

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

BẢO MẬT THÔNG TIN

Biên Soạn:

TS. Văn Thiên Hoàng

KS. Bùi Văn Thiệu



* 1 . 2 0 1 8 . C M P 2 0 9 *

BẢO MẬT THÔNG TIN

Ấn bản 2018

Các ý kiến đóng góp về tài liệu học tập này, xin gửi về e-mail của ban biên tập:
tailieuhoctap@hutech.edu.vn

MỤC LỤC

MỤC LỤC.....	I
HƯỚNG DẪN	VI
BÀI 1: TỔNG QUAN VỀ BẢO MẬT THÔNG TIN	1
1.1 GIỚI THIỆU	1
1.2 BẢO VỆ THÔNG TIN TRONG QUÁ TRÌNH TRUYỀN THÔNG TIN TRÊN MẠNG	2
1.2.1 Các loại hình tấn công.....	2
1.2.2 Kiến trúc an ninh OSI	4
1.2.3 Mô hình mạng an toàn tổng quát	5
1.2.4 Vai trò của Mã hóa trong bảo mật thông tin trên mạng	6
1.2.5 Các giao thức thực hiện bảo mật	6
1.3 BẢO VỆ HỆ THỐNG KHỎI SỰ XÂM NHẬP PHÁ HOẠI TỪ BÊN NGOÀI	6
TÓM TẮT	8
CÂU HỎI ÔN TẬP	9
BÀI 2: MÃ HÓA ĐỔI XỨNG CỔ ĐIỂN	10
2.1 TỔNG QUAN VỀ HỆ MÃ ĐỔI XỨNG.....	10
2.1.1 Khái niệm mã hóa đối xứng	10
2.1.2 Các yêu cầu.....	11
2.1.3 Phá mã	11
2.1.4 Độ an toàn	13
2.2 CÁC HỆ MÃ THẾ.....	13
2.2.1 Hệ mã đẩy (Hệ mã CAESAR).....	13
2.2.2 Mã hóa thay thế đơn bảng	15
2.2.3 Hệ mã VIGENRE.....	17
2.2.4 Mã khóa tự động	18
2.2.5 Hệ mã độn một lần(ONE-TIME PAD, OTP)	18
2.3 CÁC HỆ MÃ KIỂU HOÁN VỊ	20
2.3.1 Hệ mã Rail Fence	20
2.3.2 Hệ mã hoán vị	20
2.3.3 Mã tích	21
2.4 NHẬN XÉT HỆ MÃ CỔ ĐIỂN.....	22
TÓM TẮT	22
CÂU HỎI ÔN TẬP	22
BÀI 3: MÃ HÓA ĐỔI XỨNG HIỆN ĐẠI.....	24
3.1 MÃ DÒNG (STREAM CIPHER)	24
3.1.1 Tiny RC4	25
3.1.2 RC4.....	27
3.2 MÃ KHỐI (BLOCK CIPHER)	28
3.2.1 Mạng SPN	28

3.2.2 Mô Hình Mã Feistel	29
3.3 MÃ DES.....	31
3.3.1 Hoán Vị Khởi Tạo Và Hoán Vị Kết Thúc	32
3.3.2 Các Vòng Của DES	32
3.3.3 Thuật Toán Sinh Khóa Con Của Des	35
3.3.4 Hiệu Ứng Lan Truyền (Avalanche Effect).....	36
3.3.5 Phá mã Des.....	36
3.4 MỘT SỐ PHƯƠNG PHÁP MÃ KHỐI KHÁC.....	37
3.4.1 Triple Des	37
3.4.2 Advanced Encryption Standard (AES).....	37
3.5 CÁC MÔ HÌNH ỨNG DỤNG MÃ KHỐI	38
3.5.1 Electronic Codebook – ECB	38
3.5.2 Cipher Block Chaining – CBC.....	39
3.5.3 Cipher Feedback- CFB.....	40
3.5.4 Output Feedback – OFB	41
3.5.5 Counter – CTR	42
3.6 BỐ TRÍ CÔNG CỤ MÃ HÓA.....	42
3.6.1 Mã hóa liên kết.....	43
3.6.2 Mã hóa đầu cuối.....	43
3.7 QUẢN LÝ TRAO ĐỔI KHÓA BÍ MẬT	43
3.7.1 Các cách trao đổi khóa.....	43
3.7.2 Phương pháp trao đổi khóa bằng trung tâm phân phối khóa	44
TÓM TẮT.....	45
CÂU HỎI ÔN TẬP	47
BÀI 4: MÃ HÓA KHÓA CÔNG KHAI RSA	48
4.1 LÝ THUYẾT SỐ	48
4.1.1 Một Số Khái Niệm	48
4.1.2 Định Lý Fermat.....	50
4.1.3 Phép Logarit Rời rạc	50
4.2 MÃ KHÓA CÔNG KHAI	52
4.2.1 Ứng dụng khóa công khai.....	53
4.2.2 Tính an toàn của các sơ đồ khóa công khai	54
4.3 RSA	55
4.3.1 Khởi tạo khóa RSA	55
4.3.2 Sử dụng RSA.....	56
4.3.3 Cơ sở của RSA	56
4.3.4 Ví dụ	56
4.3.5 Sinh khóa RSA.....	57
4.3.6 Độ an toàn của RSA.....	57
TÓM TẮT.....	59
CÂU HỎI ÔN TẬP	60

BÀI 5: QUẢN LÝ KHÓA DÙNG MÃ KHÓA CÔNG KHAI.....	61
5.1 PHÂN PHỐI KHÓA.....	61
<i>5.1.1 Phân phối khóa công khai.....</i>	<i>61</i>
5.2 PHÂN PHỐI CÔNG KHAI KHÓA MẬT	64
<i>5.2.1 Trao đổi khóa hỗn hợp</i>	<i>64</i>
5.3 TRAO ĐỔI KHÓA DIFFIE HELLMAN	65
<i>5.3.1 Khởi tạo Diffie Hellman.....</i>	<i>65</i>
<i>5.3.2 Trao đổi khóa Diffie Hellman</i>	<i>66</i>
<i>5.3.3 Ví dụ</i>	<i>66</i>
5.4 CHỐNG TÂN CÔNG PHÁT LẠI THÔNG ĐIỆP (REPLAY ATTACK) KHI ỨNG DỤNG .	66
TÓM TẮT	68
CÂU HỎI ÔN TẬP	69
BÀI 6: MÃ CHỨNG THỰC THÔNG ĐIỆP, HÀM BĂM.....	70
6.1 XÁC THỰC MẪU TIN	70
<i>6.1.1 Các yêu cầu bảo mật khi truyền mẫu tin trên mạng.....</i>	<i>70</i>
<i>6.1.2 Khái niệm xác thực mẫu tin</i>	<i>71</i>
<i>6.1.3 Mã mẫu tin.....</i>	<i>71</i>
6.2 MÃ XÁC THỰC MẪU TIN.....	72
<i>6.2.1 Các tính chất của MAC</i>	<i>72</i>
<i>6.2.2 Yêu cầu đối với MAC</i>	<i>73</i>
<i>6.2.3 Sử dụng mã đối xứng cho MAC.....</i>	<i>73</i>
<i>6.2.4 Mô hình áp dụng</i>	<i>73</i>
6.3 HÀM BĂM (HASH FUCTION)	74
<i>6.3.1 Yêu cầu đối với hàm băm</i>	<i>74</i>
<i>6.3.2 Bài toán Ngày Sinh nhật.....</i>	<i>75</i>
<i>6.3.3 Hàm Băm MD5.....</i>	<i>77</i>
<i>6.3.4 SHA-1</i>	<i>81</i>
<i>6.3.5 HMAC</i>	<i>85</i>
6.4 MỘT SỐ ỨNG DỤNG CỦA HÀM BĂM	85
<i>6.4.1 Các kiểu giao thức xử lý dữ liệu ứng dụng hàm băm</i>	<i>85</i>
<i>6.4.2 Lưu trữ Mật khẩu</i>	<i>87</i>
<i>6.4.3 Đấu Giá Trực Tuyến.....</i>	<i>88</i>
<i>6.4.4 Dowload File.....</i>	<i>89</i>
TÓM TẮT	90
CÂU HỎI ÔN TẬP	91
BÀI 7: BẢO MẬT MẠNG NỘI BỘ VÀ AN TOÀN IP	92
7.1 BẢO MẬT MẠNG CỤC BỘ - KERBEROS.....	92
<i>7.1.1 KB4.....</i>	<i>92</i>
<i>7.1.2 KB5.....</i>	<i>93</i>
<i>7.1.3 Chi tiết mô hình Kerberos.....</i>	<i>93</i>
<i>7.1.4 Hạn chế của Kerberos.....</i>	<i>97</i>

7.2 AN TOÀN IP - IPSEC	97
7.2.1 Các ứng dụng của IPSec.....	98
7.2.2 Lợi ích	98
7.2.3 Công cụ mã hóa.....	98
7.2.4 Kiến trúc xử lý tổng quát.....	99
7.2.5 Bộ thỏa thuận an ninh – SA.....	99
7.2.6 Trao đổi thông số bảo mật IKE.....	100
7.2.7 Giao thức xử lý dữ liệu AH.....	106
7.2.8 Giao thức xử lý dữ liệu ESP.....	107
7.2.9 Các chế độ IPSec	108
TÓM TẮT.....	111
CÂU HỎI ÔN TẬP	112
BÀI 8: BẢO MẬT WEB VÀ MAIL.....	113
8.1 DỊCH VỤ XÁC THỰC X.509	113
8.1.1 Phân Cấp Chứng Thực.....	116
8.1.2 Các Định Dạng File Của Chứng Chỉ X.509.....	118
8.2 GIAO THỨC BẢO MẬT WEB SSL.....	119
8.2.1 Kiến trúc tổng thể	120
8.2.2 Thủ tục thay đổi đặc tả mã SSL (SSL Change Cipher Spec Protocol)	121
8.2.3 Thủ tục nhắc nhở SSL (SSL Alert Protocol).....	121
8.2.4 Giao Thức Bắt Tay	121
8.2.5 Giao Thức Truyền Số Liệu-SSL Record Protocol.....	125
8.3 BẢO MẬT MAIL PGP	127
8.3.1 Xác thực trong PGP	127
8.3.2 Bảo mật PGP	128
8.3.3 Bảo mật và xác thực trong PGP	128
8.3.4 Thao tác PGP – nén	129
8.3.5 Thao tác PGP – tương thích thư điện tử	129
8.3.6 Khóa riêng và công khai của PGP.....	130
8.3.7 Các chùm khóa PGP.....	130
8.3.8 Tạo mẫu tin PGP	131
8.3.9 Nhận mẫu tin PGP	131
8.3.10 Quản lý khóa PGP	132
TÓM TẮT.....	132
CÂU HỎI ÔN TẬP	133
BÀI 9: ỨNG DỤNG KIỂM SOÁT TRUY CẬP	134
9.1 TỔNG QUAN VỀ KIẾN TRÚC AAA	134
9.2 TACACS+	136
9.2.1 Khuôn dạng gói tin Tacacs+.....	137
9.2.2 Cơ chế mã hóa trong Tacacs+.....	138
9.2.3 Cơ Chế Hoạt Động	139

9.2.4 Authentication	140
9.2.5 Authorization	141
9.2.6 Accounting	142
9.3 RADIUS	143
9.3.1 Nguyên lý hoạt động.....	143
9.3.2 Cấu trúc gói tin RADIUS	145
9.3.3 Authentication, Authorization	148
9.3.4 Accouting	149
TÓM TẮT	151
CÂU HỎI ÔN TẬP	152
PHỤ LỤC	153
TÀI LIỆU THAM KHẢO	246

HƯỚNG DẪN

MÔ TẢ MÔN HỌC

Do các ứng dụng trên mạng Internet ngày càng phát triển và mở rộng, nên an toàn thông tin trên mạng đã trở thành nhu cầu bắt buộc cho mọi hệ thống ứng dụng. Vì vậy, Bảo mật thông tin là môn học không thể thiếu của sinh viên ngành mạng. Mục đích của môn học này là cung cấp cho sinh viên kiến thức về các vấn đề an ninh mạng hiện nay và giải pháp tổng thể trong việc triển khai mạng an toàn.

NỘI DUNG MÔN HỌC

- **Bài 1:** Sự phát triển mạnh mẽ của công nghệ thông tin và đặc biệt là internet, thông tin được lưu trữ trên máy tính và được truyền trên mạng internet, do đó xuất hiện nhu cầu an toàn và bảo mật thông tin trên máy tính. Bài học giới thiệu tổng quan về các loại tấn công mạng: xem trộm thông tin, thay đổi thông điệp, mạo danh, gửi lại thông điệp.... Nhu cầu bảo vệ dữ liệu là rất quan trọng. Để đảm bảo cho 1 hệ truyền tin trên mạng 1 cách an toàn, tiêu chuẩn x800 đặt ra 5 dịch vụ bảo mật: xác thực, quyền truy cập, bảo mật, toàn vẹn dữ liệu, và chống chối bỏ. Lý thuyết mật mã là 1 công cụ thiết yếu để bảo mật thông tin.
- **Bài 2:** Mật mã đối xứng sử dụng cùng một khóa cho việc mã hóa và giải mã. Các phương pháp mã hóa cổ điển thường dựa trên hai phương thức. Cách thứ nhất là dùng phương thức thay thế một chữ cái trong bản rõ thành một chữ cái khác trong bản mã. Các mã hóa dùng phương thức này là mã hóa Ceasar, mã hóa thay thế đơn bảng, đa bảng, độn một lần. Cách thứ hai là dùng phương thức hoán vị để thay đổi thứ tự ban đầu của các chữ cái trong bản rõ. Hai phương thức này cũng đóng vai trò quan trọng trong mã hóa đối xứng hiện đại được trình bày trong chương tiếp theo.
- **Bài 3:** Bài học cung cấp cho sinh viên kiến thức về hệ mã dòng RC4, hệ mã khối DES, các mô hình áp dụng các hệ mã này, cách bố trí các ứng dụng mã hóa và giải pháp trao đổi khóa mật. Mã dòng có các đặc tính sau: Kích thước một đơn vị mã hóa gồm k bit. Bản rõ được chia thành các đơn vị mã hóa k bit. Một bộ sinh dãy số

ngẫu nhiên dùng khóa K ban đầu để sinh ra các số ngẫu nhiên có kích thước bằng kích thước đơn vị mã hóa. Mỗi số ngẫu nhiên được XOR với đơn vị mã hóa của bản rõ để có bản mã. Mạng S-P là giải pháp mã hóa khối. Việc kết hợp các S-box và P-box tạo ra hai tính chất quan trọng của mã hóa là tính khuếch tán (diffusion) và tính gây lẫn (confusion). Mô hình mã Feistel là một dạng tiếp cận khác so với mạng SP. Mô hình do Horst Feistel đề xuất, cũng là sự kết hợp các phép thay thế và hoán vị. Trong hệ mã Feistel, bản rõ sẽ được biến đổi qua một số vòng để cho ra bản mã cuối cùng. Mã DES (DATA ENCRYPTION STANDARD) là hệ mã thuộc hệ mã Feistel gồm 16 vòng, ngoài ra DES có thêm một hoán vị khởi tạo trước khi vào vòng 1 và một hoán vị khởi tạo sau vòng 16. Kích thước của khối là 64 bit. Kích thước khóa là 56 bit. Mỗi vòng của DES dùng khóa con có kích thước 48 bit được trích ra từ khóa chính. Mã khối (như mã DES) được áp dụng để mã hóa một khối dữ liệu có kích thước xác định. Để mã hóa một bản tin dài, bản tin được chia ra thành nhiều khối ($P = P_0P_1\dots P_{n-1}$) và áp dụng mã khối cho từng khối một. Có nhiều mô hình áp dụng mã khối là ECB, CBC, CTR, OFB và CFB. Có 2 phương án cơ bản: Mã hóa liên kết và Mã hóa đầu cuối. Cách phân phối khóa hiệu quả là mỗi bên A và B đều có một kênh mã hóa đến một bên thứ ba C thì C có thể gửi khóa theo các kênh mã hóa đó đến A và B.

- **Bài 4:** Bài học cung cấp cho sinh viên kiến thức về lý thuyết toán học cơ bản hỗ trợ cho lý thuyết mã hóa, hệ mã khóa bất đối xứng và hệ mã RSA. Hai khái niệm toán học quan trọng hỗ trợ mã hóa là phép toán đồng dư và định lý Fermat. Khóa công khai ra đời hỗ trợ thêm để giải quyết một số bài toán an toàn, chứ không phải thay thế khóa riêng. Cả hai khóa cùng tồn tại, phát triển và bổ sung cho nhau. Hệ mã khóa công khai hay bất đối xứng bao gồm việc sử dụng 2 khóa: Khóa công khai, mà mọi người đều biết, được dùng để mã hóa mẫu tin và kiểm chứng chữ ký, Khóa riêng, chỉ người nhận biết, để giải mã bản tin hoặc để tạo chữ ký. RSA là mã công khai được biết đến nhiều nhất và sử dụng rộng rãi nhất hiện nay. Nó dựa trên các phép toán lũy thừa trong trường hữu hạn các số nguyên theo modulo nguyên tố. Cụ thể, mã hóa hay giải mã là các phép toán lũy thừa theo modulo số rất lớn. Việc thám mã, tức là tìm khóa riêng khi biết khóa công khai, dựa trên bài toán khó là phân tích một số rất lớn đó ra thừa số nguyên tố.

- **Bài 5:** Bài học cung cấp kiến thức về: Mã khóa công khai giúp giải bài toán phân phôi khóa, Nó bao gồm hai khía cạnh sau: Phân phôi khóa một cách công khai nhưng đảm bảo được bí mật. Sử dụng mã khóa công khai để phân phôi khóa mật (còn khóa mật dùng để mã hóa thông tin). Việc phân phôi khóa công khai có 4 giải pháp: (1) Thông báo công khai khóa của người sử dụng, (2) Thư mục truy cập công cộng cho mọi người, (3) Chủ quyền khóa công khai, người nắm giữ khóa công khai và (4) Chứng nhận khóa công khai, khóa công khai của người sử dụng được nơi có thẩm quyền chứng nhận. Phân phôi khóa mật đơn giản sử dụng khóa công khai được đề xuất bởi Merkle vào năm 1979: (1) A tạo ra một cặp khóa công khai mới tạm thời, (2) A gửi B một khóa công khai và danh tính của họ, (3) B tạo ra khóa phiên và gửi nó cho A sử dụng khóa công khai được cung cấp và (4) A giải mã khóa phiên và cả hai cùng dùng nó. Trao đổi khóa Diffie Hellman là sơ đồ khóa công khai đầu tiên được đề xuất bởi Diffie và Hellman năm 1976 cùng với khái niệm khóa công khai. Phương pháp này dùng để thiết lập khóa chung chỉ có hai đối tác biết đến. Giá trị khóa phụ thuộc vào các đối tác (và các thông tin về khóa công khai và khóa riêng của họ). Độ an toàn dựa trên độ khó của bài toán tính logarit rời rạc (giống bài toán phân tích ra thừa số là bài toán khó).
- **Bài 6:** Bài học cung cấp kiến thức về vấn đề xác thực mẫu tin sử dụng mã xác thực, hàm băm và các ứng dụng của nó. Xác thực mẫu tin nhằm bảo vệ tính toàn vẹn của mẫu tin. Bảo vệ mẫu tin không bị thay đổi hoặc có các biện pháp phát hiện nếu mẫu tin bị thay đổi trên đường truyền. Đồng thời, xác thực mẫu tin sẽ giúp kiểm chứng danh tính và nguồn gốc. Từ đó giải quyết bài toán chống chối từ bản gốc. Mã xác thực mẫu (MAC – Message Authentication Code) tin sinh ra bởi một thuật toán mà tạo ra một khối thông tin nhỏ có kích thước cố định. MAC phụ thuộc vào cả mẫu tin và khóa nào đó. MAC bổ sung vào mẫu tin như chữ ký để gửi kèm theo làm bằng chứng xác thực. Người nhận thực tạo MAC trên mẫu tin nhận được và kiểm tra xem nó có phù hợp với MAC đính kèm không. Thuật toán xác thực dữ liệu (DAA – Data Authentication Algorithm) là MAC được sử dụng rộng rãi dựa trên chế độ DES-CBC. Hàm băm tạo ra một giá trị băm có kích thước cố định từ thông báo đầu vào mà không dùng khóa: $h = H(M)$. Hàm băm không cần giữ bí mật. Giá trị băm gắn kèm với thông báo dùng để kiểm tra tính toàn vẹn của thông báo. Bất kỳ sự thay đổi M nào dù nhỏ cũng tạo ra một giá trị h khác. MD5

(Message Digest) được phát minh bởi Ron Rivest. Kích thước giá trị băm của MD5 là 128 bít. Vì MD5 không còn được xem là an toàn, nên người ta đã xây dựng thuật toán băm khác. Mỹ chọn làm chuẩn quốc gia. SHA-1 có kích thước giá trị băm là 160 bít.

- **Bài 7:** Bài học cung cấp kiến thức về giao thức ứng dụng bảo mật trong mạng nội bộ Kerberos và giao thức an toàn IP - IPSEC. Kerberos là một giao thức xác thực mạng, nó cho phép các cá nhân giao tiếp với nhau trên một mạng không an toàn bằng cách xác thực người dùng theo một cơ chế bảo mật và an toàn. Kerberos ngăn chặn việc nghe trộm thông tin cũng như tấn công thay thế và đảm bảo tính toàn vẹn của dữ liệu. Kerberos hoạt động theo mô hình máy trạm/máy chủ và nó thực hiện quá trình xác thực 2 chiều - cả người dùng và dịch vụ xác thực lẫn nhau. Kerberos được xây dựng dựa trên mô hình mã hóa khóa đối xứng và đòi hỏi một thành phần thứ ba tin cậy tham gia vào quá trình xác thực. IPsec (Internet Protocol Security) thực hiện mã hóa và xác thực ở lớp mạng. Nó cung cấp một giải pháp an toàn dữ liệu đầu cuối cũng như liên kết mạng. Các gói mã hóa có khuôn dạng giống như gói tin IP thông thường, nên chúng dễ dàng được định tuyến qua mạng Internet mà không phải thay đổi các thiết bị mạng trung gian, qua đó cho phép giảm đáng kể các chi phí cho việc triển khai và quản trị. IPsec cung cấp bốn chức năng quan trọng sau: Bảo mật - Confidentiality, Toàn vẹn dữ liệu- Data integrity, Xác thực- Authentication, và Antireplay protection (xác nhận mỗi gói tin là duy nhất và không trùng lặp).
- **Bài 8:** Bài học cung cấp kiến thức về chứng minh thư X.509, bảo mật web SSL và bảo mật mail PGP. Dịch vụ xác thực X.509 là một phần của chuẩn dịch vụ thư mục CCITT X.500. Ở đây, các máy chủ phân tán bảo trì cơ sở dữ liệu thông tin của người sử dụng và xác định khung cho các dịch vụ xác thực. Thư mục chứa các chứng nhận khóa công khai, khóa công khai của người sử dụng được ký bởi chủ quyền chứng nhận. Vì chứng chỉ được ký bằng khóa riêng của CA, nên bảo đảm rằng chữ ký không thể bị làm giả và bất cứ ai tin tưởng vào khóa công khai của CA thì có thể tin tưởng vào chứng chỉ mà CA đó cấp phát. Giao thức SSL (Secure Socket Layer) bảo mật dữ liệu trao đổi qua socket. Đây là giao thức bảo mật kết hợp mã hóa khóa công khai và khóa đối xứng trong đó mã hóa RSA được dùng để trao đổi khóa phiên của mã hóa đối xứng. Kiến trúc xử lý của SSL gồm có bốn loại

xử lý: (1) SSL Record Protocol: Xử lý dữ liệu, (2) SSL Change Cipher Spec: một message đơn 1 byte, cập nhật lại bộ mã hóa để sử dụng trên kết nối này, (3) SSL Alert được dùng để truyền cảnh báo liên kết SSL với đầu cuối bên kia, và (4) SSL Handshake Protocol: Giao thức này cho phép server và client chứng thực với nhau và thương lượng cơ chế mã hóa, thuật toán MAC và khóa mật mã được sử dụng để bảo vệ dữ liệu được gửi trong SSL record. PGP (Pretty Good Privacy) là một dịch vụ về bảo mật và xác thực được sử dụng rộng rãi cho chuẩn an toàn thư điện tử. PGP được phát triển bởi Phil Zimmermann.

- **Bài 9:** Bài học cung cấp kiến thức về mô hình kiểm soát truy cập mạng an toàn AAA và các phần mềm ứng dụng hiện có TACACS+, RADIUS. AAA cung cấp việc xác thực (authentication) người dùng nhằm bảo đảm có thể nhận dạng đúng người dùng. Một khi đã nhận dạng người dùng, ta có thể giới hạn thẩm quyền (authorization) mà người dùng có thể làm. Khi người dùng sử dụng mạng, ta cũng có thể giám sát tất cả những gì mà họ làm AAA có thể dùng để tập hợp thông tin từ nhiều thiết bị trên mạng. TACACS+ (Terminal Access controller Access- Control System Plus) là một giao thức độc quyền của Cisco, được thiết kế dùng trong kiến trúc AAA cho việc chứng thực, ủy quyền và kẽ toán trong môi trường mạng. Ba hoạt động có thể được thực hiện trong suốt quá trình hoạt động của TACACS+:
Hoạt động đầu tiên được thực hiện là xác thực để nhận diện người dùng. Hoạt động thứ 2 là ủy quyền và có thể chỉ thực hiện một lần khi user đã được nhận dạng hoàn tất. Hoạt động thứ 3 là kiểm toán. Quá trình kiểm toán theo dõi dấu vết của các hành động mà người dùng đã thực hiện khi ở trong hệ thống. Radius (Remote Access Dial In User Service) là một giao thức mạng cung cấp việc quản lý xác thực tập trung, ủy quyền, và kẽ toán (AAA) để cho các máy tính kết nối vào mạng và sử dụng dịch vụ mạng.

KIẾN THỨC TIỀN ĐỀ

Môn học bảo mật thông tin sinh viên cần có kiến thức các môn học cơ bản như mạng cơ bản, lập trình hướng đối tượng Java.

YÊU CẦU MÔN HỌC

Người học phải dự học đầy đủ các buổi lên lớp và làm bài tập đầy đủ ở nhà.

CÁCH TIẾP NHẬN NỘI DUNG MÔN HỌC

Người học đọc tài liệu giáo trình bài giảng, làm các bài tập cuối mỗi bài học, thực hành trên máy tính viết chương trình minh họa từng thuật toán bảo mật, xây dựng một chương trình ứng dụng minh họa đầy đủ các dịch vụ bảo mật và cài đặt, phân tích tính bảo mật của một số phần mềm ứng dụng đã có: SSL, PGP, Kerberos, IPSEC, RADIUS, TACCAS+.

PHƯƠNG PHÁP ĐÁNH GIÁ MÔN HỌC

Môn học được đánh giá gồm:

- Điểm thực hành (30%): Điểm học thực hành trên phòng máy. Hình thức và nội dung được đề cập trong giáo trình thực hành bảo mật thông tin.
- Điểm quá trình (20%): Hình thức và cách đánh giá do giảng viên dạy lý thuyết quyết định được phê duyệt của bộ môn.
- Điểm thi cuối kỳ (50%): Người học sẽ được phân công đồ án dưới sự hướng dẫn của Giảng viên phụ trách môn học và vấn đáp nội dung của đồ án kết hợp với nội dung lý thuyết trong các bài học.

BÀI 1: TỔNG QUAN VỀ BẢO MẬT THÔNG TIN

Sau khi học xong bài này, sinh viên hiểu:

- *Tại sao cần bảo mật thông tin*
- *Các kiểu tấn công mạng xảy ra như thế nào*
- *Giải pháp bảo mật mạng được các nhà nghiên cứu chuẩn hóa, đề nghị là gì*
- *Vai trò của lý thuyết mật mã trong bảo mật thông tin*

1.1 GIỚI THIỆU

Trước đây khi công nghệ máy tính chưa phát triển, khi nói đến vấn đề bảo mật thông tin (*Information Security*), chúng ta thường hay nghĩ đến các biện pháp nhằm đảm bảo cho thông tin được trao đổi hay cất giữ một cách an toàn và bí mật. Chẳng hạn là các biện pháp như:

- Đóng dấu và ký niêm phong một bức thư để biết rằng lá thư có được chuyển nguyên vẹn đến người nhận hay không.
- Dùng mật mã mã hóa thông điệp để chỉ có người gửi và người nhận hiểu được thông điệp. Phương pháp này thường được sử dụng trong chính trị và quân sự.
- Lưu giữ tài liệu mật trong các két sắt có khóa, tại các nơi được bảo vệ nghiêm ngặt, chỉ có những người được cấp quyền mới có thể xem tài liệu.

Với sự phát triển mạnh mẽ của công nghệ thông tin, đặc biệt là sự phát triển của mạng Internet, ngày càng có nhiều thông tin được lưu giữ trên máy tính và gửi đi trên mạng Internet. Do đó, nhu cầu về an toàn và bảo mật thông tin trên máy tính xuất hiện. Có thể phân loại mô hình an toàn bảo mật thông tin trên máy tính theo hai hướng chính như sau:

- Bảo vệ thông tin trong quá trình truyền thông tin trên mạng (*Network Security*)
- Bảo vệ hệ thống máy tính, và mạng máy tính, khỏi sự xâm nhập phá hoại từ bên ngoài (*System Security*)

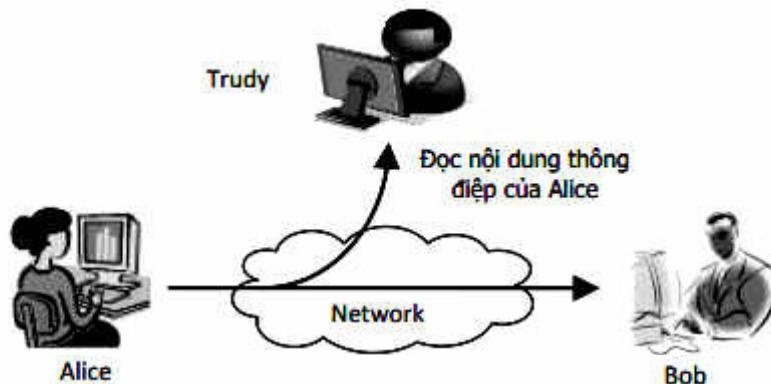
1.2 BẢO VỆ THÔNG TIN TRONG QUÁ TRÌNH TRUYỀN THÔNG TIN TRÊN MẠNG

1.2.1 Các loại hình tấn công

Để xem xét những vấn đề bảo mật liên quan đến truyền thông trên mạng, chúng ta hãy lấy một bối cảnh sau: có ba nhân vật tên là Alice, Bob và Trudy, trong đó Alice và Bob thực hiện trao đổi thông tin với nhau, còn Trudy là kẻ xấu, đặt thiết bị can thiệp vào kênh truyền tin giữa Alice và Bob. Sau đây là các loại hành động tấn công của Trudy mà ảnh hưởng đến quá trình truyền tin giữa Alice và Bob:

Xem trộm thông tin (Release of Message Content)

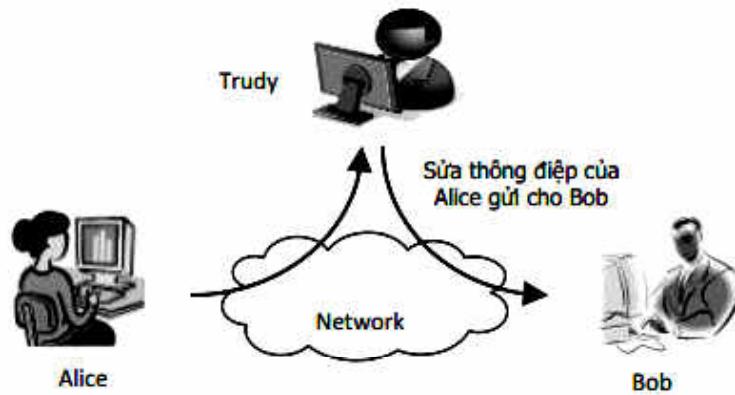
Trong trường hợp này Trudy chặn các thông điệp Alice gửi cho Bob, và xem được nội dung của thông điệp.



Hình 1.1: Xem trộm thông điệp

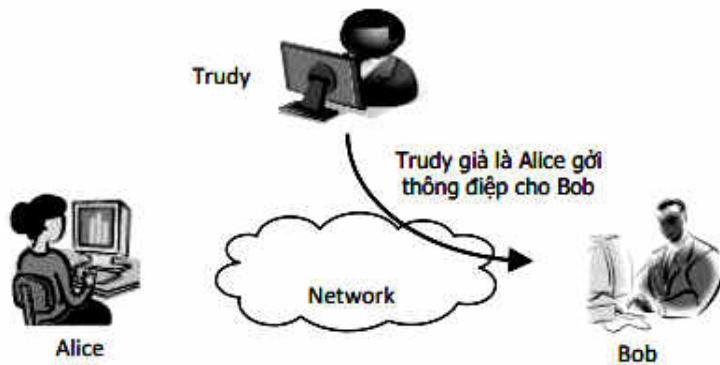
1. Thay đổi thông điệp (*Modification of Message*)

Trudy chặn các thông điệp Alice gửi cho Bob và ngăn không cho các thông điệp này đến đích. Sau đó Trudy thay đổi nội dung của thông điệp và gửi tiếp cho Bob. Bob nghĩ rằng nhận được thông điệp nguyên bản ban đầu của Alice mà không biết rằng chúng đã bị sửa đổi.

**Hình 1.2: Sửa thông điệp**

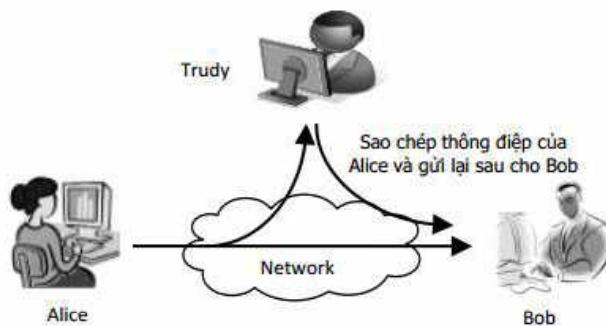
2. Mạo danh (Masquerade)

Trong trường hợp này Trudy giả là Alice gửi thông điệp cho Bob. Bob không biết điều này và nghĩ rằng thông điệp là của Alice.

**Hình 1.3: Mạo danh**

3. Phát lại thông điệp (Replay)

Trudy sao chép lại thông điệp Alice gửi cho Bob. Sau đó một thời gian Trudy gửi bản sao chép này cho Bob. Bob tin rằng thông điệp thứ hai vẫn là từ Alice, nội dung hai thông điệp là giống nhau. Thoạt đầu có thể nghĩ việc phát lại này là vô hại, tuy nhiên trong nhiều trường hợp cũng gây ra tác hại không kém so với việc giả mạo thông điệp. Xét tình huống sau: giả sử Bob là ngân hàng còn Alice là một khách hàng. Alice gửi thông điệp đề nghị Bob chuyển cho Trudy 1000\$. Alice có áp dụng các biện pháp như chữ ký điện tử với mục đích không cho Trudy mạo danh cũng như sửa thông điệp. Tuy nhiên nếu Trudy sao chép và phát lại thông điệp thì các biện pháp bảo vệ này không có ý nghĩa. Bob tin rằng Alice gửi tiếp một thông điệp mới để chuyển thêm cho Trudy 1000\$ nữa.



Hình 1.4: Phát lại thông điệp

1.2.2 Kiến trúc an ninh OSI

Để giúp cho việc hoạch định chính sách và xây dựng hệ thống an ninh tốt. Bộ phận chuẩn hóa tiêu chuẩn của tổ chức truyền thông quốc tế (*International Telecommunication Union*) đã nghiên cứu và đề ra Kiến trúc an ninh X800 dành cho hệ thống trao đổi thông tin mở OSI (*Open System Interconnection*). Kiến trúc này định ra một phương thức chung cho việc xác định các nhu cầu về an ninh thông tin. Nó cung cấp một cái nhìn tổng quan về các khái niệm môn học sẽ đề cập đến. Trong đó, nó chú trọng đến các dịch vụ an ninh, các cơ chế an ninh và các hành động tấn công.

Các dịch vụ an ninh: Theo X.800, dịch vụ an ninh là dịch vụ cung cấp bởi một tầng giao thức của các hệ thống mở kết nối nhằm đảm bảo an ninh cho các hệ thống và các cuộc truyền dữ liệu. Có 5 loại hình:

1. **Xác thực:** tin tưởng là thực thể trao đổi đúng là cái đã tuyên bố. Người đang trao đổi xưng tên với mình đúng là anh ta, không cho phép người khác mạo danh.
2. **Quyền truy cập:** ngăn cấm việc sử dụng nguồn thông tin không đúng vai trò. Mỗi đối tượng trong hệ thống được cung cấp các quyền hạn nhất định và chỉ được hành động trong khuôn khổ các quyền hạn đó.
3. **Bảo mật dữ liệu:** bảo vệ dữ liệu không bị khám phá bởi người không có quyền. Ví dụ như dùng các ký hiệu khác để thay thế các ký hiệu trong bản tin, mà chỉ người có bản quyền mới có thể khôi phục nguyên bản của nó.
4. **Toàn vẹn dữ liệu:** tin tưởng là dữ liệu được gửi từ người có quyền. Nếu có thay đổi như làm trì hoãn về mặt thời gian hay sửa đổi thông tin, thì xác thực sẽ cho cách kiểm tra nhận biết là có các hiện tượng đó đã xảy ra.

5. Không từ chối: chống lại việc chối bỏ của một trong các bên tham gia trao đổi.

Người gửi cũng không chối bỏ là mình đã gửi thông tin với nội dung như vậy và người nhận không thể nói dối là tôi chưa nhận được thông tin đó. Điều này là rất cần thiết trong việc trao đổi, thỏa thuận thông tin hàng ngày. Giả sử Bob là nhân viên môi giới chứng khoán của Alice. Alice gửi thông điệp yêu cầu Bob mua cổ phiếu của công ty Z. Ngày hôm sau, giá cổ phiếu công ty này giảm hơn 50%. Thấy bị thiệt hại, Alice nói rằng Alice không gửi thông điệp nào cả và quy trách nhiệm cho Bob. Bob phải có cơ chế để xác định rằng chính Alice là người gửi mà Alice không thể từ chối trách nhiệm được.

Cơ chế an toàn được định nghĩa trong X800 như sau:

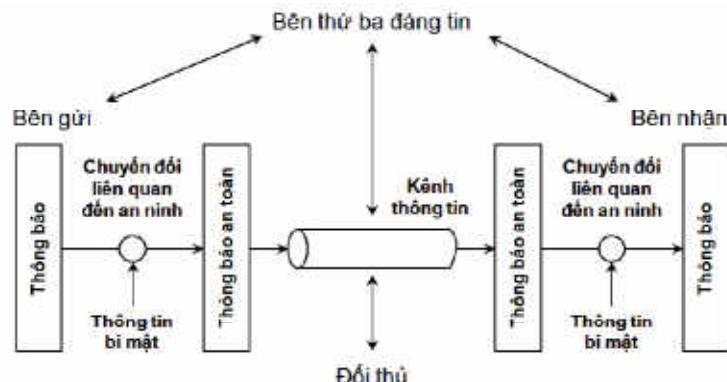
Cơ chế an toàn chuyên dụng được cài đặt trong một giao thức của một tầng vận chuyển nào đó: mã hoá, chữ ký điện tử, quyền truy cập, toàn vẹn dữ liệu, trao đổi có phép, đệm truyền, kiểm soát định hướng, công chứng.

Cơ chế an toàn phổ dụng không chỉ rõ được dùng cho giao thức trên tầng nào hoặc dịch vụ an ninh cụ thể nào: chức năng tin cậy cho một tiêu chuẩn nào đó, nhãn an toàn chứng tỏ đối tượng có tính chất nhất định, phát hiện sự kiện, vết theo dõi an toàn, khôi phục an toàn.

1.2.3 Mô hình mạng an toàn tổng quát

Sử dụng mô hình X800 đòi hỏi chúng ta phải thiết kế:

- Thuật toán phù hợp cho việc truyền an toàn.
- Phát sinh các thông tin mật (khóa) được sử dụng bởi các thuật toán.
- Phát triển các phương pháp phân phối và chia sẻ các thông tin mật.
- Đặc tả giao thức cho các bên để sử dụng việc truyền và thông tin mật cho các dịch vụ an toàn.



Hình 1.5: Mô hình bảo mật ruyền thông tin trên mạng

1.2.4 Vai trò của Mã hóa trong bảo mật thông tin trên mạng

Mật mã hay mã hóa dữ liệu (*cryptography*), là một công cụ cơ bản thiết yếu của bảo mật thông tin. Mật mã đáp ứng được các dịch vụ như xác thực, bảo mật, toàn vẹn dữ liệu, chống chối bỏ. Môn học sẽ tập trung tìm hiểu các thuật toán mật mã cài đặt các dịch vụ bảo mật trên.

1.2.5 Các giao thức thực hiện bảo mật

Sau khi tìm hiểu về mật mã, chúng ta sẽ tìm hiểu về cách ứng dụng chúng vào thực tế thông qua một số giao thức bảo mật phổ biến hiện nay là:

- Keberos: là giao thức dùng để chứng thực dựa trên mã hóa đối xứng.
- Chuẩn chứng thực X509: dùng trong mã hóa khóa công khai.
- Secure Socket Layer (SSL): là giao thức bảo mật Web, được sử dụng phổ biến trong Web và thương mại điện tử.
- PGP và S/MIME: bảo mật thư điện tử email.
- Mô hình lý thuyết và nội dung các giao thức trên được trình bày trong bài 6 và 7.

1.3 BẢO VỆ HỆ THỐNG KHỎI SỰ XÂM NHẬP PHÁ HOẠI TỪ BÊN NGOÀI

Bởi vì Internet đã kết nối các máy tính ở khắp nơi trên thế giới lại với nhau, thì vấn đề bảo vệ máy tính khỏi sự thâm nhập phá hoại từ bên ngoài là một điều cần thiết. Thông qua mạng Internet, các hacker có thể truy cập vào các máy tính trong một tổ chức (dùng telnet chẳng hạn), lấy trộm các dữ liệu quan trọng như mật khẩu, thẻ tín dụng, tài liệu... Hoặc đơn giản chỉ là phá hoại, gây trực trặc hệ thống mà tổ chức đó phải tốn nhiều chi phí để khôi phục lại tình trạng hoạt động bình thường.

Để thực hiện việc bảo vệ này, người ta dùng khái niệm “kiểm soát truy cập” (*Access Control*). Khái niệm kiểm soát truy cập này có hai yếu tố sau:

- Chứng thực truy cập (*Authentication*): xác nhận rằng đối tượng (con người hay chương trình máy tính) được cấp phép truy cập vào hệ thống. Ví dụ: để sử dụng máy tính thì trước tiên đối tượng phải logon vào máy tính bằng username và

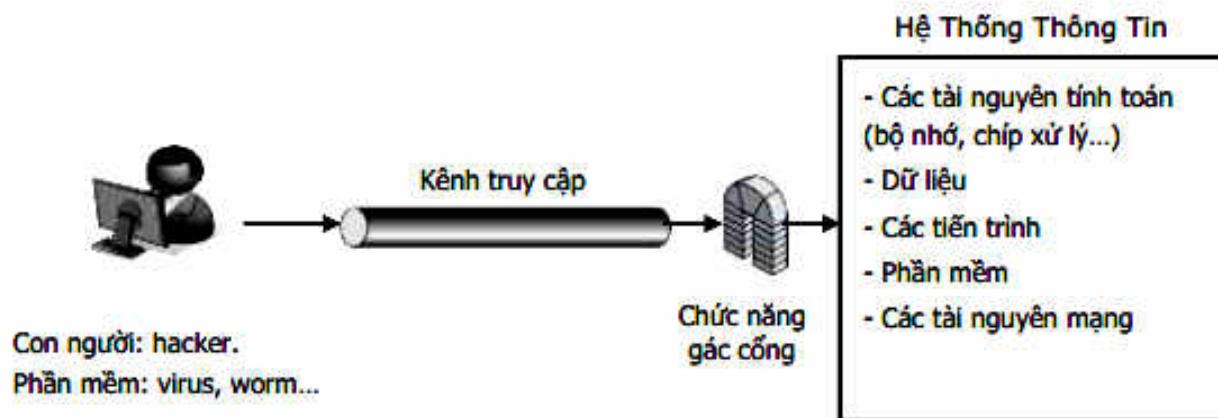
password. Ngoài ra, còn có các phương pháp chứng thực khác như sinh trắc học (dấu vân tay, mống mắt...) hay dùng thẻ (thẻ ATM...).

- Phân quyền (*Authorization*): các hành động được phép thực hiện sau khi đã truy cập vào hệ thống. Ví dụ: bạn được cấp username và password để logon vào hệ điều hành, tuy nhiên bạn chỉ được cấp quyền để đọc một file nào đó. Hoặc bạn chỉ có quyền đọc file mà không có quyền xóa file.

Với nguyên tắc như vậy thì một máy tính hoặc một mạng máy tính được bảo vệ khỏi sự thâm nhập của các đối tượng không được phép. Tuy nhiên thực tế chúng ta vẫn nghe nói đến các vụ tấn công phá hoại. Để thực hiện điều đó, kẻ phá hoại tìm cách phá bỏ cơ chế Authentication và Authorization bằng các cách thức sau:

- Dùng các đoạn mã phá hoại (*Malware*): như virus, worm, trojan, backdoor những đoạn mã độc này phát tán lan truyền từ máy tính này qua máy tính khác dựa trên sự bất cẩn của người sử dụng, hay dựa trên các lỗi của phần mềm. Lợi dụng các quyền được cấp cho người sử dụng (chẳng hạn rất nhiều người login vào máy tính với quyền administrator), các đoạn mã này thực hiện các lệnh phá hoại hoặc dò tìm password của quản trị hệ thống để gửi cho hacker, cài đặt các cổng hậu để hacker bên ngoài xâm nhập.
- Thực hiện các hành vi xâm phạm (*Intrusion*): việc thiết kế các phần mềm có nhiều lỗ hổng, dẫn đến các hacker lợi dụng để thực hiện những lệnh phá hoại. Các lệnh này thường là không được phép đổi với người bên ngoài, nhưng lỗ hổng của phần mềm dẫn đến sự cho phép (ngoài kiểm soát). Trong những trường hợp đặc biệt, lỗ hổng phần mềm cho phép thực hiện những lệnh phá hoại mà ngay cả người thiết kế chương trình không ngờ tới, hay là hacker có thể sử dụng các cổng hậu do các backdoor tạo ra để xâm nhập.

Để khắc phục các hành động phá hoại này, người ta dùng các chương trình có chức năng gác cổng, phòng chống. Những chương trình này dò tìm virus hoặc dò tìm các hành vi xâm phạm để ngăn chặn chúng, không cho chúng thực hiện hoặc xâm nhập. Đó là các chương trình chống virus, chương trình firewall ... Ngoài ra các nhà phát triển phần mềm cần có quy trình xây dựng và kiểm lỗi phần mềm nhằm hạn chế tối đa những lỗ hổng bảo mật có thể có.



Hình 1.6: Mô hình phòng chống xâm nhập và phá hoại hệ thống

Sử dụng mô hình trên đòi hỏi chúng ta phải:

- Lựa chọn hàm canh cổng phù hợp cho người sử dụng có danh tính.
- Cài đặt kiểm soát quyền truy cập để tin tưởng rằng chỉ có người có quyền mới truy cập được thông tin đích hoặc nguồn.

Các hệ thống máy tính tin cậy có thể dùng mô hình này.

TÓM TẮT

Sự phát triển mạnh mẽ của công nghệ thông tin và đặc biệt là internet, thông tin được lưu trữ trên máy tính và được truyền trên mạng internet, do đó xuất hiện nhu cầu an toàn và bảo mật thông tin trên máy tính. Bài học giới thiệu tổng quan về các loại tấn công mạng: Release of Message Content (xem trộm thông tin), Modification of Message (thay đổi thông điệp), Masquerade (mạo danh), Replay message (gửi lại thông điệp).... Lợi dụng những yếu tố chủ quan của người dùng, lỗ hổng của hệ thống mạng, hacker tấn công vào hệ thống đánh cắp dữ liệu, phá hoại dữ liệu, làm cho hệ thống mạng ko hoạt động được.

Nhu cầu bảo vệ dữ liệu là rất quan trọng. Để đảm bảo cho 1 hệ truyền tin trên mạng 1 cách an toàn, tiêu chuẩn x800 đặt ra 5 dịch vụ bảo mật: xác thực, quyền truy cập, bảo mật, toàn vẹn dữ liệu, và chống chối bỏ. Lý thuyết mật mã là 1 công cụ thiết

yếu để bảo mật thông tin và được cài đặt trong các ứng dụng bảo mật mạng cụ thể như: Keberos, chuẩn chứng thực X509, PGP và S/MIME: bảo mật thư điện tử email.

Khi mạng Internet đã kết nối các máy tính ở khắp nơi trên thế giới lại với nhau, thì vẫn đề bảo vệ máy tính khỏi sự thâm nhập phá hoại từ bên ngoài là một điều cần thiết. Thông qua mạng Internet, các hacker có thể truy cập vào các máy tính trong một tổ chức (dùng telnet chẵng hạn), lấy trộm các dữ liệu quan trọng như mật khẩu, thẻ tín dụng, tài liệu... Hoặc dùng các đoạn mã phá hoại (Malware), thực hiện các hành vi xâm phạm (Intrusion), gây trực trặc hệ thống mà tổ chức đó phải tốn nhiều chi phí để khôi phục lại tình trạng hoạt động bình thường, và khái niệm Control Access (kiểm soát truy cập) dùng cho việc bảo vệ này, đồng thời sử dụng Firewall hoặc các hệ thống phát hiện chống xâm nhập IDS/IPS, kiểm lỗi phần mềm ...

CÂU HỎI ÔN TẬP

Câu 1: Nếu các hình thức tấn công trong quá trình truyền tin trên mạng?

Câu 2: Dịch vụ bảo mật mạng là gì?

Câu 3: Cho biết các dịch vụ bảo mật mạng theo chuẩn x800?

Câu 4: Cơ chế an ninh mạng là gì?

Câu 5: Thuật toán mã hóa có vai trò như thế nào trong bảo mật thông tin?

Câu 6: Bảo vệ hệ thống khỏi sự tấn công bên ngoài là gì?

BÀI 2: MÃ HÓA ĐỔI XỨNG CỔ ĐIỂN

Sau khi học xong bài này, sinh viên hiểu:

- Các khái niệm cơ bản về hệ mã đổi xứng và các yêu cầu liên quan đến việc xây dựng hệ mã đổi xứng
- Hệ mã đầy, cách tính toán và ý nghĩa bảo mật của nó
- Hệ mã hóa hoán vị

2.1 TỔNG QUAN VỀ HỆ MÃ ĐỔI XỨNG

2.1.1 Khái niệm mã hóa đổi xứng

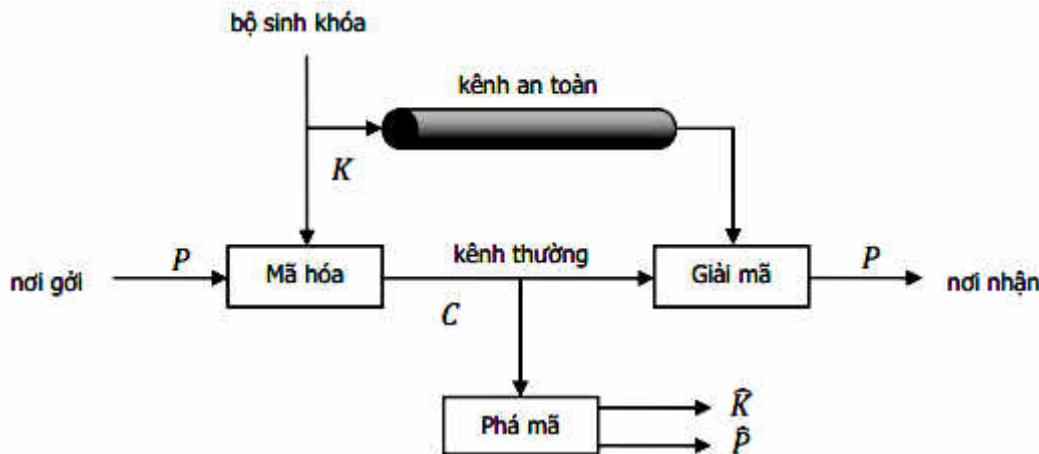
Mã đổi xứng sử dụng cùng một khóa cho việc mã hóa và giải mã. Có thể nói mã đổi xứng là mã một khóa hay mã khóa riêng hay mã khóa thỏa thuận. Mô hình hệ mã khóa đổi xứng được trình bày trong hình 2-1. Hệ mật mã đổi xứng là bộ năm (P , C , K , E , D), thỏa mãn các điều kiện sau:

- P không gian bản rõ
- C không gian bản mã
- K không gian khóa
- Với mỗi $k \in K$, tồn tại hàm lập mã $e_k \in E$ và hàm giải mã $d_k \in D$. Mỗi $e_k: P \rightarrow C$ và $d_k: C \rightarrow P$ là các hàm thỏa $d_k(e_k(x))=x$ với mỗi $x \in P$.

Trong hệ mã này, người gửi và người nhận chia sẻ khóa chung k , mà họ có thể trao đổi bí mật với nhau. Ta xét hai hàm ngược nhau: e là hàm biến đổi bản rõ thành bản mã và d là hàm biến đổi bản mã trở về bản rõ.

Mọi thuật toán mã cổ điển đều là mã khóa đổi xứng, vì ở đó thông tin về khóa được chia sẻ giữa người gửi và người nhận. Mã đổi xứng là kiểu duy nhất trước khi phát minh ra khóa mã công khai (còn được gọi là mã không đổi xứng) vào những năm

1970. Hiện nay các mã đối xứng và công khai tiếp tục phát triển và hoàn thiện. Mã công khai ra đời hỗ trợ mã đối xứng chứ không thay thế nó, do đó mã đối xứng đến nay vẫn được sử dụng rộng rãi.



Hình 2.1: Mô hình mã hóa đối xứng

2.1.2 Các yêu cầu

Một mã đối xứng có các đặc trưng là cách xử lý thông tin của thuật toán mã, giải mã, tác động của khóa vào bản mã, độ dài của khóa. Mỗi liên hệ giữa bản rõ, khóa và bản mã càng phức tạp càng tốt, nếu tốc độ tính toán là chấp nhận được. Cụ thể hai yêu cầu để sử dụng an toàn mã khóa đối xứng là:

1. *Thuật toán mã hóa mạnh*. Có cơ sở toán học vững chắc đảm bảo rằng mặc dù công khai thuật toán, mọi người đều biết, nhưng việc thám mã là rất khó khăn và phức tạp nếu không biết khóa.
2. *Khóa mật chỉ có người gửi và người nhận biết*. Có kênh an toàn để phân phối khóa giữa các người sử dụng chia sẻ khóa. Mỗi liên hệ giữa khóa và bản mã là không nhận biết được.

2.1.3 Phá mã

Phá mã là nỗ lực giải mã văn bản đã được mã hóa không biết trước khóa bí mật. Có hai phương pháp phá mã: Vét cạn và Thám mã.

1. *Tấn công duyệt toàn bộ (Brute-Force)*: Về mặt lý thuyết phương pháp duyệt tổng thể là luôn thực hiện được, do có thể tiến hành thử từng khóa, mà số khóa là hữu

hạn. Phần lớn công sức của các tấn công đều tỷ lệ thuận với kích thước khóa. Khóa càng dài thời gian tìm kiếm càng lâu và thường tăng theo hàm mũ. Ta có thể giả thiết là kẻ thám mã có thể dựa vào bối cảnh để biết hoặc nhận biết được bản rõ.

Sau đây là một số thống kê về mối liên hệ giữa độ dài khóa, kích thước không gian khóa, tốc độ xử lý và thời gian tìm duyệt tổng thể. Chúng ta nhận thấy với độ dài khóa từ 128 bit trở lên, thời gian yêu cầu là rất lớn, lên đến hàng tỷ năm, như vậy có thể coi phương pháp duyệt tổng thể là không hiện thực.

Kích thước khóa (bit)	Số lượng khóa	Thời gian cần thiết (1 giải mã/μs)	Thời gian cần thiết (10 ⁶ giải mã/μs)
32	$2^{32} = 4,3 \times 10^9$	$2^{31} \mu\text{s} = 35,8 \text{ phút}$	2,15 ms
56	$2^{56} = 7,2 \times 10^{16}$	$2^{55} \mu\text{s} = 1142 \text{ năm}$	10,01 giờ
128	$2^{128} = 3,4 \times 10^{38}$	$2^{127} \mu\text{s} = 5,4 \times 10^{24} \text{ năm}$	$5,4 \times 10^{18} \text{ năm}$
168	$2^{168} = 3,7 \times 10^{50}$	$2^{167} \mu\text{s} = 5,9 \times 10^{36} \text{ năm}$	$5,9 \times 10^{30} \text{ năm}$
26 ký tự (hoán vị)	$26! = 4 \times 10^{26}$	$2 \times 10^{26} \mu\text{s} =$ $6,4 \times 10^{12} \text{ năm}$	$6,4 \times 10^6 \text{ năm}$

Khóa DES dài 56 bit

Tuổi vũ trụ: $\sim 10^{10}$ năm

Khóa AES dài 128+ bit

Khóa 3DES dài 168 bit

Hình 2.1: Thống kê về thời gian vét cạn theo kích thước khóa

2. *Tấn công thám mã*: dựa trên thuật toán và một số thông tin về các đặc trưng chung về bản rõ hoặc một số mẫu bản rõ/bản mã. Kiểu tấn công này nhằm khai phá các đặc trưng của thuật toán để tìm bản rõ cụ thể hoặc tìm khóa. Nếu tìm được khóa thì là tai họa lớn. Các kiểu tấn công thám mã.

- *Chỉ dùng bản mã*: biết thuật toán và bản mã, dùng phương pháp thống kê, xác định bản rõ.
- *Biết bản rõ*: biết thuật toán, biết được bản mã/bản rõ tấn công tìm khóa.
- *Chọn bản rõ*: chọn bản rõ/nhận được bản mã, biết thuật toán tấn công tìm khóa.
- *Chọn bản mã*: chọn bản mã và có được bản rõ tương ứng, biết thuật toán tấn công tìm khóa.
- *Chọn bản tin*: chọn được bản rõ hoặc mã và mã hoặc giải mã tương ứng, tấn công tìm khóa.

2.1.4 Độ an toàn

Có thể phân loại an toàn thành hai kiểu như sau:

- *An toàn không điều kiện*: ở đây không quan trọng máy tính mạnh như thế nào, có thể thực hiện được bao nhiêu phép toán trong một giây, mã hóa không thể bị bẻ, vì bản mã không cung cấp đủ thông tin để xác định duy nhất bản rõ. Việc dùng bộ đệm ngẫu nhiên một lần để mã dòng cho dữ liệu mà ta sẽ xét cuối bài này được coi là an toàn không điều kiện. Ngoài ra chưa có thuật toán mã hóa nào được coi là an toàn không điều kiện.
- *An toàn tính toán*: với nguồn lực máy tính giới hạn và thời gian có hạn (chẳng hạn thời gian tính toán không quá tuổi của vũ trụ) mã hóa coi như không thể bị bẻ. Trong trường hợp này coi như mã hóa an toàn về mặt tính toán. Nói chung từ nay về sau, một thuật toán mã hóa an toàn tính toán được coi là an toàn.

2.2 CÁC HỆ MÃ THÊ

2.2.1 Hệ mã đẩy (Hệ mã CAESAR)

Hệ mã đẩy là hệ mã hóa thay thế xuất hiện sớm nhất và đơn giản nhất. Hệ mã này sử dụng đầu tiên bởi Julius Caesar vào mục đích quân sự. Hệ mã thực hiện dịch chuyển xoay vòng theo thứ tự chữ cái. Khóa k là số bước dịch chuyển. Với mỗi chữ cái của văn bản, tiến hành số hóa mỗi chữ cái là một con số nguyên từ 0 đến 25 với bảng số hóa:

A	b	c	d	E	f	g	h	i	j	k	l	m
0	1	2	3	4	5	6	7	8	9	10	11	12
N	o	p	q	R	s	t	u	v	w	x	Y	z
13	14	15	16	17	18	19	20	21	22	23	24	25

Rồi áp dụng cách thức mã hóa.

Cho $P=C=K=Z_{26}$. với $x, y \in Z_{26}$, $0 \leq k \leq 25$, ta định nghĩa:

$$y = e_k = x + k \pmod{26} \text{ và } x = dk = y - k \pmod{26}$$

Với $k=3$, hệ mã được là mã Ceasar.

Ví dụ: Mã hóa "meet me after class" với $k = 3$

Chọn k = 3, ta có bảng chuyển đổi như sau:

Chữ ban đầu: a b c d e f g h i j k l m n o p q r s t u v w x y z

Chữ thay thế: D E F G H I J K L M N O P Q R S T U V W X Y Z A B C

(sau Z sẽ vòng lại là A, do đó x → A, y → B và z → C)

Giả sử có bản tin gốc (*bản rõ*): meet me after the toga party

Như vậy bản tin mã hóa (*bản mã*) sẽ là: PHHW PH DIWHU WKH WRJD SDUWB

KEY	PHHW	PH	DIWHU	WKH	WRJD	SDUWB
1	oggv	og	chvgt	vjg	vqic	rctva
2	nffu	nf	bgufs	uif	uphb	qbsuz
3	meet	me	after	the	toga	party
4	ldds	ld	zesdq	sgd	snfz	ozqsx
5	kccr	kc	ydrcp	rfc	rmey	nyprw
6	jbbq	jb	xcqbo	qeb	qldx	mxoqv
7	iaap	ia	wbpan	pda	pkcw	lwnpu
8	hzzo	hz	vaozm	ocz	ojbv	kvmot
9	gyyn	gy	uznyl	nby	niau	julns
10	fxxm	fx	tymxk	max	mhzt	itkmr
11	ewwl	ew	sxlwj	lzw	lgys	hsjlq
12	dvvk	dv	rwkvi	kyv	kfxr	grikp
13	cuuj	cu	qvjuh	jxu	jewq	fqhjo
14	btti	bt	puitg	iwt	idvp	epgin
15	assh	as	othsf	hvs	hcuo	dofhm
16	zrrg	zr	nsgre	gur	gbtn	cnegl
17	yqqf	yq	mrfqd	ftq	fasm	bmdfk
18	xppe	xp	lqepc	esp	ezrl	alcej
19	wood	wo	kpdob	dro	dyqk	zkbdi
20	vnnc	vn	jocna	cqn	cxpj	yjach
21	ummb	um	inbmz	bpm	bwoi	xizbg
22	tlla	tl	hmaly	aol	avnh	whyaf
23	skkz	sk	glzkx	znk	zumg	vgxze
24	rjjy	rj	fkyjw	ymj	ytlf	ufwyd
25	qiix	qi	ejxiv	xli	xske	tevxc

Hình 2.2: Minh họa vét cạn

Nhận xét phá mã: Sử dụng phương pháp vét cạn. Khóa chỉ là một chữ cái (hay một số giữa 1 và 25). Tiến hành thử tất cả 25 khóa có thể. Việc thực hiện là dễ dàng.

Ba yếu tố quan trọng dẫn đến dễ phá mã là biết trước các giải thuật mã hóa và giải mã, chỉ có 25 khóa để thử và có thể dễ dàng nhận ra được ngôn ngữ của nguyên bản.

Ví dụ: Giả sử đối thủ của Ceasar có được bản mã PHHW PH DIWHU WKH WRJD SDUWB và biết được phương pháp mã hóa và giải mã là phép cộng trừ modulo 26. Đối thủ có thể thử tất cả 25 trường hợp của k như sau:

Trong 25 trường hợp trên, chỉ có trường hợp $k=3$ thì bản giải mã tương ứng là có ý nghĩa. Do đó đối thủ có thể chắc chắn rằng 'meet me after the toga party' là bản rõ ban đầu.

2.2.2 Mã hóa thay thế đơn bảng

Hệ mã hóa đơn bản khắc phục nhược điểm của hệ mã Ceasar bằng cách mã hóa các chữ không chỉ là dịch chuyển bảng chữ, mà có thể tạo ra các bước nhảy khác nhau cho các chữ.

Trong hệ mã, mỗi chữ của bản rõ được ánh xạ đến một chữ khác nhau của bản mã. Nên, mỗi cách mã như vậy sẽ tương ứng với một hoán vị của bảng chữ và hoán vị đó chính là khóa của hệ mã đã cho. Đo đó, độ dài khóa ở đây là 26 và số khóa có thể có là 26!. Số khóa như vậy là rất lớn.

Ví dụ. Ta có bản mã tương ứng với bản rõ trong mã bảng chữ đơn như sau:

Bảng chủ rõ: abcdefghijklmnopqrstuvwxyz

Bảng khóa (chùm khóa): DVQFIBJWPESCHTMYAUOLRGZN

Kết quả mã hóa:

Bản rõ: ifewishtoreplaceletters

Bản mã: WIRFRWAJUHYFTSDVFSUUUFYA

Nhận xét phá mã: Tổng cộng có $26!$ xấp xỉ khoảng 4×10^{26} khóa. Số lượng khóa lớn nên khó tấn công vét cạn nếu không có sự hỗ trợ của máy tính. Để phá hệ mã này sử dụng phương pháp thám mã dựa vào các đặc trưng về ngôn ngữ. Các đặc trưng về tần suất xuất hiện của các chữ trong bản rõ và các chữ tương ứng trong bản mã là như nhau, nên kẻ thám mã có thể đoán được ánh xạ của một số chữ và từ đó mò tìm ra chữ mã cho các chữ khác.

Trong tiếng Anh chữ E được sử dụng nhiều nhất; sau đó đến các chữ T, R, N, I, O, A, S. Một số chữ rất ít dùng như: Z, J, K, Q, X. Bằng phương pháp thống kê, ta có thể xây dựng các bảng các tần suất các chữ đơn, cặp chữ, bộ ba chữ (hình 2-2).

Chữ cái (%)	Cụm 2 chữ (%)	Cụm 3 chữ (%)	Từ (%)
E 13.05	TH 3.16	THE 4.72	THE 6.42
T 9.02	IN 1.54	ING 1.42	OF 4.02
O 8.21	ER 1.33	AND 1.13	AND 3.15
A 7.81	RE 1.30	ION 1.00	TO 2.36
N 7.28	AN 1.08	ENT 0.98	A 2.09
I 6.77	HE 1.08	FOR 0.76	IN 1.77
R 6.64	AR 1.02	TIO 0.75	THAT 1.25
S 6.46	EN 1.02	ERE 0.69	IS 1.03
H 5.85	TI 1.02	HER 0.68	I 0.94
D 4.11	TE 0.98	ATE 0.66	IT 0.93
L 3.60	AT 0.88	VER 0.63	FOR 0.77
C 2.93	ON 0.84	TER 0.62	AS 0.76
F 2.88	HA 0.84	THA 0.62	WITH 0.76
U 2.77	OU 0.72	ATI 0.59	WAS 0.72
M 2.62	IT 0.71	HAT 0.55	HIS 0.71
P 2.15	ES 0.69	ERS 0.54	HE 0.71
Y 1.51	ST 0.68	HIS 0.52	BE 0.63
W 1.49	OR 0.68	RES 0.50	NOT 0.61
G 1.39	NT 0.67	ILL 0.47	BY 0.57
B 1.28	HI 0.66	ARE 0.46	BUT 0.56
V 1.00	EA 0.64	CON 0.45	HAVE 0.55
K 0.42	VE 0.64	NCE 0.45	YOU 0.55
X 0.30	CO 0.59	ALL 0.44	WHICH 0.53
J 0.23	DE 0.55	EVE 0.44	ARE 0.50
Q 0.14	RA 0.55	ITH 0.44	ON 0.47
Z 0.09	RO 0.55	TED 0.44	OR 0.45

Hình 2.3: Bảng liệt kê tần suất chữ cái tiếng Anh

Điều quan trọng là mã thế trên bảng chữ đơn không làm thay đổi tần suất tương đối của các chữ, có nghĩa là ta vẫn có bảng tần suất trên nhưng đổi với bảng chữ mã tương ứng. Điều đó được phát hiện bởi các nhà khoa học Ai cập từ thế kỷ thứ 9. Do đó có cách thám mã trên bảng chữ đơn như sau:

- Tính toán tần suất của các chữ trong bản mã
- So sánh với các giá trị đã biết
- Tìm kiếm các chữ đơn hay dùng A-I-E, bộ đôi NO và bộ ba RST; và các bộ ít dùng JK, X-Z...

- Trên bảng chữ đơn cần xác định các chữ dùng các bảng bộ đôi và bộ ba trợ giúp.

Ví dụ. Thám mã bản mã trên bảng chữ đơn, cho bản mã:

UZQSOVUOHXMOPVGPOZPEVSGZWSZOPFPESXUDBMETSXAIZ

UEPHZHMDZSHZOWSFPAPPDTSPQUZWYMXUZUHSXE PYEP

OPDZSZUFPOUDTMOHMQ

- Tính tần suất các chữ
- Đoán P và Z là e và t.
- Khi đó ZW là th và ZWP là the.
- Suy luận tiếp tục ta có bản rõ:

it was disclosed yesterday that several informal but
direct contacts have been made with political
representatives in moscow

2.2.3 Hệ mã VIGENRE

Mã thế đa bảng đơn giản nhất là mã Vigenere. Thực chất quá trình mã hoá Vigenere là việc tiến hành đồng thời dùng nhiều mã Ceasar cùng một lúc trên bản rõ với nhiều khóa khác nhau (Keyword Ceasar).

Định nghĩa: Một hệ mật là một bộ (P, C, K, E, D) . Cho m là một số nguyên dương cố định nào đó. Định nghĩa $P = C = K = (Z_{26})^m$. Với khóa $K = (k_1, k_2, \dots, k_m)$. Ta xác định:

$$e_K(x_1, x_2, \dots, x_m) = (x_1+k_1, x_2+k_2, \dots, x_m+k_m)$$

và

$$d_K(y_1, y_2, \dots, y_m) = (y_1-k_1, y_2-k_2, \dots, y_m-k_m)$$

trong đó tất cả các phép toán được thực hiện trong Z_{26}

Ví dụ: Khóa là một từ. Ví dụ: plain $\rightarrow Z_{26}(15, 11, 0, 8, 13)$

Lấy lần lượt 5 ký tự để mã hóa bằng 5 khóa tương ứng.

Bản rõ: sendmoremoney → Bản mã: hpnlzdcueubcpv

Trên thực tế để hỗ trợ mã Vigenere, người ta đã tạo ra trang Saint – Cyr để trợ giúp cho việc mã và giải mã nhanh. Bảng cỡ 26 x 26, phần tử thứ i dòng i, cột thứ j cho biết giá trị mã hóa bản rõ i bằng khóa j.

Nhận xét phá mã Vigenere: Bởi vì khóa có thể sử dụng cho nhiều chữ cái khác nhau, nên tần suất xuất hiện các chữ trên bản mã thay đổi. Tuy nhiên, tính chất tần suất của ngôn ngữ chưa mất hoàn toàn, vì độ dài của khóa có hạn dẫn đến có thể tạo nên chu kỳ vòng lặp. Kẻ thám mã bắt đầu từ tần suất của chữ để xem có phải đây là mã đơn bảng chữ hay không. Giả sử đây là mã đa bảng chữ, sau đó xác định số bảng chữ trong từ khóa và lần tìm từng chữ.

2.2.4 Mã khóa tự động

Lý tưởng nhất là ta có khóa dài như bản tin. Do đó Vigenere đề xuất khóa tự động sinh cho bằng độ dài bản tin như sau: từ khóa được nối tiếp bằng chính bản rõ để tạo thành khóa. Sau đó dùng mã Vigenere để mã bản rõ đã cho. Khi đó biết từ khóa có thể khôi phục được một số chữ ban đầu của bản rõ. Sau đó tiếp tục sử dụng chúng để giải mã cho văn bản còn lại. Sự cải tiến này làm mất khái niệm chu kỳ, gây khó khăn cho việc thám mã, nhưng vẫn còn đặc trưng tần suất để tấn công.

