptography – Theory and Practice*, CRC Press.

[46] Tara M. Swaminatha, Charles R. Elden (2003), *Wireless Security and Privacy: Best Practices and Design Techniques,* Addison Wesley.

[47] Tran Minh Triet, Duong Anh Duc (2004), *Applying the Robust Psychoacoustic Audio Watermarking Technique in Internet Digital Traditional Music Museum in Vietnam*, ICCST 2004, 38th IEEE International Carnahan Conference on Security Technology, USA.

[48] Trần Minh Triết (2004), *Nghiên cứu một số vấn đề về bảo vệ thông tin và ứng dụng*, Luận văn Thạc sĩ Tin học, Đại học Khoa học Tự nhiên, Đại học Quốc gia thành phố Hồ Chí Minh.

[49] Xiaoyun Wang, Dengguo Feng, Xuejia Lai, Hongbo Yu (2004), *Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD*, International Association for Cryptologic Research.

[50] Bo-Yin Yang, Jiun-Ming Chen (2004), *Theoretical Analysis of XL over Small Fields*, ACISP 2004, Lecture Notes in Computer Science vol. 3108, pp.277-288.

Tạp chí Tin học và Điều khiển học, T. 19, S. 2 (2003), 147–154

XÂY DỰNG PHƯƠNG PHÁP MÃ HÓA ĐỂ SẮP XẾP DỮ LIỆU CHỮ VIỆT

CAO ĐÌNH THI

Khoa Tin học Kinh tế, Đại học Kinh tế Quốc dân

Abstract. In this paper, we construct some coding methods applied in the arrangement of the data in vietnamese.

Tóm tắt. Trong bài báo này chúng tôi sẽ xây dựng một số phương pháp mã hóa ứng dụng trong việc sắp xếp dữ liệu chữ Việt.

1. MỞ ĐẦU

Vấn đề xây dựng các phương pháp để chuẩn hóa và sắp xếp dữ liệu chữ Việt đã được đặt ra từ lâu. Ngay từ khi máy tính điện tử mới du nhập vào nước ta những người làm công tác giảng dạy và nghiên cứu tin học Việt nam ở trong nước cũng như ở nước ngoài đã nghĩ ngay đến việc phải xây dựng một bộ mã cho tiếng Việt và các chương trình phần mềm phục vụ cho việc soạn thảo, lưu trữ và xử lý dữ liệu chữ Việt trên máy tính. Đến nay đã có một số bộ mã chữ Việt được sử dụng nhiều như ABC, VIETWARE, VNI,... Bất kỳ bộ mã nào cũng đều có những ưu điểm riêng, nhược điểm riêng nhưng nói chung chúng đã đáp ứng được yêu cầu soạn thảo, lưu trữ và in ấn chữ Việt qua máy tính. Hiện nay do việc dùng các bộ mã không đồng nhất ở trong nước cũng như ở nước ngoài (ở miền Bắc thường dùng bộ mã chuẩn TCVN 5712-93 với bộ cài đặt có tên là ABC mà người ta hay gọi tắt là bộ ABC, ở miền Nam hay sử dụng bộ VIETWARE, ở nước ngoài hay dùng bộ VNI) nên việc trao đổi thông tin giữa hai miền và ra nước ngoài gặp rất nhiều khó khăn. Thông thường, người ta phải sử dụng các chương trình chuyển đổi từ bộ mã này sang bộ mã kia rất phiền phức. Để thống nhất việc sử dụng bộ mã chữ Việt trong soạn thảo, lưu trữ, xử lý dữ liệu và văn bản chữ Việt, ngay từ năm 1993 Bộ Khoa học, Công nghệ và Môi trường đã ra quyết định số 2236/QĐ/PTCN qui định dùng bộ ABC trong tất cả các cơ quan Đảng và Nhà nước. Cũng như các bộ mã khác, bên cạnh những ưu việt như font chữ đẹp, dễ soạn thảo, bộ mã ABC vẫn còn một số hạn chế nhất định mà chúng ta cần khắc phục. Khó khăn chính đối với những người xây dựng bộ mã chữ Việt nói chung, bộ mã ABC nói riêng, là ở chỗ ngoài việc sử dụng các chữ cái của hệ La tinh ta còn phải tạo thêm phụ âm Đ, đ, các nguyên âm như Ă, ă, Â, â, Ê, ê, Ô, ô, Ơ, ơ, Ư, ư, và tổ hợp của các nguyên âm này với các dấu thanh (huyền, hỏi, ngã, sắc, nặng). Người ta thường gọi phụ âm này và các nguyên âm trên với các dấu thanh là những chữ cái thuần Việt. Chữ Việt được tạo bởi những chữ cái thuần Việt và các tổ hợp (có nghĩa) của chúng với những chữ cái của hệ La tinh. Vì trong bộ mã ASCII không còn đủ vị trí để thiết kế bộ mã chuẩn cho chữ Việt nên chúng ta đã phải lấy vị trí của một số ký tự ít dùng trong chữ Việt để thiết kế các chữ thuần Việt. Bảng mã ASCII của các chữ thuần Việt trong bộ ABC xem trong [1]. Bên cạnh đó, việc tồn tại song song cùng một lúc 2 font chữ, chữ hoa (có tên bắt đầu bằng .Vn và kết thúc bằng chữ H, ví dụ, .VnTimeH, .VnArialH,... và chữ thường tương ứng với tên không có chữ H ở cuối, ví dụ, .VnTime, .VnArial,...) trong bộ mã ABC không những gây bất tiện cho người sử dụng mà còn gây nhiều khó khăn, phiền toái khi soạn thảo, khi phải sắp xếp hoặc in ấn.

