

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, SHA 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

Chương 1 Tổng quan	15
1.1 Mật mã học	15
1.2 Hệ thống mã hóa (cryptosystem)	16
1.3 Hệ thống mã hóa quy ước (mã hóa đối xứng)	18
1.4 Hệ thống mã hóa khóa công cộng (mã hóa bất đối xứng)	19
1.5 Kết hợp mã hóa quy ước và mã hóa khóa công cộng	19
Chương 2 Một số phương pháp mã hóa quy ước	20
2.1 Hệ thống mã hóa quy ước	20
2.2 Phương pháp mã hóa dịch chuyển	21
2.3 Phương pháp mã hóa thay thế	22
2.4 Phương pháp Affine	23
2.5 Phương pháp Vigenere	28
2.6 Phương pháp Hill	29
2.7 Phương pháp mã hóa hoán vị	30
2.8 Phương pháp mã hóa bằng phép nhân	31
2.8.1 Phương pháp mã hóa bằng phép nhân	31
2.8.2 Xử lý số học	32
2.9 Phương pháp DES (Data Encryption Standard)	33
2.9.1 Phương pháp DES	33
2.9.2 Nhận xét	36
2.10 Phương pháp chuẩn mã hóa nâng cao AES	37
Chương 3 Phương pháp mã hóa Rijndael	39
3.1 Giới thiệu	39
3.2 Tham số, ký hiệu, thuật ngữ và hàm	40
3.3 Một số khái niệm toán học	42

3.3.1	Phép cộng	43
3.3.2	Phép nhân	43
3.3.3	Đa thức với hệ số trên $GF(2^8)$	46
3.4	Phương pháp Rijndael	49
3.4.1	Quy trình mã hóa	50
3.4.2	Kiến trúc của thuật toán Rijndael	52
3.4.3	Phép biến đổi SubBytes	53
3.4.4	Phép biến đổi ShiftRows	55
3.4.5	Phép biến đổi MixColumns	56
3.4.6	Thao tác AddRoundKey	58
3.5	Phát sinh khóa của mỗi chu kỳ	59
3.5.1	Xây dựng bảng khóa mở rộng	59
3.5.2	Xác định khóa của chu kỳ	61
3.6	Quy trình giải mã	62
3.6.1	Phép biến đổi InvShiftRows	63
3.6.2	Phép biến đổi InvSubBytes	64
3.6.3	Phép biến đổi InvMixColumns	66
3.6.4	Quy trình giải mã tương đương	67
3.7	Các vấn đề cài đặt thuật toán	69
3.7.1	Nhận xét	72
3.8	Kết quả thử nghiệm	73
3.9	Kết luận	74
3.9.1	Khả năng an toàn	74
3.9.2	Đánh giá	75

Chương 4 Phương pháp Rijndael mở rộng 77

4.1	Nhu cầu mở rộng phương pháp mã hóa Rijndael	77
4.2	Phiên bản mở rộng 256/384/512-bit	78
4.2.1	Quy trình mã hóa	79
4.2.2	Phát sinh khóa của mỗi chu kỳ	86
4.2.3	Quy trình giải mã	88
4.2.4	Quy trình giải mã tương đương	93
4.3	Phiên bản mở rộng 512/768/1024-bit	94
4.4	Phân tích mật mã vi phân và phân tích mật mã tuyến tính	95
4.4.1	Phân tích mật mã vi phân	95
4.4.2	Phân tích mật mã tuyến tính	96

4.4.3	Branch Number	98
4.4.4	Sự lan truyền mẫu	99
4.4.5	Trọng số vết vi phân và vết tuyến tính	107
4.5	Khảo sát tính an toàn đối với các phương pháp tấn công khác	108
4.5.1	Tính đối xứng và các khóa yếu của DES	108
4.5.2	Phương pháp tấn công Square	109
4.5.3	Phương pháp nội suy	109
4.5.4	Các khóa yếu trong IDEA	110
4.5.5	Phương pháp tấn công khóa liên quan	110
4.6	Kết quả thử nghiệm	111
4.7	Kết luận	113

Chương 5 Các thuật toán ứng cử viên AES **115**

5.1	Phương pháp mã hóa MARS	115
5.1.1	Quy trình mã hóa	116
5.1.2	S-box	117
5.1.3	Khởi tạo và phân bố khóa	118
5.1.4	Quy trình mã hóa	123
5.1.5	Quy trình giải mã	135
5.2	Phương pháp mã hóa RC6	137
5.2.1	Khởi tạo và phân bố khóa	138
5.2.2	Quy trình mã hóa	139
5.2.3	Quy trình giải mã	143
5.3	Phương pháp mã hóa Serpent	144
5.3.1	Thuật toán SERPENT	144
5.3.2	Khởi tạo và phân bố khóa	144
5.3.3	S-box	147
5.3.4	Quy trình mã hóa	148
5.3.5	Quy trình giải mã	153
5.4	Phương pháp mã hóa TwoFish	154
5.4.1	Khởi tạo và phân bố khóa	154
5.4.2	Quy trình mã hóa	163
5.4.3	Quy trình giải mã	169
5.5	Kết luận	169

Chương 6 Một số hệ thống mã hóa khóa công cộng	172
6.1 Hệ thống mã hóa khóa công cộng	172
6.2 Phương pháp RSA	174
6.2.1 Phương pháp RSA	174
6.2.2 Một số phương pháp tấn công giải thuật RSA	175
6.2.3 Sự che dấu thông tin trong hệ thống RSA	182
6.2.4 Vấn đề số nguyên tố	183
6.2.5 Thuật toán Miller-Rabin	184
6.2.6 Xử lý số học	186
6.3 Mã hóa quy ước và mã hóa khóa công cộng	186
Chương 7 Chữ ký điện tử	191
7.1 Giới thiệu	191
7.2 Phương pháp chữ ký điện tử RSA	192
7.3 Phương pháp chữ ký điện tử ElGamal	193
7.3.1 Bài toán logarit rời rạc	193
7.3.2 Phương pháp ElGamal	194
7.4 Phương pháp Digital Signature Standard	194
Chương 8 Phương pháp ECC	197
8.1 Lý thuyết đường cong elliptic	197
8.1.1 Công thức Weierstrasse và đường cong elliptic	198
8.1.2 Đường cong elliptic trên trường R^2	199
8.1.3 Đường cong elliptic trên trường hữu hạn	204
8.1.4 Bài toán logarit rời rạc trên đường cong elliptic	212
8.1.5 Áp dụng lý thuyết đường cong elliptic vào mã hóa	213
8.2 Mã hóa dữ liệu	213
8.2.1 Thao tác mã hóa	214
8.2.2 Kết hợp ECES với thuật toán Rijndael và các thuật toán mở rộng	215
8.2.3 Thao tác giải mã	215
8.3 Trao đổi khóa theo phương pháp Diffie - Hellman sử dụng lý thuyết đường cong elliptic (ECDH)	216
8.3.1 Mô hình trao đổi khóa Diffie-Hellman	216
8.3.2 Mô hình trao đổi khóa Elliptic Curve Diffie - Hellman	217
8.4 Kết luận	218

Chương 9 Hàm băm mật mã **222**

9.1	Giới thiệu	222
9.1.1	Đặt vấn đề	222
9.1.2	Hàm băm mật mã	223
9.1.3	Cấu trúc của hàm băm	225
9.1.4	Tính an toàn của hàm băm đối với hiện tượng đụng độ	226
9.1.5	Tính một chiều	226
9.2	Hàm băm MD5	227
9.2.1	Giới thiệu MD5	227
9.2.2	Nhận xét	231
9.3	Phương pháp Secure Hash Standard (SHS)	232
9.3.1	Nhận xét	235
9.4	Hệ thống chuẩn hàm băm mật mã SHA	236
9.4.1	Ý tưởng của các thuật toán hàm băm SHA	236
9.4.2	Khung thuật toán chung của các hàm băm SHA	237
9.4.3	Nhận xét	240
9.5	Kiến trúc hàm băm Davies-Mayer và ứng dụng của thuật toán Rijndael và các phiên bản mở rộng vào hàm băm	241
9.5.1	Kiến trúc hàm băm Davies-Mayer	241
9.5.2	Hàm AES-Hash	242
9.5.3	Hàm băm Davies-Mayer và AES-Hash	244
9.6	Xây dựng các hàm băm sử dụng các thuật toán mở rộng dựa trên thuật toán Rijndael	245

Chương 10 Chứng nhận khóa công cộng **246**

10.1	Giới thiệu	246
10.2	Các loại giấy chứng nhận khóa công cộng	250
10.2.1	Chứng nhận X.509	250
10.2.2	Chứng nhận chất lượng	252
10.2.3	Chứng nhận PGP	253
10.2.4	Chứng nhận thuộc tính	253
10.3	Sự chứng nhận và kiểm tra chữ ký	254
10.4	Các thành phần của một cơ sở hạ tầng khóa công cộng	257
10.4.1	Tổ chức chứng nhận – Certificate Authority (CA)	257
10.4.2	Tổ chức đăng ký chứng nhận – Registration Authority (RA)	258

10.4.3	Kho lưu trữ chứng nhận – Certificate Repository (CR)	259
10.5	Chu trình quản lý giấy chứng nhận	259
10.5.1	Khởi tạo	259
10.5.2	Yêu cầu về giấy chứng nhận	259
10.5.3	Tạo lại chứng nhận	262
10.5.4	Hủy bỏ chứng nhận	262
10.5.5	Lưu trữ và khôi phục khóa	264
10.6	Các mô hình CA	264
10.6.1	Mô hình tập trung	264
10.6.2	Mô hình phân cấp	265
10.6.3	Mô hình “Web of Trust”	266
10.7	Ứng dụng “Hệ thống bảo vệ thư điện tử”	268
10.7.1	Đặt vấn đề	268
10.7.2	Quy trình mã hóa thư điện tử	269
10.7.3	Quy trình giải mã thư điện tử	270
10.7.4	Nhận xét – Đánh giá	271
Phụ lục A	S-box của thuật toán MARS	272
Phụ lục B	Các hoán vị sử dụng trong thuật toán Serpent	275
Phụ lục C	S-box sử dụng trong thuật toán Serpent	276
Phụ lục D	S-box của thuật toán Rijndael	277
Phụ lục E	Hằng số và giá trị khởi tạo của SHA	279
E.1	Hằng số sử dụng trong SHA	279
E.1.1	Hằng số của SHA-1	279
E.1.2	Hằng số của SHA-224 và SHA-256	279
E.1.3	Hằng số của SHA-384 và SHA-512	280
E.2	Giá trị khởi tạo trong SHA	281
	Tài liệu tham khảo	284

Danh sách hình

Hình 2.1. Mô hình hệ thống mã hóa quy ước	21
Hình 2.2. Biểu diễn dãy 64 bit x thành 2 thành phần L và R	34
Hình 2.3. Quy trình phát sinh dãy $L_i R_i$ từ dãy $L_{i-1} R_{i-1}$ và khóa K_i	35
Hình 3.1. Biểu diễn dạng ma trận của trạng thái ($Nb = 6$) và mã khóa ($Nk = 4$)	49
Hình 3.2. Một chu kỳ mã hóa của phương pháp Rijndael (với $Nb = 4$)	52
Hình 3.3. Thao tác SubBytes tác động trên từng byte của trạng thái	54
Hình 3.4. Thao tác ShiftRows tác động trên từng dòng của trạng thái	55
Hình 3.5. Thao tác MixColumns tác động lên mỗi cột của trạng thái	57
Hình 3.6. Thao tác AddRoundKey tác động lên mỗi cột của trạng thái	59
Hình 3.7. Bảng mã khóa mở rộng và cách xác định mã khóa của chu kỳ ($Nb = 6$ và $Nk = 4$)	61
Hình 3.8. Thao tác InvShiftRows tác động lên từng dòng của trạng thái hiện hành	63
Hình 4.1. Kiến trúc một chu kỳ biến đổi của thuật toán Rijndael mở rộng 256/384/512-bit với $Nb = 4$	80
Hình 4.2. Bảng mã khóa mở rộng và cách xác định mã khóa của chu kỳ (với $Nb = 6$ và $Nk = 4$)	88
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 $Nb = 6$	100
Hình 4.4. Sự lan truyền mẫu hoạt động (thuật toán mở rộng 256/384/512-bit)	102
Hình 4.5. Minh họa Định lý 4.1 với $Q = 2$ (thuật toán mở rộng 256/384/512-bit)	103

Hình 4.6. Minh họa Định lý 4.2 với $\mathcal{W}_c(a_1) = 1$ (th-toán mở rộng 256/384/512bit)	105
Hình 4.7. Minh họa Định lý 4.3 (thuật toán mở rộng 256/384/512-bit)	107
Hình 5.1. Quy trình mã hóa MARS	116
Hình 5.2. Cấu trúc giai đoạn “Trộn tới”	125
Hình 5.3. Hệ thống Feistel loại 3	127
Hình 5.4. Hàm E	128
Hình 5.5. Cấu trúc giai đoạn “Trộn lùi”	130
Hình 5.6. Cấu trúc mã hóa RC6	140
Hình 5.7. Chu kỳ thứ i của quy trình mã hóa RC6	141
Hình 5.8. Mô hình phát sinh khóa	146
Hình 5.9. Cấu trúc mã hóa	149
Hình 5.10. Chu kỳ thứ i ($i = 0, \dots, 30$) của quy trình mã hóa Serpent	150
Hình 5.11. Cấu trúc giải mã	153
Hình 5.12. Hàm h	157
Hình 5.13. Mô hình phát sinh các S-box phụ thuộc khóa	159
Hình 5.14. Mô hình phát sinh subkey K_j	160
Hình 5.15. Phép hoán vị q	162
Hình 5.16. Cấu trúc mã hóa	164
Hình 5.17. Hàm F (khóa 128 bit)	166
Hình 5.18. So sánh quy trình mã hóa (a) và giải mã (b)	169
Hình 6.1. Mô hình hệ thống mã hóa với khóa công cộng	174
Hình 6.2. Quy trình trao đổi khóa bí mật sử dụng khóa công cộng	187
Hình 6.3. Đồ thị so sánh chi phí công phá khóa bí mật và khóa công cộng	189
Hình 8.1. Một ví dụ về đường cong elliptic	199


Hình 8.2. Điểm ở vô cực	200
Hình 8.3. Phép cộng trên đường cong elliptic	201
Hình 8.4. Phép nhân đôi trên đường cong elliptic	203
Hình 8.5: So sánh mức độ bảo mật giữa ECC với RSA / DSA	220
Hình 9.1. Khung thuật toán chung cho các hàm băm SHA	238
Hình 10.1. Vấn đề chủ sở hữu khóa công cộng	247
Hình 10.2. Các thành phần của một chứng nhận khóa công cộng	248
Hình 10.3. Mô hình Certification Authority đơn giản	249
Hình 10.4. Phiên bản 3 của chuẩn chứng nhận X.509	251
Hình 10.5. Phiên bản 2 của cấu trúc chứng nhận thuộc tính	254
Hình 10.6. Quá trình ký chứng nhận	255
Hình 10.7. Quá trình kiểm tra chứng nhận	256
Hình 10.8. Mô hình PKI cơ bản	257
Hình 10.9. Mẫu yêu cầu chứng nhận theo chuẩn PKCS#10	260
Hình 10.10. Định dạng thông điệp yêu cầu chứng nhận theo RFC 2511	261
Hình 10.11. Phiên bản 2 của định dạng danh sách chứng nhận bị hủy	263
Hình 10.12. Mô hình CA tập trung	264
Hình 10.13. Mô hình CA phân cấp	266
Hình 10.14. Mô hình “Web of trust”	267
Hình 10.15. Quy trình mã hóa thư điện tử	269
Hình 10.16. Quy trình giải mã thư điện tử	270

Danh sách bảng

Bảng 3.1. Giá trị di số shift(r, Nb)	55
Bảng 3.2. Tốc độ xử lý của phương pháp Rijndael	73
Bảng 4.1. Ảnh hưởng của các phép biến đổi lên mẫu hoạt động	101
Bảng 4.2. Tốc độ xử lý phiên bản 256/384/512-bit trên máy Pentium IV 2.4GHz	111
Bảng 4.3. Tốc độ xử lý phiên bản 512/768/1024-bit trên máy Pentium IV 2.4 GHz	112
Bảng 4.4. Bảng so sánh tốc độ xử lý của phiên bản 256/384/512-bit	112
Bảng 4.5. Bảng so sánh tốc độ xử lý của phiên bản 512/768/1024-bit	112
Bảng 6.1. So sánh độ an toàn giữa khóa bí mật và khóa công cộng	188
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	211
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	218
Bảng 8.3. So sánh kích thước khóa RSA và ECC với cùng mức độ an toàn	219
Bảng 9.1. Chu kỳ biến đổi trong MD5	230
Bảng 9.2. Các tính chất của các thuật toán băm an toàn	241
Bảng D.1. Bảng thay thế S-box cho giá trị $\{xy\}$ ở dạng thập lục phân.	277
Bảng D.2. Bảng thay thế nghịch đảo cho giá trị $\{xy\}$ ở dạng thập lục phân.	278

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 giấu 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 mẫu tin nguồn cần mã hóa có thể có*
2. *Tập đích C là tập hữu hạn tất cả các mẫu 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 $e_k : P \rightarrow C$ và luật giải mã $d_k : C \rightarrow P$ 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 mẫu 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 $\{0, 1, \dots, m-1\}$, được trang bị phép cộng (ký hiệu $+$) và phép nhân (ký hiệu là \times). 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

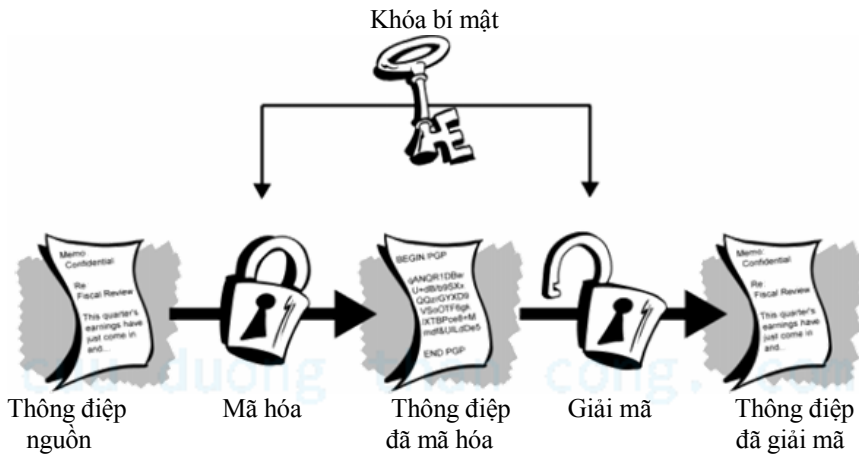
Một số phương pháp mã hóa quy ước

 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



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:

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

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

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

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

Với mỗi khóa $\pi \in K$, định nghĩa:

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

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

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

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

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

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

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

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 $ax + b \equiv y \pmod{n}$ phải có nghiệm duy nhất $x \in \mathbb{Z}_n$.

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

Định lý 2.1: Phương trình $ax + b \equiv y \pmod{n}$ 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 $e_i \in \mathbb{Z}^+$, $1 \leq i \leq m$ thì $\phi(n) = \prod_{i=1}^m (p_i^{e_i} - p_i^{e_i-1})$.