Ví dụ. Cho từ khóa deceptive. Ta viết bản rõ nối tiếp vào từ khóa tạo thành từ khóa mới có độ dài bằng độ dài bản rõ.

key: deceptivewearediscoveredsav

plaintext: wearediscoveredsaveyourself

ciphertext: ZICVTWQNGKZEIIGASXSTSLVVWLA

2.2.5 Hệ mã độn một lần(ONE-TIME PAD, OTP)

Có thể thấy rằng điểm yếu của mã hóa đa bảng là do sự lặp lại các từ trong khóa, ví dụ từ DECEPTIVE được lặp đi lặp lại nhiều lần. Điều này làm cho vẫn tồn tại một mối liên quan giữa bản rõ và bản mã. Người phá mã tận dụng mối liên quan này để thực hiện phá mã. Do đó vẫn đề ở đây là làm sao để giữa bản rõ và bản mã thật sự ngẫu nhiên, không tồn tại mối quan hệ nào. Để giải quyết vấn đề này, Joseph

Mauborgne, giám đốc viện nghiên cứu mật mã của quân đội Mỹ, vào cuối cuộc chiến tranh thế giới lần thứ nhất, đã đề xuất phương án là dùng khóa *ngẫu nhiên*. Khóa ngẫu nhiên có chiều dài bằng chiều dài của bản rõ, mỗi khóa chỉ sử dụng một lần.

Ví dụ mã hóa bản tin '*wearediscoveredsaveyourself*'

Bản tin P: wearediscoveredsaveyourself

Khóa K1: FHWYKLVMKVXCVKDJSFSAPXZCVP

Bản mã C: BLWPOODEMJFBTZNJVNJQOJORGGU

Nếu ta dùng khóa K1 để giải mã thì sẽ có được lại bản tin P '*wearediscoveredsaveyourself*'. Tuy nhiên xét hai trường hợp giải mã bản mã trên với 2 khóa khác như sau:

Trường hợp 1: Bản mã C: BLWPOODEMJFBTZNJVNJQOJORGGU

Khóa K2: IESRLKBWJFCIFZUCJLZXAXAAPSY

Bản giải mã: theydecidedtoattacktomorrow

(they decided to attack tomorrow)

Trường hợp 2: Bản mã C: BLWPOODEMJFBTZNJVNJQOJORGGU

Khóa K3: FHAHDDRAIQFIASJGJWQSVVBJAZB

Bản giải mã: wewillmeetatthepartytonight

(we will meet at the party tonight)

Nhận xét phá mã độn một lần: Trong cả hai trường hợp trên thì bản giải mã đều có ý nghĩa. Điều này có nghĩa là nếu người phá mã thực hiện phá mã vét cạn thì sẽ tìm được nhiều khóa ứng với nhiều bản tin có ý nghĩa, do đó sẽ không biết được bản tin nào là bản rõ. Điều này chứng minh phương pháp One-Time Pad là phương pháp mã hóa an toàn tuyệt đối.

Một điều cần chú ý là để phương pháp OTP là an toàn tuyệt đối thì mỗi khóa chỉ được sử dụng một lần. Nếu một khóa được sử dụng nhiều lần thì cũng không khác gì việc lặp lại một từ trong khóa (ví dụ khóa có từ DECEPTIVE được lặp lại). Ngoài ra các khóa phải thật sự ngẫu nhiên với nhau. Nếu các điều này bị vi phạm thì sẽ có một mối liên hệ giữa bản rõ và bản mã, mà người phá mã sẽ tận dụng mối quan hệ này.

Tuy nhiên, phương pháp OTP không có ý nghĩa sử dụng thực tế. Vì chiều dài khóa bằng chiều dài bản tin, mỗi khóa chỉ sử dụng một lần, nên thay vì truyền khóa trên kênh an toàn thì có thể truyền trực tiếp bản rõ mà không cần quan tâm đến vấn đề mã hóa.

2.3 CÁC HỆ MÃ KIỂU HOÁN VỊ

2.3.1 Hệ mã Rail Fence

Đây là mã hoán vị đơn giản. Viết các chữ của bản rõ theo đường chéo trên một số dòng. Sau đó đọc các chữ theo từng dòng sẽ nhận được bản mã. Số dòng chính là khóa của mã. Vì khi biết số dòng ta sẽ tính được số chữ trên mỗi dòng và lại viết bản mã theo các dòng sau đó lấy bản rõ bằng cách viết lại theo các cột.

Ví dụ. Viết bản tin “meet me after the toga party” lần lượt trên hai dòng như sau

m e m a t r h t g p r y

e t e f e t e o a a t

Sau đó ghép các chữ ở dòng thứ nhất với các chữ ở dòng thứ hai cho bản mã:

MEMATRHTGPRYETEFETEOAAT

2.3.2 Hệ mã hoán vị

Hệ mã hóa hoán vị che đậy nội dung văn bản bằng cách sắp xếp lại trật tự các chữ cái. Hệ mã không thay đổi các chữ cái của nguyên bản. Bản mã có tần số xuất hiện các chữ cái giống như nguyên bản.

Định nghĩa: Một hệ mật là một bộ (P, C, K, E, D)

Cho m là một số nguyên dương xác định nào đó. Cho $P = C = (\mathbb{Z}_{26})_m$ và cho K gồm tất cả các hoán vị của $\{1, \dots, m\}$. Đối một khóa π (tức là một hoán vị) ta xác định

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

$$\text{và } d_{\pi}(x_1, \dots, x_m) = (y_{\pi^{-1}(1)}, \dots, y_{\pi^{-1}(m)})$$

trong đó π^{-1} là hoán vị ngược của π

Ví dụ: $m=6$ và hoán vị là:

1	2	3	4	5	6
3	5	1	6	4	2

Hoán vị ngược sẽ là:

1	2	3	4	5	6
3	6	1	5	2	4

Bản rõ: shesellseashellsbytheseashore

Tách thành nhóm 6 chữ cái:

shesel | lsseas | hellsb | ythese | ashore

Bảng mã = Bảng hóa vị từng nhóm

eeslsh | salses | lshble | hsyeet | hraeos

eeslshsalseslshblehsyeethraeos

2.3.3 Mã tích

Mã hóa bằng cách hoán vị không an toàn vì các đặc trưng tần xuất của ngôn ngữ không thay đổi. Có thể sử dụng một số hệ mã liên tiếp nhau sẽ làm cho mã khó hơn. Mã cổ điển chỉ sử dụng một trong hai phương pháp thay thế hoặc hoán vị. Người ta nghĩ đến việc kết hợp cả hai phương pháp này trong cùng một mã và có thể sử dụng đan xen hoặc lặp nhiều vòng. Đôi khi ta tưởng lặp nhiều lần cùng một loại mã sẽ tạo nên mã phức tạp hơn, nhưng trên thực tế trong một số trường hợp về bản chất chúng cũng tương đương với một lần mã cùng loại nào đó như: tích của hai phép thay thế sẽ là một phép thay thế; tích của hai phép hoán vị sẽ là một phép hoán vị. Nhưng nếu hai loại mã đó khác nhau thì sẽ tạo nên mã mới phức tạp hơn, chính vì vậy phép thay thế được nối tiếp bằng phép hoán vị sẽ tạo nên mã mới khó hơn rất nhiều. Đây chính là chiếc cầu nối từ mã cổ điển sang mã hiện đại.

2.4 NHẬN XÉT HỆ MÃ CỔ ĐIỂN

Điểm yếu của mã cổ điển là:

- Phương pháp mã hoá cổ điển có thể dễ dàng bị giải mã bằng cách đoán chữ dựa trên phương pháp thống kê tần xuất xuất hiện các chữ cái trên mã và so sánh với bảng thống kê quan sát của bản rõ.
- Để dùng được mã hoá cổ điển thì bên mã hoá và bên giải mã phải thống nhất với nhau về cơ chế mã hoá cũng như giải mã. Nếu không thì hai bên sẽ không thể làm việc được với nhau.

TÓM TẮT

Mật mã đổi xứng sử dụng cùng một khóa cho việc mã hóa và giải mã. Các phương pháp mã hóa cổ điển thường dựa trên hai phương thức. Cách thứ nhất là dùng phương thức thay thế một chữ cái trong bản rõ thành một chữ cái khác trong bản mã. Các mã hóa dùng phương thức này là mã hóa Ceasar, mã hóa thay thế đơn bảng, đa bảng, độn một lần. Cách thứ hai là dùng phương thức hoán vị để thay đổi thứ tự ban đầu của các chữ cái trong bản rõ. Hai phương thức này cũng đóng vai trò quan trọng trong mã hóa đổi xứng hiện đại được trình bày trong chương tiếp theo.

CÂU HỎI ÔN TẬP

Câu 1: Tại sao khi gửi bản mã trên kênh truyền thì không sợ bị lộ thông tin?

Câu 2: Khóa là gì? Tại sao cần giữ bí mật khóa chỉ có người gửi và người nhận biết?

Câu 3: Tại sao lại gửi khóa qua kênh an toàn mà không gửi trực tiếp bản rõ trên kênh an toàn?

Câu 4: Phá mã theo hình thức vét cạn khóa thực hiện như thế nào? Cần làm gì để chống lại hình thức phá mã theo vét cạn khóa?

Câu 5: Các phương pháp Ceasar, mã hóa đơn bảng, đa bảng, one-time pad dùng nguyên tắc gì để mã hóa?

Câu 6: Phương pháp hoán vị dùng nguyên tắc gì để mã hóa?

Câu 7: Tại sao phương pháp mã hóa đơn bảng có thể bị tấn công phá mã dùng thống kê tần suất?

Câu 8: Cho biến đoạn mã sau dùng mã Cesar

"GCUA VQ DTGCM"

Suy luận tìm bản rõ.

Câu 9: Sử dụng kỹ thuật thám mã bảng chữ đơn, lập bảng tần suất các chữ, bộ chữ đôi, bộ chữ ba của đoạn mã sau:

UZQSOVUOHXMOPVGPOZPEVSGZWSZOPFPESXUDBMETSXAIUEPHZHMDZSH
ZOWSFPAPPDTSPQUZWYMXUZUHSXEPLYEPOPDZSZUFPOUDTMOHMQ

Lập luận và cho biết ánh xạ của bảng chữ đơn và đưa ra bản rõ phù hợp

Câu 10: Nếu thuật toán dùng bảng Saint Cyr để mã hóa và giải mã Vigenere khi biết từ khóa. Áp dụng thuật toán đó mã hóa bản rõ sau: "Network Security is very important for software development" với từ khóa là "COMPUTER SCIENCE"

Câu 11: Tại sao có thể nói "Có thể nói mã bộ đệm một lần là an toàn tuyệt đối, vì với bản rõ bất kỳ và bản mã bất kỳ, luôn tồn tại một khóa để ánh xạ bản rõ đó sang bản mã đã cho". Giải thích nhận định sau "Về mặt lý thuyết, xác suất để mọi mẫu tin (có cùng độ dài với bản rõ) trên bảng chữ mã là mã của một bản rõ cho trước là như nhau".

Câu 12: Tìm bản mã của bản rõ "We are studying cryptography this year" sử dụng mã Playfair với từ khóa "information technology".

Câu 13: Chứng minh rằng: tích của hai phép thay thế sẽ là một phép thay thế; tích của hai phép hoán vị sẽ là một phép hoán vị.

BÀI 3: MÃ HÓA ĐỔI XỨNG HIỆN ĐẠI

Sau khi học xong bài này, sinh viên hiểu:

- *Hiểu mã hóa khối hiện đại là kết quả của việc kết hợp các ưu điểm của các hệ mã khóa cổ điển.*
- *Hiểu nguyên tắc của hệ mã hóa dòng và hệ mã RC4*
- *Hiểu nguyên tắc của hệ mã hóa khối và hệ mã DES*
- *Giải pháp xây dựng ứng dụng mã hóa*

3.1 MÃ DÒNG (STREAM CIPHER)

Mã dòng có các đặc tính sau:

- Kích thước một đơn vị mã hóa: gồm k bit. Bản rõ được chia thành các đơn vị mã hóa: $P \rightarrow p_0p_1p_2\dots p_{n-1}$ (p_i : k bit)
- Một bộ sinh dãy số ngẫu nhiên: dùng một khóa K ban đầu để sinh ra các số ngẫu nhiên có kích thước bằng kích thước đơn vị mã hóa:

$$\text{StreamCipher}(k) \rightarrow S = s_0s_1s_2\dots s_{n-1} \quad (s_i: \text{k bit})$$

- Mỗi số ngẫu nhiên được XOR với đơn vị mã hóa của bản rõ để có bản mã.

$$C_0 = p_0 \oplus s_0, C_1 = p_1 \oplus s_1 \dots ; C = c_0c_1c_2\dots c_{n-1}$$

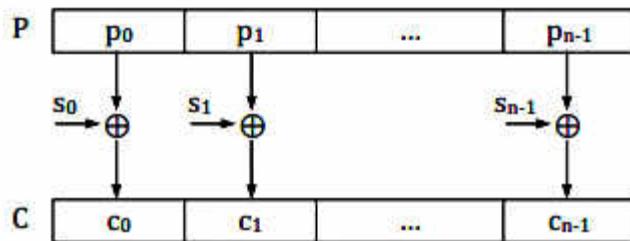
Quá trình giải mã được thực hiện ngược lại, bản mã C được XOR với dãy số ngẫu nhiên S để cho ra lại bản rõ ban đầu: $p_0 = c_0 \oplus s_0, p_1 = c_1 \oplus s_1, \dots$

Ví dụ: Đơn vị mã hóa có chiều dài k = 4 bit, n = 3:

Bản rõ: $p_0 = 1111, p_1 = 0000, p_2 = 0011$, Khóa: $s_0 = s_1 = s_2 = k = 0101$,

Bản mã: $c_0 = 1010, c_1 = 0101, c_2 = 0110$

Ví dụ này không phải là mã dòng vì s_0, s_1, s_2 lặp lại khóa K. Về phương diện khóa, ví dụ này giống mã Vigenere hơn. Đối với mã dòng, các số s_i được sinh ra phải đảm bảo một độ ngẫu nhiên nào đó (chu kỳ tuần hoàn dài):



Hình 3.1: Mô hình mã dòng

Như vậy có thể thấy mã hóa dòng là sự kết hợp hệ mã hóa Vigenere và mã hóa OTP. Điểm quan trọng nhất của các mã dòng là bộ sinh số ngẫu nhiên. Nếu chọn khóa có chiều dài ngắn như mã hóa Vigenere thì không bảo đảm an toàn, còn nếu chọn khóa có chiều dài bằng chiều dài bản tin như OTP thì lại không thực tế. Bộ sinh số của mã dòng cần bằng giữa hai điểm này, cho phép dùng một khóa ngắn nhưng dãy số sinh ra bảo đảm một độ ngẫu nhiên cần thiết như khóa của OTP. Phương pháp mã hóa dòng tiêu biểu là RC4. RC4 được dùng trong giao thức SSL, WEP.

3.1.1 Tiny RC4

Đơn vị mã hóa của TinyRC4 là 3 bít. TinyRC4 dùng 2 mảng S và T mỗi mảng gồm 8 số nguyên 3 bít. Khóa là một dãy gồm N số nguyên 3 bít. Bộ sinh số mỗi lần sinh ra 3 bít để sử dụng trong phép XOR. Quá trình sinh số của TinyRC4 gồm hai giai đoạn:

a. *Giai đoạn khởi tạo:*

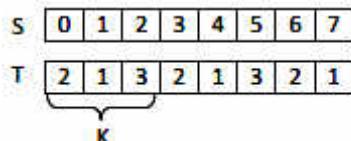
```

/* Khoi tao day so S va T */
for i = 0 to 7 do
    S[i] = i;
    T[i] = K[i mod N];
next i
/* Hoan vi day S */
j = 0;
for i = 0 to 7 do
    j = (j + S[i] + T[i]) mod 8;
    Swap(S[i], S[j]);
next i
    
```

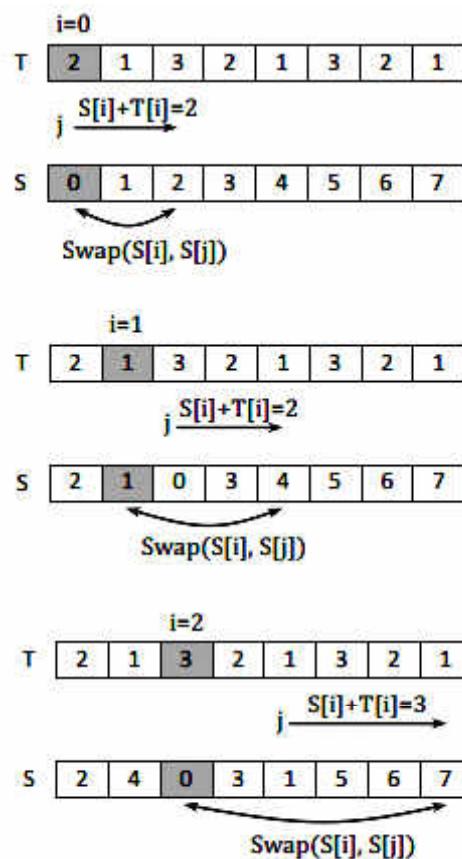
Trong giai đoạn này, trước tiên dãy S gồm các số nguyên 3 bít từ 0 đến 7 được sắp thứ tự tăng dần. Sau đó dựa trên các phần tử của khóa K, các phần tử của S được hoán vị lẫn nhau đến một mức độ ngẫu nhiên nào đó.

Ví dụ: mã hóa bản rõ $P = 001000110$ (từ "bag") với khóa K gồm 3 số 2, 1, 3 ($N=3$).

- Khởi tạo S và T :



- Hoán vị S



Quá trình thực hiện đến khi $i=7$ và lúc đó dãy S là 6 0 7 1 2 3 5 4

- b. Giai đoạn sinh số:

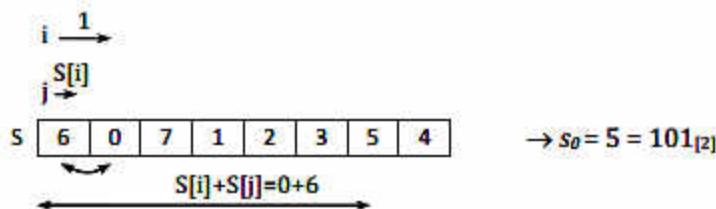
```

i, j = 0;
while (true)
    i = (i + 1) mod 8;
    j = (j + S[i]) mod 8;
    Swap (S[i], S[j]);
    t = (S[i] + S[j]) mod 8;
    k = S[t];
end while;

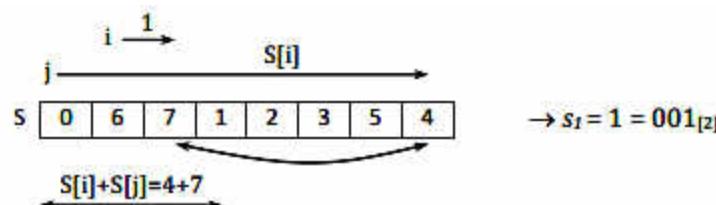
```

Trong giai đoạn này, các phần tử của S tiếp tục được hoán vị. Tại mỗi bước sinh số, hai phần tử của dãy S được chọn để tính ra số k 3 bít là số được dùng để XOR với đơn vị mã hóa của bản rõ. Tiếp tục ví dụ trên, quá trình sinh số mã hóa bản rõ "bag" thực hiện như sau:

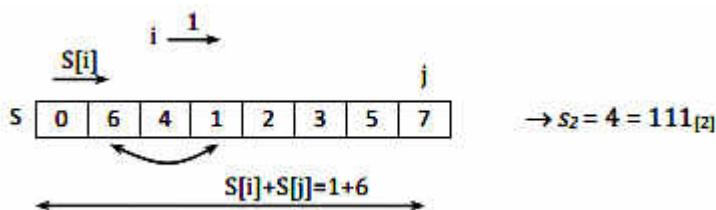
Bước 0:



Bước 1:



Bước 2:



Vậy bản mã là $C = 001.000.110 \oplus 101.001.111 = 100.001.001$ (từ EBB)

3.1.2 RC4

Cơ chế hoạt động của RC4 cũng giống như TinyRC4 với các đặc tính sau:

- Đơn vị mã hóa của RC4 là một byte 8 bít.
- Mảng S và T gồm 256 số nguyên 8 bít
- Khóa K là một dãy gồm N số nguyên 8 bít với N có thể lấy giá trị từ 1 đến 256.
- Bộ sinh số mỗi lần sinh ra một byte để sử dụng trong phép XOR.

Hai giai đoạn của RC4 là:

a. Giai đoạn khởi tạo:

```
/* Khoi tao day S va T*/
for i = 0 to 255 do
    S[i] = i;
    T[i] = K[i mod N];
next i
/* Hoan vi day S */
j = 0;
for i = 0 to 255 do
    j = (j + S[i] + T[i]) mod 256;
    Swap(S[i], S[j]);
next i
```

b. Giai đoạn sinh số:

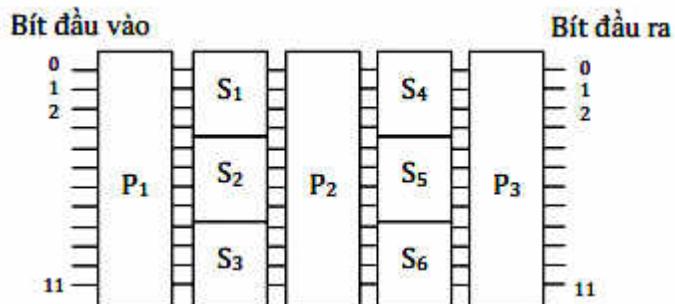
```
i, j = 0;
while (true)
    i = (i + 1) mod 256;
    j = (j + S[i]) mod 256;
    Swap (S[i], S[j]);
    t = (S[i] + S[j]) mod 256;
    k = S[t];
end while;
```

Quá trình sinh số của RC4 cũng sinh ra dãy số ngẫu nhiên, khó đoán trước, vì vậy RC4 đạt được mức độ an toàn cao theo tinh thần của mã hóa OTP. Mã hóa RC4 hoàn toàn được thực hiện trên các số nguyên một byte do đó tối ưu cho việc thiết lập bằng phần mềm và tốc độ thực hiện nhanh hơn so với mã khôi.

3.2 MÃ KHỐI (BLOCK CIPHER)

3.2.1 Mạng SPN

Kết hợp hai hay nhiều mã hóa đơn giản lại với nhau để tạo thành một mã hóa tổng (product cipher), trong đó mã hóa tổng an toàn hơn rất nhiều so với các mã hóa thành phần. Các mã hóa đơn giản thường là phép thay thế (substitution, S-box) và hoán vị (Permutation, P-box). Do đó người ta hay gọi mã hóa tổng là Substitution-Permutation Network (mạng SPN). Hình dưới minh họa một mạng SP.



Việc kết hợp các S-box và P-box tạo ra hai tính chất quan trọng của mã hóa là tính khuếch tán (diffusion) và tính gây lẫn (confusion). Hai tính chất này do Claude Shannon giới thiệu vào năm 1946, và là cơ sở của tất cả các mã khối hiện nay.

- *Tính khuếch tán*: một bít của bản rõ tác động đến tất cả các bít của bản mã, hay nói cách khác, một bít của bản mã chịu tác động của tất cả các bít trong bản rõ. Việc làm như vậy nhằm làm giảm tối đa mối liên quan giữa bản rõ và bản mã, ngăn chặn việc suy ra lại khóa. Tính chất này có được dựa vào sử dụng P-box kết hợp S-box.
- *Tính gây lẫn*: làm phức tạp hóa mối liên quan giữa bản mã và khóa. Do đó cũng ngăn chặn việc suy ra lại khóa. Tính chất này có được dựa vào sử dụng S-box.

3.2.2 Mô Hình Mã Feistel

Mô hình mã Feistel là một dạng tiếp cận khác so với mạng SP. Mô hình do Horst Feistel đề xuất, cũng là sự kết hợp các phép thay thế và hoán vị. Trong hệ mã Feistel, bản rõ sẽ được biến đổi qua một số vòng để cho ra bản mã cuối cùng:

$$P \xrightarrow{K_1} C_1 \xrightarrow{K_2} C_2 \xrightarrow{K_3} \dots \xrightarrow{K_{n-1}} C_n$$

Trong đó bản rõ P và các bản mã C_i được chia thành nửa trái và nửa phải:

$$P = (L_0, R_0)$$

$$C_i = (L_i, R_i) \quad i = 1, 2, \dots, n$$

Quy tắc biến đổi các nửa trái phải này qua các vòng được thực hiện như sau:

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

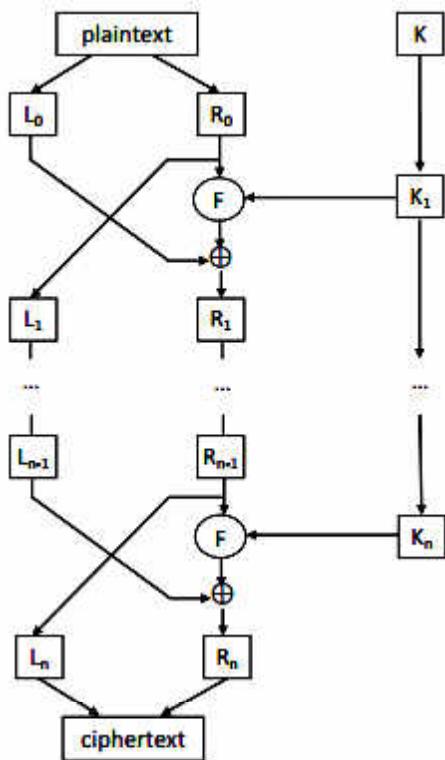
$$R_i = L_{i-1} \oplus F(R_{i-1}, K_i)$$

K_i là một khóa con cho vòng thứ i . Khóa con này được sinh ra từ khóa K ban đầu theo một thuật toán sinh khóa con (key schedule): $K \rightarrow K_1 \rightarrow K_2 \rightarrow \dots \rightarrow K_n$

F là một hàm mã hóa dùng chung cho tất cả các vòng. Hàm F đóng vai trò như là phép thay thế còn việc hoán đổi các nửa trái phải có vai trò hoán vị. Bản mã C được tính từ kết xuất của vòng cuối cùng:

$$C = C_n = (L_n, R_n)$$

Sơ đồ tính toán của hệ mã Feistel được thể hiện trong hình bên dưới:



Hình 3.2: Mô hình mã khối Feistel

Để giải mã quá trình được thực hiện qua các vòng theo thứ tự ngược lại:

$$C \rightarrow L_n, R_n$$

$$R_{i-1} = L_i \quad (\text{theo mã hóa } L_i = R_{i-1})$$

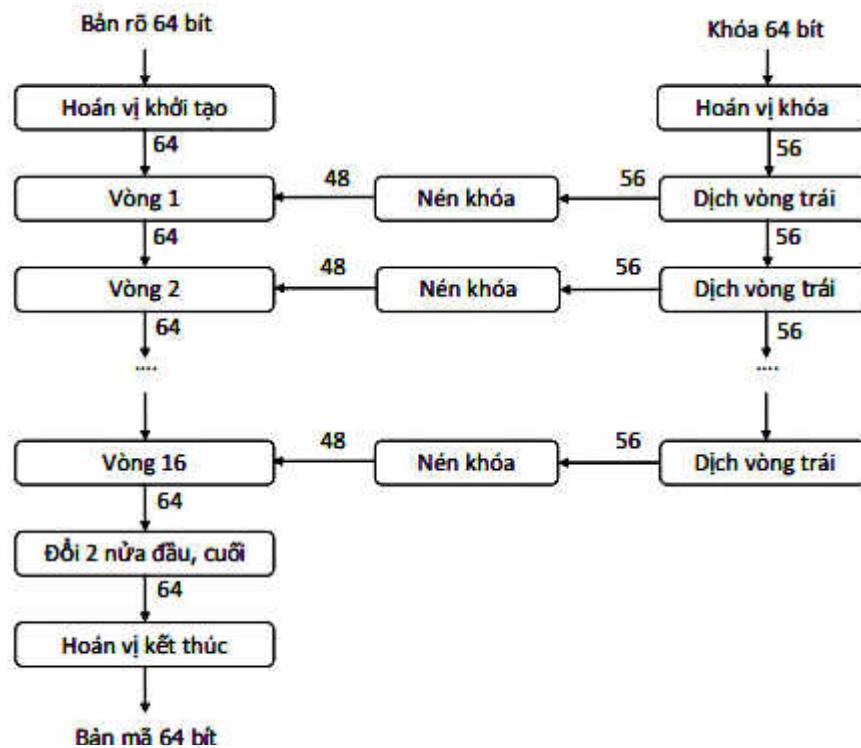
$$L_{i-1} = R_i \oplus F(R_{i-1}, K_i) \quad (\text{theo mã hóa } R_i = L_{i-1} F(R_{i-1}, K_i))$$

Và cuối cùng bản rõ là $P = (L_0, R_0)$.

Hệ mã Feistel có điểm quan trọng là việc chia các bản mã thành hai nửa trái phải giúp cho hàm F không cần khả nghịch (không cần có F^{-1}). Mã hóa và giải mã đều dùng chiểu thuận của hàm F . Hàm F và thuật toán sinh khóa con càng phức tạp thì càng khó phá mã.

Ứng với các hàm F và thuật toán sinh khóa con khác nhau thì ta sẽ có các phương pháp mã hóa khác nhau, phần tiếp theo sẽ trình bày mã hóa DES, là một phương pháp mã hóa dựa trên nguyên tắc của hệ mã Feistel.

3.3 MÃ DES



Hình 3.3: Cấu trúc một vòng của mã DES

Mã DES (DATA ENCRYPTION STANDARD) có các tính chất sau:

- Là mã thuộc hệ mã Feistel gồm 16 vòng, ngoài ra DES có thêm một hoán vị khởi tạo trước khi vào vòng 1 và một hoán vị khởi tạo sau vòng 16
- Kích thước của khối là 64 bít: ví dụ bản tin “meetmeafterthetogaparty” biểu diễn theo mã ASCII thì mã DES sẽ mã hóa làm 3 lần, mỗi lần 8 chữ cái (64 bít): *meetmeaf - tertheto - gaparty*.
- Kích thước khóa là 56 bít
- Mỗi vòng của DES dùng khóa con có kích thước 48 bít được trích ra từ khóa chính.

Hình 3-6 minh họa các vòng của mã DES. Sơ đồ mã DES trên gồm ba phần, phần thứ nhất là các hoán vị khởi tạo và hoán vị kết thúc. Phần thứ hai là các vòng Feistel, phần thứ ba là thuật toán sinh khóa con. Chúng ta sẽ đi vào chi tiết của từng phần.

3.3.1 Hoán Vị Khởi Tạo Và Hoán Vị Kết Thúc

Ta đánh số các bít của khối 64 bít theo thứ tự từ trái sang phải là 0, 1, ..., 62, 63:
 $b_0b_1b_2\dots b_{62}b_{63}$

Hoán vị khởi tạo sẽ hoán đổi các bít theo quy tắc sau:

57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7
56	48	40	32	24	16	8	0
58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6

$(b_0b_1b_2\dots b_{62}b_{63} \rightarrow b_{57}b_{49}b_{41}\dots b_{14}b_6)$

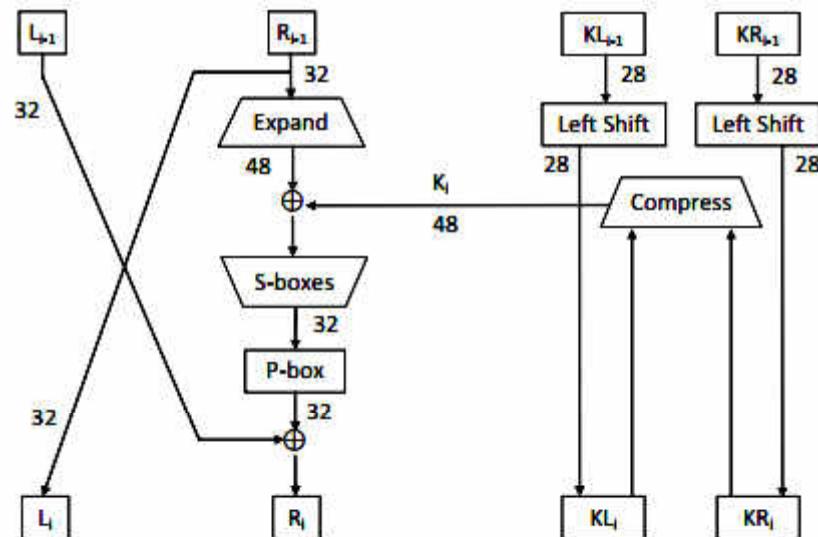
Hoán vị kết thúc hoán đổi các bít theo quy tắc sau:

39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25
32	0	40	8	48	16	56	24

Hoán vị kết thúc chính là hoán vị nghịch đảo của hoán vị khởi tạo.

3.3.2 Các Vòng Của DES

Hình sau minh họa một vòng Feistel của DES



Hình 3.4: Cấu trúc một vòng của mã DES

Trong DES, hàm F của Feistel là:

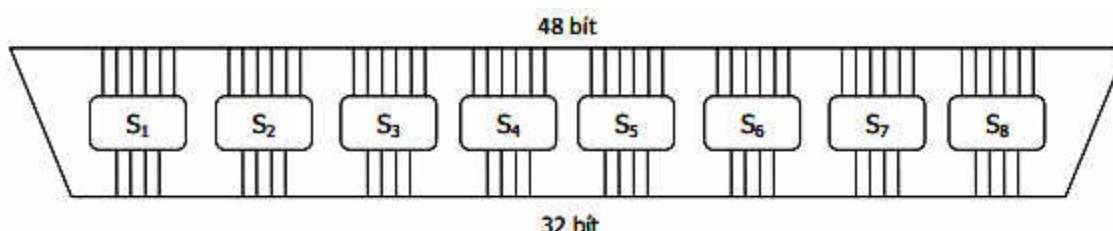
$$F(R_{i-1}, K_i) = P\text{-box}(S\text{-boxes}(E\text{-expand}(R_{i-1}) \oplus K_i))$$

Trong đó hàm Expand vừa mở rộng vừa hoán vị R_{i-1} từ 32 bít lên 48 bít. Hàm Sboxes nén 48 bít lại còn 32 bít. Hàm P-box là một hoán vị 32 bít. Mô tả của các hàm trên là như sau:

- *Expand*: đánh số các bít của R_{i-1} theo thứ tự từ trái sang phải là 0, 1, 2, ..., 31. Hàm Expand thực hiện vừa hoán vị vừa mở rộng 32 bít thành 48 bít theo quy tắc:

31	0	1	2	3	4
3	4	5	6	7	8
7	8	9	10	11	12
11	12	13	14	15	16
15	16	17	18	19	20
19	20	21	22	23	24
23	24	25	26	27	28
27	28	29	30	31	0

- *S-box*: Hàm S-boxes của DES biến đổi một số 48 bít thành một số 32 bít. Gồm 8 hàm S-box, mỗi hàm biến đổi số 6 bít thành số 4 bít (hình dưới)



Hàm S-box có nội dung như sau:

		$b_1 b_2 b_3 b_4$															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$b_0 b_5$	0	E	4	D	1	2	F	B	8	3	A	6	C	5	9	0	7
	1	0	F	7	4	E	2	D	1	A	6	C	B	9	5	3	8
	2	4	1	E	8	D	6	2	B	F	C	9	7	3	A	5	0
	3	F	C	8	2	4	9	1	7	5	B	3	E	A	0	6	D

DES S-box 1

		$b_1 b_2 b_3 b_4$															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$b_0 b_5$	0	F	1	8	E	6	B	3	4	9	7	2	D	C	0	5	A
	1	3	D	4	7	F	2	8	E	C	0	1	A	6	9	B	5
	2	0	E	7	B	A	4	D	1	5	8	C	6	9	3	2	F
	3	D	8	A	1	3	F	4	2	B	6	7	C	0	5	E	9

DES S-box 2

		$b_1b_2b_3b_4$															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
b_0b_5	0	A	0	9	E	6	3	F	5	1	D	C	7	B	4	2	8
	1	D	7	0	9	3	4	6	A	2	8	5	E	C	B	F	1
	2	D	6	4	9	8	F	3	0	B	1	2	C	5	A	E	7
	3	1	A	D	0	6	9	8	7	4	F	E	3	B	5	2	C

DES S-box 3

		$b_1b_2b_3b_4$															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
b_0b_5	0	7	D	E	3	0	6	9	A	1	2	8	5	B	C	4	F
	1	D	8	B	5	6	F	0	3	4	7	2	C	1	A	E	9
	2	A	6	9	0	C	B	7	D	F	1	3	E	5	2	8	4
	3	3	F	0	6	A	1	D	8	9	4	5	B	C	7	2	E

DES S-box 4

		$b_1b_2b_3b_4$															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
b_0b_5	0	2	C	4	1	7	A	B	6	8	5	3	F	D	0	E	9
	1	E	B	2	C	4	7	D	1	5	0	F	A	3	9	8	6
	2	4	2	1	B	A	D	7	8	F	9	C	5	6	3	0	E
	3	B	8	C	7	1	E	2	D	6	F	0	9	A	4	5	3

DES S-box 5

		$b_1b_2b_3b_4$															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
b_0b_5	0	C	1	A	F	9	2	6	8	0	D	3	4	E	7	5	B
	1	A	F	4	2	7	C	9	5	6	1	D	E	0	B	3	8
	2	9	E	F	5	2	8	C	3	7	0	4	A	1	D	B	6
	3	4	3	2	C	9	5	F	A	B	E	1	7	6	0	8	D

DES S-box 6

		$b_1b_2b_3b_4$															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
b_0b_5	0	4	B	2	E	F	0	8	D	3	C	9	7	5	A	6	1
	1	D	0	B	7	4	9	1	A	E	3	5	C	2	F	8	6
	2	1	4	B	D	C	3	7	E	A	F	6	8	0	5	9	2
	3	6	B	D	8	1	4	A	7	9	5	0	F	E	2	3	C

DES S-box 7

		$b_1 b_2 b_3 b_4$															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$b_0 b_5$	0	D	2	8	4	6	F	B	1	A	9	3	E	5	0	C	7
	1	1	F	D	8	A	3	7	4	C	5	6	B	0	E	9	2
	2	7	B	4	1	9	C	E	2	0	6	A	D	F	3	5	8
	3	2	1	E	7	4	A	8	D	F	C	9	0	3	5	6	B

DES S-box 8

Sự phức tạp của S-boxes là yếu tố chính làm cho DES có độ an toàn cao.

- *P-box*: hàm P-box cũng thực hiện hoán vị 32 bít đầu vào theo quy tắc:

15	6	19	20	28	11	27	16
0	14	22	25	4	17	30	9
1	7	23	13	31	26	2	8
18	12	29	5	21	10	3	24

3.3.3 Thuật Toán Sinh Khóa Con Của Des

Khóa K 64 bít ban đầu được rút trích và hoán vị thành một khóa 56 bít (tức chỉ sử dụng 56 bít) theo quy tắc:

56	48	40	32	24	16	8
0	57	49	41	33	25	17
9	1	58	50	42	34	26
18	10	2	59	51	43	35
62	54	46	38	30	22	14
6	61	53	45	37	29	21
13	5	60	52	44	36	28
20	12	4	27	19	11	3

56 bít

Khóa 56 bít này được chia thành 2 nửa trái phải KL_0 và KR_0 , mỗi nửa có kích thước 28 bít. Tại vòng thứ i ($i = 1, 2, 3, \dots, 16$), KL_{i-1} và KR_{i-1} được dịch vòng trái r_i bít để có được KL_i và KR_i , với r_i được định nghĩa:

$$r_i = \begin{cases} 1 & \text{nếu } i \in \{1, 2, 9, 16\} \\ 2 & \text{với những } i \text{ khác} \end{cases}$$

Cuối cùng khóa K_i của mỗi vòng được tạo ra bằng cách hoán vị và nén 56 bít của KL_i và KR_i thành 48 bít theo quy tắc:

3.3.4 Hiệu Ứng Lan Truyền (Avalanche Effect)

Một tính chất quan trọng cần thiết của mọi thuật toán mã hóa là chỉ cần một thay đổi nhỏ trong bản rõ hay trong khóa sẽ dẫn đến thay đổi lớn trong bản mã. Cụ thể, chỉ cần thay đổi một bít trong bản rõ hay khóa thì dẫn đến sự thay đổi của nhiều bít bản mã. Tính chất này được gọi là hiệu ứng lan truyền. Nhờ có tính chất này mà người phá mã không thể giới hạn miền tìm kiếm của bản rõ hay của khóa nên phải thực hiện vét cạn khóa.

3.3.5 Phá mã Des

Tấn công vét cạn khóa (Brute Force Attack): Vì khóa của mã DES có chiều dài là 56 bít nên để tiến hành brute-force attack, cần kiểm tra 2^{56} khóa khác nhau. Hiện nay với những thiết bị phổ dụng, thời gian gian để thử khóa là rất lớn nên việc phá mã là không khả thi (xem bảng). Tuy nhiên vào năm 1998, tổ chức Electronic Frontier Foundation (EFF) thông báo đã xây dựng được một thiết bị phá mã DES gồm nhiều máy tính chạy song song, trị giá khoảng 250.000\$. Thời gian thử khóa là 3 ngày. Hiện nay mã DES vẫn còn được sử dụng trong thương mại, tuy nhiên người ta đã bắt đầu áp dụng những phương pháp mã hóa khác có chiều dài khóa lớn hơn (128 bít hay 256 bít) như TripleDES hoặc AES.

Phá mã DES theo phương pháp vi sai (differential cryptanalysis): Năm 1990 Biham và Shamir đã giới thiệu phương pháp phá mã vi sai. Phương pháp vi sai tìm khóa ít tốn thời gian hơn brute-force. Tuy nhiên phương pháp phá mã này lại đòi hỏi phải có 2^{47} cặp bản rõ - bản mã được lựa chọn (chosen-plaintext). Vì vậy phương pháp này là bất khả thi dù rằng số lần thử có thể ít hơn phương pháp brute-force.

Phá mã DES theo phương pháp thử tuyến tính (linear cryptanalysis): Năm 1997 Matsui đưa ra phương pháp phá mã tuyến tính. Trong phương pháp này, cần phải biết trước 2^{43} cặp bản rõ-bản mã (known-plaintext). Tuy nhiên 2^{43} cũng là một con số lớn nên phá mã tuyến tính cũng không phải là một phương pháp khả thi.

3.4 MỘT SỐ PHƯƠNG PHÁP MÃ KHỐI KHÁC

3.4.1 Triple Des

Một trong những cách để khắc phục yếu điểm kích thước khóa ngắn của mã hóa DES là sử dụng mã hóa DES nhiều lần với các khóa khác nhau cho cùng một bản tin. Đơn giản nhất là dùng DES hai lần với hai khóa khác nhau, cách thức này được gọi là Double DES:

$$C = E(E(P, K_1), K_2)$$

Điều này giống như là Double DES dùng một khóa có kích thước là 112 byte, chỉ có một hạn chế là tốc độ chậm hơn DES vì phải dùng DES hai lần. Tuy nhiên người ta đã tìm được một phương pháp tấn công Double DES có tên gọi là gấp-nhau-ở-giữa (meet-in-the middle). Đây là một phương pháp tấn công chọn bản rõ.

Vì vậy người ta chọn dùng DES ba lần với ba khóa khác nhau, cách thức này được gọi là Triple DES:

$$C = E(E(E(P, K_1), K_2), K_3)$$

Chiều dài khóa là 168 bít sẽ gây phức tạp hơn nhiều cho việc phá mã bằng phương pháp tấn công gấp-nhau-ở-giữa. Trong thực tế người ta chỉ dùng Triple DES với hai khóa K_1, K_2 mà vẫn đảm bảo độ an toàn cần thiết. Công thức như sau:

$$C = E(D(E(P, K_1), K_2), K_1)$$

Nguyên nhân của việc dùng EDE thay cho EEE là nếu với $K_1 = K_2 = K$ thì

$$C = E(D(E(P, K), K), K) = E(P, K)$$

Nghĩa là Triple DES suy giảm thành một DES đơn.

3.4.2 Advanced Encryption Standard (AES)

Vào những năm 1990, nhận thấy nguy cơ của mã hóa DES là kích thước khóa ngắn, có thể bị phá mã trong tương lai gần, Cục tiêu chuẩn quốc gia Hoa Kỳ đã kêu gọi xây dựng một phương pháp mã hóa mới. Cuối cùng một thuật toán có tên là

Rijndael được chọn và đổi tên thành Andvanced Encryption Standard hay AES. Có thể nói rằng mã hóa AES với khóa có kích thước 256 bit là an toàn tuyệt đối.

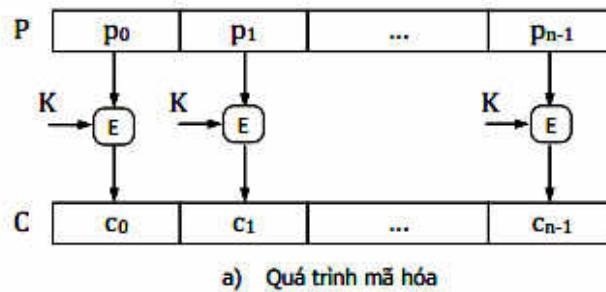
- Đặc điểm hệ mã này như sau: An ninh hơn và nhanh hơn 3DES
- Kích thước khối: 128 bit
- Kích thước khóa: 128/192/256 bit
- Số vòng: 10/12/14
- Cấu trúc mạng S-P, nhưng không theo hệ Feistel

3.5 CÁC MÔ HÌNH ỨNG DỤNG MÃ KHỐI

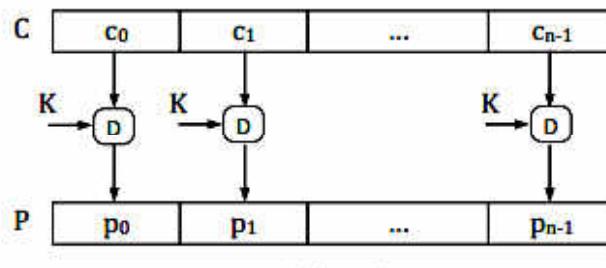
Mã khối (như mã DES) được áp dụng để mã hóa một khối dữ liệu có kích thước xác định. Để mã hóa một bản tin dài, bản tin được chia ra thành nhiều khối ($P = P_0P_1P_2\dots P_{n-1}$) và áp dụng mã khối cho từng khối một. Có nhiều mô hình áp dụng mã khối là ECB, CBC, CTR, OFB và CFB.

3.5.1 Electronic Codebook – ECB

Trong mô hình ECB, mỗi khối được mã hóa một cách riêng rẽ, dùng chung một khóa K.



a) Quá trình mã hóa



b) Quá trình giải mã

Hình 3.5: Mô hình ECB của mã khối

Trong mã hóa ECB, nếu $P_i = P_j$ thì $C_i = C_j$ và ngược lại. Có thể thấy rằng mã ECB tương tự như mã hóa đơn bảng cổ điển, trong đó P_i và C_i giống như là các chữ cái, còn khóa K cùng với mã khối giống như là một phép thế. Do đó, phá mã ECB dựa vào một số đặc tính thống kê của dữ liệu, giống như dùng thống kê tần suất chữ cái để phá mã mã hóa đơn bảng. Vì vậy mã hóa ECB chỉ thích hợp để mã hóa những bản tin ngắn.

Để minh họa đặc tính thống kê của dữ liệu, hình trên thể hiện một tấm ảnh được mã hóa bằng ECB. Dù rằng mỗi khối đã được biến đổi qua phép mã hóa, tuy nhiên nhìn tổng thể thì vẫn tồn tại một sự liên hệ nào đó giữa các khối.

3.5.2 Cipher Block Chaining – CBC

Trong mô hình CBC, bản mã của một lần mã hóa được sử dụng cho lần mã hóa tiếp theo:

$$C_i = E(P_i \oplus C_{i-1}, K) \text{ với } i = 1, 2, 3 \dots n-1$$

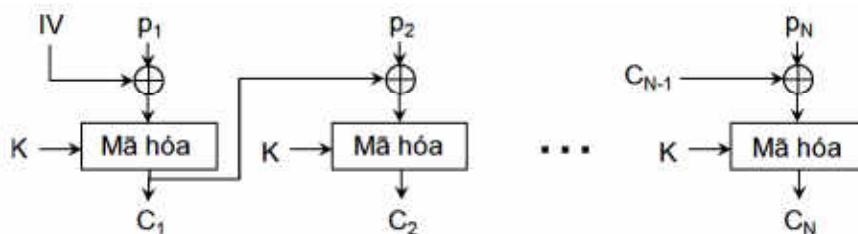
Do đó để mã hóa khối đầu tiên, một khối dữ liệu giả được sử dụng, gọi là vector khởi tạo (initialization vector – IV). Vectơ này được chọn ngẫu nhiên:

$$C_0 = E(P_0 \oplus IV, K)$$

Để giải mã, tiến hành ngược lại:

$$P_0 = D(C_0, K) \oplus IV$$

$$P_i = D(C_i, K) \oplus C_{i-1} \text{ với } i = 1, 2, \dots n-1$$

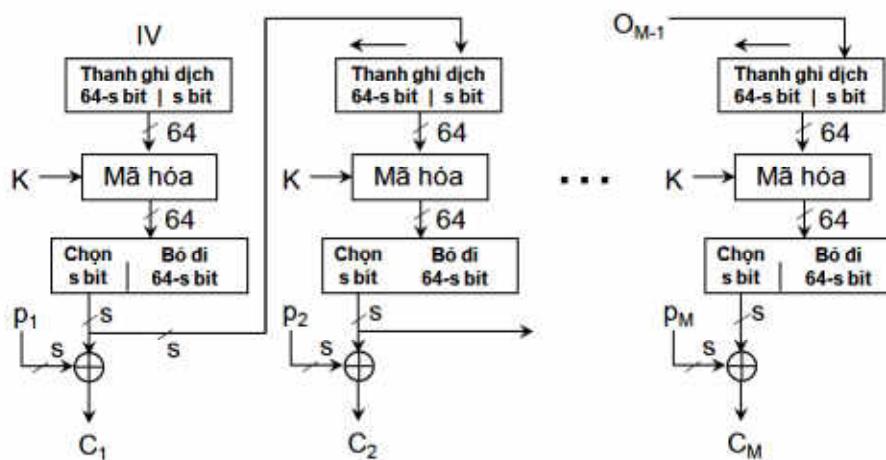


Hình 3.6: Mô hình CBC của mã khối

Người mã hóa và người giải mã phải dùng chung vector khởi tạo IV. Vector khởi tạo không cần giữ bí mật nên thường được gắn vào trước bản mã trước khi truyền thông điệp ($IVC_0C_1C_2\dots C_{n-1}$).

Nội dung của bản mã C_i không chỉ phụ thuộc vào bản rõ P_i mà còn phụ thuộc vào tất cả các bản rõ đứng trước và IV. Do đó nếu có hai bản rõ giống nhau thì hai bản mã sẽ không giống nhau (do IV ngẫu nhiên). Điều này khắc phục được hạn chế của mô hình ECB, từ bản mã người phá mã không thể phát hiện ra những đặc tính thống kê của dữ liệu. Việc giải mã, bản rõ P_i không chỉ phụ thuộc vào bản mã C_i mà còn phụ thuộc vào bản mã C_{i-1} đứng trước. Do đó nếu xảy lỗi trên đường truyền, chỉ cần một bít bị hỏng thì dẫn đến không thể giải mã được bản mã đó và bản mã tiếp theo sau.

3.5.3 Cipher Feedback- CFB



Hình 3.7: Mô hình CFB của mã khối

Mô hình OFB dùng s bít của khóa do bộ sinh khóa tạo ra để ghép với IV cho lần mã hóa tiếp theo:

```

Register = IV;
for i = 0 to n-1 do
    Ti= E(Register, K);
    Ti= TiSHR (b-s);           // lấy s bít đầu của Ti
    Ci= Pi XOR Ti;
    Register = Register SHL s;
    Register = Register OR Ci;
next i

```

Do đó giống như mô hình CBC, có thể thấy rằng nội dung của bản mã C_i không chỉ phụ thuộc vào bản rõ P_i mà còn phụ thuộc vào tất cả các bản rõ đứng trước và IV. Ngược lại, đối với việc giải mã, bản rõ P_i không chỉ phụ thuộc vào bản mã C_i mà còn phụ thuộc vào bản mã C_{i-1} đứng trước.

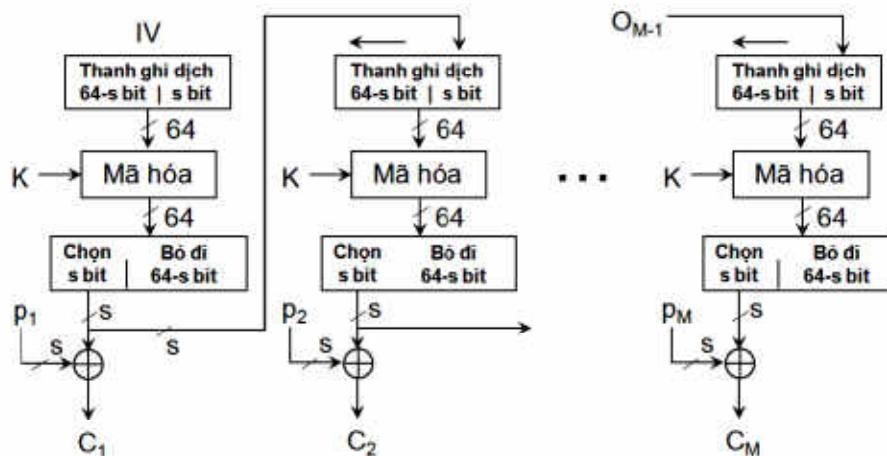
3.5.4 Output Feedback – OFB

Tương tự CFB chỉ khác là phản hồi lấy từ đầu ra giải thuật mã hóa, độc lập với thông báo. Không bao giờ sử dụng lại cùng khóa và IV. Lỗi truyền 1 khối mã hóa không ảnh hưởng đến các khối khác. Thông báo dễ bị sửa đổi nội dung. Chỉ nên dùng OFB-64. Có thể tiết kiệm thời gian bằng cách thực hiện giải thuật mã hóa trước khi nhận được dữ liệu.

```

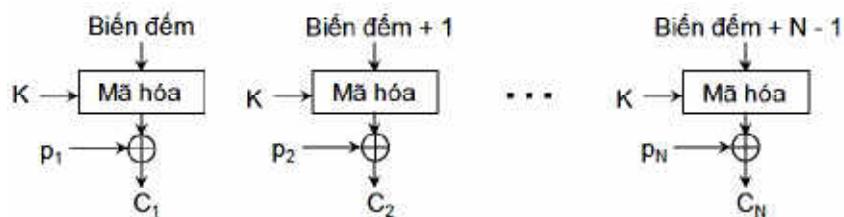
Register = IV;
for i = 0 to n-1 do
    Ti= E(Register, K);
    Ti= TiSHR (b-s);           // lấy s bit đầu của Ti
    Ci= Pi XOR Ti;
    Register = Register SHL s;
    Register = Register OR Ti;
next i

```



Hình 3.8: Mô hình OFB của mã khối

3.5.5 Counter – CTR



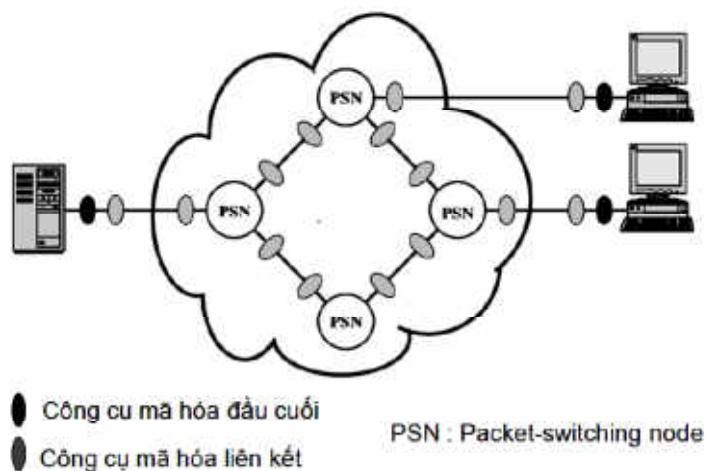
Hình 3.9: Mô hình mã hóa khối CTR

Mô hình này có những ưu điểm như sau:

- Hiệu quả cao
- Có thể thực hiện mã hóa (hoặc giải mã) song song
- Có thể thực hiện giải thuật mã hóa trước nếu cần.
- Có thể xử lý bất kỳ khối nào trước các khối khác.
- An ninh không kém gì các phương thức khác.
- Đơn giản, chỉ cần cài đặt giải thuật mã hóa, không cần đến giải thuật giải mã.
- Không bao giờ sử dụng lại cùng giá trị khóa và biến đếm (tương tự OFB)

3.6 BỐ TRÍ CÔNG CỤ MÃ HÓA

Giải pháp hữu hiệu và phổ biến nhất chống lại các mối đe dọa đến an ninh mạng là mã hóa. Để thực hiện mã hóa, cần xác định: Mã hóa những gì và Thực hiện mã hóa ở đâu. Có 2 phương án cơ bản: Mã hóa liên kết và Mã hóa đầu cuối



Hình 3.10: Minh họa mã cách bố trí công cụ mã hóa

3.6.1 Mã hóa liên kết

Công cụ mã hóa được sắp đặt ở 2 đầu của mọi liên kết có nguy cơ bị tấn công. Đảm bảo an ninh việc lưu chuyển thông tin trên tất cả các liên kết mạng. Các mạng lớn cần đến rất nhiều công cụ mã hóa. Cần cung cấp rất nhiều khóa. Nguy cơ bị tấn công tại mỗi chuyển mạch. Các gói tin cần được mã hóa mỗi khi đi vào một chuyển mạch gói để đọc được địa chỉ ở phần đầu. Thực hiện ở tầng vật lý hoặc tầng liên kết

3.6.2 Mã hóa đầu cuối

Quá trình mã hóa được thực hiện ở 2 hệ thống đầu cuối. Đảm bảo an ninh dữ liệu người dung. Chỉ cần một khóa cho 2 đầu cuối. Đảm bảo xác thực ở mức độ nhất định. Mẫu lưu chuyển thông tin không được bảo vệ. Các phần đầu gói tin cần được truyền tải tường minh. Thực hiện ở tầng mạng trở lên. Càng lên cao càng ít thông tin cần mã hóa và càng an ninh nhưng càng phức tạp với nhiều thực thể và khóa

3.7 QUẢN LÝ TRAO ĐỔI KHÓA BÍ MẬT

Vấn đề đối với mã hóa đổi xứng là làm sao phân phối khóa an ninh đến các bên truyền tin. Thường hệ thống mất an ninh là do không quản lý tốt việc phân phối khóa bí mật. Phân cấp khóa: Khóa phiên (tạm thời) và Khóa chủ (lâu dài). Khóa phiên dùng mã hóa dữ liệu trong một phiên kết nối. Hủy bỏ khi hết phiên. Khóa chủ (lâu dài) dùng để mã hóa các khóa phiên, đảm bảo phân phối chúng một cách an ninh.

3.7.1 Các cách trao đổi khóa

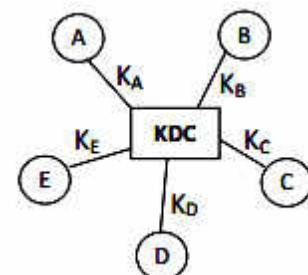
Có 4 cách phân phối khóa:

1. Khóa có thể được chọn bởi bên A và gửi theo đường vật lý đến bên B
2. Khóa có thể được chọn bởi một bên thứ ba, sau đó gửi theo đường vật lý đến A và B
3. Nếu A và B đã có một khóa dùng chung thì một bên có thể gửi khóa mới đến bên kia, sử dụng khóa cũ để mã hóa khóa mới
4. Nếu mỗi bên A và B đều có một kênh mã hóa đến một bên thứ ba C thì C có thể gửi khóa theo các kênh mã hóa đó đến A và B

Trong 4 cách phân phối này, giải pháp (4) là được sử dụng an toàn nhất.

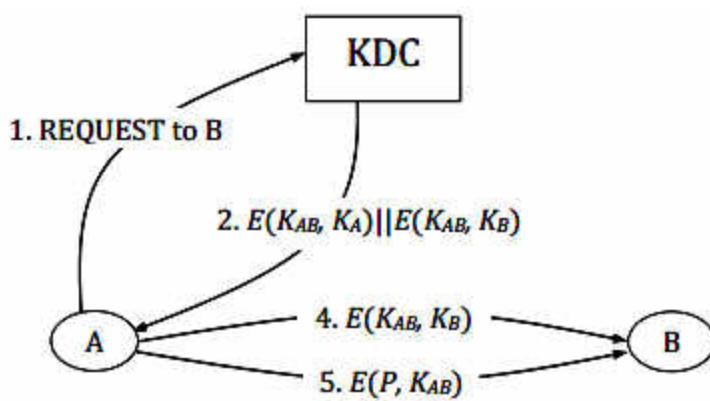
3.7.2 Phương pháp trao đổi khóa bằng trung tâm phân phối khóa

Phương pháp trao đổi khóa dựa vào trung tâm phân phối khóa KDC (Key Distribution Center – KDC) thể hiện giải pháp trao đổi khóa (4). Trong mô hình sử dụng KDC, mỗi người sử dụng chỉ cần có một khóa bí mật với KDC. Còn khóa dùng để trao đổi dữ liệu giữa các người sử dụng sẽ do KDC cung cấp.



Giả sử Alice có khóa bí mật K_A với KDC và Bob có khóa bí mật K_B với KDC. Bây giờ Alice muốn trao đổi dữ liệu với Bob. Quá trình thiết lập khóa chung K_{AB} giữa Alice và Bob gồm các bước:

1. Alice gửi yêu cầu muốn trao đổi dữ liệu với Bob cho KDC.
2. KDC tạo một khóa bí mật K_{AB} và mã hóa thành hai bản mã. Một bản mã được mã hóa bằng khóa bí mật của Alice $E(K_{AB}, K_A)$ và một bản mã được mã hóa bằng khóa bí mật của Bob $E(K_{AB}, K_B)$.
3. Alice giải mã $E(K_{AB}, K_A)$ để có K_{AB}
4. Alice gửi $E(K_{AB}, K_B)$ cho Bob, Bob giải mã để có được K_{AB}
5. Alice và Bob trao đổi dữ liệu qua khóa bí mật K_{AB}



Hình 3.1: Trao đổi khóa bí mật dùng KDC

Như vậy, khóa K_{AB} chỉ có KDC, Alice và Bob biết. Trách nhiệm của KDC là giữ bí mật khóa này. Alice và Bob dùng khóa K_{AB} để mã hóa dữ liệu. Khi kết thúc quá trình truyền dữ liệu, K_{AB} được hủy bỏ. Lần sau nếu Alice lại truyền số liệu với Bob thì KDC

sẽ cung cấp khóa K_{AB} khác. Như vậy chỉ cần Alice có thiết lập khóa bí mật K_A với KDC thì Alice có thể truyền số liệu không chỉ với Bob mà còn với những người khác.

Một khái niệm quan trọng khác có thể rút ra từ mô hình dùng KDC là khái niệm khóa chủ và khóa phiên (master key và session key). Trong ví dụ trên các khóa K_A , K_B không được sử dụng trực tiếp để mã hóa dữ liệu, chúng chỉ được dùng để mã hóa các khóa tạm K_{AB} . Các khóa K_{AB} này mới trực tiếp mã hóa dữ liệu và bị hủy bỏ khi sau quá trình truyền dữ liệu kết thúc. Vì vậy K_A , K_B được gọi là khóa chủ, chúng ít được sử dụng nên người phá mã khó có cơ hội thu thập bản mã để phá mã. Khóa K_A , K_B được sử dụng lâu dài. Còn K_{AB} được gọi là khóa phiên, K_{AB} chỉ tồn tại trong một phiên truyền dữ liệu duy nhất mà thôi.

Chương 7 trình bày giao thức Kerberos, là một giao thức dựa trên khái niệm trung tâm phân phối khóa. Kerberos được sử dụng trong các hệ điều hành ngày nay, để mã hóa dữ liệu trong mạng cục bộ LAN.

TÓM TẮT

Bài học cung cấp cho sinh viên kiến thức về hệ mã dòng RC4, hệ mã khối DES, các mô hình áp dụng các hệ mã này, cách bố trí các ứng dụng mã hóa và giải pháp trao đổi khoa mật.

Mã dòng có các đặc tính sau: Kích thước một đơn vị mã hóa gồm k bit. Bản rõ được chia thành các đơn vị mã hóa k bit. Một bộ sinh dãy số ngẫu nhiên dung khóa K ban đầu để sinh ra các số ngẫu nhiên có kích thước bằng kích thước đơn vị mã hóa. Mỗi số ngẫu nhiên được XOR với đơn vị mã hóa của bản rõ để có bản mã.

Mạng S-P là giải pháp mã hóa khối. Việc kết hợp các S-box và P-box tạo ra hai tính chất quan trọng của mã hóa là tính khuếch tán (diffusion) và tính gây lẫn (confusion).

Mô hình mã Feistel là một dạng tiếp cận khác so với mạng SP. Mô hình do Horst Feistel đề xuất, cũng là sự kết hợp các phép thay thế và hoán vị. Trong hệ mã Feistel, bản rõ sẽ được biến đổi qua một số vòng để cho ra bản mã cuối cùng.

Mã DES (DATA ENCRYPTION STANDARD) là hệ mã thuộc hệ mã Feistel gồm 16 vòng, ngoài ra DES có thêm một hoán vị khởi tạo trước khi vào vòng 1 và một hoán vị

khởi tạo sau vòng 16. Kích thước của khối là 64 bít. Kích thước khóa là 56 bít. Mỗi vòng của DES dùng khóa con có kích thước 48 bít được trích ra từ khóa chính. Mã khối (như mã DES) được áp dụng để mã hóa một khối dữ liệu có kích thước xác định. Để mã hóa một bản tin dài, bản tin được chia ra thành nhiều khối ($P = P_0P_1\dots P_{n-1}$) và áp dụng mã khối cho từng khối một. Có nhiều mô hình áp dụng mã khối là ECB, CBC, CTR, OFB và CFB.

Giải pháp hữu hiệu và phổ biến nhất chống lại các mối đe dọa đến an ninh mạng là mã hóa. Để thực hiện mã hóa, cần xác định: Mã hóa những gì và Thực hiện mã hóa ở đâu. Có 2 phương án cơ bản: Mã hóa liên kết và Mã hóa đầu cuối.

Có 4 cách phân phối khóa:

1. Khóa có thể được chọn bởi bên A và gửi theo đường vật lý đến bên B
2. Khóa có thể được chọn bởi một bên thứ ba, sau đó gửi theo đường vật lý đến A và B
3. Nếu A và B đã có một khóa dùng chung thì một bên có thể gửi khóa mới đến bên kia, sử dụng khóa cũ để mã hóa khóa mới
4. Nếu mỗi bên A và B đều có một kênh mã hóa đến một bên thứ ba C thì C có thể gửi khóa theo các kênh mã hóa đó đến A và B

CÂU HỎI ÔN TẬP

Câu 1: Mã hóa đổi xứng hiện đại và mã hóa đổi xứng cổ điển khác nhau ở điểm nào?

Câu 2: Mã dòng hoạt động dựa trên nguyên tắc thay thế hay hoán vị?

Câu 3: Từ nguyên tắc sinh số của mã hóa A5/1 và RC4, hãy cho biết lý do mã dòng lại dùng bộ sinh số để sinh ra dãy bít? Tại sao không dùng trực tiếp khóa K để thực hiện phép XOR ?

Câu 4: Hệ mã Fiestel có thuận lợi gì trong việc thực hiện mã khối?

Câu 5: Tại sao mã hóa DES lại dùng các phép biến đổi phức tạp chỉ để mã hóa một khối 64 bít?

Câu 6: Xét mô hình ECB, để mã hóa một bản tin dài bằng mã DES, chúng ta phải lần lượt mã hóa từng khối 64 bít. Việc thực hiện như vậy giống và khác với mã dòng ở những điểm nào?

Câu 7: Mô hình CBC có đặc tính gì mà các phương pháp mã hóa theo nguyên tắc thay thế (như ECB) không có?

Câu 8: Tại sao nói mô hình CTR, OFB và CFB thực ra là mã dòng?

Câu 9: Một bản rõ phải có đặc điểm gì thì mới có thể nói phương pháp mã hóa đổi xứng có tính chứng thực? Nếu Trudy không biết khóa bí mật của Alice và Bob, Trudy có thể mạo danh Alice gửi thông điệp mà Trudy muốn cho Bob được không?

Câu 10: Trong mã hóa đổi xứng, việc hai người cùng biết khóa dẫn đến nhược điểm gì của phương pháp mã hóa này?

Câu 11: Trình bày các phương pháp bố trí ứng dụng mã hóa?

Câu 12: Hãy nêu lợi ích của việc dùng khóa chủ và khóa phiên?

Câu 13: Trình bày quá trình trao đổi khóa mật dựa vào trung tâm phân phối khóa, nhận xét tính bí mật của nó?

BÀI 4: MÃ HÓA KHÓA CÔNG KHAI RSA

Sau khi học xong bài này, sinh viên hiểu:

- Các khái niệm toán học và biết cách áp dụng tính toán liên quan.
- Nguyên tắc hoạt động của hệ mã bắt đổi xứng và bài toán áp dụng của nó. Phân biệt được khóa riêng và khóa bí mật
- Cách tạo khóa và quá trình mã hóa của hệ mã RSA, tính an toàn của nó

4.1 LÝ THUYẾT SỐ

4.1.1 Một Số Khái Niệm

1. *Phép chia modulo:*

Phép chia modulo là phép chia lấy phần dư. Ví dụ: $27 \bmod 8 = 3$, $35 \bmod 9 = 8$.
Một cách tổng quát:

$$a \bmod n = r \text{ với } a \geq 0; n > 0; 0 \leq r \leq n-1$$

Nếu hai số a , b có cùng số dư trong phép chia cho n thì ta nói rằng a và b là đồng dư trong phép chia modulo cho n , phép so sánh đồng dư được ký hiệu bằng dấu \equiv :

$$a \equiv b \pmod{n} \text{ hay viết tắt là } a \equiv b \bmod n$$

Ta có thể thấy, phép toán modulo phân hoạch tập số tự nhiên N thành n lớp tương đương đồng dư - ứng với các giá trị của r trong tập $\{0, 1, 2, 3, \dots, n-1\}$. Ví dụ với giá trị $n = 4$ ta có 4 lớp tương đương sau:

$$\{0, 4, 8, 12, 16 \dots\}, \{1, 5, 9, 13, 17 \dots\}, \{2, 6, 10, 14, 18 \dots\}, \{3, 7, 11, 15, 19 \dots\}$$

2. *Một số tính chất của modulo:*

Cho a , b và n là các số nguyên, phép modulo có các tính chất:

a. $(a + b) \bmod n = [(a \bmod n) + (b \bmod n)] \bmod n$

b. $(a - b) \bmod n = [(a \bmod n) - (b \bmod n)] \bmod n$

c. $(a \times b) \bmod n = [(a \bmod n) \times (b \bmod n)] \bmod n$

3. Ước số

Nếu $a \bmod n = 0$ (viết cách khác $a \equiv 0 \pmod n$) thì có nghĩa là a chia hết cho n, hay n là ước số của a.

Ước số chung lớn nhất của hai số: ký hiệu $\text{gcd}(a, b)$. Để tìm USCLN của hai số a, b, chúng ta có thể dùng thuật toán Euclid (xem Phụ lục 2).

4. Số nguyên tố

Một số p được gọi là số nguyên tố nếu p chỉ chia hết cho 1 và chính nó, ngoài ra không chia hết cho số nào khác từ 2 đến $p - 1$.

5. Số nguyên tố cùng nhau

Hai số nguyên a, b được gọi là nguyên tố cùng nhau nếu USCLN của a và b là 1. Ký hiệu: $a \perp b$. Ví dụ: $3 \perp 8, 7 \perp 9, 4 \perp 15$. Hai số 20 và 15 không nguyên tố cùng nhau vì có USCLN là 5.