Mặc dù còn một số khiếm khuyết nhưng từ khi được quy định sử dụng chính thức trong các cơ quan Đảng, Chính phủ và Nhà nước Việt nam, bộ ABC đã đóng vai trò rất quan trọng trong việc trao đổi thông tin không những giữa các cơ quan quan trọng này mà nó còn được sử dụng rất rộng rãi trong giới Tin học nước ta. Có rất nhiều văn bản, cơ sở dữ liệu đã được soạn thảo, lưu trữ, xử lý bằng các font chữ của bộ mã này. Từ đó xuất hiện nhu cầu rất lớn về việc chuẩn hóa và sắp xếp dữ liệu chữ Việt được soạn thảo bằng các font chữ của bộ ABC. Muốn đáp ứng được nhu cầu này trước hết chúng ta phải xây dựng các phương pháp, sau đó lập các chương trình phục vụ cho mục đích trên.

Trong bài này chúng tôi sẽ xây dựng một số phương pháp mã hóa, một số thuật toán sử dụng để lập các chương trình sắp xếp dữ liệu chữ Việt trong FoxPro for Windows và trong Microsoft Excel. Ở đây khi nói đến văn bản hoặc dữ liệu chữ Việt chúng ta sẽ hiểu các văn bản hoặc dữ liệu đó được soạn thảo bằng các font chữ của bộ ABC. Vấn đề chuẩn hóa văn bản soạn thảo trong Microsoft Word đã được trình bày ở bài báo [2], chuẩn hóa dữ liệu soạn thảo trong FoxPro ở bài báo [1], trong Excel ở [5]. Trong bài này chúng tôi chỉ xét vấn đề sắp xếp dữ liệu trong FoxPro và trong Excel. Sở dĩ chúng ta xét các dữ liệu trong hai chương trình phần mềm này vì hiện nay có khá nhiều dữ liệu chữ Việt đã được soạn thảo và lưu trữ trong FoxPro và trong Excel. Như trong [3] đã chỉ rõ, những phần mềm nghiệp vụ chính của ngành ngân hàng Việt nam đều được viết bằng FoxPro và hơn 60% dữ liệu của ngành ngân hàng nước ta được soạn thảo và lưu trữ trong FoxPro. Các nguồn dữ liệu này đã được cài đặt trên hơn 500 máy chủ. Còn khối lượng dữ liệu chữ Việt được soạn thảo trên Excel cũng rất lớn. Như chúng ta đã biết, khi làm việc với một tệp dữ liệu bất kỳ, chúng ta phải làm việc với các trường (fields) (hay còn gọi là các cột) và các bản ghi (records) (các hàng) của tệp đó. Trong bài này chúng ta sẽ hiểu dữ liệu chữ Việt là dữ liệu mà giá trị của các trường, các bản ghi được soạn thảo bằng các font chữ của bộ ABC. Trên thực tế chúng ta rất hay gặp các trường hợp phải xử lý thông tin trong các tệp dữ liệu có trường họ tên (hoten) lưu giữ họ, đệm, tên của các đối tượng phải quản lý như khách hàng, học sinh, sinh viên,... Việc xử lý thường là chuẩn hóa, sắp xếp, tìm kiếm theo một tiêu chuẩn nào đó. Ví dụ, hàng năm các trường đại học, cao đẳng phải lập danh sách các thí sinh thi vào trường mình. Để tiện cho việc đánh số báo danh và để cho thí sinh dễ dàng tìm xem mình thi ở phòng thi nào chúng ta phải sắp xếp danh sách thí sinh theo tên, theo thứ tự chữ cái của chữ Việt. Việc quản lý các khách hàng, các nhân viên của một doanh nghiệp, một Công ty cũng có những vấn đề tương tự cần xử lý. Hay một ví dụ khác, khi quản lý một kho hàng hoặc một kho vật tư, nhiều khi chúng ta phải sắp xếp, tìm kiếm theo tên hàng hoặc tên vật tư. Nói chung, vấn đề sắp xếp, tìm kiếm trong một trường dữ liệu theo một tiêu chuẩn nào đó là vấn đề chúng ta thường gặp hàng ngày trong cuộc sống, trong xử lý dữ liệu. Nếu trường dữ liệu đó được soạn thảo bằng tiếng Anh, tiếng Pháp hay một ngôn ngữ nào đó chỉ sử dụng các ký tự Latinh thì việc sắp xếp, tìm kiếm không có gì khó khăn vì trong các chương trình phần mềm đã có sẵn các công cụ để làm việc này. Nhưng nếu trường dữ liệu được soạn thảo bằng chữ Việt nói chung và chữ Việt theo các font chữ của bộ ABC nói riêng thì chúng ta không thể sử dụng trực tiếp các công cụ này được. Do đó xuất hiện nhu cầu rất cấp thiết phải xây dựng các chương trình phần mềm để chuẩn hóa và sắp xếp các trường dữ liệu soạn thảo bằng chữ Việt. Để cho việc trình bày được cụ thể, ở đây chúng ta sẽ xét trường dữ liệu là trường họ tên lưu giữ họ, đệm, tên của người Việt nam.