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:

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

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, \dots, t_m theo công thức truy hồi sau:

$$\begin{aligned} t_0 &= 0 \\ t_1 &= 1 \\ t_j &= (t_{j-2} - q_{j-1}t_{j-1}) \bmod r_0 \text{ với } j \geq 2 \end{aligned} \tag{2.2}$$

Định lý 2.3: Với mọi j , $0 \leq j \leq m$, ta có $r_j \equiv t_j r_1 \pmod{r_0}$, 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} .

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

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 $P = C = K = (\mathbb{Z}_n)^m$

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

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

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

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

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:

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

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 $\{1, 2, \dots, m\}$

Với mỗi khóa $\pi \in K$, định nghĩa:

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

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

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 $\{1, 2, \dots, m\}$, 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 Án 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 Án 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  $b_{l-1}, b_{l-2}, \dots, b_2, b_1, b_i \in \{0, 1\}, 0 \leq i < l$ 
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

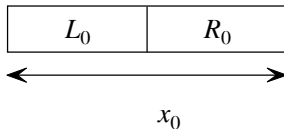
Khoảng những năm 1970, tiến sĩ Horst Feistel đã đặt nền móng đầu tiên cho chuẩn mã hóa dữ liệu DES với phương pháp mã hóa Feistel Cipher. Vào năm 1976 Cơ quan Bảo mật Quốc gia Hoa Kỳ (NSA) đã công nhận DES dựa trên phương pháp Feistel là chuẩn mã hóa dữ liệu [25]. Kích thước khóa của DES ban đầu là 128 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 $x_0 = IP(x) = L_0 R_0$, 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

- 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 $1 \leq i \leq 16$) được xác định theo quy tắc sau:

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

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

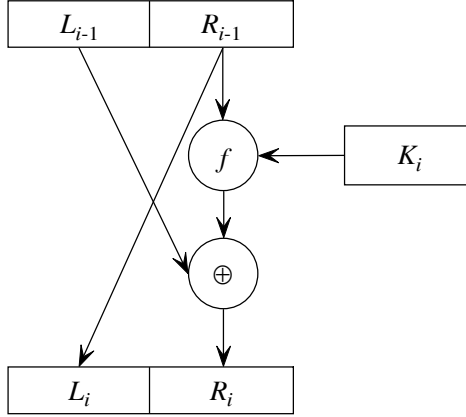
với \oplus biểu diễn phép toán XOR trên hai dãy bit, K_1, K_2, \dots, K_{16} là các dãy 48 bit phát sinh từ khóa K cho trước (Trên thực tế, mỗi khóa K_i được phát sinh bằng cách hoán vị các bit trong khóa K cho trước).

- 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 $L_i R_i$ 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: $B = B_1 B_2 B_3 B_4 B_5 B_6 B_7 B_8$.

Sử dụng tám ma trận S_1, S_2, \dots, 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 $B_j = b_1 b_2 b_3 b_4 b_5 b_6$, $S_j(B_j)$ 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 $b_1 b_6$, chỉ số cột c có biểu diễn nhị phân là $b_2 b_3 b_4 b_5$. Bằng cách này, ta xác định được các dãy 4 bit $C_j = S_j(B_j)$, $1 \leq j \leq 8$.

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 khóa 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.

cuu duong than cong. com

cuu duong than cong. com

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](https://www.nist.gov/) – 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 hiện 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 $8 \times Nw$ 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 ($8 \times Nw$ 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 $x^{Nw} + 1$.
•	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 $c_i = a_i \oplus b_i, 0 \leq i \leq 7$.

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

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

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

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

với đa thức x cho kết quả là đa thức

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

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

1. Trường hợp $b_7 = 0$

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

2. Trường hợp $b_7 = 1$

$$\begin{aligned} x \bullet b(x) &= (b_7x^8 + b_6x^7 + b_5x^6 + b_4x^5 + b_3x^4 + b_2x^3 + b_1x^2 + b_0x) \bmod m(x) \\ &= (b_7x^8 + b_6x^7 + b_5x^6 + b_4x^5 + b_3x^4 + b_2x^3 + b_1x^2 + b_0x) - m(x) \quad (3.5) \end{aligned}$$

Như vậy, phép nhân với đa thức x (hay phần tử $\{00000010\} \in \text{GF}(2^8)$) 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 $\text{GF}(2^8)$ 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:

$$\begin{aligned} \{57\} \bullet \{13\} &= \{fe\} \text{ vì} \\ \{57\} \bullet \{02\} &= \text{xtime}(\{57\}) = \{ae\} \\ \{57\} \bullet \{04\} &= \text{xtime}(\{ae\}) = \{47\} \\ \{57\} \bullet \{08\} &= \text{xtime}(\{47\}) = \{8e\} \\ \{57\} \bullet \{10\} &= \text{xtime}(\{8e\}) = \{07\}, \end{aligned}$$

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 $GF(2^8)$:

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

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:

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

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

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

với

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

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

$$d(x) = d_3x^3 + d_2x^2 + d_1x + d_0 \quad (3.10)$$

với

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

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

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

Do $x^4 + 1$ không phải là một đa thức tối giản trên $\text{GF}(2^8)$ 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 $M(x)$)

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

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

3.3.3.1 Phép nhân với x

Xét đa thức

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

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

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

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 $a_1 = \{01\}$.

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

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 (\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.

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.

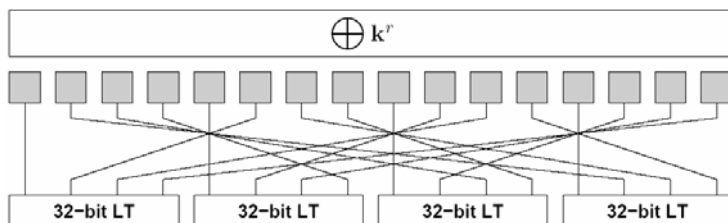
Trong thuật toán dưới đây, mảng **w[]** chứa bảng mã khóa mở rộng; mảng **in[]** và **out[]** lần lượt chứa dữ liệu vào và kết quả ra của thuật toán mã hóa.

```
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 $N_b = 4$)

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:

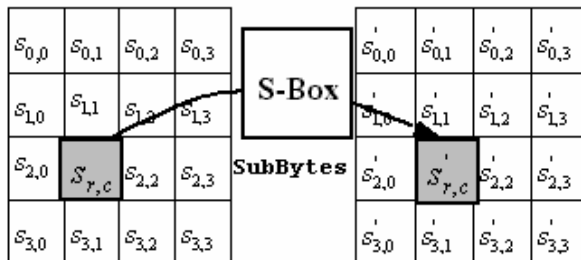
1. Xác định phân tử nghịch đảo $x^{-1} \in \text{GF}(2^8)$. Quy ước $\{00\}^{-1} = \{00\}$.
2. Áp dụng phép biến đổi affine (trên $\text{GF}(2)$) đối với x^{-1} (giả sử x^{-1} có biểu diễn nhị phân là $\{x_7x_6x_5x_4x_3x_2x_1x_0\}$):

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

hay

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

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



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.

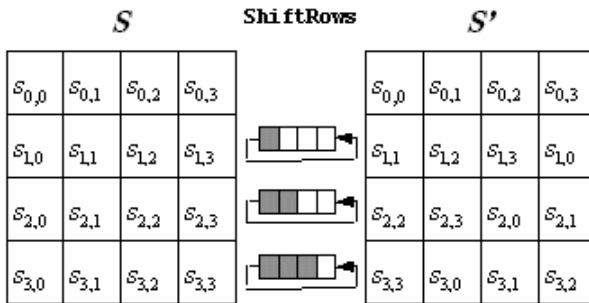
□ Ví dụ: nếu giá trị $\{xy\}$ cần thay thế là $\{53\}$ thì giá trị thay thế S-box ($\{xy\}$) được xác định bằng cách lấy giá trị tại dòng 5 cột 3 của Bảng D.1. Như vậy, $\text{S-box}(\{xy\}) = \{ed\}$.

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 $(c - \text{shift}(r, Nb)) \bmod Nb$ hay:

$$s'_{r,c} = s_{r,(c+\text{shift}(r,Nb))\bmod Nb} \quad \text{với } 0 < r < 8 \quad \text{và } 0 \leq c < Nb \quad (3.20)$$

Giá trị di số $\text{shift}(r, Nb)$ phụ thuộc vào chỉ số dòng r và kích thước Nb của khối dữ liệu.

Bảng 3.1. Giá trị di số $\text{shift}(r, Nb)$

$\text{shift}(r, Nb)$		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 $\text{GF}(2^8)$. Thực hiện phép nhân

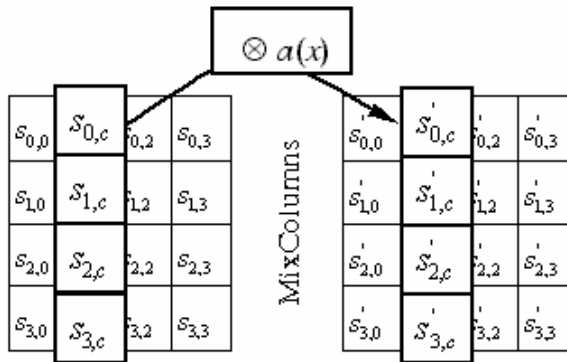
$$s'(x) = a(x) \otimes s(x) \quad (3.21)$$

với

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

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

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



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 phân 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:

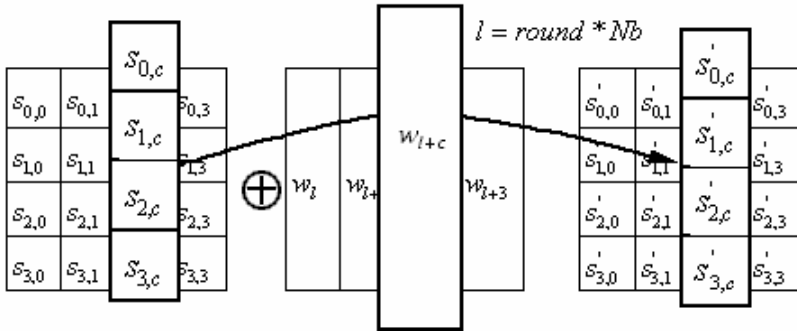
$$[s'_{0,c}, s'_{1,c}, s'_{2,c}, s'_{3,c}] = [s_{0,c}, s_{1,c}, s_{2,c}, s_{3,c}] \oplus [w_{round * Nb + c}], \quad (3.24)$$

với $0 \leq c < Nb$.

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 hiện 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 $RC[i] \in GF(2^8)$ và thỏa:

$$\begin{aligned} RC[1] &= 1 (\{01\}) \\ RC[i] &= x (\{02\}) \bullet (RC[i-1]) = x^{(i-1)} \end{aligned} \quad (3.25)$$

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 $w[Nb * i], w[Nb * i + 1], \dots, w[Nb * (i + 1) - 1]$.

w_0	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	w_{10}	w_{11}	w_{12}	w_{13}	w_{14}	w_{15}	w_{16}	w_{17}	...
Mã khóa chu kỳ 0						Mã khóa chu kỳ 1						Mã khóa chu kỳ 2						...

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
```

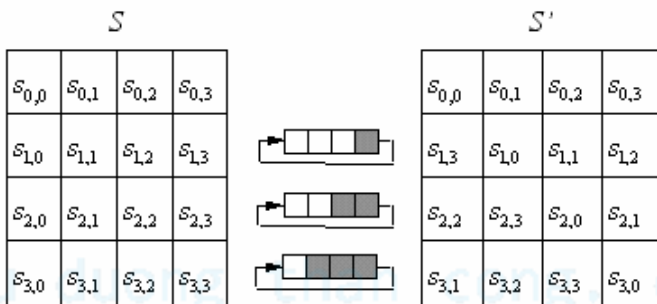
```

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.

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

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*, sử dụng bảng thay thế nghịch đảo của *S-box* trên $GF(2^8)$, ký hiệu là $S\text{-box}^{-1}$. Quá trình thay thế 1 byte y dựa vào $S\text{-box}^{-1}$ 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 $\{05\}$, $0 \leq i \leq 7$. (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 $\text{GF}(2^8)$ có biểu diễn nhị phân là $\{x_7x_6x_5x_4x_3x_2x_1x_0\}$.

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

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

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

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

Trong đoạn mã chương trình dưới đây, hàm `FFmul(x, y)` thực hiện phép nhân (trên trường $GF(2^8)$) hai phân 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:

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

Sau biến đổi ShiftRows:

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

Sau biến đổi MixColumns:

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

Sau biến đổi AddRoundKey:

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

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

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

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

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

Khai triển phép nhân ma trận, ta có:

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

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

$$\begin{aligned} T_0[a] &= \begin{bmatrix} S[a] \bullet 02 \\ S[a] \\ S[a] \\ S[a] \bullet 03 \end{bmatrix}, T_1[a] = \begin{bmatrix} S[a] \bullet 03 \\ S[a] \bullet 02 \\ S[a] \\ S[a] \end{bmatrix}, \\ T_2[a] &= \begin{bmatrix} S[a] \\ S[a] \bullet 03 \\ S[a] \bullet 02 \\ S[a] \end{bmatrix}, T_3[a] = \begin{bmatrix} S[a] \\ S[a] \\ S[a] \bullet 03 \\ S[a] \bullet 02 \end{bmatrix} \end{aligned} \quad (3.38)$$

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

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

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 kỳ đầu. Do chu kỳ cuối cùng không thực hiện phép biến đổi MixColumns nên cần xây dựng 4 bảng tra cứu riêng cho chu kỳ 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 $a \in \text{GF}(2^8)$, đặt:

$$\begin{aligned} T_0[a] &= \begin{bmatrix} S[a] \bullet 02 \\ S[a] \\ S[a] \\ S[a] \bullet 03 \end{bmatrix}, & T_1[a] &= \begin{bmatrix} S[a] \bullet 03 \\ S[a] \bullet 02 \\ S[a] \\ S[a] \end{bmatrix}, \\ T_2[a] &= \begin{bmatrix} S[a] \\ S[a] \bullet 03 \\ S[a] \bullet 02 \\ S[a] \end{bmatrix}, & T_3[a] &= \begin{bmatrix} S[a] \\ S[a] \\ S[a] \bullet 03 \\ S[a] \bullet 02 \end{bmatrix} \end{aligned} \quad (3.41)$$

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

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

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:

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

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.

cuu duong than cong. com

cuu duong than cong. com

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 thể 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 $N_w=8$ byte. Mỗi trạng thái có thể được biểu diễn dưới dạng một ma trận gồm 8 dòng và N_b cột với N_b bằng với độ dài của khối chia cho 64. Khóa chính cũng được biểu diễn dưới dạng một ma trận gồm 8 dòng và N_k cột với N_k 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 (N_w byte), mỗi phần tử tương ứng với 1 cột của ma trận.

Số lượng chu kỳ, ký hiệu là N_r , có giá trị là

$$N_r = \max \{ N_b, N_k \} + 6 \quad (4.1)$$

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.

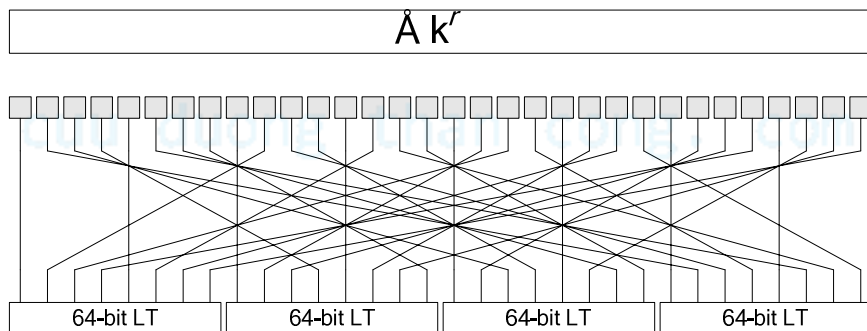
Mỗi phép biến đổi thao tác trên trạng thái hiện hành S . Kết quả S' của mỗi phép biến đổi sẽ trở thành đầu vào của phép biến đổi kế tiếp trong quy trình mã hóa.

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. $N_r - 1$ 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 phân tử nghịch đảo $x^{-1} \in \text{GF}(2^8)$. 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 $\{x_7, x_6, x_5, x_4, x_3, x_2, x_1, x_0\}$):

$$y_i = x_i \oplus x_{(i+4) \bmod 8} \oplus x_{(i+5) \bmod 8} \oplus x_{(i+6) \bmod 8} \oplus x_{(i+7) \bmod 8} \oplus c_i \quad (4.2)$$

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

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 $(c - \text{shift}(r, Nb)) \bmod Nb$ hay:

$$S'_{r,c} = S_{r, (c + \text{shift}(r, Nb)) \bmod Nb} \quad \text{với } 0 < r < 8 \text{ và } 0 \leq c < Nb \quad (4.3)$$

với

$$\text{shift}(r, Nb) = r \bmod Nb \quad (4.4)$$

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 $\text{GF}(2^8)$. Thực hiện phép nhân:

$$s'(x) = a(x) \otimes s(x) \text{ với } a(x) = \sum_{i=0}^7 a_i x^i, a_i \in \text{GF}(2^8) \quad (4.5)$$

$$\text{Đặt } M_a = \begin{bmatrix} \alpha_0 & \alpha_7 & \alpha_6 & \alpha_5 & \alpha_4 & \alpha_3 & \alpha_2 & \alpha_1 \\ \alpha_1 & \alpha_0 & \alpha_7 & \alpha_6 & \alpha_5 & \alpha_4 & \alpha_3 & \alpha_2 \\ \alpha_2 & \alpha_1 & \alpha_0 & \alpha_7 & \alpha_6 & \alpha_5 & \alpha_4 & \alpha_3 \\ \alpha_3 & \alpha_2 & \alpha_1 & \alpha_0 & \alpha_7 & \alpha_6 & \alpha_5 & \alpha_4 \\ \alpha_4 & \alpha_3 & \alpha_2 & \alpha_1 & \alpha_0 & \alpha_7 & \alpha_6 & \alpha_5 \\ \alpha_5 & \alpha_4 & \alpha_3 & \alpha_2 & \alpha_1 & \alpha_0 & \alpha_7 & \alpha_6 \\ \alpha_6 & \alpha_5 & \alpha_4 & \alpha_3 & \alpha_2 & \alpha_1 & \alpha_0 & \alpha_7 \\ \alpha_7 & \alpha_6 & \alpha_5 & \alpha_4 & \alpha_3 & \alpha_2 & \alpha_1 & \alpha_0 \end{bmatrix} \quad (4.6)$$

Ta có:

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \\ s'_{4,c} \\ s'_{5,c} \\ s'_{6,c} \\ s'_{7,c} \end{bmatrix} = M_a \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \\ s_{4,c} \\ s_{5,c} \\ s_{6,c} \\ s_{7,c} \end{bmatrix}, 0 \leq c \leq Nb \quad (4.7)$$

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(2⁸)) 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:

$$\begin{aligned} [s'_{0,c}, s'_{1,c}, s'_{2,c}, s'_{3,c}, s'_{4,c}, s'_{5,c}, s'_{6,c}, s'_{7,c}] = \\ [s_{0,c}, s_{1,c}, s_{2,c}, s_{3,c}, s_{4,c}, s_{5,c}, s_{6,c}, s_{7,c}] \oplus [w_{round * Nb + c}] \end{aligned} \quad \text{với } 0 \leq c < Nb, \quad (4.8)$$

- ❖ *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 $\text{SubWord}(W)$ thay thế (sử dụng S-box) từng byte thành phần của một từ (có độ dài 8 byte).

Hàm $\text{RotWord}(W)$ 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 $\text{Rcon}[i] = (x^{i-1}, 0, 0, 0, 0, 0, 0, 0), i \geq 1$

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 $w[Nb * i], w[Nb * i + 1], \dots, w[Nb * (i+1) - 1]$.

w_0	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	w_{10}	w_{11}	w_{12}	w_{13}	w_{14}	w_{15}	w_{16}	w_{17}	...
Mã khóa chu kỳ 0						Mã khóa chu kỳ 1						Mã khóa chu kỳ 2						...

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 $Nb - \text{shift}(r, Nb)$ 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.

$$S'_{r, (c + \text{shift}(r, Nb)) \bmod Nb} = S_{r, c} \text{ với } 0 < r < 8 \text{ và } 0 \leq c < Nb \quad (4.9)$$

```

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 $S\text{-box}^{-1}$. Quá trình thay thế 1 byte y dựa vào $S\text{-box}^{-1}$ 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 $\{y_7y_6y_5y_4y_3y_2y_1y_0\}$):

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

với d_i là bit thứ i của giá trị $\{05\}$, $0 \leq i \leq 7$. (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:

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \\ s'_{4,c} \\ s'_{5,c} \\ s'_{6,c} \\ s'_{7,c} \end{bmatrix} = M_{a^{-1}} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \\ s_{4,c} \\ s_{5,c} \\ s_{6,c} \\ s_{7,c} \end{bmatrix}, 0 \leq c \leq Nb \quad (4.13)$$

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
        end for
    end for

```

```

        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:

$$B(F) = \min_{a \neq 0} (W(a) + W(F(a))) \quad (4.16)$$

❖ **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:

$$B(\text{MixColumns}) \leq Nw + 1 \quad (4.17)$$

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.