6. Phần tử nghịch đảo của phép nhân modulo:

Nếu hai số nguyên a và n nguyên tố cùng nhau, thì tồn tại số nguyên w sao cho:

$$a \cdot w \equiv 1 \pmod{n}$$

Ta gọi w là phần tử nghịch đảo của a trong phép modulo cho n và ký hiệu là a^{-1}

Ví dụ:

- $n = 10, a = 7$ là hai số nguyên tố cùng nhau, do đó tìm được $a^{-1} = 3$ ($21 \equiv 1 \pmod{10}$)

a^{-1}	0	1	2	3	4	5	6	7	8	9
$a^{-1} \times 7$	0	7	4	1	8	5	2	9	6	3

- $n = 10, a = 2$ không phải là hai số nguyên tố cùng nhau, ta có bảng phép nhân sau:

a^{-1}	0	1	2	3	4	5	6	7	8	9
$a^{-1} \times 2$	0	2	4	6	8	0	2	4	6	8

Trong bảng trên không tồn tại số a^{-1} nào sao cho $a \cdot a^{-1} \equiv 1 \pmod{10}$. Vậy không tồn tại phần tử nghịch đảo.

4.1.2 Định Lý Fermat

Định lý:

Nếu p là số nguyên tố và a là số nguyên không chia hết cho p thì $a^{p-1} \equiv 1 \pmod{p}$

Chứng minh:

Xét tập X gồm $p - 1$ phần tử sau:

$$X = \{ a \pmod{p}, 2a \pmod{p}, \dots, (n-1)a \pmod{p} \}$$

Ta có hai nhận xét sau:

- Không có phần tử nào của tập X bằng 0 vì a nguyên tố cùng nhau với p .
- Không tồn tại hai phần tử thứ i và thứ j ($i \neq j$) sao cho: $ia \pmod{p} = ja \pmod{p}$.

Vì a nguyên tố cùng nhau với p nên tồn tại a^{-1} trong phép modulo p . Do đó nếu $ia \equiv ja \pmod{p}$ thì $iaa^{-1} \equiv jaa^{-1} \pmod{p}$ nghĩa là $i \equiv j \pmod{p}$. Điều này trái với giả thiết $i \neq j$.

Từ hai nhận xét trên ta suy ra các phần tử của X sẽ là một hoán vị của các giá trị $\{1, 2, \dots, p-1\}$. Do đó:

$$\begin{aligned} A \times 2a \times \dots \times (p-1)a &\equiv [1 \times 2 \times \dots \times (p-1)] \pmod{n} \\ \Rightarrow a \times a \times \dots \times a &= a^{p-1} \equiv 1 \pmod{n} \quad (\text{đpcm}) \end{aligned}$$

Sau đây là một số ví dụ của định lý Fermat:

- $p = 5, a = 7 \Rightarrow 7^4 = 49 \cdot 49 = 2401, 2401 \equiv 1 \pmod{5}$
- $p = 7, a = 4 \Rightarrow 4^6 = 64 \cdot 64 = 4096, 4096 \equiv 1 \pmod{7}$

4.1.3 Phép Logarit Rời rạc

Ta định nghĩa phép lũy thừa modulo như sau, để tính y từ a, x và n là các số nguyên:

$$y = a^x \pmod{n} = (a \cdot a \cdot \dots \cdot a) \pmod{n} \quad \text{với } x \text{ số } a \text{ nhân với nhau}$$

Ta chỉ xét trường hợp n là số nguyên tố. Bảng sau minh họa các giá trị của phép lũy thừa modulo với $n = 19$, a và x từ 1 đến 18.

a	a^2	a^3	a^4	a^5	a^6	a^7	a^8	a^9	a^{10}	a^{11}	a^{12}	a^{13}	a^{14}	a^{15}	a^{16}	a^{17}	a^{18}
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	4	8	16	13	7	14	9	18	17	15	11	3	6	12	5	10	1
3	9	8	5	15	7	2	6	18	16	10	11	14	4	12	17	13	1
4	16	7	9	17	11	6	5	1	4	16	7	9	17	11	6	5	1
5	6	11	17	9	7	16	4	1	5	6	11	17	9	7	16	4	1
6	17	7	4	5	11	9	16	1	6	17	7	4	5	11	9	16	1
7	11	1	7	11	1	7	11	1	7	11	1	7	11	1	7	11	1
8	7	18	11	12	1	8	7	18	11	12	1	8	7	18	11	12	1
9	5	7	6	16	11	4	17	1	9	5	7	6	16	11	4	17	1
10	5	12	6	3	11	15	17	18	9	14	7	13	16	8	4	2	1
11	7	1	11	7	1	11	7	1	11	7	1	11	7	1	11	7	1
12	11	18	7	8	1	12	11	18	7	8	1	12	11	18	7	8	1
13	17	12	4	14	11	10	16	18	6	2	7	15	5	8	9	3	1
14	6	8	17	10	7	3	4	18	5	13	11	2	9	12	16	15	1
15	16	12	9	2	11	13	5	18	4	3	7	10	17	8	6	14	1
16	9	11	5	4	7	17	6	1	16	9	11	5	4	7	17	6	1
17	4	11	16	6	7	5	9	1	17	4	11	16	6	7	5	9	1
18	1	18	1	18	1	18	1	18	1	18	1	18	1	18	1	18	1

Bảng 4.1: Bảng giá trị lũy thừa modulo với $n=19$

Nhìn vào bảng trên, ta thấy rằng không phải hàng nào cũng có đầy đủ các giá trị từ 1 đến 18. Xét hàng $a = 11$, ta có:

- $11^1 \equiv 11 \pmod{19} \quad (*)$
- $11^2 = 121 \equiv 7 \pmod{19}$
- $11^3 = 1331 \equiv 1 \pmod{19}$
- $11^4 \equiv 11^3 \cdot 11 \equiv 11 \pmod{19}$ (giống như hàng $(*)$)
- $11^5 \equiv 11^2 \pmod{19} \dots$

Do đó hàng $a = 11$ (tương ứng với dãy $11^1, 11^2, \dots, 11^{18}$) chỉ có ba giá trị 11, 7, 1 được lặp lại theo chu kỳ.

Trong bảng trên chỉ có các giá trị $a = 2, 3, 10, 13, 14, 15$ là làm cho dãy a^1, a^2, \dots, a^{18} có đầy đủ các giá trị từ 1 đến 18 với phép modulo 19. Như vậy chỉ có $a = 2, 3, 10, 13, 14, 15$ thì phép lũy thừa modulo trên mới *khả nghịch*.

Trong trường hợp tổng quát với mỗi n chỉ có một số trường hợp của a thì phép lũy thừa là khả nghịch. Lúc này a được gọi là *primitive root* của n .

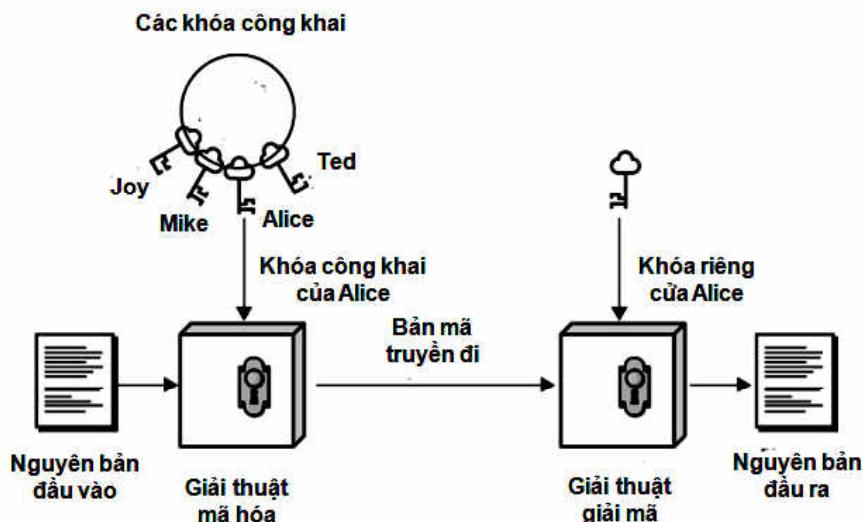
Việc tính logarithm rắc rối đã được chứng minh là rất tốn kém về mặt thời gian. Và được xem như là bất khả thi nếu a và n là các số lớn. Do đó phép lũy thừa modulo cũng được xem là hàm một chiều và được ứng dụng trong phương pháp trao đổi khóa Diffie – Hellman.

4.2 MÃ KHÓA CÔNG KHAI

Khóa công khai ra đời vào đầu những năm 1970. Có thể nói đây là bước tiến quan trọng nhất trong lịch sử 3000 năm mã hóa. Ở đây người ta sử dụng 2 khóa: một khóa riêng và một khóa công khai. Hai khóa này khác nhau, không đối xứng với nhau, do đó mã khóa công khai, còn được gọi là mã không đối xứng. Người ta đã ứng dụng một cách thông minh các kết quả của lý thuyết số về hàm số.

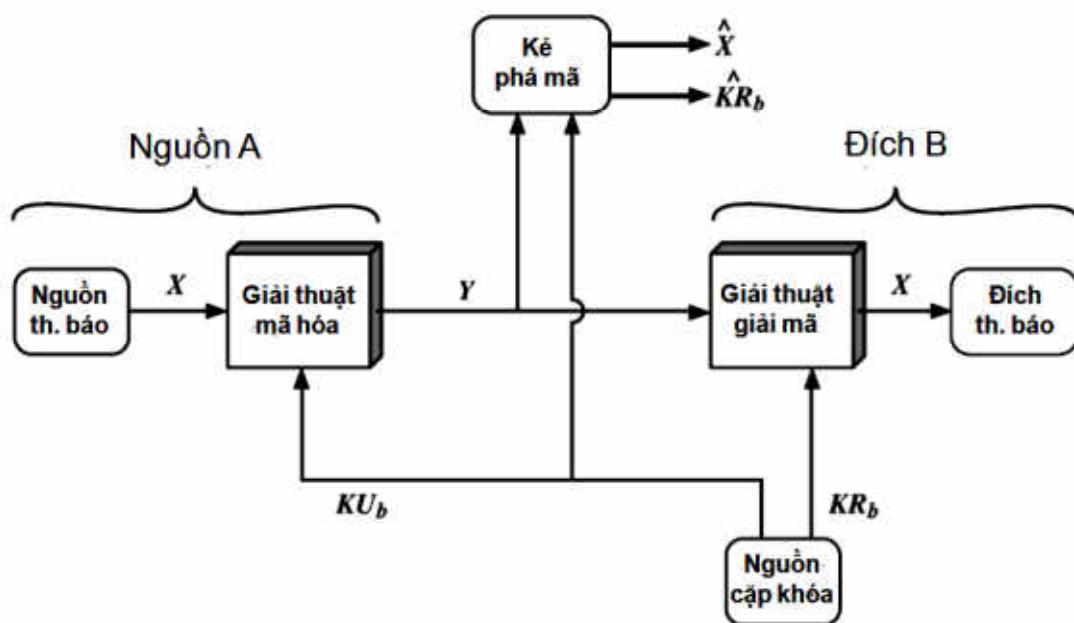
Khóa công khai ra đời hỗ trợ thêm để giải quyết một số bài toán an toàn, chứ không phải thay thế khóa riêng. Cả hai khóa cùng tồn tại, phát triển và bổ sung cho nhau. Khóa công khai/hai khóa/không đối xứng bao gồm việc sử dụng 2 khóa:

- Khóa công khai, mà mọi người đều biết, được dùng để mã hóa mẫu tin và kiểm chứng chữ ký.
- Khóa riêng, chỉ người nhận biết, để giải mã bản tin hoặc để tạo chữ ký.
- Là không đối xứng vì những người mã hóa và kiểm chứng chữ ký không thể giải mã hoặc tạo chữ ký.



Hình 4.1: Sơ đồ mã khóa công khai

4.2.1 Ứng dụng khóa công khai

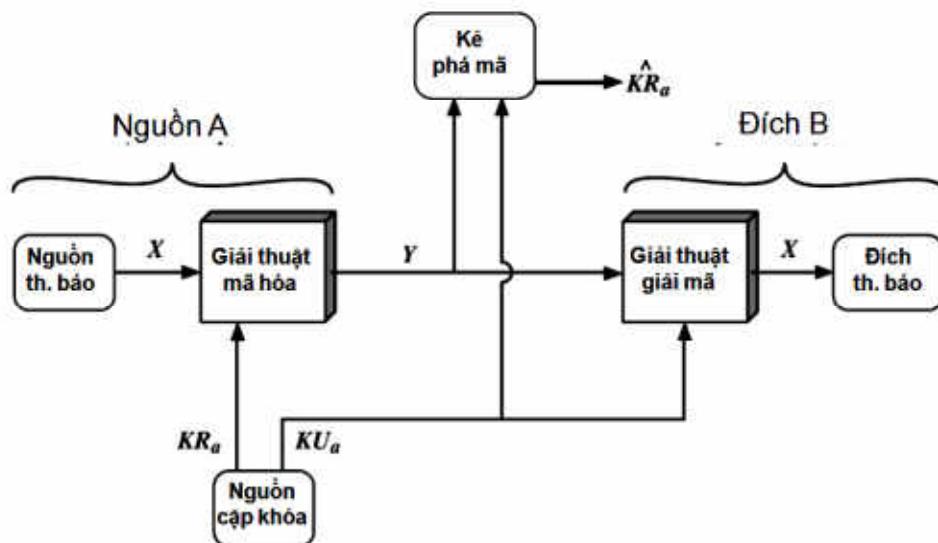


Hình 4.2: Mô hình ứng dụng bảo mật

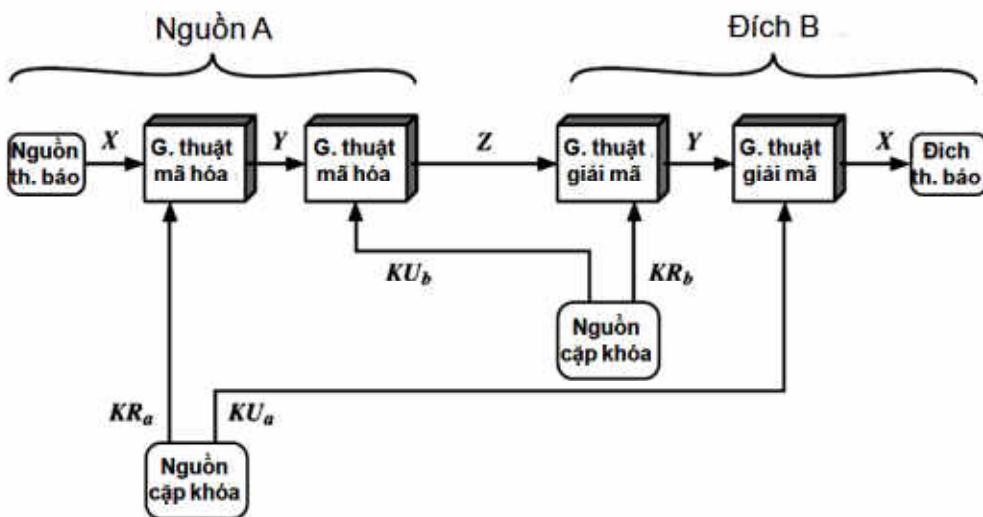
Có thể phân loại các ứng dụng của khóa công khai thành 3 loại khác nhau:

- *Mã/giải mã – cung cấp bảo mật*: Đây là ứng dụng bảo mật truyền thống giống như ta vẫn thường dùng với khóa đối xứng.
- *Chữ ký điện tử – cung cấp xác thực*: Một trong các ứng dụng mới của khóa công khai mà khóa đối xứng không thể thực hiện được, đó là khóa công khai có đủ cơ sở để xác nhận người gửi và có thể là một lựa chọn để tạo chữ ký điện tử của người gửi.
- *Trao đổi khóa*: Cho phép chia sẻ khóa phiên trong mã hóa đối xứng.

Một số giải thuật khóa công khai thích hợp cho cả 3 loại ứng dụng; một số khác chỉ có thể dùng cho 1 hay 2 loại.



Hình 4.3: Mô hình ứng dụng xác thực



Hình 4.4: Mô hình ứng dụng bảo mật và xác thực

4.2.2 Tính an toàn của các sơ đồ khóa công khai

Cũng giống như khóa riêng việc tìm kiếm vét cạn luôn luôn có thể, tức là khi biết một trong hai khóa và thuật toán mã hoá về nguyên tắc ta có thể dò tìm khóa thứ hai bằng cách tính toán các giá trị liên quan. Nói chung khối lượng cần tính toán là rất lớn do độ phức tạp của bài toán xác định khóa. Nếu khóa sử dụng là rất lớn cỡ hơn 512 bit, thì hầu như bài toán tìm khóa thứ hai là không khả thi, không thể thực hiện được trong thời gian có nghĩa, cho dù nguồn lực có thể rất lớn.

Tính an toàn dựa trên sự khác biệt đủ lớn giữa các bài toán dễ là mã/giải mã khi biết khóa và bài toán khó là thám mã khi không biết khóa tương ứng. Vì bài toán

thám mã nằm trong lớp các bài toán khó tổng quát hơn đã được biết đến và về mặt lý thuyết đã được chứng minh là nó rất khó có thể thực hiện trên thực tế. Bởi vì nó đòi hỏi sử dụng số rất lớn, nên số phép toán cần thực hiện là rất nhiều. Đây là ý tưởng chính để tạo nên một mã công khai. Ta tìm kiếm các bài toán mà nếu biết thông tin mật nào đó được che dấu thì nó rất dễ thực hiện, còn nếu không thì nó thuộc lớp bài toán rất khó giải, hầu như không thể giải trên thực tế.

Mã công khai thường chậm hơn khá nhiều so với mã đối xứng, nên nó thường được dùng mã những thông tin nhỏ quan trọng.

4.3 RSA

RSA là một phương pháp mã hóa khóa công khai. RSA được xây dựng bởi các tác giả Ron Rivest, Adi Shamir và Len Adleman tại học viện MIT vào năm 1977, và ngày nay đang được sử dụng rộng rãi. Về mặt tổng quát RSA là một phương pháp mã hóa theo khối. Trong đó bản rõ M và bản mã C là các số nguyên từ 0 đến 2^i với i số bít của khối. Kích thước thường dùng của i là 1024 bít. RSA sử dụng hàm một chiều là vẫn đề phân tích một số thành thừa số nguyên tố.

4.3.1 Khởi tạo khóa RSA

Mỗi người sử dụng tạo một cặp khóa công khai – riêng như sau:

- Chọn ngẫu nhiên 2 số nguyên tố lớn p và q
- Tính số làm modulo của hệ thống: $N = p \cdot q$
- Ta đã biết $\Phi(N) = (p-1)(q-1)$, và có thể dùng Định lý Trung Hoa để giảm bớt tính toán
- Chọn ngẫu nhiên khóa mã e , trong đó $1 < e < \Phi(N)$, $\text{gcd}(e, \Phi(N)) = 1$
- Giải phương trình sau để tìm khóa giải mã d sao cho: $e \cdot d \equiv 1 \pmod{\Phi(N)}$ với $0 \leq d \leq \Phi(N)$
- Khóa mã công khai $KU = \{e, N\}$, giữ khóa riêng bí mật $KR = \{d, p, q\}$

4.3.2 Sử dụng RSA

Để mã hoá mẫu tin, người gửi:

- Lấy khóa công khai của người nhận KU={e, N}
- Tính $C = M^e \text{ mod } N$, trong đó $0 \leq M < N$

Để giải mã bản mã, người sở hữu nhận:

- Sử dụng khóa riêng KR={d, p, q}
- Tính $M = C^d \text{ mod } N$

Lưu ý rằng bản tin $M < N$, do đó khi cần chia khối bản rõ.

4.3.3 Cơ sở của RSA

Theo Định lý Ole: $a^{\Phi(n)} \text{ mod } N = 1$ trong đó $\gcd(a, N) = 1$.

Ta có $N = p \cdot q$, với $\Phi(N) = (p-1)(q-1)$, $e \cdot d = 1 \text{ mod } \Phi(N) \rightarrow e \cdot d = 1 + k \cdot \Phi(N)$ đối với một giá trị k nào đó.

Suy ra: $C^d = (M^e)^d = M^{1+k \cdot \Phi(N)} = M^1 \cdot (M^{\Phi(N)})^k$

Nên $C^d \text{ mod } N = M^1 \cdot (1)^k \text{ mod } N = M^1 \text{ mod } N = M \text{ mod } N$

4.3.4 Ví dụ

Tạo cặp khóa:

- Chọn các số nguyên tố: $p=17$ & $q=11$.
- Tính $n = pq$, $n = 17 \times 11 = 187$
- Tính $\Phi(n) = (p-1)(q-1) = 16 \times 10 = 160$
- Chọn e : $\gcd(e, 160) = 1$; Lấy $e=7$
- Xác định d : $de = 1 \text{ mod } 160$ và $d < 160$, Giá trị cần tìm là $d=23$, vì $23 \times 7 = 161 = 10 \times 160 + 1$
- In khóa công khai KU={7, 187} và giữ khóa riêng bí mật KR={23, 17, 11}
- Áp dụng mã RSA trên như sau:

- Cho mẫu tin $M = 88$ (vậy $88 < 187$)
- Mã $C = 887 \text{ mod } 187 = 11$
- Giải mã $M = 1123 \text{ mod } 187 = 88$

4.3.5 Sinh khóa RSA

Người sử dụng RSA cần phải xác định ngẫu nhiên 2 số nguyên tố rất lớn, thông thường khoảng 512 bit. Do đó việc sinh ra ngẫu nhiên p, q và kiểm tra xác suất tính nguyên tố của chúng có nhiều giải pháp khác nhau với độ tin cậy cao. Sau khi chọn được một khóa e hoặc d nguyên tố cùng nhau với $\Phi(n)$, dễ dàng tính được khóa kia chính là số nghịch đảo của nó qua thuật toán Euclide mở rộng.

4.3.6 Độ an toàn của RSA

Số chữ số của N	Số bit	Năm phá mã	Thuật toán
100	322	1991	Quadratic sieve
110	365	1992	Quadratic sieve
120	398	1993	Quadratic sieve
129	428	1994	Quadratic sieve
130	431	1996	GNFS
140	465	1999	GNFS
155	512	1999	GNFS
160	530	2003	Lattice sieve
174	576	2003	Lattice sieve
200	633	2005	Lattice sieve

Hình 4.5: Hình liệt kê các mốc phá mã RSA

Sau đây ta sẽ xem xét một số các tấn công phương pháp RSA.

1. *Vết cạn khóa:* cách tấn công này thử tất cả các khóa d có thể có để tìm ra bản giải mã có ý nghĩa, tương tự như cách thử khóa K của mã hóa đối xứng. Với N lớn, việc tấn công là bất khả thi.

2. *Phân tích N thành thừa số nguyên tố N = pq*: Chúng ta đã nói rằng việc phân tích phải là bất khả thi thì mới là hàm một chiều, là nguyên tắc hoạt động của RSA. Tuy nhiên, nhiều thuật toán phân tích mới đã được đề xuất, cùng với tốc độ xử lý của máy tính ngày càng nhanh, đã làm cho việc phân tích N không còn quá khó khăn như trước đây. Năm 1977, các tác giả của RSA đã treo giải thưởng cho ai phá được RSA có kích thước của N vào khoảng 428 bít, tức 129 chữ số. Các tác giả này ước đoán phải mất 40 nghìn triệu triệu năm mới có thể giải được. Nhưng vào năm 1994, câu đố này đã được giải trong vòng 8 tháng. Hình 4-2 liệt kê kích thước N của các RSA đã phá mã được cho đến hiện nay.

Dĩ nhiên là việc phá mã trên chỉ được thực hiện trong phòng thí nghiệm. Tuy nhiên người ta cho rằng kích thước của N phải khoảng 1024 bít (309 chữ số) thì mới bảo đảm an toàn thật sự.

3. *Đo thời gian*: Đây là một phương pháp phá mã không dựa vào mặt toán học của thuật toán RSA, mà dựa vào một "hiệu ứng lề" sinh ra bởi quá trình giải mã RSA. Hiệu ứng lề đó là thời gian thực hiện giải mã. Giả sử người phá mã có thể đo được thời giải mã $M = C^d \text{ mod } N$ dùng thuật toán bình phương liên tiếp. Trong thuật toán bình phương liên tiếp, nếu một bít của d là 1 thì xảy ra hai phép modulo, nếu bít đó là 0 thì chỉ có một phép modulo, do đó thời gian thực hiện giải mã là khác nhau. Bằng một số phép thử chosen-plaintext, người phá mã có thể biết được các bít của d là 0 hay 1 và từ đó biết được d.

Phương pháp phá mã này là một ví dụ cho thấy việc thiết kế một hệ mã an toàn rất phức tạp. Người thiết kế phải lường trước được hết các tình huống có thể xảy ra.

TÓM TẮT

Bài học cung cấp cho sinh viên kiến thức về lý thuyết toán học cơ bản hỗ trợ cho lý thuyết mã hóa, hệ mã khóa bất đối xứng và hệ mã RSA.

Hai khái niệm toán học quan trọng hỗ trợ mã hóa là phép toán đồng dư và định lý Fermat. Phép toán đồng dư: Nếu hai số a, b có cùng số dư trong phép chia cho n thì ta nói rằng a và b là đồng dư trong phép chia modulo cho n . Định lý Fermat: Nếu p là số nguyên tố và a là số nguyên không chia hết cho p thì $a^{p-1} \equiv 1 \pmod{p}$.

Khóa công khai ra đời hỗ trợ thêm để giải quyết một số bài toán an toàn, chứ không phải thay thế khóa riêng. Cả hai khóa cùng tồn tại, phát triển và bổ sung cho nhau. Hệ mã khóa công khai hay bất đối xứng bao gồm việc sử dụng 2 khóa:

- Khóa công khai, mà mọi người đều biết, được dùng để mã hóa mẫu tin và kiểm chứng chữ ký.
- Khóa riêng, chỉ người nhận biết, để giải mã bản tin hoặc để tạo chữ ký.

Hệ mã khóa công khai có những ứng dụng là: Mã/giải mã – cung cấp bảo mật. Đây là ứng dụng bảo mật truyền thống giống như ta vẫn thường dùng với khóa đối xứng. Chữ ký điện tử - cung cấp xác thực. Một trong các ứng dụng mới của khóa công khai mà khóa đối xứng không thể thực hiện được, đó là khóa công khai có đủ cơ sở để xác nhận người gửi và có thể là một lựa chọn để tạo chữ ký điện tử của người gửi. Trao đổi khóa: Cho phép chia sẻ khóa phiên trong mã hóa đối xứng.

RSA là mã công khai được biết đến nhiều nhất và sử dụng rộng rãi nhất hiện nay. Nó dựa trên các phép toán lũy thừa trong trường hữu hạn các số nguyên theo modulo nguyên tố. Cụ thể, mã hóa hay giải mã là các phép toán lũy thừa theo modulo số rất lớn. Việc thám mã, tức là tìm khóa riêng khi biết khóa công khai, dựa trên bài toán khó là phân tích một số rất lớn đó ra thừa số nguyên tố.

CÂU HỎI ÔN TẬP

Câu 1: Nêu điểm yếu của mã hóa đối xứng?

Câu 2: Hàm một chiều là gì? Cho ví dụ về hàm một chiều ?

Câu 3: Trong số học modulo n, khi nào thì một số có số nghịch đảo của phép nhân?

Câu 4: Logarit rời rạc khác logarit liên tục ở những điểm nào?

Câu 5: Để kiểm tra tính nguyên tố của một số nguyên, thuật toán Miller-Rabin có thể cho kết quả sai, vậy tại sao người ta vẫn sử dụng thuật toán này?

Câu 6: Tại sao trong thuật toán RSA cần dùng phương pháp bình phương liên tiếp để tính lũy thừa modulo?

Câu 7: Nêu nguyên tắc của mã hóa khóa công khai? Tại sao trong mã hóa khóa công khai không cần dùng đến kênh an toàn để truyền khóa?

Câu 8: Trong mã hóa khóa công khai, khóa riêng và khóa công khai có phải là 2 khóa tùy ý, không liên quan? Nếu có liên quan, tại sao không thể tính khóa riêng từ khóa công khai?

Câu 9: Ngoài vấn đề truyền khóa, mã hóa khóa công khai còn ưu điểm hơn mã hóa đối xứng ở điểm nào?

Câu 10: Nêu nhược điểm của mã hóa khóa công khai ?

Câu 11: Trình bày quá trình tạo khóa và mã hóa của RSA ?

BÀI 5: QUẢN LÝ KHÓA DÙNG MÃ KHÓA CÔNG KHAI

Sau khi học xong bài này, sinh viên hiểu:

- *Bài toán phân phối khóa công khai và các giải pháp của nó*
- *Giải pháp trao đổi công khai khóa mật dựa vào hệ mã khóa công khai*
- *Tính an toàn của hệ mã khóa công khai DiffieHellman trong việc trao đổi công khai khóa bí mật.*

5.1 PHÂN PHỐI KHÓA

Nhu cầu cấp bách là cần phải tạo ra một cơ chế chia sẻ khóa trong môi trường thường xuyên trao đổi thông tin và thường xuyên thay đổi khóa. Mã khóa công khai giúp giải bài toán phân phối khóa, Nó bao gồm hai khía cạnh sau:

- Phân phối khóa một cách công khai nhưng đảm bảo được bí mật.
- Sử dụng mã khóa công khai để phân phối khóa mật (còn khóa mật dùng để mã hóa thông tin).

5.1.1 Phân phối khóa công khai

Có thể xem xét để được sử dụng vào một trong những việc sau:

- Thông báo công khai khóa của người sử dụng.
- Thư mục truy cập công cộng cho mọi người.
- Chủ quyền khóa công khai, người nắm giữ khóa công khai.
- Chứng nhận khóa công khai, khóa công khai của người sử dụng được nơi có thẩm quyền chứng nhận.

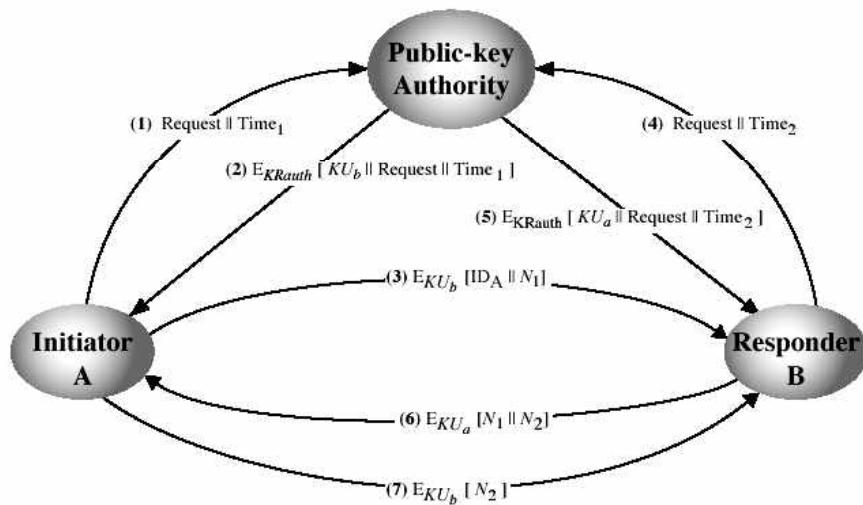
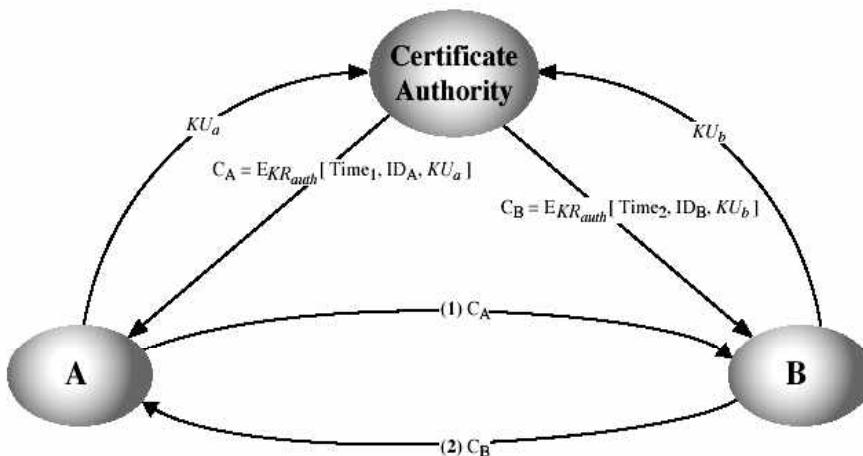
Thông báo công khai: Người dùng phân phối khóa công khai cho người nhận hoặc thông báo rộng rãi cho cộng đồng. Chẳng hạn như người sử dụng có thể tự bổ sung khóa PGP vào thư điện tử hoặc gửi cho nhóm chia sẻ tin hoặc một danh sách thư điện tử. Điểm yếu chính của thông báo công khai là mạo danh: một người nào đó có thể tạo khóa và tuyên bố mình là một người khác và gửi thông báo cho mọi người khác. Cho đến khi giả mạo bị phát hiện thì kẻ mạo danh đã có thể lừa trong vai trò của người khác.

Thư mục truy cập công cộng: Dùng thư mục truy cập công cộng có thể đạt được tính an toàn cao hơn bằng cách đăng ký khóa với thư mục công cộng để đăng tải và chia sẻ cho mọi người. Thư mục cần được đảm bảo tin cậy với các tính chất sau:

- Chứa việc nhập tên và khóa công khai
- Người dùng đăng ký mật với Thư mục
- Người dùng có thể thay khóa bất cứ lúc nào
- Thư mục được in định kỳ
- Thư mục có thể truy cập qua mạng

Mô hình trên vẫn còn có các lỗ hổng để kẻ xâm nhập sửa hoặc giả mạo khi truy cập vào hệ thống.

Chủ quyền khóa công khai: Đây là bước cải thiện tính an toàn bằng kiểm soát chặt chẽ tập trung việc phân phối khóa từ Thư mục. Nó bao gồm các tính chất của một Thư mục như đã nêu ở phần trước và đòi hỏi người dùng biết khóa công khai của Thư mục đó. Sau đó người dùng nhận được bất kỳ khóa công khai mong muốn nào một cách an toàn, bằng cách truy cập thời gian thực đến Thư mục khi cần đến khóa. Tuy nhiên yêu cầu truy cập thời gian thực là một nhược điểm của cách phân phối khóa này. Cụ thể trong kịch bản sau hai người sử dụng chia sẻ khóa công khai của mình cho nhau thông qua việc sử dụng khóa công khai của Chủ quyền để nhận được khóa công khai của đối tác và trao đổi qua lại để khẳng định người này đã biết thông tin của người kia.

**Hình 5.1: Phương pháp chủ quyền khóa công khai****Hình 5.2: Phương pháp chứng thực khóa công khai**

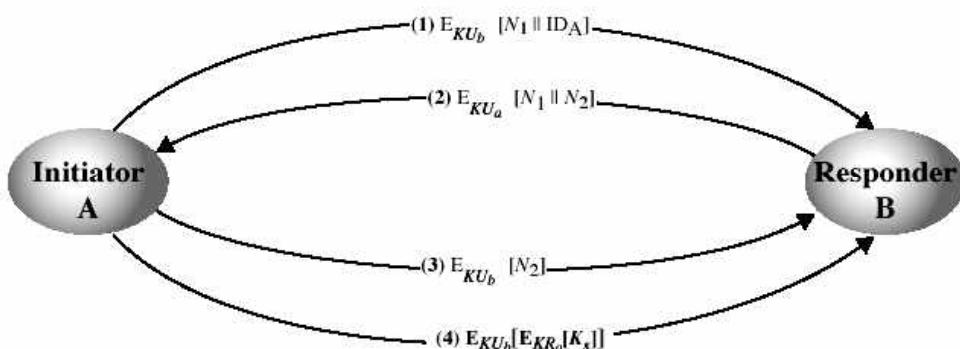
Chứng thực khóa công khai: Chứng nhận cho phép trao đổi khóa không cần truy cập thời gian thực đến Chủ quyền thư mục khóa công khai. Để làm việc đó chứng nhận trói danh tính của người sử dụng với khóa công khai của anh ta và “đóng dấu và giấy chứng nhận” đó để tránh giả mạo. Các thông tin đi kèm thông thường là chu kỳ kiểm định, quyền sử dụng, thời hạn, ... Nội dung trên được ký bởi khóa riêng tin cậy của Chủ quyền chứng nhận (CA, Certificate Authority). Do khóa công khai của CA được thông báo rộng rãi, nên chứng nhận đó có thể được kiểm chứng bởi một người nào đó biết khóa công khai của Chủ quyền chứng nhận.

5.2 PHÂN PHỐI CÔNG KHAI KHÓA MẬT

Nói chung có thể sử dụng các phương pháp trên để nhận được khóa công khai của người định trao đổi thông tin. Khóa công khai đó dùng cho mục đích mã hoá, giải mã hoặc xác nhận thông tin là của đối tác. Nhưng các thuật toán khóa công khai chậm, nên giá để bảo mật thông tin là đắt. Do đó thông thường dùng khóa đối xứng để mã hoá và giải mã nội dung bản tin, còn hệ mã khóa công khai được áp dụng để trao đổi khóa mật của hệ mã khóa đối xứng. Phân phối khóa mật đơn giản được đề xuất bởi Merkle vào năm 1979:

- A tạo ra một cặp khóa công khai mới tạm thời
- A gửi B một khóa công khai và danh tính của họ
- B tạo ra khóa phiên và gửi nó cho A sử dụng khóa công khai được cung cấp
- A giải mã khóa phiên và cả hai cùng dùng nó.

Vấn đề nằm ở chỗ, kẻ thù có thể ngăn hoặc đóng giả cả hai bên của thủ tục. Nếu có khóa công khai thì khóa phiên được trao đổi an toàn.



Hình 5.3: sơ đồ trao đổi khóa Merkle

5.2.1 Trao đổi khóa hỗn hợp

Ta có thể kết hợp sử dụng Trung tâm phân phối khóa để phân phối khóa phiên như trên mô hình máy chủ của IBM. Trung tâm chia sẻ khóa chính (master key) với mỗi người sử dụng. Và phân phối khóa phiên sử dụng khóa chính với Trung tâm. Sơ đồ khóa công khai được dùng để phân phối khóa chính. Sơ đồ ba lớp này đặc biệt hữu ích

khi người sử dụng phân tán rộng. Các yêu cầu căn bản của hệ thống là chất lượng thực hiện và sự tương thích nền tảng.

5.3 TRAO ĐỔI KHÓA DIFFIE HELLMAN

Trao đổi khóa Diffie Hellman là sơ đồ khóa công khai đầu tiên được đề xuất bởi Diffie và Hellman năm 1976 cùng với khái niệm khóa công khai. Sau này được biết đến bởi James Ellis (Anh), người đã đề xuất bí mật năm 1970 mô hình tương tự. Đây là phương pháp thực tế trao đổi công khai các khóa mật. Nó thúc đẩy việc nghiên cứu đề xuất các mã khóa công khai. Sơ đồ được sử dụng trong nhiều sản phẩm thương mại. Đây là sơ đồ trao đổi khóa mật dùng khóa công khai:

- Không thể dùng để trao đổi mẫu tin bất kỳ.
- Tuy nhiên nó có thể thiết lập khóa chung.
- Chỉ có hai đối tác biết đến.
- Giá trị khóa phụ thuộc vào các đối tác (và các thông tin về khóa công khai và khóa riêng của họ).
- Dựa trên phép toán lũy thừa trong trường hữu hạn (modulo theo số nguyên tố hoặc đa thức) là bài toán dễ.
- Độ an toàn dựa trên độ khó của bài toán tính logarit rời rạc (giống bài toán phân tích ra thừa số) là bài toán khó.

5.3.1 Khởi tạo Diffie Hellman

Mọi người dùng thỏa thuận dùng tham số chung:

- Số nguyên tố rất lớn q hoặc đa thức.
- a là căn nguyên tố của mod q .

Mỗi người dùng (A chẳng hạn) tạo khóa của mình:

- Chọn một khóa mật (số) của A: $x_A < q$
- Tính khóa công khai của A: $y_A = a^{x_A} \text{ mod } q$.
- Mỗi người dùng thông báo công khai khóa của mình y_A .

5.3.2 Trao đổi khóa Diffie Hellman

Khóa phiên dùng chung cho hai người sử dụng A, B là K_{AB}

$$K_{AB} = a^{x_A \cdot x_B} \bmod q$$

$$= y_A^{x_B} \bmod q \text{ (mà B có thể tính)}$$

$$= y_B^{x_A} \bmod q \text{ (mà A có thể tính)}$$

K_{AB} được sử dụng như khóa phiên trong sơ đồ khóa riêng giữa A và B. A và B lần lượt trao đổi với nhau, họ có khóa chung K_{AB} cho đến khi họ chọn khóa mới. Kẻ thám mã cần x, do đó phải giải tính logarit rời rạc.

5.3.3 Ví dụ

Hai người sử dụng Alice & Bob muốn trao đổi khóa phiên:

- Đồng ý chọn số nguyên tố $q=353$ và $a=3$
- Chọn các khóa mật ngẫu nhiên:

A chọn $x_A=97$, B chọn $x_B=233$

- Tính các khóa công khai:

$$y_A = 3^{97} \bmod 353 = 40 \quad (\text{Alice})$$

$$y_B = 3^{233} \bmod 353 = 248 \quad (\text{Bob})$$

- Tính khóa phiên chung:

$$K_{AB} = y_B^{x_A} \bmod 353 = 248^{97} \bmod 353 = 160 \quad (\text{Alice})$$

$$K_{AB} = y_A^{x_B} \bmod 353 = 40^{233} \bmod 353 = 160 \quad (\text{Bob})$$

5.4 CHỐNG TÂN CÔNG PHÁT LẠI THÔNG ĐIỆP (REPLAY ATTACK) KHI ỨNG DỤNG

Trong hình thức tấn công phát lại thông điệp, Trudy chặn thông điệp của Alice gửi cho Bob, và sau đó một thời gian gửi lại thông điệp này cho Bob. Như vậy Bob sẽ nghĩ rằng Alice gửi thông điệp hai lần khác nhau. Tuy nhiên thực sự thì Alice chỉ gửi một

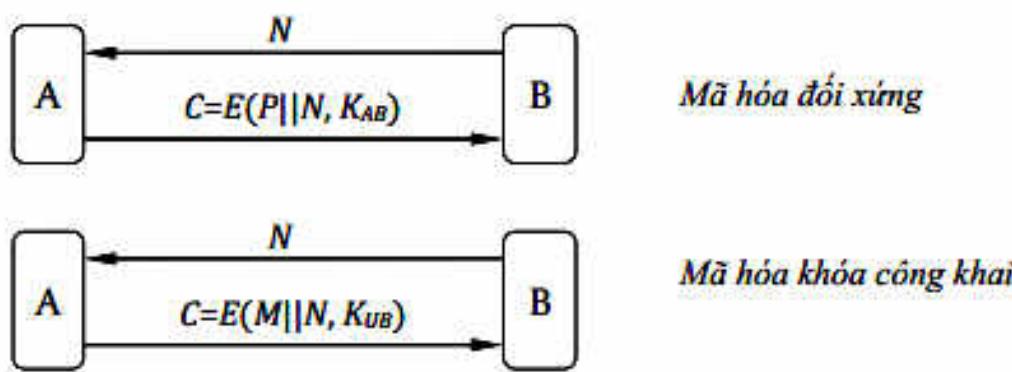
lần. Chỉ sử dụng mã hóa đối xứng và mã hóa khóa công khai thì không thể ngăn cản hình thức tấn công này. Để chống lại replay attack có 3 phương pháp:

1. *Dùng số định danh*: trong mỗi thông điệp gửi cho Bob, Alice nhúng vào đó một con số định danh thông điệp S. Mỗi thông điệp ứng với một S khác nhau.

$$C = E(P||S, K_{AB}) \quad || \text{ là phép nối dãy bít}$$

Do đó nếu Trudy phát lại thông điệp, Bob biết được hai thông điệp có cùng số định danh và loại bỏ thông điệp thứ hai. Tuy nhiên, phương pháp này có hạn chế là Bob phải lưu trữ số định danh của Alice để có cơ sở so sánh. Do đó phương pháp này thường chỉ sử dụng cho một phiên làm việc (connection oriented).

2. *Dùng timestamp*: trong mỗi thông điệp gửi cho Bob, Alice nhúng vào một timestamp T xác định thời điểm gửi. Bob chỉ chấp nhận thông điệp nếu nó đến được Bob trong một giới hạn thời gian nào đó kể từ lúc gửi. Tuy nhiên phương pháp này yêu cầu đồng hồ của Alice và của Bob phải đồng bộ, không được sai lệch đáng kể. Ngoài ra độ trễ của việc truyền tin trên mạng cũng là một trở ngại đối với phương pháp này.
3. *Dùng cơ chế challenge/response*: để bảo đảm thông điệp từ Alice không phải là replay, Bob gửi 1 số ngẫu nhiên N cho Alice (gọi là nonce). Alice sẽ nhúng N trong thông điệp gửi cho Bob.



Khi Bob giải mã thì sẽ kiểm tra N mà Bob nhận được xem có trùng khớp với N Bob gửi đi không. Như vậy Trudy không thể replay thông điệp $E(P||N, K_{AB})$ được vì mỗi lần Bob sẽ gửi một số N khác nhau. Tuy nhiên phương pháp này đòi hỏi thêm một

bước là Bob phải gửi N trước cho Alice. Vì vậy trong thực tế tùy trường hợp mà người ta sẽ sử dụng một trong 3 kỹ thuật trên cho hợp lý.

TÓM TẮT

Bài học cung cấp kiến thức về: Mã khóa công khai giúp giải bài toán phân phối khóa, Nó bao gồm hai khía cạnh sau: Phân phối khóa một cách công khai nhưng đảm bảo được bí mật. Sử dụng mã khóa công khai để phân phối khóa mật (còn khóa mật dùng để mã hóa thông tin).

Việc phân phối khóa công khai có 4 giải pháp: (1) Thông báo công khai khóa của người sử dụng, (2) Thư mục truy cập công cộng cho mọi người, (3) Chủ quyền khóa công khai, người nắm giữ khóa công khai và (4) Chứng nhận khóa công khai, khóa công khai của người sử dụng được nơi có thẩm quyền chứng nhận.

Phân phối khóa mật đơn giản sử dụng khóa công khai được đề xuất bởi Merkle vào năm 1979: (1) A tạo ra một cặp khóa công khai mới tạm thời, (2) A gửi B một khóa công khai và danh tính của họ, (3) B tạo ra khóa phiên và gửi nó cho A sử dụng khóa công khai được cung cấp và (4) A giải mã khóa phiên và cả hai cùng dùng nó.

Trao đổi khóa Diffie Hellman là sơ đồ khóa công khai đầu tiên được đề xuất bởi Diffie và Hellman năm 1976 cùng với khái niệm khóa công khai. Phương pháp này dùng để thiết lập khóa chung chỉ có hai đối tác biết đến. Giá trị khóa phụ thuộc vào các đối tác (và các thông tin về khóa công khai và khóa riêng của họ). Dựa trên phép toán lũy thừa trong trường hữu hạn (modulo theo số nguyên tố hoặc đa thức) là bài toán dễ. Độ an toàn dựa trên độ khó của bài toán tính logarit rời rạc (giống bài toán phân tích ra thừa số) là bài toán khó.

CÂU HỎI ÔN TẬP

Câu 1: Trình bày các giải pháp trao đổi khóa công khai? Cho biết hoàn cảnh áp dụng từng giải pháp?

Câu 2: Giải thích tính an toàn của giải pháp trao đổi khóa bí mật sử dụng hệ mã khóa công khai ?

Câu 3: Trao đổi khóa Difie Hellman: Chọn số nguyên tố dùng chung $q = 131$ và $\alpha = 7$,

Câu 4: NSD A chọn khóa riêng $x_A = 11$, NSD B chọn khóa riêng $x_B = 19$. Tính khóa công khai của A và B. Nêu cách A và B tính khóa mật dùng chung giữa A và B ?

Câu 5: Tấn công phát lại thông điệp là gì? Nêu tác hại của thao tác tấn công này và so sánh với việc sửa đổi thông điệp vào mạo danh?

Câu 6: Nêu các phương pháp chống lại tấn công phát lại thông điệp?

BÀI 6: MÃ CHỨNG THỰC THÔNG ĐIỆP, HÀM BĂM

Sau khi học xong bài này, sinh viên hiểu:

- Khái niệm xác thực mẫu tin, vai trò của xác thực mẫu tin trong việc cài đặt các dịch vụ toàn vẹn dữ liệu, chống chối bỏ.
- Khái niệm mã xác thực, các tính chất, nguyên tắc xây dựng mã xác thực và các mô hình triển khai ứng dụng của nó.
- Hàm băm, MD5, SHA1, và các mô hình ứng dụng của nó.

6.1 XÁC THỰC MẪU TIN

6.1.1 Các yêu cầu bảo mật khi truyền mẫu tin trên mạng

Tìm các biện pháp cần thiết để chống đổi lại các hành động phá hoại như sau:

- Để lộ bí mật: giữ bí mật nội dung mẫu tin, chỉ cho người có quyền biết.
- Thám mã đường truyền: không cho theo dõi hoặc làm trì hoãn việc truyền tin.
- Giả mạo: lấy danh nghĩa người khác để gửi tin.
- Sửa đổi nội dung: thay đổi, cắt xén, thêm bớt thông tin.
- Thay đổi trình tự các gói tin nhỏ của mẫu tin truyền.
- Sửa đổi thời gian: làm trì hoãn mẫu tin.
- Từ chối gốc: không cho phép người gửi từ chối trách nhiệm của tác giả mẫu tin.
- Từ chối đích: không cho phép người nhận phủ định sự tồn tại và đến đích của mẫu tin đã gửi.

6.1.2 Khái niệm xác thực mẫu tin

Xác thực mẫu tin liên quan đến các khía cạnh sau khi truyền tin trên mạng:

- Bảo vệ tính toàn vẹn của mẫu tin: bảo vệ mẫu tin không bị thay đổi hoặc có các biện pháp phát hiện nếu mẫu tin bị thay đổi trên đường truyền.
- Kiểm chứng danh tính và nguồn gốc: xem xét mẫu tin có đúng do người xưng tên gửi không hay một kẻ mạo danh nào khác gửi.
- Không chối từ bản gốc: trong trường hợp cần thiết, bản thân mẫu tin chứa các thông tin chứng tỏ chỉ có người xưng danh gửi, không một ai khác có thể làm điều đó. Như vậy người gửi không thể từ chối hành động gửi, thời gian gửi và nội dung của mẫu tin.

Ngoài ra có thể xem bổ sung thêm các yêu cầu bảo mật như mã hoá. Với mong muốn đáp ứng các yêu cầu trên, có 3 hàm lựa chọn sau đây được sử dụng:

- Mã mẫu tin bằng mã đối xứng hoặc mã công khai.
- Mã xác thực mẫu tin (MAC): dùng khóa và một hàm nén mẫu tin cần gửi để nhận được một đặc trưng đính kèm với mẫu tin và người gửi đó.
- Hàm hash (hàm băm) là hàm nén mẫu tin tạo thành “dấu vân tay” cho mẫu tin.

6.1.3 Mã mẫu tin

Mã mẫu tin bản thân đã cung cấp một phần tính xác thực, vì khóa được chia sẻ giữa người gửi và người nhận cũng như việc thay đổi nội dung cũng không dễ dàng thực hiện nếu không có khóa. Cụ thể nếu mã đối xứng được sử dụng thì người nhận biết người gửi phải tạo ra mẫu tin, vì chỉ có người gửi và người nhận biết được khóa sử dụng. Người nhận có thể biết nội dung không bị sửa đổi, nếu mẫu tin có cấu trúc phù hợp, tính dư thừa và tổng kiểm tra để phát hiện bất cứ thay đổi nào.

Nếu khóa công khai được sử dụng thì mã cung cấp không đủ độ tin cậy về người gửi, vì mọi người đều có thể biết khóa công khai của người nhận. Tuy nhiên nếu người gửi ký mẫu tin sử dụng khóa riêng của họ và sau đó mã với khóa công khai của người nhận, thì khi đó đảm bảo cả tính bảo mật và xác thực của mẫu tin. Cần phải bổ sung các biện pháp để phát hiện các mẫu tin đã bị làm hỏng. Việc sử dụng khóa riêng của

người gửi kết hợp với khóa công khai của người nhận có nhiều ưu việt, nhưng với giá phải trả là chậm do dùng 2 mã khóa công khai trên mẫu tin. Giải pháp mã xác thực mẫu tin sử dụng với khóa riêng của hệ mã công khai là an toàn.

6.2 MÃ XÁC THỰC MẪU TIN

Mã xác thực mẫu (MAC – Message Authentication Code) tin sinh ra bởi một thuật toán mà tạo ra một khối thông tin nhỏ có kích thước cố định. MAC phụ thuộc vào cả mẫu tin và khóa nào đó. MAC giống như mã nhưng không cần phải giải mã. MAC bổ sung vào mẫu tin như chữ ký để gửi kèm theo làm bằng chứng xác thực. Người nhận thực hiện tính toán nào đó trên mẫu tin và kiểm tra xem nó có phù hợp với MAC đính kèm không.

Các mã xác thực mẫu tin MAC cung cấp sự tin cậy cho người nhận là mẫu tin không bị thay đổi và từ đích danh người gửi. Cũng có thể sử dụng mã xác thực MAC kèm theo việc mã hoá để bảo mật. Nói chung người ta sử dụng các khóa riêng biệt cho mỗi MAC và có thể tính MAC trước hoặc sau mã hoá, tốt hơn là thực hiện MAC trước và mã hoá sau.

Sử dụng MAC có nhược điểm là MAC phụ thuộc vào cả mẫu tin và cả người gửi, nhưng đôi khi chỉ cần xác thực mẫu tin và thông tin xác thực đó chỉ phụ thuộc mẫu tin để lưu trữ làm bằng chứng cho tính toàn vẹn của nó. Khi đó người ta sử dụng hàm Hash thay vì MAC. Cần lưu ý rằng MAC không phải là chữ ký điện tử, vì cả người gửi và người nhận đều biết thông tin về khóa.

6.2.1 Các tính chất của MAC

MAC là thông tin nén của mẫu tin kết hợp với khóa $MAC = C_K(M)$. Nén bản tin M có độ dài tùy ý. Thuật toán tạo MAC sử dụng khóa mật K. Thuật toán tạo nên dấu xác thực có độ dài cố định (MAC). Thuật toán tạo MAC là hàm nhiều - một, nghĩa là có nhiều bản tin khác nhau nhưng có cùng MAC. Tuy nhiên ta phải lựa chọn hàm MAC sao cho xác suất để các mẫu tin có ý nghĩa có MAC trùng nhau là rất nhỏ. Việc tìm được các mẫu tin như vậy là rất khó khăn.

6.2.2 Yêu cầu đối với MAC

Tuỳ thuộc vào kiểu tấn công mà MAC phải có các tính chất khác nhau để chống đối lại. Nhưng nói chung MAC phải thỏa mãn các điều sau:

- Biết mẫu tin và MAC, không thể tìm được mẫu tin khác có cùng MAC.
- Các MAC cần phải phân bố đều.
- MAC phải phụ thuộc như nhau vào tất cả các bit trong mẫu tin. Tức là khi thay đổi một bit thông tin nào đó, MAC sẽ có những thay đổi kéo theo.

6.2.3 Sử dụng mã đối xứng cho MAC

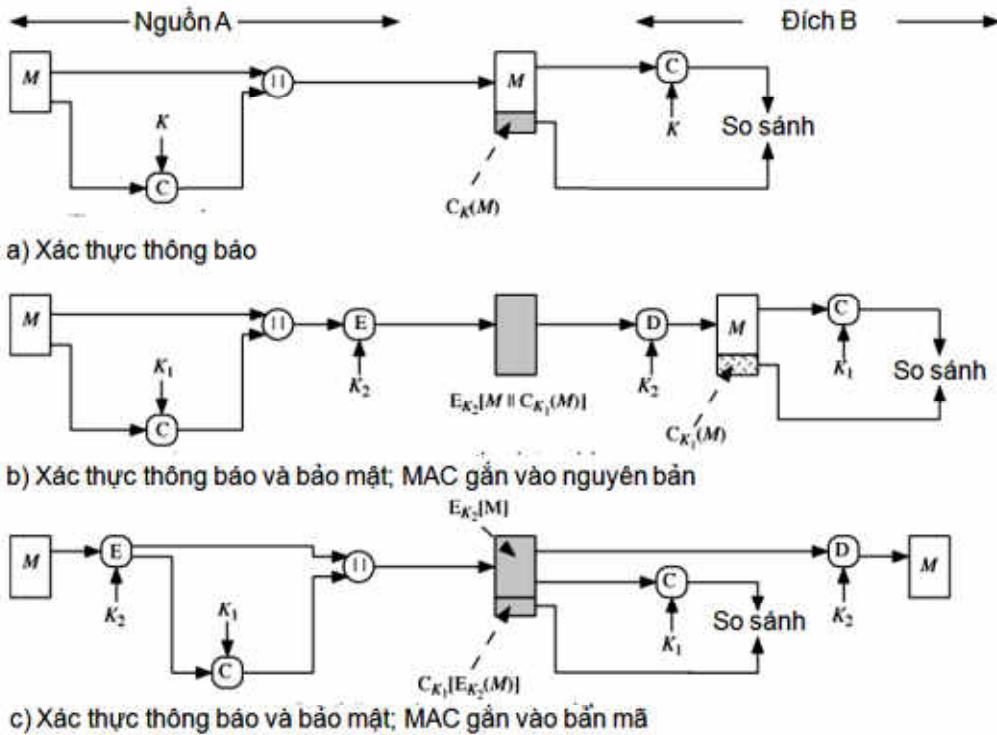
Có thể dùng mã khối với chế độ chuỗi mốc nối bất kỳ và sử dụng khối cuối cùng của mã khối làm MAC của mẫu tin. Thuật toán xác thực dữ liệu (DAA – Data Authentication Algorithm) là MAC được sử dụng rộng rãi dựa trên chế độ DES-CBC, trong đó:

- Sử dụng véc tơ ban đầu $IV = 0$ và bộ đệm 0 của block cuối cùng.
- Và mã mẫu tin sử dụng chuẩn mã dữ liệu DES trong chế độ CBC.
- Gửi lấy block cuối cùng như là MAC của cả mẫu tin hoặc M bit trái nhất ($16 \leq M \leq 64$) của khối cuối cùng. Nhưng bây giờ MAC cuối cùng với kích thước 64 bit cũng là quá nhỏ để đảm bảo an toàn. Do đó người ta tìm cách tạo nên các MAC có kích thước lớn hơn.

6.2.4 Mô hình áp dụng

Mô hình ứng dụng MAC để xác thực thông điệp trình bày trong hình 6-1. Trong mô hình xác thực (a), Trudy nếu chỉ sửa M thành M_T thì giá trị MAC_B sẽ khác MAC_A và Bob phát hiện được. Nếu Trudy muốn sửa thông điệp mà Bob không biết, thì cần sửa luôn MAC_A thành MAC_T tính được từ M_T . Tuy nhiên Trudy không biết khóa K , do đó không tính được MAC_T cần thiết.

Mô hình xác thực không đảm bảo tính bảo mật. Để có tính bảo mật, M và MAC_A cần được mã hóa trước khi truyền đi (mô hình (b) và (c)).



Hình 6.1: Các mô hình áp dụng MAC

6.3 HÀM BĂM (HASH FUNCTION)

Hàm băm tạo ra một giá trị băm có kích thước cố định từ thông báo đầu vào mà không dùng khóa:

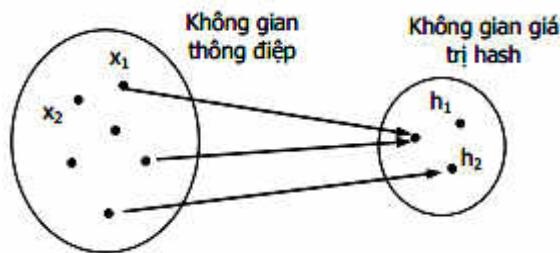
$$h = H(M)$$

Hàm băm không cần giữ bí mật. Giá trị băm gắn kèm với thông báo dùng để kiểm tra tính toàn vẹn của thông báo. Bất kỳ sự thay đổi M nào dù nhỏ cũng tạo ra một giá trị h khác

6.3.1 Yêu cầu đối với hàm băm

Hàm băm có thể áp dụng với thông báo M có độ dài bất kỳ và tạo ra giá trị băm h có độ dài cố định. $H(M)$ dễ dàng tính được với bất kỳ M nào. Từ h rất khó tìm được M sao cho $H(M) = h$ (Tính một chiều). Từ M_1 rất khó tìm được M_2 sao cho $H(M_2) = H(M_1)$ (Tính chống xung đột yếu). Rất khó tìm được (M_1, M_2) sao cho $H(M_1) = H(M_2)$ (Tính chống xung đột mạnh).

Kích thước của input x là bất kỳ còn kích thước của h là nhỏ, ví dụ giả sử kích thước của x là 512 bít còn kích thước của h là 128 bít. Như vậy trung bình có khoảng 2^{384} giá trị x mà có cùng giá trị h . Việc trùng lặp không thể loại bỏ. Tính chống trùng của hàm Hash là yêu cầu rằng việc tìm ra hai input x như vậy thì phải là rất khó về mặt thời gian tính toán.



Hình 6.2: Ánh xạ giữa thông điệp và giá trị hash không phải là song ánh

Một yêu cầu nữa của hàm Hash là kích thước của output h không được quá lớn. Nếu kích thước h lớn thì dễ đạt được tính chống trùng tuy nhiên sẽ tốn dung lượng đường truyền. Vậy kích thước của output h cần thiết là bao nhiêu để thực hiện chống trùng có hiệu quả? Chúng ta sẽ tìm hiểu vấn đề này qua một lý thuyết gọi là bài toán ngày sinh nhật.

6.3.2 Bài toán Ngày Sinh nhật

Bài toán 1: Giả sử trong phòng có 30 người. Vậy xác suất để có hai người có cùng ngày sinh là bao nhiêu phần trăm?

Nguyên lý chuồng bồ câu Dirichlet phát biểu rằng là cần có $365+1 = 366$ người để tìm thấy hai người có cùng ngày sinh với xác suất 100% (để đơn giản, chúng ta bỏ qua năm nhuận). Do đó hầu hết chúng ta sẽ nghĩ rằng với 30 người thì xác suất hai người cùng ngày sinh là nhỏ, chắc chắn nhỏ hơn 50%. Tuy nhiên nếu kiểm tra bằng toán học thì chỉ cần 23 người là đủ để xác suất lớn hơn 50%. Vì vậy bài toán này còn được gọi dưới tên nghịch lý ngày sinh. Ta có thể phát biểu lại bài toán và chứng minh như sau.

Giả sử trong phòng có M người. Hỏi M tối thiểu phải là bao nhiêu để tồn tại hai người có cùng ngày sinh với xác suất lớn hơn 50%?

Ta đánh số thứ tự của M người lần lượt là $0, 1, 2, \dots, M - 1$. Xác suất để người thứ 1 khác ngày sinh với người thứ 0 là $364/365$. Tiếp theo, xác suất để người thứ 2 khác ngày sinh với người thứ 0 và thứ 1 là $363/365$. Tiếp tục như vậy đến người thứ $M-1$ thì xác suất để người này khác ngày sinh với tất cả những người trước là $(365-M+1)/365$. Vậy xác suất để M người này đều có ngày sinh khác nhau là:

$$p(M) = \left(\frac{364}{365}\right)\left(\frac{363}{365}\right)\cdots\left(\frac{365-M+1}{365}\right) = \left(1-\frac{1}{365}\right)\left(1-\frac{2}{365}\right)\cdots\left(1-\frac{M-1}{365}\right)$$

Xét hàm lũy thừa e^x , chúng ta đã biết một xấp xỉ của e^x khi x nhỏ là $e^x=1+x$. Do đó $p(M)$ có thể viết lại thành:

$$p(M) \approx e^{-\frac{1}{365}} \cdot e^{-\frac{2}{365}} \cdot e^{-\frac{3}{365}} \cdots e^{-\frac{M-1}{365}} = e^{-\frac{1+2+3+\dots+(M-1)}{365}} = e^{-\frac{M(M-1)}{2 \times 365}}$$

Dẫn đến xác suất để tồn tại ít nhất hai người có ngày sinh giống nhau là

$$1-p(M) \approx 1-e^{-\frac{M(M-1)}{2 \times 365}}$$

Để xác suất này lớn hơn 50%, chúng ta cho biểu thức trên lớn hơn 0.5:

$$1-e^{-\frac{M(M-1)}{2 \times 365}} \geq \frac{1}{2}$$

$$e^{-\frac{M(M-1)}{2 \times 365}} \leq \frac{1}{2}$$

$$M(M-1) \geq 2 \times 365 \times \log_e 2 \quad (*)$$

và giải bất đẳng thức, ta có được $M \geq 23$.

Bài toán 2: Giả sử bạn đang ở trong một căn phòng với M người khác. Hỏi M tối thiểu là bao nhiêu để tồn tại một người có cùng ngày sinh với bạn với xác suất lớn hơn 50% ?

Xác suất để một người không có cùng ngày sinh với bạn là $364/365$. Như vậy xác suất để M người đều khác ngày sinh với bạn là $(364/365)^M$. Từ đó ta có xác suất để tồn tại ít nhất một người có cùng ngày sinh với bạn là:

$$1-(364/365)^M$$

Để xác suất này lớn hơn 50% thì suy ra $M \geq 253$. Vậy tối thiểu phải có 253 người.

Áp dụng vẫn đề ngày sinh nhật vào hàm băm, ta thấy rằng tính chống trùng mạnh giống bài toán 1, còn tính chống trùng yếu giống bài toán 2. Giả sử số bít của kết xuất h của hàm băm là n bít, như vậy số lượng giá trị có thể có của h là $N = 2^n$. Giả sử thêm rằng $2n$ giá trị băm này đều là *ngẫu nhiên*, có khả năng xuất hiện như nhau. Thay giá trị 365 của bất phương trình (*) bằng 2^n

$$M(M-1) \geq 2 \times 2^n \times \log_2 2$$

Giải bất phương trình trên, ta có xấp xỉ $M \geq \sqrt{2^n} = 2^{n/2}$

Giống như vẫn đề ngày sinh nhật, kết quả trên cho thấy, đối với hàm băm chúng ta phải thử khoảng $2^{n/2}$ thông điệp khác nhau để tìm ra hai thông điệp mà có cùng giá trị băm (xác suất lớn hơn 50%). Nếu $n=128$ thì phải thử khoảng 264 thông điệp, một con số khá lớn, nghĩa là hàm băm này đạt được tính chống trùng mạnh. Do đó việc phá hàm băm cũng khó giống như là việc tấn công vét cạn khóa của mã hóa đối xứng DES.

Tóm lại có thể phát biểu tính chất chống trùng của hàm băm dưới dạng toán học như sau:

$$\nexists x \neq y \text{ sao cho } H(x) = H(y)$$

Nói cách khác:

$$\forall x, y \text{ nếu } H(x) = H(y) \text{ thì } x = y \quad (*)$$

Hai hàm băm được dùng phổ biến hiện nay là MD5 và SHA-1.

6.3.3 Hàm Băm MD5

Tương tự như mã hóa đối xứng, các hàm băm mạnh đều có hiệu ứng lan truyền (*avalanche effect*). Chỉ cần thay đổi 1 bít trong thông điệp đầu vào thì $\frac{1}{2}$ các bít của giá trị băm sẽ thay đổi theo. Điều này làm cho người phá hàm băm không thể thử sai theo kiểu chosen-plaintext, nghĩa là không tồn tại cách tấn công nào khác được và buộc phải thử vét cạn $2^{n/2}$ thông điệp khác nhau, mà chúng ta đã chứng minh là bất khả thi về mặt thời gian.

MD5 được phát minh bởi Ron Rivest, người cũng đã tham gia xây dựng RSA. MD5, viết tắt từ chữ "Message Digest". Kích thước giá trị băm của MD5 là 128 bít. Vào năm 1994 và 1998, một phương pháp tấn công MD5 đã được tìm thấy và một số thông

điệp có cùng giá trị băm MD5 được chỉ ra (vi phạm tính chống trùng mạnh). Tuy vậy, ngày nay MD5 vẫn còn được sử dụng phổ biến.

Thuật toán có đầu vào là một thông điệp có độ dài tùy ý và có đầu ra là một chuỗi có độ dài cố định 128 bit. Thuật toán được thiết kế để chạy trên các máy tính 32 bit.

Thuật toán:

Thông điệp đầu vào có độ dài b bit bất kỳ. Biểu diễn các bit dưới dạng như sau: $m[0] m[1] m[2] \dots m[b-1]$

- *Bước 1:* Các bit gắn thêm: Thông điệp được mở rộng, thêm bit vào phía sau sao cho độ dài của nó (bit) đồng dư với 448 theo môđun 512. Nghĩa là thông điệp được mở rộng sao cho nó còn thiếu 64 bit nữa thì sẽ có một độ dài chia hết cho 512. Việc thêm bit này được thực hiện như sau: một bit '1' được thêm vào sau thông điệp, sau đó các bit '0' được thêm vào để có một độ dài đồng dư với 448 môđun 512.
- *Bước 2:* Gắn thêm độ dài: Dạng biểu diễn 64 bit độ dài b của chuỗi ban đầu được thêm vào phía sau kết quả của bước 1.
- *Bước 3:* Khởi tạo bộ đệm MD: Một bộ đệm 4 từ (A, B, C, D) được dùng để tính mã số thông điệp. Ở đây mỗi A, B, C, D là một thanh ghi 32 bit. Những thanh ghi này được khởi tạo theo những giá trị hex sau:

$A=0x01234567$

$B=0x89abcdef$

$C=0xfedcba98$

$D=0x76543210$

- *Bước 4:* Xử lý thông điệp theo từng khối 16 từ. Định nghĩa các hàm phụ, các hàm này nhận giá trị đầu vào là 3 từ 32 bit và tạo tạo ra một word 32 bit.

$F(X, Y, Z) = XY \vee \text{not}(X) Z$

$G(X, Y, Z) = XZ \vee Y \text{ not}(Z)$

$H(X, Y, Z) = X \text{ xor } Y \text{ xor } Z$

$I(X, Y, Z) = Y \text{ xor } (X \vee \text{not}(Z))$

Bước này sử dụng một bảng 64 giá trị T[1.. 64] được tạo ra từ hàm sin. Gọi T là phần tử thứ i của bảng, thì T là phần nguyên của $4294967296 * |\sin(i)|$, i được tính theo radian.