2. SẮP XẾP DỮ LIỆU CHỮ VIỆT TRONG FOXPRO

2.1. Thứ tự trong chữ cái tiếng Việt

Muốn sắp xếp một trường dữ liệu kiểu ký tự trước hết ta phải xác định được thứ tự của

các ký tự tạo lên giá trị của trường đó. Đối với các dữ liệu được soạn thảo bằng các ký tự của hệ Latinh thì thứ tự đó được xác định bằng thứ tự của các chữ cái của hệ này. Nhưng như trên đã trình bày, chữ Việt ngoài việc sử dụng các chữ cái của hệ Latinh chúng ta còn có các chữ cái thuần Việt và thứ tự sắp xếp của chúng lại khác so với hệ Latinh (còn thứ tự các chữ cái của hệ Latinh mà chúng ta sử dụng vẫn theo thứ tự của hệ đó). Dựa vào chỉ dẫn trong [4] chúng tôi xác định thứ tự của các chữ cái trong tiếng Việt như sau:

A,a,à,ả,ã,á,ạ,Ă,ă,ằ,ẳ,ẵ,ắ,ặ,Â,â,ầ,ẩ,ẫ,ấ,ậ,B,b,B,c,D,d,Đ,đ,E,e,è,ẻ,ẽ,é,ẹ,Ê,ê,ề,ể,ễ,ế,ệ,F,f,G,g,
H,h,I,i,ì,ỉ,ĩ,í,ị,J,j,K,k,L,l,M,m,N,n,O,o,ò,ỏ,õ,ó,ọ,Ô,ô,ồ,ổ,ỗ,ố,ộ,Ơ,ơ, ờ,ở,ỡ,ớ,ợ,P,p,Q,q,R,r,S,
s,T,t,U,u,ù,ủ,ũ,ú,ụ,Ư,ư,ừ,ử,ữ,ứ,ự,V,v,W,w,X,x,Y,y,ỳ,ỷ,ỹ,ý,ỵ,Z,z. (1)

Ta tạm gọi thứ tự này là thứ tự gốc. Trong (1) ta có 126 ký tự. Qua thực tiễn sử dụng chúng tôi thấy sắp xếp theo thứ tự này tạo điều kiện rất thuận lợi cho việc tìm kiếm. Cần nhấn mạnh rằng các phương pháp và thuật toán sắp xếp trình bày trong bài này chỉ phụ thuộc vào thứ tự gốc. Nếu muốn sắp xếp theo một trật tự khác ta chỉ cần thay đổi thứ tự của các chữ cái ở đây mà thôi.