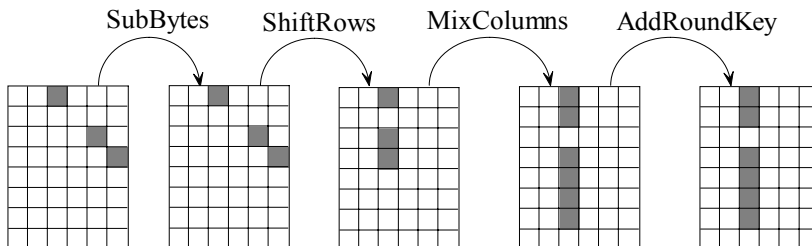
Trong phương pháp tuyến tính, số lượng S-box hoạt động được xác định bằng số lượng byte khác 0 trong các vector được chọn ở trạng thái bắt đầu của chu kỳ [10]. Gọi *mẫu (tương quan) hoạt động* (correlation activity pattern) là mẫu xác định vị trí các byte khác 0 trong trạng thái và gọi *trọng số byte* 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 $W(a)|_j$, 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

STT	Phép biến đổi	Sự ảnh hưởng		
		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.

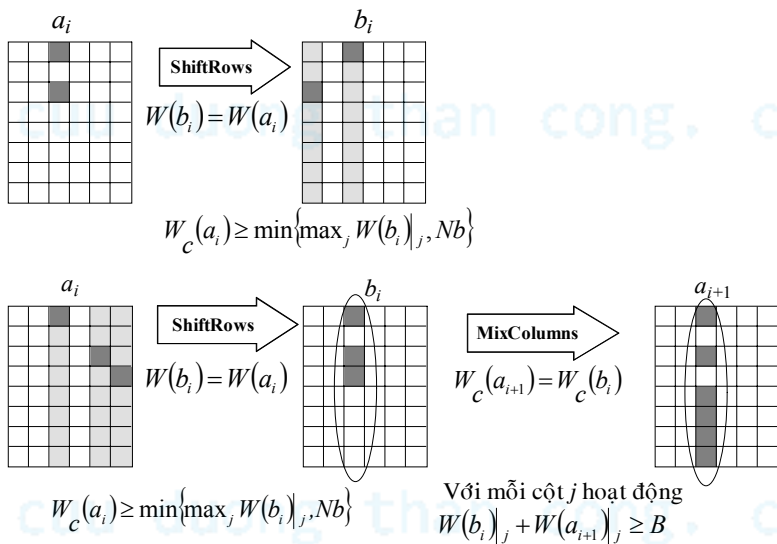
Trong phép biến đổi MixColumns, với mỗi cột hoạt động trong mẫu đầu vào (hoặc mẫu đầu ra) của một chu kỳ, tổng trọng số byte của cột này trong mẫu đầu vào và đầu ra bị chặn dưới bởi Branch Number.

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-1} , mẫu hoạt động kết quả sau khi thực hiện phép biến đổi ShiftRows được ký hiệu là b_{i-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-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, \dots, a_{m-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.

$$W_c(a_1) = Q \Rightarrow W(a_0) + W(a_1) \geq B * Q \quad (4.18)$$

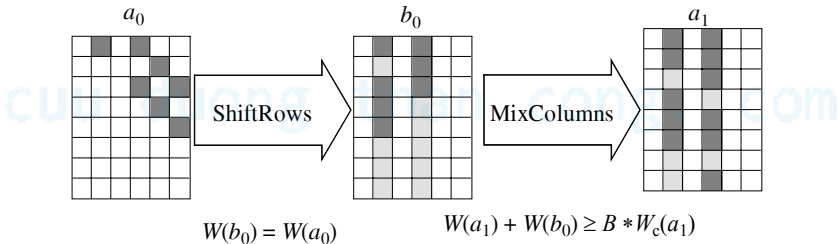
với $B = \text{BranchNumber}(\text{MixColumns})$

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 $B*Q$ phân 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.

$$W_c(a_0) + W_c(a_2) \geq Nb + 1 \quad (4.19)$$

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 .

$$W(b_0)|_g + W(a_1)|_g \geq B \quad (4.20)$$

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.

$$W_c(a_i) \geq \min\{Nb, \max_j W(b_i)|_j\} \quad (4.21)$$

$$W_c(b_i) \geq \min\{Nb, \max_j W(a_i)|_j\} \quad (4.22)$$

$$\Rightarrow W_c(a_0) + W_c(b_1) \geq \min\{Nb, \max_j W(b_0)|_j\} + \min\{Nb, \max_j W(a_1)|_j\} \quad (4.23)$$

$$\Rightarrow W_c(a_0) + W_c(b_1) \geq \min\{Nb, W(b_0)|_g\} + \min\{Nb, W(a_1)|_g\} \quad (4.24)$$

1. Trường hợp 1: Nếu $W(b_0)|_g \geq Nb$ hay $W(a_1)|_g \geq Nb$ thì

$$W_c(a_0) + W_c(b_1) \geq Nb + 1 \quad (4.25)$$

2. Trường hợp 2: Nếu $W(b_0)|_g < Nb$ và $W(a_1)|_g < Nb$ thì

$$W_c(a_0) + W_c(b_1) \geq W(b_0)|_g + W(a_1)|_g \geq B \quad (4.26)$$

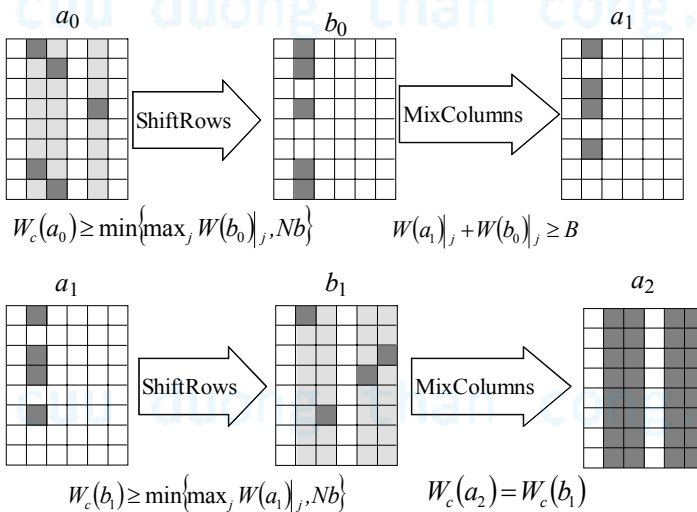
Do Nb chỉ nhận một trong ba giá trị 4, 6, hay 8 và B chỉ nhận một trong ba giá trị là 5, 9 hay 17 (tương ứng với thuật toán gốc, thuật toán mở rộng 256/384/512-bit hay 512/768/1024-bit). Vậy:

$$W_c(a_0) + W_c(b_1) \geq B \geq Nb + 1 \quad (4.27)$$

Do a_2 và b_1 có cùng trọng số cột nên suy ra

$$W_c(a_0) + W_c(b_2) \geq Nb + 1 \quad (4.28)$$

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:

$$\begin{cases} W(a_0) + W(a_1) \geq BW_c(a_1) \\ W(a_2) + W(a_3) \geq BW_c(a_3) \end{cases} \quad (4.29)$$

$$\Rightarrow \sum_{i=0}^3 W(a_i) \geq B(W_c(a_1) + W_c(a_3)) \quad (4.30)$$

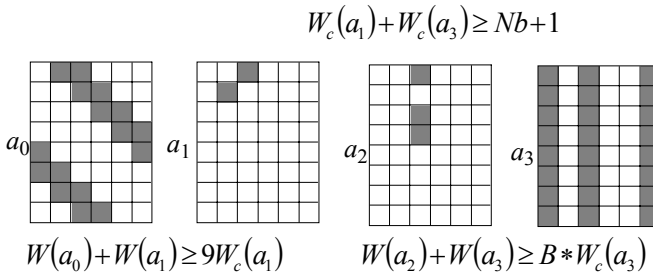
Như vậy, trọng số byte của vết bị chặn dưới bởi $B(W_c(a_1) + W_c(a_3))$

Theo Định lý 4.2, tổng trọng số cột của a_1 và a_3 bị chặn dưới bởi $Nb + 1$.

$$W_c(a_1) + W_c(a_3) \geq Nb + 1 \quad (4.31)$$

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.

cuu duong than cong. com

cuu duong than cong. com

Chương 5

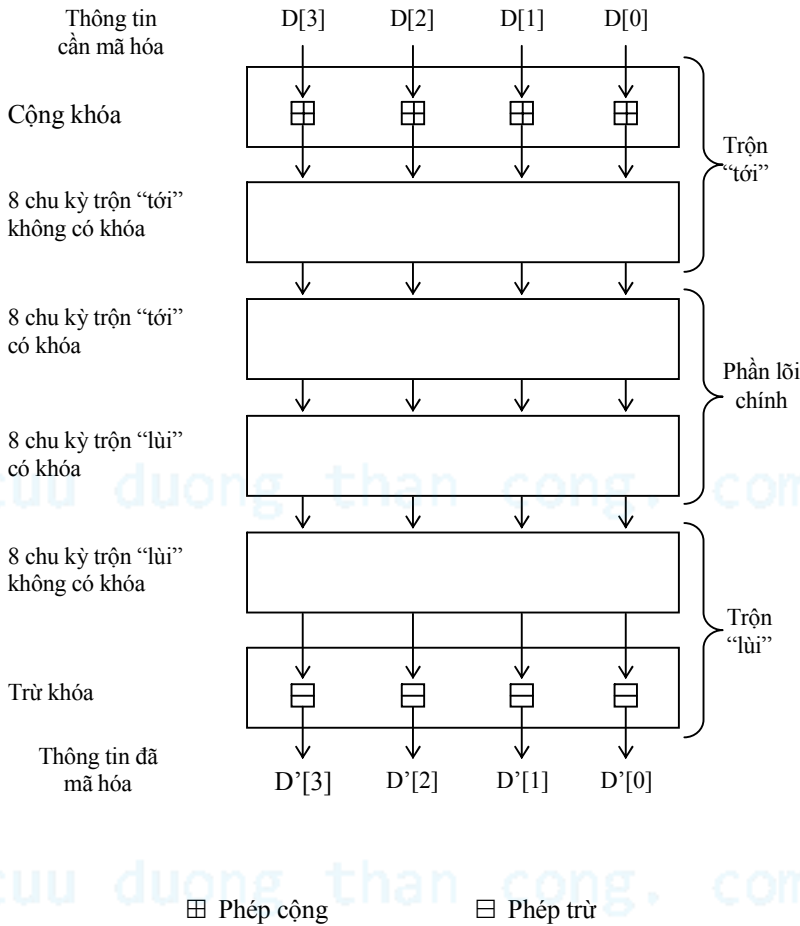
Các thuật toán ứng cử viên AES

 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,

$$S[5i + j] = \text{SHA} - 1(5i \mid c_1 \mid c_2 \mid c_3)_j \quad (5.1)$$

(ở đây $\text{SHA} - 1(.)_j$ là từ thứ j trong kết quả của $\text{SHA} - 1$). 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 $c_1 = 0xb7415162$, $c_2 = 0x283f6a88$ (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 $S[i] \oplus S[j]$ 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 $3 \cdot S[i]$ 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

$$T[i] = T[i] \oplus ((T[i-7 \bmod 15] \oplus T[i-2 \bmod 15]) \lll 3) \oplus (4i + j)$$

với j là số thứ tự của lần lặp ($j = 0, 1, \dots$)

- b) Kế đến, mảng $T[]$ sẽ được biến đổi qua bốn chu kỳ của mạng Feistel loại 1:

$$T[i] = (T[i] + S[9 \text{ bit thấp của } T[i-1 \bmod 15]]) \lll 9$$

với $i = 0, 1, \dots, 14$.

- 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 $K[]$.

$$K[10j + i] = T[4i \bmod 15], i = 0, 1, \dots, 9$$

với j là số thứ tự của lần lặp, $j = 0, 1, \dots$

3. Cuối cùng, xét 16 từ dùng cho phép nhân trong mã hóa (bao gồm các từ $K[5], K[7], \dots, K[35]$) 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 $K[5], K[7], \dots, K[35]$ 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:

$$p = B[j] \lll (5 \text{ bit thấp nhất của } K[i-1])$$

- 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 $K[5], K[7], \dots, K[35]$ 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 suất 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  $k[]$ , ( $4 \leq n \leq 14$ )
//  $K[]$  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  $B[]$ 
 $B[] = \{0\text{xa4a8d57b}, 0\text{x5b5d193b}, 0\text{xc8a8309b}, 0\text{x73f9a978}\}$ 

// Khởi tạo mảng  $T$  với giá trị của mảng khóa  $k[]$ 
 $T[0 \dots n-1] = k[0 \dots n-1]$ ,  $T[n] = n$ ,  $T[n+1 \dots 14] = 0$ 

// 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
         $T[i] = T[i] \oplus ((T[i-7 \bmod 15] \oplus T[i-2 \bmod 15]) \lll 3) \oplus (4i+j)$ 
    repeat 4 lần                                    // 4 chu kỳ biến đổi
        for  $i = 0$  to 14
             $T[i] = (T[i] + S[9 \text{ bit thấp của } T[i-1 \bmod 15]]) \lll 9$ 
    end repeat
    for  $i = 0$  to 9                                // Lưu kết quả vào 10 từ kế tiếp của  $K[]$ 
         $K[10j + i] = T[4i \bmod 15]$ 
    end for