Thuật toán

```
/* Xử lý với mỗi khối 16 bit từ */
For i = 0 to N/16-1 do
    /* Sao khôi i vào X. */
    For j = 0 to 15 do
        Set X[j] to M[i*16+j].
    end
    AA = A
    BB = B
    CC = C
    DD = D
    /* Vòng 1: Ký hiệu [abcd k s i] là thao tác sau
       a = b + ((a + F(b, c, d) + X[k] + T[i]) <<< s). */
    /* Làm 16 thao tác sau đây*/
    [ABCD 0 7 1] [DABC 1 12 2] [CDAB 2 17 3] [BCDA 3 22 4]
    [ABCD 4 7 5] [DABC 5 12 6] [CDAB 6 17 7] [BCDA 7 22 8]
    [ABCD 8 7 9] [DABC 9 12 10] [CDAB 10 17 11] [BCDA 11 22 12]
    [ABCD 12 7 13] [DABC 13 12 14] [CDAB 14 17 15] [BCDA 15 22 16]
    /* Vòng 2: Ký hiệu [abcd k s i] là thao tác sau đây
       a = b + ((a + G(b, c, d) + X[k] + T[i]) <<< s). */
    /* Làm 16 thao tác sau đây*/
    [ABCD 1 5 17] [DABC 6 9 18] [CDAB 11 14 19] [BCDA 0 20 20]
    [ABCD 5 5 21] [DABC 10 9 22] [CDAB 15 14 23] [BCDA 4 20 24]
    [ABCD 9 5 25] [DABC 14 9 26] [CDAB 3 14 27] [BCDA 8 20 28]
```

```
[ABCD 13 5 29] [DABC 2 9 30] [CDAB 7 14 31] [BCDA 12 20 32]
```

/* Vòng 3: Ký hiệu [abcd k s t] là thao tác sau đây

$a = b + ((a + H(b, c, d) + X[k] + T[i]) <<< s)$. */

/* Làm 16 thao tác sau đây*/

```
[ABCD 5 4 33] [DABC 8 11 34] [CDAB 11 16 35] [BCDA 14 23 36]
```

```
[ABCD 1 4 37] [DABC 4 11 38] [CDAB 7 16 39] [BCDA 10 23 40]
```

```
[ABCD 13 4 41] [DABC 0 11 42] [CDAB 3 16 43] [BCDA 6 23 44]
```

```
[ABCD 9 4 45] [DABC 12 11 46] [CDAB 15 16 47] [BCDA 2 23 48]
```

/* Vòng 4: Ký hiệu [abcd k s t] là thao tác sau đây

$a = b + ((a + I(b, c, d) + X[k] + T[i]) <<< s)$. */

/* Làm 16 thao tác sau đây*/

```
[ABCD 0 6 49] [DABC 7 10 50] [CDAB 14 15 51] [BCDA 5 21 52]
```

```
[ABCD 12 6 53] [DABC 3 10 54] [CDAB 10 15 55] [BCDA 1 21 56]
```

```
[ABCD 8 6 57] [DABC 15 10 58] [CDAB 6 15 59] [BCDA 13 21 60]
```

```
[ABCD 4 6 61] [DABC 11 10 62] [CDAB 2 15 63] [BCDA 9 21 64]
```

/* Tính */

A = A + AA

B = B + BB

C = C + CC

D = D + DD

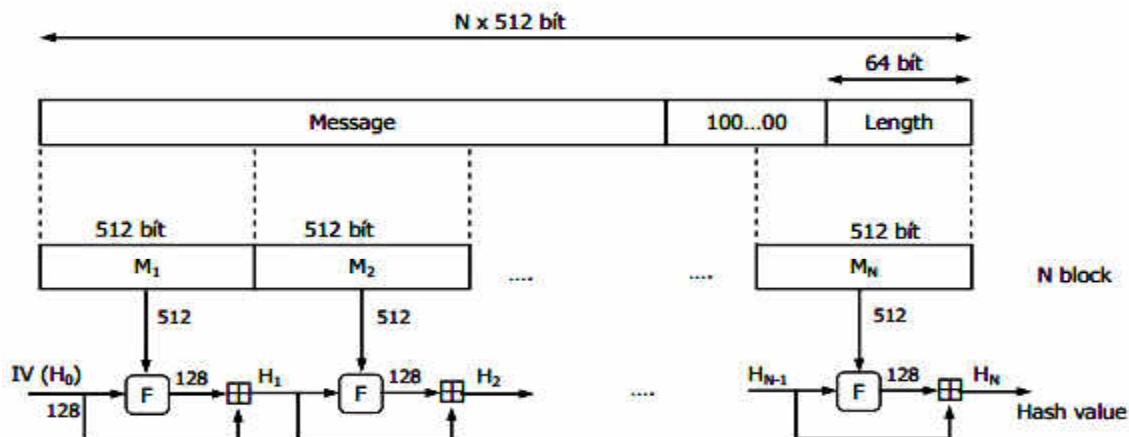
end /* Kết thúc vòng lặp trên i*/

- *Bước 5:* Thông điệp rút gọn = A||B||C||D.

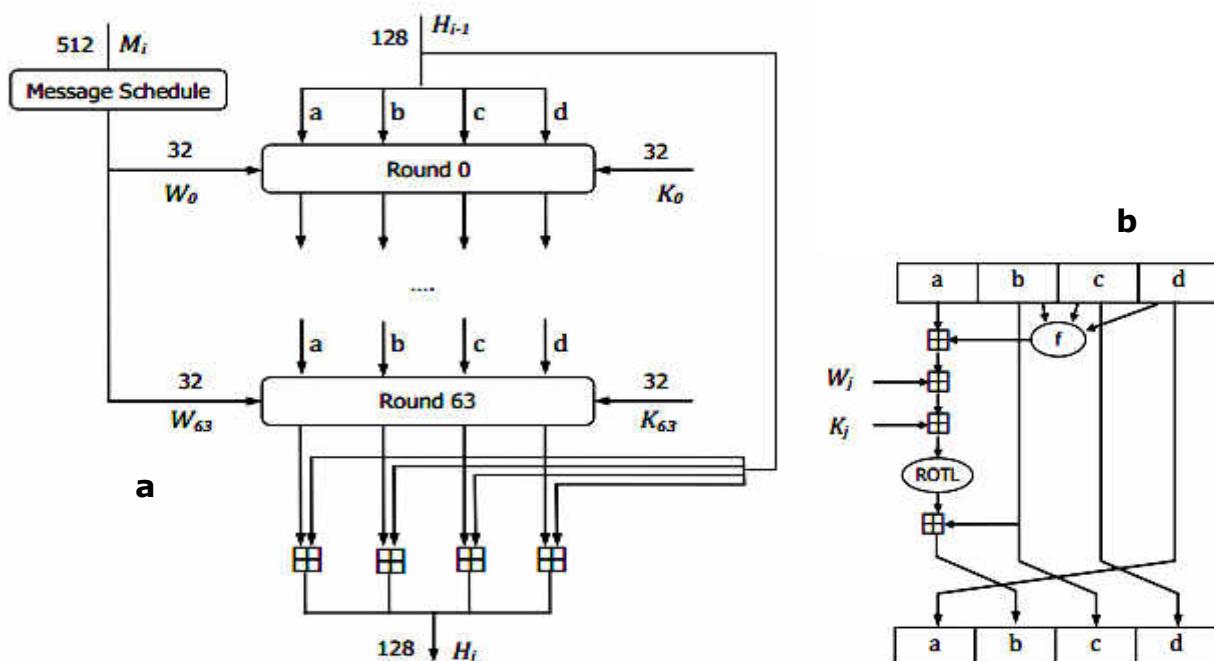
Đánh giá thuật toán MD5

Về tốc độ sinh ra chuỗi cốt yếu thì MD5 chậm hơn so với MD4 nhưng nó lại an toàn hơn rất nhiều so với MD4. Thuật toán mã hóa thông điệp MD5 khá đơn giản để thực hiện, cung cấp một giá trị băm của thông điệp với độ dài tùy ý. Người ta cho rằng độ khó để tìm được 2 thông điệp có cùng giá trị băm là khoảng 264 bước tính, và độ khó để tìm được một thông điệp với giá trị băm cho trước là 2128 bước tính. Tuy nhiên lỗ

hỗng mới phát hiện trong thuật toán MD5 sẽ cho phép kẻ tấn công có thể tạo ra file giả mạo trong vòng vài giờ với loại máy tính đạt chuẩn.



Hình 6-3 Sơ đồ tổng thể



Hình 6.4: cấu trúc của hàm F và sự biến đổi các giá trị abcd trong vòng thứ i.

6.3.4 SHA-1

Vì MD5 không còn được xem là an toàn, nên người ta đã xây dựng thuật toán băm khác. Một trong những thuật toán đó là SHA-1 (Secure Hash Algorithm) mà đã được chính phủ Mỹ chọn làm chuẩn quốc gia. SHA-1 có kích thước giá trị băm là 160 bit. Ngày nay còn có ba phiên bản khác của SHA là SHA-256, SHA-384, SHA-512 mà có kích thước giá trị băm tương ứng là 256, 384 và 512 bit.

Thuật toán SHA-1

Đầu vào của thuật toán là một thông điệp có chiều dài bất kỳ nhỏ hơn 264 bit, SHA-1 cho ra kết quả là một thông điệp rút gọn có độ dài là 160 bit

Mở rộng thông điệp: $f(t; B, C, D)$ được định nghĩa như sau.

$$f(t; B, C, D) = (B \text{ AND } C) \text{ OR } ((\text{NOT } B) \text{ AND } D) \quad (0 \leq t \leq 19)$$

$$f(t; B, C, D) = B \text{ XOR } C \text{ XOR } D \quad (20 \leq t \leq 39)$$

$$f(t; B, C, D) = (B \text{ AND } C) \text{ OR } (B \text{ AND } D) \text{ OR } (C \text{ AND } D) \quad (40 \leq t \leq 59)$$

$$f(t; B, C, D) = B \text{ XOR } C \text{ XOR } D \quad (60 \leq t \leq 79).$$

Thông điệp M được mở rộng trước khi thực hiện băm. Mục đích của việc mở rộng này là để đảm bảo cho thông điệp mở rộng có độ dài là bội số của 512.

Giả sử độ dài của thông điệp là l bit. Thêm bit 1 vào cuối thông điệp, theo sau là k bit 0 (k là số dương không âm nhỏ nhất sao cho $l+1+k=448 \pmod{512}$). Sau đó thêm khối 64 bit là biểu diễn nhị phân của l .

Phân tích thông điệp mở rộng:

Sau khi thông điệp đã được mở rộng, thông điệp mở rộng được phân tích thành N khối 512 bit $M(1), M(2), \dots, M(N)$. Trong đó 512 bit của khối dữ liệu đầu vào có thể được thể hiện bằng 16 từ 32 bit,

Khởi tạo giá trị băm:

Giá trị băm là một chuỗi bit có kích thước bằng kích thước của thông điệp băm (trừ SHA-384) gồm các từ ghép lại. Trong đó $H_j(i)$ là từ j trong giá trị băm ở lần lặp i với $0 \leq i \leq N$ (số block có được sau khi chia văn bản được đệm) và $0 \leq j \leq (\text{số từ trong giá trị băm} - 1)$. Trước khi thực hiện giá trị 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 $H(0)$ tuỳ thuộc vào kích thước thông điệp rút gọn.

SHA-1 sử dụng dãy các hằng số $K(0), \dots, K(79)$ có giá trị như sau:

$$K(t) = 5A827999 \quad (0 \leq t \leq 19)$$

$$K(t) = 6ED9EBA1 \quad (20 \leq t \leq 39)$$

$$K(t) = 8F1BBCDC \quad (40 \leq t \leq 59)$$

$$K(t) = CA62C1D6 \quad (60 \leq t \leq 79).$$

Thuật toán của các bước tính giá trị băm SHA-1

SHA-1 được sử dụng để băm thông điệp M có độ dài l bit thỏa mãn điều kiện $0 \leq l \leq 2^{64}$. Thuật toán sử dụng:

- Một bảng phân bố thông điệp gồm 80 từ 32 bit
- 5 biến 32 bit
- Một giá trị băm gồm 5 từ 32 bit

Kết quả của SHA-1 là một thông điệp rút gọn có độ dài 160 bit. Các từ của bảng phân bố thông điệp được ký hiệu $W(0), W(1), \dots, W(79)$. 5 biến được ký hiệu là a, b, c, d, e. Các từ của giá trị băm ký hiệu $H_0(i), H_1(i), H_2(i), H_3(i), H_4(i)$. $H(0)$ giữ giá trị băm ban đầu và được thay thế bằng các giá trị băm thành công. $H(i)$ sau mỗi khôi thông điệp được xử lý và kết thúc bằng giá trị băm cuối cùng $H(N)$.

Tính toán thông điệp băm

Định nghĩa: $S^n(X) = (X << n) \text{ or } (X >> 32-n)$.

$X << n$ có nghĩa là loại bỏ từ trái sang phải n bit và thêm vào kết quả n số 0 vào bên phải. $X >> n$ có nghĩa là loại bỏ từ phải qua trái n bit và thêm vào kết quả n số 0 vào bên trái.

Khởi tạo H

```

H0 = 67452301 ; H1 = EFCDAB89
H2 = 98BADCFE ; H3 = 10325476
H4 = C3D2E1F0.

```

Chia M(i) thành 16 từ W(0), W(1), ..., W(15)

```

For t = 16 to 79
    W(t) = S^1(W(t-3) XOR W(t-8) XOR W(t-14) XOR W(t-16)).

    Đặt a=H0, b=H1, c=H2, d=H3, e=H4

For t = 0 to 79 do
    TEMP = S^5(A) + f(t;B, C, D) + E + W(t) + K(t);
    e = d; d = c; c = S^30(b); b = a; a = TEMP;

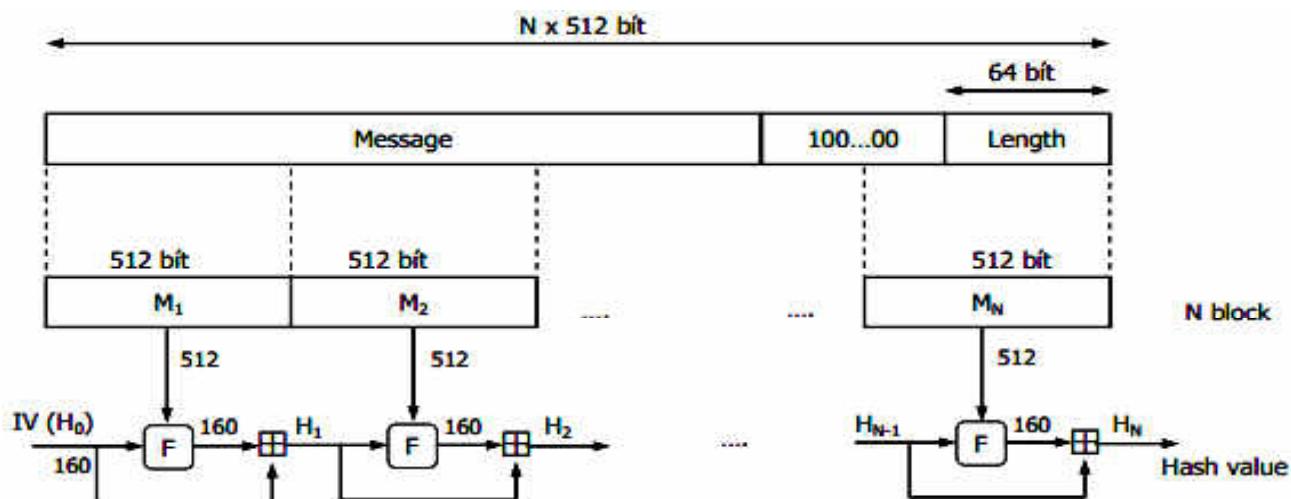
```

Đặt $H_0 = H_0 + a, H_1 = H_1 + b, H_2 = H_2 + c, H_3 = H_3 + d, H_4 = H_4 + e$.

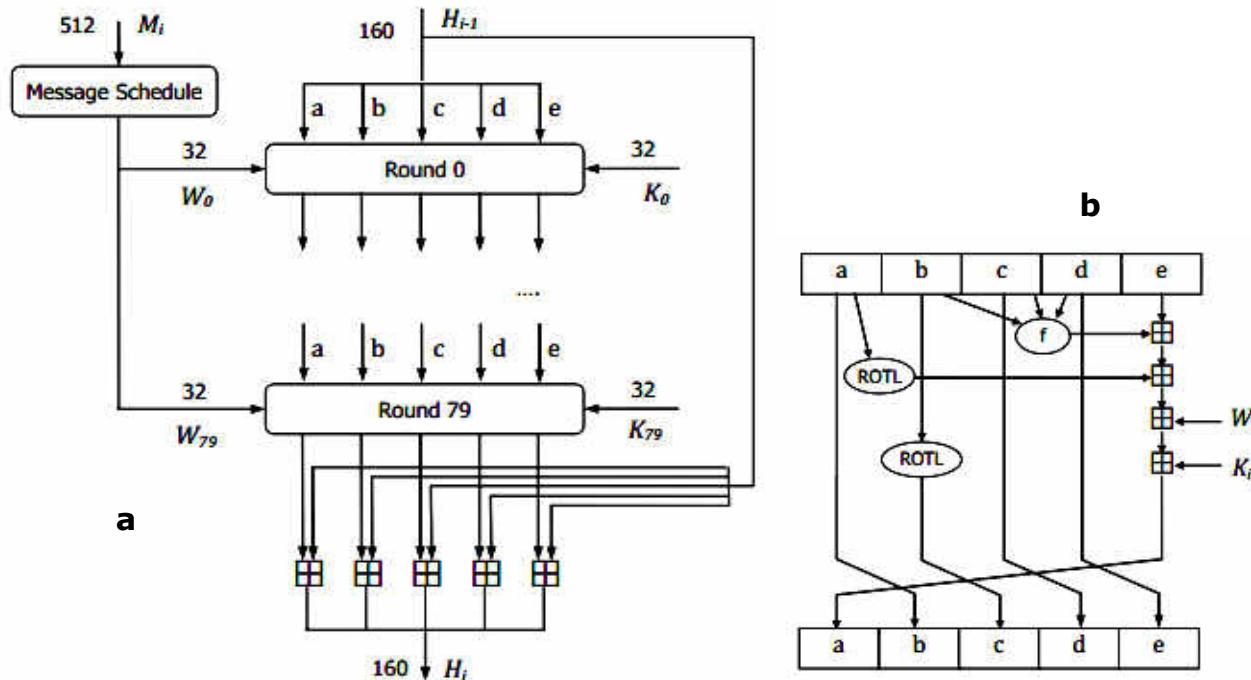
Sau khi tính toán được hết $M(n)$, thông điệp rút gọn là một chuỗi 160 bit là biểu diễn của 5 từ: $H_0 \ H_1 \ H_2 \ H_3 \ H_4$

Đánh giá thuật toán

- SHA-1 được xem là an toàn đối với hiện tượng đụng độ vì rất khó tìm được hai thông điệp khác nhau có giá trị băm giống nhau.
- SHA-1 được coi là chuẩn của việc bảo vệ các kênh liên lạc trực tuyến tồn tại trong 9 năm qua.
- SHA-1 được thiết kế cho bộ xử lý 32 bit, thế hệ sắp tới của máy tính dùng các bộ xử lý 64 bit mà SHA-1 không hiệu quả trên bộ xử lý này.
- Tháng 2 năm 2005 SHA-1 bị tấn công bởi 3 chuyên gia người Trung Quốc. Thuật toán này đã bị giải mã thông qua phương pháp tính phân bổ.



Hình 6.5: Sơ đồ tổng thể của SHA1



Hình 6.6: Cấu trúc của hàm F (a) và biến đổi abcde trong vòng thứ i (b)

6.3.5 HMAC

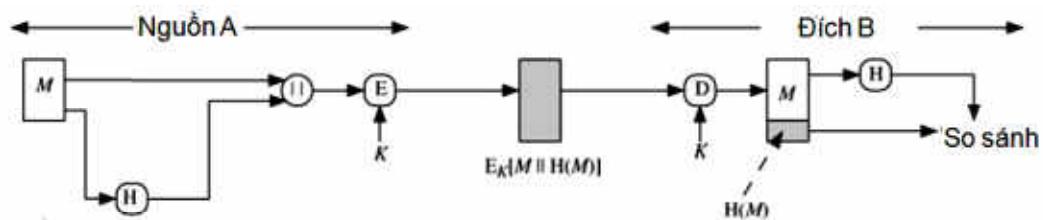
Hàm băm cũng có thể dùng để tính MAC bằng cách truyền thêm khóa bí mật K vào hàm băm. Lúc này, giá trị kết xuất được gọi là HMAC.

$$\text{HMAC} = H(M||K)$$

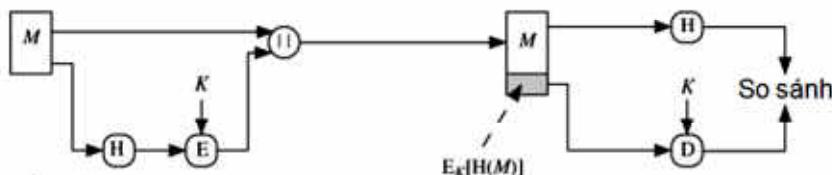
6.4 MỘT SỐ ỨNG DỤNG CỦA HÀM BĂM

6.4.1 Các kiểu giao thức xử lý dữ liệu ứng dụng hàm băm

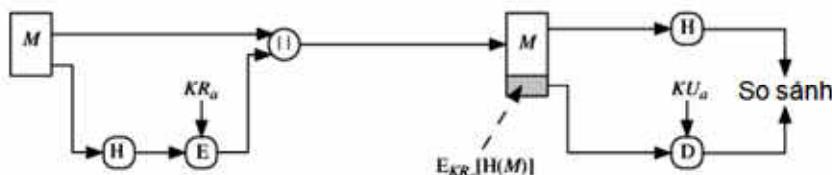
Hàm băm được ứng dụng trong việc kiểm tra tính toàn vẹn của dữ liệu. Hình 6-9a minh họa hàm băm kết hợp với hệ mã khóa bí mật để xác thực và bảo mật thông báo. Hình 6-9b minh họa áp dụng hàm băm để xác thực thông báo, trong đó giá trị băm được bí mật. Hình 6-9c minh họa hàm băm kết hợp với mã khóa công khai với khóa riêng hẻ để tạo chữ ký điện tử vừa xác thực thông báo vừa đảm bảo dịch vụ chống chối bỏ. Hình 6-9d và 6-9e minh họa xác thực thông báo dựa vào hàm băm và giá trị độn bí mật.



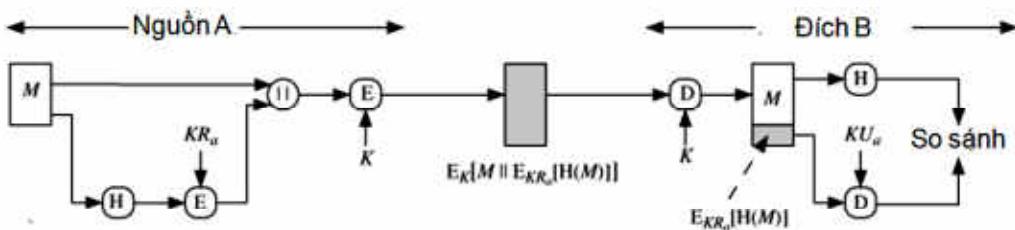
a) Xác thực thông báo và bảo mật; mã băm gắn vào nguyên bản



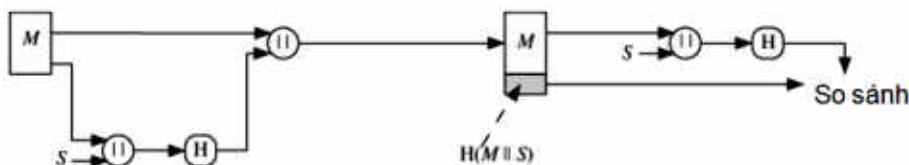
b) Xác thực thông báo; mã băm được mã hóa sử dụng phương pháp đổi xứng



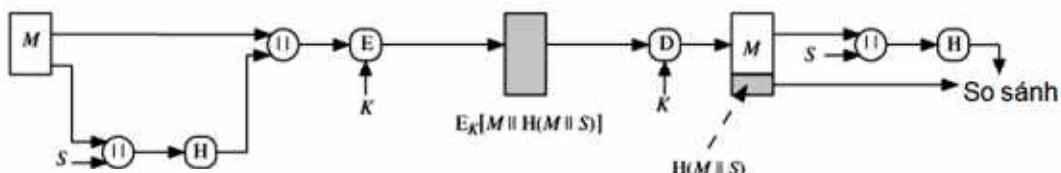
c) Xác thực thông báo; mã băm được mã hóa sử dụng phương pháp khóa công khai



d) Xác thực bằng mã hóa khóa công khai và bảo mật bằng mã hóa đổi xứng



e) Xác thực không cần mã hóa nhờ hai bên chia sẻ một giá trị bí mật chung



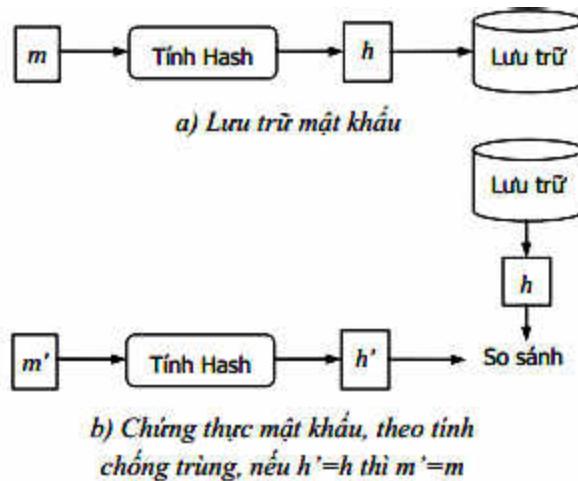
f) Xác thực nhờ một giá trị bí mật chung; bảo mật bằng phương pháp đổi xứng

Hình 6.7: Các kiểu giao thức xử lý dữ liệu có áp dụng hàm băm

6.4.2 Lưu trữ Mật khẩu

Hầu hết các ứng dụng phần mềm ngày nay, dù trên máy đơn hay trên web, đều có chứng thực người sử dụng. Nghĩa là để sử dụng ứng dụng, người sử dụng phải qua một cơ chế chứng thực username và mật khẩu, và từ đó được cung cấp các quyền sử dụng phần mềm khác nhau. Do đó vấn đề bảo mật mật khẩu là vấn đề quan trọng đối với mọi phần mềm.

Mật khẩu người sử dụng thường gồm các chữ cái thường và hoa, cộng thêm các chữ số. Giả sử mật khẩu được lưu trữ dưới dạng thường, không mã hóa, tại một nơi nào đó trên máy tính cá nhân hay máy chủ, trong một file dữ liệu hay trong hệ quản trị cơ sở dữ liệu. Như vậy sẽ xuất hiện một nguy cơ là có một người khác, hoặc là người quản trị administrator, hoặc là hacker, có thể mở được file dữ liệu hoặc cơ sở dữ liệu, và xem trộm được mật khẩu. Như vậy mật khẩu không thể được giữ bí mật tuyệt đối.



Hình 6.8: Dùng hàm Hash để lưu trữ mật khẩu

Một phương pháp để bảo vệ mật khẩu là dùng mã hóa, chương trình phần mềm sẽ dùng một khóa bí mật để mã hóa mật khẩu trước khi lưu mật khẩu xuống file hay cơ sở dữ liệu. Do đó tránh được vấn đề xem trộm mật khẩu. Tuy nhiên phương pháp này có yếu điểm là lại phải lo bảo vệ khóa bí mật này. Nếu khóa bí mật bị lộ thì việc mã hóa không còn ý nghĩa.

Phương pháp bảo vệ mật khẩu hiệu quả nhất là dùng hàm băm. Khi người sử dụng đăng ký mật khẩu, giá trị băm của mật khẩu được tính bằng một hàm băm nào đó (MD5 hay SHA-1, ...) Giá trị băm được lưu trữ vào file hay cơ sở dữ liệu. Vì hàm băm

là một chiều, nên dù biết được giá trị băm và loại hàm băm, hacker cũng không thể suy ra được mật khẩu. Khi người sử dụng đăng nhập, mật khẩu đăng nhập được tính giá trị băm và so sánh với giá trị băm đang được lưu trữ. Do tính chống trùng, chỉ có một mật khẩu duy nhất có giá trị băm tương ứng, nên không ai khác ngoài người sử dụng có mật khẩu đó mới có thể đăng nhập ứng dụng.

	username	password	email
1	admin	nhx64312	nguyen@yahoo.com
2	devil	kin32xz	nam@hotmail.com
3	vampire	62ntt34	hung@gmail.com

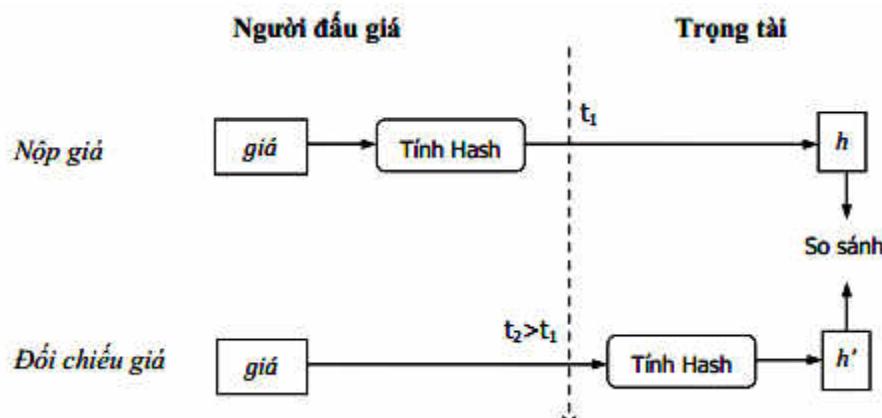
	username	password	Salt	email
1	admin	23dacd8cd768c95ad2e63cdc399c0535	64f84a9bdec1999e43c97ab12f8d9a36	nguyen@yahoo.com
2	devil	d400c472ab7a09ba87bf5c9715bbe118	66436db921558ae452f1e76a44e40aac	nam@hotmail.com
3	vampire	0d7b12ce9cf7ef3534aa5fee204eb0f5	4c798886f01910bad5a85fbec1c4bfea	hung@gmail.com

Hình 6.9: Minh họa mật khẩu rõ và được băm

6.4.3 Đấu Giá Trực Tuyến

Phương pháp lưu trữ mật khẩu bằng giá trị Hash cũng được áp dụng tương tự cho việc đấu giá trực tuyến bằng hình thức đấu giá bí mật. Giả sử Alice, Bob và Trudy cùng tham gia đấu giá, họ sẽ cung cấp mức giá của mình cho trọng tài. Các mức giá này được giữ bí mật cho đến khi cả ba đều nộp xong. Nếu ai là người đưa ra mức giá cao nhất thì thắng thầu. Điểm quan trọng của phương pháp đấu giá này là giá của Alice, Bob, và Trudy phải được giữ bí mật trước khi công bố. Giả sử mức giá của Alice là 100, mức giá của Bob là 110, nếu Trudy thông đồng với trọng tài và biết được giá của Alice và Bob, Trudy có thể đưa ra mức giá 111 và thắng thầu.

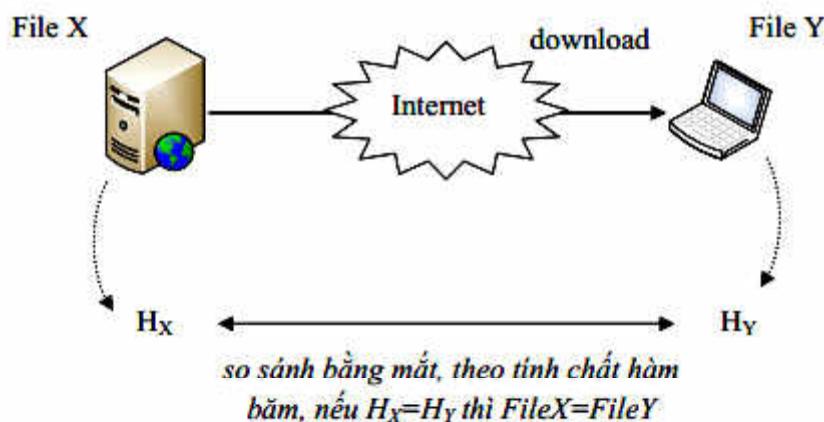
Có thể tránh những hình thức lừa đảo như vậy bằng cách sử dụng hàm băm. Từ mức giá bỏ thầu, Alice và Bob sẽ tính các giá trị băm tương ứng và chỉ cung cấp cho trọng tài các giá trị băm này. Vì hàm băm là một chiều, nếu trọng tài và Trudy bắt tay nhau thì cũng không thể biết được giá của Alice và Bob là bao nhiêu. Đến khi công bố, Alice, Bob và Trudy sẽ đưa ra mức giá của mình. Trọng tài sẽ tính các giá trị băm tương ứng và so sánh với các giá trị băm đã nộp để bảo đảm rằng mức giá mà Alice, Bob và Trudy là đúng với ý định ban đầu của họ. Vì tính chống trùng của hàm băm nên Alice, Bob và Trudy không thể thay đổi giá so với ý định ban đầu.

**Hình 6.10: Đấu giá bí mật**

6.4.4 Dowload File

Khi chúng ta download file từ mạng internet, nếu chất lượng mạng không tốt thì có thể xảy ra lỗi trong quá trình download làm cho file tại máy client khác với file trên server. Hàm băm có thể giúp chúng ta phát hiện ra những trường hợp bị lỗi như vậy.

Gọi file cần download trên server là X, và giá trị hash theo MD5 của file X mà server đã tính sẵn và cung cấp trên trang web là H_x (có thể xem bằng mắt). Gọi Y là file mà người sử dụng download được tại máy. Người sử dụng sẽ tính giá trị MD5 H_y cho file Y. Như vậy nếu $H_x = H_y$ thì theo tính chống trùng của hàm hash, file Y hoàn toàn giống file X và quá trình download không xảy ra lỗi.

**Hình 6.11: Kiểm tra tính toàn vẹn của tập tin**

TÓM TẮT

Bài học cung cấp kiến thức về vấn đề xác thực mẫu tin sử dụng mã xác thực, hàm băm và các ứng dụng của nó.

Xác thực mẫu tin nhằm bảo vệ tính toàn vẹn của mẫu tin. Bảo vệ mẫu tin không bị thay đổi hoặc có các biện pháp phát hiện nếu mẫu tin bị thay đổi trên đường truyền. Đồng thời, xác thực mẫu tin sẽ giúp kiểm chứng danh tính và nguồn gốc. Xem xét mẫu tin có đúng do người xưng tên gửi không hay một kẻ mạo danh nào khác gửi. Từ đó giải quyết bài toán chống chối từ bản gốc. Trong trường hợp cần thiết, bản thân mẫu tin chứa các thông tin chứng tỏ chỉ có người xưng danh gửi, không một ai khác có thể làm điều đó. Như vậy người gửi không thể từ chối hành động gửi, thời gian gửi và nội dung của mẫu tin.

Mã xác thực mẫu (MAC – Message Authentication Code) tin sinh ra bởi một thuật toán mà tạo ra một khối thông tin nhỏ có kích thước cố định. MAC phụ thuộc vào cả mẫu tin và khóa nào đó. MAC giống như mã nhưng không cần phải giải mã. MAC bổ sung vào mẫu tin như chữ ký để gửi kèm theo làm bằng chứng xác thực. Người nhận thực tạo MAC trên mẫu tin nhận được và kiểm tra xem nó có phù hợp với MAC đính kèm không. Có thể dùng mã khối với chế độ chuỗi móc nối bất kỳ và sử dụng khối cuối cùng của mã khối làm MAC của mẫu tin. Thuật toán xác thực dữ liệu (DAA – Data Authentication Algorithm) là MAC được sử dụng rộng rãi dựa trên chế độ DES-CBC.

Hàm băm tạo ra một giá trị băm có kích thước cố định từ thông báo đầu vào mà không dùng khóa: $h = H(M)$. Hàm băm không cần giữ bí mật. Giá trị băm gắn kèm với thông báo dùng để kiểm tra tính toàn vẹn của thông báo. Bất kỳ sự thay đổi M nào dù nhỏ cũng tạo ra một giá trị h khác.

MD5 (Message Digest) được phát minh bởi Ron Rivest. Kích thước giá trị băm của MD5 là 128 bít. Vì MD5 không còn được xem là an toàn, nên người ta đã xây dựng thuật toán băm khác. Mỹ chọn làm chuẩn quốc gia. SHA-1 có kích thước giá trị băm là 160 bít.

Ngoài ra, bài học còn giới thiệu các ứng dụng của hàm băm: chữ ký số, lưu trữ mật khẩu, đấu giá trực tuyến, download file.

CÂU HỎI ÔN TẬP

Câu 1: Để bảo đảm tính chứng thực dùng mã hóa đối xứng hay mã hóa khóa công khai, bắn rõ phải có tính chất gì?

Câu 2: Nếu bắn rõ là một dãy bít ngẫu nhiên, cần làm gì để bắn rõ trở thành có cấu trúc?

Câu 3: Sử dụng MAC để chứng thực có ưu điểm gì so với chứng thực bằng mã hóa đối xứng?

Câu 4: Về mặt lý thuyết, giá trị Hash có thể trùng không? Vậy tại sao nói giá trị Hash có thể xem là “dấu vân tay của thông điệp”?

Câu 5: Tại sao để chứng thực một thông điệp M , người ta chỉ cần mã hóa khóa công khai giá trị Hash của M là đủ? Thực hiện như vậy có lợi ích gì hơn so với cách thức mã hóa toàn bộ M ?

Câu 6: Trình bày các đặc điểm của hàm băm MD5 và SHA-1?

Câu 7: Trình bày các ứng dụng của hàm băm?

BÀI 7: BẢO MẬT MẠNG NỘI BỘ VÀ AN TOÀN IP

Sau khi học xong bài này, sinh viên hiểu:

- *Giải pháp bảo mật mạng nội bộ theo mô hình Domain sử dụng Kerberos.*
- *Hiểu được cách thức xác thực thành viên và cơ chế trao đổi khóa bí mật trong miền.*
- *Hiểu được giải pháp bảo mật dữ liệu đi trong môi trường mạng IPSEC: về cơ chế xác thực, trao đổi bộ tham số mã hóa, khóa (IKE) và cách xử lý dữ liệu đi trong môi trường mạng (AH, ESP).*

7.1 BẢO MẬT MẠNG CỤC BỘ - KERBEROS

Đây là mô hình Hệ thống khóa máy chủ tin cậy của MIT (Trường Đại học Kỹ thuật Massachusetts) để cung cấp xác thực có bến thứ ba dùng khóa riêng và tập trung. Cho phép người sử dụng truy cập vào các dịch vụ phân tán trong mạng. Tuy nhiên không cần thiết phải tin cậy mọi máy trạm, thay vì đó chỉ cần tin cậy máy chủ xác thực trung tâm. Các yêu cầu của Kerberos gồm có An toàn, Tin cậy, và Trong suốt. Đã có hai phiên bản đang sử dụng là: Kerberos 4 (KB4) và Kerberos 5 (KB5).

7.1.1 KB4

KB4 là sơ đồ xác thực dùng bến thứ ba cơ bản và có máy chủ xác thực (AS – Authentication Server). Người dùng thỏa thuận với AS về danh tính của mình, AS cung cấp sự tin cậy xác thực thông qua thẻ cấp thẻ TGT (Ticket Granting Ticket) và máy chủ cung cấp thẻ (TGS – Ticket Granting Server). Người sử dụng thường xuyên yêu cầu TGS cho truy cập đến các dịch vụ khác dựa trên thẻ cấp thẻ TGT của người sử dụng.

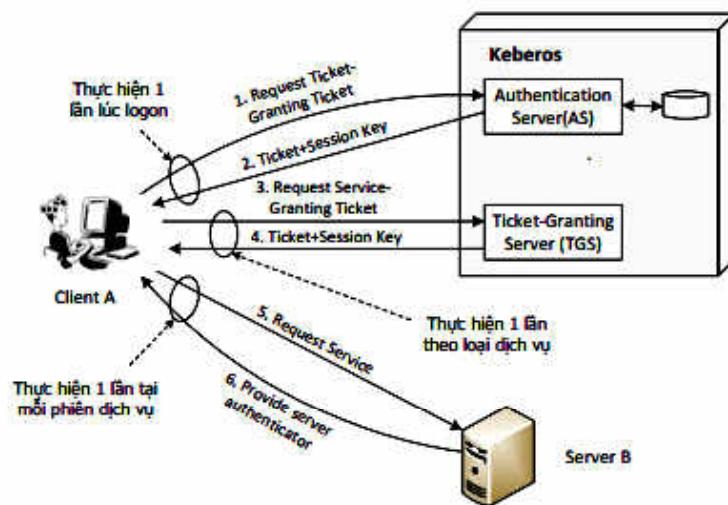
Trao đổi KB4: Người sử dụng nhận thẻ được cấp từ máy chủ xác thực AS, mỗi thẻ cho một phiên làm việc và cũng nhận thẻ cấp dùng dịch vụ (service granting ticket) từ TGT. Mỗi thẻ dùng cho một dịch vụ khác nhau được yêu cầu, thông qua việc trao đổi giữa máy chủ/trạm để nhận được dịch vụ.

7.1.2 KB5

Kerberos 5 được phát triển vào giữa những năm 1990, được thiết kế theo chuẩn RFC 1510. Nó cung cấp những cải tiến so với phiên bản 4, cụ thể hướng tới các thiếu sót về môi trường, thuật toán mã, thủ tục mạng thứ tự byte, thời gian sử dụng thẻ, truyền tiếp xác thực, xác thực lãnh địa con. Và các sự khác biệt về kỹ thuật như: mã kép, các dạng sử dụng không chuẩn, khóa phiên, chống tấn công mật khẩu.

7.1.3 Chi tiết mô hình Kerberos

Kerberos là một giao thức xác thực mạng, nó cho phép các cá nhân giao tiếp với nhau trên một mạng không an toàn bằng cách xác thực người dùng này với người dùng khác theo một cơ chế bảo mật và an toàn. Kerberos ngăn chặn việc nghe trộm thông tin cũng như tấn công thay thế và đảm bảo tính toàn vẹn của dữ liệu. Kerberos hoạt động theo mô hình máy trạm/máy chủ và nó thực hiện quá trình xác thực 2 chiều - cả người dùng và dịch vụ xác thực lẫn nhau. Kerberos được xây dựng dựa trên mô hình mã hóa khóa đối xứng và đòi hỏi một thành phần thứ ba tin cậy tham gia vào quá trình xác thực.



Hình 7.1: Mô hình chứng thực và trao đổi khóa phiên Kerberos

Mô tả giao thức:

Kerberos sử dụng một đối tác tin cậy thứ ba để thực hiện quá trình chứng thực được gọi là Trung tâm phân phối khóa bao gồm 2 phần riêng biệt: một máy chủ chứng thực (AS) và một máy chủ cấp thẻ (TGS). Kerberos làm việc dựa trên các thẻ để thực hiện quá trình chứng thực người dùng.

Kerberos duy trì một cơ sở dữ liệu chứa các khóa bí mật. Mỗi thực thể trên mạng (máy trạm hoặc máy chủ) đều chia sẻ một khóa bí mật chỉ giữa bản thân nó với Kerberos. Để thực hiện quá trình giao tiếp giữa 2 thực thể, Kerberos tạo ra một khóa phiên. Khóa này dùng để bảo mật quá trình tương tác giữa các thực thể với nhau.

Ký hiệu

C: Client

AS: Server xác thực

V: Server dịch vụ

IDC: Danh tính người dùng trên C

IDV: Danh tính của V

PC: Mật khẩu của người dùng trên C

ADC: Địa chỉ mạng của C

KV: Khóa bí mật chia sẻ bởi AS và V

||: Phép ghép

TGS: Server cấp thẻ

TS: Nhãn thời gian

Lược đồ tuần tự trao đổi gói tin:

(a) Trao đổi với dịch vụ xác thực: để có thẻ cấp thẻ

(1) $C \rightarrow AS: IDC \parallel ID_{TGS} \parallel TS_1$

(2) $AS \rightarrow C: EKC[KC, tgs] \parallel ID_{TGS} \parallel TS_2 \parallel H_{\text{tgs}} \parallel Th_{\text{tgs}}$

$$Th_{\text{tgs}} = EK_{\text{tgs}}[KC, tgs \parallel IDC \parallel ADC \parallel ID_{TGS} \parallel TS_2 \parallel H_{\text{tgs}}]$$

(b) Trao đổi với dịch vụ cấp thẻ: để có thẻ dịch vụ

(3) C → TGS: IDV || Thẻtgs || DấuC

(4) TGS → C: EKC, tgs[KC, V || IDV || TS4 || ThẻV]

ThẻV = EKV[KC, V || IDC || ADC || IDV || TS4 || Hạn4]

DấuC = EKC, tgs[IDC || ADC || TS3]

(c) Trao đổi xác thực client/server: để có dịch vụ

(5) C → V: ThẻV || DấuC

(6) V → C: EKC, V[TS5 + 1]

DấuC = EKC, V[IDC || ADC || TS5]

Hoạt động của Kerberos:

Quá trình hoạt động của giao thức (AS = Máy chủ xác thực, TGS = Máy chủ cấp thẻ, C = Máy trạm, S = Dịch vụ):

1. Người dùng nhập vào tên truy cập và mật khẩu ở phía máy trạm.
2. Máy trạm thực hiện thuật toán băm một chiêu trên mật khẩu được nhập vào và nó trở thành khóa bí mật của máy trạm.
3. Máy trạm gửi một thông điệp dưới dạng bản rõ đến AS để yêu cầu dịch vụ. Không có khóa bí mật cũng như mật khẩu nào được gửi đến AS.
Nếu có, nó gửi ngược lại cho máy trạm 2 thông điệp:
 - Thông điệp A: chứa khóa phiên Máy trạm/TGS được mã hóa bởi khóa bí mật của người dùng.
 - Thông điệp B: chứa Thẻ (bao gồm ID của máy trạm, địa chỉ mạng của máy trạm, kỳ hạn thẻ có giá trị và một khóa phiên máy trạm/TGS) được mã hóa sử dụng khóa bí mật của TGS.
5. Khi máy trạm nhận được thông điệp A và B, nó giải mã thông điệp A để lấy khóa phiên máy trạm/TGS. Khóa phiên này được sử dụng cho quá trình giao tiếp

theo với TGS. Ở đây máy trạm không thể giải mã thông điệp B bởi vì nó được mã hóa bởi khóa bí mật của TGS.

6. Khi yêu cầu dịch vụ (S), máy trạm gởi 2 thông điệp sau đến TGS:

- Thông điệp C: Gồm thông điệp B và ID của dịch vụ được yêu cầu.
- Thông điệp D: chứa Authenticator (gồm ID máy trạm và nhãn thời gian - timestamp) được mã hóa bởi khóa phiên Máy trạm/TGS.

7. Khi nhận được thông điệp C và D, TGS giải mã thông điệp D sử dụng khóa phiên máy trạm/TGS và gởi 2 thông điệp ngược lại cho máy trạm:

- Thông điệp E: chứa thẻ (máy trạm đến máy chủ) (bao gồm ID máy trạm, địa chỉ mạng của máy trạm, kỳ hạn thẻ có giá trị và một khóa phiên máy trạm/dịch vụ) được mã hóa bởi khóa bí mật của dịch vụ.
- Thông điệp F: chứa khóa phiên của máy trạm/máy chủ được mã hóa bởi khóa phiên máy trạm/TGS.

8. Khi nhận được thông điệp E và F, máy trạm sau đó gởi một Authenticator mới và một thẻ (máy trạm đến máy chủ) đến máy chủ chứa dịch vụ được yêu cầu.

- Thông điệp G: chứa thẻ (máy trạm đến máy chủ) được mã hóa sử dụng khóa bí mật của máy chủ.
- Thông điệp H: một Authenticator mới chứa ID máy trạm, Timestamp và được mã hóa sử dụng khóa phiên máy trạm/máy chủ.

9. Sau đó, máy chủ giải mã thẻ sử dụng khóa bí mật của chính nó, và gởi một thông điệp cho máy trạm để xác nhận tính hợp lệ thực sự của máy trạm và sự sẵn sàng cung cấp dịch vụ cho máy trạm.

- Thông điệp I: chứa giá trị Timestamp trong Authenticator được gởi bởi máy trạm sẽ được cộng thêm 1, được mã hóa bởi khóa phiên máy trạm/máy chủ.

10. Máy trạm sẽ giải mã sự xác nhận này sử dụng khóa chia sẻ giữa nó với máy chủ, và kiểm tra xem giá trị timestamp có được cập nhật đúng hay không. Nếu đúng, máy trạm có thể tin tưởng máy chủ và bắt đầu đưa ra các yêu cầu dịch vụ gởi đến máy chủ.

11. Máy chủ cung cấp dịch vụ được yêu cầu đến máy trạm.

7.1.4 Hạn chế của Kerberos

Kerberos thích hợp cho việc cung cấp các dịch vụ xác thực, phân quyền và bảo đảm tính mật của thông tin trao đổi trong phạm vi một mạng hay một tập hợp nhỏ các mạng. Tuy nhiên, nó không thật thích hợp cho một số chức năng khác, chẳng hạn như ký điện tử (yêu cầu đáp ứng cả hai nhu cầu xác thực và bảo đảm không chối cãi được). Một trong những giả thiết quan trọng của giao thức Kerberos là các máy chủ trên mạng cần phải tin cậy được. Ngoài ra, nếu người dùng chọn những mật khẩu dễ đoán thì hệ thống dễ bị mất an toàn trước kiểu tấn công từ điển, tức là kẻ tấn công sẽ sử dụng phương thức đơn giản là thử nhiều mật khẩu khác nhau cho đến khi tìm được giá trị đúng.

Do hệ thống hoàn toàn dựa trên mật khẩu để xác thực người dùng, nếu bản thân các mật khẩu bị đánh cắp thì khả năng tấn công hệ thống là không có giới hạn. Điều này dẫn đến một yêu cầu rất căn bản là Trung tâm phân phối khóa cần được bảo vệ nghiêm ngặt. Nếu không thì toàn bộ hệ thống sẽ trở nên mất an toàn.

7.2 AN TOÀN IP - IPSEC

IPSec (Internet Protocol Security) thực hiện mã hóa và xác thực ở lớp mạng. Nó cung cấp một giải pháp an toàn dữ liệu đầu cuối cũng như liên kết mạng. Vì vậy vẫn đề an toàn được thực hiện mà không cần thay đổi các ứng dụng cũng như các hệ thống cuối. Các gói mã hóa có khuôn dạng giống như gói tin IP thông thường, nên chúng dễ dàng được định tuyến qua mạng Internet mà không phải thay đổi các thiết bị mạng trung gian, qua đó cho phép giảm đáng kể các chi phí cho việc triển khai và quản trị. IPSec cung cấp bốn chức năng quan trọng sau:

- *Bảo mật (mã hóa) - Confidentiality:* Bên gửi có thể mã hóa dữ liệu trước khi truyền chúng qua mạng. Bằng cách đó, không ai có thể nghe trộm trên đường truyền. Nếu giao tiếp bị ngăn chặn, dữ liệu không thể đọc được.
- *Toàn vẹn dữ liệu - Data integrity:* Bên nhận có thể xác minh các dữ liệu được truyền qua mạng Internet mà không bị thay đổi. IPSec đảm bảo toàn vẹn dữ liệu bằng cách sử dụng giá trị băm.

- *Xác thực - Authentication:* Xác thực đảm bảo kết nối được thực hiện và các đúng đối tượng. Người nhận có thể xác thực nguồn gốc của gói tin, bảo đảm, xác thực nguồn gốc của thông tin.
- *Antireplay protection:* xác nhận mỗi gói tin là duy nhất và không trùng lặp.

7.2.1 Các ứng dụng của IPSec

IPSec cung cấp dịch vụ bảo mật cho các công nghệ mạng khác như: Xây dựng mạng riêng ảo an toàn trên Internet. Từ đó, tiết kiệm chi phí thiết lập và quản lý mạng riêng. Truy nhập từ xa an toàn thông qua Internet nhằm tiết kiệm chi phí đi lại. Tăng cường an ninh thương mại điện tử. Hỗ trợ thêm cho các giao thức an ninh có sẵn của các ứng dụng Web và thương mại điện tử.

7.2.2 Lợi ích

Tại tường lửa hoặc bộ định tuyến, IPSec đảm bảo an ninh cho mọi luồng thông tin vượt biên. Tại tường lửa, IPSec ngăn chặn thâm nhập trái phép từ Internet vào. IPSec nằm dưới tầng giao vận, do vậy trong suốt với các ứng dụng. IPSec có thể trong suốt với người dùng cuối. IPSec có thể áp dụng cho người dùng đơn lẻ. IPSec bảo vệ an ninh kiến trúc định tuyến.

7.2.3 Công cụ mã hóa

IPSEC áp dụng các công cụ mã hóa như sau:

- *Mã hoá đối xứng:* DES (Data Encryption Standard) hoặc 3 DES (Triple DES).
- *Xác thực toàn vẹn dữ liệu:* các hàm băm HMAC (Hash – ased Message Authentication Code) hoặc MD5 (Message Digest 5) hoặc SHA-1 (Secure Hash Algorithm -1).
- *Chứng thực đối tác (peer Authentication):* Rivest, Shamir, and Adelman (RSA) Digital Signatures, RSA Encrypted Nonces.
- *Quản lý khóa:* DH (Diffie- Hellman), CA (Certificate Authority).

7.2.4 Kiến trúc xử lý tổng quát

Giao thức IPSEC hoạt động gồm có 2 giai đoạn xử lý cơ bản: Giai đoạn bắt tay thỏa thuận các thông số bảo mật chung (như các thuật toán mã hóa, trao đổi chìa khóa, cơ chế xử lý dữ liệu), gọi là giao thức IKE (Internet Key Exchange) và giai đoạn xử lý dữ liệu với một trong 2 giao thức là AH (Authentication Header) hoặc ESP (Encapsulation Security Payload).

Các bước hoạt động như sau:

- *Bước 1:* Nhận biết lưu lượng cần được bảo vệ để khởi tạo quá trình IPSec. Ở đây, các thiết bị IPSec sẽ nhận ra đâu là lưu lượng cần được bảo vệ, chẳng hạn thông qua trường địa chỉ.
- *Bước 2:* IKE giai đoạn 1 – IKE xác thực các đối tác IPSec và một tập các dịch vụ bảo mật được thỏa thuận và công nhận (Bộ an ninh thỏa thuận SA (Security association)). Trong giai đoạn này, thiết lập một kênh truyền thông an toàn để tiến hành thỏa thuận IPSec SA trong giai đoạn 2.
- *Bước 3:* IKE giai đoạn 2 – IKE thỏa thuận các tham số IPSec SA và thiết lập các IPSec SA tương đương ở hai phía. Những thông số an ninh này được sử dụng để bảo vệ dữ liệu và các bản tin trao đổi giữa các điểm đầu cuối. Kết quả cuối cùng của hai bước IKE là một kênh thông tin bảo mật được tạo ra giữa hai phía.
- *Bước 4:* Truyền dữ liệu – Dữ liệu được truyền giữa các đối tác IPSec dựa trên cơ sở các thông số bảo mật và các khóa được lưu trữ trong cơ sở dữ liệu SA.
- *Bước 5:* Kết thúc đường hầm IPSec – kết thúc các SA IPSec do bị xoá hoặc do hết hạn (time out).

7.2.5 Bộ thỏa thuận an ninh – SA

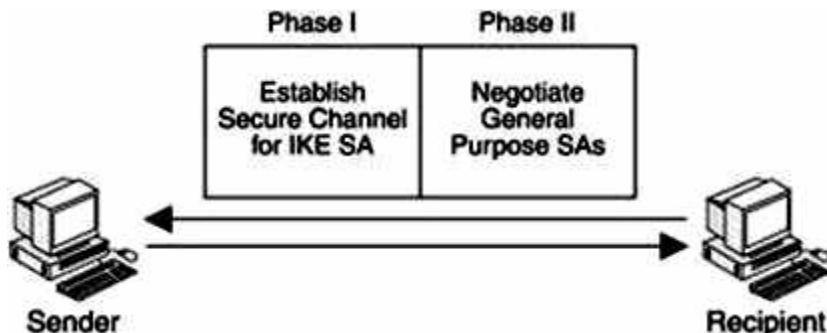
Bộ thỏa thuận an ninh (Security Associations - SA) là một khái niệm cơ bản của bộ giao thức IP Sec. SA quy định các thông số bảo mật chung của bên gửi và nhận của IPSEC gồm có:

- Các giao thức xác nhận, các khóa, và các thuật toán.

- Phương thức và các khóa cho các thuật toán xác nhận được dùng bởi các giao thức Authentication Header (AH) hay Encapsulation Security Payload (ESP) của bộ IP Sec.
- Thuật toán mã hóa và giải mã và các khóa.
- Thông tin liên quan khóa, như khoảng thời gian thay đổi hay khoảng thời gian làm tươi của các khóa.
- Thông tin liên quan đến chính bản thân SA bao gồm địa chỉ nguồn SA và khoảng thời gian làm tươi.
- Cách dùng và kích thước của bất kỳ sự đồng bộ mã hóa dùng, nếu có.
- IP Sec SA gồm có 3 trường:
 - *SPI – Security Parameter Index*: Đây là một trường 32 bit dùng nhận dạng giao thức bảo mật, được định nghĩa bởi trường Security protocol, trong bộ IPSec đang dùng. SPI được mang theo như là một phần đầu của giao thức bảo mật và thường được chọn bởi hệ thống đích trong suốt quá trình thỏa thuận của SA.
 - *Destination IP address*: Đây là địa chỉ IP của nút đích. Mặc dù nó có thể là địa chỉ broadcast, unicast, hay multicast, nhưng cơ chế quản lý hiện tại của SA chỉ được định nghĩa cho hệ thống unicast.
 - *Security protocol*: đây là giao thức bảo mật IP Sec, có thể là AH hoặc ESP.

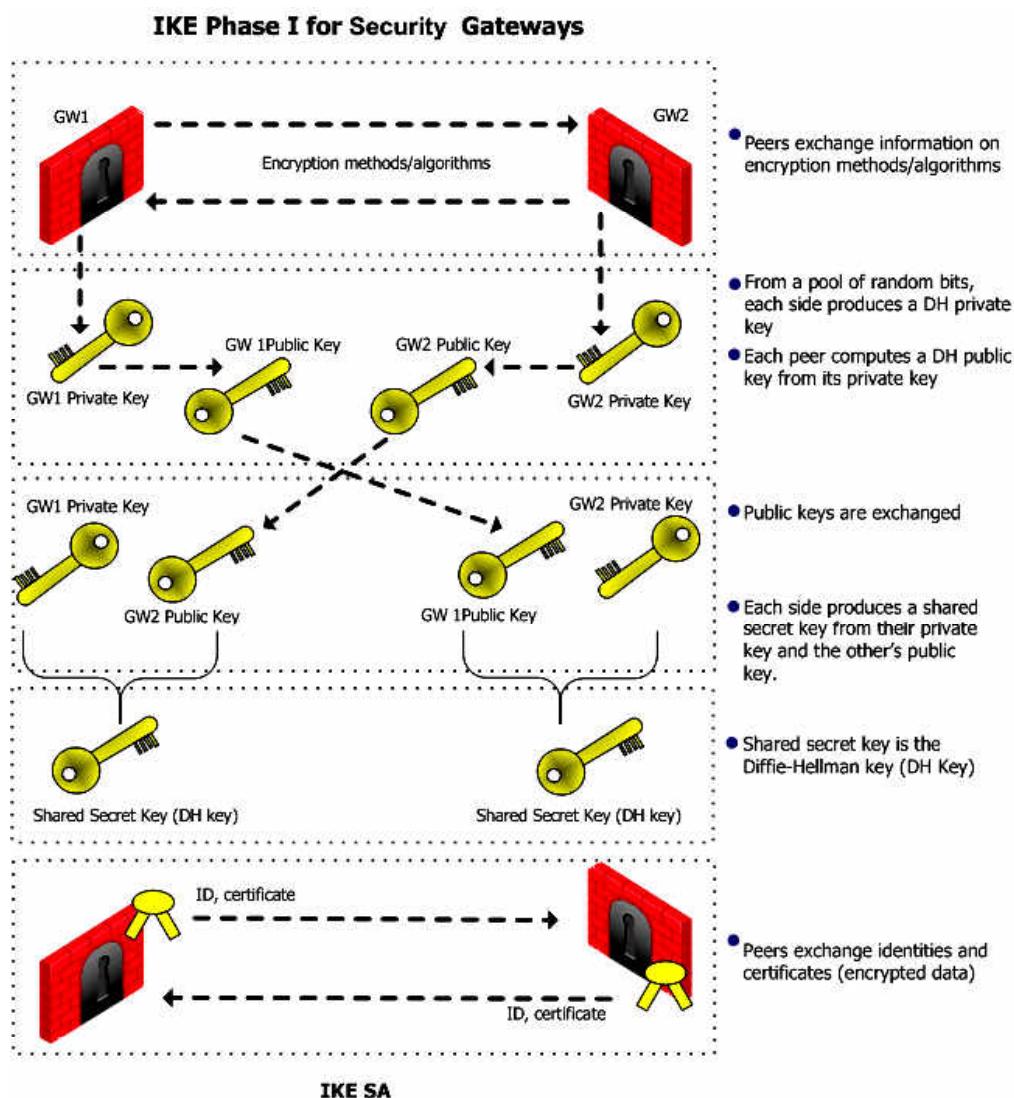
7.2.6 Trao đổi thông số bảo mật IKE

Giao thức IKE (*Internet Key Exchange*) sẽ có chức năng trao đổi key giữa các thiết bị tham gia VPN và trao đổi chính sách an ninh giữa các thiết bị. Nếu không có giao thức này thì người quản trị phải cấu hình thủ công, và những chính sách an ninh trên những thiết bị này được gọi là SA (*Security Associate*). Do đó, các thiết bị trong quá trình IKE sẽ trao đổi với nhau tất cả những SA mà nó có. Các thiết bị này sẽ tự tìm ra cho mình những SA phù hợp với đối tác nhất. Những key được trao đổi trong quá trình IKE cũng được mã hóa và những key này sẽ thay đổi theo thời gian (*Generate Key*) để tránh tình trạng BruteForce của Attacker.



Hình 7.2: Các giai đoạn xử lý của IKE

Các giai đoạn hoạt động của IKE thể hiện trong hình 7-2.



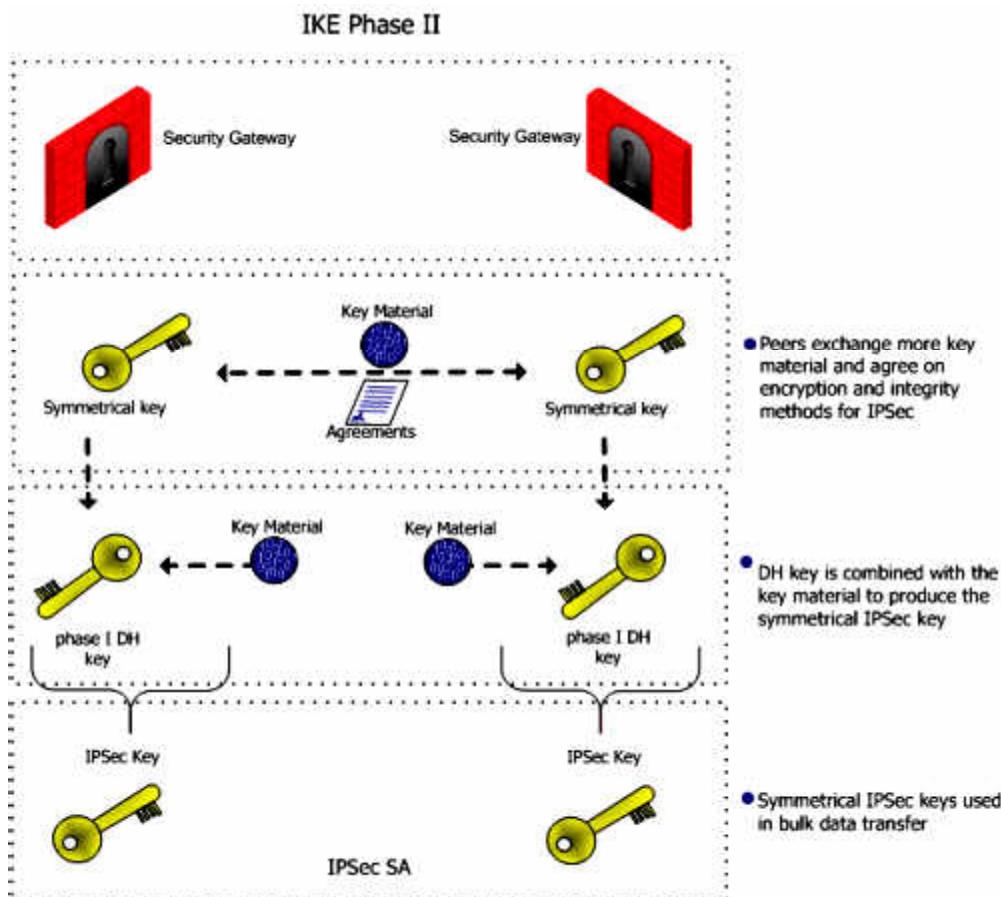
Hình 7.3: Trao đổi khóa ở giai đoạn 1

Giai đoạn 1 - Bắt buộc xảy ra trong quá trình IKE: Giai đoạn 1 xác nhận các điểm thông tin, và sau đó thiết lập một kênh bảo mật cho sự thiết lập SA. Tiếp đó, các bên

thông tin thỏa thuận một ISAKMP SA đồng ý lẫn nhau, bao gồm các thuật toán mã hóa, hàm băm, và các phương pháp xác nhận bảo vệ mã khóa.

Sau khi cơ chế mã hóa và hàm băm đã được đồng ý ở trên, một khóa chia sẻ bí mật được phát sinh. Theo sau là những thông tin được dùng để phát sinh khóa bí mật:

- Giá trị Diffie-Hellman.
- SPI của ISAKMP SA ở dạng cookies.
- Số ngẫu nhiên - known as nonces (used for signing purposes).
- Nếu hai bên đồng ý sử dụng phương pháp xác nhận dựa trên Public Key, chúng cũng cần trao đổi IDs. Sau khi trao đổi các thông tin cần thiết, cả hai bên phát sinh những key riêng của chính mình sử dụng chúng để chia sẻ bí mật. Theo cách này, những khóa mã hóa được phát sinh mà không cần thực sự trao đổi bất kỳ khóa nào thông qua mạng.



Hình 7.4: Thỏa thuận khóa ở giai đoạn 2

IKE Phases 2 – Bắt buộc xảy ra trong quá trình IKE: Trong khi giai đoạn 1 thỏa thuận thiết lập SA cho ISAKMP, giai đoạn II giải quyết bằng việc thiết lập SAs cho IP Sec. Trong giai đoạn này, SAs dùng nhiều dịch vụ khác nhau thỏa thuận. Cơ chế xác nhận, hàm băm, và thuật toán mã hóa bảo vệ gói dữ liệu IP Sec tiếp theo (sử dụng AH và ESP) dưới hình thức một phần của giai đoạn SA.

Sự thỏa thuận của giai đoạn xảy ra thường xuyên hơn giai đoạn 1. Điển hình, sự thỏa thuận có thể lặp lại sau 4-5 phút. Sự thay đổi thường xuyên các mã khóa ngắn cản các hacker bẻ gãy những khóa này và sau đó là nội dung của gói dữ liệu.

Tổng quát, một phiên làm việc ở giai đoạn 2 tương đương với một phiên làm việc đơn của giai đoạn 1. Tuy nhiên, nhiều sự thay đổi ở giai đoạn 2 cũng có thể được hỗ trợ bởi một trường hợp đơn ở giai đoạn 1. Điều này làm qua trình giao dịch chậm chạp của IKE tỏ ra tương đối nhanh hơn.

Các chế độ hoạt động của IKE:

Có 4 chế độ IKE phổ biến thường được triển khai:

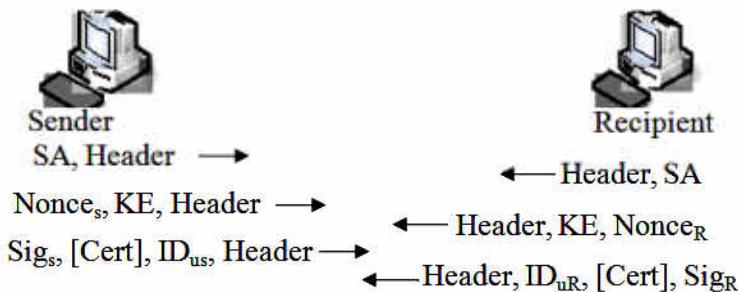
- Chế độ chính – Main mode (thuộc giai đoạn 1).
- Chế độ linh hoạt – Aggressive mode (thuộc giai đoạn 1).
- Chế độ nhanh – Quick mode (thuộc giai đoạn 2).
- Chế độ nhóm mới – New Group mode (thực hiện sau giai đoạn 1, nhưng nó không thuộc giai đoạn 2).

Ngoài 4 chế độ IKE phổ biến trên, còn có thêm Informational mode. Chế độ này kết hợp với quá trình thay đổi của giai đoạn 2 và SAs. Chế độ này cung cấp cho các bên có liên quan một số thông tin thêm, xuất phát từ những thất bại trong quá trình thỏa thuận. Ví dụ, nếu việc giải mã thất bại tại người nhận hoặc chữ ký không được xác minh thành công, Informational mode được dùng để thông báo cho các bên khác biết.

Main Mode: Main mode xác nhận và bảo vệ tính đồng nhất của các bên có liên quan trong quá trình giao dịch. Trong chế độ này, 6 thông điệp được trao đổi giữa các điểm:

- Hai thông điệp đầu tiên dùng để thỏa thuận chính sách bảo mật cho sự thay đổi.

- Hai thông điệp kế tiếp phục vụ để thay đổi các khóa Diffie-Hellman và nonces. Những khóa sau này thực hiện một vai trò quan trọng trong cơ chế mã hóa.
- Hai thông điệp cuối cùng của chế độ này dùng để xác nhận các bên giao dịch với sự giúp đỡ của chữ ký, các hàm băm, và tùy chọn với chứng nhận.

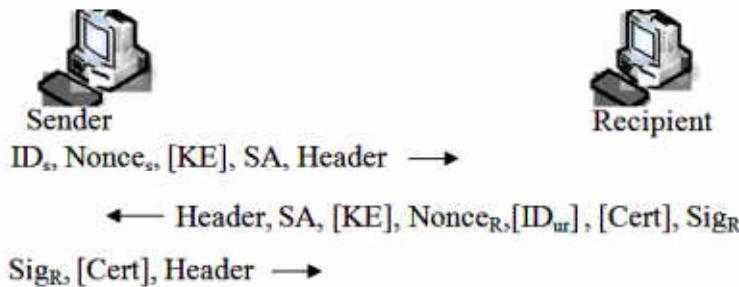


Hình 7.5: IKE main mode

- Header: Một tiêu đề ISAKMP tương ứng với chế độ được dùng.
- SA: Kết hợp bảo mật được thương lượng.
- Nonce: Một số ngẫu nhiên gửi cho việc ký số.
- KE: Dữ liệu trao đổi khóa với trao đổi khóa Diffie-Hellman.
- Sig: Chữ ký dùng cho xác thực.
- Cert: Một chứng chỉ số cho khóa công khai.
- ID: Định danh (ss là của Sender, sr là của Recipient trong pha 1).

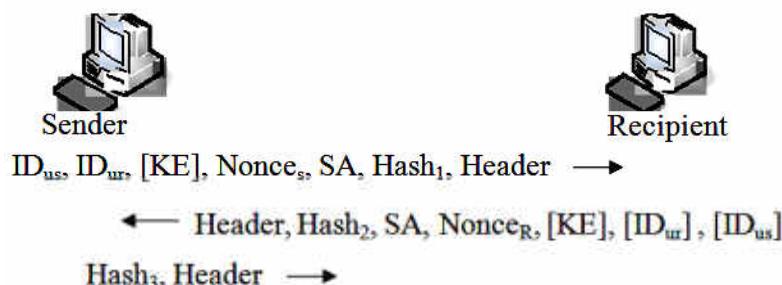
Aggressive Mode: Aggressive mode về bản chất giống Main mode. Chỉ khác nhau thay vì main mode có 6 thông điệp thì chế độ này chỉ có 3 thông điệp được trao đổi. Do đó, Aggressive mode nhanh hơn main mode. Các thông điệp đó bao gồm:

- Thông điệp đầu tiên dùng để đưa ra chính sách bảo mật, pass data cho khóa chính, và trao đổi nonces cho việc ký và xác minh tiếp theo.
- Thông điệp kế tiếp hồi đáp lại cho thông tin đầu tiên. Nó xác thực người nhận và hoàn thành chính sách bảo mật bằng các khóa.
- Thông điệp cuối cùng để xác nhận người gửi (hoặc bộ khởi tạo của phiên làm việc).

**Hình 7.6: IKE – Aggressive Mode**

- Header: Một tiêu đề ISAKMP tương ứng với chế độ được dùng.
- SA: Kết hợp bảo mật được thương lượng.
- Nonce: Một số ngẫu nhiên gửi cho việc ký số.
- KE: Khóa trao đổi dữ liệu cho trao đổi khóa Differ-Helman.
- Sig: Chữ ký dùng cho xác thực.
- Cert: Một chứng chỉ số cho khóa công khai.
- ID: Định danh (ss là của Sender, sr là của Recipient trong pha 2).

Quick Mode: Chế độ thứ ba của IKE, Quick mode, là chế độ trong giai đoạn II. Nó dùng để thỏa thuận SA cho các dịch vụ bảo mật IP Sec. Ngoài ra, Quick mode cũng có thể phát sinh khóa chính mới. Nếu chính sách của Perfect Forward Secrecy (PFS) được thỏa thuận trong giai đoạn I, một sự thay đổi hoàn toàn Diffie-Hellman key được khởi tạo. Mặt khác, khóa mới được phát sinh bằng các giá trị băm.