Trên thực tế vấn đề xác định đâu là họ, đâu là đệm, đâu là tên của người Việt nam hiện nay vẫn còn là vấn đề mở. Ví dụ, trong "Nguyễn Trần Thanh Hương" có người cho rằng "Nguyễn" là họ, "Hương là tên", "Trần Thanh" là đệm; nhưng có người lại cho rằng "Nguyễn Trần" là họ kép, "Thanh Hương" là tên kép; nhưng trong "Nguyễn Thị Mai Hương" thì người ta chấp nhận "Nguyễn" là họ, "Mai Hương" là tên kép còn "Thị" là đệm. Hiện nay xu hướng sử dụng họ kép và tên kép rất phổ biến ở nước ta. Người ta thường lấy họ của bố đặt trước, họ của mẹ đặt sau tạo thành họ kép. Như đã thành quy luật, muốn tìm kiếm trong một danh sách họ tên người Việt nam người ta thường tìm theo tên chứ không tìm theo họ như người nước ngoài. Mà tên người Việt nam lại viết sau cùng. Hơn nữa, do các công cụ tìm kiếm trong các chương trình phần mềm hiện có lại tìm từ trái qua phải trong một xâu ký tự nên ta không thể sử dụng một cách trực tiếp các công cụ này để tìm kiếm họ tên người Việt được.

Hiện nay vẫn còn tồn tại những tệp dữ liệu mà trong đó người ta lại tách trường họ tên thành hai trường riêng biệt là trường họ đệm và trường tên (người ta coi chữ cuối cùng là tên, còn lại là họ đệm). Danh sách thí sinh thi vào các trường đại học, cao đẳng là một ví dụ. Làm như vậy không những rất bất tiện khi nhập dữ liệu (làm giảm tốc độ nhập) mà khi in ấn lại không đẹp, không đáp ứng đòi hỏi của chuẩn quốc gia. Hơn nữa, trong trường hợp này nếu sắp theo tên sau đó tìm kiếm thì chỉ một việc tìm trong những người có tên là "Hương" thôi cũng rất vất vả vì có rất nhiều người tên là "Hương". Muốn tìm tiếp theo ta lại phải dò theo trường họ đệm, mà một số họ như họ "Nguyễn" ở Việt nam thì lại vô cùng lớn. Như vậy việc tìm kiếm trong trường hợp này là cực kỳ khó khăn. Trong bài này chúng ta sẽ xét trường hợp giá trị của trường họ tên là một xâu ký tự biểu diễn đầy đủ họ tên của người Việt nam, ví dụ, "Nguyễn Trần Thanh Hương".

2.2. Phương pháp mã hóa

Muốn sắp xếp được đúng trước hết chúng ta phải đưa dữ liệu của trường họ tên về dạng chuẩn quốc gia [4], tức là trong xâu ký tự biểu diễn họ tên người Việt mỗi chữ phải cách nhau một ký tự trống (1 dấu cách), các ký tự đầu tiên trong các chữ phải viết hoa. Chương trình chuẩn hóa dữ liệu trong FoxPro được trình bày trong [1]. Sau khi đã chuẩn hóa , để tạo điều kiện thuận lợi cho việc tìm kiếm họ tên người Việt nam chúng ta phải sắp xếp theo chữ cuối cùng trong xâu ký tự biểu diễn họ tên. Ở đây cần nhấn mạnh là chữ cuối cùng chứ không phải ký tự cuối cùng. Ví dụ, chữ "Hương" chứ không phải ký tự "g". Nếu chữ cuối cùng trùng nhau thì sắp theo chữ trước đó. Cứ như vậy sắp đến hết xâu thì thôi. Để thực

hiện được điều đó ta phải đảo ngược xâu ký tự, ví dụ, "Nguyễn Trần Thanh Hương" thành "Hương Thanh Trần Nguyễn". Sau đó sử dụng phương pháp mã hóa và các công cụ sắp xếp trong FoxPro để sắp theo trường đã mã hóa .

Điều thuận lợi trong FoxPro là chúng ta sử dụng được mã của bộ ASCII. Trong bộ ASCII thứ tự sắp xếp của các ký tự được xác định theo mã của nó. Chúng ta sẽ tận dụng khả năng này để mã hóa các chữ cái trong tiếng Việt theo thứ tự (1).

Nội dung chính của phương pháp mã hóa chúng tôi sử dụng ở đây như sau:

- Tương ứng với 126 chữ cái trong (1) chúng ta sẽ tạo ra một dẫy gồm 126 ký tự trong bộ ASCII có mã tăng dần. Không làm mất tính tổng quát, ta có thể lấy các ký tự có mã từ 1 đến 126.