// Sửa đổi các giá trị khóa sẽ sử dụng trong phép nhân
```

for $i = 5, 7, \dots, 35$

$j = 2$ 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 $2 \leq \ell \leq 30$ 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

$p = B[j] \lll r$

// 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:

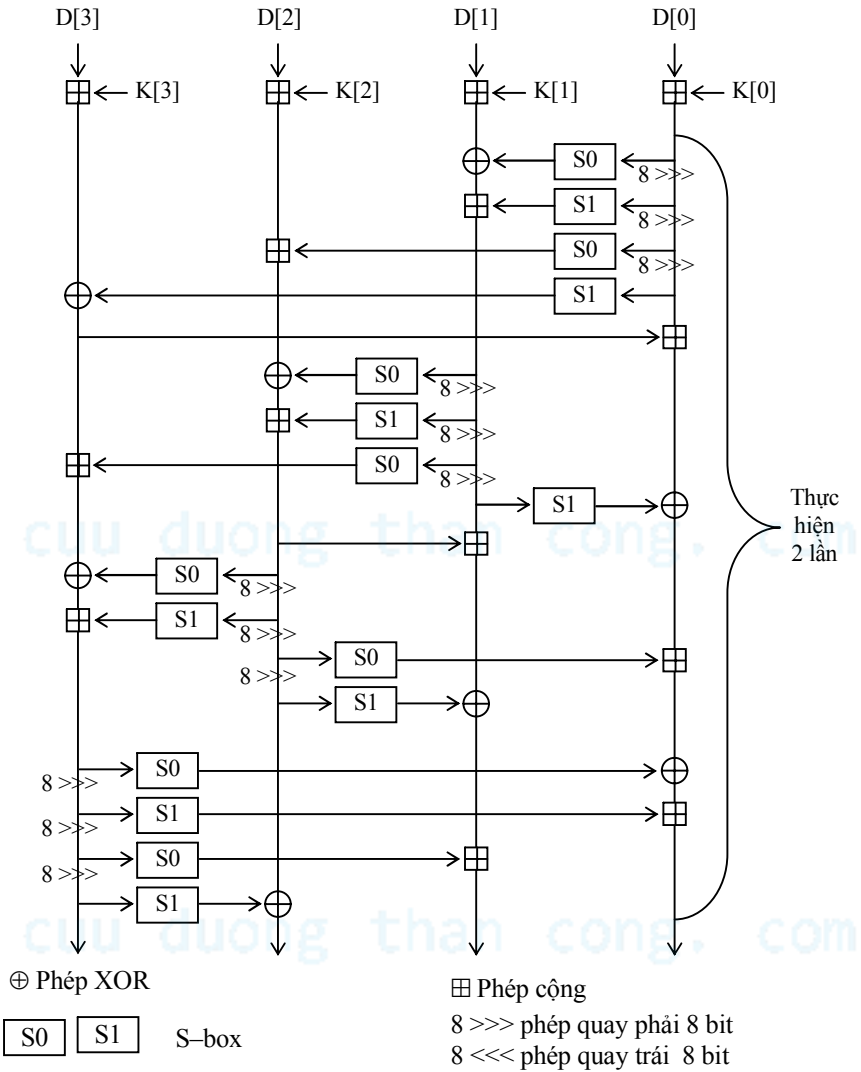
1. $D[]$ là một mảng bốn từ dữ liệu 32 bit. Ban đầu D chứa các từ của văn bản ban đầu (thông tin cần mã hóa). Khi kết thúc quá trình mã hóa, D chứa các từ của thông tin đã được mã hóa.
2. $K[]$ là mảng khóa mở rộng, bao gồm 40 từ 32 bit.
3. $S[]$ là một S-box, bao gồm 512 từ 32 bit, được chia thành hai 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 $S\text{-box}$ $S0$ và b_1, b_3 làm chỉ số trong $S\text{-box}$ $S1$. 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.



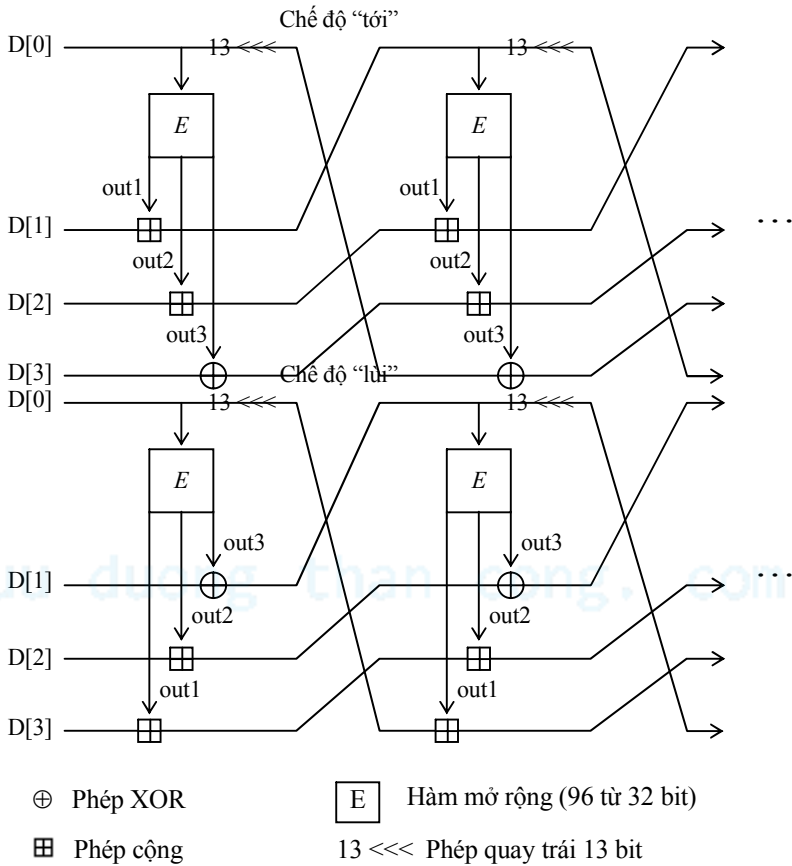
Hình 5.2. Cấu trúc giai đoạn “Trộn cột”

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 lại 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.

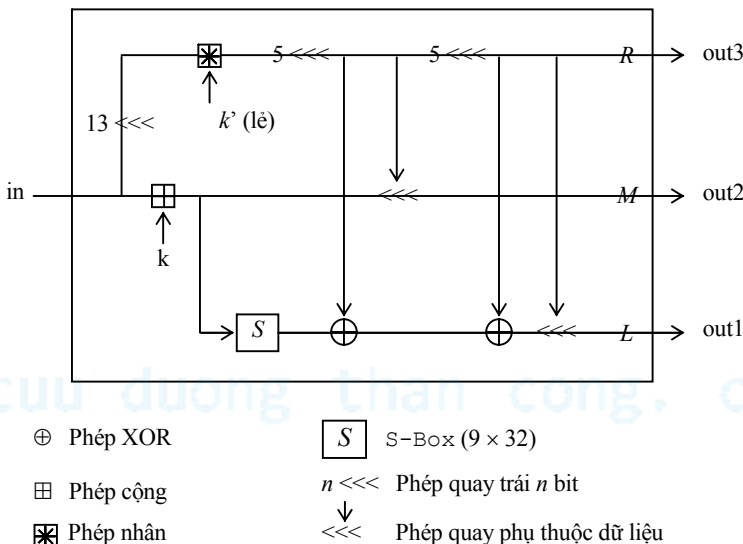


Hình 5.3. Hệ thống Feistel loại 3

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 $S\text{-box}$ 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 $S\text{-box}$.



Hình 5.4. Hàm E

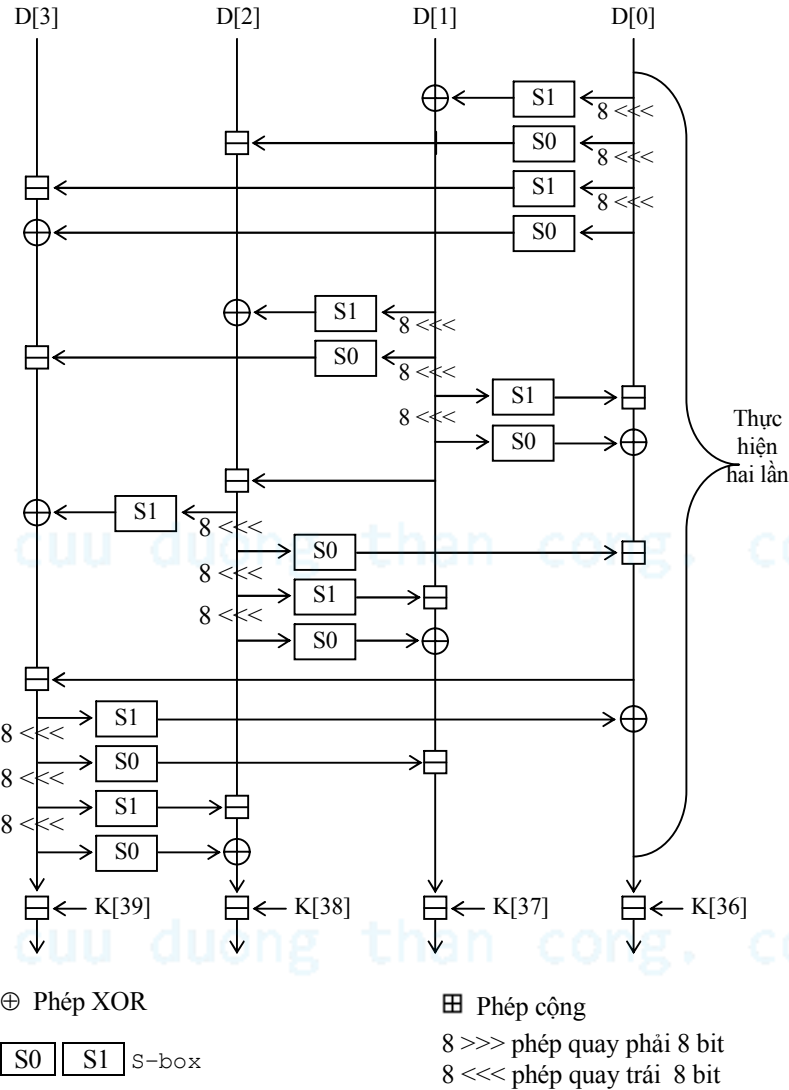
Tiếp theo nhân từ khóa thứ hai (phải chứa một số nguyên lẻ) với R và quay R 5 bit về bên trái (do đó 5 bit cao nhất của tích số trở thành 5 bit thấp nhất của R sau khi quay). Kế đến xor R và L , và cũng xem 5 bit thấp nhất của R như một số bit quay trong khoảng 0 và 31, và quay M về bên trái với số bit 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 lù”

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 $S\text{-box } S1$; b_1 và b_3 làm chỉ số cho $S\text{-box } S0$. 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} .

$a \lll b$ và $a \ggg b$ biểu diễn phép quay của từ 32 bit a sang phải hoặc sang trái b bit.

$(D[3], D[2], D[1], D[0]) \leftarrow (D[0], D[3], D[2], D[1])$ 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

$D[1] = D[1] \oplus S1[\text{byte thứ 1 của } D[0]]$

$D[2] = D[2] - S0[\text{byte thứ 4 của } D[0]]$

$D[3] = D[3] - S1[\text{byte thứ 3 của } D[0]]$

$D[4] = D[4] \otimes S0[\text{byte thứ 2 của } D[0]]$

//Quay từ nguồn sang trái

$D[0] = D[0] \lll 24$

//Quay $D[]$ sang phải 1 từ để chuẩn bị cho chu kỳ tiếp theo

$(D[3], D[2], D[1], D[0]) \leftarrow (D[0], D[3], D[2], D[1])$

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 “tối”

```
// Cộng các subkey vào dữ liệu
```

```
for i = 0 to 3
```

```
    D[i] = D[i] + K[36 + i]
```

```
//Thực hiện 8 chu kỳ trộn “tối”
```

```
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]  $\oplus$  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]  $\oplus$  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
$a \lll b$	quay chu kỳ tròn bên trái b bit
$a \ggg b$	quay chu kỳ tròn bên phải b bit

5.2.1 Khởi tạo và phân bố khóa

RC6 lấy các 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 ($0 \leq b \leq 255$), 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 $L[0], \dots, L[c-1]$. Như vậy byte đầu tiên của khóa sẽ lưu vào vị trí byte thấp của $L[0], \dots$ 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 $S[0, \dots, 2r + 3]$.

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 $L[0, \dots, c-1]$ (gồm c -từ)

r là số lượng chu kỳ

Output: Các khóa chu kỳ w bit $S[0, \dots, 2r+3]$

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

$A = S[i] = (S[i] + A + B) \lll 3$

$B = L[j] = (L[j] + A + B) \lll (A + B)$

$i = (i + 1) \bmod (2r + 4)$

$j = (j + 1) \bmod c$

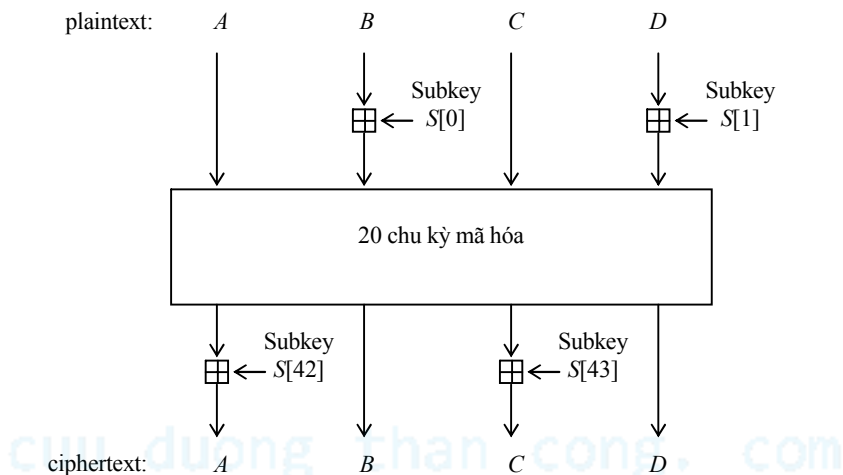
end for

End

5.2.2 Quy trình mã hóa

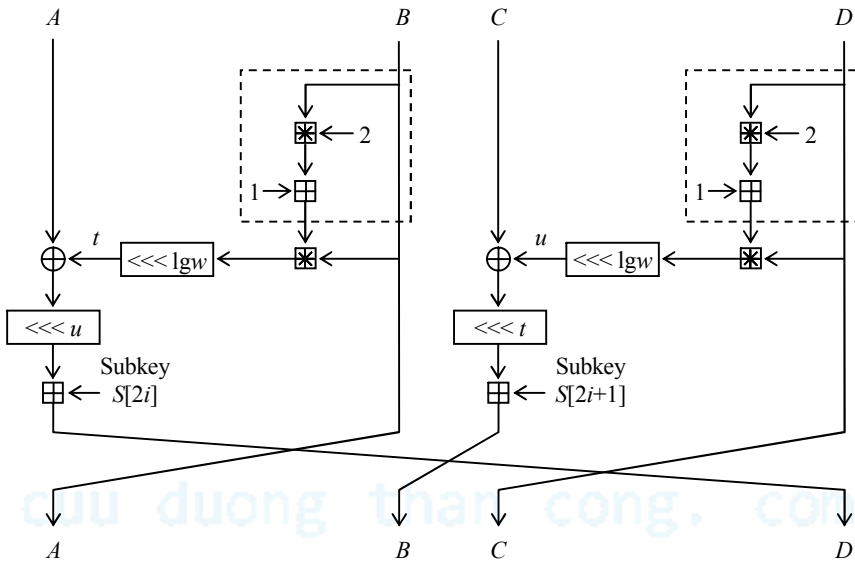
RC6 làm việc với bốn từ w bit A, B, C, D chứa các dữ liệu đưa vào ban đầu cũng như dữ liệu mã hóa đưa ra cuối quy trình mã hóa. Byte đầu tiên của văn bản ban

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 lgw (= 5 cho kích thức từ = 32 bit) vị trí và lưu vào biến t . Tương tự, quay $f(d) = d \times (2d + 1)$ sang trái lgw 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 $2i$ (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 $2i + 1$.



\oplus phép XOR

\otimes phép nhân

\boxplus phép cộng

$\lll n$ 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) $S[0, \dots, 2r + 3]$

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

$t = (B \times (2B + 1)) \lll \lg w$

$u = (D \times (2D + 1)) \lll \lg w$

$A = ((A \oplus t) \lll u) + S[2i]$

$C = ((C \oplus u) \lll t) + S[2i + 1]$

$(A, B, C, D) = (B, C, D, A)$

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) $S[0, \dots, 2r + 3]$

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

$(A, B, C, D) = (D, A, B, C)$

$u = (D \times (2D + 1)) \lll \lg w$

$t = (B \times (2B + 1)) \lll \lg w$

$C = ((C - S[2i + 1]) \ggg t) \oplus u$

$A = ((A - S[2i]) \ggg u) \oplus t$

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 (KA_0, \dots, KA_{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, \dots, K_{32}) bằng cách ghi khóa K thành 8 từ 32 bit (w_{-8}, \dots, w_{-1}) và mở rộng các từ này thành khóa trung gian w_0, \dots, w_{131} bằng công thức sau:

$$w_i = (w_{i-8} \oplus w_{i-5} \oplus w_{i-3} \oplus w_{i-1} \oplus \phi \otimes i) \lll 11 \quad (5.3)$$

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.

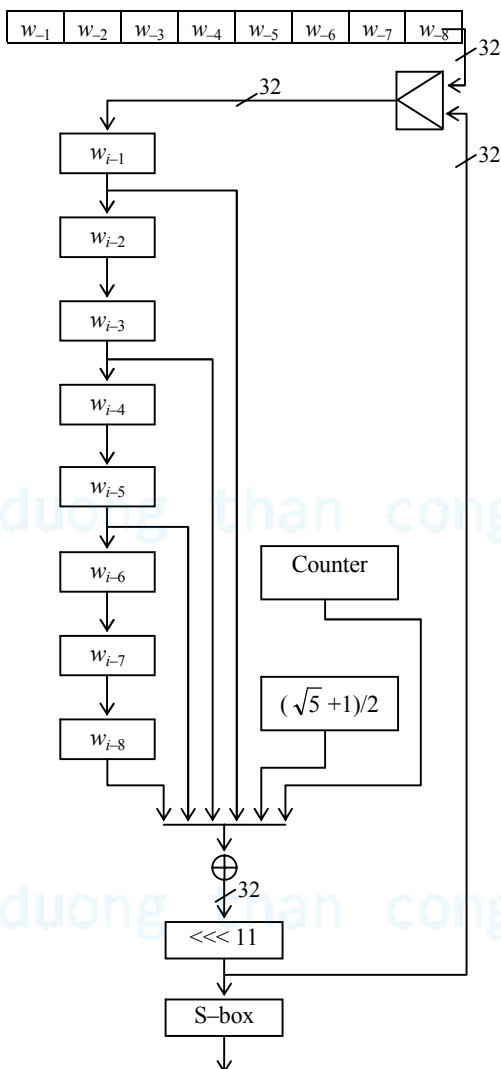
Những khóa thực hiện một chu kỳ được suy ra từ các khóa trước khi sử dụng các S-box. Sử dụng S-box để biến đổi các khóa w_i thành các từ k_i của khóa chu kỳ theo cách sau:

$$\begin{aligned}
 \{k_0, k_1, k_2, k_3\} &= S_3(w_0, w_1, w_2, w_3) \\
 \{k_4, k_5, k_6, k_7\} &= S_2(w_4, w_5, w_6, w_7) \\
 \{k_8, k_9, k_{10}, k_{11}\} &= S_1(w_8, w_9, w_{10}, w_{11}) \\
 \{k_{12}, k_{13}, k_{14}, k_{15}\} &= S_0(w_{12}, w_{13}, w_{14}, w_{15}) \\
 \{k_{16}, k_{17}, k_{18}, k_{19}\} &= S_7(w_{16}, w_{17}, w_{18}, w_{19}) \\
 &\dots \\
 \{k_{124}, k_{125}, k_{126}, k_{127}\} &= S_4(w_{124}, w_{125}, w_{126}, w_{127}) \\
 \{k_{128}, k_{129}, k_{130}, k_{131}\} &= S_3(w_{128}, w_{129}, w_{130}, w_{131})
 \end{aligned} \tag{5.4}$$

Ta đánh số lại các giá trị 32 bit k_j giống các subkey 128 bit K_i (cho $i \in 0, \dots, r$) 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 $\text{serpent}[\cdot]$ là một dãy chứa 4 bit thấp nhất (thấp nhất) của mỗi 16 kí tự ASCII "sboxesforserpent". Cho $\text{sbox}[\cdot][\cdot]$ là một dãy (32×16) chứa 32 hàng của 8 S-box DES, ở đây $\text{sbox}[r][\cdot]$ là hàng r . Hàm $\text{swapentries}(\cdot, \cdot)$ dùng để hoán vị hai phần tử.