**Hình 7.7: IKE – Quick Mode**

- Header: Một tiêu đề ISAKMP tương ứng với chế độ được dùng.
- SA: Kết hợp bảo mật được thương lượng.
- Nonce: Một số ngẫu nhiên gửi cho việc ký số.
- KE: Khóa trao đổi dữ liệu cho trao đổi khóa Differ-Helman.

- Hash: Giá trị hàm băm của dữ liệu được tải.
- ID: Định danh (ss là của Sender, sr là của Recipient trong pha 2).

New Group Mode: New Group mode được dùng để thỏa thuận một private group mới nhằm tạo điều kiện trao đổi Diffie-Hellman key được dễ dàng. Hình 6-18 mô tả New Group mode. Mặc dù chế độ này được thực hiện sau giai đoạn 1, nhưng nó không thuộc giai đoạn 2.



Hình 7.8: IKE-New Group Mode

- Header: Một tiêu đề ISAKMP tương ứng với chế độ được dùng.
- SA: Kết hợp bảo mật được thương lượng.
- Hash: Giá trị hàm băm của dữ liệu được tải.

7.2.7 Giao thức xử lý dữ liệu AH

Giao thức xác thực AH thêm một tiêu đề vào gói IP. Tiêu đề này phục vụ cho việc xác thực gói dữ liệu IP gốc tại người nhận cuối cùng, tiêu đề này giúp nhận biết bất kỳ sự thay đổi nào về nội dung của gói dữ liệu bởi người dùng không mong muốn trong khi đang truyền. Giao thức AH có các đặc trưng cơ bản như sau:

- Cung cấp tính toàn vẹn dữ liệu và bảo vệ chống phát lại.
- Sử dụng mã xác thực thông điệp được băm(HMAC), dựa trên chia sẻ bí mật.
- Nội dung các gói tin không được mã hoá.
- Không sử dụng các trường changeable IP header để tính toán giá trị kiểm tra tính toàn vẹn (IVC).

Khuôn dạng gói tin gồm các trường:

Next Header: Trường này nhận biết giao thức bảo mật, có độ dài 8 bít để xác định kiểu dữ liệu của phần Payload phía sau AH. Giá trị của trường này được chọn từ các

giá trị của IP Protocol number được định nghĩa bởi IANA (*Internet Assigned Numbers Authority*).

Payload Length: Trường này chỉ định độ dài của thông điệp gắn sau tiêu đề AH.

Reserved: Trường 16 bit dự trữ để sử dụng cho tương lai, và có giá trị bằng 0.

SPI: Là một số 32 bit bất kỳ, cùng với địa chỉ IP đích và giao thức an ninh mạng cho phép nhận dạng một thiết lập an toàn duy nhất cho gói dữ liệu. SPI thường được lựa chọn bởi phía thu.

Sequence Number (SN): gồm 32 bit không dấu đếm tăng dần để sử dụng cho việc chống trùng lặp. Chống trùng lặp là một lựa chọn nhưng trường này là bắt buộc đối với phía phát. Bộ đếm của phía phát và thu khởi tạo 0 khi một liên kết an toàn (SA) được thiết lập, giá trị SN mỗi gói trong một SA phải hoàn toàn khác nhau để tránh trùng lặp. Nếu số gói vượt quá con số 232 thì một SA khác phải được thiết lập.

Authentication Data: Trường có độ dài biến đổi chứa một giá trị kiểm tra tính toàn vẹn (ICV) cho gói tin, ICV được tính bằng thuật toán đã được chọn khi thiết lập SA. Độ dài của trường này là số nguyên lần của 32 bit, chứa một phần dữ liệu đệm để đảm bảo độ dài của AH là $n * 32$ bit. Giao thức AH sử dụng một hàm băm và băm toàn bộ gói tin trừ trường Authentication Data để tính ICV.

7.2.8 Giao thức xử lý dữ liệu ESP

Mục đích chính của ESP là cung cấp dịch vụ bảo mật và xác minh tính toàn vẹn của dữ liệu trong khi truyền. ESP mã hoá nội dung của gói dữ liệu bằng cách dùng các thuật toán mã hoá, như đã xác định bởi SA. Một số thuật toán được sử dụng bởi ESP bao gồm: DES-CBG, NULL, CAST-128, IDEA và 3DES. Các thuật toán xác thực thường được dùng tương tự như trong AH là HMAC-MD5 và HMAC-SHA.

Khuôn dạng gói tin ESP gồm ESP header mà còn ESP trailer và ESP Authentication data. Dữ liệu tải (*Payload*) được định vị giữa header và trailer. Các trường trong ESP đều là bắt buộc:

SPI: Là số 32 bit bất kỳ, cùng với địa chỉ IP đích và giao thức an ninh, ESP cho phép nhận dạng duy nhất SA cho gói dữ liệu này. Giá trị SPI từ 1 đến 255 được dành riêng để sử dụng trong tương lai. Giá trị SPI = 0 để chỉ ra chưa có SA nào tồn tại.

SN: Giống như AH, trường SN chứa một giá trị đếm tăng dần để chống lặp lại. Mỗi SA được lựa chọn thì giá trị của trường này bắt đầu là 0.

Payload Data: Trường có độ dài biến đổi chứa dữ liệu mô tả trong Next Header. Payload Data là trường bắt buộc, được mã hoá bởi các thuật toán mã hoá, các thuật toán mã hoá này được lựa chọn ngay khi thiết lập SA. Trường này có độ dài bằng một số nguyên lần 1 byte.

Padding: Trường này được thêm vào để đoạn được mã hoá là một số nguyên lần của một khối các byte. Ngoài ra trường còn dùng để che dấu độ dài thực của Payload.

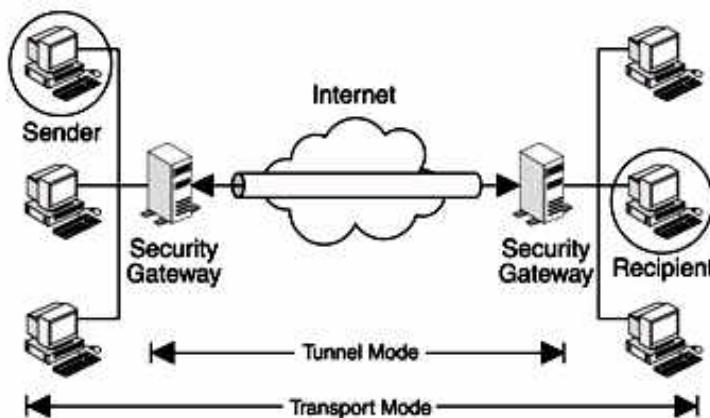
Pad Length: Trường này xác định số byte padding đã thêm vào (0 đến 225)

Next Header: Là trường 8 bit bắt buộc, nó chỉ ra kiểu dữ liệu trong Payload Data. Ví dụ một giao thức bậc cao hơn như TCP. Giá trị của trường được chọn trong chuẩn IP Protocol Number được đưa ra bởi IANA.

Authentication Data: Trường có độ dài biến đổi chứa giá trị ICV được tính cho gói ESP từ SPI đến Next Header. Authentication là trường không bắt buộc, được thêm vào nếu dịch vụ Authentication được lựa chọn cho SA đang xét. Các thuật toán để tính ICV là các thuật toán hàm băm một chiều MD5 hoặc SHA giống với AH.

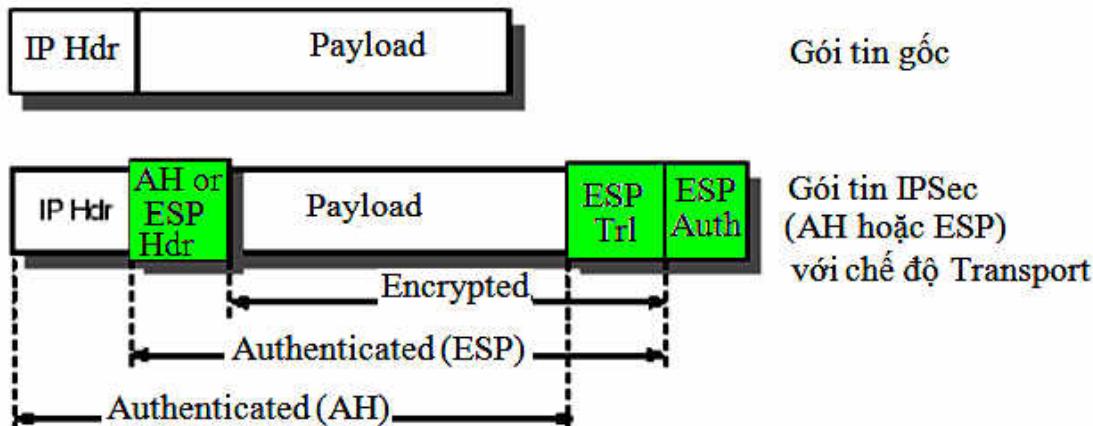
7.2.9 Các chế độ IPSec

Các giao thức của IPSec có thể thực hiện các liên kết an toàn trong hai chế độ: Transport và Tunnel. Như được mô tả trong hình 7-9, cả AH và ESP đều có thể hoạt động cả trong hai chế độ.



Hình 7.9: Chế độ hoạt động của IPSEC

Chế độ Transport: Chế độ Transport bảo vệ các giao thức tầng trên và các ứng dụng. Trong chế độ này, tiêu đề IPSec được chèn vào giữa tiêu đề IP và tiêu đề của giao thức tầng trên.

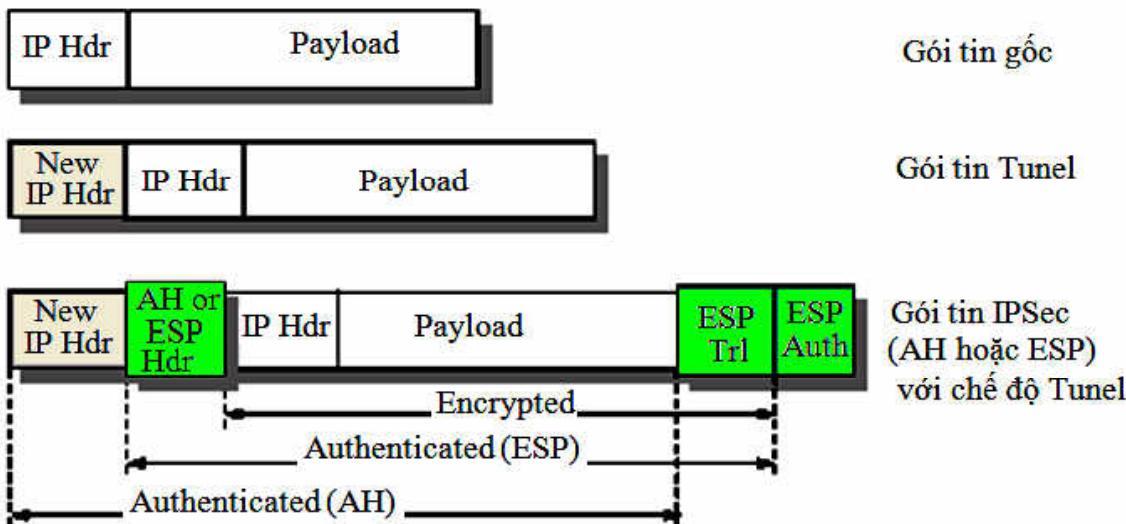


Hình 7.10: IPSEC- chế độ Transport

Trong giao thức ESP, ESP trailer và dữ liệu xác thực ESP được thêm vào sau phần dữ liệu gốc được tải. Tiêu đề mới được chèn vào trước phần dữ liệu được tải. Chế độ Transport được dùng bởi các host, không được dùng bởi các gateway. Các Gateway thậm chí không yêu cầu hỗ trợ chế độ Transport.

Ưu điểm của chế độ Transport là xử lý ít Overhead, vì vậy nó nhanh hơn. Một nhược điểm của nó là các trường hay thay đổi không được xác thực. ESP trong chế độ Transport không cung cấp tính năng xác thực và mã hóa với các tiêu đề IP. Đây là một nhược điểm vì các gói tin sai (tấn công giả mạo) có thể vẫn được phân phối cho quá trình xử lý ESP. Một nhược điểm khác là địa chỉ gói IP gốc phải được dùng để phân phối. Điều này có thể là một vấn đề nơi địa chỉ IP riêng được dùng, hoặc nơi cấu trúc địa chỉ mạng bên trong cần được dấu trong mạng công cộng.

Chế độ Tunnel: Không giống như chế độ Transport, chế độ Tunnel bảo vệ toàn bộ gói IP. Toàn bộ gói IP được đóng gói vào trong một gói IP khác và tiêu đề IPSec được chèn vào giữa tiêu đề IP gốc và tiêu đề IP mới. Trong chế độ này, khái niệm đường hầm được áp dụng.



Hình 7.11: IPSEC – Chế độ tunel

Với giao thức ESP thì gói dữ liệu gốc trở thành gói dữ liệu tải cho gói ESP mới và kết quả là cả mã hóa cũng như xác thực được thực thi nếu được chọn. Tuy nhiên, tiêu đề IP mới vẫn không được bảo vệ.

Chế độ Tunnel được sử dụng bởi các Getway. Như vậy, giữa 2 firewall chế độ Tunnel luôn được dùng cho luồng thông tin đang lưu chuyển qua các firewall giữa các mạng an toàn qua một đường hầm IPsec.

Mặc dù các Getway được hỗ trợ chỉ với chế độ Tunnel, thông thường chúng vẫn có thể làm việc được trong chế độ Transport. Chế độ này được cho phép khi Getway hoạt động như một Host, đó là trường hợp luồng thông tin được giành riêng cho chính nó. Ví dụ: các lệnh SNMP, hoặc các yêu cầu báo lỗi ICMP.

Trong chế độ Tunnel các địa chỉ IP của tiêu đề ngoài không cần phải giống với các địa chỉ của tiêu đề bên trong. Ví dụ, hai getway bảo mật có thể thực hiện một đường hầm AH có xác thực tất cả các luồng thông tin giữa các mạng chúng kết nối cùng nhau. Ưu điểm của chế độ Tunnel là nó bảo vệ tất cả gói IP đã được đóng gói và khả năng sử dụng các địa chỉ riêng. Tuy nhiên, có một quá trình xử lý Overhead nhiều hơn bình thường gắn liền với chế độ này.

TÓM TẮT

Bài học cung cấp kiến thức về giao thức ứng dụng bảo mật trong mạng nội bộ Kerberos và giao thức an toàn IP - IPSEC.

Kerberos là một giao thức xác thực mạng, nó cho phép các cá nhân giao tiếp với nhau trên một mạng không an toàn bằng cách xác thực người dùng theo một cơ chế bảo mật và an toàn. Kerberos ngăn chặn việc nghe trộm thông tin cũng như tấn công thay thế và đảm bảo tính toàn vẹn của dữ liệu. Kerberos hoạt động theo mô hình máy trạm/máy chủ và nó thực hiện quá trình xác thực 2 chiều - cả người dùng và dịch vụ xác thực lẫn nhau. Kerberos được xây dựng dựa trên mô hình mã hóa khóa đối xứng và đòi hỏi một thành phần thứ ba tin cậy tham gia vào quá trình xác thực.

Tuy nhiên, nó không hỗ trợ chữ ký điện tử (yêu cầu đáp ứng cả hai nhu cầu xác thực và bảo đảm không chối cãi được). Một trong những giả thiết quan trọng của giao thức Kerberos là các máy chủ trên mạng cần phải tin cậy được. Ngoài ra, nếu người dùng chọn những mật khẩu dễ đoán thì hệ thống dễ bị mất an toàn trước kiểu tấn công từ điển.

IPSec (Internet Protocol Security) thực hiện mã hóa và xác thực ở lớp mạng. Nó cung cấp một giải pháp an toàn dữ liệu đầu cuối cũng như liên kết mạng. Các gói mã hóa có khuôn dạng giống như gói tin IP thông thường, nên chúng dễ dàng được định tuyến qua mạng Internet mà không phải thay đổi các thiết bị mạng trung gian, qua đó cho phép giảm đáng kể các chi phí cho việc triển khai và quản trị. IPSec cung cấp bốn chức năng quan trọng sau: Bảo mật(mã hóa)- Confidentiality, Toàn vẹn dữ liệu- Data integrity, Xác thực- Authentication, và Antireplay protection (xác nhận mỗi gói tin là duy nhất và không trùng lặp).

IPSEC áp dụng các công cụ mã hóa như sau:Mã hoá đối xứng: DES (Data Encryption Standard) hoặc 3 DES (Triple DES), Xác thực toàn vẹn dữ liệu: các hàm băm HMAC (Hash – ased Message Authentication Code) hoặc MD5 (Message Digest 5) hoặc SHA-1 (Secure Hash Algorithm -1), Chứng thực đối tác (peer Authentication): Rivest, Shamir, and Adelman (RSA) Digital Signatures, RSA Encrypted Nonces, và Quản lý khóa: DH (Diffie- Hellman), CA (Certificate Authority).

CÂU HỎI ÔN TẬP

Câu 1: Trình bày nhu cầu sử dụng Kerberos?

Câu 2: Cho biết các dịch vụ bảo mật Kerberos hỗ trợ?

Câu 3: Cho biết Kerberos có sử dụng những khóa gì? Giải thích tính bảo mật của các khóa này trong quá trình trao đổi và ý nghĩa sử dụng của nó?

Câu 4: Trình bày các nhược điểm của Kerberos?

Câu 5: Trình bày nhu cầu sử dụng IPSEC?

Câu 6: Trình bày các ưu điểm của IPSEC?

Câu 7: Giải thích ý nghĩa bảo mật của sơ đồ tổng quan các bước xử lý của IPSEC?

Câu 8: IKE là gì? Trình bày chi tiết quá trình hoạt động của IKE?

Câu 9: AH là gì? Ý nghĩa sử dụng? Trình bày cấu trúc gói tin và quá trình xử lý theo AH?

Câu 10: ESP là gì? Ý nghĩa sử dụng? Trình bày cấu trúc gói tin và quá trình xử lý của ESP?

BÀI 8: BẢO MẬT WEB VÀ MAIL

Sau khi học xong bài này, sinh viên hiểu:

- *Chứng minh thư điện tử theo chuẩn X.509.*
- *Phần mềm bảo mật web SSL. Tính bí mật của quá trình xác thực và trao đổi khóa. Ý nghĩa của chứng minh thư điện tử trong việc xác thực của SSL.*
- *Giải pháp bảo mật mail PGP, ý nghĩa của hệ mã khóa công khai trong quá trình xác thực và sự kết hợp giữa hệ mã khóa công khai và khóa bí mật trong việc bảo mật và xác thực mail.*

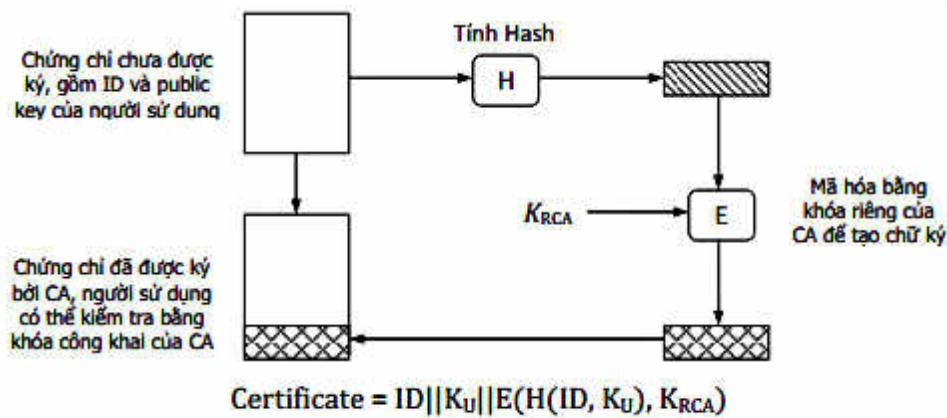
8.1 DỊCH VỤ XÁC THỰC X.509

Dịch vụ xác thực X.509 là một phần của chuẩn dịch vụ thư mục CCITT X.500. Ở đây các máy chủ phân tán bảo trì cơ sở dữ liệu thông tin của người sử dụng và xác định khung cho các dịch vụ xác thực. Thư mục chứa các chứng nhận khóa công khai, khóa công khai của người sử dụng được ký bởi chủ quyền chứng nhận. Để thống nhất dịch vụ cũng xác định các thủ tục xác thực, sử dụng mã khóa công khai và chữ ký điện tử. Tuy thuật toán không chuẩn nhưng được RSA đề xuất. Các chứng nhận X.509 được sử dụng rộng rãi.

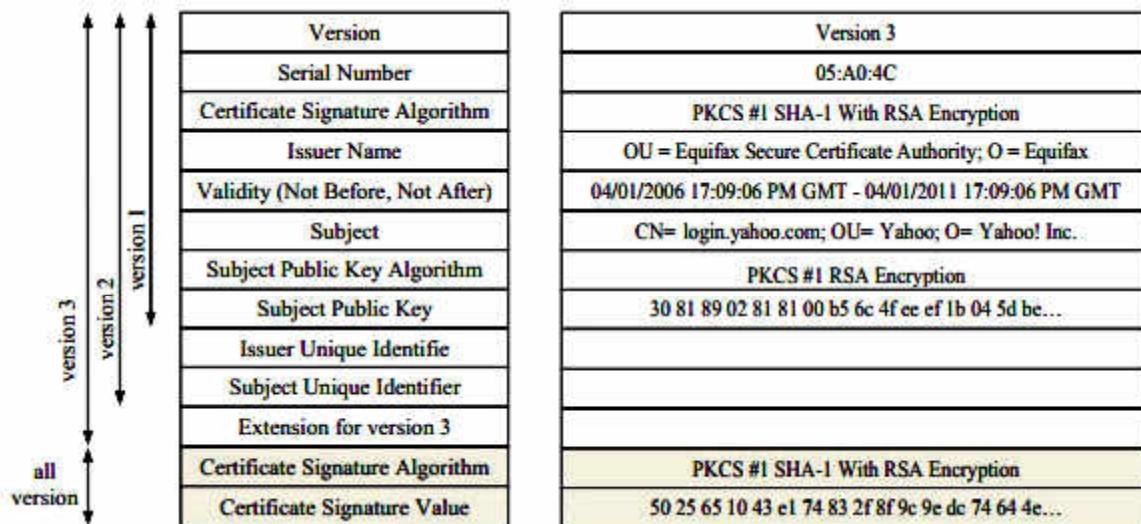
Sơ đồ nguyên tắc để sinh ra chứng thực X.509 được trình bày trong hình 8-1. Cấu trúc một chứng chỉ X.509 gồm có các thành phần được trình bày trong hình 8-2.

Mục đích của các thành phần trên là:

- *Version: phiên bản X.509 của chứng chỉ này, có 3 phiên bản là 1, 2 và 3.*
- *Serial Number: số serial của chứng chỉ này do trung tâm chứng thực CA ban hành.*



Hình 8.1: Sơ đồ tạo chứng chỉ X.509

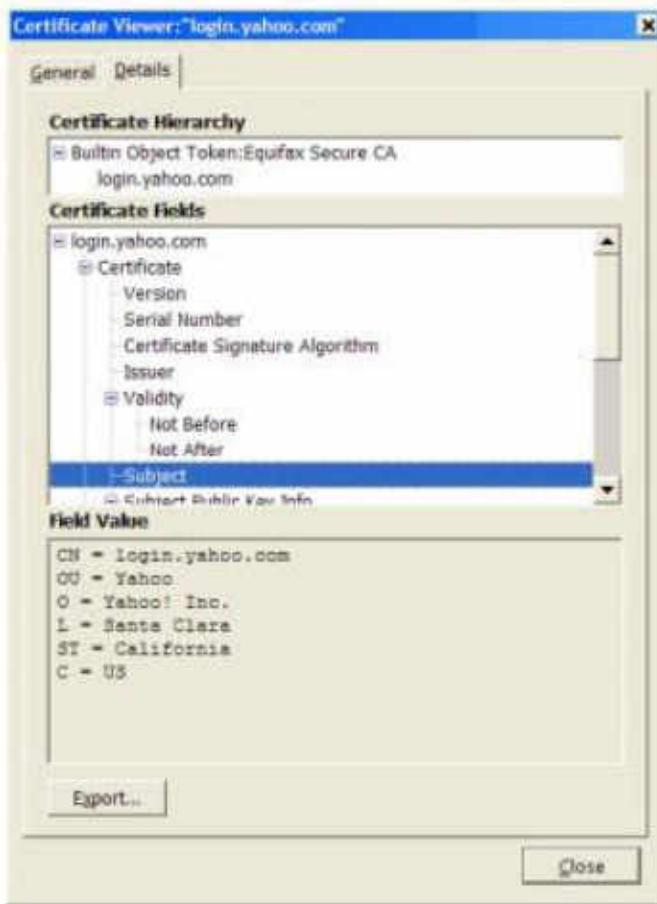


Hình 8.2: Cấu trúc và ví dụ một chứng chỉ X.509

- *Certificate Signature Algorithm*: thuật toán ký chứng chỉ, gồm loại hàm Hash và phương pháp mã hóa khóa công khai.
- *Issuer name*: Tên của trung tâm chứng thực CA (CN: common name, O: organization, OU: organization unit).
- *Validity*: thời gian hiệu lực của chứng chỉ.
- *Subject*: tên chủ sở hữu chứng chỉ, cũng gồm có CN, O, OU.
- *Subject Public Key Algorithm*: thuật toán mã hóa khóa công khai mà tương ứng với khóa công khai trong chứng chỉ.
- *Subject Public Key*: khóa công khai trong chứng chỉ, tức khóa công khai của chủ sở hữu. Đối với RSA thì thuộc tính này lưu giữ giá trị Modulus và Exponent nối tiếp nhau (N và e).

- Issuer Unique Identifier, Subject Unique Identifier: dành cho version 2, ít được sử dụng.
- *Extension*: dành cho version 3.
- *Certificate Signature Algorithm*: thuật toán ký chứng chỉ, giống mục thứ 3.
- *Certificate Signature Value*: giá trị của chữ ký.
- Đối với version 3 phần Extension có thể gồm các thông tin sau:
 - *Authority key identifier*: Một con số dùng để định danh trung tâm chứng thực. Thuộc tính Issuer Name cung cấp tên trung tâm chứng thực dưới dạng text, điều này có thể gây nhầm lẫn.
 - *Subject key identifier*: Một con số dùng để định danh người sử dụng được chứng thực. Tương tự như Issuer Name, thuộc tính Subject cũng cung cấp tên người dưới dạng text, điều này có thể gây nhầm lẫn. Ngoài ra việc dùng một con số định danh cho phép một người sử dụng có thể có nhiều chứng chỉ khác nhau.
- *Key Usage*: mục đích sử dụng của chứng chỉ. Mỗi chứng chỉ có thể có một hoặc nhiều mục đích sử dụng như: mã hóa dữ liệu, mã hóa khóa, chữ ký điện tử, không thoái thác ...
- *CRL Distribution Point*: địa chỉ để lấy danh sách các chứng chỉ đã hết hạn hay bị thu hồi (certificate revocation list). Một chứng chỉ thường được lưu trên một file có phần mở rộng là.cer.

Vì chứng chỉ được ký bằng khóa riêng của CA, nên bảo đảm rằng chữ ký không thể bị làm giả và bất cứ ai tin tưởng vào khóa công khai của CA thì có thể tin tưởng vào chứng chỉ mà CA đó cấp phát. Do đó khóa công khai của CA phải được cung cấp một cách tuyệt đối an toàn đến tay người sử dụng. Trong ví dụ trên chứng thực của Yahoo được cung cấp bởi Equifax Secure. FireFox tin tưởng vào Equifax và khóa công khai của Equifax được tích hợp sẵn trong bộ cài đặt của FireFox. Vì vậy khi duyệt đến trang web của Yahoo, FireFox có được chứng chỉ của Yahoo, vì FireFox tin tưởng vào Equifax nên cũng sẽ tin tưởng vào Yahoo và cho phép người sử dụng duyệt trang web này (xem thêm phần giao thức SSL bên dưới).



Hình 8.3: Nội dung một chứng thực dùng trong giao thức SSL

Trên thế giới hiện nay có nhiều tổ chức cung cấp chứng thực X509 như VeriSign, Equifax, Thawte, SecureNet... VeriSign hiện là tổ chức lớn nhất. Verisign cung cấp chứng chỉ X509 theo ba mức độ (class):

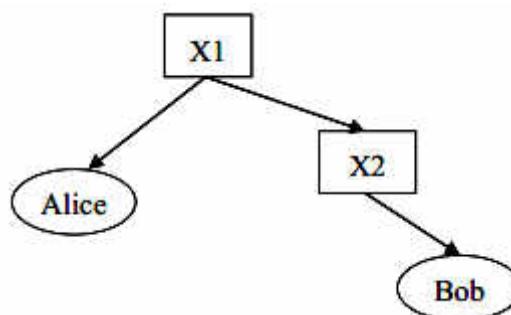
- *Class 1:* ID của một đối tượng là email của đối tượng đó. Sau khi đối tượng đăng ký email và public key qua mạng Internet, Verisign gửi email để kiểm tra địa chỉ email hợp lệ và cấp chứng thực.
- *Class 2:* ID là địa chỉ nơi ở của đối tượng, Verisign sẽ gửi confirm qua đường bưu điện để kiểm tra địa chỉ hợp lệ.
- *Class 3:* đối tượng cần có giấy tờ pháp lý để chứng minh tư cách pháp nhân.

8.1.1 Phân Cấp Chứng Thực

Trên thế giới không thể chỉ có một trung tâm chứng thực CA duy nhất mà có thể có nhiều trung tâm chứng thực. Những người sử dụng khác nhau có thể đăng ký

chứng thực tại các CA khác nhau. Do đó để có thể trao đổi dữ liệu, một người cần phải tin tưởng vào khóa công khai của tất cả các trung tâm chứng thực. Để giảm bớt gánh nặng này, X.509 đề ra cơ chế phân cấp chứng thực.

Ví dụ, Alice chỉ tin tưởng vào trung tâm chứng thực X1, còn chứng thực của Bob là do trung tâm chứng thực X2 cung cấp. Nếu Alice không có khóa công khai của X2, thì làm sao Alice có thể kiểm tra được chứng thực của Bob? Biện pháp giải quyết là Alice có thể đọc Authority key identifier (tức ID của X2) trong chứng thực của Bob. Sau đó Alice kiểm tra xem X1 có cấp chứng thực nào cho X2 hay không. Nếu có, Alice có thể tìm thấy được khóa công khai của X2 và tin tưởng vào khóa này (do đã được X1 xác nhận). Từ đó Alice có thể kiểm tra tính xác thực của chứng chỉ của Bob.



Việc phân cấp chứng thực này không chỉ giới hạn trong hai trung tâm chứng thực mà có thể thông qua một dãy các trung tâm chứng thực tạo thành một mạng lưới chứng thực (Web of Trust). Hình dưới minh họa một ví dụ thực tế.

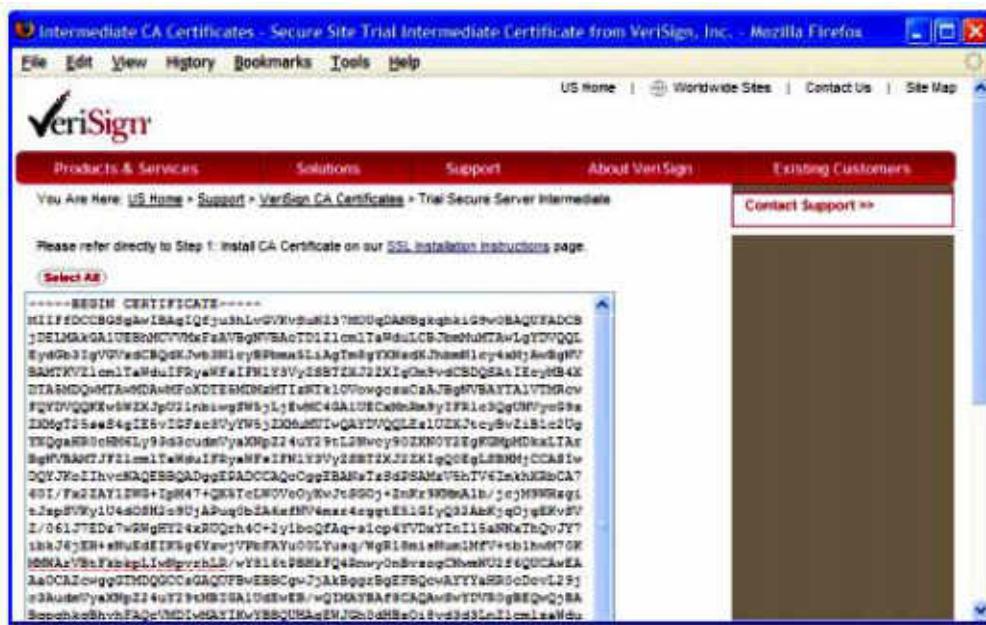


Hình 8.4: Minh họa mô hình phân cấp chứng thực

Trong ví dụ trên chứng thực Atheros Communication Inc. được chứng thực bởi "Verisign Class 3 Code Signing 2009-2 CA", IE không có sẵn khóa công khai của trung tâm này. Tuy nhiên, IE có khóa công khai của "Verisign Class 3 Public Primary CA", từ

đó FireFox có thể chứng thực trung tâm “Verisign Class 3 Public Primary CA” và qua đó có thể chứng thực được “Verisign Class 3 Code Signing 2009-2 CA”.

8.1.2 Các Định Dạng File Của Chứng Chỉ X.509



Hình 8.5: chứng chỉ của Verisign được cung cấp dưới dạng PEM

Dạng DER (.cer): nội dung của chứng chỉ X.509 được lưu dưới format DER, một định dạng dữ liệu binary chuẩn cho các môi trường máy tính.

Dạng PEM (.pem): là dạng DER và được mã hóa dưới dạng text theo chuẩn Base64. Một file text PEM bắt đầu bằng dòng -----BEGIN CERTIFICATE----- và kết thúc bằng dòng -----END CERTIFICATE-----

Dạng PKCS#7 (.p7c hay .p7b): là một định dạng dữ liệu được mã hóa hay ký. Do đó có đi kèm cả chứng chỉ.

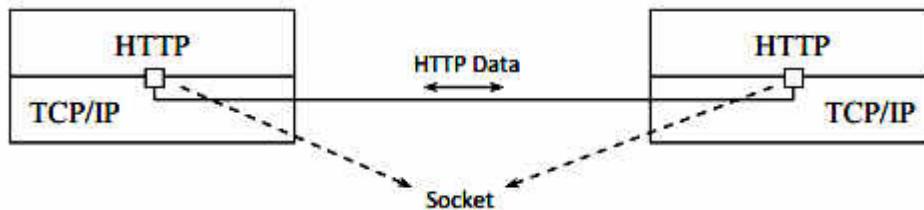
Dạng PKCS#10 (.p10 hay p10): là một định dạng dùng để gửi yêu cầu cấp chứng chỉ X509 đến trung tâm chứng thực. Định dạng này có ID và public key của người yêu cầu.

Dạng PKCS#12 (.p12): lưu trữ chứng chỉ X509 và private key tương ứng (có password bảo vệ) trong cùng 1 file.

Dạng PFX (.pfx): cũng lưu chứng chỉ X509 và private key theo định dạng của Microsoft.

8.2 GIAO THỨC BẢO MẬT WEB SSL

Dữ liệu Web được trao đổi giữa trình duyệt và web server được thực hiện qua giao thức HTTP. Client kết nối với server qua socket của giao thức TCP/IP.



```
GET /search?p=Nha+Trang&fcss=on&fr=yfp-t-101&toggle=1&cop=&ei=UTF-8 HTTP/1.1
Host: vn.search.yahoo.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.0.13) Gecko/2009073022
Firefox/3.0.13 (.NET CLR 3.5.30729)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://vn.yahoo.com/?p=us
```

```
HTTP/1.1 200 OK
Date: Fri, 14 Aug 2009 10:25:49 GMT
Cache-Control: private
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Connection: Keep-Alive
Content-Encoding: gzip

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html lang="vi"><head> ... </head>
...
</html>
```

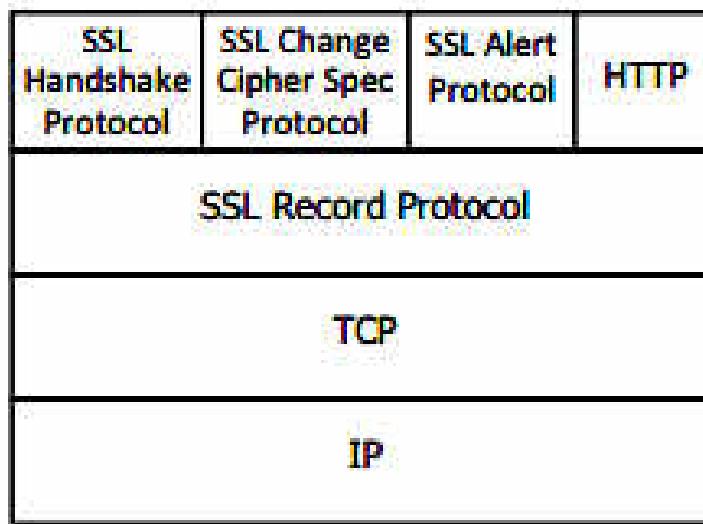
Hình 8.6: Gói tin HTTP Request và response chứa văn bản rõ

Giao thức SSL (Secure Socket Layer) bảo mật dữ liệu trao đổi qua socket. Đây là giao thức bảo mật kết hợp mã hóa khóa công khai và khóa đối xứng trong đó mã hóa RSA được dùng để trao đổi khóa phiên của mã hóa đối xứng.



Hình 8.7: Minh họa SSL hỗ trợ cho Http (Https)

8.2.1 Kiến trúc tổng thể



Hình 8.8: Kiến trúc các giao thức SSL

Kiến trúc xử lý của SSL gồm có bốn công đoạn:

- SSL Record Protocol: Xử lý dữ liệu.
- SSL Change Cipher Spec: một message đơn 1 byte, cập nhật lại bộ mã hóa để sử dụng trên kết nối này.
- SSL Alert: được dùng để truyền cảnh báo liên kết SSL với đầu cuối bên kia.
- SSL Handshake Protocol: Giao thức này cho phép server và client chứng thực với nhau và thương lượng cơ chế mã hóa, thuật toán MAC và khóa mật mã được sử dụng để bảo vệ dữ liệu được gửi trong SSL record.

8.2.2 Thủ tục thay đổi đặc tả mã SSL (SSL Change Cipher Spec Protocol)

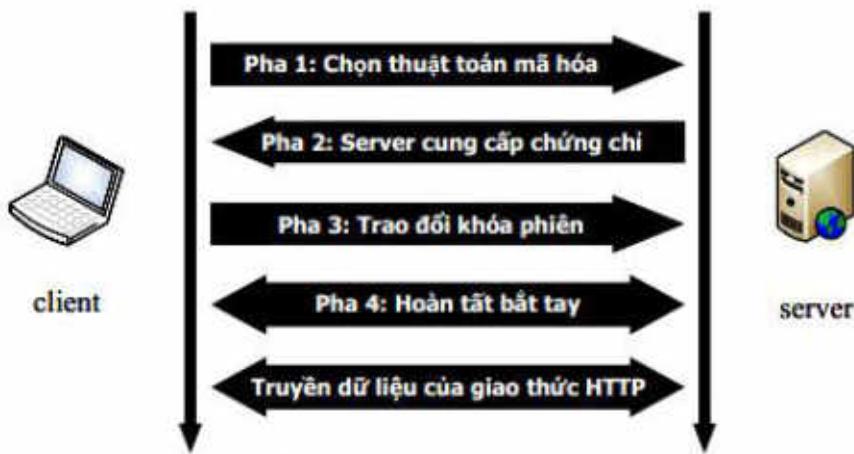
Đây là một trong 3 giao thức chuyên biệt của SSL sử dụng thủ tục bản ghi SSL. Đây là mẫu tin đơn, buộc trạng thái treo trở thành hiện thời và cập nhật bộ mã đang dùng.

8.2.3 Thủ tục nhắc nhở SSL (SSL Alert Protocol)

Truyền đi lời nhắc của SSL liên quan cho thành viên. Nghiêm khắc: nhắc nhở hoặc cảnh báo. Nhắc nhở đặc biệt:

- *Cảnh báo*: mẫu tin không chờ đợi, bản ghi MAC tồi, lỗi giải nén, lỗi Handshake, tham số không hợp lệ.
- *Nhắc nhở*: đóng ghi chú, không chứng nhận, chứng nhận tồi, chứng nhận không được hỗ trợ, chứng nhận bị thu hồi, chứng nhận quá hạn, chứng nhận không được biết đến.

8.2.4 Giao Thức Bắt Tay



Hình 8.9: Minh họa hoạt động của SSL

Trước khi tiến hành truyền số liệu, SSL thực hiện giao thức bắt tay (*SSL Handshaking Protocol*) để chứng thực website và chứng thực người duyệt web, trao đổi khóa phiên và thống nhất các thuật toán mã hóa được sử dụng. Sơ đồ bắt tay được minh họa trong hình 8-6. Sơ đồ trên gồm có 10 loại thông điệp và được chia thành 4 pha:

1. **Pha 1:** thỏa thuận về phương pháp mã hóa được sử dụng. Pha này bắt đầu bằng thông điệp *client_hello* được gửi từ client đến website, thông điệp này gồm các tham số sau:

Version: phiên bản SSL cao nhất mà client sử dụng

Random: là một cấu trúc ngẫu nhiên gồm 32 byte

SessionID: nếu bằng 0 có nghĩa là client muốn thiết lập một session mới hoàn toàn. Nếu khác 0 nghĩa là client muốn thiết lập một kết nối mới trong session này. Việc dùng session giúp cho client và server giảm các bước thỏa thuận trong quá trình bắt tay.

CompressionMethod: phương pháp nén dữ liệu sử dụng trong quá trình truyền dữ liệu.

CipherSuite: Các phương pháp mã hóa khóa công khai dùng để trao đổi khóa phiên như RSA, Fixed Diffie-Hellman, Ephemeral Diffie-Hellman, Anonymous Diffie-Hellman. Phương pháp nào liệt kê trước thì có được ưu tiên hơn. Ứng với mỗi phương pháp trao đổi khóa là danh sách các loại mã hóa đối xứng được sử dụng. Gồm các tham số sau:

- *CipherAlgorithm:* phương pháp mã hóa đối xứng sử dụng (là một trong các phương pháp mã khối RC2, DES, 3DES, IDEA, AES, Fortezza hay mã dòng RC4).
- *Hash Algorithm:* MD5 hay SHA-1.
- *CipherType:* mã hóa đối xứng là mã khối hay mã dòng.
- *KeyMaterial:* một chuỗi byte được dùng để sinh khóa.
- *IV Size:* kích thước của IV dùng trong mô hình CBC của mã khối.

Sau khi nhận được *client_hello* server sẽ trả lời bằng thông điệp *server_hello* để xác các thuật toán được sử dụng.

2. **Pha 2:** chứng thực server và trao đổi khóa của mã hóa công khai. Sau khi đã xác nhận thuật toán mã hóa với client, server tiếp tục thực hiện các thông điệp sau:

- Thông điệp certificate: server cung cấp certificate của mình cho client (dưới dạng chứng chỉ X.509).

- Thông điệp certificate_request: trong trường hợp server cần chứng thực người sử dụng, server sẽ gửi thông điệp này để yêu cầu client cung cấp chứng chỉ.
- Thông điệp server_hello_done: báo hiệu server đã hoàn tất pha 2.

3. Pha 3: chứng thực client và trao đổi khóa của mã hóa đối xứng.

- Thông điệp certificate: nếu server yêu cầu certificate, client cung cấp certificate của mình cho server.
- Thông điệp client_key_exchange: trong bước này client gửi các thông số cần thiết cho server để tạo khóa bí mật. Ta cũng sẽ chỉ đề cập đến trường hợp RSA. Trong trường hợp này client tạo một giá trị bất kỳ gọi là "tiền khóa chủ" (pre-master secret) có kích thước 48 byte, mã hóa bằng khóa công khai của server. Sau khi có "pre-master secret", client và server sẽ tính giá trị "khóa chủ" (master-secret) như sau:

```
master_secret = MD5(pre_master_secret || SHA('A' ||
                                             pre_master_secret||ClientHello.random || ServerHello.random) ) ||
                  MD5(pre_master_secret || SHA('BB' || pre_master_secret ||
                                             ClientHello.random || ServerHello.random) ) ||
                  MD5(pre_master_secret || SHA('CCC' || pre_master_secret ||
                                             ClientHello.random || ServerHello.random) )
```

Master_secret cũng có chiều dài là 48 byte (384 bit). Phép toán || là phép nối

- Thông điệp certificate_verify: là chữ ký của client trong trường hợp server cần chứng thực client. Client phải dùng khóa riêng để ký chữ ký, do đó server có thể đảm bảo được là không ai khác dùng certificate của client để giả mạo.
- 4. Pha 4:** hoàn tất quá trình bắt tay. Trong pha này client và server gửi thông điệp finished để thông báo hoàn tất quá trình bắt tay lẫn nhau. Tham số của thông điệp này là một giá trị hash để hai bên có thể kiểm tra lẫn nhau. Giá trị hash này kết nối của 2 giá trị hash:

```

MD5(master_secret || pad2 || MD5(handshake_messages) || Sender ||  

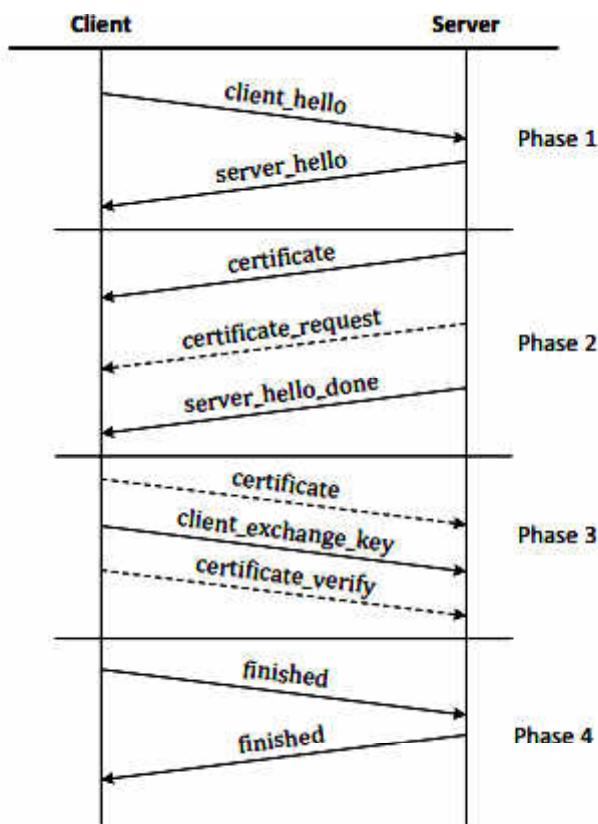
master_secret || pad1))

SHA(master_secret || pad2 || SHA(handshake_messages) || Sender ||  

master_secret || pad1))

```

Trong đó handshake_messages là tất cả các thông điệp đầu đến trước thông điệp finished này. Sender là mã để phân biệt thông điệp finished này là từ client hay từ server. Đây là cơ chế chống replay attack dùng hàm hash mà chúng ta đã tìm hiểu trong chương 6.



(đường nét đứt là các thông điệp không bắt buộc, chỉ sử dụng khi cần chứng thực từ phía client)

Hình 8.10: Giao thức bắt tay SSL

Dựa trên giá trị master_secret, client và server sẽ tính các tham số cần thiết cho mã hóa đối xứng như sau:

- Hai khóa dành cho việc mã hóa dữ liệu, một khóa dành cho chiều server gửi client và 1 khóa dành cho chiều client và server.
- Hai giá trị IV, cũng dành cho server và client tương ứng

- Hai khóa dành cho việc tính giá trị MAC, cũng tương ứng cho server và client.

Tùy theo phương pháp mã hóa đổi xứng được sử dụng mà các tham số này có chiều dài khác nhau. Tuy nhiên, chúng được lấy từ dãy bít theo công thức sau:

```
key_block = MD5( master_secret || SHA('A' || master_secret ||

    ServerHello.random || ClientHello.random) ) ||
    MD5(master_secret || SHA('BB' || master_secret ||

    ServerHello.random || ClientHello.random) ) ||
    MD5(master_secret || SHA('CCC' || master_secret ||

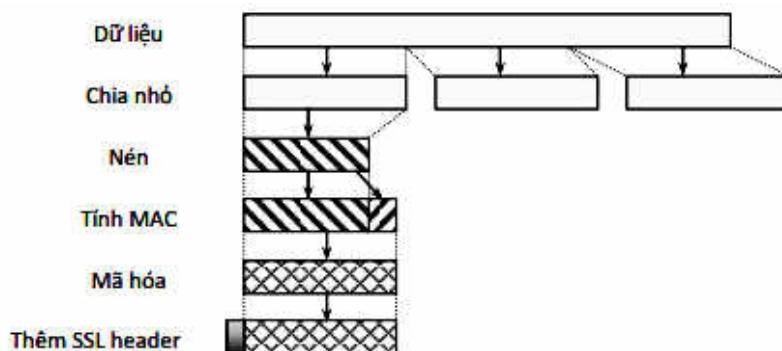
    ServerHello.random || ClientHello.random) )
```

Việc dùng các giá trị ClientHello.random và ServerHello.random sẽ làm phức tạp việc phá mã hơn.

Đến đây client và server đã hoàn tất quá trình bắt tay trao đổi khóa, sẵn sàng để truyền số liệu theo giao thức truyền số liệu.

8.2.5 Giao Thức Truyền Số Liệu-SSL Record Protocol

Hình bên dưới minh họa các bước thực hiện trong quá trình truyền số liệu:



Hình 7.1: Truyền dữ liệu theo khối trong SSL

Trong giao thức truyền số liệu, dữ liệu được chia thành các khối có kích thước là 214 byte (16384) Sau đó, dữ liệu này được nén lại. Tuy nhiên hiện nay trong SSL version 3 chưa mô tả cụ thể một phương pháp nén nào nên mặc định xem như là không nén.

Bước tiếp theo giá trị MAC của khối dữ liệu nén được tính theo công thức sau:

```
hash(MAC_key || pad_2 || hash(MAC_key || pad_1 || seq_num || type || length || data))
```

trong đó:

- *Hàm hash*: là hàm MD5 hay SHA-1.
- *MAC_key*: khóa tính MAC đã được client và server thống nhất trong phần bắt tay.
- *pad_1*: byte 0x36 (00110110) được lặp lại 48 lần (384 bit) đối với hàm hash MD5 và 40 lần (320 bit) đối với hàm hash SHA-1.
- *pad_2*: byte 0x5C (10101100) được lặp lại 48 lần đối với MD5 và 40 lần với SHA-1.
- *seq_num*: số thứ tự của khối dữ liệu.
- *type*: loại khối dữ liệu (xem phần bên dưới).
- *length*: kích thước khối dữ liệu.
- *data*: khối dữ liệu.

Sau khi tính MAC xong, khối dữ liệu cùng với giá trị MAC được mã hóa bằng một thuật toán mã khối đã được lựa chọn trong giao thức bắt tay.

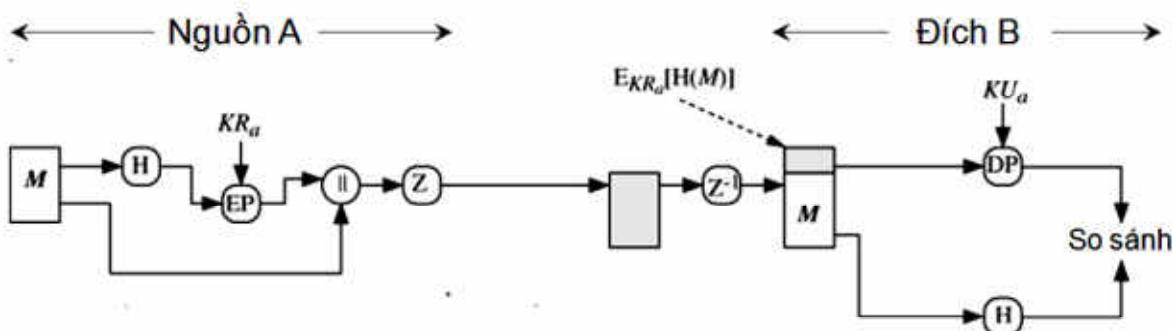
Cuối cùng một SSL header được gắn vào đầu khối dữ liệu. SSL header gồm các field sau:

- *Content Type* (1 byte): Ngoài việc truyền dữ liệu của giao thức HTTP, SSL Record Protocol còn được dùng để truyền dữ liệu của giao thức Handshake cũng như hai giao thức còn lại SSL Change Cipher Spec và SSL Alert. Giá trị của field này dùng để xác định loại giao thức đang được sử dụng. Đối với giao thức giao thức Handshake dữ liệu được truyền thẳng không cần nén, tính MAC và mã hóa.
- *Major Version* (1 byte): số hiệu chính của phiên bản SSL. Với SSLv3 field này có giá trị là 3.
- *Minor Version* (1 byte): số hiệu phụ của phiên bản SSL. Với SSLv3 field này có giá trị là 0.
- *Compressed Length* (2 byte): kích thước tính bằng byte của khối dữ liệu sau bước nén.

8.3 BẢO MẬT MAIL PGP

PGP (*Pretty Good Privacy*) là một dịch vụ về bảo mật và xác thực được sử dụng rộng rãi cho chuẩn an toàn thư điện tử. PGP được phát triển bởi Phil Zimmermann. Ở đây lựa chọn các thuật toán mã hoá tốt nhất để dùng, tích hợp thành một chương trình thống nhất, có thể chạy trên Unix, PC, Macintosh và các hệ thống khác. Ban đầu là miễn phí, bây giờ có các phiên bản thương mại. Sau đây chúng ta xem xét hoạt động của PGP.

8.3.1 Xác thực trong PGP



M = Thông báo gốc
 H = Hàm băm
 \parallel = Ghép
 Z = Nén
 Z^{-1} = Cởi nén

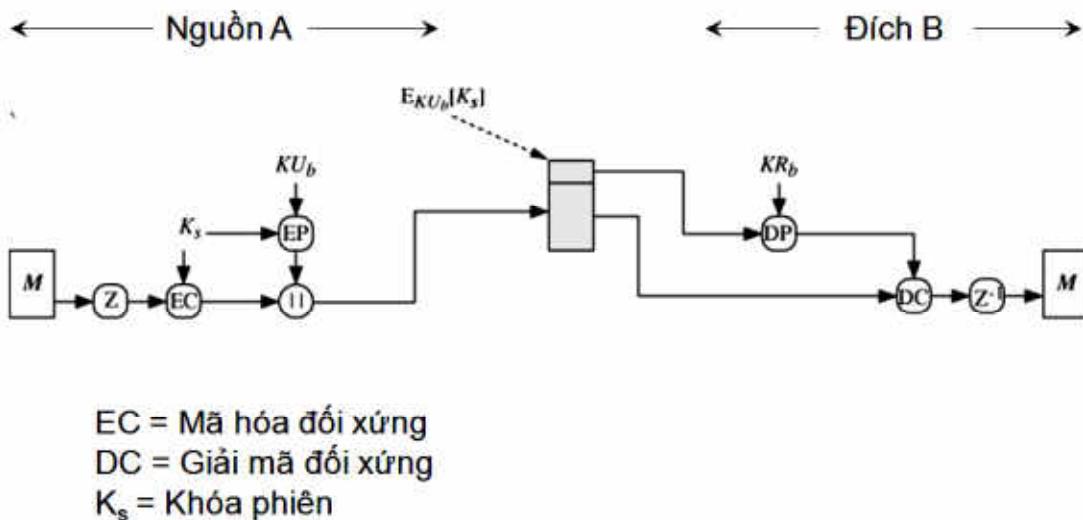
EP = Mã hóa khóa công khai
 DP = Giải mã khóa công khai
 KR_a = Khóa riêng của A
 KU_a = Khóa công khai của A

Hình 8.11: Sơ đồ xác thực PGP

Người gửi tạo mẫu tin, sử dụng SHA-1 để sinh Hash 160 bit của mẫu tin, ký hash với RSA sử dụng khóa riêng của người gửi và đính kèm vào mẫu tin.

Người nhận sử dụng RSA với khóa công khai của người gửi để giải mã và khôi phục bản hash. Người nhận kiểm tra mẫu tin nhận sử dụng bản hash của nó và so sánh với bản hash đã được giải mã.

8.3.2 Bảo mật PGP

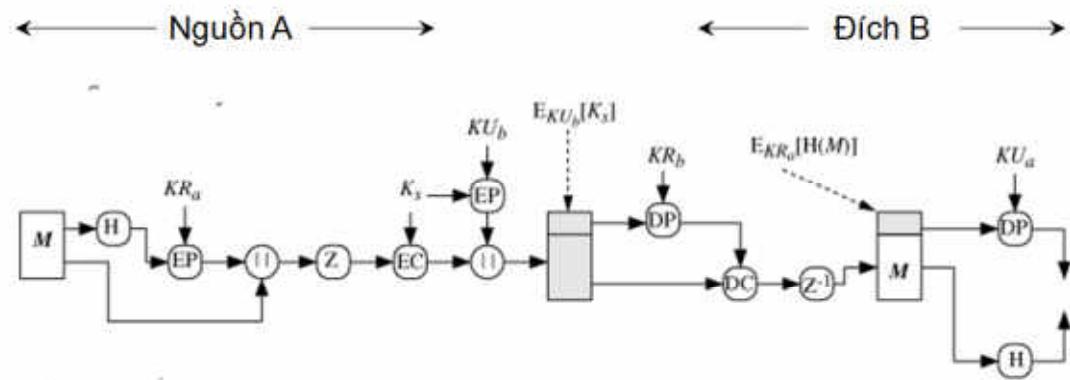


Hình 8.12: Sơ đồ bảo mật PGP

Người gửi tạo mẫu tin và số ngẫu nhiên 128 bit như khóa phiên cho nó, mã hoá mẫu tin sử dụng CAST-128/IDEA /3DES trong chế độ CBC với khóa phiên đó. Khóa phiên được mã sử dụng RSA với khóa công khai người nhận và đính kèm với mẫu tin.

Người nhận sử dụng RSA với khóa riêng để giải mã và khôi phục khóa phiên. Khóa phiên được sử dụng để giải mã mẫu tin.

8.3.3 Bảo mật và xác thực trong PGP



Hình 8.13: Sơ đồ PGP cung cấp cả dịch vụ xác thực và bảo mật

Có thể sử dụng cả hai dịch vụ trên cùng một mẫu tin. Tạo chữ ký và đính vào mẫu tin, sau đó mã cả mẫu tin và chữ ký. Đính khóa phiên đã được mã hoá RSA/ElGamal.

8.3.4 Thao tác PGP – nén

Theo mặc định PGP nén mẫu tin sau khi ký nhưng trước khi mã. Như vậy cần lưu mẫu tin chưa nén và chữ ký để kiểm chứng về sau. Vì rằng nén là không duy nhất. Ở đây sử dụng thuật toán nén ZIP.

8.3.5 Thao tác PGP – tương thích thư điện tử

Khi sử dụng PGP sẽ có dữ liệu nhị phân để gửi (mẫu tin được mã). Tuy nhiên thư điện tử có thể thiết kế chỉ cho văn bản. Vì vậy PGP cần mã dữ liệu nhị phân thô vào các ký tự ASCII in được. Sau đó sử dụng thuật toán Radix 64, ánh xạ 3 byte vào 4 ký tự in được và bổ sung kiểm tra thừa quay vòng CRC để phát hiện lỗi khi truyền. PGP sẽ chia đoạn mẫu tin nếu nó quá lớn.

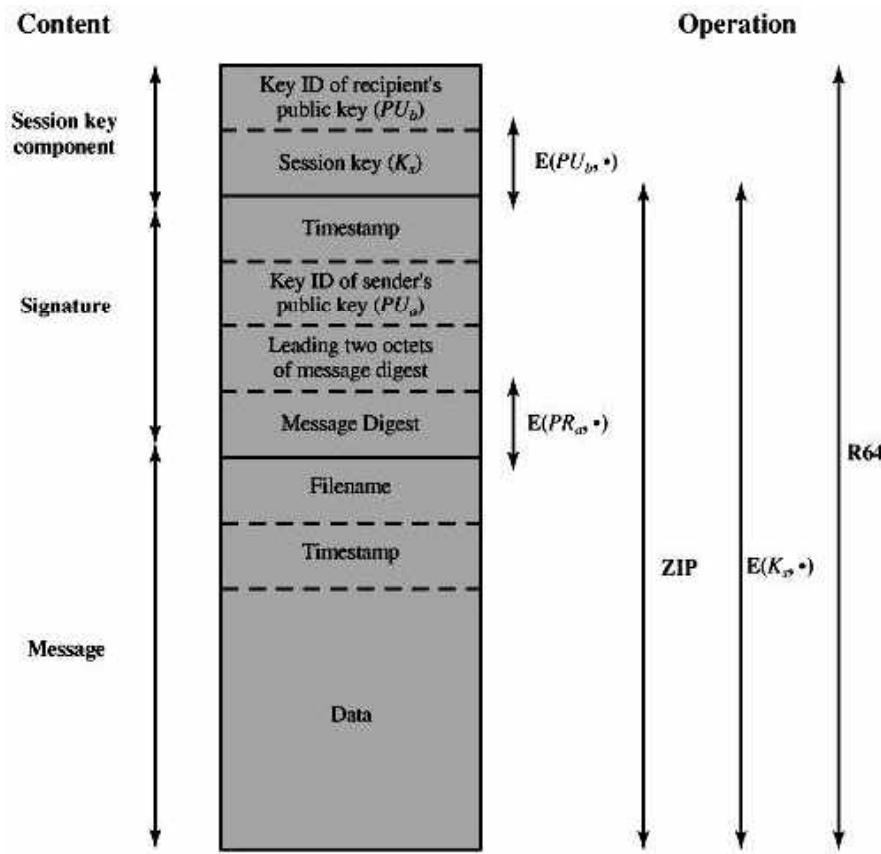
Tóm lại, cần có khóa phiên cho mỗi mẫu tin, có kích thước khác nhau: 56 bit – DES, 128 bit CAST hoặc IDEA, 168 bit Triple – DES, được sinh ra sử dụng dữ liệu đầu vào ngẫu nhiên lấy từ sử dụng trước và thời gian gõ bàn phím của người sử dụng.

6-bit value	character encoding						
0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	C	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/
					(pad)		=

Hình 8.14: Bảng chuyển đổi cơ số 64

8.3.6 Khóa riêng và công khai của PGP

Vì có nhiều khóa riêng và khóa công khai có thể được sử dụng, nên cần phải xác định rõ cái nào được dùng để mã hóa phiên trong mẫu tin. Có thể gửi khóa công khai đầy đủ với từng mẫu tin. Nhưng điều đó là không đủ, vì cần phải nêu rõ danh tính của người gửi. Do đó có thể sử dụng định danh khóa để xác định người gửi. Có ít nhất 64 bit có ý nghĩa của khóa và là duy nhất, có thể sử dụng định danh của khóa trong chữ ký.



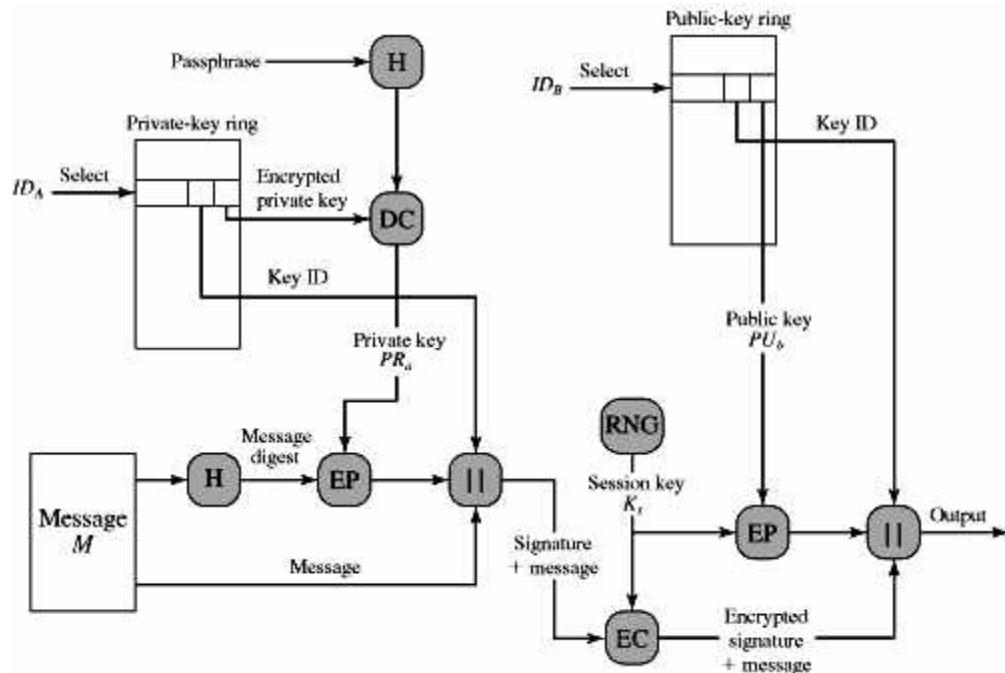
Hình 8.15: Khuôn dạng gói tin PGP

8.3.7 Các chùm khóa PGP

Mỗi người sử dụng PGP có một cặp chùm khóa. Chùm khóa công khai chứa mọi khóa công khai của các người sử dụng PGP khác được người đó biết và được đánh số bằng định danh khóa (ID key). Chùm khóa riêng chứa các cặp khóa công khai/riêng của người đó được đánh số bởi định danh khóa và mã của khóa lấy từ giai đoạn duyệt hash. An toàn của khóa công khai như vậy phụ thuộc vào độ an toàn của giai đoạn duyệt.

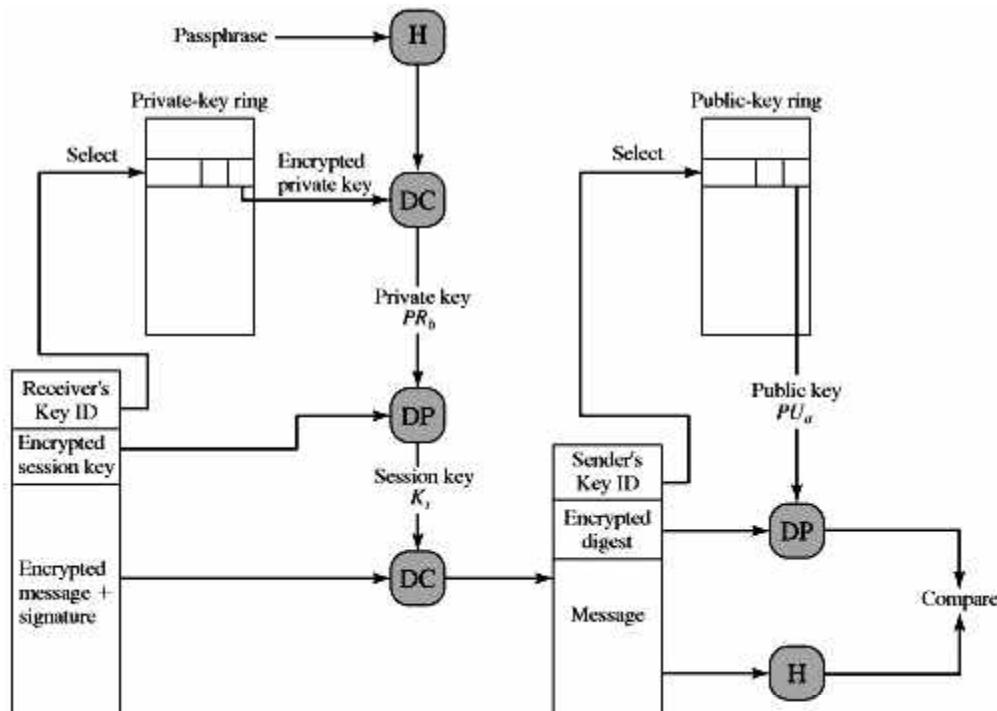
8.3.8 Tạo mẫu tin PGP

Sơ đồ sau mô tả qui trình sinh mẫu tin PGP để gửi cho người nhận.



Hình 8.16: Sơ đồ tạo thông báo PGP

8.3.9 Nhận mẫu tin PGP



Hình 8.17: Sơ đồ nhận mẫu tin PGP.

Sơ đồ trên nêu chi tiết cách người nhận giải mã, kiểm chứng thông tin để đọc mẩu tin.

8.3.10 Quản lý khóa PGP

Tốt hơn hết dựa vào chủ quyền chứng nhận. Trong PGP mỗi người sử dụng có một CA của mình. Có thể ký khóa cho người sử dụng mà anh ta biết trực tiếp. Tạo thành "Web của niềm tin". Cần tin cậy khóa đã được ký, và tin cậy các khóa mà các người khác ký khi dùng một dây chuyền các chữ ký đến nó. Chùm khóa chứa cả các chỉ dẫn tin cậy. Người sử dụng có thể thu hồi khóa của họ.

TÓM TẮT

Bài học cung cấp kiến thức về chứng minh thư X.509, bảo mật web SSL và bảo mật mail PGP.

Dịch vụ xác thực X.509 là một phần của chuẩn dịch vụ thư mục CCITT X.500. Ở đây, các máy chủ phân tán bảo trì cơ sở dữ liệu thông tin của người sử dụng và xác định khung cho các dịch vụ xác thực. Thư mục chứa các chứng nhận khóa công khai, khóa công khai của người sử dụng được ký bởi chủ quyền chứng nhận. Vì chứng chỉ được ký bằng khóa riêng của CA, nên bảo đảm rằng chữ ký không thể bị làm giả và bất cứ ai tin tưởng vào khóa công khai của CA thì có thể tin tưởng vào chứng chỉ mà CA đó cấp phát. Do đó khóa công khai của CA phải được cung cấp một cách tuyệt đối an toàn đến tay người sử dụng.

Giao thức SSL (Secure Socket Layer) bảo mật dữ liệu trao đổi qua socket. Đây là giao thức bảo mật kết hợp mã hóa khóa công khai và khóa đối xứng trong đó mã hóa RSA được dùng để trao đổi khóa phiên của mã hóa đối xứng.

Kiến trúc xử lý của SSL gồm có bốn loại xử lý: (1) SSL Record Protocol: Xử lý dữ liệu, (2) SSL Change Cipher Spec: một message đơn 1 byte, cập nhật lại bộ mã hóa để sử dụng trên kết nối này, (3) SSL Alert được dùng để truyền cảnh báo liên kết SSL với đầu cuối bên kia, và (4) SSL Handshake Protocol: Giao thức này cho phép server và client chứng thực với nhau và thương lượng cơ chế mã hóa, thuật toán MAC và khóa mật mã được sử dụng để bảo vệ dữ liệu được gửi trong SSL record.

PGP (*Pretty Good Privacy*) là một dịch vụ về bảo mật và xác thực được sử dụng rộng rãi cho chuẩn an toàn thư điện tử. PGP được phát triển bởi Phil Zimmermann.

CÂU HỎI ÔN TẬP

Câu 1: Tại sao nếu Bob tin tưởng vào khóa công khai của trung tâm chứng thực X thì Bob có thể tin tưởng vào khóa công khai của Alice? (khóa này được nhúng trong chứng chỉ X.509 do X cấp cho Alice)?

Câu 2: Trong giao thức SSL, client có cần cung cấp chứng chỉ X.509 cho server không?

Câu 3: Trong giao thức SSL, dữ liệu Web (HTML) được mã hóa dùng phương pháp mã hóa khóa công khai hay mã hóa đối xứng?

Câu 4: Trình bày các khóa được sử dụng của SSL, giải thích ý nghĩa bảo mật của nó?

Câu 5: PGP là gì? PGP cung cấp những dịch vụ bảo mật gì? Giải thích tính bảo mật của các sơ đồ sơ đồ xử lý dịch vụ của PGP?

BÀI 9: ỨNG DỤNG KIỂM SOÁT TRUY CẬP

Sau khi học xong bài này, sinh viên hiểu:

- Các khái niệm cơ bản về hệ mã đối xứng và các yêu cầu liên quan đến việc xây dựng hệ mã đối xứng.
- Hệ mã đẩy, cách tính toán và ý nghĩa bảo mật của nó.
- Hệ mã hóa hoán vị.