- Thay thế từng ký tự trong xâu ký tự biểu diễn họ tên người Việt nam (đã đảo ngược) bằng ký tự tương ứng trong dẫy vừa tạo ra. Ta sẽ sử dụng hàm Chrtran() trong FoxPro để làm việc này.

Cuối cùng sử dụng công cụ sắp xếp trong FoxPro để sắp xếp trường ký tự mới tạo ra.

Hàm Chrtran() có cú pháp như sau:

Chrtran(< biểu thức ký tự C1 >,< biểu thức ký tự C2 >,<biểu thức ký tự C3>).

Hàm này cho kết quả là một biểu thức ký tự nhận được từ C1 khi thay thế các ký tự có trong C2 bằng ký tự tương ứng trong C3. Ví dụ, Chrtran("abcdefghac", "ace", "xyz") cho kết quả là "xbydzfghxy", (ở đây a thay bằng x, c bằng y, e bằng z).

Từ đó ta có thuật toán sắp xếp trường họ tên người Việt nam theo thứ tự từ dưới lên.

2.3. Thuật toán

• Mở tệp dữ liệu có trường họ tên cần sắp xếp;

• Thêm 2 trường trung gian, ví dụ, htnguoc và mhtnguoc lưu giá trị của trường họ tên đã đảo ngược và mã của trường này khi đã thay thế bằng các ký tự của bộ ASCII.

2.3.1. Đảo ngược xâu ký tự biểu diễn họ tên

1) Khai báo một mảng M, mảng một chiều có 10 phần tử (số chữ tối đa trong họ tên người Việt nam, ta có thể thêm hoặc bớt số phần tử của mảng), bằng lệnh Dimension M(10);

2) Đối với mỗi bản ghi, tính toán số ký tự trống bên trong xâu ký tự biểu diễn họ tên. Hàm occurs(" ", allt(hoten)) giúp ta làm việc này, sktt=occurs(" ", allt(hoten)). Dựa vào số ký tự trống ta có thể xác định được số chữ có trong trường họ tên của bản ghi hiện thời. Ví dụ, trong "Vũ Bão" thì sktt=1, số chữ bằng 2; trong "Nguyễn Trần Thanh Hương" thì sktt=3. Số chữ bằng 4. Tóm lại, số chữ=sktt+1.

3) Gán từng chữ vào từng phần tử của mảng M theo thứ tự tăng dần của các phần tử của mảng. Trong ví dụ trên M(1)="Nguyễn", M(2)="Trần", M(3)="Thanh", M(4)= "Hương".

4) Tạo xâu mới bằng cách ghép từ phần tử cuối cùng của mảng đến phần tử đầu tiên.

```
For i = sktt+1 to 1 step -1
    htng = htng + allt(M(i))
Endfor
```

5) Thay thế tất cả các giá trị của xâu mới này vào trường htnguoc.

2.3.2. Mã hóa theo các ký tự của bộ ASCII

1) Tạo xâu ký tự CV bao gồm 126 ký tự trong (1) theo thứ tự gốc;

2) Tạo xâu ASC bao gồm 126 ký tự của bộ ASCII

```
For i=1 to 126
```

ASC = ASC + chr(i)

Endfor

(Hàm chr(i) cho biết tên của ký tự có mã là i);

3) Thay thế tất cả các giá trị của trường htnguoc bằng các ký tự tương ứng của xâu ASC vào trường mhtnguoc

REPLACE all mhtnguoc WITH chr(htnguoc,CV,ASC);

4) Sắp xếp theo trường mhtnguoc

SORT TO kq ON mhtnguoc

Tệp kq lưu kết quả đã được sắp xếp.

Dựa theo thuật toán này chúng tôi đã lập chương trình chạy với nhiều loại dữ liệu khác nhau. Kết quả cho thấy trường họ tên được sắp xếp từ chữ cuối cùng, các chữ cái theo thứ tự như trong (1). Nếu chữ cuối trùng nhau thì sắp theo chữ trước đó. Cứ thế sắp xếp cho đến hết. Việc sắp xếp theo cách này rất thuận tiện cho việc tìm kiếm theo tên của người Việt nam.