Dưới đây là đoạn mã giả phát sinh S-box

```

index = 0
repeat
    currentstbox = index mod 32;
    for i = 0 to 15
        j = sbox[(currentstbox+1) mod 32][serpent[i]];
        swapentries (sbox[currentstbox][i], sbox[currentstbox][j]);
    end for
    if sbox[currentstbox][.] 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 $S\text{-box}$ và $S\text{-box}$ nghịch đảo được sử dụng trong thuật toán Serpent.

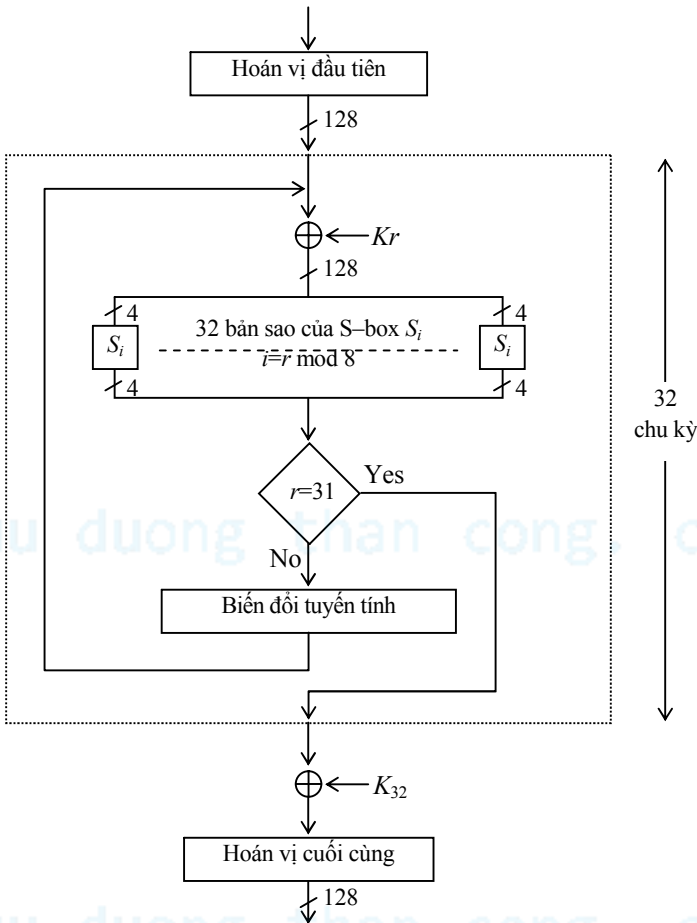
5.3.4 Quy trình mã hóa

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 $S\text{-box}$ 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 $B\hat{A}_{i+1} \dots$ 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:

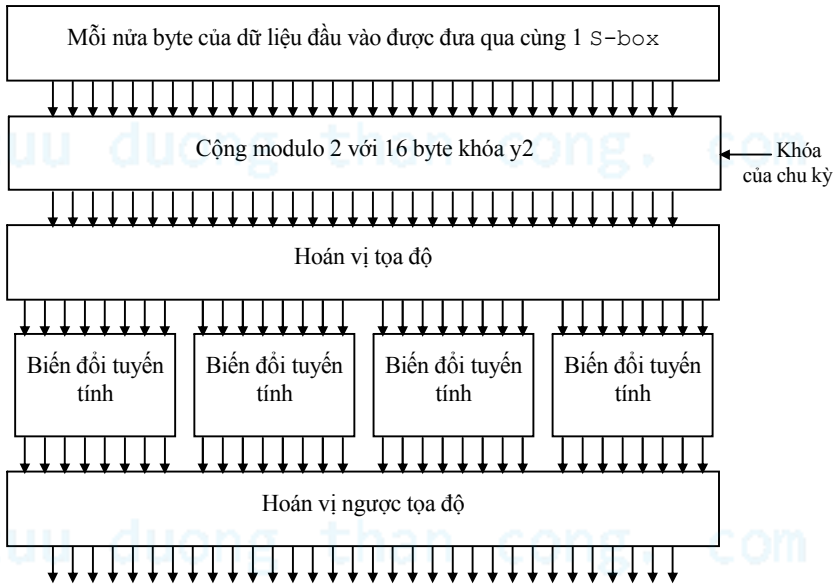
$$X_i \leftarrow B_i \oplus K_i$$

$$Y_i \leftarrow S_i(X_i)$$

$$B_{i-1} \leftarrow L(Y_i), i = 0, \dots, 30$$

$$B_{i-1} \leftarrow Y_i \oplus K_{i-1}, i = 31 \quad (5.6)$$

Hình 5.8 thể hiện các bước thực hiện trong chu kỳ thứ i ($i = 0, \dots, 30$) 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 ($i = 0, \dots, 30$) của quy trình mã hóa Serpent

Ở mỗi chu kỳ hàm R_i ($i \in \{0, \dots, 31\}$) 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:

$$\begin{aligned}
 y_0 &\leftarrow y_0 \lll 13 \\
 y_2 &\leftarrow y_2 \lll 3 \\
 y_1 &\leftarrow y_0 \oplus y_1 \oplus y_2 \\
 y_3 &\leftarrow y_2 \oplus y_3 \oplus (y_0 \ll 3) \\
 y_1 &\leftarrow y_1 \lll 1 \\
 y_3 &\leftarrow y_3 \lll 7 \\
 y_0 &\leftarrow y_0 \oplus y_1 \oplus y_3 \\
 y_2 &\leftarrow y_2 \oplus y_3 \oplus (y_1 \ll 7) \\
 y_0 &\leftarrow y_0 \lll 5 \\
 y_2 &\leftarrow y_2 \lll 22 \\
 B_{i+1} &\leftarrow (y_0, y_1, y_2, y_3)
 \end{aligned} \tag{5.7}$$

Trong các biểu thức trên đây, ký hiệu \lll là phép quay trái và \ll là phép dịch trái.

Bộ tám S-box ($S_0 \dots S_7$) 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.

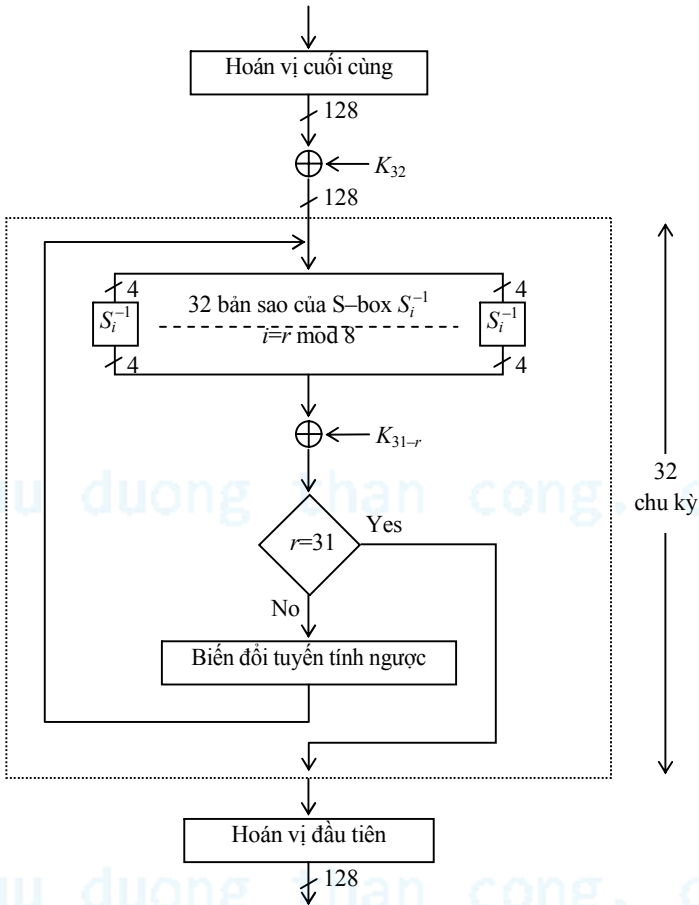
Vậy 32 chu kỳ sử dụng 8 S-box khác nhau, mỗi S-box ánh xạ 4 bit vào thành 4 bit ra. Mỗi S-box sử dụng 4 chu kỳ riêng biệt và trong mỗi chu kỳ S-box được sử dụng 32 lần song song.

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:

$$\begin{aligned}
 B\hat{A}_0 &= \text{IP}(P) \\
 B\hat{A}_{i+1} &= R_i(B\hat{A}_i) \\
 C &= \text{FP}(B\hat{A}_{32}) \\
 R_i(X) &= L(S\hat{A}_i(X \oplus K\hat{A}_i)), i = 0, \dots, 30 \\
 R_i(X) &= S\hat{A}_i(X \oplus K\hat{A}_i) \oplus K\hat{A}_{32}, i = 31
 \end{aligned} \tag{5.8}$$

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 ($S\text{-box}^{-1}$) 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, \dots, 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, \dots, m_{8k-1} . Các byte này được biến đổi thành $2k$ từ 32 bit.

$$M_i = \sum_{j=0}^3 m_{(4i+j)} \cdot 2^{8j}, I = 0, \dots, 2k-1 \quad (5.9)$$

sau đó biến đổi thành hai từ vector có chiều dài k

$$\begin{aligned} M_e &= (M_0, M_2, \dots, M_{2k-2}) \\ M_o &= (M_1, M_3, \dots, M_{2k-1}) \end{aligned} \quad (5.10)$$

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.

$$\begin{pmatrix} S_{i,0} \\ S_{i,1} \\ S_{i,2} \\ S_{i,3} \end{pmatrix} = \begin{pmatrix} \cdot & \dots & \cdot \\ \vdots & RS & \vdots \\ \cdot & \dots & \cdot \end{pmatrix} \cdot \begin{pmatrix} m_{8i} \\ m_{8i+1} \\ m_{8i+2} \\ m_{8i+3} \\ m_{8i+4} \\ m_{8i+5} \\ m_{8i+6} \\ m_{8i+7} \end{pmatrix} \quad (5.11)$$

$$S_i = \sum_{j=0}^3 S_{i,j} \cdot 2^{8j} \quad (5.12)$$

với $i = 0, \dots, k-1$ và $S = (S_{k-1}, S_{k-2}, \dots, S_0)$

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:

$$RS = \begin{pmatrix} 01 & A4 & 55 & 87 & 5A & 58 & DB & 9E \\ A4 & 56 & 82 & F3 & 1E & C6 & 68 & E5 \\ 02 & A1 & FC & C1 & 47 & AE & 3D & 19 \\ A4 & 55 & 87 & 5A & 58 & DB & 9E & 03 \end{pmatrix} \quad (5.13)$$

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 ($\neq 128, 192, 256$), các khóa này được thêm vào các số 0 cho đủ chiều dài xác định. Ví dụ: một khóa 80 bit m_0, \dots, m_9 sẽ mở rộng bằng các đặt $m_i = 0$ với $i = 10, \dots, 15$ 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 $L = (L_0, \dots, L_{k-1})$ 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

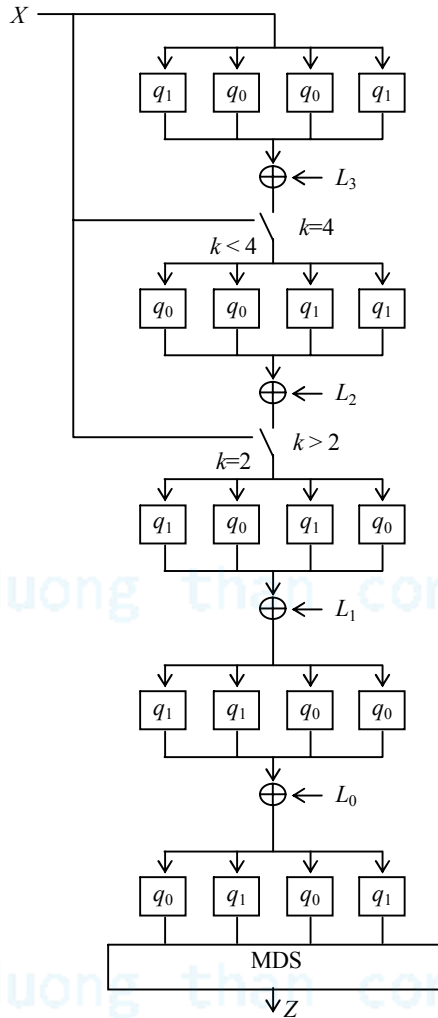
$$\begin{aligned} l_{i,j} &= \left\lfloor L_i / 2^{8j} \right\rfloor \bmod 2^{8j} \\ x_j &= \left\lfloor X / 2^{8j} \right\rfloor \bmod 2^8 \end{aligned} \quad (5.14)$$

với $i = 0, \dots, k-1$ và $j = 0, \dots, 3$. Sau đó lần lượt thay thế và áp dụng phép XOR.

$$y_{k,j} = x_j, j = 0, \dots, 3 \quad (5.15)$$

Nếu $k = 4$, ta có:

$$\begin{aligned} y_{3,0} &= q_1[y_{4,0}] \oplus l_{3,0} \\ y_{3,1} &= q_0[y_{4,1}] \oplus l_{3,1} \\ y_{3,2} &= q_0[y_{4,2}] \oplus l_{3,2} \\ y_{3,3} &= q_1[y_{4,3}] \oplus l_{3,3} \end{aligned} \quad (5.16)$$



Hình 5.12. Hàm h

Nếu $k \geq 3$, ta có:

$$\begin{aligned}
 y_{2,0} &= q_1[y_{3,0}] \oplus l_{2,0} \\
 y_{2,1} &= q_0[y_{3,1}] \oplus l_{2,1} \\
 y_{2,2} &= q_0[y_{3,2}] \oplus l_{2,2} \\
 y_{2,3} &= q_1[y_{3,3}] \oplus l_{2,3}
 \end{aligned} \tag{5.17}$$

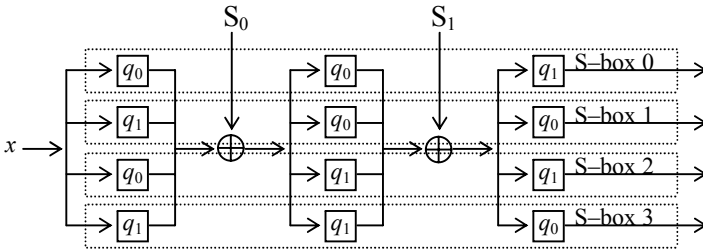
Trong mọi trường hợp ta có

$$\begin{aligned}
 y_0 &= q_1[q_0[y_{2,0}] \oplus l_{1,0}] \oplus l_{0,0} \\
 y_1 &= q_0[q_0[q_1[y_{2,1}] \oplus l_{1,1}] \oplus l_{0,1} \\
 y_2 &= q_1[q_1[q_0[y_{2,2}] \oplus l_{1,2}] \oplus l_{0,2} \\
 y_3 &= q_0[q_1[q_1[y_{2,3}] \oplus l_{1,3}] \oplus l_{0,3}
 \end{aligned} \tag{5.18}$$

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

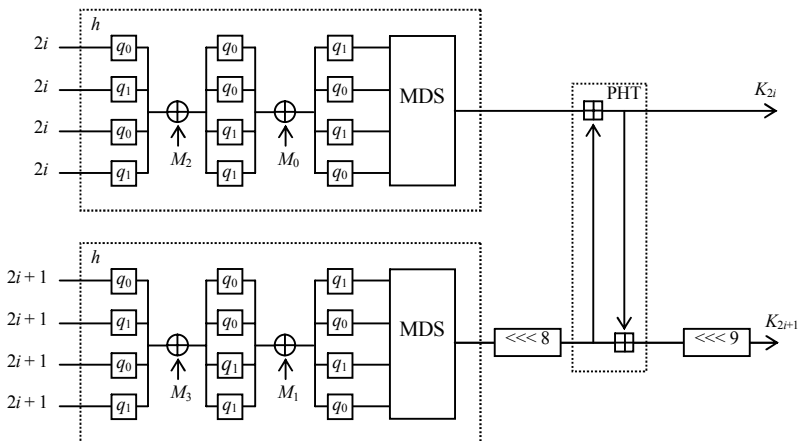
Ở đây $s_{i,j}$ là các byte lấy từ các byte khóa sử dụng ma trận RS. Để ý rằng với các byte khóa bằng nhau sẽ không có cặp S-box bằng nhau. Khi mọi $s_{i,j} = 0$ thì $s_0(x) = q_1[s_1^{-1}(x)]$.

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 .

$$\begin{aligned}
 \rho &= 2^{24} + 2^{16} + 2^8 + 2^0 \\
 A_i &= h(2i\rho, M_e) \\
 B_i &= \text{ROL}(h((2i+1)\rho, M_o), 8) \\
 K_{2i} &= (A_i + B_i) \bmod 2^{32} \\
 K_{2i+1} &= \text{ROL}((A_i + 2B_i) \bmod 2^{32}, 9)
 \end{aligned} \tag{5.20}$$



Hình 5.14. Mô hình phát sinh subkey K_i

Hằng số ρ sử dụng để nhân đôi các byte, $i \in 0, \dots, 255$, $i\rho$ 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:

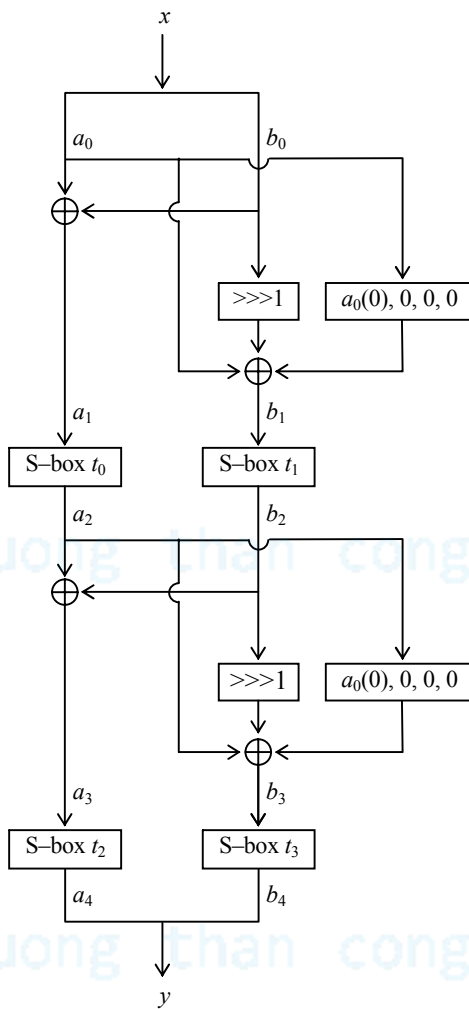
$$\begin{aligned}
 a_0, b_0 &= [x/16], x \bmod 16 \\
 a_1 &= a_0 \oplus b_0 \\
 b_1 &= a_0 \oplus \text{ROR}_4(b_0, 1) \oplus 8a_0 \bmod 16 \\
 a_2, b_2 &= t_0[a_1], t_1[b_1] \\
 a_3 &= a_2 \oplus b_2 \\
 b_3 &= a_2 \oplus \text{ROR}_4(b_2, 1) \oplus 8a_2 \bmod 16 \\
 a_4, b_4 &= t_2[a_3], t_3[b_3] \\
 y &= 16b_4 + a_4
 \end{aligned} \tag{5.21}$$

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 $S\text{-box}$ 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 $S\text{-box}$ 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 $S\text{-box}$ 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 $S\text{-box}$ 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

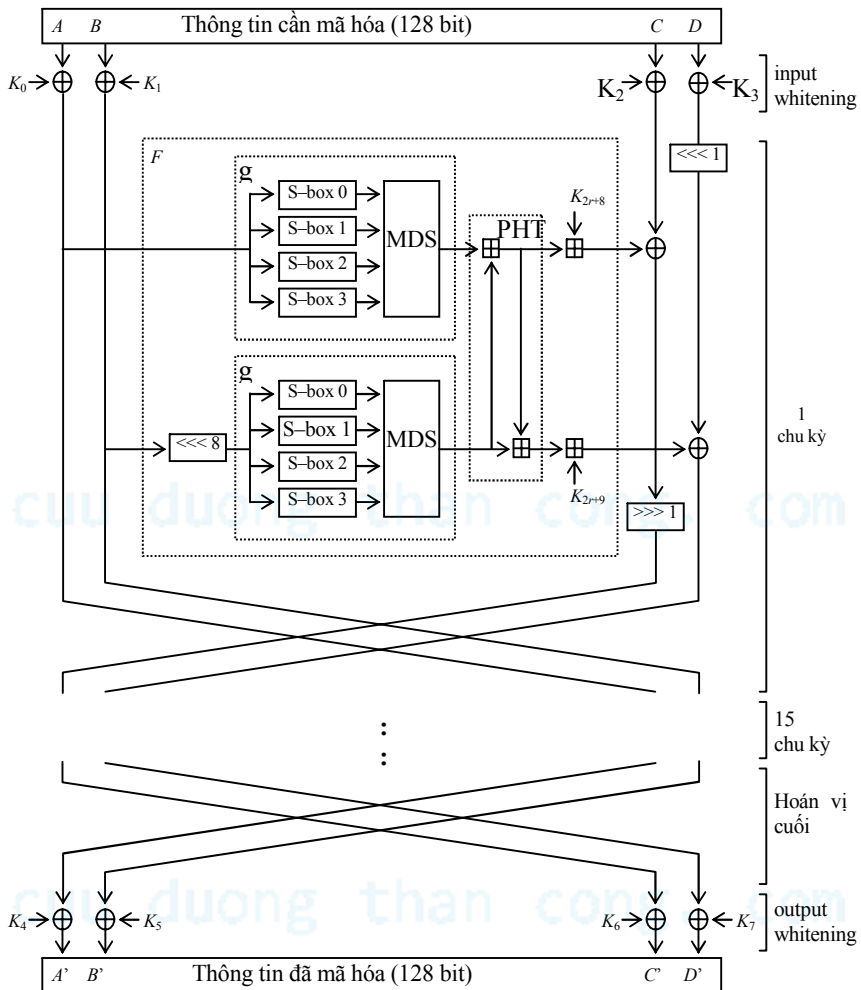
5.4.2 Quy trình mã hóa

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.

$$P_i = \sum_{j=0}^3 P_{(4i+j)} \cdot 2^{8j}, \quad i = 0, ..., 3 \quad (5.24)$$



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:

$$R_{0,i} = P_i \oplus K_i, \quad i = 0, \dots, 3 \quad (5.25)$$

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:

$$\begin{aligned} (F_{r,0}, F_{r,1}) &= F(R_{r,0}, R_{r,1}, r) \\ R_{r+1,0} &= \text{ROR}(R_{r,2} \oplus F_{r,0}, 1) \\ R_{r+1,1} &= \text{ROL}(R_{r,3}, 1) \oplus F_{r,1} \\ R_{r+1,2} &= R_{r,0} \\ R_{r+1,3} &= R_{r,1} \end{aligned} \quad (5.26)$$

$r \in (0, \dots, 15)$, 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.

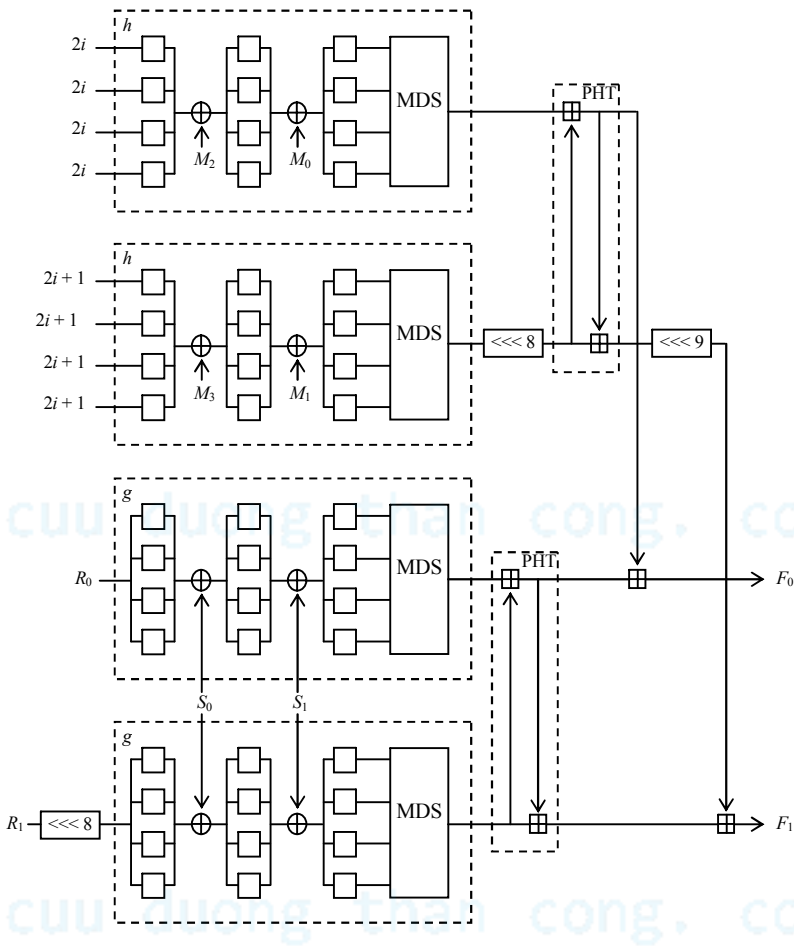
Bước whitening dữ liệu ra không thực hiện thao tác hoán chuyển ở chu kỳ cuối mà nó thực hiện phép XOR các từ dữ liệu với bốn từ khóa mở rộng.

$$C_i = R_{16, (i+2) \bmod 4} \oplus K_{i+4}, \quad i = 0, \dots, 3 \quad (5.27)$$