9.1 TỔNG QUAN VỀ KIẾN TRÚC AAA

AAA cho phép nhà quản trị mạng biết được các thông tin quan trọng về tình hình cũng như mức độ an toàn trong mạng. Nó cung cấp việc xác thực (authentication) người dùng nhằm bảo đảm có thể nhận dạng đúng người dùng. Một khi đã nhận dạng người dùng, ta có thể giới hạn thẩm quyền (authorization) mà người dùng có thể làm. Khi người dùng sử dụng mạng, ta cũng có thể giám sát tất cả những gì mà họ làm. AAA có thể dùng để tập hợp thông tin từ nhiều thiết bị trên mạng. Ta có thể bật các dịch vụ AAA trên router, switch, firewall, các thiết bị VPN, server, ...

Các dịch vụ AAA được chia thành ba phần, xác thực (*authentication*), cấp quyền (*accounting*), tính cước (*accounting*). Ta sẽ tìm hiểu sự khác nhau của ba phần này và cách thức chúng làm việc như thế nào.

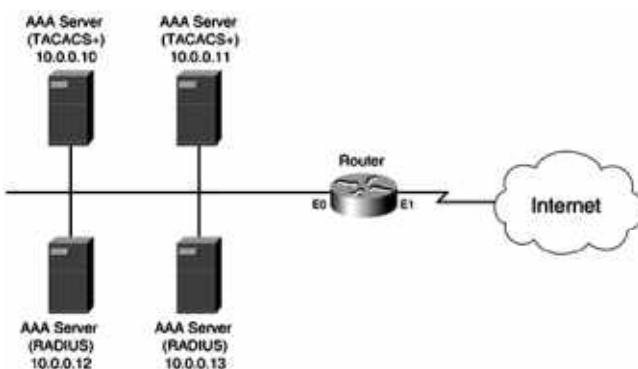
- *Xác thực (Authentication)*: Xác thực dùng để nhận dạng người dùng. Trong suốt quá trình xác thực, username và password của người dùng được kiểm tra và đối chiếu với cơ sở dữ liệu lưu trong AAA Server. Tất nhiên, tùy thuộc vào giao thức mà AAA hỗ trợ mã hóa đến đâu, ít nhất thì cũng mã hóa username và password. Xác thực sẽ xác định người dùng là ai. Một khi username và password được chấp

nhận, AAA có thể dùng để định nghĩa thẩm quyền mà người dùng được phép làm trong hệ thống.

- *Thẩm quyền (Authorization)*: Authorization cho phép nhà quản trị điều khiển việc cấp quyền trong một khoảng thời gian, hay trên từng thiết bị, từng nhóm, từng người dùng cụ thể hay trên từng giao thức. AAA cho phép nhà quản trị tạo ra các thuộc tính mô tả các chức năng của người dùng được phép làm. Do đó, người dùng phải được xác thực trước khi cấp quyền cho người đó.

AAA Authorization làm việc giống như một tập các thuộc tính mô tả những gì mà người dùng đã được xác thực có thể có. Ví dụ: người dùng THIEN sau khi đã xác thực thành công có thể chỉ được phép truy cập vào server VNLABPRO_SERVER thông qua FTP. Những thuộc tính này được so sánh với thông tin chứa trong cơ sở dữ liệu của người dùng đó và kết quả được trả về AAA để xác định khả năng cũng như giới hạn thực tế của người đó. Điều này yêu cầu cơ sở dữ liệu phải giao tiếp liên tục với AAA server trong suốt quá trình kết nối đến thiết bị truy cập từ xa (RAS).

- *Tính cước (Accounting)*: Accounting cho phép nhà quản trị có thể thu thập thông tin như thời gian bắt đầu, thời gian kết thúc người dùng truy cập vào hệ thống, các câu lệnh đã thực thi, thống kê lưu lượng, việc sử dụng tài nguyên và sau đó lưu trữ thông tin trong hệ thống cơ sở dữ liệu quan hệ. Nói cách khác, accounting cho phép giám sát dịch vụ và tài nguyên được người dùng sử dụng. Ví dụ: thống kê cho thấy người dùng có tên truy cập là SON đã truy cập vào SERVER bằng giao thức FTP với số lần là 5 lần. Điểm chính trong Accounting đó là cho phép người quản trị giám sát tích cực và tiên đoán được dịch vụ và việc sử dụng tài nguyên. Thông tin này có thể được dùng để tính cước khách hàng, quản lý mạng, kiểm toán sổ sách.



Hình 9.1: Các kiểu ứng dụng cung cấp dịch vụ AAA

Có hai giao thức bảo mật dùng trong dịch vụ AAA đó là TACACS (*Terminal Access Controller Access Control System*) và RADIUS (*Remote Authentication Dial-In User Service*). Cả hai giao thức đều có phiên bản và thuộc tính riêng. Chẳng hạn như phiên bản riêng của TACACS là TACACS+, tương thích hoàn toàn với TACACS. RADIUS cũng có sự mở rộng khi cho phép khách hàng thêm thông tin xác định được mang bởi RADIUS.

TACACS và RADIUS được dùng từ một thiết bị như là server truy cập mạng (NAS) đến AAA server. Xem xét một cuộc gọi từ xa. Người dùng gọi từ PC đến NAS. NAS sẽ hỏi thông tin để xác thực người dùng. Từ PC đến NAS, giao thức sử dụng là PPP, và một giao thức như là CHAP hay PAP được dùng để truyền thông tin xác thực. NAS sẽ truyền thông tin đến AAA Server để xác thực. Nó được mang bởi giao thức TACACS hoặc RADIUS.

9.2 TACACS+

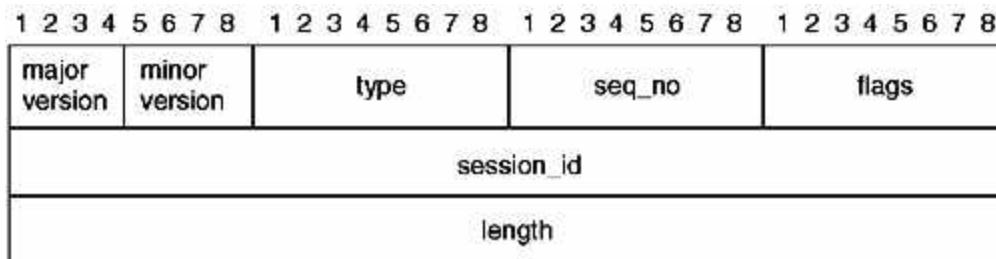
TACACS (*Terminal Access Controller Access Control System*) là giao thức được chuẩn hóa sử dụng giao thức hướng kết nối (*connection-oriented*) là dựa trên nền TCP trên port 49. Bản thân TACACS+ là một giao thức non-standard, hay chính xác hơn là proprietary protocol, TACACS+ là kết quả của quá trình phát triển dựa trên hai giao thức TACACS và TACACS mở rộng (XTACACS). Hệ điều hành IOS của Cisco hỗ trợ cả 3 loại giao thức này.

TACACS+ (*Terminal Access controller Access- Control System Plus*) là một giao thức độc quyền của Cisco, được thiết kế dùng trong kiến trúc AAA cho việc chứng thực, ủy quyền và kế toán trong môi trường mạng.

Tương tự như một cơ sở dữ liệu bảo mật nội bộ, TACACS+ hỗ trợ ba tính năng yêu cầu của một hệ thống bảo mật tốt.

9.2.1 Khuôn dạng gói tin Tacacs+

TACACS+ ID định nghĩa 12 bytes header xuất hiện trong tất cả các gói tin của TACACS+. Cấu trúc của giao thức TACACS+: gồm cấu trúc phần header và Data.



Hình 9.2: Cấu trúc Header của TACACS+

Version number: bao gồm major_version và minor_version. Major_version: số phiên bản chính của Tacacs+. Giá trị xuất hiện trong tiêu đề như TAC_PLUS_MAJOR_VER = 0xc. Minor_version: số phiên bản phụ của Tacacs+.cung cấp số serial cho giao thức. Nó cũng cung cấp cho khả năng tương thích của giao thức. Một giá trị mặc định, cũng như phiên bản một, được định nghĩa cho một số lệnh. Những giá trị này xuất hiện trong tiêu đề như TAC_PLUS_MINOR_VER_DEFAULT = 0x0, TAC_PLUS_MINOR_VER_ONE = 0x1. Trường này được thiết kế để cho phép sửa đổi các giao thức TACACS + trong khi duy trì tính tương thích ngược. Nếu một máy chủ AAA chạy TACACS+ nhận được gói tin TACACS xác định một phiên bản nhỏ hơn khác phiên bản hiện tại, nó sẽ gửi một trạng thái lỗi trả lại và yêu cầu các minor_version với phiên bản gần nhất được hỗ trợ.

Type: dùng để phân biệt loại gói tin. Các loại hợp lệ như: **TAC_PLUS_AUTHEN = 0x01** đây là loại gói nghĩa xác thực. **TAC_PLUS_AUTHOR = 0x02** đây là loại gói tin mà nghĩa ủy quyền. **TAC_PLUS_ACCT = 0x03** đây là loại gói tin mà nghĩa kế toán.

Seq-no: số thứ tự của các gói tin hiện tại trong phiên làm việc.TACACS có thể tạo một hoặc nhiều phiên TACACS cho mỗi khách hàng AAA.

Flags: những giá trị biểu xem gói dữ liệu được mã hóa. Có thể chứa các cờ khác nhau, các cờ này có thể là:

- **TAC_PLUS_UNENCRYPTED_FLAG:** cờ này được chỉ định nếu mã hóa đang thực hiện trên phần thân của gói tin TACACS+. Nếu cờ này được thiết lập (1), mã hóa không được thực hiện ngược lại (0) gói tin đang được mã hóa.

- TAC_PLUS_SINGLE_CONNECT_FLAG: quyết định có hoặc không ghép nhiều phiên TACACS+ trên một phiên TCP được hỗ trợ.

Session_ID: ID của phiên làm việc, nó được cấp ngẫu nhiên chỉ định phiên hiện tại giữa AAA client và AAA server.

Length: tổng chiều dài gói tin Tacacs+ (không bao gồm phần header).

Data: chứa thông tin liên lạc giữa Tacacs+ client (Network Access Server) và Tacacs+ Server (AAA server).

9.2.2 Cơ chế mã hóa trong Tacacs+

Để tăng thêm sự an toàn cho việc truyền thông điệp, giao thức TACACS+ cung cấp mã hóa cho toàn bộ gói tin. Sự mã hóa này được gửi giữa AAA client và AAA server chạy TACACS+ daemon. Điều này không làm lẫn lộn với sự mã hóa dữ liệu của user. TACACS+ sử dụng hàm băm MD5 và sử dụng một khóa bí mật được cung cấp cả 2 đầu server và client. Quy trình mã hóa diễn ra như sau:

Bước 1: Thông tin được lấy từ header của gói tin và khóa chia sẻ trước tính toán một chuỗi băm. Hash đầu được tính toán với đầu vào là một dãy session_id, version, sep_no, và giá trị khóa chia sẻ trước. Mỗi hash được tạo ra có hash trước đó. Điều này được thực hiện một số lần phụ thuộc vào sự triển khai đặc thù của TACACS+.

Bước 2: Hash sau khi được tính toán sẽ được nối vào nhau và sau đó được cắt thành độ dài dữ liệu được mã hóa. Mỗi hash có hash trước đó được nối vào nó thành dữ liệu đầu vào. Kết quả được gọi là pseudo_pad.

(pseudo_pad = {MD5_1 [, MD5_2 [..., MD5_n]]} cắt ngắn để len (dữ liệu))

Bước 3: Văn bản mật mã được đưa ra bằng cách làm phép toán XOR giữa pseudo_pad với dữ liệu mà đang được mã hóa.

Bước 4: Thiết bị nhận sử dụng khóa được chia sẻ trước của nó để tính toán pseudo_pad và sau đó XOR với pseudo_pad mới nhất được tạo ra trong dữ liệu gốc trong đoạn văn bản sạch.

Độ dài của các MD5 Hash (pseudo_pad) = 16 bytes, số liệu được mã hóa bằng cách: ENCRYPTED {data} = data (XOR) pseudo_pad.

Các MD5 Hash được tính như sau:

```

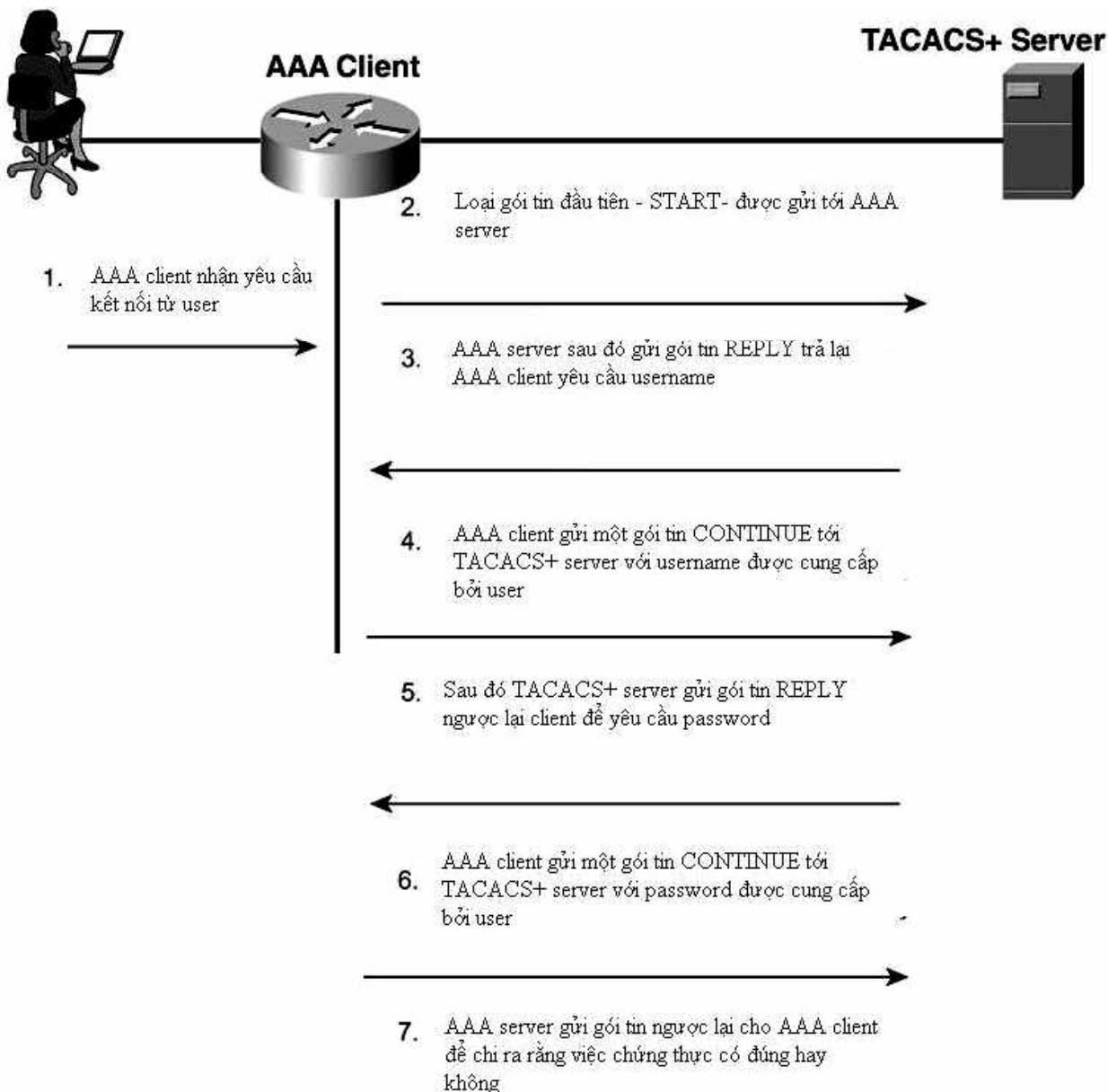
MD5_1 = MD5{session_id, key, version, seq_no}

MD5_2 = MD5{session_id, key, version, seq_no, MD5_1}

...
MD5_n = MD5{session_id, key, version, seq_no, MD5_n-1}

```

9.2.3 Cơ Chế Hoạt Động



Hình 9.3: Quá trình hoạt động Authentication của TACACS+

Ba hoạt động có thể được thực hiện trong suốt quá trình hoạt động của TACACS+.

- Hoạt động đầu tiên được thực hiện là xác thực để nhận diện người dùng.
- Hoạt động thứ 2 là ủy quyền và có thể chỉ thực hiện một lần khi user đã được nhận dạng hoàn tất. Bởi vậy, người dùng phải được chứng minh là hợp lệ trước khi ủy quyền cho họ.
- Hoạt động thứ 3 là kiểm toán. Quá trình kiểm toán theo dõi dấu vết của các hành động mà người dùng đã thực hiện khi ở trong hệ thống.

Trong giao thức TACACS+ cả ba tiến trình có thể hoạt động độc lập với nhau.

9.2.4 Authentication

Khi sự xác thực được thực hiện trong TACACS+, ba gói tin trao đổi riêng biệt diễn ra. Ba loại gói tin đó là:

START: gói tin này được sử dụng vào lúc đầu khi người dùng cố gắng kết nối.

REPLY: được gửi bởi AAA server trong suốt quá trình xác thực.

CONTINUE: được sử dụng bởi AAA client để trả về username và password cho AAA server

Các bước của tiến trình chứng thực:

Bước 1: AAA nhận kết nối yêu cầu từ người dùng

Bước 2: Loại gói tin đầu tiên - **START** – được gửi tới AAA server mà chạy TACACS+ daemon. Thông điệp **START** này chứa thông tin loại xác thực.

Bước 3: Sau đó TACACS+ server gửi gói tin REPLY ngược lại AAA client. Tại thời điểm này máy chủ yêu cầu username.

Bước 4: AAA client gửi một gói tin **CONTINUE** tới TACACS+ server với username được cung cấp bởi user.

Bước 5: Sau đó TACACS+ server gửi gói tin **REPLY** ngược lại client để yêu cầu password.

Bước 6: AAA client gửi một gói tin **CONTINUE** tới TACACS+ server với password được cung cấp bởi user.

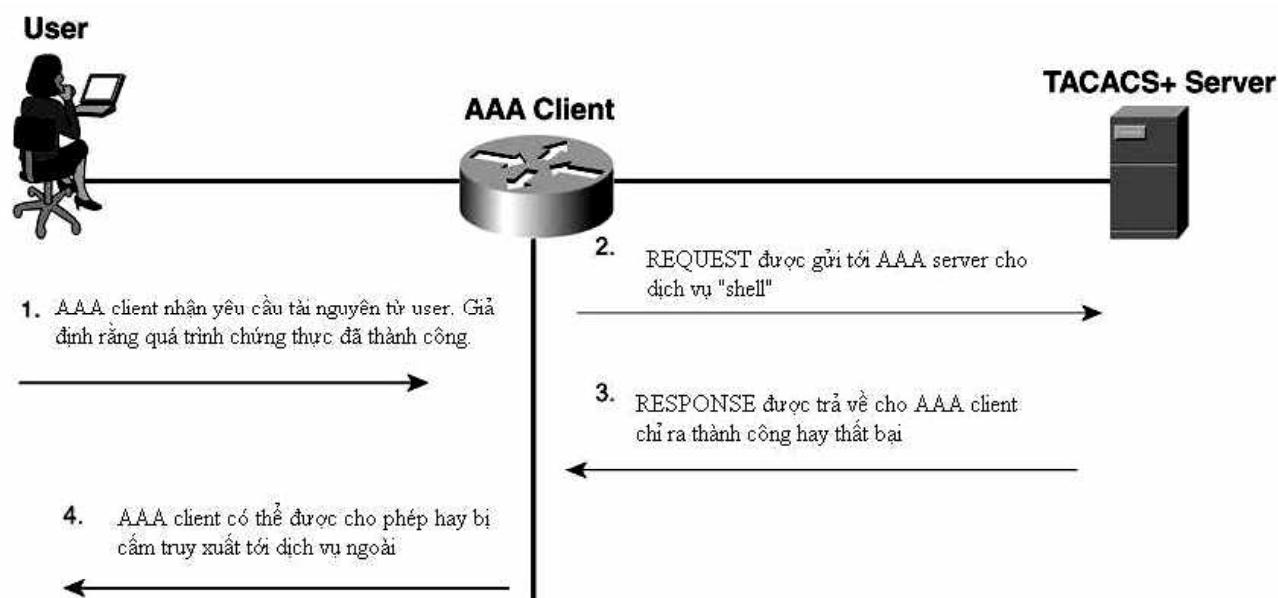
Bước 7: AAA server gửi gói tin ngược lại cho AAA client để chỉ ra rằng việc chứng thực có đúng hay không. Các giá trị có thể trả về là:

- **ACCEPT:** user đã được chứng thực và dịch vụ có thể bắt đầu. Nếu NAS được đấu hình để yêu cầu ủy quyền, ủy quyền bắt đầu tại thời điểm này.
- **REJECT:** user chứng thực thất bại. User có thể bị cấm truy cập hoặc được nhắc thử lại để đăng nhập tuân tự phụ thuộc vào TACACS+ daemon.
- **ERROR:** Một lỗi xảy ra tại thời điểm chứng thực. Điều này có thể tại daemon hoặc kết nối mạng giữa daemon và NAS. Nếu một **ERROR** phản hồi được nhận, tính đặc trưng của NAS là sẽ cố gắng sử dụng một phương thức khác để chứng thực user.
- **CONTINUE:** User được nhắc cung cấp thông tin chứng thực.

→ Điểm lưu ý ở đây là gói tin **CONTINUE** luôn được gửi bởi AAA client và gói tin **REPLY** luôn được gửi bởi TACACS+ server.

9.2.5 Authorization

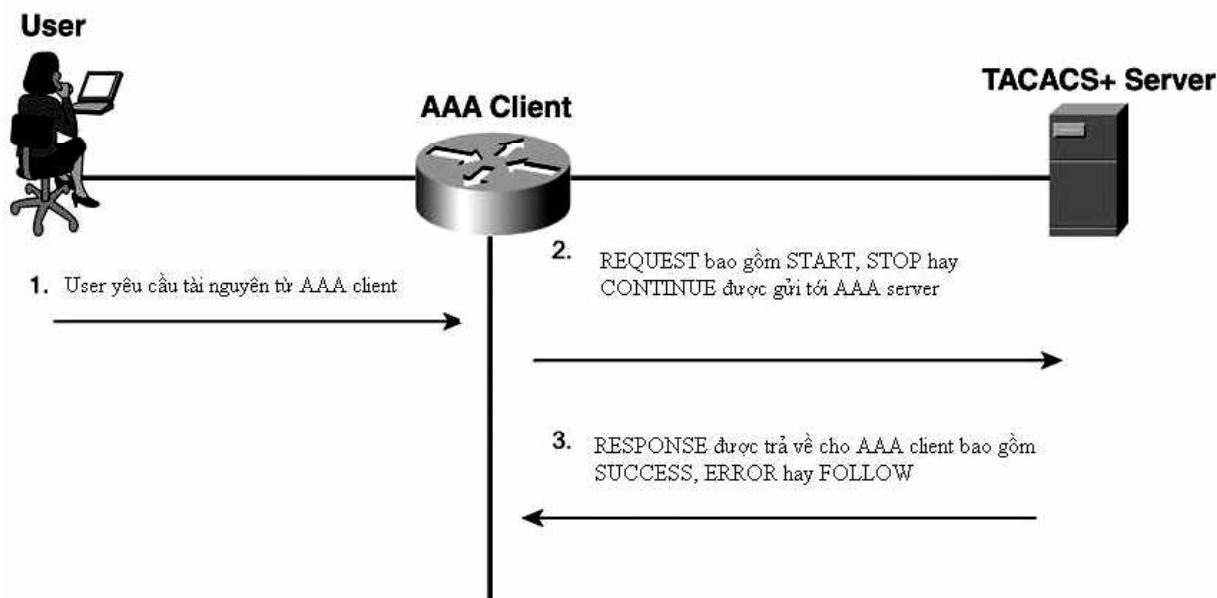
Để sự ủy quyền trong TACACS+ được thuận lợi hơn, có hai loại thông điệp được sử dụng. Thông điệp đầu tiên là một ủy quyền **REQUEST**. Thông điệp thứ hai là **RESPONSE**. Các nguồn **REQUEST** xuất phát từ AAA client và **RESPONSE** từ AAA server.



Hình 9.4: Quá trình hoạt động Authorization của TACACS+

Thông điệp **RESPONSE** trong bước 3 chứa 1 trong những phản hồi như: **FAIL**: server chỉ ra rằng những dịch vụ được yêu cầu cho ủy quyền thì không được cấp phép. **PASS_ADD**: yêu cầu đã được ủy quyền, và thông tin được trả về trong **RESPONSE** được dùng thêm vào thông tin đã yêu cầu. Nếu không có đổi số thêm vào nào được trả về bởi AAA server trong **RESPONSE**, yêu cầu đã được ủy quyền. **PASS_REPL**: trong một vài trường hợp thông điệp này có thể được trả về cho AAA client. Khi đó, server đang chọn bỏ qua **REQUEST** và đang thay thế nó với thông tin được trả về trong **RESPONSE**. Nếu trạng thái được thiết lập là **FOLLOW**, điều này chỉ ra rằng AAA server mà đang gửi **RESPONSE** muốn có sự ủy quyền lấy từ server khác, và thông tin server này được liệt kê ra trong gói **RESPONSE**. AAA client có lựa chọn là sử dụng server này hoặc có thể coi nó như là **FAIL**. Nếu trạng thái được trả về là **ERROR**, điều này chỉ ra rằng có lỗi trên AAA server. Lỗi này thông thường là không khớp khóa chia sẻ trước (pre-shared key): tuy nhiên, nó có thể là một số vấn đề và xa hơn nữa là việc xử lý sự cố cần đặt lại. Trong sự ủy quyền, Attribute-Value(AV) – giá trị thuộc tính- quyết định các dịch vụ được ủy quyền. Tham khảo bảng các cặp AV cho kiểm toán và ủy quyền.

9.2.6 Accounting



Hình 9.5: Quá trình hoạt động Accounting của TACACS+

Tính năng Accounting trên TACACS+ tương tự như sự ủy quyền. Accounting lấy bằng cách gửi một bản ghi tới AAA server. Mỗi bản ghi này chứa một cặp AV cho việc kiểm toán. Có 3 loại bản ghi có thể được gửi tới AAA server. Start Record: chỉ ra khi một dịch vụ bắt đầu và chứa thông tin đã được bao gồm trong tiến trình ủy quyền và thông tin đặc trưng của tài khoản. Stop Record: chỉ ra khi một dịch vụ dừng hoặc bị gián đoạn và cũng bao gồm thông tin đã được chứa trong tiến trình ủy quyền, cũng chỉ ra thông tin đặc trưng của tài khoản. Continue Record: hay còn gọi là Watchdog. Nó được gửi khi một dịch vụ vẫn trong tiến trình và cho phép AAA client cung cấp thông tin được cập nhật tới AAA server. Như đã thấy trong bản ghi trước, điều này cũng bao gồm thông tin mà được chứa trong tiến trình ủy quyền, cũng chỉ ra thông tin đặc trưng của tài khoản.

Accounting cũng sử dụng 2 loại thông điệp mà Authorization đã dùng. Một REQUEST và RESPONSE. AAA server có khả năng gửi các thông điệp RESPONSE sau:

- SUCCESS chỉ ra rằng server đã nhận được bản ghi từ AAA client.
- ERROR chỉ ra rằng server không thành công trong việc bỏ bản ghi vào database của nó.
- FOLLOW tương tự như FOLLOW trong Authorization. Điều này chỉ ra rằng server muốn AAA client gửi bản ghi tới một server khác và thông tin về server đó đã bao gồm trong gói tin RESPONSE.

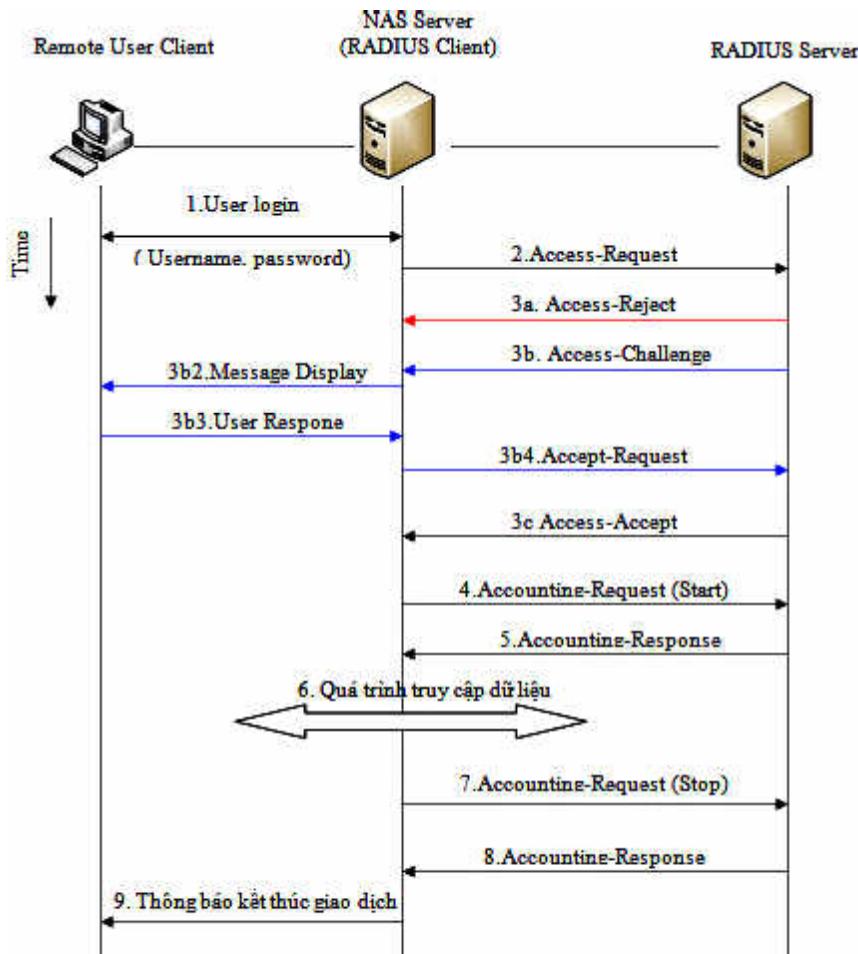
9.3 RADIUS

Radius (Remote Access Dial In User Service) là một giao thức mạng cung cấp việc quản lý xác thực tập trung, ủy quyền, và kế toán (AAA) để cho các máy tính kết nối vào mạng và sử dụng dịch vụ mạng. RADIUS được phát triển bởi Livingston Enterprises, Inc vào năm 1991 như là một giao thức xác thực truy cập máy chủ và kế toán và sau đó được đưa vào tiêu chuẩn Internet Engineering Task Force (IETF).

9.3.1 Nguyên lý hoạt động

Các máy chủ RADIUS server sử dụng mô hình AAA để quản lý quá trình truy cập mạng qua hai bước: xác thực, ủy quyền (authentication, authorization) và kế toán

(accounting). Các bước thực hiện xác thực, cấp quyền, kiểm toán trong giao thức Radius như sau:



Hình 9.6: Sơ đồ hoạt động tổng thể của RADIUS

Bước 1: User tiến hành nhập Username & Password đăng nhập vào hệ thống.

Bước 2: NAS (Radius Server) nhận được thông tin của người dùng tiến hành gửi thông điệp Access-Request để yêu cầu Server chứng thực cho người dùng.

Bước 3: Server tiến hành kiểm tra thông tin của người dùng để xác định bước kế tiếp:

- Nếu các thông tin không hợp lệ, Server sẽ phản hồi lại Acess-Reject để từ chối yêu cầu.
- Để đảm bảo Server gửi một yêu cầu Access-Challenge yêu cầu người dùng phản hồi. Người dùng tiến hành nhập mã phản hồi và gửi lên lại hệ thống. NAS chuyển tiếp phản hồi của người dùng lên Server. Server lúc này có thể từ chối bằng

Access-Reject, chấp nhận bằng Access-Accept hoặc gửi một Access-Challenger khác.

- Nếu thông tin hợp lệ Server sẽ phản hồi Access-Accept.

Bước 4: Radius Client gửi yêu cầu thực hiện dịch vụ Accounting bằng yêu cầu Accounting-Request (Start).

Bước 5: Server phản hồi đã nhận được yêu cầu bằng Accounting-Response.

Bước 6: Quá trình truy cập dữ liệu của user.

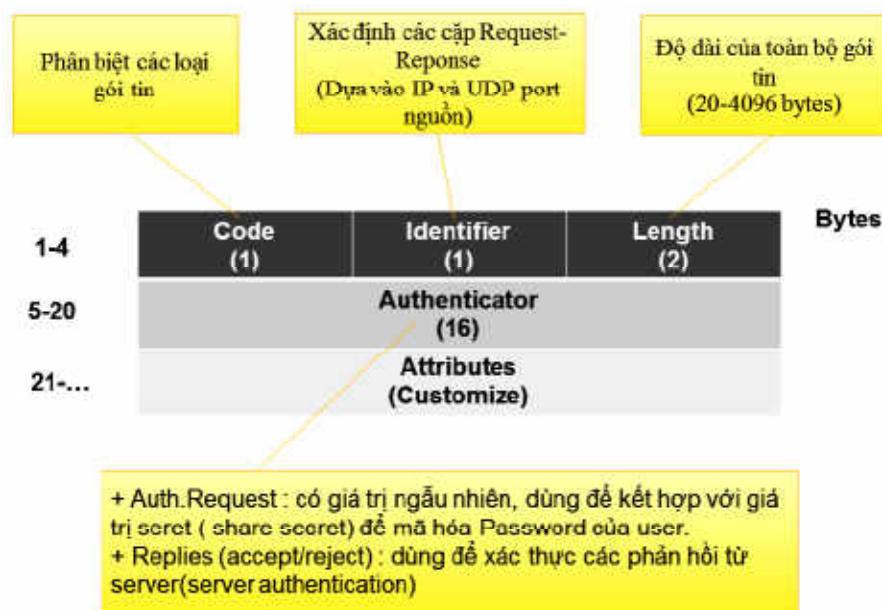
Bước 7: Khi người dùng ngắt kết nối, Radius Client gửi một yêu cầu ngưng dịch vụ Accounting bằng yêu cầu Accounting-Request (Stop).

Bước 8: Server sau khi đã nhận được yêu cầu sẽ phản hồi bằng Accounting-Response.

Bước 9: Thông báo cho người dùng phiên làm việc kết thúc.

9.3.2 Cấu trúc gói tin RADIUS

Hình 9-7 minh họa các thành phần của gói tin RADIUS. Gói tin chứa các dữ liệu như:



Hình 9.7: Minh họa cấu trúc gói tin RADIUS

Quyền

Trường Code quy định kiểu gói tin: (1) Access-Request, (2) Access-Accept, (3) Access-Reject, (11) Access-Challenge, (4) Accounting-Request, và (5) Accounting-Response.

Trường Authenticator giá trị chứng thực. Với mỗi kiểu gói tin, trường này được tính như sau:

Access-Request (RFC 2865)

Request-Authenticator = số ngẫu nhiên 16 bytes (octet)

Access-Accept/Reject/Challenge (RFC 2865)

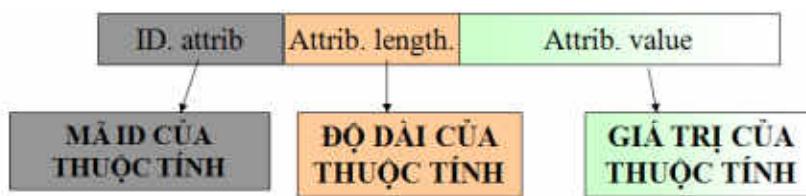
Response-Authenticator = MD5 (Code + Identifier + Length + RequestAuthenticator + Attributes + Secret)

Accounting-Request (RFC 2866)

Request-Authenticator = MD5 (Code + Identifier + Length + 16 zero octets + Attributes + Secret)

Accounting-Response (RFC 2866)

Response-Authenticator = MD5 (Code + Identifier + Length + RequestAuthenticator + Attributes + Secret)



Hình 9.8: Cấu trúc của trường thuộc tính

Trường thuộc tính có một số loại thuộc tính thông dụng sau:

User-Name (1) – Length (3 octet trở lên) thường xuất hiện trong Access-Request & Accounting-Request. Ý nghĩa là phân biệt giữa các khách hàng tham gia truy cập vào hệ thống trên mạng. Ngoài ra các máy chủ RADIUS cũng có thể sử dụng một tên người dùng để xác định hành vi thích hợp cho một giao dịch.

User-Password (2) – Length (18 -130 octet) chỉ xuất hiện trong Access-Request. Thuộc tính này được thiết kế để mang thông tin xác thực mà người dùng cung cấp để

truy cập các dịch vụ mạng. Chủ yếu, nội dung của giá trị này sẽ là một mật khẩu mã hóa, nhưng đôi khi nó có thể là phản ứng từ một gói tin Access-Challenge gửi cho khách hàng từ các máy chủ RADIUS.

CHAP-Password (3) – Length (19 octet) chỉ xuất hiện: Access-Request. Mục đích đầu tiên là cho biết giao thức xác thực người dùng sử dụng là CHAP, sẽ được sử dụng trong quá trình giao dịch. Chức năng tương tự như User-Password.

CHAP-Challenge (60) – Length (7 octet trở lên) xuất hiện trong Access-Request. Phục vụ cho việc xác thực user bằng cơ chế xác thực CHAP. CHAP challenge được tính bằng MD5 (Password + chuỗi challenge).

NAS-Identifier (32) – Length 3 octet trở lên xuất hiện trong Access-Request. Thuộc tính này xác định các NAS mà xây dựng các gói tin Access-Request. Thông thường, tên miền đầy đủ (FQDN) được sử dụng trong phần giá trị của thuộc tính này.

NAS-IP-Address (4) – Length: 6 octet xuất hiện trong Access-Request. Thuộc tính này xác định địa chỉ IP của thiết bị NAS có yêu cầu dịch vụ thay mặt cho các máy tính khách hàng nhưng RFC RADIUS không cho phép cả hai thuộc tính NAS-IP-Address và thuộc tính NAS-Identifier được sử dụng trong cùng một gói tin. Tuy nhiên, một trong hai phải có mặt trong gói tin bất kỳ.

NAS-Port (5) – Length: 6 octet xuất hiện trong Access-Request. Hiển thị port mà user đang kết nối với NAS (không phải là port dịch vụ kết nối –socket port), đại diện cho các port thực tế, hữu hình, vật lý trên các thiết bị kết nối. Nếu không có thì giá trị được đặt theo số ảo.

NAS-Port-Type (61) – Length: 6 octet (giá trị theo bảng định danh) xuất hiện trong Access-Request. Cho biết kiểu NAS-port kết nối là gì. VD: Virtual(5)

Proxy-State (61) – Length: 3 octet trở lên xuất hiện trong tất cả gói tin. Thuộc tính này được sử dụng khi một máy chủ RADIUS hoạt động như một proxy và nhu cầu để lưu thông tin về một yêu cầu nổi bật, chẳng hạn như địa chỉ IP, tên miền, hoặc các định danh duy nhất số nguyên.

Reply-Message (18) – Length: 3 octet trở lên xuất hiện trong tất cả gói tin ngoại trừ Access-Request. Giá trị này được sử dụng để cung cấp một tin nhắn cho khách hàng cho một gói tin khác. Thường được tìm thấy trong Acess-Accept, dùng để cung

cấp một thông điệp chào mừng, một thông báo lỗi, hoặc các thông tin khác cho người sử dụng.

Service-Type (6) – Length: 6 octet xuất hiện trong: Access-Request, Access-Accept. Cho biết các loại dịch vụ mạng mà Radius client cung cấp cho user.

Acct-Status-Type (40) – Length: 6 octet xuất hiện trong: Accounting-Request. Cho biết trạng thái kết nối hiện tại của dịch vụ Accounting.

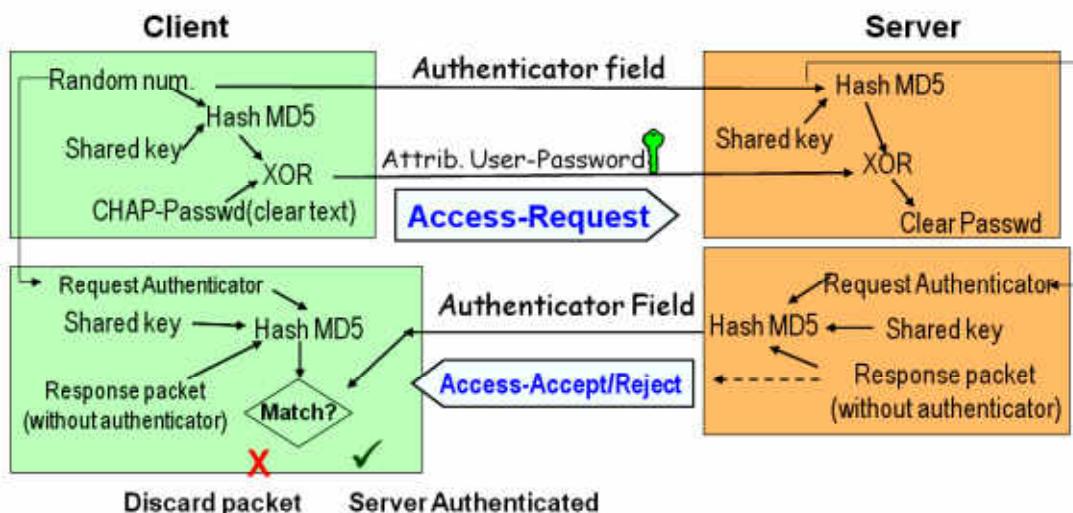
Acct-Session-Id (44) – Length: từ 3 octet trở lên xuất hiện trong: Accounting-Request, Access Request. Dùng để xác định 1 phiên làm việc.

Acct-Authentic (45) – Length: 6 octet xuất hiện trong: Accounting-Request.

Acct-Session-Time (46) – Length: 6 octet xuất hiện trong: Accounting-Request. Tính bằng giây, cho biết thời gian kết nối của người dùng.

Acct-Terminate-Cause (49) – Length: 6 octet xuất hiện trong: Accounting-Request. Cho biết nguyên nhân kết thúc một phiên làm việc của người dùng.

9.3.3 Authentication, Authorization



Hình 9.9: Quá trình xác thực của RADIUS

Người dùng hoặc máy sẽ gửi yêu cầu đến một Remote Access Server (RAS) để được quyền truy cập vào tài nguyên mạng bằng các thông tin truy cập (username, password). Các thông tin truy cập được truyền đi cho các thiết bị RAS thông qua một giao thức phù hợp chẳng hạn như: PPP PAP, PPP CHAP, PPTP, L2TP ...

Sau đó, RAS server (đảm nhiệm vai trò là một Radius Client) gửi một thông điệp RADIUS Access Request đến máy chủ RADIUS server, yêu cầu cấp quyền để truy cập thông qua giao thức RADIUS. Yêu cầu này bao gồm các thông tin truy cập, thông thường là tên người dùng và mật khẩu hoặc giấy chứng nhận bảo mật được cung cấp bởi người sử dụng.

Các máy chủ RADIUS kiểm tra thông tin đó là chính xác bằng cách sử dụng hệ thống xác thực như PAP, CHAP hoặc EAP. Thông tin về nhận dạng của người dùng được xác minh, cùng với các thông tin khác liên quan đến yêu cầu, chẳng hạn như địa chỉ mạng của người dùng hoặc số điện thoại, tình trạng tài khoản và phân quyền truy cập dịch vụ cụ thể (hình 9-9).

Các máy chủ RADIUS server sau đó trả về cho RAS server một trong các gói tin sau: Access Reject, Access Challenge, Access Accept.

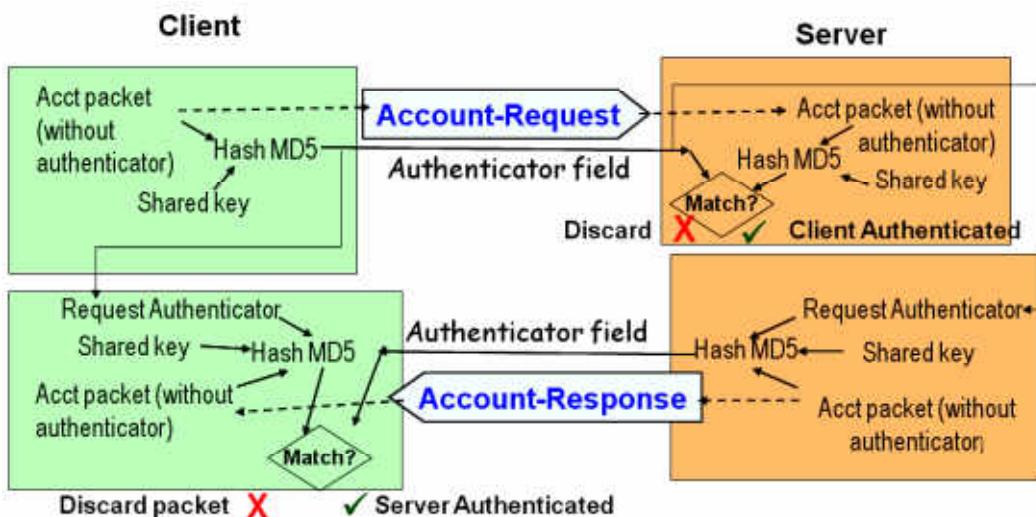
- *Access Reject* người sử dụng bị từ chối một cách vô điều kiện yêu cầu truy cập vào tất cả các tài nguyên mạng.
- *Access Challenge* - Yêu cầu thêm thông tin từ người sử dụng như một mật khẩu thứ cấp, PIN, mã thông báo, hoặc thẻ.
- *Access Accept* người sử dụng được cấp quyền truy cập. Một khi người dùng được xác thực, máy chủ RADIUS sẽ thường xuyên kiểm tra người dùng được phép sử dụng các dịch vụ mạng nào.

Mỗi gói tin trả lời trong ba gói tin trả lời mà RADIUS server phản hồi cho RADIUS client có thể bao gồm một thuộc tính Reply-Message cung cấp lý do về việc từ chối truy cập, nhắc nhở cho challenge, hoặc một tin nhắn chào mừng khi yêu cầu truy cập được chấp nhận.

9.3.4 Accounting

Khi người dùng được NAS cấp quyền truy cập vào mạng, một *Accounting Start* (một gói tin RADIUS Accounting Request có chứa một thuộc tính Acct-Status-Type với giá trị "start") được NAS gửi cho máy chủ RADIUS để báo hiệu người sử dụng bắt đầu truy cập mạng. Bản ghi "start" thường có thông tin nhận dạng của người dùng, địa chỉ mạng, điểm tập tin đính kèm và một định danh session duy nhất.

Theo định kỳ, bản ghi *Interim Update* (một gói tin RADIUS Accounting Request có chứa một thuộc tính Acct-Status-Type với giá trị "interim-update") có thể được NAS gửi đến máy chủ RADIUS, để cập nhật trạng thái của một phiên đang hoạt động. Bản ghi "Interim" thường truyền tải thời gian làm việc hiện tại của phiên và thông tin về cách sử dụng dữ liệu hiện tại.



Hình 9.10: Sơ đồ minh họa quá trình chứng thực trong accounting

Cuối cùng, khi truy cập mạng của người sử dụng được đóng lại, NAS gửi một gói tin *Accounting Stop* (một gói tin RADIUS Accounting Request có chứa một thuộc tính Acct-Status-Type với giá trị "stop") đến máy chủ RADIUS, cung cấp thông tin cuối cùng về thời gian sử dụng, các gói tin, dữ liệu được chuyển giao, lý do ngắt kết nối và các thông tin khác liên quan đến truy cập mạng của người dùng.

Thông thường, client sẽ gửi các gói tin Accounting-Request cho đến khi nó nhận được một gói tin xác nhận Accounting-Response bằng cách gửi các gói tin Accounting-Request theo những khoảng thời gian.

Mục đích chính của dữ liệu này là người sử dụng có thể được lập hóa đơn cho phù hợp, dữ liệu cũng thường được sử dụng cho mục đích thống kê và giám sát tổng quát mạng.

TÓM TẮT

Bài học cung cấp kiến thức về mô hình kiểm soát truy cập mạng an toàn AAA và các phần mềm ứng dụng hiện có TACACS+, RADIUS.

AAA cung cấp việc xác thực (authentication) người dùng nhằm bảo đảm có thể nhận dạng đúng người dùng. Một khi đã nhận dạng người dùng, ta có thể giới hạn thẩm quyền (authorization) mà người dùng có thể làm. Khi người dùng sử dụng mạng, ta cũng có thể giám sát tất cả những gì mà họ làm AAA có thể dùng để tập hợp thông tin từ nhiều thiết bị trên mạng.

TACACS+ (Terminal Access controller Access- Control System Plus) là một giao thức độc quyền của Cisco, được thiết kế dùng trong kiến trúc AAA cho việc chứng thực, ủy quyền và kế toán trong môi trường mạng. Ba hoạt động có thể được thực hiện trong suốt quá trình hoạt động của TACACS+: Hoạt động đầu tiên được thực hiện là xác thực để nhận diện người dùng. Hoạt động thứ 2 là ủy quyền và có thể chỉ thực hiện một lần khi user đã được nhận dạng hoàn tất. Bởi vậy, người dùng phải được chứng minh là hợp lệ trước khi ủy quyền cho họ. Hoạt động thứ 3 là kiểm toán. Quá trình kiểm toán theo dõi dấu vết của các hành động mà người dùng đã thực hiện khi ở trong hệ thống.

Radius (Remote Access Dial In User Service) là một giao thức mạng cung cấp việc quản lý xác thực tập trung, ủy quyền, và kế toán (AAA) để cho các máy tính kết nối vào mạng và sử dụng dịch vụ mạng.

CÂU HỎI ÔN TẬP

Câu 1: AAA là gì? Giải thích ý nghĩa từng thành phần dịch vụ hỗ trợ của AAA?

Câu 2: TACCAS+ là gì? Được ứng dụng ở đâu? Trình bày các dịch vụ TACCAS+ hỗ trợ?

Câu 3: Trình bày cấu trúc gói tin TACCAS? Ý nghĩa các thành phần của nó?

Câu 4: Giải thích tính bảo mật của quá trình xác thực, chứng thực và tính cước của TACCAS+?

Câu 5: RADIUS là gì? Trình bày các dịch vụ bảo mật RADIUS hỗ trợ?

Câu 6: Trình bày cấu trúc gói tin RADIUS, ý nghĩa từng trường của nó?

Câu 7: Giải thích tính bảo mật của quá trình xác thực của RADIUS?

PHỤ LỤC

Bài thực hành số 1



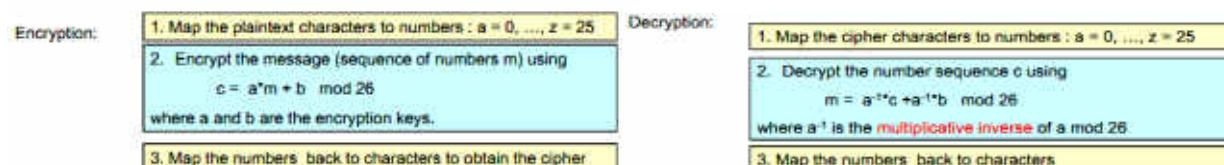
Bài 1: Viết chương trình mã hóa và giải mã văn bản với thuật toán mã hóa Ceasar.

Chương trình có thể thực hiện các chức năng sau:

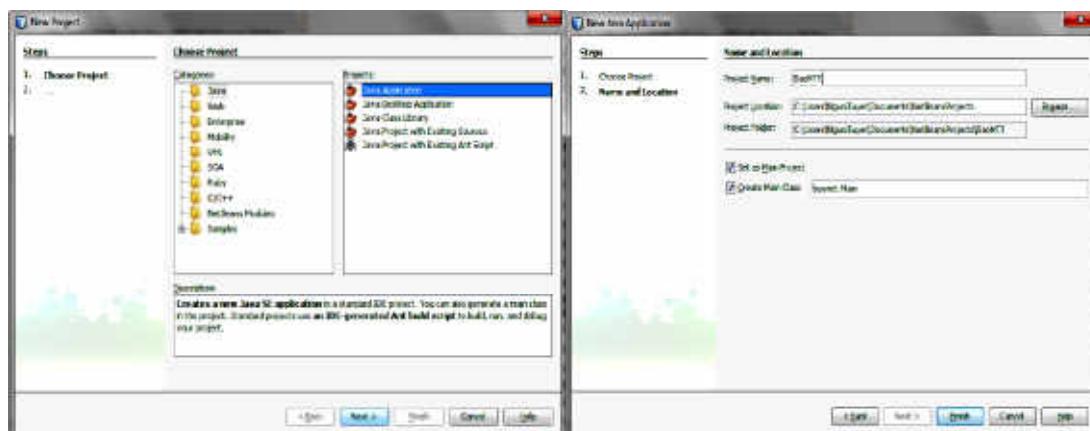
- Cho phép nhập văn bản vào hệ thống.
- Cho phép nhập khóa bảo vệ văn bản.
- Cho phép ghi File và mở File.

Hướng dẫn mã dịch chuyển Caesar:

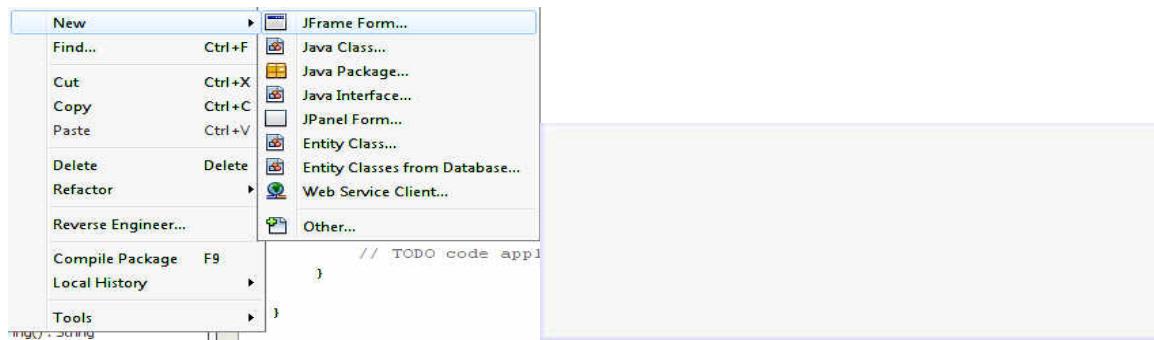
- Ta lần lượt đánh chỉ số cho các chữ cái bắt đầu từ 0.
- Gọi k là 1 số nguyên từ 0 ->25 được gọi là khóa.
- Hàm mã hóa: $E(p,k)=(p+k) \bmod 26$ với p là chỉ số của ký tự cần mã hóa.
- Hàm giải mã: $D(c,k)=|c-k| \bmod 26$ với c là chỉ số của ký tự cần giải mã.



Bước 1: Tạo project mới: File → New Project



Bước 2: Tạo mới jFrame Form thiết kế:



Bước 3: Thiết kế Form



Bước 4: Viết hàm xử lý sự kiện

a. Hàm xử lý sự kiện Encrypt

```
private void btntmahoaActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    String plainText=txtvanban.getText();
    String khoa=txtkhoa.getText();
    int key;
    //chuyển kiểu chuỗi thành kiểu int
    key=Integer.parseInt(khoa);
    if(key < 0)
    {
        key = 26 - (-key % 26);
    }
    String result = "";
    for(int i=0 ; i<plainText.length();i++)
    {
        char ch = mahoa(plainText.charAt(i),key);
        result = result + ch;
    }
    txtmhoa.setText(result);
}
```

b. Hàm xử lý sự kiện Ghi File

```
private void bntGhiFileActionPerformed(java.awt.event.ActionEvent evt) {  
    try {  
        // TODO add your handling code here:  
        BufferedWriter bw = null;  
        //Nơi lưu dữ liệu  
        String fileName = "D:\\Dulieu.txt";  
        //luu van ban  
        String s = txtmahoa.getText();  
        // Ghi dữ Liệu S vào tạo tin fileName  
        bw = new BufferedWriter(new FileWriter(fileName));  
        bw.write(s);  
        bw.close();  
        JOptionPane.showMessageDialog(null, "Đã Ghi File Thành Công !!!");  
    } catch (IOException ex) {  
        Logger.getLogger(Ceasar.class.getName()).log(Level.SEVERE, null, ex);  
    }  
  
}
```

c. Hàm xử lý sự kiện Dencypt

```
private void bntgiaimaActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
    String cipherText=txtvanban.getText();  
    String Khoa=txtkhoa.getText();  
    int key;  
    key=Integer.parseInt(khoa);  
    //key từ 0->25 là 1 số nguyên  
    if(key < 0)  
    {  
        key = 26 - (-key % 26);  
    }  
    String result = "";  
    //mã hóa các ký tự của văn bản  
    for(int i=0 ; i<cipherText.length();i++)  
    {  
        char ch = mahoa(cipherText.charAt(i),key);  
        result = result + ch;  
    }  
    //hiển thị văn bản đã mã hóa  
    txtmahoa.setText(result);  
}
```

```
//hàm mã hóa : mahoa(ch,k)=(ch+k)mod26 với ch là chỉ số của ký tự cần mã hóa
private char mahoa(char ch, int key)
{
    if( Character.isLetter(ch)){
        ch = (char) ('A' +(Character.toUpperCase(ch)-'A'+ key)%26);
    }
    return ch;
}
/**
```

d. Hàm xử lý sự kiện Mở File

```
private void bntMoFileActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        // TODO add your handling code here:
        // Lớp BufferedReader kế thừa từ lớp Reader thuộc nhóm Input
        // Nhóm này là 1 trong 2 loại chính của Các luồng Ký Tự
        // Có chức năng đọc dữ liệu dạng ký tự
        BufferedReader br = null;
        String fileName = "D:\\\\Dulieu.txt";
        br = new BufferedReader(new FileReader(fileName));
        //nhận dữ liệu
        StringBuffer sb = new StringBuffer();
        JOptionPane.showMessageDialog(null, "Đã mở File Thành Công !!!");
        //Đọc mỗi lần tối đa 5 ký tự
        char[] ca = new char[5];
        while (br.ready()) {
            int len = br.read(ca);
            sb.append(ca, 0, len);
        }
        br.close();

        //xuat chuoi
        System.out.println("Du Lieu la :" + " " + sb);
        String chuoi = sb.toString();
        // Hiển thị lên Form
        txtvanban.setText(chuoi);
    } catch (IOException ex) {
        Logger.getLogger(Ceasar.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

Bài 2: Viết chương trình mã hóa và giải mã văn bản với thuật toán mã hóa Vigenere. Chương trình có thể thực hiện các chức năng sau:

- Cho phép nhập văn bản vào hệ thống.
- Cho phép nhập khóa bảo vệ văn bản.
- Cho phép mở File và Ghi File.

Hướng Dẫn: Mật mã Vigenere còn gọi là mật mã nhiều bảng mã. Ưu điểm của mã này là việc sử dụng 26 bảng mã khác nhau. Do đó mà không bị phá trong một thời gian dài. Ngoài ra mã này còn hỗ trợ việc sử dụng từ khóa vô cùng tiện lợi.

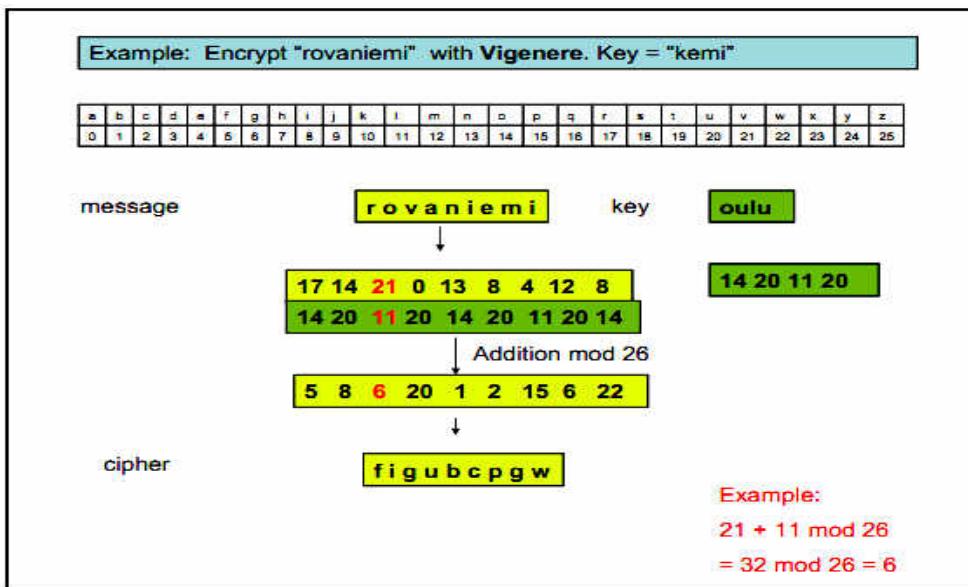
Thuật toán:

- Khóa K là một bộ gồm nhiều khóa con
 - $K = (k_1, k_2, \dots, k_m)$
- **Mã hóa:**
 - $e_K(x_1, x_2, \dots, x_m) = (x_1 + k_1, x_2 + k_2, \dots, x_m + k_m)$

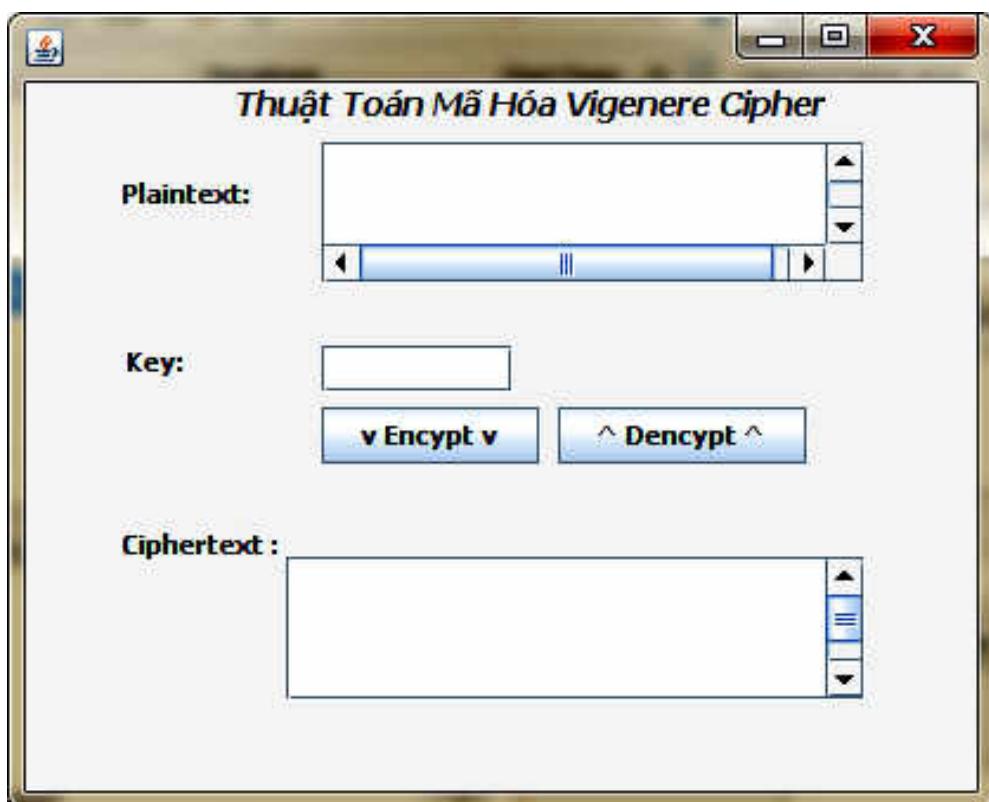
- **Giải mã:**
 - $d_K(y_1, y_2, \dots, y_m) = (y_1 - k_1, y_2 - k_2, \dots, y_m - k_m)$
 - (cộng, trừ theo modulo 26)
- => "MÃ KHỐI" (block cipher)

Ví dụ:

- $\{A, B, C, \dots, X, Y, Z\} = \mathbf{Z}_{26} = \{0, 1, \dots, 25\}$.
- $K = (2, 8, 15, 7, 4, 17)$ ("CIPHER").
- $p = "thiscryptosy"$.
- $c = "VPXZGIA\xivWP"$



Bước 1: Thiết Kế Form:



Bước 2: Viết hàm xử lý sự kiện

a. Hàm xử lý sự kiện Encrypt

```
private void bntmahoaActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
    int tableRowSize = 26;  
    int tableColumnSize = 26;  
    int vignereTable[][] = new int[26][26];  
  
    for (int rows = 0; rows < tableRowSize; rows++) {  
  
        for (int columns = 0; columns < tableColumnSize; columns++) {  
  
            vignereTable[rows][columns] = (rows + columns) % 26;  
        }  
    }  
    // nhập văn bản cần mã hóa  
    System.out.println("Enter the plaintext");  
    String plainText= txtvanban.getText();  
    // chuyển văn bản sang chữ hoa  
    plainText = plainText.toUpperCase();  
    // nhập khóa K  
    System.out.print("Enter the key: ");
```

```
String key = txtkhoa.getText();
key = key.toUpperCase();
String cipherText = "";
int keyIndex = 0;
for (int ptextIndex = 0; ptextIndex < plainText.length(); ptextIndex++)
{
    char pChar = plainText.charAt(ptextIndex);
    int asciiVal = (int) pChar;
    if (pChar == ' '){
        cipherText += pChar;
        continue;
    }
    if (asciiVal < 65 || asciiVal > 90){
        cipherText += pChar;
        continue;
    }
    int basicPlainTextValue = ((int) pChar) - 65;
    char kChar = key.charAt(keyIndex);
    int basicKeyValue = ((int) kChar ) - 65;
    int tableEntry = vignereTable[basicPlainTextValue] [basicKeyValue];

        char cChar = (char) (tableEntry + 65);
        cipherText += cChar;
        keyIndex++;
        if (keyIndex == key.length())
            keyIndex = 0;
    }
    System.out.println(" cipher text is "+cipherText);
    txtmahoa.setText(cipherText.toString().toUpperCase());
}
```

b. Hàm xử lý sự kiện Dencypt

```
// TODO add your handling code here:  
int tableRowSize = 26;  
int tableColumnSize = 26;  
int vignereTable[][] = new int[26][26];  
String cipherText=txtmahoa.getText();  
String plainText="";  
for (int rows = 0; rows < tableRowSize; rows++) {  
  
    for (int columns = 0; columns < tableColumnSize; columns++) {  
        vignereTable[rows][columns] = (rows + columns) % 26;  
    }  
}  
  
System.out.println("Enter the cipher text");  
cipherText = cipherText.toUpperCase();  
System.out.print("Enter the key: ");  
String key = txtkhoa.getText();  
key = key.toUpperCase();  
int keyIndex = 0;
```

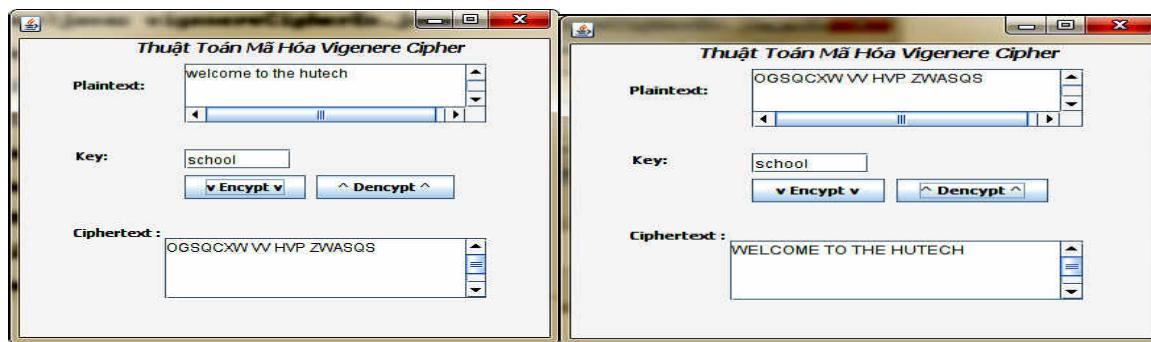
```
for (int ctextIndex = 0; ctextIndex < cipherText.length(); ctextIndex++) {
    char cChar = cipherText.charAt(ctextIndex);
    int asciiVal = (int) cChar;
    if (cChar == ' '){
        plainText += cChar;
        continue;
    }
    if (asciiVal < 65 || asciiVal > 90){
        plainText += cChar;
        continue;
    }
    int basiccipherTextValue = ((int) cChar) - 65;
    char kChar = key.charAt(keyIndex);
    int basicKeyValue = ((int) kChar ) - 65;
    for (int pIndex = 0; pIndex < tableColumnSize; pIndex++){
        if (vignereTable[basicKeyValue][pIndex] == basiccipherTextValue){
            char potcChar = (char) (vignereTable[basicKeyValue][pIndex] + 65);
            char potpChar = (char) (pIndex + 65);
            plainText += potpChar;
        }
    }
}
keyIndex++;

if (keyIndex == key.length())
    keyIndex = 0;

}

System.out.println(" plain text is "+plainText);
txtvanban.setText(cipherText.toString().toUpperCase());
txtmahoa.setText(plainText.toString().toUpperCase());
```

Kết quả:

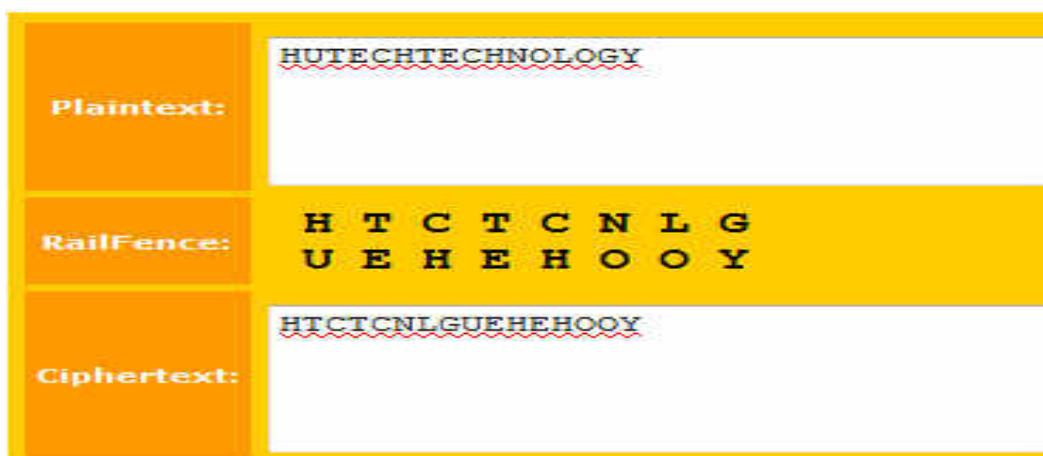


Bài 3: Viết chương trình mã hóa và giải mã văn bản với thuật toán mã hóa Rail Fence. Chương trình có thể thực hiện các chức năng sau:

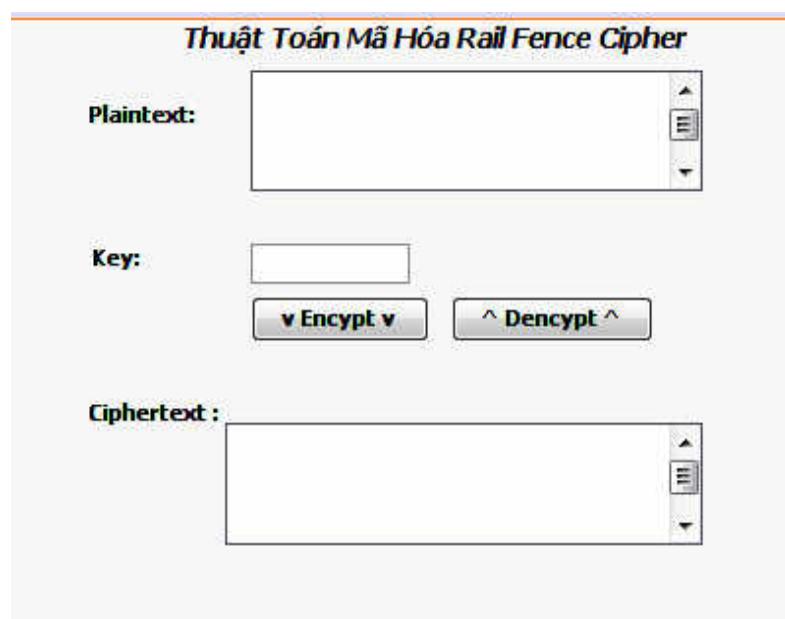
- Cho phép nhập văn bản vào hệ thống.
- Cho phép nhập khóa bảo vệ văn bản.
- Cho phép mở File và Ghi File.

Hướng Dẫn: Mã Rail Fence còn được gọi là mã zig zag là một hình thức của mã chuyển vị:

- Thông điệp được viết lần lượt từ trái qua phải trên các cột (rail) của một hàng dào tưởng tượng theo đường chéo từ trên xuống dưới.
- Theo đường chéo từ dưới lên khi đạt tới cột thấp nhất.
- Và khi đạt tới cột cao nhất, lại viết theo đường chéo từ trên xuống. Cứ lặp đi lặp lại như thế nào cho đến khi viết hết toàn bộ nội dung của thông điệp.
- Ví dụ: mã hóa chuỗi HUTECH TECHNOLOGY với khóa là 2.



Bước 1: Thiết Kế Form:



Bước 2: Viết hàm xử lý sự kiện

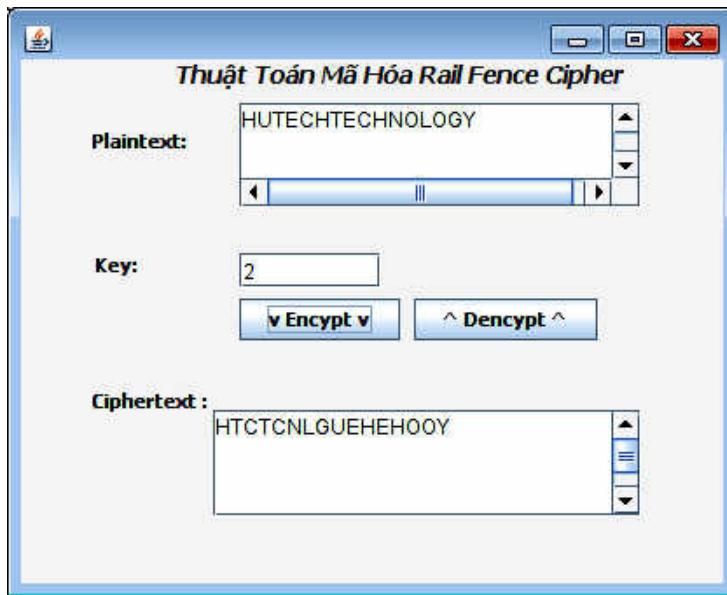
a. Hàm xử lý sự kiện Encrypt