3. SẮP XẾP DỮ LIỆU CHỮ VIỆT TRONG MS EXCEL

Vì trong MS Excel một số ký tự của bộ ASCII không còn phù hợp nên chúng ta không thể sử dụng trực tiếp bộ mã này để mã hóa các ký tự trong (1). Do đó thuật toán và chương trình có khác một chút. Tư tưởng của phương pháp vẫn là bên cạnh dãy các chữ cái của tiếng Việt được sắp theo thứ tự gốc (1) ta phải xây dựng một dãy các ký tự tương ứng mà sự sắp xếp của chúng tuân theo thứ tự tăng dần. Sau đó thực hiện việc mã hóa các chữ cái trong (1) theo dãy mới rồi sử dụng các công cụ sắp xếp của Excel sắp xếp cột đã mã hóa . Trong Excel ta sẽ coi mỗi sheet là một bảng dữ liệu mà trong đó mỗi cột là một trường, các tên trường được ghi ở hàng đầu tiên của bảng, mỗi hàng là một bản ghi. Từ đó muốn sắp xếp theo một trường nào đó ta chỉ cần chỉ rõ tên của cột chứa trường cần sắp xếp và thực hiện khai báo những thông tin cần thiết. Cũng như trong FoxPro, trước khi sắp xếp chúng ta phải đưa giá trị của cột họ tên về dạng chuẩn quốc gia. Chương trình chuẩn hóa dữ liệu trong Excel được trình bày trong [5]. Việc sắp xếp ở đây chúng ta cũng sẽ sắp xếp từ chữ cuối cùng như trong FoxPro.

Thuật toán

- Mở tệp có bảng dữ liệu với trường họ tên cần sắp xếp;
- Xác định vùng dữ liệu cần sắp xếp và tên cột chứa trường họ tên;
- Đối với mỗi bản ghi xác định số ký tự trống bên trong xâu ký tự biểu diễn họ tên người Việt nam, từ đó tính được số chữ thực tế của bản ghi đó;

a) Đảo ngược xâu ký tự biểu diễn họ tên

1) Tạo một mảng, ví dụ C(10) để lưu các chữ trong xâu ký tự biểu diễn họ tên. Nếu số ký tự trống trong xâu là n1 thì số chữ sẽ là n1+1. Giả sử xâu ký tự là "Vũ Thanh Bình" thì số ký tự trống là 2, số chữ là 3 và C(1) = "Vũ", C(2) = "Thanh", C(3) = "Bình".

2) Xâu ký tự mới Chng được tạo bằng cách ghép các phần tử của mảng C theo chiều từ dưới lên.

For i = n1+1 to 1 step -1

Chng = Chng + Trim(C(i))

Next i

Giả sử xâu ký tự ở cột họ tên là "Vũ Thanh Bình" thì xâu ký tự Chng là "BìnhThanhVũ".

3) Chọn cột để lưu các xâu ký tự Chng;

b) Mã hóa

Trong phần này ta sẽ tạo một bộ mã mới sao cho mỗi chữ cái trong (1) tương ứng với một xâu gồm 2 ký tự được sắp theo thứ tự tăng dần.

1) Tạo xâu ký tự gốc bao gồm 126 ký tự theo thứ tự như (1);

2) Tạo xâu ký tự mới gồm 252 ký tự: a0a1...a9b0b1...b9c0...c9.....m5;

Từ 1) và 2) ta có Bảng 1 biểu diễn sự tương ứng giữa các chữ cái tiếng Việt theo thứ tự (1) và mã của chúng.

3) Mã hóa từng hàng của cột lưu giữ các chuỗi Chng bằng cách tìm kiếm, và thay thế mỗi ký tự trong xâu ký tự gốc bằng 2 ký tự tương ứng. Ví dụ, "BìnhThanhVũ" theo bảng 1 sẽ mã hóa thành "c1f1g5e8j5e8a1g5e8l1k1". Ghi giá trị mã hóa này vào cột bên cạnh của vùng dữ liệu đã chọn.

4) Sử dụng công cụ sắp xếp của Excel để sắp xếp cột đã mã hóa .