Sau đó, bốn từ của văn bản mã hóa được ghi ra thành 16 byte c_0, \dots, c_{15} sử dụng quy ước little-endian như đã áp dụng với văn bản ban đầu.

$$c_i = \left[\frac{C_{\lfloor i/4 \rfloor}}{2^{8(i \bmod 4)}} \right] \bmod 2^8, \quad i = 0, \dots, 15 \quad (5.28)$$

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.

$$\begin{aligned} T_0 &= g(R_0) \\ T_1 &= g(\text{ROL}(R_1, 8)) \\ F_0 &= (T_0 + T_1 + K_{2r+8}) \bmod 2^{32} \\ F_1 &= (T_0 + 2T_1 + K_{2r+9}) \bmod 2^{32}, (F_0, F_1) \text{ là kết quả của } F. \end{aligned} \quad (5.29)$$

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 $\text{GF}(2^8)$ và vector này nhân với ma trận MDS 4×4 (sử dụng vùng $\text{GF}(2^8)$ cho việc tính toán). Vector kết quả được xem như một từ 32 bit và nó cũng là kết quả của hàm g .

$$\begin{aligned} x_i &= [X/2^{8i}] \bmod 2^8, i = 0, \dots, 3 \\ y_i &= s_i[x_i], i = 0, \dots, 3 \\ \begin{pmatrix} z_0 \\ z_1 \\ z_2 \\ z_3 \end{pmatrix} &= \begin{pmatrix} \cdot & \dots & \cdot \\ \vdots & \text{MDS} & \vdots \\ \cdot & \dots & \cdot \end{pmatrix} \cdot \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix} \\ Z &= \sum_{i=0}^3 z_i \cdot 2^{8i} \end{aligned} \quad (5.30)$$

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 $a_i \in GF(2)$

($i = 0, \dots, k-1$) đồng nhất với giá trị byte $\sum_{i=0}^7 a_i 2^i$.

Ta có ma trận MDS cho như sau:

$$\text{MDS} = \begin{pmatrix} 01 & EF & 5B & 5B \\ 5B & EF & EF & 01 \\ EF & 5B & 01 & EF \\ EF & 01 & EF & 5B \end{pmatrix} \quad (5.31)$$

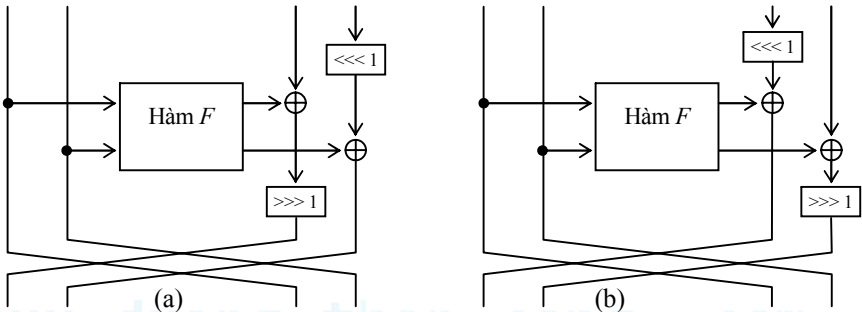
ở đây các phần tử được viết dưới dạng giá trị byte hexa.

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.

cuu duong than cong. com

cuu duong than cong. com

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.

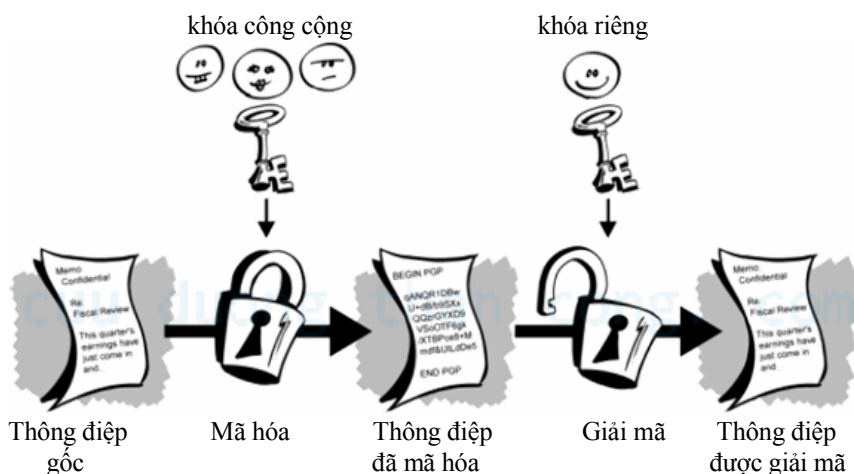
6.1 Hệ thống mã hóa khóa công cộng

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:

$K = \{(n, p, q, a, b): n = pq, p, q \text{ là số nguyên tố}, ab \equiv 1 \pmod{\phi(n)}\}$

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

$e_k(x) = x^a \bmod n$ và $d_k(y) = y^b \bmod n$, 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:

$$n = p \cdot q$$

$$\phi(n) = (p-1)(q-1) \quad (6.1)$$

Thay $q = n/p$, ta được phương trình bậc hai:

$$p^2 - (n - \phi(n) + 1)p + n = 0 \quad (6.2)$$

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 $n = p \cdot q$ (p, q chưa biết) và B là một số nguyên đủ lớn, với mỗi thừa số nguyên tố k , $k \leq B \wedge k \mid (p-1) \Rightarrow (p-1) \mid B!$

Ở cuối vòng lặp (bước 2), ta có

$$a \equiv 2^{B!} \pmod{n} \quad (6.3)$$

Suy ra

$$a \equiv 2^{B!} \pmod{p} \quad (6.4)$$

Do $p \mid n$ nên theo định lý Fermat, ta có :

$$2^{p-1} \equiv 1 \pmod{p} \quad (6.5)$$

Do $(p-1) \mid B!$, nên ở bước 3 của thuật toán, ta có:

$$a \equiv 1 \pmod{p}. \quad (6.6)$$

Vì thế, ở bước 4:

$$p \mid (a - 1) \text{ và } p \mid n, \quad (6.7)$$

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 $x = \pm 1 \pmod{p}$.

Tương tự, phương trình $x^2 \equiv 1 \pmod{q}$ có hai nghiệm (modulo q) là $x = \pm 1 \pmod{q}$.

Do

$$x^2 \equiv 1 \pmod{n} \Leftrightarrow x^2 \equiv 1 \pmod{p} \wedge x^2 \equiv 1 \pmod{q} \quad (6.8)$$

nên ta có

$$x^2 \equiv 1 \pmod{n} \Leftrightarrow x = \pm 1 \pmod{p} \wedge x = \pm 1 \pmod{q} \quad (6.9)$$

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 $w^r, w^{2r}, w^{4r}, \dots$ cho đến khi tồn tại t sao cho:

$$w^{2^t r} \equiv 1 \pmod{n} \quad (6.10)$$

Do $ab - 1 = 2^s r \equiv 0 \pmod{\phi(n)}$ nên $w^{2^s r} \equiv 1 \pmod{n}$. 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 $1 \leq w \leq n - 1$

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 $v \not\equiv 1 \pmod{n}$ **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:

$$C_1 = C^e \pmod{n}$$

$$C_2 = C_1^e \pmod{n}$$

...

$$C_i = C_{i-1}^e \pmod{n} \quad (6.11)$$

Nếu có một phần tử C_j trong chuỗi $C_1, C_2, C_3, \dots, C_i$ sao cho $C_j = C$ thì khi đó anh ta sẽ tìm được $M = C_{j-1}$ bởi vì:

$$C_j = C_{j-1}^e \pmod{n}$$

$$C = M^e \pmod{n} \quad (6.12)$$

□ 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 gửi có $e = 17$, $n = 35$. Nếu anh ta muốn gửi 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:

$$\forall M, M = M^{865} \bmod (109 \cdot 97),$$

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 $n = p \cdot q$ (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

tổ khi và chỉ khi n không có ước số dương nào thuộc đoạn $[2, \dots, \lfloor \sqrt{n} \rfloor]$. Như vậy, ta có: n là số nguyên tố $\Leftrightarrow \forall i \in [2, \dots, \lfloor \sqrt{n} \rfloor], \neg(n \equiv 0 \pmod i)$

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-Rabin; trong đó, thuật toán Miller-Rabin 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 $a \in \{1, 2, \dots, n-1\}$

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 $1 - \frac{1}{4^k} \rightarrow 1$, 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 $b_{l-1}b_{l-2}...b_1b_0$, $b_i \in \{0, 1\}$, $0 \leq i < l$

$z = 1$

$x = x \bmod n$

for $i = l-1$ **downto** 0

$z = z^2 \bmod n$

if $b_i = 1$ **then**

$z = z \times x \bmod n$

end if

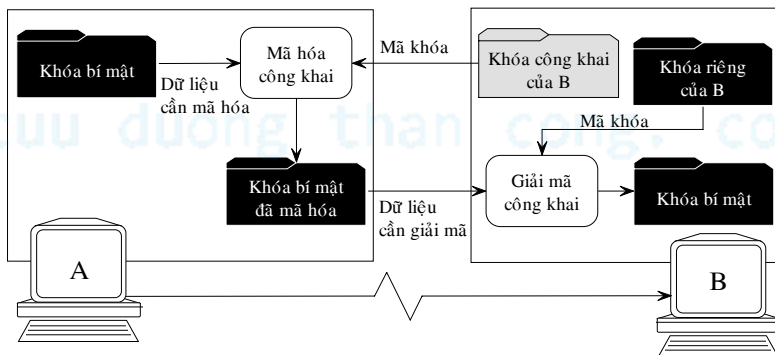
end for

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

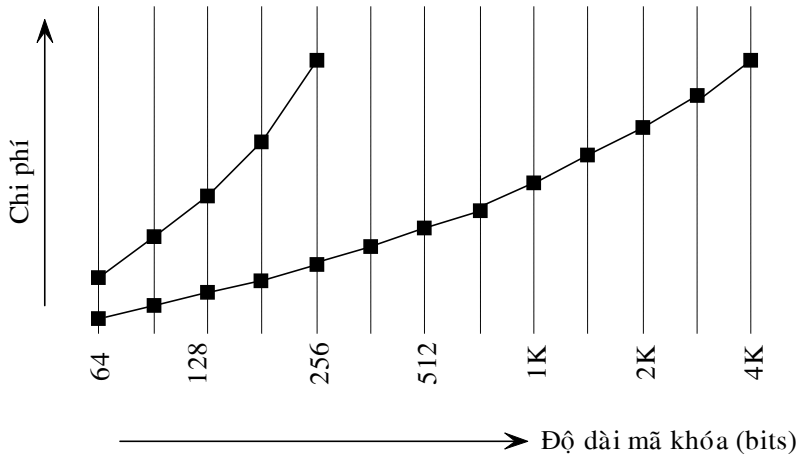
Vấn đề còn lại đối với khóa công cộng là làm cách nào xác nhận được chính xác người chủ thật sự của một khóa công cộng (xem Chương 10).

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.

cuu duong than cong. com

cuu duong than cong. com

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 $\text{sig}_k \in S$ và thuật toán xác nhận chữ ký tương ứng $\text{ver}_k \in V$. Mỗi thuật toán $\text{sig}_k : P \rightarrow A$ và $\text{ver}_k : P \times A \rightarrow \{true, false\}$ là các hàm thỏa điều kiện:

$$\forall x \in P, \forall y \in A : \text{ver}(x, y) = \begin{cases} true & \text{nếu } y = \text{sig}(x) \\ false & \text{nếu } y \neq \text{sig}(x) \end{cases} \quad (7.1)$$

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:

$$\text{sig}_K(x) = x^a \bmod n$$

và

$$\text{ver}_K(x, y) = true \Leftrightarrow x \equiv y^b \pmod{n}, \text{ với } x, y \in \mathbb{Z}_n$$

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

$$\alpha^a \equiv \beta \pmod{p} \quad (7.2)$$

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:

$$\text{sig}_K(x, k) = (\gamma, \delta)$$

với

$$\gamma = \alpha^k \pmod{p}$$

và

$$\delta = (x - a\gamma) k^{-1} \pmod{p-1}$$

Với $x, \gamma \in Z_p^*$ và $\delta \in Z_{p-1}$, định nghĩa

$$\text{ver}_K(x, \gamma, \delta) = \text{true} \Leftrightarrow \beta^\gamma \gamma^\delta \equiv \alpha^x \pmod{p}$$

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:

$$\text{sig}_K(x, k) = (\gamma, \delta)$$

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

$$\text{ver}_K(x, \gamma, \delta) = \text{true} \Leftrightarrow (\alpha^{e_1} \beta^{e_2} \bmod p) \bmod q = \gamma$$

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.

cuu duong than cong. com

cuu duong than cong. com

Chương 8

Phương pháp ECC

 Trong chương 6 và 7, chúng ta đã tìm hiểu 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 nay. 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].

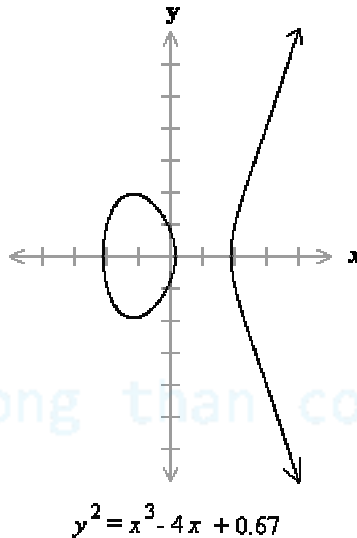
8.1.1 Công thức Weierstrasse và đường cong elliptic

Gọi K là một trường hữu hạn hoặc vô hạn. Một đường cong elliptic được định nghĩa trên trường K bằng công thức Weierstrass:

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \quad (8.1)$$

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

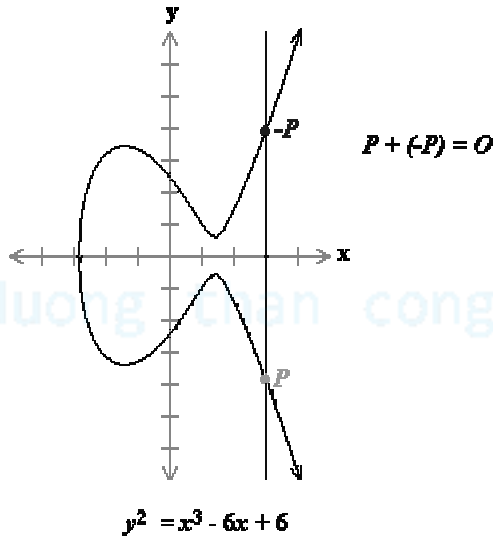
Đường cong elliptic E trên trường số thực R là tập hợp các điểm (x, y) thỏa mãn công thức:

$$y^2 = x^3 + a_4x + a_6 \text{ với } a_4, a_6 \in R \quad (8.2)$$

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, $\forall P(x, y) \in E(R), \quad P + O = O + P = P :$