```
private void bntmahoaActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    String plainText=txtvanban.getText();
    int i=0,j,l,r,s,c=0,t=0;
    String cipherText="";
    System.out.println("Enter the Rails:");
    String khoa=txtkhoa.getText();
    r=Integer.parseInt(khoa);
    // chuyển kiểu chuỗi sang kiểu int
    l=plainText.length()/r;
    s=2*l;
    char ct[][]=new char [r][s];
    while(i<r)
    {
        ct[i][c]=plainText.charAt(t);
        t++;
        if(i==r-1&&t<=plainText.length()-1)
        {
            c=c+1;
            for(j=r-2;j>=0;j--)
        }
    }
}
```

```
        {
            ct[j][c]=plainText.charAt(t);
            t++;
            if(j==0&&t<=plainText.length()-1)
            {
                c++;
                i=0;
            }
        }
    }
    i=i+1;
}
for(i=0;i<r;i++)
{
    for(j=0;j<ct[i].length;j++)
    {
        char d=ct[i][j];
        if(d=='\0')
        {
            continue;
        }
        else
        {
            cipherText=cipherText+ct[i][j];
        }
    }
    System.out.println("The Cipher text is:\n"+cipherText);
    System.out.println(cipherText.length());
    txtmahoa.setText(cipherText.toString().toString());
}
```

- b. Hàm xử lý sự kiện Dencypt: (Sinh viên tự làm)

Bước 3: Kiểm Tra



Bài 4: Viết chương trình mã hóa và giải mã văn bản với thuật toán mã hóa PlayFair. Chương trình có thể thực hiện các chức năng sau:

- Cho phép nhập văn bản vào hệ thống.
- Cho phép nhập khóa bảo vệ văn bản.
- Cho phép mở File và Ghi File.

Hướng dẫn:

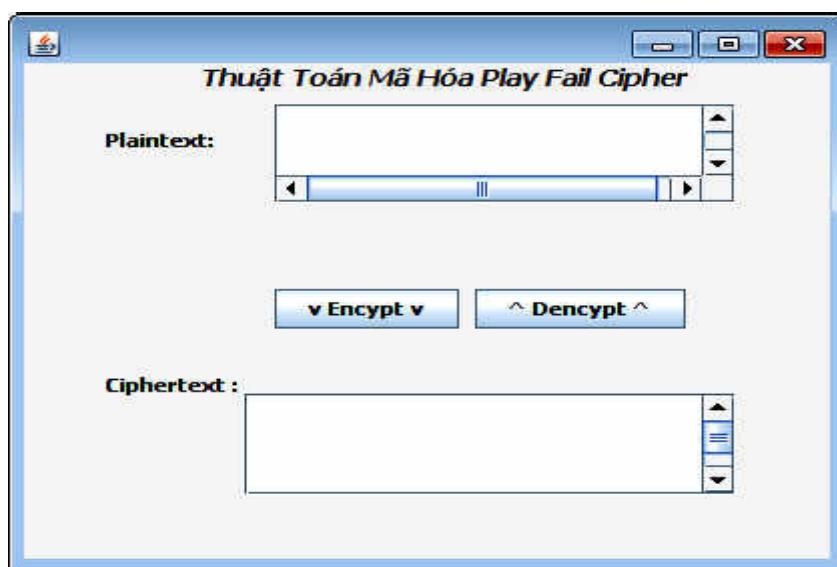
- ❖ **Phương pháp** là lập ma trận 5x5 dựa trên từ khóa cho trước và các ký tự trên bảng chữ cái:
 - Trước hết viết các chữ của từ khoá vào các hàng của ma trận bắt từ hàng thứ nhất.
 - Nếu ma trận còn trống, viết các chữ khác trên bảng chữ cái chưa được sử dụng vào các ô còn lại. Có thể viết theo một trình tự qui ước trước, chẳng hạn từ đầu bảng chữ cái cho đến cuối.
 - Vì có 26 chữ cái tiếng Anh, nên thiếu một ô. Thông thường ta dồn hai chữ nào đó vào một ô chung, chẳng hạn I và J.
 - Giả sử sử dụng từ khoá MORNACHY. Lập ma trận khoá Playfair tương ứng như sau:

M	O	N	A	R
C	H	Y	B	D
E	F	G	I,J	K
L	P	Q	S	T
U	V	W	X	Z

❖ Quy tắc mã hóa và giải mã

- Chia bản rõ thành từng cặp chữ. Nếu một cặp nào đó có hai chữ như nhau, thì ta chèn thêm một chữ lọc chẵng hạn X. Ví dụ, trước khi mã “balloon” biến đổi thành “ba lx lo on”.
- Nếu cả hai chữ trong cặp đều rơi vào cùng một hàng, thì mã mỗi chữ bằng chữ ở phía bên phải nó trong cùng hàng của ma trận khóa (cuộn vòng quanh từ cuối về đầu), chẵng hạn “ar” biến đổi thành “RM”.
- Nếu cả hai chữ trong cặp đều rơi vào cùng một cột, thì mã mỗi chữ bằng chữ ở phía bên dưới nó trong cùng cột của ma trận khóa (cuộn vòng quanh từ cuối về đầu), chẵng hạn “mu” biến đổi thành “CM”.
- Trong các trường hợp khác, mỗi chữ trong cặp được mã bởi chữ cùng hàng với nó và cùng cột với chữ cùng cặp với nó trong ma trận khóa. Chẳng hạn, “hs” mã thành “BP”, và “ea” mã thành “IM” hoặc “JM”.

Bước 1: Thiết Kế Form:



Bước 2: Viết hàm xử lý sự kiện

a. Hàm xử lý sự kiện Encrypt

```

private void bntmahoaActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
}

String plainText="",plainTxt, cipherText="";
String playFairMatrix[][]= {
    {"M", "O", "N", "A", "R"}, 
    {"C", "H", "Y", "B", "D"}, 
    {"E", "F", "G", "I", "K"}, 
    {"L", "P", "Q", "S", "T"}, 
    {"U", "V", "W", "X", "Z"}, 
};

System.out.println("Enter a plaintext:");
plainTxt = txtvanban.getText();
String temp="";
String arr[]=plainTxt.split(" ");
for(int j=0;j<arr.length;j++)
{
    temp=arr[j];
    if(temp.length()%2!=0)

```

```
        {
            temp=temp+"x";
        }
        plainText=plainText+ temp+" ";
    }

    for(int i=0; i<plainText.length(); i+=2)
    {
        char c = plainText.charAt(i);
        char d=plainText.charAt(i);
        if(i+1<plainText.length())
        {
            d = plainText.charAt(i+1);
        }

        String val = String.valueOf(c);
        String vald = String.valueOf(d);
        String index1,index2;
        if(val.equals(" "))
        {
            cipherText=cipherText+" ";
            i--;
            continue;
        }
        else
        {
            if(val.equalsIgnoreCase("J"))//to insert the index position for char J
            {
                index1 = findIndex(playFairMatrix, String.valueOf("I"));
            }
        }
        else
        {
            index1 = findIndex(playFairMatrix, String.valueOf(plainText.charAt(i)));
        }
        if(vald.equalsIgnoreCase("J"))//to insert the index position for char J
        {
            index2 = findIndex(playFairMatrix, String.valueOf("I"));

        }
        else
        {
            index2 = findIndex(playFairMatrix, String.valueOf(plainText.charAt(i+1)));
        }

        if(index1.charAt(0) == index2.charAt(0))//row same
        {
            int m = Integer.parseInt(String.valueOf(index1.charAt(1))); //column of index 1
            if(m>=0 && m<=3)
            {
                cipherText=cipherText+playFairMatrix[index1.charAt(0)][m];
            }
            else
            {
                cipherText=cipherText+playFairMatrix[index1.charAt(0)][m-4];
            }
        }
        else
        {
            cipherText=cipherText+playFairMatrix[index1.charAt(0)][index2.charAt(0)];
        }
    }
}
```

```

int n = Integer.parseInt(String.valueOf(index2.charAt(1))); //column of index 2
int o = Integer.parseInt(String.valueOf(index1.charAt(0))); //row of index 1
int p = Integer.parseInt(String.valueOf(index2.charAt(0))); //row of index 2
if(m==4)
{
    m=-1;
}
if(n==4)
{
    n=-1;
}
cipherText=cipherText+playFairMatrix[o][m+1];
cipherText=cipherText+playFairMatrix[p][n+1];
}
else if(index1.charAt(1) == index2.charAt(1)) //column same
{
    int o = Integer.parseInt(String.valueOf(index1.charAt(0))); //row of index 1
    int m = Integer.parseInt(String.valueOf(index1.charAt(1))); //column of index 1
    int p = Integer.parseInt(String.valueOf(index2.charAt(0))); //row of index 2
    int n = Integer.parseInt(String.valueOf(index2.charAt(1))); //column of index 2
    if(p>3)
    {
        p=-1;
    }
    if(o>3)
    {
        o=-1;
    }
    cipherText=cipherText+playFairMatrix[o+1][m];
    cipherText=cipherText+playFairMatrix[p+1][n];
}
else
{
    int o = Integer.parseInt(String.valueOf(index1.charAt(0))); //row of index 1
    int m = Integer.parseInt(String.valueOf(index1.charAt(1))); //column of index 1
    int p = Integer.parseInt(String.valueOf(index2.charAt(0))); //row of index 2
    int n = Integer.parseInt(String.valueOf(index2.charAt(1))); //column of index 2
    cipherText=cipherText+playFairMatrix[o][n];
    cipherText=cipherText+playFairMatrix[p][m];
}

}
System.out.println("The cipher text of the above plain text is:");
System.out.println(cipherText);
txtmahoa.setText(cipherText.toString().toUpperCase());
}
}

```

b. Hàm xử lý sự kiện Dencypt

```
private void bntgiaimaActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    String plainText="",cipherText="";
    String playFairMatrix[][] = {
        {"M", "O", "N", "A", "R"}, 
        {"C", "H", "Y", "B", "D"}, 
        {"E", "F", "G", "I", "K"}, 
        {"L", "P", "Q", "S", "T"}, 
        {"U", "V", "W", "X", "Z"}};
    System.out.println("Enter a Ciphertext:");
    cipherText = txtmahoa.getText();
    txtmahoa.setText(cipherText.toString().toUpperCase());
    for(int i=0; i<cipherText.length(); i+=2)
    {
        char c = cipherText.charAt(i);
        char d=cipherText.charAt(i);
        if(i+1<cipherText.length())
        {
            d = cipherText.charAt(i+1);
        }
        String val = String.valueOf(c);
        String vald = String.valueOf(d);
        String index1,index2;
        if(val.equals(" "))
        {
            plainText=plainText+" ";
            i--;
            continue;
        }
        else
        if(val.equalsIgnoreCase("J"))//to insert the index position for char J
        {
            index1 = findIndex(playFairMatrix, String.valueOf("I"));
        }
        else
        {
            index1 = findIndex(playFairMatrix, String.valueOf(cipherText.charAt(i)));
        }
        if(vald.equalsIgnoreCase("J"))//to insert the index position for char J
        {
            index2 = findIndex(playFairMatrix, String.valueOf("I"));
        }
        else
        {
            index2 = findIndex(playFairMatrix, String.valueOf(cipherText.charAt(i+1)));
        }
        if(index1.charAt(0) == index2.charAt(0))//row same
        {
            int m = Integer.parseInt(String.valueOf(index1.charAt(1))); //column of index 1
            int n = Integer.parseInt(String.valueOf(index2.charAt(1))); //column of index 2
            plainText=plainText+playFairMatrix[m][n];
        }
    }
}
```

```

        int o = Integer.parseInt(String.valueOf(index1.charAt(0))); //row of index 1
        int p = Integer.parseInt(String.valueOf(index2.charAt(0))); //row of index 2
        if(m==0)
        {
            m=5;
        }
        if(n==0)
        {
            n=5;
        }
        plainText=plainText+playFairMatrix[o] [m-1];
        plainText=plainText+playFairMatrix[p] [n-1];
    }
    else if(index1.charAt(1) == index2.charAt(1))//column same
    {
        int o = Integer.parseInt(String.valueOf(index1.charAt(0))); //row of index 1
        int m = Integer.parseInt(String.valueOf(index1.charAt(1))); //column of index 1
        int p = Integer.parseInt(String.valueOf(index2.charAt(0))); //row of index 2
        int n = Integer.parseInt(String.valueOf(index2.charAt(1))); //column of index 2
        if(p==0)
        {
            p=5;
        }
        if(o==0)
        {
            o=5;
        }
        plainText=plainText+playFairMatrix[o-1] [m];
        plainText=plainText+playFairMatrix[p-1] [n];
    }
    else
    {
        int o = Integer.parseInt(String.valueOf(index1.charAt(0))); //row of index 1
        int m = Integer.parseInt(String.valueOf(index1.charAt(1))); //column of index 1
        int p = Integer.parseInt(String.valueOf(index2.charAt(0))); //row of index 2
        int n = Integer.parseInt(String.valueOf(index2.charAt(1))); //column of index 2
        plainText=plainText+playFairMatrix[o] [n];
        plainText=plainText+playFairMatrix[p] [m];
    }
}

System.out.println("plaintext text of the above cipher text is:");
System.out.println(plainText);
txtmahoa.setText(plainText.toString().toUpperCase());
}

```

c. Hàm FindIndex

```
public static String findIndex(String[][] arr, String test)
{
    String index = "";
    for(int i=0; i<arr.length; i++)
    {
        for(int j=0; j<arr[i].length; j++)
        {
            if(test.equalsIgnoreCase(arr[i][j]))
            {
                index = String.valueOf(i) + String.valueOf(j);
                return index;
            }
        }
    }
    return null;
}
```

Bước 3: Kết quả



Bài 5: Viết chương trình mã hóa và giải mã văn bản với thuật toán mã hóa Transposition cipher. Chương trình có thể thực hiện các chức năng sau:

- Cho phép nhập văn bản vào hệ thống.
- Cho phép nhập khóa bảo vệ văn bản.
- Cho phép mở File và Ghi File.

Hướng dẫn: Hệ mã hóa đổi chỗ (Transposition Cipher)

Là hệ mã hóa trong đó các kí tự của bản gốc được giữ nguyên, nhưng vị trí bị thay đổi.

Đảo ngược toàn bộ plaintext: nghĩa là bản gốc được viết theo thứ tự ngược lại từ sau ra trước.

Ví dụ Plaintext: SECURE EMAIL

Bản mã: LIAMEERUCES

Mã hóa theo mẫu hình học: bản gốc được sắp xếp lại theo một mẫu hình học nào đó, thường là một mảng hoặc ma trận hai chiều.

Ví dụ: bản gốc ban đầu là BAO MAT

Ví dụ mã hóa theo mẫu hình học.

Cột 1	Cột 2	Cột 3
B	A	O
M	A	T

Nếu lấy các cột theo thứ tự 2, 3, 1. Bản mã sẽ là AAOTBM

Đổi chỗ cột: đổi chỗ các kí tự trong plaintext thành dạng hình chữ nhật theo cột.

Ví dụ: Bản gốc BAO MAT THU DIEN TU

Bản gốc được chuyển thành ma trận 3x5 như sau:

Bảng ví dụ mã hóa bằng phương pháp đổi chỗ cột

Cột 1	Cột 2	Cột 3	Cột 4	Cột 5
B	M	T	D	N
A	A	H	I	T
O	T	U	E	U

Vì có 5 cột nên chúng có thể được sắp lại theo $5! = 120$ cách khác nhau

Nếu ta chuyển vị các cột theo thứ tự 3, 5, 2, 4, 1 rồi lấy các kí tự theo hàng ngang ta sẽ thu được bản mã: TNMDBHTAIAUUETO.

Hoán vị các kí tự của bản gốc theo chu kỳ cố định d: Nếu hàm f là hoán vị của một khối gồm d kí tự thì khóa mã hóa được biểu diễn bởi $K(d, f)$

Ví dụ: với $d = 5$, f hoán vị của dãy 12345 thành 35142

Bảng Hoán vị các kí tự của bản gốc theo chu kỳ cố định d

Vị trí ban đầu	Vị trí hoán vị	Nội dung mã hóa	Mã hóa
1	3	G	O
2	5	R	P
3	1	O	G
4	4	U	U
5	2	P	R

Theo bảng trên bản gốc ban đầu được mã hóa thành OPGUP.

Bước 1: Thiết Kế Form:

Thuật Toán Mã Hóa Transposition Cipher

Plaintext:

v Encrypt v **^ Dencypt ^**

Ciphertext:

Bước 2: Viết hàm xử lý sự kiện

a. Hàm xử lý sự kiện Encrypt

```

private void bntmahoaActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    char letter[]=new char [25];
    String plainText,cipherText="",jpt="ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    int c=0,l=1,v=0,i=0,t=1,vc=0,r=0;
    char mat[][]=new char[50][50];
    int key[]={4,3,1,2,5,6,7};
    char z;
    System.out.println("Enter the plain text:");
    plainText=txtvanban.getText();
    for(i=0;i<c+1;i++)
    {
        for(int j=0;j<key.length;j++)
        {
            if(c<plainText.length())
            {
                char a=plainText.charAt(c);
                mat[i][j]=a;
                c++;
                v=i+1; //for row count
            }
            if(c==plainText.length()&&j!=vc)//to last string X Y Z..
            {
                r=key.length-j;
                if(r<=jpt.length())
                {
                    z=jpt.charAt(jpt.length()-r);
                    mat[i][j+1]= z;
                }
            }
        }
    }
    //them doan nay
    t=2;
    if(c==plainText.length()&&r==0)// thoat vong lap
    {
        i=c+2;
    }
}
c=1;

```

```

        while(c<=key.length)

    {
        //het
        for(i=0;i<key.length;i++)
        {
            if(key[i]==c)//to match the sequence of key position
            {
                int k=0;
                for(l=0;l<v;l++)
                {
                    cipherText=cipherText+mat[k][i];
                    k++;//to increase the row
                }
            }
        }
        c++;
    }
    cipherText=cipherText.toUpperCase();
    System.out.println("\nThe Cipher Text is:\n"+cipherText);
    txtmahoa.setText(cipherText.toString().toUpperCase());
}

```

b. Hàm xử lý sự kiện Dencypt

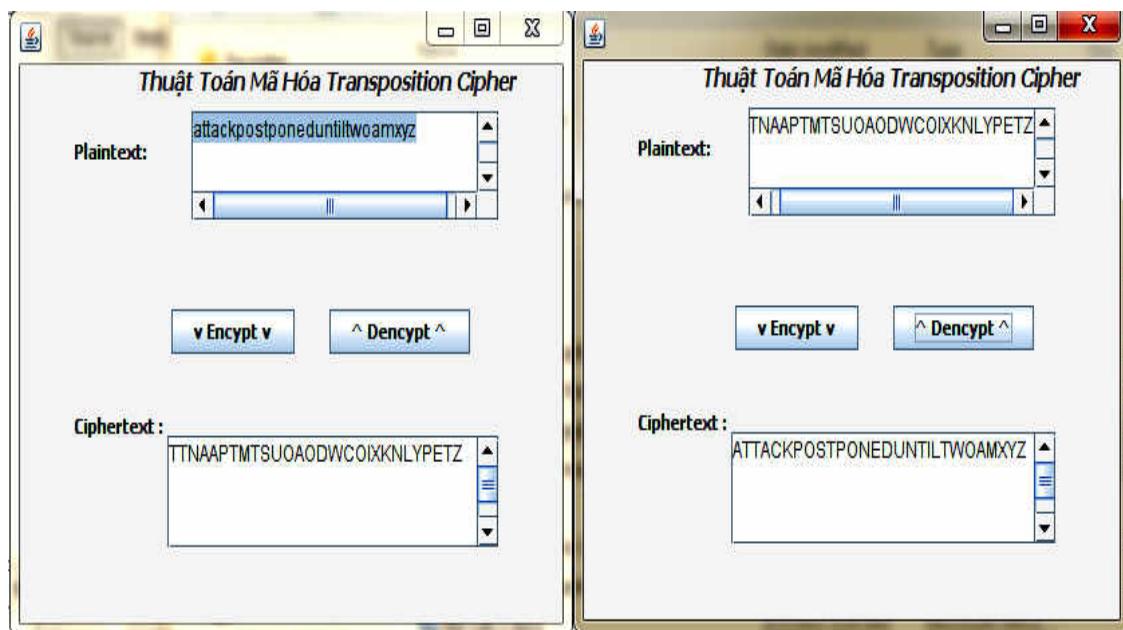
```

private void bntgiaimaActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    String plainText="",cipherText="",jpt="ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    int c=0,l=1,v=0,i=0,j=0,k=0;
    char mat[][]=new char[50][50];
    char pmat[][]=new char[50][50];
    int key[]={4,3,1,2,5,6,7};
    char z;
    System.out.println("Enter the Cipher text:");
    cipherText=txtmahoa.getText();
    txtvanban.setText(cipherText.toString().toUpperCase());
    v=(cipherText.length()/key.length);
    while(c<key.length)
    {
        for(j=0;j<key.length;j++)
        {
            if(key[j]==c+1)
            {
                for(i=0;i<v;i++)
                {
                    mat[i][j]=cipherText.charAt(k);
                    k++;
                }
            }
        }
        c++;
    }
}

```

```
        for(int p=0;p<4;p++)  
        {  
            for(int q=0;q<key.length;q++)  
            {  
                plainText=plainText+mat[p][q];  
            }  
        }  
        System.out.println("\nThe Plain text is:\n"+plainText);  
        txtmahoa.setText(plainText.toString().toUpperCase());  
  
    }  
}
```

Bước 3: Kết Quả



Bài thực hành số 2

Viết chương trình mã hóa và giải mã văn bản với thuật toán mã hóa DES.

Chương trình có thể thực hiện các chức năng sau:

- Cho phép nhập văn bản vào hệ thống.
- Cho phép nhập khóa bảo vệ văn bản.
- Cho phép ghi File và mở File.

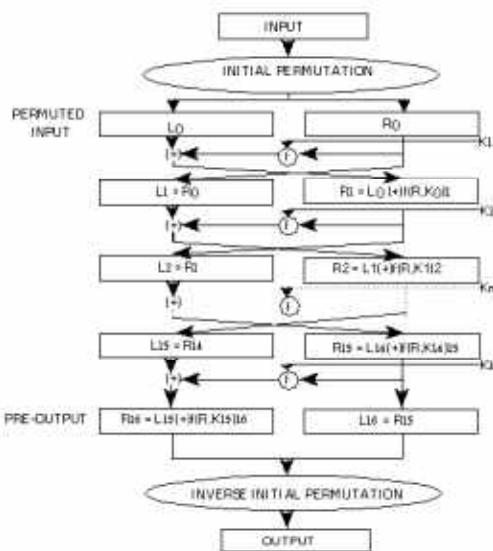
❖ **Hướng dẫn mã hóa DES:**

DES Là một hệ mật mã được sử dụng rộng rãi nhất trên thế giới. DES được IBM phát triển vào những năm 1970 và được xem như cải biên của hệ mật mã LUCIPHER. DES được chấp nhận bởi National Bureau of Standards, ngày nay gọi là NIST (National Institute of Standards and Technology). DES trở thành chuẩn mã hóa dữ liệu chính thức của chính phủ Hoa Kỳ vào năm 1977.

Mô tả thuật toán:

DES là thuật toán mã hóa khối (block cipher), mỗi khối dữ liệu có độ dài 64 bit. Một block bản gốc sau khi mã hóa tạo ra một block bản mã. Quá trình mã hóa và giải mã đều dùng chung một khóa.

Khóa có độ dài là 56 bit, cộng thêm 8 bit chẵn lẻ được sử dụng để kiểm soát lỗi. Các bit chẵn lẻ nằm ở các vị trí 8, 16, 24...64. tức là cứ 8 bit thì có một bit kiểm soát lỗi.

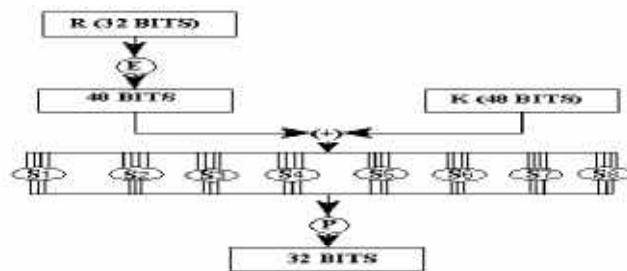


Hình 1: Sơ đồ hoạt động của DES

Theo sơ đồ hoạt động của DES như trên ta có thể thấy:

DES thực hiện trên từng block bản gốc. Sau khi thực hiện hoán vị khởi đầu (Initial Permutation – IP) khối dữ liệu được chia làm hai nửa trái và phải, mỗi nửa 32 bit. Quá trình được lặp lại qua 16 vòng, mỗi vòng là một hàm f. Sau 16 vòng lặp, hai nửa trái và phải được kết hợp lại và thực hiện hoán vị cuối cùng (hoán vị ngược – Inverse Initial Permutation) để kết thúc thuật toán.

Mỗi vòng của DES được thực hiện theo các bước sau:



Hình 2: Một vòng hoạt động của DES

Bước 1: Sử dụng hoán vị khởi đầu để thay đổi thứ tự các bit.

Bảng P3.B10: Bảng hoán vị khởi đầu: (hoán vị bit 1 thành bit 58, bit 2 thành bit 50....)

58	50	42	34	26	18	10	2	60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6	64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1	59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5	63	55	47	39	31	23	15	7

Bước 2: Bản gốc được chia làm hai nửa trái và phải, mỗi nửa 32 bit.

Bước 3: Ban đầu khóa 64 bit được bỏ đi 8 bit kiểm soát lỗi. Sự loại bỏ được thực hiện theo bảng sau:

Bảng 1:Bảng loại bỏ 8 bit kiểm soát lỗi

57	49	41	33	25	17	9	1	58	50	42	34	26	18
10	2	59	51	43	35	27	19	11	3	60	52	44	36
63	55	47	39	31	23	15	7	62	54	46	38	30	22
14	6	61	53	45	37	29	21	13	5	28	20	12	4

Sau đó khóa được chia làm hai nửa, mỗi nửa 28 bit.

Bước 4: Các nửa của khóa lần lượt được dịch trái (số bit dịch là 1 hay 2 tùy theo vòng thực hiện).

Bảng 2: Bảng dịch:

Vòng	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Số bit dịch	1	1	2	2	2	2	2	2	2	1	2	2	2	2	2	1

Các nửa được ghép lại với nhau, hoán vị và chọn ra 48 bit bằng cách đổi chỗ các bit theo bảng hoán vị nén - compression permutation (hay còn gọi là hoán vị lựa chọn- permuted choice)

Bảng 3: Bảng hoán vị nén: (bit ở vị trí 14 của khóa dịch được chuyển tới vị trí 1 của đầu ra, bit ở vị trí 17 của khóa dịch được chuyển tới vị trí 2 của đầu ra,..., bit thứ 18 bị loại bỏ...)

14	17	11	24	1	5	3	28	15	6	21	10
23	19	12	4	26	8	16	7	27	20	13	2
41	52	31	37	47	55	30	40	51	45	33	48
44	49	39	56	34	53	46	42	50	36	29	32

Bước 5: 32 bit của bản gốc bên phải được mở rộng thành 48 bit để XOR với 48 bit khóa. Khối bit này lại thực hiện hoán vị một lần nữa, thay đổi thứ tự các bit bằng cách lặp lại một số bit (hoán vị mở rộng - Expansion Permutation).

Bảng 4: Bảng hoán vị mở rộng – hộp E (bit ở vị trí thứ 32 của khối dữ liệu vào được chuyển tới vị trí thứ nhất trong khối dữ liệu ra, bit ở vị trí thứ 4 của khối dữ liệu vào được chuyển tới vị trí thứ 5 và 7 trong khối dữ liệu ra,...)

32	1	2	3	4	5	4	5	6	7	8	9
8	9	10	11	12	12	12	13	14	15	16	17
16	17	18	19	20	21	20	21	22	23	24	25
24	25	26	27	28	29	28	29	30	31	32	1

Bước 6: kết quả của bước 3 và bước 5 được XOR với nhau.

Bước 7: Kết quả của bước 6 được chuyển thành 32 bit bằng cách sử dụng hàm thay thế và lựa chọn.

Sự thay thế được thực hiện bởi 8 hộp thay thế (substitution boxes, S-boxes). Khối 48 bit được chia thành 8 khối 6 bit. Mỗi khối được thực hiện trên một hộp S riêng biệt (separate S-box): khối 1 được thực hiện trên hộp S1, khối 2 được thực hiện trên hộp S2...

Mỗi hộp S là một bảng gồm 4 hàng và 16 cột. Mỗi phần tử của hộp là một số 4 bit. Với sáu bit vào hộp S sẽ xác định được số hàng và số cột để tìm ra kết quả.

Cách thức xác định kết quả: nhận vào 6 bit lần lượt là b1, b2, b3, b4, b5, và b6. Bit b1 và b6 được kết hợp lại thành một số 2 bit tương ứng với số hàng trong bảng (có giá trị từ 0 đến 3). Bốn bit ở giữa được kết hợp lại thành một số 4 bit tương ứng với số cột trong bảng (nhận giá trị từ 0 đến 15).

Ví dụ: Dùng hộp S thứ 6. Nếu dữ liệu nhận vào là 110010. Bit đầu tiên kết hợp với bit cuối tạo thành 10 (khi đổi sang số thập phân có giá trị bằng 2 tương ứng với hàng thứ 2). Bốn bit giữa kết hợp lại thành 1001(khi đổi sang số thập phân có giá trị bằng 9 tương ứng với cột thứ 9) => Giá trị cần tìm hàng 2 cột 9 là 0. Như vậy giá trị 0000 được thay thế cho 110010.

Dùng hộp S thứ nhất. Nếu dữ liệu nhận vào là 011011. Bit đầu tiên kết hợp với bit cuối tạo thành 01 (khi đổi sang số thập phân có giá trị bằng 1 tương ứng với hàng 1). Bốn bit giữa kết hợp lại thành 1101(khi đổi sang số thập phân có giá trị bằng 13 tương ứng với cột thứ 13) => Giá trị cần tìm hàng 1 cột 13 là 5. Như vậy giá trị 0101 được thay thế cho 011011.

Bảng 5: Bảng hộp S:

Hộp S thứ nhất.

14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

Hộp S thứ hai.

15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

Hộp S thứ ba.

10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

Hộp S thứ tư.

7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

Hộp S thứ năm.

2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

Hộp S thứ sáu.

12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

Hộp S thứ bảy.

4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

Hộp S thứ tám.

13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

Kết quả của sự thay thế là 8 khối 4 bit được sinh ra, chúng được kết hợp lại thành một khối 32 bit. Khối này được chuyển tới bước tiếp theo: hộp hoán vị P (P-box permutation). Hoán vị ở bước này ánh xạ mỗi bit dữ liệu vào tới một vị trí trong khối dữ liệu ra, không có bit nào bị bỏ qua cũng như được sử dụng hai lần. nó còn được gọi là hoán vị trực tiếp (straight permutation).

Bảng 6: Bảng hộp hoán vị P cho biết vị trí của mỗi bit cần chuyển (bit 1 chuyển tới bit 16, bit 2 chuyển tới bit 7...)

16	7	20	21	29	12	28	17	1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9	19	13	30	6	22	11	4	25

Bước 8: Kết quả của bước 7 được XOR với nửa trái 32 bit được tạo ra ở bước 2.

Bước 9: kết quả tạo ra ở bước 8 trở thành nửa phải mới, nửa phải cũ (tạo ở bước 2) trở thành nửa trái mới.

Sau khi thực hiện hết 16 vòng lặp hoán vị cuối cùng được thực hiện để kết thúc thuật toán.

Hoán vị cuối cùng là nghịch đảo của hoán vị khởi đầu.

Bảng 7: Bảng hoán vị cuối

40	8	48	16	56	24	64	32	39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30	37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28	35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26	33	1	41	9	49	17	57	25

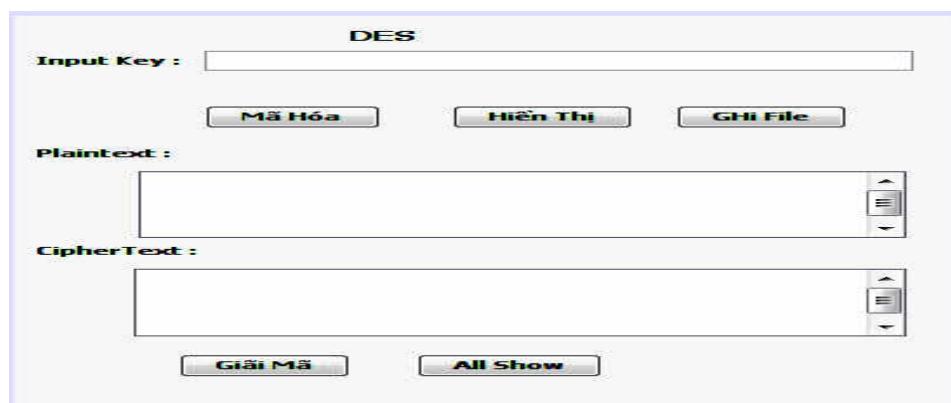
Các chế độ hoạt động của DES

Có bốn chế độ làm việc đã được phát triển cho DES:

- Chế độ sách mã điện tử (ECB).
- Chế độ phản hồi mã (CFB).
- Chế độ liên kết khối mã (CBC – Cipher Block Chaining).
- Chế độ phản hồi đầu ra (OFB).

❖ Hướng dẫn thực hành

Bước 1: Thiết Kế Form:



Bước 2: Viết hàm xử lý sự kiện

B2.1 Hàm doCopy

```
private int mode;
private static void doCopy(InputStream is, OutputStream os) throws IOException{
    byte[] bytes = new byte[64];
    int numBytes;
    while ((numBytes = is.read(bytes)) != -1) {
        os.write(bytes, 0, numBytes);
    }
    os.flush();
    os.close();
    is.close();
}
```

B2.2 Hàm mã Hóa và giải mã

```
public static void encrypt(String key, InputStream is, OutputStream os) throws Throwable {
    encryptOrDecrypt(key, Cipher.ENCRYPT_MODE, is, os);
}

public static void decrypt(String key, InputStream is, OutputStream os) throws Throwable {
    encryptOrDecrypt(key, Cipher.DECRYPT_MODE, is, os);
}
```

B2.3 Hàm thực hiện

```
public static void encryptOrDecrypt(String key, int mode, InputStream is, OutputStream os) throws Throwable {
    DESKeySpec dks = new DESKeySpec(key.getBytes());
    SecretKeyFactory skf = SecretKeyFactory.getInstance("DES");
    SecretKey desKey = skf.generateSecret(dks);
    Cipher cipher = Cipher.getInstance("DES"); // DES/ECB/PKCS5Padding for SunJCE

    if (mode == Cipher.ENCRYPT_MODE) {
        cipher.init(Cipher.ENCRYPT_MODE, desKey);
        CipherInputStream cis = new CipherInputStream(is, cipher);
        doCopy(cis, os);
    } else if (mode == Cipher.DECRYPT_MODE) {
        cipher.init(Cipher.DECRYPT_MODE, desKey);
        CipherOutputStream cos = new CipherOutputStream(os, cipher);
        doCopy(is, cos);
    }
}
```

B2.4 Viết chức năng Mã Hóa

```
private void bntMaHoaActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        String key=txtkhoa.getText(); // needs to be at least 8 characters for DES

        FileInputStream fis = new FileInputStream("D:\\\\Des.txt");
        FileOutputStream fos = new FileOutputStream("D:\\\\EnDes.txt");
        encrypt(key, fis, fos);
        JOptionPane.showMessageDialog(null,"Đã mã hóa văn bản");
        //FileInputStream fis2 = new FileInputStream("D:\\\\encrypted.txt");
        //FileOutputStream fos2 = new FileOutputStream("D:\\\\decrypted.txt");
        //decrypt(key, fis2, fos2);
    } catch (Throwable e) {
        e.printStackTrace();
    }
}
```

B2.5 Viết Chức năng Giải Mã

```
private void bntGiaiMaActionPerformed(java.awt.event.ActionEvent evt) {
    FileInputStream fis2 = null;
    try {
        String key = txtkhoa.getText();
        fis2 = new FileInputStream("D:\\\\EnDes.txt");
        FileOutputStream fos2 = new FileOutputStream("D:\\\\DeDes.txt");
        decrypt(key, fis2, fos2);
        BufferedReader br = null;

        String fileName= "D:\\\\DeDes.txt";
    }
}
```

```
        br = new BufferedReader(new FileReader(fileName));
        StringBuffer sb = new StringBuffer();
        JOptionPane.showMessageDialog(null, "Đã Giải Mã");
        char[] ca = new char[5];
        while (br.ready()) {
            int len = br.read(ca);
            sb.append(ca, 0, len);
        }
        br.close();
        //xuat chuoi
        System.out.println("Dữ Liệu là :" + " " + sb);
        String chuoi = sb.toString();

        txtmahoa.setText(chuoi);
    } catch (Throwable ex) {
        Logger.getLogger(DESCS.class.getName()).log(Level.SEVERE, null, ex);
    } finally {
        try {
            fis2.close();
        } catch (IOException ex) {
            Logger.getLogger(DESCS.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```

B2.6 Viết chức năng Ghi FILE

```
private void bntGhiFileActionPerformed(java.awt.event.ActionEvent evt) {
    try {

        BufferedWriter bw = null;
        //ghi van ban da ma hoa
        String fileName = "D:\\\\Des.txt";
        //luu van ban
        String s = txtvanban.getText();
        bw = new // van ban sau khi ma hoa
        BufferedWriter(new FileWriter(fileName));
        // ghi van ban
        bw.write(s);
        bw.close();
        JOptionPane.showMessageDialog(null,"Đã ghi file");
        txtmahoa.setText(s);
    } catch (IOException ex) {
        Logger.getLogger(DESCS.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

B2.7 Viết chức năng Mở FILE

```

private void bntMoFileActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        BufferedReader br = null;

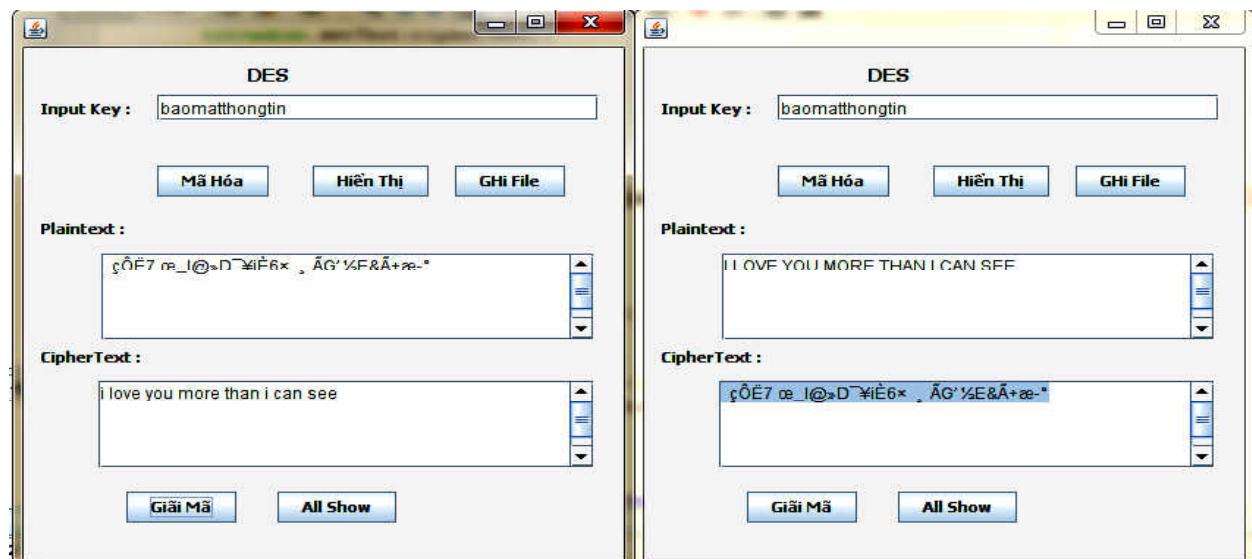
        String fileName = "D:\\EnDes.txt"; //GEN-
        br = new BufferedReader(new FileReader(fileName));
        StringBuffer sb = new StringBuffer();

        JOptionPane.showMessageDialog(null, "Đã mở file");
        char[] ca = new char[5];
        while (br.ready()) {
            int len = br.read(ca);
            sb.append(ca, 0, len);
        }
        br.close();
        //xuat chuoi
        System.out.println("Du Lieu la :" + " " + sb);
        String chuoi = sb.toString();

        txtvanban.setText(chuoi);
    } catch (IOException ex) {
        Logger.getLogger(DESCS.class.getName()).log(Level.SEVERE, null, ex);
    }
}

```

Bước 3: Kiểm Tra



Bài thực hành số 3

Viết chương trình mã hóa và giải mã văn bản với thuật toán mã hóa 3DES.

Chương trình có thể thực hiện các chức năng sau:

- **Cho phép nhập văn bản vào hệ thống.**
- **Cho phép nhập khóa bảo vệ văn bản.**
- **Cho phép ghi File và mở File.**

❖ **Hướng dẫn thuật toán TRIPLEDES:**

TripleDES một biến thể an toàn hơn của DES còn được gọi là DESede hay 3DES. TripleDES có tính bảo mật cao hơn DES do sử dụng 3 vòng DES với các khóa khác nhau. Vòng đầu tiên và vòng thứ ba là vòng mã hóa, vòng thứ hai là vòng giải mã. DESede có thể dùng hai hoặc ba khóa có độ dài 56, 112 hoặc 168. nếu dùng hai khóa thì khóa đầu tiên được dùng cho vòng thứ nhất và vòng thứ ba, khóa thứ hai dùng cho vòng thứ hai.

- Mã hóa với ba khóa 56 bit (168 bit).

Bảng B1: 3DES Mã hóa với ba khóa 56 bit

NGƯỜI GỬI	NGƯỜI NHẬN
Bước 1: mã hóa plaintext bằng khóa thứ nhất	Bước 1: giải mã bản mã với khóa thứ ba
Bước 2: mã hóa văn bản được tạo ra ở bước 1 bằng khóa thứ hai	Bước 2: giải mã văn bản được tạo ra ở bước 1 bằng khóa thứ hai
Bước 3: mã hóa văn bản được tạo ra ở bước 2 bằng khóa thứ ba, tạo ra bản mã gửi cho người nhận.	Bước 3: giải mã văn bản được tạo ra ở bước 2 bằng khóa thứ nhất, tạo ra bản gốc do người gửi gửi.

- Mã hóa với hai khóa 56 bit (112 bit)

Bảng B2: 3DES Mã hóa với hai khóa 56 bit

NGƯỜI GỬI	NGƯỜI NHẬN
Bước 1: mã hóa plaintext bằng khóa nhất	Bước 1: giải mã bản mã với khóa thứ nhất
Bước 2: giải mã văn bản được tạo ra ở bước 1 bằng khóa thứ hai	Bước 2: mã hóa văn bản được tạo ra ở bước 1 bằng khóa thứ hai
Bước 3: mã hóa văn bản được tạo ra ở bước 2 bằng khóa thứ nhất, tạo ra bản mã gửi cho người nhận.	Bước 3: giải mã văn bản được tạo ra ở bước 2 bằng khóa thứ nhất, tạo ra bản gốc do người gửi gửi.

- Mã hóa với một khóa 56 bit

Bảng B3: 3DES Mã hóa với một khóa 56 bit

NGƯỜI GỬI	NGƯỜI NHẬN
Bước 1: mã hóa plaintext	Bước 1: giải mã bản mã nhận được từ người gửi.
Bước 2: giải mã văn bản được tạo ra ở bước 1	
Bước 3: mã hóa văn bản được tạo ra, tạo ra bản mã gửi cho người nhận.	

Mặc dù 3DES có tính bảo mật cao hơn DES, nhưng thực tế ít được sử dụng vì để tạo ra được bản mã phải chạy ba lần DES, nên tốc độ chậm, chiếm nhiều tài nguyên.

❖ Hướng dẫn thực hành

Bước 1: Thiết Kế Form:

The screenshot shows a software window titled "DESEDE". It has three main input fields: "Input Key", "Plaintext", and "CipherText", each with a dropdown arrow icon. Below these fields are two buttons: "Mã Hóa" (Encrypt) and "Giải Mã" (Decrypt). At the top right are three more buttons: "Hiển Thị" (Display), "Ghi File" (Save File), and "All Show".

Bước 2: Viết hàm xử lý sự kiện

B2.1: Khai báo các biến sau

```

private static final String UNICODE_FORMAT = "UTF8";
public static final String DESEDE_ENCRYPTION_SCHEME = "DESede";
private KeySpec myKeySpec;
private SecretKeyFactory mySecretKeyFactory;
private Cipher cipher;
byte[] keyAsBytes;
private String myEncryptionKey;
private String myEncryptionScheme;
SecretKey key;

```

B2.2: Viết phương thức mã hóa encrypt

```

public String encrypt(String unencryptedString) {
    String encryptedString = null;
    try {
        cipher.init(Cipher.ENCRYPT_MODE, key);
        byte[] plainText = unencryptedString.getBytes(UNICODE_FORMAT);
        byte[] encryptedText = cipher.doFinal(plainText);
        BASE64Encoder base64encoder = new BASE64Encoder();
        encryptedString = base64encoder.encode(encryptedText);
    } catch (Exception e) {
        e.printStackTrace();
    }
    return encryptedString;
}

```

B2.3: Viết phương thức giải mã

```

public String decrypt(String encryptedString) {
    String decryptedText=null;
    try {
        cipher.init(Cipher.DECRYPT_MODE, key);
        BASE64Decoder base64decoder = new BASE64Decoder();
        byte[] encryptedText = base64decoder.decodeBuffer(encryptedString);
        byte[] plainText = cipher.doFinal(encryptedText);
        String a= new String(plainText);
        System.out.println("chuoi plaintext :" + a);
        // decryptedText= bytes2String(plainText);
        decryptedText=a;
    } catch (Exception e) {
        e.printStackTrace();
    }
    return decryptedText;
}

```

B2.4 Viết hàm xử lý sự kiện mã hóa

```
private void bntMaHoaActionPerformed(java.awt.event.ActionEvent evt) {  
    try{  
        // "Sanjaal.com"  
        myEncryptionKey =txtkhoa.getText();  
        myEncryptionScheme = DESEDE_ENCRYPTION_SCHEME;  
        keyAsBytes = myEncryptionKey.getBytes(UNICODE_FORMAT);  
        myKeySpec = new DESedeKeySpec(keyAsBytes);  
        mySecretKeyFactory = SecretKeyFactory.getInstance(myEncryptionScheme);  
        cipher = Cipher.getInstance(myEncryptionScheme);  
        key = mySecretKeyFactory.generateSecret(myKeySpec);  
        System.out.println(" khoa ma hoa k :" + " " + key);  
        // sử dụng lớp DESEDE_EM  
        String plainText=txtvanban.getText();  
        // gọi phương thức mã hóa  
        String encrypted=encrypt(plainText);  
        System.out.println("Encrypted Value :" + encrypted);  
        txtmahoa.setText(encrypted);  
    } catch( Exception ex){}  
}
```

Bước 3: Kiểm Tra

The image displays four screenshots of a software application titled "DESEDE". The application has a standard Windows-style window with a title bar, menu bar, and toolbars.

Input Key: baomathongtinbuivanthieu

Plaintext: mon thuc hanh bao mat thong tin

CipherText: QVaSWai+0cdC614kJxfi6qAagUs7ZKKgM4Mzf21Lm1I4fu/lnTmsVQ==

Buttons: Mã Hóa, Hiển Thị, Ghi File, Giải Mã, All Show

The screenshots show the following sequence of operations:

- In the first screenshot, the plaintext "mon thuc hanh bao mat thong tin" is entered into the Plaintext field. The ciphertext field contains the encoded string "QVaSWai+0cdC614kJxfi6qAagUs7ZKKgM4Mzf21Lm1I4fu/lnTmsVQ==".
- In the second screenshot, the ciphertext "QVaSWai+0cdC614kJxfi6qAagUs7ZKKgM4Mzf21Lm1I4fu/lnTmsVQ==" is selected in the CipherText field, and the "Giải Mã" (Decrypt) button is clicked. The plaintext "mon thuc hanh bao mat thong tin" is displayed in the Plaintext field.
- In the third screenshot, the plaintext "mon thuc hanh bao mat thong tin" is selected in the Plaintext field, and the "Mã Hóa" (Encrypt) button is clicked. The ciphertext "QVaSWai+0cdC614kJxfi6qAagUs7ZKKgM4Mzf21Lm1I4fu/lnTmsVQ==" is displayed in the CipherText field.
- In the fourth screenshot, the ciphertext "QVaSWai+0cdC614kJxfi6qAagUs7ZKKgM4Mzf21Lm1I4fu/lnTmsVQ==" is selected in the CipherText field, and the "Giải Mã" (Decrypt) button is clicked again. The plaintext "mon thuc hanh bao mat thong tin" is displayed in the Plaintext field.

Bài thực hành số 4

* * *

Viết chương trình mã hóa và giải mã văn bản với thuật toán mã hóa AES.

Chương trình có thể thực hiện các chức năng sau:

- Cho phép nhập văn bản vào hệ thống.
- Cho phép nhập khóa bảo vệ văn bản.
- Cho phép ghi File và mở File.

❖ **Hướng dẫn thuật toán AES:**

AES chỉ làm việc với khối dữ liệu 128 bít và khóa có độ dài 128, 192 hoặc 256 bít. Các khóa con sử dụng trong các chu trình được tạo ra bởi quá trình tạo khóa con Rijndael. Rijndael có thể làm việc với dữ liệu và khóa có độ dài bất kỳ là bội số của 32 bít nằm trong khoảng từ 128 tới 256 bít.

Hầu hết các phép toán trong thuật toán AES đều thực hiện trong một trường hữu hạn.

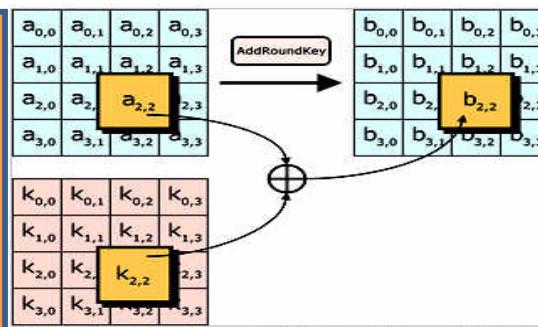
AES làm việc với từng khối dữ liệu 4×4 byte (tiếng Anh: state, khối trong Rijndael có thể có thêm cột). Quá trình mã hóa bao gồm 4 bước:

- a. AddRoundKey — mỗi byte của khối được kết hợp với khóa con, các khóa con này được tạo ra từ quá trình tạo khóa con Rijndael.
- b. SubBytes — đây là phép thay (phi tuyến) trong đó mỗi byte sẽ được thay bằng một byte khác theo bảng tra (Rijndael S-box).
- c. ShiftRows — đổi chỗ, các hàng trong khối được dịch vòng.
- d. MixColumns — quá trình trộn làm việc theo các cột trong khối theo một phép biến đổi tuyến tính.

Tại chu trình cuối thì bước MixColumns được thay thế bằng bước AddRoundKey

Bước AddRoundKey:

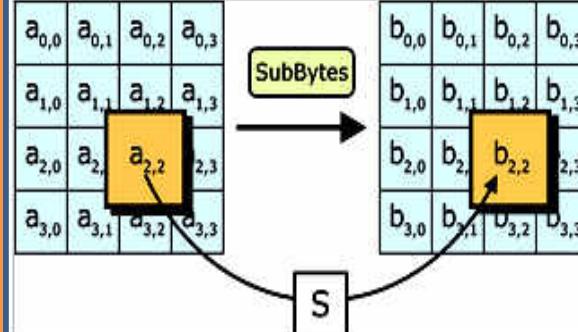
Tại bước này, khóa con được kết hợp với các khối. Khóa con trong mỗi chu trình được tạo ra từ khóa chính với quá trình tạo khóa con Rijndael; mỗi khóa con có độ dài giống như các khối. Quá trình kết hợp được thực hiện bằng cách XOR từng bit của khóa con với khối dữ liệu.



Trong bước AddRoundKey, mỗi byte được kết hợp với một byte trong khóa con của chu trình sử dụng phép toán XOR (\oplus).

Bước SubBytes

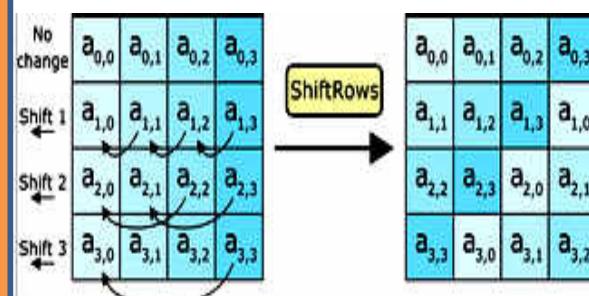
Các byte được thay thế thông qua bảng tra S-box. Đây chính là quá trình phi tuyến của thuật toán. Hộp S-box này được tạo ra từ một phép nghịch đảo trong trường hữu hạn $GF(2^8)$ có tính chất phi tuyến. Để chống lại các tấn công dựa trên các đặc tính đại số, hộp S-box này được tạo nên bằng cách kết hợp phép nghịch đảo với một phép biến đổi affine khả nghịch. Hộp S-box này cũng được chọn để tránh các điểm bất động (fixed point).



Trong bước SubBytes, mỗi byte được thay thế bằng một byte theo bảng tra, $b_j = S(a_j)$.

Bước ShiftRows

Các hàng được dịch vòng một số vị trí nhất định. Đối với AES, hàng đầu được giữ nguyên. Mỗi byte của hàng thứ 2 được dịch trái một vị trí. Tương tự, các hàng thứ 3 và 4 được dịch 2 và 3 vị trí. Do vậy, mỗi cột khối đầu ra của bước này sẽ bao gồm các byte ở đủ 4 cột khối đầu vào. Đối với Rijndael với độ dài khối khác nhau thì số vị trí dịch chuyển cũng khác nhau.

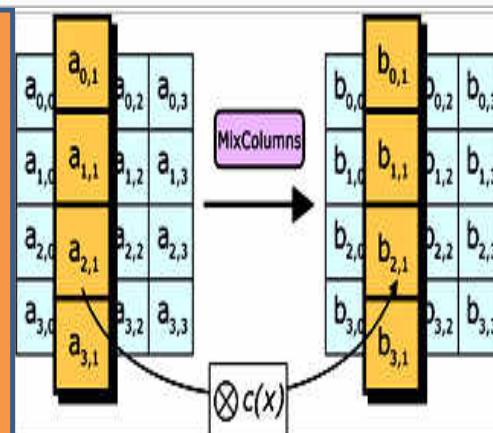


Trong bước ShiftRows, các byte trong mỗi hàng được dịch vòng trái. Số vị trí dịch chuyển tùy thuộc từng hàng.

Bước MixColumns

Bốn byte trong từng cột được kết hợp lại theo một phép biến đổi tuyến tính khả nghịch. Mỗi khối 4 byte đầu vào sẽ cho một khối 4 byte ở đầu ra với tính chất là mỗi byte ở đầu vào đều ảnh hưởng tới cả 4 byte đầu ra. Cùng với bước ShiftRows, MixColumns đã tạo ra tính chất khuyếch tán cho thuật toán. Mỗi cột được xem như một đa thức trong trường hữu hạn và được nhân với đa thức:

$c(x) = 3x^3 + x^2 + x + 2$ (modulo $x^4 + 1$). Vì thế, bước này có thể được xem là phép nhân ma trận trong trường hữu hạn.

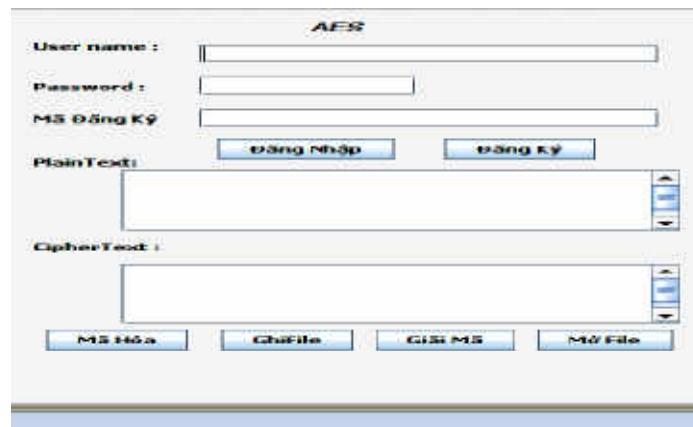


Trong bước MixColumns, mỗi cột được nhân với một hệ số cố định $c(x)$.

Tóm lại: Thuật toán AES có khối dữ liệu 128 bit, độ dài khóa 128,192, 256 bit, cấu trúc mạng thay thế-hoán vị, số chu trình 10,12,14 tùy theo độ dài khóa.

❖ Hướng dẫn thực hành

Bước 1: Thiết Kế Form:



Bước 2: Viết hàm xử lý sự kiện

B2.1: Viết hàm xử lý sự kiện đăng nhập

```
private void bntDangNhapActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        user = txtuser.getText();
        pass = txtpass.getText();
        khoa = user + pass;
        BufferedReader br = null;
        String fileName = "D:\\AES.txt"; //GEN-
        br = new BufferedReader( new FileReader(fileName));
        StringBuffer sb = new StringBuffer();
        char[] ca = new char[5];
        while (br.ready()) {
            int len = br.read(ca);
            sb.append(ca, 0, len);
        }
        br.close();
        //xuat chuoi
        System.out.println("Khoa la :" + " " + sb);
        String chuoit = sb.toString();
        Boolean k=khoa.equals(chuoit);
        if(k==true) JOptionPane.showMessageDialog(null, " Đăng Nhập Thành Công");
        else
            JOptionPane.showMessageDialog(null, " Đăng Nhập Thất bại");
        txtkhoa.setText(chuoit.getBytes().toString());
        KeyGenerator keyGen = KeyGenerator.getInstance("AES");
        keyGen.init(128);
        secretKey = keyGen.generateKey();
    } catch (NoSuchAlgorithmException ex) {
        // Logger.getLogger(FrAES.class.getName()).log(Level.SEVERE, null, ex);
    } catch (Exception ex) {
        // Logger.getLogger(FrAES.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

B2.2: Viết hàm xử lý sự kiện đăng ký

```
private void bntDangKyActionPerformed(java.awt.event.ActionEvent evt) {  
    try {  
        // TODO add your handling code here:  
        user = txtuser.getText();  
        pass = txtpass.getText();  
        khoa = user + pass;  
        BufferedWriter bw = null;  
        //ghi van ban da ma hua  
        String fileName = "D:\\AES.txt";  
        //luu van ban  
        String s = txtvanban.getText();  
        bw = new BufferedWriter(new FileWriter(fileName));  
        // ghi van ban  
        bw.write(khoa);  
        bw.close();  
        JOptionPane.showMessageDialog(null, "Bạn Đã Đăng Ký Thành Công .Vui lòng Đăng nhập lại !!!");  
        txtkhoa.setText(khoa.getBytes().toString());  
    } catch (IOException ex) {  
        Logger.getLogger(FraAES.class.getName()).log(Level.SEVERE, null, ex);  
    }  
}
```

B2.3: Viết hàm xử lý sự kiện mã hóa

```
private void bntMahoaActionPerformed(java.awt.event.ActionEvent evt) {  
    try {  
        System.out.println(" Sinh khoa: "+ secretKey);  
        //thuat toan AES  
        Cipher aesCipher = Cipher.getInstance("AES");  
        // Sinh khoa  
        aesCipher.init(Cipher.ENCRYPT_MODE, secretKey);  
        //*/ Mã hóa văn bản  
        String strData = txtvanban.getText();  
        //covert văn bản sang kiểu byte  
        byte[] byteDataToEncrypt = strData.getBytes();  
        // Gọi phương thức doFinal để mã hóa  
        byteCipherText = aesCipher.doFinal(byteDataToEncrypt);  
        String strCipherText = new BASE64Encoder().encode(byteCipherText);  
        System.out.println("Cipher Text generated using AES is " + strCipherText);  
        txtmahoa.setText(strCipherText);  
    } catch (Exception ex) {  
        System.out.println(" Lỗi mã hóa: "+ ex);  
    }  
}
```

B2.3 Viết hàm xử lý sự kiện giải mã

```

private void bntGiaiMaActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        // TODO add your handling code here:
        // txtvanban.setText(txtmahoa.getText());
        String cipherText=txtmahoa.getText();
        txtvanban.setText(cipherText);
        Cipher aesCipher = Cipher.getInstance("AES");
        aesCipher.init(Cipher.DECRYPT_MODE, secretKey, aesCipher.getParameters());
        //String giaima =txtmahoa.getText();
        // byte[] giaima=cipherText.getBytes();
        //byteCipherText
        byte[] byteDecryptedText = aesCipher.doFinal(byteCipherText);
        String strDecryptedText = new String(byteDecryptedText);
        System.out.println(" Decrypted Text message is " + strDecryptedText);
        txtmahoa.setText(strDecryptedText);

    } catch (Exception ex) {
        System.out.println(" Lỗi giải Mã: " + ex);
    }
}

```

B2.4 Viết hàm xử lý sự kiện ghi File

```

private void bntGhiFileActionPerformed(java.awt.event.ActionEvent evt) {
    try {

        BufferedWriter bw = null;
        //ghi van ban da ma hoa
        String fileName = "D:\\\\GhiAES.txt";
        //luu van ban
        String s = txtmahoa.getText();
        bw = new // van ban sau khi ma hoa
        BufferedWriter(new FileWriter(fileName));
        // ghi van ban
        bw.write(s);
        bw.close();
        JOptionPane.showMessageDialog(null, " Đã ghi file D:\\\\GhiAES.txt");
    } catch (IOException ex) {
        Logger.getLogger(FrAES.class.getName()).log(Level.SEVERE, null, ex);
    }
}

```

B2.5 Viết hàm xử lý sự kiện mở File

```

private void bntMoFileActionPerformed(java.awt.event.ActionEvent evt) {
    try {

        BufferedReader br = null;

        String fileName = "D:\\\\GhiAES.txt"; //GEN-
        br = new BufferedReader(new FileReader(fileName));
        StringBuffer sb = new StringBuffer();

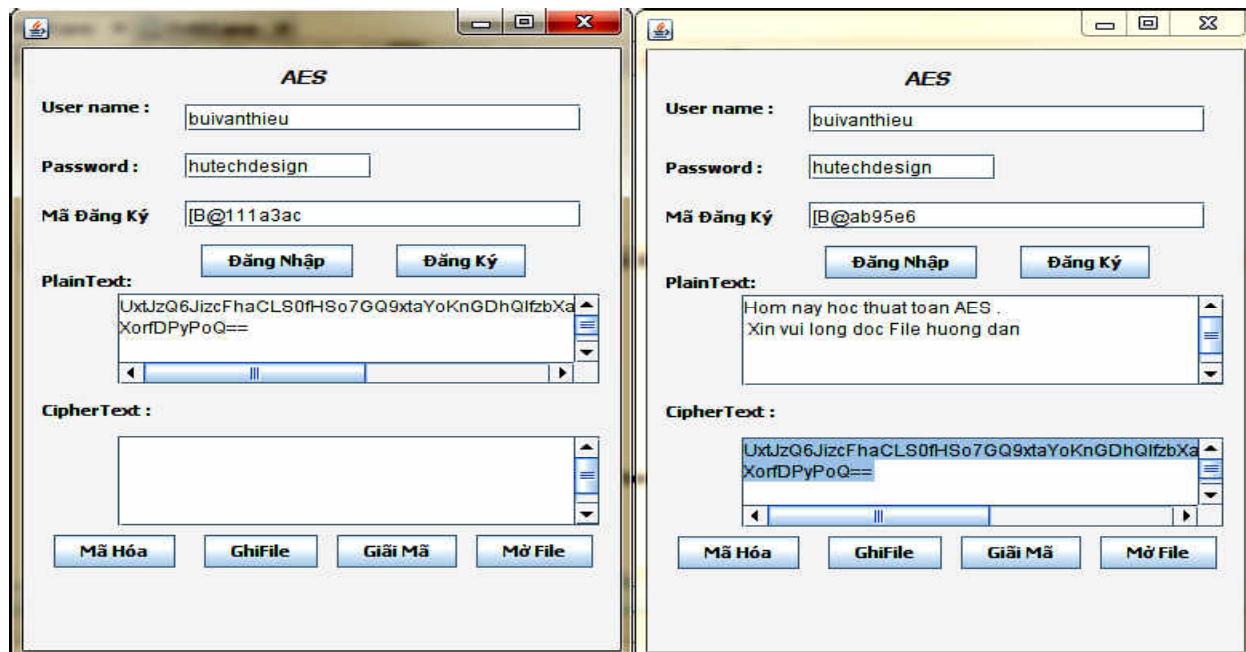
        JOptionPane.showMessageDialog(null, "Đã mở file");
        char[] ca = new char[5];
        while (br.ready()) {
            int len = br.read(ca);
            sb.append(ca, 0, len);
        }
        br.close();
        //xuat chuoi
        System.out.println("Du Lieu la :" + " " + sb);
        String chuoi = sb.toString();

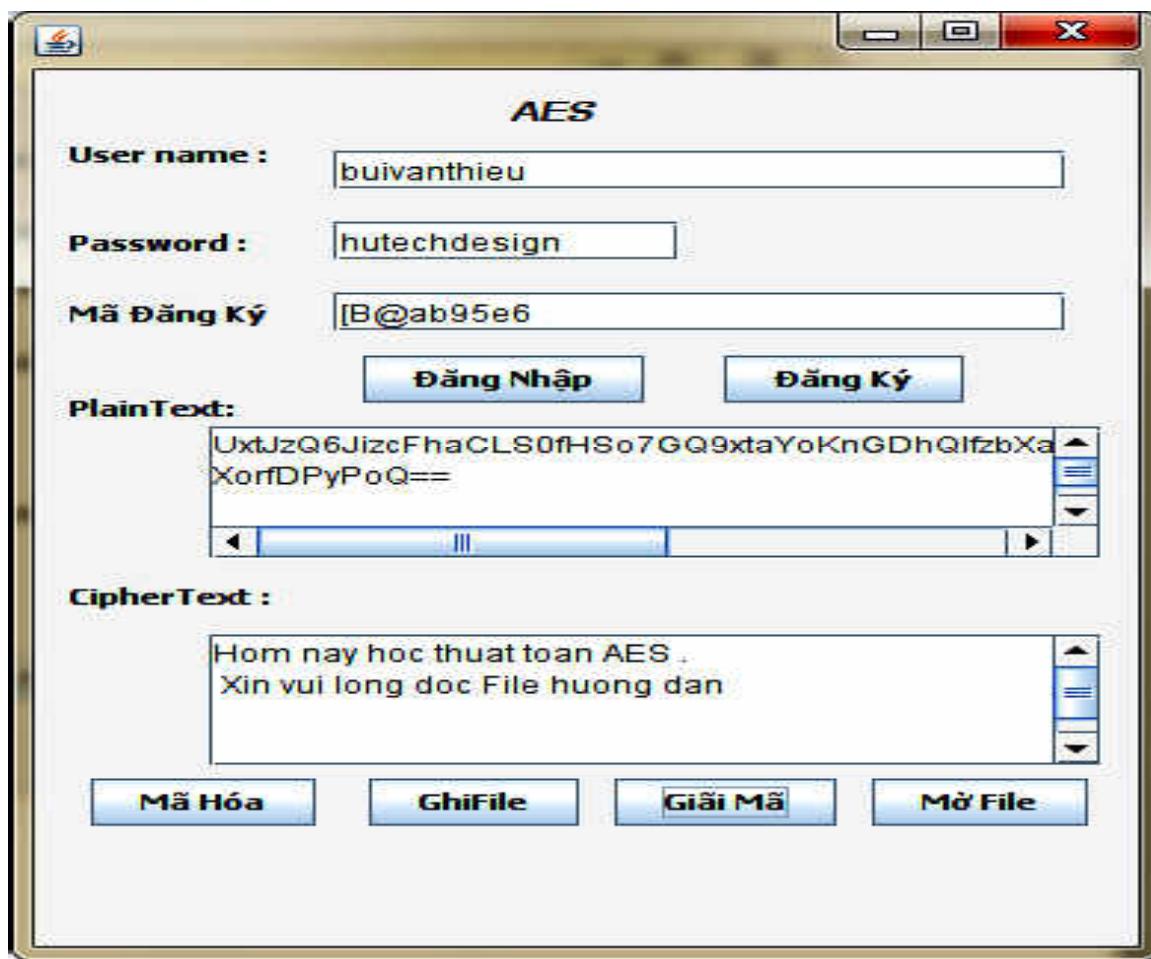
        txtvanban.setText(chuoi);
        // -----
        bntGiaiMa.setEnabled(true);

    } catch (IOException ex) {
        Logger.getLogger(FrAES.class.getName()).log(Level.SEVERE, null, ex);
    }
}

```

Bước 3: Kiểm Tra





Bài Thực Hành Số 5

Viết chương trình trao đổi dữ liệu theo mô hình Client-Server theo yêu cầu sau:

- **Sử dụng thuật toán Diffie-Hellman trao đổi quá cho nhau và sinh ra cặp khóa chung.**
- **Dùng khóa chung mã hóa và giải mã văn bản (văn bản được lưu ở dạng.doc,.dat,...)**

1.1 Hướng dẫn thuật toán trao đổi khóa Diffie-Hellman

Diffie-Hellman là một thuật toán dùng để trao đổi khóa chứ không dung để bảo vệ tính bí mật của dữ liệu. Tuy nhiên, Diffie-Hellman lại có ích trong giai đoạn trao đổi khóa bí mật của các thuật toán mật mã đối xứng.