Bảng 1. Tương ứng giữa chữ cái tiếng Việt với mã

A	a0	Â	b4	đ	c8	ệ	e2	J	f6	õ	h0	ỡ	i4	u	j8	v	l2
a	a1	â	b5	E	c9	F	e3	j	f7	ó	h1	ớ	i5	ù	j9	W	l3
à	a2	ầ	b6	e	d0	f	e4	K	f8	ọ	h2	ợ	i6	ủ	k0	w	l4
ả	a3	ẩ	b7	è	d1	G	e5	k	f9	Ô	h3	P	i7	ũ	k1	X	l5
ã	a4	ẫ	b8	ẻ	d2	g	e6	L	g0	ô	h4	p	i8	ú	k2	x	l6
á	a5	ấ	b9	ẽ	d3	H	e7	l	g1	ồ	h5	Q	i9	ụ	k3	Y	l7
ạ	a6	ậ	c0	é	d4	h	e8	M	g2	ổ	h6	q	j0	Ư	k4	y	l8
Ă	a7	B	c1	ẹ	d5	I	e9	m	g3	ỗ	h7	R	j1	ư	k5	ỳ	l9
ă	a8	b	c2	Ê	d6	i	f0	N	g4	ố	h8	r	j2	ừ	k6	ỷ	m0
ằ	a9	C	c3	ê	d7	ì	f1	n	g5	ộ	h9	S	j3	ử	k7	ỹ	m1
ẳ	b0	c	c4	ề	d8	ỉ	f2	O	g6	Ơ	i0	s	j4	ữ	k8	ý	m2
ẵ	b1	D	c5	ể	d9	ĩ	f3	o	g7	ơ	i1	T	j5	ứ	k9	y	m3
ắ	b2	d	c6	ễ	e0	í	f4	ò	g8	ờ	i2	t	j6	ự	l0	Z	m4
ặ	b3	Đ	c7	ế	e1	ị	f5	ỏ	g9	ở	i3	U	j7	V	l1	z	m5

Dựa vào thuật toán này chúng tôi đã lập chương trình bằng ngôn ngữ VBA (Visual Basic for Application) rất tiện lợi cho việc tạo một Macro để sử dụng cho các bảng dữ liệu khác nhau trong một tệp dữ liệu của Excel. Nếu đã tạo được Macro cho một bảng dữ liệu thì ta cũng dễ dàng ghi vào thư viện chương trình mẫu trong Excel để sử dụng chung cho bất kỳ một tệp dữ liệu nào khác có cột họ tên được soạn thảo bằng các font chữ của bộ ABC sau khi đã chuẩn hóa theo tiêu chuẩn Việt nam.

4. XÂY DỰNG MACRO CHO CÁC TỆP DỮ LIỆU TRONG EXCEL

Dựa theo thuật toán ở Mục 3 ta có thể xây dựng một Macro dùng chung cho các bảng dữ liệu trong Excel. Để làm được điều đó ta có thể thực hiện các bước như sau:

1) Mở một bảng dữ liệu rỗng (chưa có dữ liệu);

2) Tạo một Macro rỗng và gán cho nó một tên nào đó, ví dụ SxCV. Cách tạo một Macro rỗng có thể làm tương tự như trong [2];

3) Vào Macro SxCV (*Tools, Macro,* chọn Macro SxCV, *Edit*) để soạn thảo chương trình bằng VBA. Trong chương trình có các chữ cái tiếng Việt (xâu ký tự từ (1)) nên ta phải chọn font chữ của bộ ABC để soạn thảo chương trình. Để chọn được font chữ này, trong cửa sổ Macro có tên là Visual Basic chọn *Tools, Options, Editor Format;* ở mục Font chọn một font chữ Việt nào đó, ví dụ .VnTime, chọn cỡ chữ ở mục Size. Ta cũng có thể soạn thảo chương trình trong MS Word sau đó copy vào Macro này. Khi copy chương trình từ Word vào Macro cần kiểm tra xem có đủ 126 chữ cái từ (1) không, (thường hay thiếu chữ "ả" và chữ "ơ") nếu không đủ thì phải bổ sung cho đủ, đúng vị trí. Nhấn Ctrl S để ghi lại với một tên nào đó, ví dụ SxChuViet.xls chẳng hạn;

4) Nhấn Alt Q (hoặc vào *Menu File, Close and Return to Microsoft Excel*) để trở về bảng dữ liệu trong Excel;

5) Ra khỏi Excel;

6) Mở lại Excel; mở tệp SxChuViet, nhấn vào nút *Enable Macros;*

7) Gán cho Macro này một tổ hợp phím (Sortcut Key) bằng cách vào *Tools, Macro,* chọn Macro SxCV, *Options.* Gõ từ bàn phím một chữ cái nào đó, ví dụ chữ "T";

8) Trở về bảng dữ liệu SxChuViet; vào *File, Save As;* ghi tệp này vào thư mục *Microsoft Office\Office\Library* với kiểu là *Microsoft Add-In;* (Thông thường thư mục này ở C: Program Files).

9) Ra khỏi Excel;

10) Vào lại Excel, chọn *Tools, Add-In,* đánh dấu vào ô SxChuViet, OK.