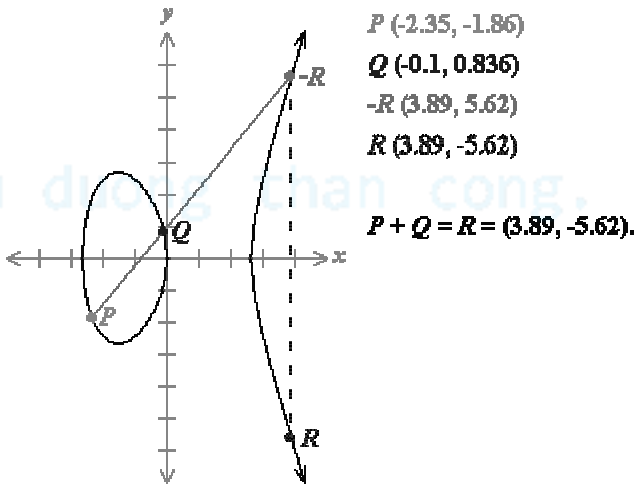
$$\forall P(x, y) \in E(R) : \pm y = \sqrt{x^3 + a_4x + a_6} \quad (8.3)$$

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:

$$P + (-P) = (x, y) + (x, -y) = O \quad (8.4)$$

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 $P + Q = R, R \in E(R)$.



$$y^2 = x^3 - 7x$$

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:

$$\begin{aligned} P &= (x_1, y_1) \\ Q &= (x_2, y_2) \\ R &= P + Q = (x_3, y_3) \end{aligned} \tag{8.5}$$

trong đó $P, Q, R \in E(R)$ và:

$$\begin{aligned} x_3 &= \theta^2 - x_1 - x_2 \\ y_3 &= \theta(x_1 + x_3) - y_1 \end{aligned} \tag{8.6}$$

$$\theta = \frac{y_2 - y_1}{x_2 - x_1} \text{ nếu } P \neq Q \tag{8.7}$$

$$\text{hoặc } \theta = \frac{3x_1^2 + a_4}{2y_1} \text{ nếu } P = Q \tag{8.8}$$

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

$$\theta \leftarrow \frac{y_2 - y_1}{x_2 - x_1}$$

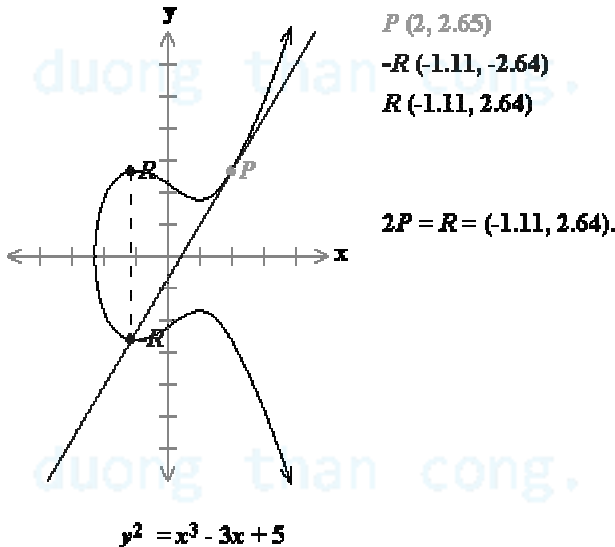
end if

$$x_3 = \theta^2 - x_1 - x_2$$

$$y_3 = \theta(x_1 + x_3) - y_1$$

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

$$y^2 = x^3 + ax + b \quad (8.9)$$

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:

$$p + 1 - 2\sqrt{p} \leq \#E(F_p) \leq p + 1 + 2\sqrt{p} \quad (8.10)$$

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:

$$\underbrace{A + A + \dots + A}_r = O \quad (8.11)$$

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:

$$y^2 + xy = x^3 + ax^2 + b \quad (8.12)$$

cùng với điểm O là điểm tại vô cực. Số lượng các điểm thuộc $E(F_{2^m})$ ký hiệu $\#E(F_{2^m})$ thỏa định lý Hasse:

$$q + 1 - 2\sqrt{q} \leq \#E(F_{2^m}) \leq q + 1 + 2\sqrt{q} \quad (8.13)$$

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 $P + Q = R = (x_3, y_3) \in E(F_{2^m})$.

$$\text{Nếu } P \neq Q \text{ thì } \begin{cases} \theta = \frac{y_1 + y_2}{x_1 + x_2} \\ x_3 = \theta^2 + \theta + x_1 + x_2 + a_2 \\ y_3 = (x_1 + x_3)\theta + x_3 + y_1 \end{cases} \quad (8.14)$$

$$\text{Nếu } P = Q \text{ thì } \begin{cases} \theta = \frac{y_1}{x_1} + x_1 \\ x_3 = \theta^2 + \theta + a_2 \\ y_3 = x_1^2 + (\theta + 1)x_3 \end{cases} \quad (8.15)$$

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**

$$\theta \leftarrow \frac{y_1}{x_1} + x_1 \text{ và } x_3 \leftarrow \theta^2 + \theta + a_2$$

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:

$$y^2z + xyz = x^3 + a^2x^2z^2 + a^6z^3 \quad (8.16)$$

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:

$$M(a, b) = M'(a, b, 1) \quad (8.17)$$

$$N(p, q, r) = N\left(\frac{p}{r}, \frac{q}{r}, 1\right) = N\left(\frac{p}{r}, \frac{q}{r}\right) \quad (8.18)$$

❖ 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:

$$\begin{aligned}x'_3 &= \frac{B^2}{A^2} + \frac{B}{A} + \frac{A}{z_1} + a_2 \\y'_3 &= \frac{B}{A} \left(\frac{x_1}{z_1} + x'_3 \right) + x'_3 + \frac{y_1}{z_1}\end{aligned}\tag{8.19}$$

Trong đó $A = (x_2 z_1 + x_1)$ và $B = (y_2 z_1 + y_1)$. Đặt $z_3 = A^3 z_1$ và $x_3 = x'_3 z_3$, $y_3 = y'_3 z_3$, nếu $P + Q = (x_3 : y_3 : z_3)$ thì:

$$\begin{aligned}x_3 &= AD, \\y_3 &= CD + A^2(Bx_1 + Ay_1) \\z_3 &= A^3 z_1\end{aligned}\tag{8.20}$$

với $C = A + B$ và $D = A^2(A + a_2 z_1) + z_1 BC$.

Tương tự $2P = (x_3 : y_3 : z_3)$ với

$$\begin{aligned}x_3 &= AB, \\y_3 &= x_1^4 A + B(x_1^2 + y_1 z_1 + A) \\z_3 &= A^3\end{aligned}\tag{8.21}$$

Trong đó $A = x_1 z_1$ và $B = a_6 z_1^4 + x_1^4$. 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 chiều. 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.

$$Q = c \times P = \underbrace{P + P + \dots + P}_c \tag{8.22}$$

Thuật toán 8.4: *Thuật toán nhân điểm trong hệ tọa độ chiều*

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 chiều: 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 ($2 \leq m \leq n-2$) 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 gửi và người nhận cần phải sở hữu một cặp khóa công cộng – khóa riêng. Các giá trị sau được quy ước chung giữa người gửi 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 gửi (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 $k \in [2, n-2]$.
- 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 gửi 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 gửi Q_A cho B
- B chọn một số ngẫu nhiên n . B tính giá trị $Q_B = g^n$ và gửi 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 gửi Q_A cho B
- B chọn một giá trị n ngẫu nhiên. B tính giá trị điểm $Q_B = n \times P$ và gửi 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 gửi 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

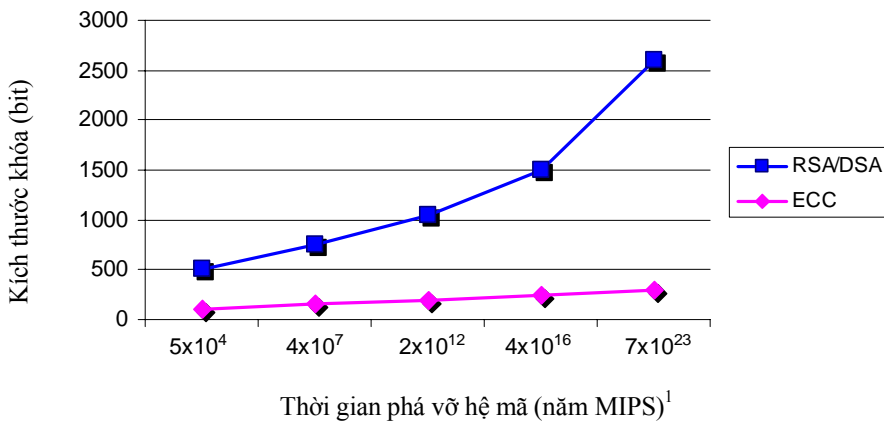
So sánh giữa các phương pháp mã hóa khóa công cộng

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 độ² [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, \dots, 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 $i = 1, 2, 3, \dots, s$
- 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 $|x| \bmod 2^{64}$.

$$X = x \parallel 1 \parallel 0^d \parallel l$$

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: $X = X[0] X[1] \dots X[N-1]$ 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.

$$F(X, Y, Z) = (X \wedge Y) \vee ((\neg X) \wedge Z)$$

$$G(X, Y, Z) = (X \wedge Z) \vee (Y \wedge (\neg Z))$$

$$H(X, Y, Z) = X \oplus Y \oplus Z$$

$$I(X, Y, Z) = Y \oplus (X \vee (\neg Z)) \quad (9.1)$$

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})

$X \lll s$ Các bit của X được dịch chuyển xoay vòng sang trái s vị trí ($0 \leq s < 32$)

Định nghĩa các hàm:

$FF(a, b, c, d, M_j, s, t_i) :$

$$a = b + ((a + F(b, c, d) + M_j + t_i) \lll s)$$

$GG(a, b, c, d, M_j, s, t_i) :$

$$a = b + ((a + G(b, c, d) + M_j + t_i) \lll s)$$

$HH(a, b, c, d, M_j, s, t_i) :$

$$a = b + ((a + H(b, c, d) + M_j + t_i) \lll s)$$

$II(a, b, c, d, M_j, s, t_i) :$

$$a = b + ((a + I(b, c, d) + M_j + t_i) \lll s)$$

với M_j là $M[j]$ và hằng số t_i xác định theo công thức:

$$t_i = \lfloor 2^{32} |\sin(i)| \rfloor, i \text{ tính bằng radian.}$$

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 ,22,0xc1bdceee)	GG(b,c,d,a,M0 ,20,0xe9b6c7aa)
FF(a,b,c,d,M4 , 7,0xf57c0faf)	GG(a,b,c,d,M5 , 5,0xd62f105d)
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,M15,14,0xd8a1e681)
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 , 5,0x21e1cde6)
FF(d,a,b,c,M9 ,12,0x8b44f7af)	GG(d,a,b,c,M14, 9,0xc33707d6)
FF(c,d,a,b,M10,17,0xfffff5bb1)	GG(c,d,a,b,M3 ,14,0xf4d50d87)
FF(b,c,d,a,M11,22,0x895cd7be)	GG(b,c,d,a,M8 ,20,0x455a14ed)
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,M10,15,0xffefff47d)
HH(b,c,d,a,M10,23,0xbebfbcb70)	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,M13,21,0x4e0811a1)
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,M15,16,0x1fa27cf8)	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 tỉ 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 $X[0], X[1], \dots, X[15]$ sẽ được mở rộng thành 80 từ-32 bit $W[0], W[1], \dots, W[79]$ theo công thức:

$$W[t] = \begin{cases} X[t], & 0 \leq t \leq 15 \\ X[j-3] \oplus X[j-8] \oplus X[j-14] \oplus X[j-16], & 16 \leq t \leq 79 \end{cases} \quad (9.2)$$

Trong phiên bản cải tiến của SHS, công thức trên được thay bằng:

$$W[t] = \begin{cases} X[t], & 0 \leq t \leq 15 \\ (X[j-3] \oplus X[j-8] \oplus X[j-14] \oplus X[j-16]) \lll 1, & 16 \leq t \leq 79 \end{cases} \quad (9.3)$$

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:

$$f[t](X, Y, Z) = \begin{cases} (X \wedge Y) \vee ((\neg X) \wedge Z), & 0 \leq t \leq 19 \\ X \oplus Y \oplus Z, & 20 \leq t \leq 39 \\ (X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z), & 40 \leq t \leq 59 \\ X \oplus Y \oplus Z, & 60 \leq t \leq 79 \end{cases} \quad (9.4)$$

$$K[t] = \begin{cases} 0x5a827999, & 0 \leq t \leq 19 \\ 0x6ed9eba1, & 20 \leq t \leq 39 \\ 0x8f1bbcd6, & 40 \leq t \leq 59 \\ 0xca62c1d6, & 60 \leq t \leq 79 \end{cases} \quad (9.5)$$