Sau khi đã tạo và ghi Macro này vào thư viện chương trình ta có thể sử dụng nó để sắp xếp một bảng dữ liệu bất kỳ có cột họ tên người Việt được soạn thảo bằng các font chữ của bộ ABC bằng cách chỉ cần mở bảng dữ liệu đó và nhấn Ctrl T rồi làm theo chỉ dẫn là ta có thể sắp xếp cột này (các cột khác cũng chuyển theo) từ chữ cuối cùng theo thứ tự (1).

5. KẾT LUẬN

Trên đây chúng tôi đã trình bày các phương pháp mã hóa và xây dựng các thuật toán để sắp xếp dữ liệu chữ Việt được soạn thảo bằng các font chữ của bộ ABC. Như ta đã thấy, thứ tự sắp xếp chỉ phụ thuộc vào thứ tự gốc mà ở đây được biểu diễn bằng (1). Chỉ cần thay thứ tự của các ký tự trong (1) ta sẽ được một cách sắp xếp mới. Điều cơ bản là chúng ta phải chọn một thứ tự nào đó để sau khi sắp xếp ta có thể tìm kiếm được dễ dàng. Đó mới là mục đích chính của việc sắp xếp. Nếu trường (cột) dữ liệu là tên hàng, tên vật tư,... thì trong thuật toán cũng như trong chương trình chúng ta không phải đảo xâu ký tự trong trường (cột) đó mà thực hiện mã hóa rồi sắp xếp luôn. Cần phải nhấn mạnh rằng các kỹ thuật, các thuật toán ở đây không những chỉ dùng cho các font chữ của bộ ABC mà còn có thể cải tiến để sử dụng được cho các bộ mã khác. Sắp tới có thể chúng ta sẽ dùng bộ UNICODE để soạn thảo chữ Việt. Như ta đã thấy, thứ tự của các chữ cái sử dụng trong bộ này không trùng với thứ tự của chữ cái tiếng Việt. Do đó nếu sử dụng bộ UNICODE hay một bộ mã nào khác chúng ta sẽ phải xây dựng các chương trình phần mềm phục vụ cho việc sắp xếp dữ liệu chữ Việt soạn thảo bằng các font chữ của các bộ mã này. Những kỹ thuật trình bày trong bài này chắc chắn sẽ giúp ích cho chúng ta xây dựng các chương trình phần mềm đó. Lưu ý rằng xâu ký tự ASC được tạo bởi Mục 2.3.2.2) có thể bắt đầu từ một ký tự bất kỳ chứ không nhất thiết phải bắt đầu từ ký tự có mã ASCII là 1 nhưng bắt buộc các ký tự trong xâu này phải có mã tăng dần và số ký tự phải đủ 126. Cũng tương tự như vậy xâu ký tự tạo bởi Mục 3.b.2) có thể bắt đầu từ ký tự khác chứ không nhất thiết bắt đầu từ chữ a nhưng bắt buộc phải có đủ 252 ký tự và các cặp 2 ký tự một này phải có thứ tự tăng dần. Dựa theo các thuật toán trên chúng tôi đã xây dựng các chương trình trọn vẹn và đã

thử nghiệm trên nhiều số liệu thực tế. Kết quả chứng tỏ tính hiệu quả rất cao của các thuật toán này. Độc giả muốn sao chép các chương trình này có thể liên hệ với tác giả theo địa chỉ: cdthi@cfvghn.org.vn

TÀI LIỆU THAM KHẢO

[1] Cao Đình Thi, Chuẩn hóa dữ liệu & sắp xếp chữ Việt trong font chữ in hoa, *Tạp chí Tin học Ngân hàng,* (2) (1999) 27–32.

[2] Cao Đình Thi, Chuẩn hóa văn bản chữ Việt soạn thảo trong Word, *Tạp chí Tin học và Điều khiển học,* **17** (2) (2001) 82–86.

[3] Tạ Quang Tiến, Y2K - những vấn đề cần quan tâm, *Tạp chí Tin học Ngân hàng,* (2) (1999) 3–5.

[4] *Từ điển bách khoa Việt nam 1.* Trung tâm biên soạn từ điển bách khoa Việt nam, Hà nội 1995, trang 8.

[5] Vũ Văn Thái, Cao Đình Thi, Chuẩn hóa dữ liệu tiếng Việt trong Excel, *Tạp chí Tin học Ngân hàng,* (5) (1999) 22–25.

Nhận bài ngày 12 - 11 - 2002