```
A = 0x67452301;
B = 0xefcdab89;
C = 0x98badcfe;
D = 0x10325476;
```

```

E = 0xc3d2elf0;
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 $M^{(1)}, M^{(2)}, \dots, M^{(N)}$. 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 $M^{(1)}, M^{(2)}, \dots, M^{(N)}$. 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  $63$ 

         $T_1 = h + \Sigma_1(e) + \text{Ch}(e, f, g) + K_t + W_t$ 

         $T_2 = \Sigma_0(a) + \text{Maj}(a, b, c)$ 

         $h = g$ 
         $g = f$ 
         $f = e$ 
         $e = d + T_1$ 
         $d = c$ 
         $c = b$ 
    
```

$$b = a$$

$$a = T_1 + T_2$$

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:

$$\text{Ch}(x, y, z) = (x \wedge y) \oplus (\neg x \wedge z)$$

$$\text{Maj}(x, y, z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)$$

$$\sum_0(x) = \text{ROTR}^2(x) \oplus \text{ROTR}^{13}(x) \oplus \text{ROTR}^{22}(x) \quad (9.6)$$

$$\sum_1(x) = \text{ROTR}^6(x) \oplus \text{ROTR}^{11}(x) \oplus \text{ROTR}^{25}(x)$$

$$\sigma_0(x) = \text{ROTR}^7(x) \oplus \text{ROTR}^{18}(x) \oplus \text{SHR}^3(x)$$

$$\sigma_1(x) = \text{ROTR}^{17}(x) \oplus \text{ROTR}^{19}(x) \oplus \text{SHR}^{10}(x)$$

Trong phương pháp SHA-384 và SHA-512, chúng ta cần sử dụng các hàm sau:

$$\begin{aligned}
 \text{Ch}(x, y, z) &= (x \wedge y) \oplus (\neg x \wedge z) \\
 \text{Maj}(x, y, z) &= (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) \\
 \sum_0(x) &= \text{ROTR}^{28}(x) \oplus \text{ROTR}^{34}(x) \oplus \text{ROTR}^{29}(x) \\
 \sum_1(x) &= \text{ROTR}^{14}(x) \oplus \text{ROTR}^{18}(x) \oplus \text{ROTR}^{41}(x) \\
 \sigma_0(x) &= \text{ROTR}^1(x) \oplus \text{ROTR}^8(x) \oplus \text{SHR}^7(x) \\
 \sigma_1(x) &= \text{ROTR}^{19}(x) \oplus \text{ROTR}^{61}(x) \oplus \text{SHR}^6(x)
 \end{aligned} \tag{9.7}$$

9.4.3 Nhận xét

Chuẩn SHS đặc tả 5 thuật toán băm an toàn SHA-1, SHA-224³, 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:

$$H_i = E_{X_i}(H_{i-1}) \oplus X_i \quad (9.8)$$

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 101000₂

Thông điệp mở rộng:

$$\underbrace{1110\ 1011\ 0010\ 0110\ 0011\ 0110\ 0111\ 1011\ 1001\ 1001}_{40\text{bit}} \underbrace{000\dots000}_{88\text{bit}} \underbrace{0\dots00101000}_{128\text{bit}}$$

- Chia thông điệp mở rộng thành n khối x_1, \dots, x_n , mỗi khối kích thước 256 bit.
- Áp dụng Davies-Mayer Hash bằng thuật toán Rijndael n lần cho n khối.

$$H_i = E_{x_i}(H_{i-1}) \oplus X_i \quad (9.9)$$

- Áp dụng thao tác bổ sung cuối để thu được giá trị băm.

$$H_{n+1} = E_{H_n}(H_n) \oplus H_n \quad (9.10)$$

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 độn 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 “độn 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 toàn⁵.

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.

cuu duong than cong. com

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.

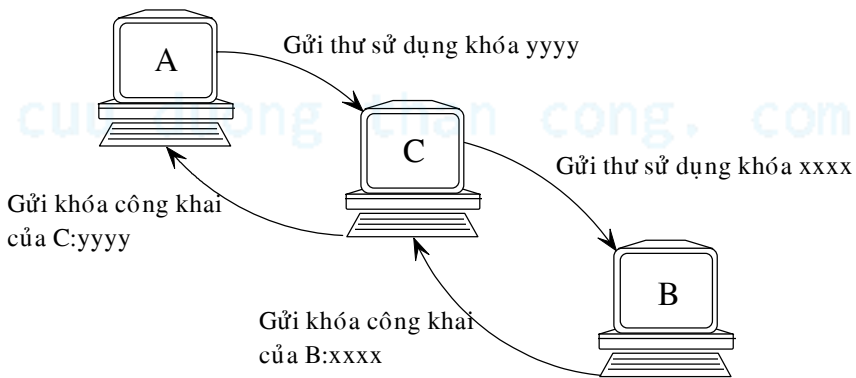
10.1 Giới thiệu

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

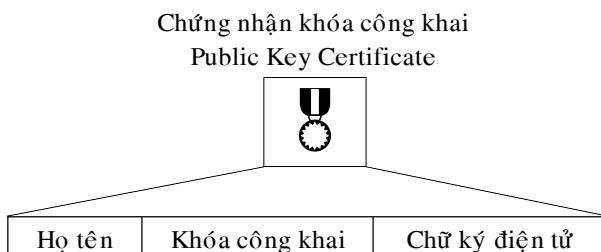
Trên thực tế, vấn đề này được giải quyết theo hai cách:

- 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 dùng 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.

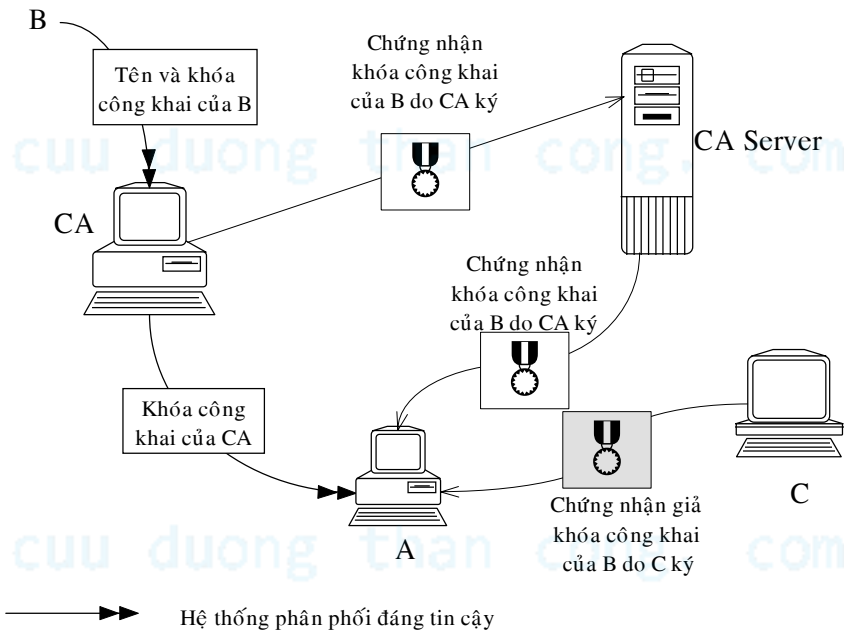


Hình 10.2. Các thành phần của một chứng nhận khóa công cộng

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.

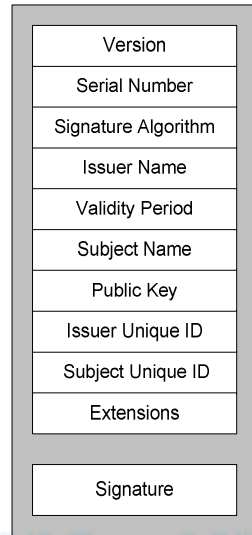
10.2.1 Chứng nhận X.509

Chứng nhận X.509 là chứng nhận khóa công cộng phổ biến nhất. Hiệp hội viễn thông quốc tế (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]:

- *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 Distinguished 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.



Hình 10.4. Phiên bản 3 của chuẩn chứng nhận X.509

- *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 tùy 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

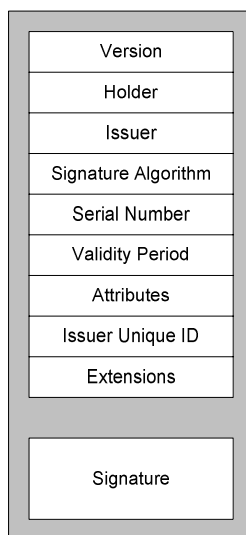
Các giấy chứng nhận thuộc tính (Attribute Certificates – AC [2]) là các giấy chứng nhận điện tử không chứa khóa công cộng. Thay vì thao tác chứng nhận khóa công cộng, ACs chỉ thao tác chứng nhận một tập hợp các thuộc tính.

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.

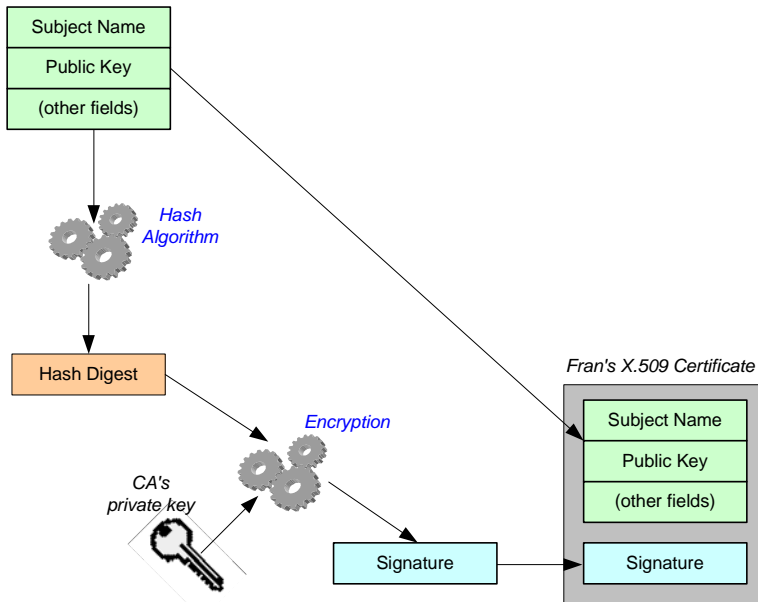
Động cơ chính cho việc sử dụng ACs là để cấp phép. Vì một người dùng có thể chỉ giữ một vai trò nào đó trong tổ chức trong một thời gian ngắn, nên khác với giấy chứng nhận khóa công cộng, AC chỉ có giá trị trong một vài ngày hoặc ngắn hơn.



Hình 10.5. Phiên bản 2 của cấu trúc chứng nhận thuộc tính

10.3 Sự chứng nhận và kiểm tra chữ ký

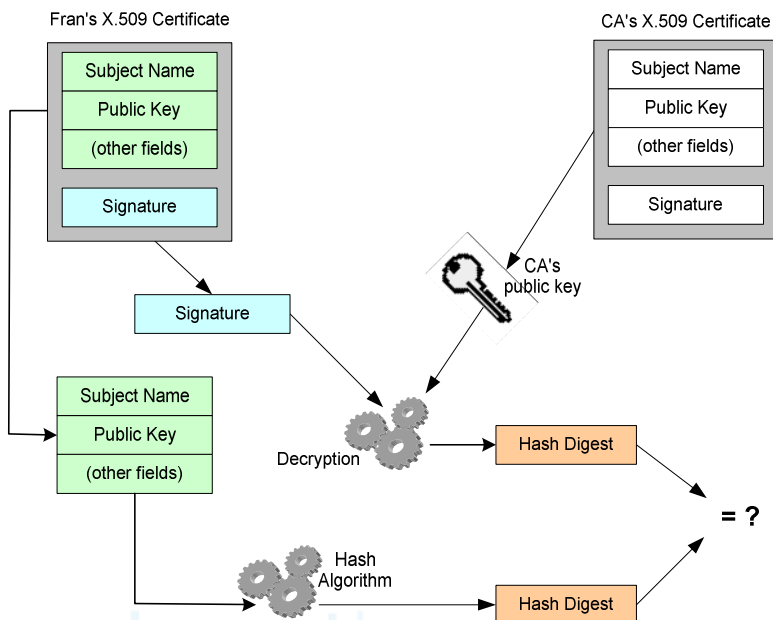
Quá trình chứng nhận chữ ký diễn ra theo hai bước. Đầu tiên, các trường của chứng nhận được ký và nén bởi thuật toán trộn cho trước. Sau đó, kết quả xuất của hàm trộn, được gọi là hash digest, được mã hóa với khóa bí mật của tổ chức CA đã phát hành chứng nhận này.



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.

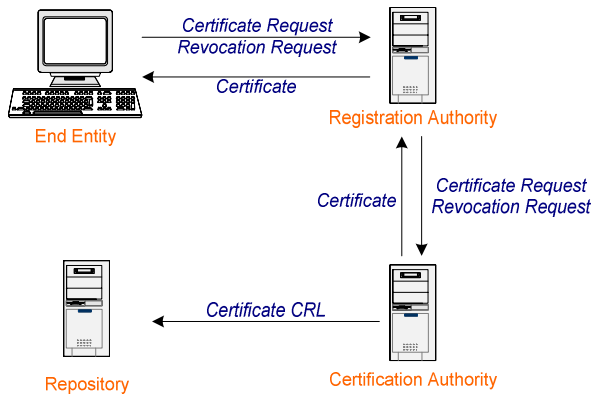
Để có một chứng nhận, một tổ chức CA chỉ cần tạo ra và ký giấy chứng nhận cho chính nó, chứ không cần áp dụng cho một CA khác để chứng nhận. Điều này được hiểu như sự tự chứng nhận (self-certification), và một giấy chứng như thế được gọi là giấy chứng nhận tự ký (self-signed certificate)



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 cơ 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 (Distinguished name- DN [2]) của mình.

10.5.2 Yêu cầu về giấy chứng nhận

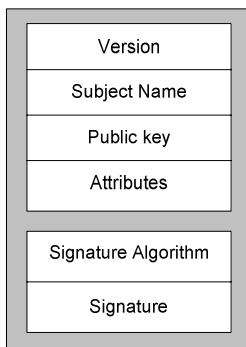
Đối tác có thể yêu cầu một chứng nhận từ CA thông qua nhiều kỹ thuật. Trong trường hợp phát sinh lại, đối tác không cần yêu cầu, tổ chức CA sẽ tạo ra một giấy chứng nhận thay cho đối tác. Kỹ thuật này yêu cầu tổ chức CA cũng phải phát sinh cặp khóa bất đối xứng để có được khóa công cộng được kèm theo 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]:

- *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.



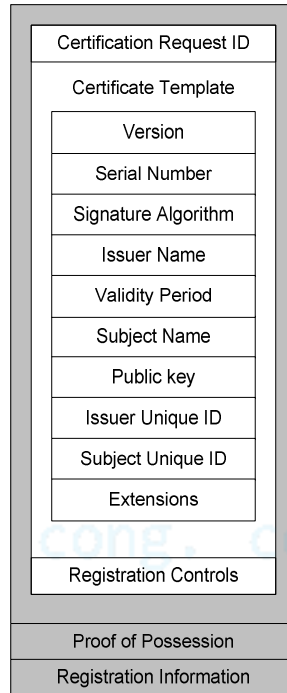
- *Public Key*: chỉ ra thuật toán của khóa công cộng, chứa khóa công cộng có định dạng tùy thuộc vào loại của nó.

Hình 10.9. Mẫu yêu cầu chứng nhận theo chuẩn PKCS#10

- *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 Possession* : 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 để dành 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.

Version
Signature Algorithm
Issuer Name
This Update
Next Update
Revoked Certificates
Serial Number
Revocation Date
CRL Entry Extensions
CRL Extensions
Signature

- *Revoked Certificates* : danh sách các chứng nhận bị hủy bỏ. Mỗi chứng nhận bị hủy bỏ 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.

Hình 10.11. Phiên bản 2 của định dạng danh sách chứng nhận bị hủy

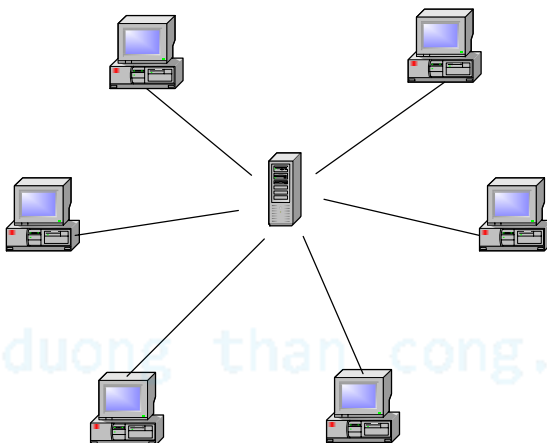
10.5.5 Lưu trữ và khôi phục khóa

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.

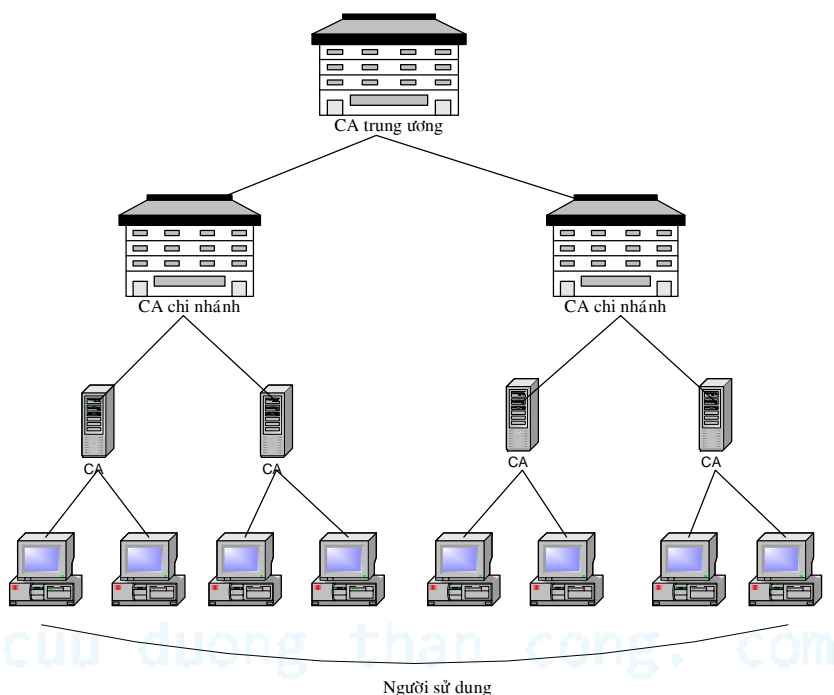
Khuyết điểm chính của mô hình này là hiện tượng “nút cổ chai” tại trung tâm [2].

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.



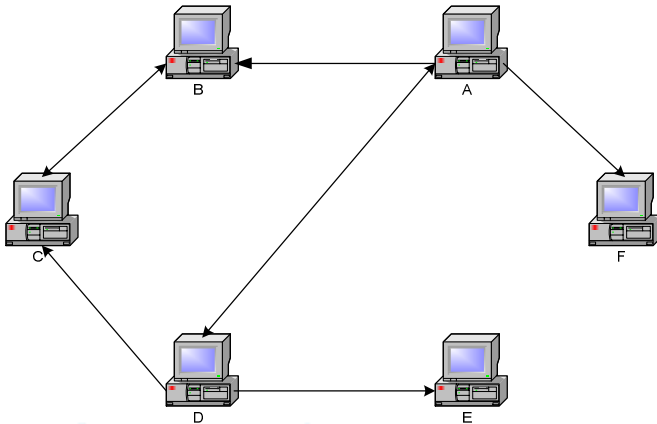
Hình 10.13. Mô hình CA phân cấp

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 Pretty 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 có 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.

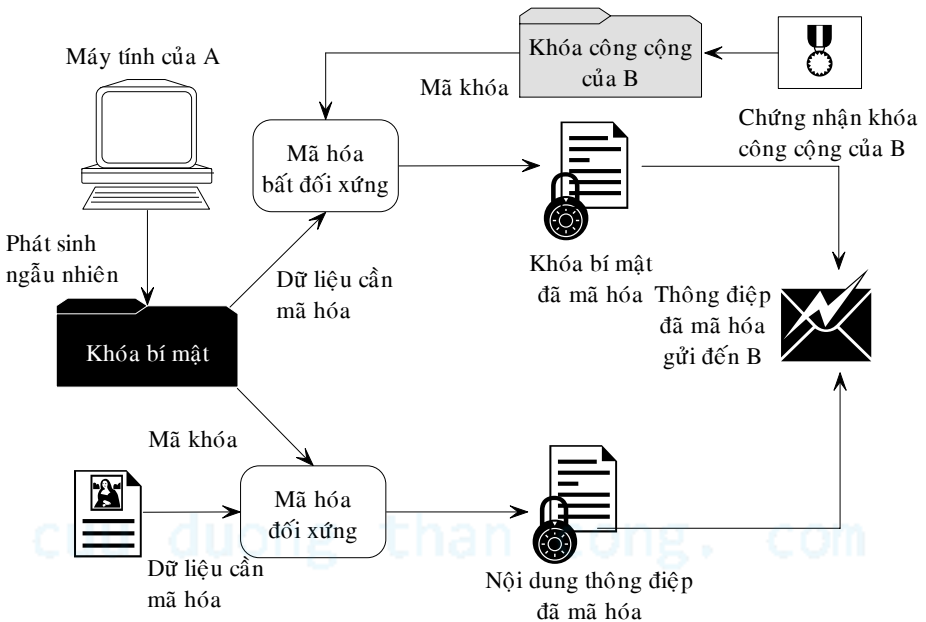
10.7 Ứng dụng “Hệ thống bảo vệ thư điện tử”

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



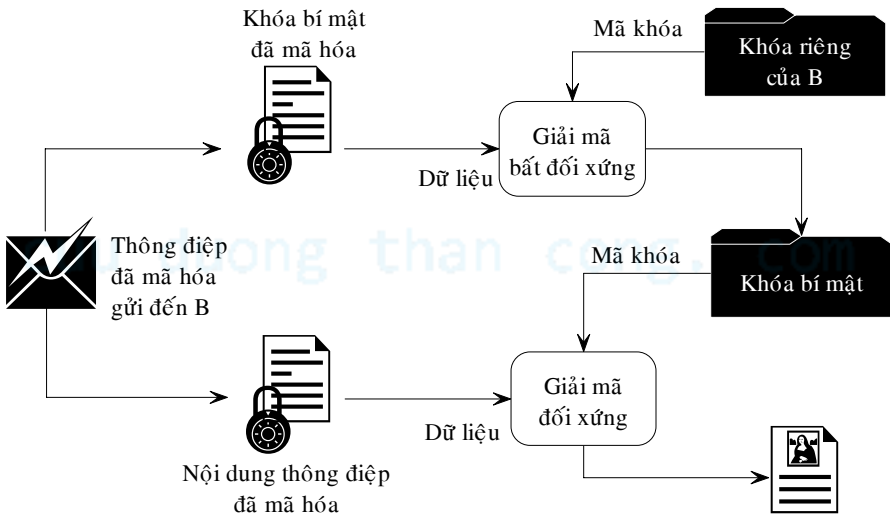
Hình 10.15. Quy trình mã hóa thư điện tử

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.

cuu duong than cong. com

cuu duong than cong. com

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, 0xf5407fb,
    0x6167d9a8, 0xd1f45763, 0x4daa96c3, 0x3bec5958,
    0xababa014, 0xb6ccd201, 0x38d6279f, 0x02682215,
    0x8f376cd5, 0x092c237e, 0xbfc56593, 0x32889d2c,
    0x854b3e95, 0x05bb9b43, 0x7dcd5dcd, 0xa0e926c,
    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, 0xfb10c53,
    0xcf3b870f, 0xb414935c, 0x664465ed, 0x024acac7,
    0x59a744c1, 0x1d2936a7, 0xdc580aa6, 0xcf574ca8,
    0x040a7a10, 0x6cd81807, 0xa98be4c, 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, 0x5b3fbdb6, 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,
0xdfa5272a, 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, 0xcda25ce,
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,
0x419cflad, 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, 0x2ddb49a, 0x6d5c5a54,
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, 0xae0347, 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, \dots, K_{79}

$$K_t = \begin{cases} 5a82799 & 0 \leq t \leq 19 \\ 6ed9eba1 & 20 \leq t \leq 39 \\ 8f1bbcdc & 40 \leq t \leq 59 \\ ca62c1d6 & 60 \leq t \leq 79 \end{cases}$$

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 $K_0^{\{256\}}, K_1^{\{256\}}, \dots, K_{63}^{\{256\}}$.

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 $K_0^{\{512\}}, K_1^{\{512\}}, \dots, K_{79}^{\{512\}}$. 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
5cb0a9dc bd41fbd4	76f988da831153b5
983e5152ee66dfab	a831c66d2db43210
b00327c898fb213f	bf597fc7beef0ee4
c6e00bf33da88fc2	d5a79147930aa725
06ca6351e003826f	142929670a0e6e70
27b70a8546d22ffc	2e1b21385c26c926
4d2c6dfc5ac42aed	53380d139d95b3df
650a73548baf63de	766a0abb3c77b2a8
81c2c92e47edae6	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
ca273ecee26619c	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:

$$H_0^{(0)} = 67452301$$

$$H_1^{(0)} = \text{efcdab89}$$

$$H_2^{(0)} = 98badcfe$$

$$H_3^{(0)} = 10325476$$

$$H_4^{(0)} = \text{c3d2e1f0}$$

SHA – 224:

$$H_0^{(0)} = c1059ed8$$

$$H_1^{(0)} = 367cd507$$

$$H_2^{(0)} = 3070dd17$$

$$H_3^{(0)} = f70e5939$$

$$H_4^{(0)} = ffc00b31$$

$$H_5^{(0)} = 68581511$$

$$H_6^{(0)} = 64f98fa7$$

$$H_7^{(0)} = befa4fa4$$

SHA – 256:

$$H_0^{(0)} = 6a09e667$$

$$H_1^{(0)} = bb67ae85$$

$$H_2^{(0)} = 3c6ef372$$

$$H_3^{(0)} = a54ff53a$$

$$H_4^{(0)} = 510e527f$$

$$H_5^{(0)} = 9b05688c$$

$$H_6^{(0)} = 1f83d9ab$$

$$H_7^{(0)} = 5be0cd19$$

SHA-384:

$$H_0^{(0)} = cbbb9d5dc1 \quad 059ed8$$

$$H_1^{(0)} = 629a292a36 \quad 7cd507$$

$$H_2^{(0)} = 9159015a30 \quad 70dd17$$

$$H_3^{(0)} = 152fec8f7 \quad 0e5939$$

$$H_4^{(0)} = 67332667ff \quad c00b31$$

$$H_5^{(0)} = 8eb44a8768 \quad 581511$$

$$H_6^{(0)} = db0c2e0d64 \quad f98fa7$$

$$H_7^{(0)} = 47b5481dbe \quad fa4fa4$$

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

cuu duong than cong. com

cuu duong than cong. com

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] Carolyn 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 Tuấn, 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. Helleseht, 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 "CryptoBytes 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. Stinson (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.

cuu duong than cong. com

cuu duong than cong. com