Thuật toán trao đổi khóa Diffie-Hellman dựa trên phép logarit rời rạc. Cho trước một số g và $x=g^k$, để tìm k ta thực hiện phép logarit: $k = \log_g(x)$. Tuy nhiên, nếu cho trước g , n và $(g^k \bmod n)$, thì quá trình xác định k được thực hiện theo cách khác với cách ở trên và được gọi là logarit rời rạc.

Thuật toán Diffie-Hellman được mô tả như sau:

- Gọi n là một số nguyên tố lớn và g là một cơ số sinh (generator, số nguyên nhỏ) thỏa điều kiện: với mọi $x \in \{1, 2, \dots, n-1\}$, ta luôn tìm được số y sao cho $x=g^y \bmod n$.
- Giá trị n và g được phổ biến công khai giữa các thực thể trao đổi khóa. Sau đó user A tạo ra một số riêng $X_a < n$, tính giá trị $Y_a = (g^{x_a} \bmod n)$ và gởi cho B. Tương tự, user B cũng tạo ra một số riêng $X_b < n$ tính giá trị $Y_b = (g^{x_b} \bmod n)$ và gởi lại cho A. X_a và X_b tương đương khóa private, Y_a và Y_b tương đương khóa public.
- User B xác định được khóa bí mật dùng cho phiên làm việc bằng cách tính giá trị $(g^{x_a} \bmod n)^{x_b} = (g^{x_a x_b} \bmod n)$. Bằng cách tương tự, user A cũng xác định được cùng khóa bí mật này bằng cách tính giá trị $(g^{x_b} \bmod n)^{x_a} = (g^{x_a x_b} \bmod n)$.

- Giả sử trong quá trình trao đổi các giá trị, phía tấn công bắt được($g^{xa} \bmod n$) và ($g^{xb} \bmod n$), họ rất khó xác định được X_a và X_b vì độ phức tạp của phép toán logarit rắc rối là rất cao.
- Ví dụ: với $n=31$ và $g=3$

$$\begin{array}{l} A: x=8 \Rightarrow y=3^8 \bmod 31 = 20 \quad k = 16^8 \bmod 31 = 4 \\ B: x=6 \Rightarrow y=3^6 \bmod 31 = 16 \quad k = 20^6 \bmod 31 = 4 \end{array}$$

Khóa bí mật không được tạo trước và chuyển từ A sang B hoặc ngược lại, khóa bí mật chỉ được tạo ra sau khi A và B trao đổi với nhau Y_a và Y_b . Vì vậy khóa bí mật này thường được gọi là khóa phiên.

1.2 Hướng dẫn thực hành

1.2.1 Tạo Class CryptoUtil viết phương thức sau:

```
public static final String toHexString(byte[] block)
{
    StringBuffer buf = new StringBuffer();
    int len = block.length;

    for (int i = 0; i < len; i++)
    {
        byte2hex(block[i], buf);
        if (i < len-1)
        {
            buf.append(":");
        }
    }
    return buf.toString();
}
```

Hàm này dùng để chuyển từ kiểu hex sang kiểu String.

```
public static final void byte2hex(byte b, StringBuffer buf)
{
    char[] hexChars = { '0', '1', '2', '3',
                        '4', '5', '6', '7',
                        '8', '9', 'A', 'B',
                        'C', 'D', 'E', 'F' };

    int high = ((b & 0xf0) >> 4);
    int low = (b & 0x0f);
    buf.append(hexChars[high]);
    buf.append(hexChars[low]);
}
```

1.2.2 Bên Alice

1.2.2.1 Thiết kế form Alice

Alice

Khóa Alice : **Tạo Khóa A**

Khóa Bob : **Hiển Thị KB**

Khóa KAB: **Khóa Chung**

Mã Hóa KAB: **Mã Hóa KAB**

Mã Hóa/Giải Mã

1.2.2.1 Viết hàm xử lý sự kiện Tạo Khóa A

Các biến cần khai báo bên Alice

```

KeyAgreement aliceKeyAgree;
PublicKey bobPubKey;
SecretKey aliceDesKey;
Cipher aliceCipher;

private void bntkhoaAAActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    try{
        AlgorithmParameterGenerator paramGen = AlgorithmParameterGenerator.getInstance("DH");
        paramGen.init(512);
        AlgorithmParameters params = paramGen.generateParameters();

        DHParameterSpec dhSkipParamSpec =
            (DHParameterSpec) params.getParameterSpec(DHParameterSpec.class);

        System.out.println("Generating a DH KeyPair...");
        KeyPairGenerator aliceKpairGen = KeyPairGenerator.getInstance("DH");
        aliceKpairGen.initialize(dhSkipParamSpec);
        KeyPair aliceKpair = aliceKpairGen.generateKeyPair();

        System.out.println("Initializing the KeyAgreement Engine with DH private key");
        aliceKeyAgree = KeyAgreement.getInstance("DH");
        aliceKeyAgree.init(aliceKpair.getPrivate());

        byte[] alicePubKeyEnc = aliceKpair.getPublic().getEncoded();
        FileOutputStream fos=new FileOutputStream("D:/A.pub");
        fos.write(alicePubKeyEnc);
        fos.close();
        txtkhoaaa.setText(alicePubKeyEnc.toString());

    }catch(Exception ex){}
}

```

1.2.2.2 Viết hàm xử lý sự kiện hiện thị KB

```
private void bntkhoaBActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    try{
        FileInputStream fis=new FileInputStream("D:/B.pub");
        byte[] bkeyP=new byte[fis.available()];
        fis.read(bkeyP);
        fis.close();
        txtkhoaB.setText(bkeyP.toString());
    }
    catch(Exception ex){
    }
}
```

1.2.2.3 Viết hàm xử lý sự kiện Khóa Chung

```
private void bntkhoaABActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    try{
        FileInputStream fis=new FileInputStream("D:/B.pub");
        byte[] bobPubKeyEnc=new byte[fis.available()];
        fis.read(bobPubKeyEnc);
        fis.close();

        KeyFactory aliceKeyFac = KeyFactory.getInstance("DH");
        X509EncodedKeySpec x509KeySpec = new X509EncodedKeySpec(bobPubKeyEnc);
        bobPubKey = aliceKeyFac.generatePublic(x509KeySpec);
        System.out.println("Executing PHASE1 of key agreement...");
        aliceKeyAgree.doPhase(bobPubKey, true);
        byte[] aliceSharedSecret = aliceKeyAgree.generateSecret();

        System.out.println("khoa chung: secret (DEBUG ONLY):" + CryptoUtil.toHexString(aliceSharedSecret));
        txtkhoaChung.setText(CryptoUtil.toHexString(aliceSharedSecret));

    }
    catch(Exception ex){}
}
```

1.2.2.4 Viết hàm xử lý sự kiện Mã Hóa KAB

```

private void bntmahoakabActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    // ma hoa khoa chung bang thuat toan DES
    try{
        aliceKeyAgree.doPhase(bobPubKey, true);
        aliceDesKey = aliceKeyAgree.generateSecret("DES");
        txtmahoakab.setText(aliceDesKey.toString());
        // khoa chung A-B
        BufferedWriter bw = null;
        // ghi van ban da ma hoa
        String fileName = "D:\\\\KhoaA.txt";
        // luu van ban
        bw = new BufferedWriter(new FileWriter(fileName));
        // ghi van ban
        bw.write(aliceDesKey.toString());
        bw.close();
    }catch(Exception ex){}
}

```

1.2.2.5 Viết hàm xử lý sự kiện Mã Hóa/ Giải mã

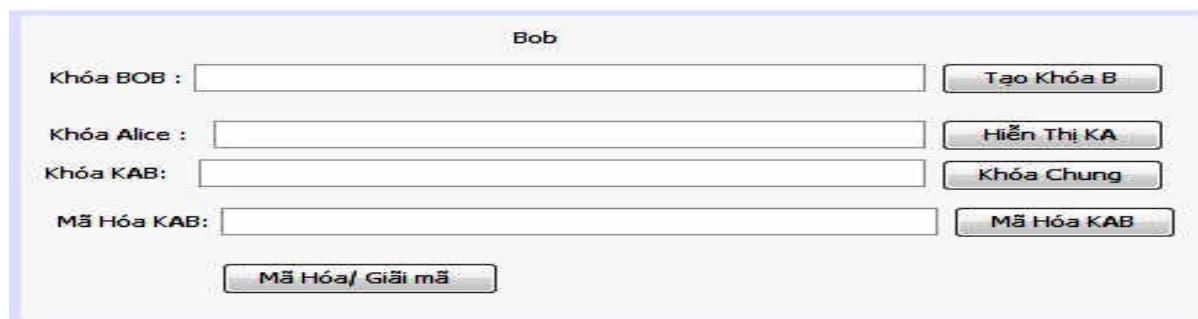
```

private void bntmahoagiaimaActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    DESCs des= new DESCs();
    des.setVisible(true);
}

```

1.2.3 Bên Bob

1.2.3.1 Thiết kế form Bob



Các biến cần khai báo cho bob

```

KeyAgreement bobKeyAgree;
PublicKey alicePubKey ;
SecretKey bobDesKey;
Cipher bobCipher;

```

1.2.3.2 Viết hàm xử lý sự kiện Hiển Thị Khóa KA(Alice)

```

private void bntkhoaAActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    try{
        FileInputStream fis=new FileInputStream("D:/A.pub");
        byte[] akeyP=new byte[fis.available()];
        fis.read(akeyP);
        fis.close();
        txtkhoaA.setText(akeyP.toString());
    }
    catch(Exception ex){
    }
}

```

1.2.3.3 Viết hàm xử lý sự kiện Tạo khóa KB (bob)

```

private void bntkhoaBActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    try{
        boolean read=false;
        //Read file A.pub to read Alice's public key
        while(!read){
            try{
                FileInputStream fis=new FileInputStream("D:/A.pub");
                fis.close();
                read=true;
            }
            catch(Exception ex){}
        }
        FileInputStream fis=new FileInputStream("D:/A.pub");
        byte[] alicePubKeyEnc=new byte[fis.available()];
        fis.read(alicePubKeyEnc);
        fis.close();
        KeyFactory bobKeyFac = KeyFactory.getInstance("DH");
        X509EncodedKeySpec x509KeySpec = new X509EncodedKeySpec(alicePubKeyEnc);
        alicePubKey = bobKeyFac.generatePublic(x509KeySpec);
        DHParameterSpec dhParamSpec = ((DHPublicKey) alicePubKey).getParams();
        System.out.println("Generate DH keypair ...");
        KeyPairGenerator bobKpairGen = KeyPairGenerator.getInstance("DH");
        bobKpairGen.initialize(dhParamSpec);
        KeyPair bobKpair = bobKpairGen.generateKeyPair();
        System.out.println("Initializing KeyAgreement engine...");
        bobKeyAgree = KeyAgreement.getInstance("DH");
        bobKeyAgree.init(bobKpair.getPrivate());
        byte[] bobPubKeyEnc = bobKpair.getPublic().getEncoded();
        FileOutputStream fos=new FileOutputStream("D:/B.pub");
        fos.write(bobPubKeyEnc);
        fos.close();
        txtkhoaB.setText(bobPubKeyEnc.toString());
    }catch(Exception ex){}
}

```

1.2.3.4 Viết hàm xử lý sự kiện Khóa Chung và sự kiện Mã Hóa KAB

```

private void bntkhoaAActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    try{
        bobKeyAgree.doPhase(alicePubKey, true);
        byte[] bobSharedSecret = bobKeyAgree.generateSecret();
        System.out.println("Khoa chung :Shared secret (DEBUG ONLY): " + CryptoUtil.toHexString(bobSharedSecret));
        txtkhoauchung.setText(CryptoUtil.toHexString(bobSharedSecret));
    }
    catch(Exception ex){}
}

private void bntmaoakabActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handing code here:
    try{
        bobKeyAgree.doPhase(alicePubKey, true);
        bobDesKey = bobKeyAgree.generateSecret("DES");
        txtmaoakab.setText(bobDesKey .toString());
        // khoa chung A-B
        BufferedWriter bw = null;
        //ghi van ban da ma hoa
        String fileName = "D:\\KhoaB.txt";
        //luu van ban
        bw = new // van ban sau khi ma hoa
        BufferedWriter(new FileWriter(fileName));
        // ghi van ban
        bw.write(bobDesKey .toString());
        bw.close();
    }catch(Exception ex){}
}

```

1.2.3.5 Viết hàm xử lý sự kiện Mã Hóa/ Giải mã

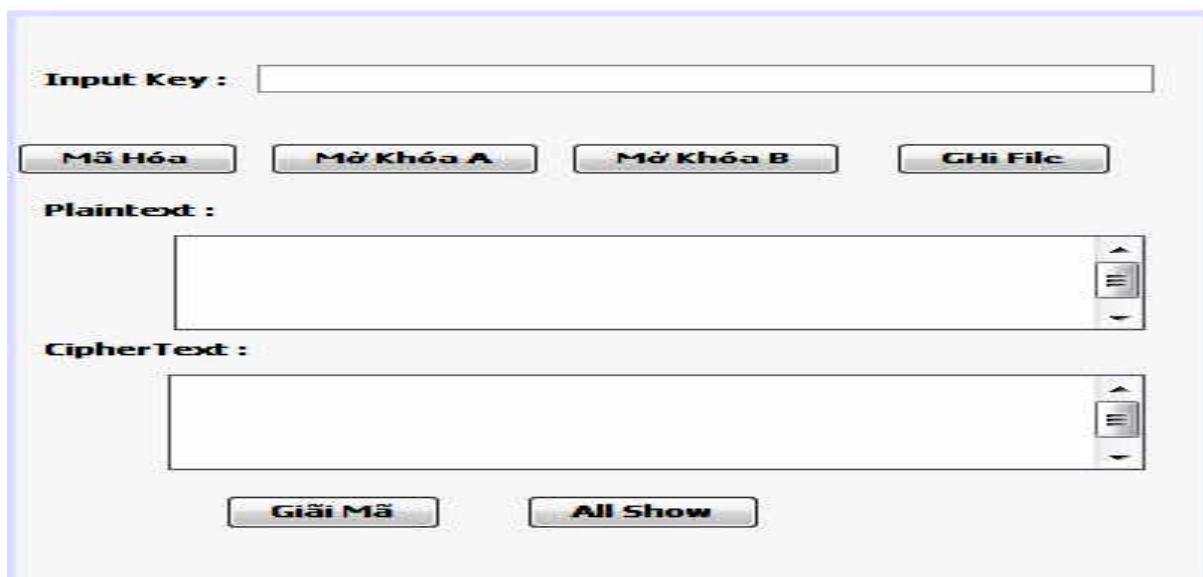
```

private void bntmaoagiaimaActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    DESCS des= new DESCS ();
    des.setVisible(true);
}

```

1.2.4 Viết Frame DESCS

1.2.4.1 Thiết kế form sao



1.2.4.2 Viết chức năng Mã Hóa

```

private void bntMaHoaActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        String key=txtkhoa.getText() ; // needs to be at least 8 characters for DES

        FileInputStream fis = new FileInputStream("D:\\Des.txt");
        FileOutputStream fos = new FileOutputStream("D:\\EnDes.txt");
        encrypt(key, fis, fos);
        JOptionPane.showMessageDialog(null, "Đã mã hóa văn bản");
        //FileInputStream fis2 = new FileInputStream("D:\\encrypted.txt");
        //FileOutputStream fos2 = new FileOutputStream("D:\\decrypted.txt");
        //decrypt(key, fis2, fos2);
    } catch (Throwable e) {
        e.printStackTrace();
    }
}

```

1.2.4.3 Viết chức năng mở khóa A

```

private void bntMoKhoaAActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    try {
        BufferedReader br = null;

        String fileName = "D:\\KhoaA.txt"; //GEN-
        br = new BufferedReader(new FileReader(fileName));
        StringBuffer sb = new StringBuffer();

        JOptionPane.showMessageDialog(null, "Đã mở file");
        char[] ca = new char[5];
        while (br.ready()) {
            int len = br.read(ca);
            sb.append(ca, 0, len);
        }
        br.close();
        //xuat chuoi
        System.out.println("Du Lieu la :" + " " + sb);
        String chuoit = sb.toString();

        txtkhoa.setText(chuoit);
    } catch (IOException ex) {
        Logger.getLogger(DESCS.class.getName()).log(Level.SEVERE, null, ex);
    }
}

```

1.2.4.4 Viết chức năng mở khóa B

```

private void bntMoKhoaBActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        BufferedReader br = null;

        String fileName = "D:\\KhoaB.txt"; //GEN-
        br = new BufferedReader(new FileReader(fileName));
        StringBuffer sb = new StringBuffer();

        JOptionPane.showMessageDialog(null, "Đã mở file");
        char[] ca = new char[5];
        while (br.ready()) {
            int len = br.read(ca);
            sb.append(ca, 0, len);
        }
        br.close();
        //xuat chuoi
        System.out.println("Du Lieu la :" + " " + sb);
        String chuoit = sb.toString();

        txtkhoa.setText(chuoit);
    } catch (IOException ex) {
        Logger.getLogger(DESCS.class.getName()).log(Level.SEVERE, null, ex);
    }
}

```

1.2.4.5 Viết chức năng ghi File

```
private void bntGhiFileActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        BufferedWriter bw = null;
        //ghi van ban da ma hoa
        String fileName = "D:\\\\Des.txt";
        //luu van ban
        String s = txtvanban.getText();
        bw = new // van ban sau khi ma hoa
        BufferedWriter(new FileWriter(fileName));
        // ghi van ban
        bw.write(s);
        bw.close();
        JOptionPane.showMessageDialog(null, "Đã ghi file");
        txtmahoa.setText(s);
    } catch (IOException ex) {
        Logger.getLogger(DESCS.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

1.2.4.6 Viết chức năng giải mã

```
private void bntGiaiMaActionPerformed(java.awt.event.ActionEvent evt) {
    FileInputStream fis2 = null;
    try {
        String key = txtkhoa.getText();
        fis2 = new FileInputStream("D:\\\\EnDes.txt");
        FileOutputStream fos2 = new FileOutputStream("D:\\\\DeDes.txt");
        decrypt(key, fis2, fos2);
        BufferedReader br = null;
        String fileName = "D:\\\\DeDes.txt"; //GEN-
        br = new BufferedReader(new FileReader(fileName));
        StringBuffer sb = new StringBuffer();
        JOptionPane.showMessageDialog(null, "Đã Giải Mã");
        char[] ca = new char[5];
        while (br.ready()) {
            int len = br.read(ca);
            sb.append(ca, 0, len);
        }
        br.close();
        //xuat chuoi
        System.out.println("Du Lieu la :" + " " + sb);
        String chuoi = sb.toString();
        txtmahoa.setText(chuoi);
    } catch (Throwable ex) {
    }
}
```

1.2.4.7 Viết chức năng hiển thị

```

private void bnthienthitatcaActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        BufferedReader br = null;
        String fileName = "D:\\DeDes.txt"; //GEN-
        br = new BufferedReader(new FileReader(fileName));
        StringBuffer sb = new StringBuffer();
        JOptionPane.showMessageDialog(null, "Đã mở file");
        char[] ca = new char[5];
        while (br.ready()) {
            int len = br.read(ca);
            sb.append(ca, 0, len);
        }
        br.close();
        String ff = "D:\\\\EnDes.txt";
        br = new BufferedReader(new FileReader(ff));
        StringBuffer sb1 = new StringBuffer();
        char[] ca1 = new char[5];
        while (br.ready()) {
            int len = br.read(ca1);
            sb1.append(ca1, 0, len);
        }
        //xuat chuoi
        System.out.println("Du Lieu la :" + " " + sb);
        String chuoi = sb.toString();
        String chuoi1 = sb1.toString();
        txtvanban.setText(chuoi);
        txtmahoa.setText(chuoi1);
    } catch (IOException ex) {
    }
}

```

1.2.5 Kết Quả

B1: Run Frame Alice: check button tạo khóa A.

B2: Run Frame Bob: check button tạo khóa B.

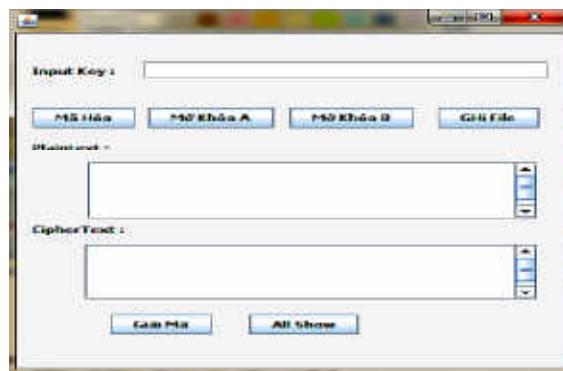
B3: Frame Alice:Check button hiển thị KB; Frame Bob: Check button hiển thị KA.

B4: Frame Alice và Frame Bob lần lượt check button tạo khóa Chung.

B5: Frame Alice và Frame Bob lần lượt check button mã hóa KAB



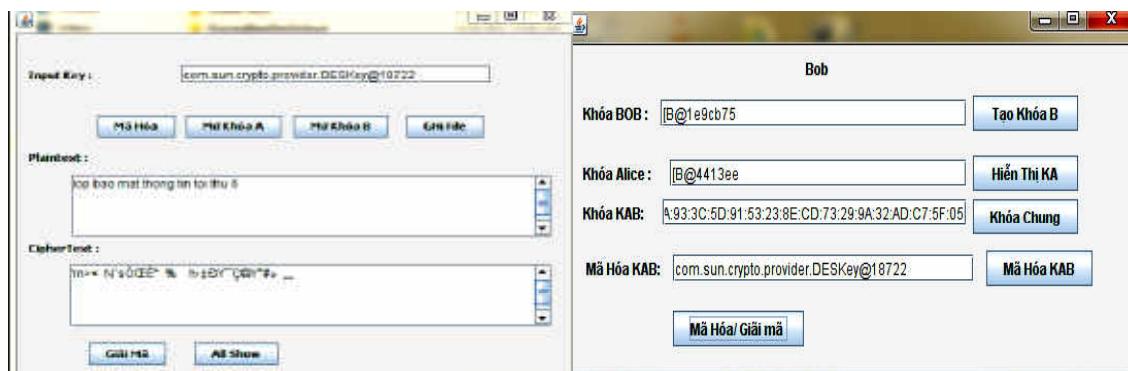
B5. Frame Alice: check button mã hóa và giải mã, button cho phép hiển thị form mã hóa và giải mã văn bản với thuật toán DES với khóa chung của Alice và Bob.



B6: Nhập nội dung văn bản cần mã hóa và ghi xuống File, trước khi nhập cần check button Mở Khóa A (khóa Alice) để dùng khóa Alice mã hóa văn bản.



B7: Frame Bob check button mã hóa và giải mã, button cho phép hiển thị form mã hóa và giải mã văn bản với thuật toán DES với khóa chung của Alice và Bob. Check button Mở Khóa B, dung khóa của Bob để giải mã văn bản do Alice mã hóa.



Bài Thực Hành Số 6



Viết chương trình mã hóa và giải mã văn bản với thuật toán mã hóa RSA.

- Thuật toán RSA:

- RSA là một thuật toán mật mã hoá công khai. Đây là thuật toán đầu tiên phù hợp với việc tạo ra chữ ký điện tử đồng thời với việc mã hóa. Nó đánh dấu một sự tiến bộ vượt bậc của lĩnh vực mật mã học trong việc sử dụng khoá công cộng
- Giả sử Alice và Bob cần trao đổi thông tin bí mật thông qua một kênh không an toàn (ví dụ như Internet). Với thuật toán RSA, Alice đầu tiên cần tạo ra cho mình cặp khóa gồm khóa công khai và khóa bí mật theo các bước sau
 1. Chọn 2 số nguyên tố lớn p và q với $p \neq q$, lựa chọn ngẫu nhiên và độc lập.
 2. Tính: $n = p \cdot q$
 3. Tính: giá trị hàm số $\phi(n)$.

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

4. Chọn một số tự nhiên e sao cho

$$1 < e < \phi(n)$$

và là số nguyên tố cùng nhau với $\phi(n)$

5. Tính: d sao cho

$$de \equiv 1 \pmod{\phi(n)}$$

- **Khóa công khai** bao gồm: n , môđun, và e , số mũ công khai (cũng gọi là *số mũ mã hóa*).
- **Khóa bí mật** bao gồm: n , môđun, xuất hiện cả trong khóa công khai và khóa bí mật, và d , số mũ bí mật (cũng gọi là *số mũ giải mã*).

- Ví dụ: Giả sử Bob muốn gửi đoạn thông tin M cho Alice. Đầu tiên Bob chuyển M thành một số $m < n$ theo một hàm có thể đảo ngược (từ m có thể xác định lại M) được thỏa thuận trước.

Lúc này Bob có m và biết n cũng như e do Alice gửi. Bob sẽ tính c là ẩn mã hóa của m theo công thức: $c = m^e \pmod{n}$

Cuối cùng Bos gửi c cho ALice.

Alice nhận c từ Bob và biết khóa bí mật d . Alice có thể tìm được m từ c theo công thức sau:

$$m \equiv c^d \pmod{n}$$

Biết m , Alice tìm lại M theo phương pháp đã thỏa thuận trước. Quá trình giải mã hoạt động vì ta có $c^d \equiv (m^e)^d \equiv m^{ed} \pmod{n}$. Do $ed \equiv 1 \pmod{p-1}$ và $ed \equiv 1 \pmod{q-1}$, (theo [Định lý Fermat nhỏ](#)) nên: $m^{ed} \equiv m \pmod{p}$

$$\text{Và } m^{ed} \equiv m \pmod{q}$$

Do p và q là hai số nguyên tố cùng nhau, ta có: $m^{ed} = m \pmod{pq}$

hay: $c^d = m \pmod{n}$

❖ Ví dụ 2:

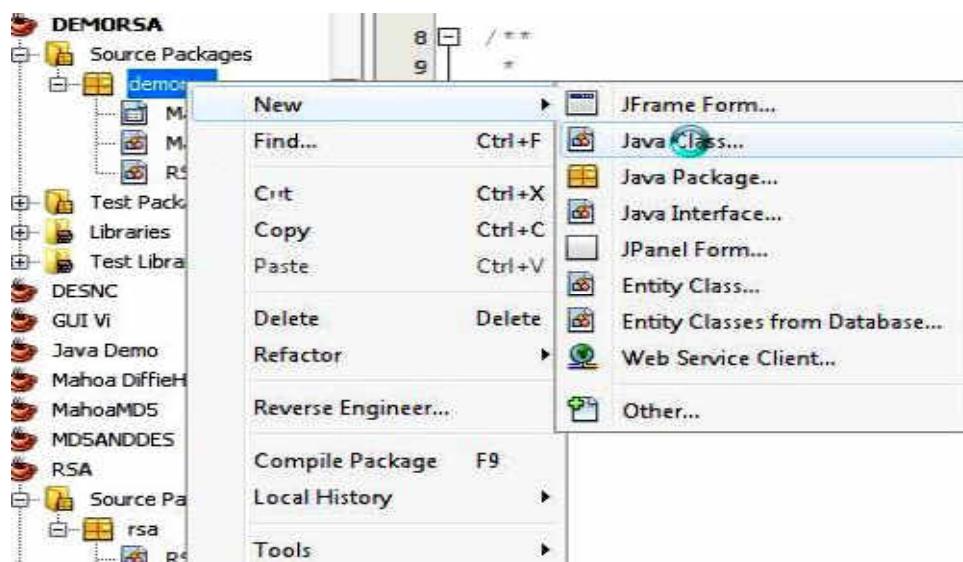
- Ta chọn hai số nguyên tố p và q , với $p=5$ và $q=7$
- Tính $n = p*q = 5*7 = 35$.
- $Z = (p-1)*(q-1) = (5-1)(7-1) = 24$
- Tiếp đến chọn e thỏa $1 < e < n$
 - -> chọn $e=5$.
- Tìm d , sao cho $e*d - 1$ chia hết cho z (24)
 - -> chọn $d=29$.
- Do đó: Public key $=(n,e) = (35,5)$
 - Private key $=(n,d) = (35,29)$
- Mã hóa chuỗi sau: secure

- Ta có bảng sau

Nội dung bị mã hóa	$M = c^d \text{ mod } n$	Dữ liệu gốc
24	19	S
10	5	E
33	3	C
21	21	u
23	18	R
10	5	e

Hướng dẫn thực hành: Mã hóa và giải mã văn bản sử dụng thuật toán RSA.

- B1: Tạo Projects DEMORSA
- B2: Tạo Class RSA
- Tạo một Class mới và đặt tên là RSA



- Viết Các phương thức cho Class RSA

```
package rsa;
import java.math.BigInteger;
import java.util.Random;
import java.io.*;
```

```
public class RSA {
    /**
     * Bit length of each prime number.
     */
    int primeSize;
    /**
     * Two distinct large prime numbers p and q.
     */
    BigInteger p, q;
    /**
     * Modulus N.
     */
    BigInteger N;
    /**
     *  $r = (p - 1) * (q - 1)$ 
     */
    BigInteger r;
    /**
     * Public exponent E and Private exponent D
     */
    BigInteger E, D;

    public RSA() {
    }
    public RSA( int primeSize )
    {
        this.primeSize = primeSize;
        // Generate two distinct large prime numbers p and q.
        generatePrimeNumbers();

        // Generate Public and Private Keys.
        generatePublicPrivateKeys();
    }

    public void generatePrimeNumbers()
    {
        p = new BigInteger( primeSize, 10, new Random() );
        do
        {
            q = new BigInteger( primeSize, 10, new Random() );
        }
        while( q.compareTo( p ) == 0 );
    }
    /**
     * Generate Public and Private Keys.
     */
}

public void generatePublicPrivateKeys()
{
    //  $N = p * q$ 
    N = p.multiply( q );
    //  $r = (p - 1) * (q - 1)$ 
    r = p.subtract( BigInteger.valueOf( 1 ) );
    r = r.multiply( q.subtract( BigInteger.valueOf( 1 ) ) );
    // Choose E, coprime to r and less than r
    do
    {
        E = new BigInteger( 2 * primeSize, new Random() );
    }
    while( ( E.compareTo( r ) != -1 ) ||
           ( E.gcd( r ).compareTo( BigInteger.valueOf( 1 ) ) != 0 ) );
    // Compute D, the inverse of E mod r
    D = E.modInverse( r );
}
```

```

public BigInteger[] encrypt( String message )
{
    int i ;
    byte[] temp = new byte[1] ;
    byte[] digits = message.getBytes() ;
    BigInteger[] bigdigits = new BigInteger[digits.length] ;
    for( i = 0 ; i < bigdigits.length ; i++ )
    {
        temp[0] = digits[i] ;
        bigdigits[i] = new BigInteger( temp ) ;
    }
    BigInteger[] encrypted = new BigInteger[bigdigits.length] ;
    for( i = 0 ; i < bigdigits.length ; i++ )
        encrypted[i] = bigdigits[i].modPow( E, N ) ;
    return( encrypted ) ;
}

public BigInteger[] encrypt( String message,BigInteger userD,BigInteger userN)
{
    int i ;
    byte[] temp = new byte[1] ;
    byte[] digits = message.getBytes() ;
    BigInteger[] bigdigits = new BigInteger[digits.length] ;
    for( i = 0 ; i < bigdigits.length ; i++ )
    {
        temp[0] = digits[i] ;
        bigdigits[i] = new BigInteger( temp ) ;
    }
    BigInteger[] encrypted = new BigInteger[bigdigits.length] ;
    for( i = 0 ; i < bigdigits.length ; i++ )
        encrypted[i] = bigdigits[i].modPow( userD, userN ) ;
    return( encrypted ) ;
}

public String decrypt( BigInteger[] encrypted,BigInteger D,BigInteger N )
{
    int i ;
    BigInteger[] decrypted = new BigInteger[encrypted.length] ;
    for( i = 0 ; i < decrypted.length ; i++ )
        decrypted[i] = encrypted[i].modPow( D, N ) ;
    char[] charArray = new char[decrypted.length] ;
    for( i = 0 ; i < charArray.length ; i++ )
        charArray[i] = (char) ( decrypted[i].intValue() ) ;
    return( new String( charArray ) ) ;
}

public BigInteger getE()
{
    return( E ) ;
}

public BigInteger getQ()
{
    return( Q ) ;
}

public BigInteger getR()
{
    return( R ) ;
}

public BigInteger getN()
{
    return( N ) ;
}

public BigInteger getK()
{
    return( K ) ;
}

public BigInteger getD()
{
    return( D ) ;
}

```

```

public static void main( String[] args ) throws IOException
{
    int primeSize =8;
    // Generate Public and Private Keys
    RSA rsa = new RSA( primeSize );
    System.out.println( "Key Size: [" + primeSize + "]" );
    System.out.println( "" );
    System.out.println( "Generated prime numbers p and q" );
    System.out.println( "p: [" + rsa.getp().toString( 16 ).toUpperCase() + "]" );
    System.out.println( "q: [" + rsa.getq().toString( 16 ).toUpperCase() + "]" );
    System.out.println( "" );
    System.out.println( "The public key is the pair (N, E) which will be published." );
    System.out.println( "N: [" + rsa.getN().toString( 16 ).toUpperCase() + "]" );
    System.out.println( "E: [" + rsa.getE().toString( 16 ).toUpperCase() + "]" );
    System.out.println( "" );
    System.out.println( "The private key is the pair (N, D) which will be kept private." );
    System.out.println( "N: [" + rsa.getN().toString( 16 ).toUpperCase() + "]" );
    System.out.println( "D: [" + rsa.getD().toString( 16 ).toUpperCase() + "]" );
    System.out.println( "" );
    // Get message (plaintext) from user
    System.out.println( "Please enter message (plaintext):" );
    String plaintext = ( new BufferedReader( new InputStreamReader( System.in ) ) ).readLine();
    System.out.println( "" );
    // Encrypt Message
    BigInteger[] ciphertext = rsa.encrypt( plaintext );
    System.out.print( "Ciphertext: [" );
    for( int i = 0 ; i < ciphertext.length ; i++ )
    {
        System.out.print( ciphertext[i].toString( 16 ).toUpperCase() );
        if( i != ciphertext.length - 1 )
            System.out.print( " " );
    }
    System.out.println( "]" );
    System.out.println( "" );
    RSA rsa1 = new RSA(8);
    String recoveredPlaintext = rsa1.decrypt( ciphertext ,rsa.getD(),rsa.getN() );
    System.out.println( "Recovered plaintext: [" + recoveredPlaintext + "]" );
}

```

- B3: Thiết kế Form



- B4: Viết hàm xử lý sự kiện

Khai báo biến:

```

public class frmRSA extends javax.swing.JFrame {
    RSA rsa=new RSA(8);
    BigInteger[] ciphertext=null;
    BigInteger n=null;
    BigInteger d=null;
    String message=null;

```

B4.1 Xử lý sự kiện mã hóa dữ liệu

```

private void bntmahoaActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    String vanban=txtvanban.getText();
    System.out.println(" Van Ban can ma hoa"+ vanban);
    n=rsa.getN();
    d=rsa.getD();
    ciphertext=rsa.encrypt(vanban);
    StringBuffer bf=new StringBuffer();
    for(int i=0;i<ciphertext.length;i++)
    {
        bf.append(ciphertext[i].toString(16).toUpperCase());
        if(i!=ciphertext.length-1)
            System.out.print(" ");
    }
    message=bf.toString();
    System.out.println("CipherText : " +message);
    txtcipher.setText(message);
}

```

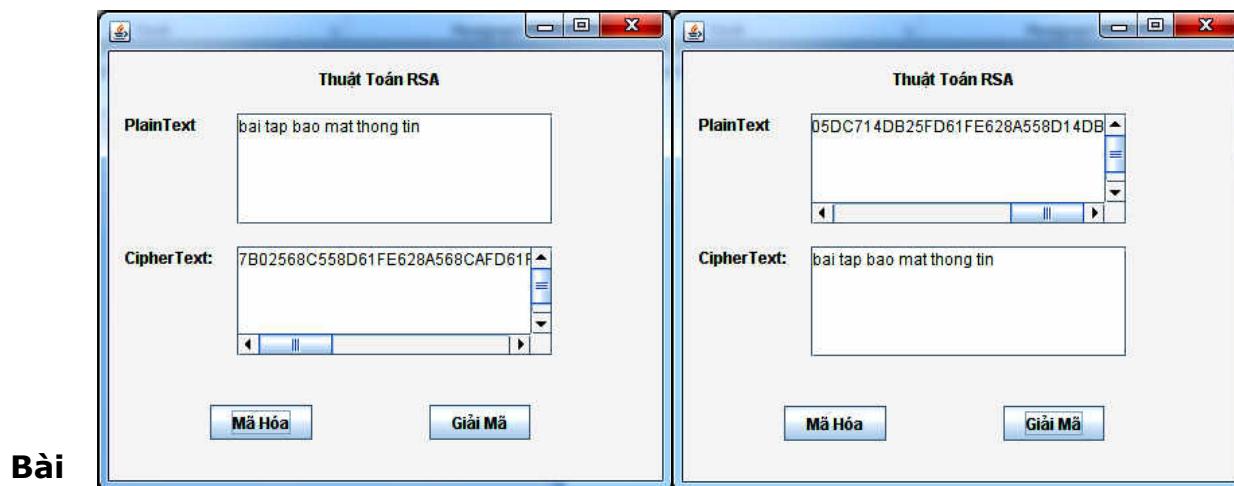
B4.2 Xử lý sự kiện giải mã dữ liệu

```

private void bntgiaimaActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    txtvanban.setText(message);
    String dhash=rsa.decrypt(ciphertext, d, n);
    System.out.println(" Van ban goc la " +dhash);
    txtcipher.setText(dhash);
}

```

- B5: Kiểm tra kết quả



Thực Hành Số 7

80*08

Bài 1: Hiện thực thuật toán hàm băm MD5 với yêu cầu sau:

- 1.1 Cho phép người dùng nhập username và password.**
- 1.2 Dùng thuật toán MD5 băm username, password và lưu vào File**
- 1.3 Dùng username và password đăng nhập, chứng thực với File đã ghi username, password.**

I. GIỚI THIỆU THUẬT TOÁN MD5:

MD5 (Message - Digest - algorithm 5) giải thuật

mã hóa tập tin là một chuẩn Internet (RFC 1321).

Có khả năng băm mã hóa tập tin bất kỳ thành chuỗi HEX 32 ký tự, tương đương 128-bit (*mỗi ký tự hex 4-*

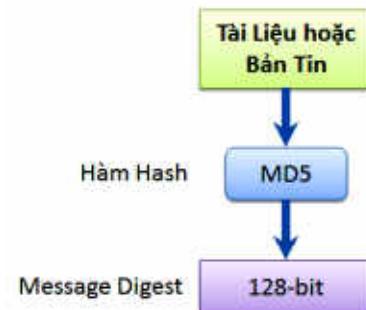
bit x 32 ký tự = 128 bit).

Hoặc có thể định nghĩa theo cách khác. MD5

là cách căn bản để lấy chùm ký tự (là digest, alphabetic hay gì khác), được gọi là string nhập vào và cho ra là 32 ký tự hexa.

(0,1,2,3,4,5,6,7,8,9,a,b,c,d,e,f).

MD5 được thiết kế bởi Ronald Rivest vào năm 1991 để thay thế cho hàm băm trước đó, MD4. Vào năm 1996, người ta phát hiện ra một lỗ hổng trong MD5; trong khi vẫn chưa biết nó có phải là lỗi nghiêm trọng hay không, những chuyên gia mã hóa bắt đầu đề nghị sử dụng những giải thuật khác, như SHA-1 (khi đó cũng bị xem là không an toàn). Trong năm 2004, nhiều lỗ hổng hơn bị khám phá khiến cho việc sử dụng giải thuật này cho mục đích bảo mật đang bị đặt nghi vấn.



❖ ĐẶC ĐIỂM MD5

Việc tính MD đơn giản, có khả năng xác định được file có kích thước nhiều Gb.

Không có khả năng tính ngược, khi tìm ra MD.

Do bản chất ngẫu nhiên của hàm băm và số lượng cực lớn các giá trị hash có thể, nên hầu như không có khả năng hai bản tin phân biệt có cùng giá trị hash.

Giá trị MD phụ thuộc vào bản tin tương ứng.

Một chuỗi chỉ có duy nhất một hash.

Giá trị MD phụ thuộc vào tất cả các bit của bản tin tương ứng.

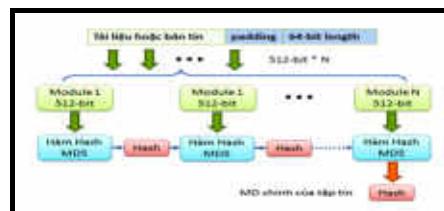
Ví dụ:

love is blue → 03d4ad6e7fee3f54eb46b5ccde58249c

love is Blue → 82b76f8eeb4a91aa640f9a23016c7b1c

II. THUẬT TOÁN

Giải thuật MD5 chính hoạt động trên trạng thái 128-bit, được chia thành 4 từ 32-bit, với ký hiệu A, B, C và D. Chúng được khởi tạo với những hằng số cố định. Giải thuật chính sau đó sẽ xử lý các khối tin 512-bit, mỗi khối xác định một trạng thái. Quá trình xử lý khối tin bao gồm bốn giai đoạn giống nhau, gọi là vòng; mỗi vòng gồm có 16 tác vụ giống nhau dựa trên hàm phi tuyến F, cộng Module, và dịch trái.



Thực hiện qua các bước sau:

Bước 1: Thêm các bit vào chuỗi

Thực hiện nối dài thông điệp. (theo hình vẽ thông điệp là B) để chi nhỏ thành các module 512.



- Thêm bit '1' vào cuối thông điệp để đánh dấu.
- Thêm vào k bit '0' sao cho $(b \text{ bit} + \text{bit } 1 + k \text{ bit } 0) \bmod 512 = 448$
- 64 bit tiếp theo sẽ được thêm vào biểu thị chiều dài của chuỗi bit ban đầu.

$$(B \text{ bit} + \text{bit '1'} + k \text{ bit '0'} + 64 \text{ bit chiều dài}) \bmod 512 = 0$$

Ví dụ: Ta có chuỗi 384bit

1	2	3	4	...	383	384	385	386	387	...
1	0	1	1	...	0	1	1	0	0	0
...	444	445	446	447	448	449	450	...	509	510
...	0	0	0	0	0	1	0	...	0	1

Quá trình thêm bit

Bước 2: Khởi tạo bộ đếm MD

Một bộ đếm 4 word (A,B,C,D) được dùng để tính mã số thông điệp. Ở đây mỗi A,B,C,D là một thanh ghi 32 bit. Những thanh ghi này được khởi tạo theo những giá trị hex sau (các byte thấp trước):

word A: 01 23 45 67

word B: 89 ab cd ef

word C: fe dc ba 98

word D: 76 54 32 10

Bước 3: Xử lý thông điệp theo từng khối 16 word

Trước hết ta định nghĩa các hàm phụ, các hàm này nhận đầu vào là 3 word 32 bit và tạo ra một word 32 bit.

$$\begin{aligned} F_1(X, Y, Z) &= (X \wedge Y) \vee (\neg X \wedge Z) \\ F_2(X, Y, Z) &= (X \wedge Z) \vee (Y \wedge \neg Z) \\ F_3(X, Y, Z) &= X \oplus Y \oplus Z \\ F_4(X, Y, Z) &= Y \oplus (X \vee \neg Z) \end{aligned}$$

Với $\oplus, \wedge, \vee, \neg$ lần lượt là **XOR, AND, OR, NOT**

Đây là quá trình thực hiện xử lý của 4 hàm F ở trên:

Quá trình này sử dụng một bảng có 64 giá trị T[1.. 64] được tạo ra từ hàm sin. Gọi T[i] là phần tử thứ i của bảng, thì T[i] là phần nguyên của $4294967296 * |\sin(i)|$, i được tính theo radian.

Thực hiện:

```
/* Xử lý mỗi khối 16 word */
For (i = 0 to N/16-1) do
    /* Copy block i into X. */
```

```

For j = 0 to 15 do
    Set X[j] to M[i*16+j].
end /* of loop on j */

/* Lưu A vào AA, B vào BB, C vào CC, D và DD. Làm buffer */
AA = A
BB = B
CC = C
DD = D

```

Quá trình thực hiện qua các vòng

Vòng 1

- **[abcd k s t]** là bước thực hiện của phép toán

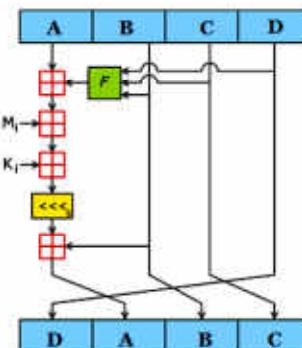
$$a = b + ((a + F(b, c, d) + X[k] + T[i]) \lll s)$$

/* Do the following 16 operations. */
[ABCD 0 7 1] [DABC 1 12 2] [CDAB 2 17 3] [BCDA 3 22 4]
[ABCD 4 7 5] [DABC 5 12 6] [CDAB 6 17 7] [BCDA 7 22 8]
[ABCD 8 7 9] [DABC 9 12 10] [CDAB 10 17 11] [BCDA 11 22 12]
[ABCD 12 7 13] [DABC 13 12 14] [CDAB 14 17 15] [BCDA 15 22 16]

- Nhận xét: Vòng Một dùng hàm F_1 với giá trị
 - t từ 1..16 và
 - k từ 0..15

Một Thao Tác MD5 - MD5

- Một thao tác MD5—MD5 bao gồm 64 tác vụ thế này, nhóm trong 4 vòng 16 tác vụ. F là một hàm phi tuyêt; một hàm được dùng trong mỗi vòng. M_i chỉ ra một khối tin nhập vào 32-bit, và K_i chỉ một hằng số 32-bit, khác nhau cho mỗi tác vụ.
- $\lll s$ chỉ sự xoay bit về bên trái đơn vị; s thay đổi tùy theo từng tác vụ. \boxplus chỉ cộng thêm với modulo 2^{32} .



Vòng 2

- **[abcd k s t]** là bước thực hiện của phép toán

$$a = b + ((a + F_2(b, c, d) + X[k] + T[i]) \lll s)$$

/* Do the following 16 operations. */
[ABCD 1 5 17] [DABC 8 9 10] [CDAB 13 14 19] [BCDA 0 20 20]
[ABCD 5 5 21] [DABC 10 8 22] [CDAB 15 14 23] [BCDA 4 20 24]
[ABCD 9 5 25] [DABC 14 9 26] [CDAB 5 14 27] [BCDA 8 20 28]
[ABCD 13 4 29] [DABC 2 6 30] [CDAB 9 14 31] [BCDA 12 20 32]

- Nhận xét: Vòng Một dùng hàm F_2 với giá trị
 - t từ 17..32 và
 - $k = (1..5t) \bmod 16$

Vòng 3

- [abcd k s t] là bước thực hiện của phép toán
 $a = b + ((a + F_3(b, c, d) + X[k] + T[i]) <<< s)$

```
/* Do the following 16 operations. */
[ABCD  5  4 33] [DABC  8 11 34] [CDAB 11 16 35] [BCDA 14 23 36]
[ABCD  1  4 37] [DABC  4 11 38] [CDAB  7 16 39] [BCDA 10 23 40]
[ABCD 13  4 41] [DABC  0 11 42] [CDAB  3 16 43] [BCDA  6 23 44]
[ABCD  9  4 45] [DABC 12 11 46] [CDAB 15 16 47] [BCDA  2 23 48]
```

- Nhận xét: Vòng Một dùng hàm F_2 với giá trị
 - t từ 33..48 và
 - $k = (5+3t) \bmod 16$

Vòng 4

- [abcd k s t] là bước thực hiện của phép toán
 $a = b + ((a + F_4(b, c, d) + X[k] + T[i]) <<< s)$

```
/* Do the following 16 operations. */
[ABCD  0  6 49] [DABC  7 10 50] [CDAB 14 15 51] [BCDA  5 21 52]
[ABCD 12  6 53] [DABC  3 10 54] [CDAB 10 15 55] [BCDA  1 21 56]
[ABCD  8  6 57] [DABC 15 10 58] [CDAB  6 15 59] [BCDA 13 21 60]
[ABCD  4  6 61] [DABC 11 10 62] [CDAB  2 15 63] [BCDA  9 21 64]
```

- Nhận xét: Vòng Một dùng hàm F_2 với giá trị
 - t từ 49..64 và
 - $k = 7t \bmod 16$

/* Sau đó làm các phép cộng sau. (Nghĩa là cộng vào mỗi thanh ghi giá trị của nó trước khi vào vòng lặp) */

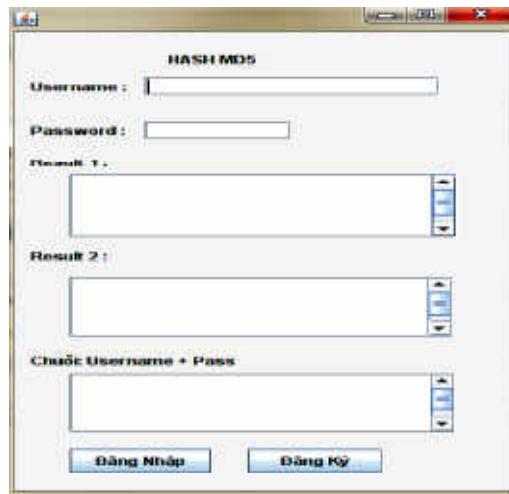
```
A = A + AA
B = B + BB
C = C + CC
D = D + DD
end /* of loop on i */
```

Bước 4: In ra

Mã số thông điệp được tạo ra là A,B,C,D. Nghĩa là chúng ta bắt đầu từ byte thấp của A, kết thúc với byte cao của D.

III. THUẬT TOÁN

❖ Thiết Kế Form



❖ Thư Viện cần sử dụng

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.security.MessageDigest;
import javax.swing.JOptionPane;
```

❖ Viết hàm xử lý sự kiện

- Xử lý sự kiện “Đăng Ký”

```
private void bntDangKyActionPerformed(java.awt.event.ActionEvent evt) {  
    try {  
        // TODO add your handling code here:  
        String user = txtuser.getText();  
        String pass = txtpass.getText();  
        String bam = "";  
        bam = user + pass;  
        MessageDigest md = MessageDigest.getInstance("MD5");  
        md.update(bam.getBytes());  
        byte[] byteData = md.digest();  
        //convert the byte to hex format method 1  
        StringBuffer sb = new StringBuffer();  
        for (int i = 0; i < byteData.length; i++) {  
            sb.append(Integer.toHexString((byteData[i] & 0xff) + 0x100, 16).substring(1));  
        }  
        System.out.println("Digest(in hex format):: " + sb.toString());  
        txtbam1.setText(sb.toString());  
        //convert the byte to hex format method 2  
        StringBuffer hexString = new StringBuffer();  
        for (int i = 0; i < byteData.length; i++) {  
            String hex = Integer.toHexString(0xff & byteData[i]);  
            if (hex.length() == 1) {  
                hexString.append('0');  
            }  
            hexString.append(hex);  
        }  
        System.out.println("Digest(in hex format):: " + hexString.toString());  
        txtbam2.setText(hexString.toString());  
        txtgoc.setText(bam.toString());  
        //Viết chức năng ghi File  
        BufferedWriter bw = null;  
        //ghi van ban da ma hoa  
        String fileName = "D:\\BamMD5.txt";  
        //luu van ban  
        bw = new BufferedWriter(new FileWriter(fileName));  
        // ghi van ban  
        bw.write(hexString.toString());  
        bw.close();  
        JOptionPane.showMessageDialog(null, "Bạn Đã Đăng Ký Thành Công .Vui lòng Đăng nhập lại !!!");  
  
    } catch (Exception ex) {  
        System.out.println("Loi bam username và password :" + ex);  
    }  
}
```

- Xử lý sự kiện đăng nhập

```

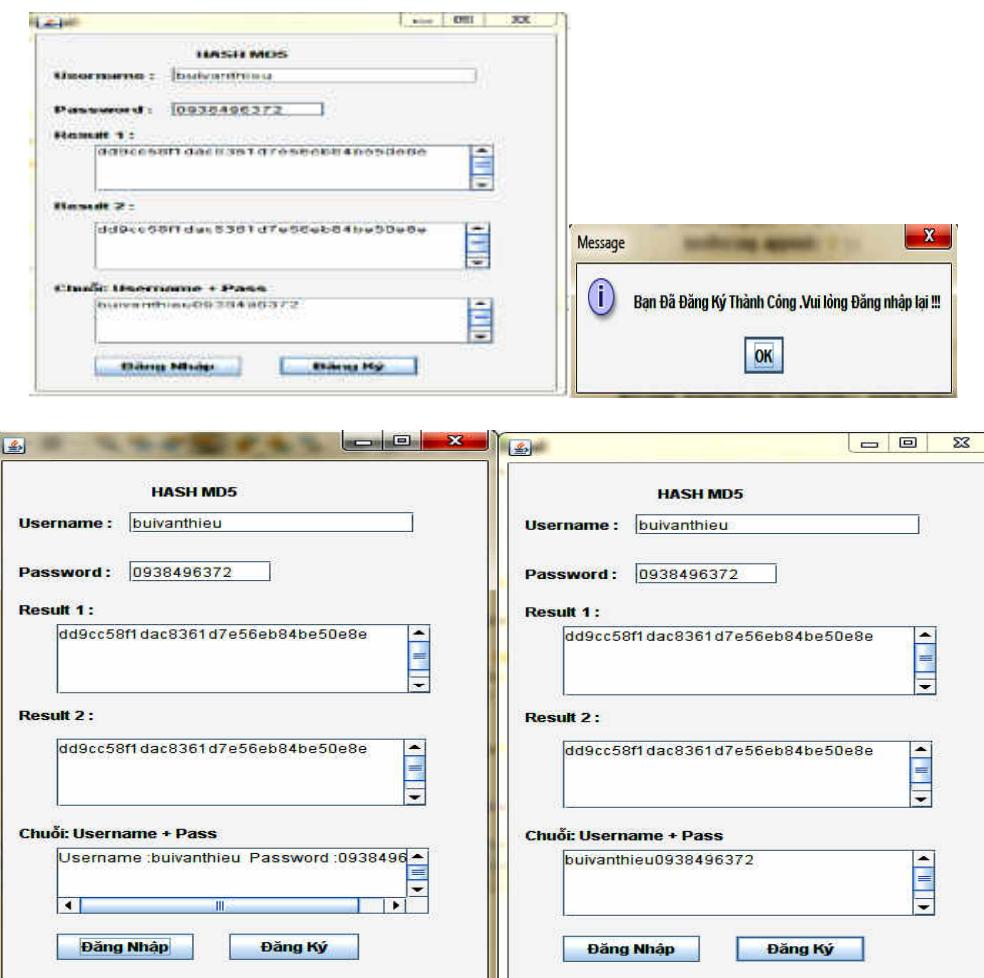
private void bntDangNhapActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    String user = txtuser.getText();
    String pass = txtpass.getText();
    String bam = "";
    bam = user + pass;
    //mở File đã lưu username và password
    BufferedReader br = null;
    String fileName = "D:\\\\BamMD5.txt"; //GEN-
    try{
        br = new BufferedReader( new FileReader(fileName));
        StringBuffer sb = new StringBuffer();
        char[] ca = new char[5];
        while (br.ready()) {
            int len = br.read(ca);
            sb.append(ca, 0, len);
        }
        br.close();
        //hiển thị File đã lưu
        System.out.println("chung thuc :" + " " + sb);
        String chuoi = sb.toString();

        // Thực hiện băm username và pass cho người dùng đăng nhập
        MessageDigest md = MessageDigest.getInstance("MD5");
        md.update(bam.getBytes());
        byte[] byteData = md.digest();
        StringBuffer hexString = new StringBuffer();
        for (int i = 0; i < byteData.length; i++) {
            String hex = Integer.toHexString(0xff & byteData[i]);
            if (hex.length() == 1) {
                hexString.append('0');
            }
            hexString.append(hex);
        }
        System.out.println("Băm username và password :" + " " + hexString.toString());
        // Thực hiện so sánh username và password
        Boolean k=hexString.toString().equals(chuoi);
        if(k==true)
        {
            JOptionPane.showMessageDialog(null, " Đăng Nhập Thành Công");
            // hiển thi username và password bị băm
            txtbam1.setText(hexString.toString());

            //hiển thi mã băm lưu ở File
            txtbam2.setText(chuoi);
            txtgoc.setText("Username :" +user+ " " + " Password :" + pass );
        }
        else
            JOptionPane.showMessageDialog(null, " Đăng Nhập Thất bại");
    } catch (Exception ex) {
        System.out.println(" Lỗi Đăng Nhập :" + ex);
    }
}

```

- Kiểm tra kết quả



Bài 2: Hiện thực thuật toán hàm băm SHA với yêu cầu sau:

Nhập chuỗi và sử dụng thuật toán SHA băm chuỗi theo 2 cách.

❖ Thiết kế Frame:

Chương Trình Hiện Thực Hàm Băm SHA

Nhập Chuỗi	<input style="width: 85%; height: 30px; background-color: #00ff00; color: black; font-size: 14pt; border: none;" type="text"/>
Hash SHA C1	<input style="width: 100%; height: 50px; border: none;" type="text"/>
Hash SHA C2	<input style="width: 100%; height: 50px; border: none;" type="text"/>
<input style="width: 15%; border: 1px solid black; border-radius: 5px; padding: 2px 10px;" type="button" value="BămSHA"/> <input style="width: 15%; border: 1px solid black; border-radius: 5px; padding: 2px 10px;" type="button" value="Thoát"/>	

❖ Viết hàm xử lý sự kiện

- Xử lý sự kiện button BămSHA

```
private void bntbamSHAActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        String chuoi = "";
        chuoi = txtchuoi.getText();
        MessageDigest md = MessageDigest.getInstance("SHA-256");
        md.update(chuoi.getBytes());
        byte byteData[] = md.digest();
        StringBuffer sb = new StringBuffer();
        for (int i = 0; i < byteData.length; i++) {
            sb.append(Integer.toHexString((byteData[i] & 0xff) + 0x100, 16).substring(1));
        }

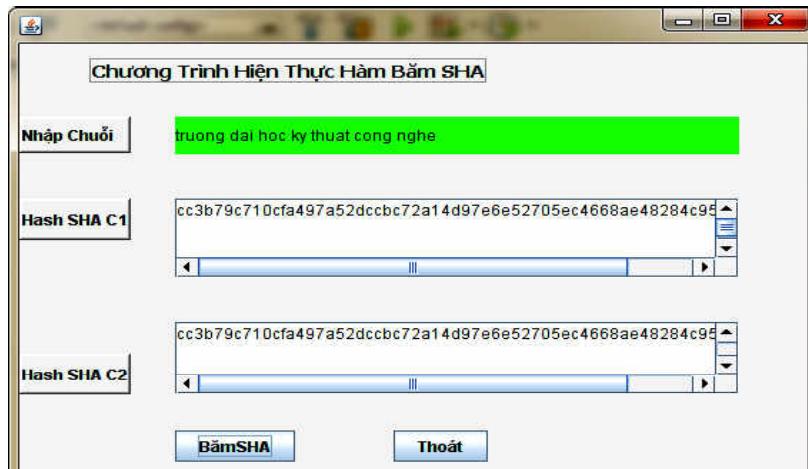
        System.out.println("Hex format1 : " + sb.toString());
        txtsha1.setText(sb.toString());

        //convert the byte to hex format method 2
        StringBuffer hexString = new StringBuffer();
        for (int i=0;i<byteData.length;i++) {
            String hex=Integer.toHexString(0xff & byteData[i]);
            if(hex.length()==1) hexString.append('0');
            hexString.append(hex);
        }
        System.out.println("Hex format2 : " + hexString.toString());
        txtsha2.setText(hexString.toString());
    } catch (NoSuchAlgorithmException ex) {
        Logger.getLogger(FrameSHA.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

- Xử lý sự kiện thoát Form

```
private void bntthoatActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    System.exit(WIDTH);
}
```

- Kết quả



Bài thực hành số 8

8*8

Viết chương trình với yêu cầu sau:

- Cho phép người dùng nhập username và password.
- Dùng thuật toán MD5 băm username, password và lưu vào File.
- Người dùng đăng nhập vào trong hệ thống sau đó nhập nội dung văn bản, văn bản sẽ được đính kèm đoạn mã xác thực (chữ ký số) và lưu xuống File.

❖ Digital Signature (chữ ký số)

1. Định nghĩa

Chữ ký số là một cơ chế xác thực cho phép người tạo ra thông tin gắn thêm một đoạn mã đặc biệt vào thông tin, có tác dụng như một chữ ký. Chữ ký được tạo ra bằng cách áp dụng một hàm băm lên thông tin gốc tạo thành message digest(MAC), sau đó mã hóa MAC với khóa private của người gửi. Chữ ký số tích hợp cùng văn bản gốc được gửi cho bên nhận. Bên nhận dùng khóa public (của người gửi) giải mã chữ ký số và nhận được MAC đối chứng. Người nhận có thể chắc chắn rằng văn bản mình nhận chỉ có thể do người gửi tạo ra vì chỉ có người gửi mới có khóa private cùng cặp với khóa public dùng giải mã.

Mặc dù đã có cơ chế xác thực thông tin (message authentication) thực hiện chức năng xác thực nguồn gốc thông tin, ta vẫn phải dùng chữ ký số. Các cơ chế xác thực thông tin sử dụng các hàm băm một chiều có tác dụng bảo vệ thông tin trao đổi giữa hai thực thể thông tin khỏi sự xâm phạm của một thực thể thứ 3, tuy nhiên nó không có tác dụng ngăn chặn được sự xâm phạm của chính hai thực thể. Ví dụ:

- Thực thể A gửi một bản tin X cho thực thể B sử dụng một cơ chế xác thực nào đó, cơ chế này đảm bảo chỉ có A và B dùng chung khóa bí mật K để tạo ra các mã xác thực từ thông tin gốc. Tuy nhiên, thực thể B có thể đổi bản tin X thành một bản tin Y, và với khóa bí mật K, thực thể B hoàn toàn có thể tạo ra thông tin xác thực mới để gắn vào Y, làm cho nó trở thành một bản tin hợp lệ mặc dù thực chất đây không phải là bản tin do thực thể A tạo ra.

- Thực thể A có thể từ chối xác nhận việc mình đã gửi bản tin X cho thực thể B, vì cơ chế xác thực trên, thực thể B toàn hoàn có khả năng giả mạo thông tin đưa ra từ thực thể A.

2. Thuộc tính-chức năng-phân loại-phương pháp thực hiện

Giống như một chữ ký thông thường (chữ ký bằng tay), một chữ ký số phải có đầy đủ các thuộc tính sau đây:

- Phải xác nhận chính xác người ký và ngày giờ phát sinh chữ ký.
- Phải xác thực nội dung thông tin ngay tại thời điểm phát sinh chữ ký.
- Phải có khả năng cho phép kiểm chứng bởi một người thứ 3 để giải quyết các tranh chấp nếu có.

Như vậy, chức năng của chữ ký số là xác thực thông tin. Các yêu cầu đối với chữ ký số:

- Là một chuỗi bit phát sinh từ khối thông tin cần được xác nhận.
- Chữ ký phải chứa thông tin nhận dạng riêng của người ký để tránh giả mạo và tránh phủ nhận.
- Quy trình tạo ra chữ ký cũng như xác minh chữ ký phải đơn giản, nhanh chóng.
- Chữ ký không thể bị giả mạo bằng bất cứ cách nào.
- Có thể sao chép một bản sao của chữ ký dành cho mục đích lưu trữ.

Có thể phân loại chữ ký số theo thuật toán phát sinh chữ ký số:

- Chữ ký cố định và chữ ký ngẫu nhiên: thuật toán tạo chữ ký cố định(deterministic) tạo ra một chữ ký duy nhất ứng với một khối thông tin xác định nghĩa là nếu thực hiện nhiều lần thuật toán tạo chữ ký trên một bản tin thì vẫn cho ra một kết quả duy nhất. Ngược lại, chữ ký ngẫu nhiên tạo ra những chữ ký khác nhau đối với cùng một bản tin.
- Chữ ký phục hồi được và chữ ký không phục hồi được: cơ chế tạo chữ ký phục hồi được cho phép người nhận phục hồi lại thông tin gốc từ chữ ký, điều này cũng có nghĩa là chữ ký phải có chứa thông tin gốc thong nó dưới một dạng mã hóa nào đó, và kết quả là chữ ký số sẽ có kích thước lớn hơn thông tin gốc. Khi đó, người

gởi chỉ cần gởi đi chữ ký là đủ. Do vậy, cơ chế tạo chữ ký này cũng còn được gọi là chữ ký khôi phục bản tin. Ngược lại, cơ chế tạo chữ ký không phục hồi được không cho phép hồi phục thông tin gốc từ chữ ký, do vậy, chữ ký chỉ là một khối thông tin cộng thêm có kích thước nhỏ hơn thông tin gốc. Người gửi cần phải gởi chữ ký đi kèm với thông tin gốc như một dạng phục lục, do đó cơ chế tạo chữ ký này cũng còn được gọi là chữ ký với phụ lục.

Có hai phương pháp thực hiện chữ ký số là ký trực tiếp và ký thông qua trọng tài.

- Ký trực tiếp (direct signature): Ở phương pháp này, giả thiết rằng phía nhận biết được khóa public của phía gửi. Do đó, chữ ký có thể được tạo ra bằng cách mã hóa toàn bộ bản tin bằng khóa private của người tạo ra thông tin, hoặc chỉ mã hóa phần mã băm dùng khóa riêng của người tạo thông tin. Để đạt được tính bảo mật của thông tin thì thông tin gốc cùng với chữ ký vừa được tạo ra sẽ được mã hóa sử dụng khóa public của thực thể nhận chữ ký (trong trường hợp dùng mật mã bất đối xứng) hoặc dùng khóa bí mật(trong trường hợp mật mã đối xứng).

Một nhược điểm rất dễ thấy của phương thức ký trực tiếp đó là độ an toàn của chữ ký phụ thuộc cao vào khóa private của người tạo ra chữ ký. Do vậy, nếu khóa private này bị mất hoặc bị tiết lộ thì ý nghĩa của chữ ký số sẽ không còn.

- Ký thông qua trọng tài(arbitrated signature): đây là một giải pháp được xây dựng để khắc phục nhược điểm của chữ ký trực tiếp. Khi thực thể A muốn gởi một bản tin cho thực thể B, quá trình tạo ra một chữ ký được thực hiện bình thường như đối với chữ ký trực tiếp. Tuy nhiên, trước khi bản tin này được gởi đến B, nó phải được gởi đến một thực thể thứ 3 gọi là trọng tài. Trọng tài thực hiện việc kiểm tra, xác nhận tính chính xác của thông tin và chữ ký, sau đó ghi lại ngày giờ rồi mới gởi cho thực thể B, kèm theo thông tin xác nhận của trọng tài. Sự xuất hiện của trọng tài trong quy trình đảm bảo được thực thể A sẽ không phủ nhận được thông tin mình đã gởi.

3. Chuẩn chữ ký DSS

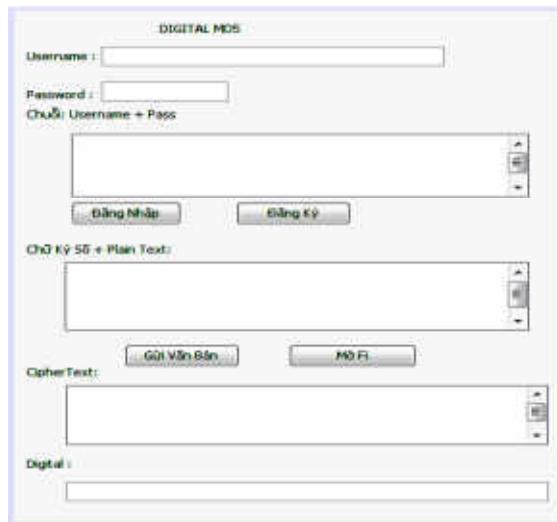
DSS (Digital Signature Standard) là một chuẩn về chữ ký số, được chuẩn hóa năm 1991, sửa đổi năm 1993 và 1996, sau đó mở rộng vào năm 2000. DSS sử dụng hàm

băm SHA và thuật toán tạo chữ ký DSA (Digital Signature Algorithm). DSS thuộc loại chữ ký ngẫu nhiên và không phục hồi được.

Trong thuật toán xác thực thông tin dùng mật mã RSA, thông tin gốc được đưa vào hàm băm SHA để tạo ra mã băm có kích thước cố định. Mã băm này sau đó được mã hóa (bằng thuật toán RSA) dùng khóa private của thực thể tạo thông tin (phía gửi). Kết quả cho phép mã hóa được gắn vào thông tin gốc và gửi đi. Phía nhận nhận được thông tin, tách phần mã băm ra khỏi thông tin gốc và giải mã nó bằng khóa public của phía gửi. Chú ý rằng khóa public là thông tin được công bố rộng rãi cho bất kỳ thực thể nào có quan tâm. Đồng thời, thông tin gốc cũng được đưa vào hàm băm để tính mã băm, sau đó đem so sánh với mã băm vừa nhận được. Nếu hai mã này giống nhau thì thông tin vừa nhận được chấp nhận như là thông tin hợp lệ.

❖ BÀI TẬP

1. Thiết kế Form



2. Thư Viện cần sử dụng

2.1. Thư viện cho Frame DIGITAL_MD5

1. import digital.ThuVien.Keys;
2. import java.io.BufferedReader;
3. import java.io.BufferedWriter;
4. import java.io.FileReader;
5. import java.io.FileWriter;
6. import java.security.MessageDigest;

7. import javax.swing.JOptionPane;

```
public class DIGITAL_MD5 extends javax.swing.JFrame {
    Keys keys = new Keys();
    int blockSize = 4;
    String user;
    String pass;
    ThuVien obj = new ThuVien();
    // String signedMsg;
    /** Creates new form MD5HASH */
    public DIGITAL_MD5() {
        initComponents();
    }
}
```

2.2. Tạo Class thư viện có tên ThuVien

```
import java.math.BigInteger;
/* @author nguoituyet

public class ThuVien {

    static class Keys{
        BigInteger n = new BigInteger("9617");
        BigInteger d = new BigInteger("3869");
        BigInteger e = new BigInteger("17");
        }//end inner class Keys
        String encode(String message)
        {
            byte[] textChars = message.getBytes();
            String temp = "";
            String encodedMsg = "";
            //Build the encoded text string two numeric
            // characters at a time. Each message
            // character is converted into two numeric
            // characters according to the relationships
            // given above.
            for(int cnt = 0; cnt < message.length();cnt++){
                temp = String.valueOf(textChars[cnt] - ' ');
                //Convert all single-character numeric
                // values to two characters with a leading
                // zero, as in 09.
                if(temp.length() < 2) temp = "0" + temp;
                encodedMsg += temp;    }//end for loop
                return encodedMsg;
        }//end encode
}
```

```
//-----
```

```
String decode(String encodedMsg){  
    String temp = "";  
    String decodedText = "";  
    for(int cnt = 0; cnt < encodedMsg.length(); cnt += 2){  
        temp = encodedMsg.substring(cnt,cnt + 2);  
        //Convert two numeric text characters to a  
        // value of type int.  
        int val = Integer.parseInt(temp) + 32;  
        //Convert the ASCII character values to  
        // numeric String values and build the  
        // output String one character at a time.  
        decodedText += String.valueOf((char)val);  
    }  
    return decodedText;  
}
```

```
String doRSA(String inputString,BigInteger exp,BigInteger n,int blockSize){  
    BigInteger block;  
    BigInteger output;  
    String temp = "";  
    String outputString = "";  
    //Iterate and process one block at a time.  
    for(int cnt = 0; cnt < inputString.length();cnt += blockSize){  
        //Get the next block of characters  
        // and encapsulate them in a BigInteger  
        // object.  
        temp = inputString.substring(cnt,cnt + blockSize);  
        block = new BigInteger(temp);  
        //Raise the block to the power exp, apply  
        // the modulus operand n, and save the  
        // remainder. This is the essence of the  
        // RSA algorithm.  
        output = block.modPow(exp,n);  
        //Convert the numeric result to a  
        // four-character string, appending leading  
        // zeros as necessary.
```

```
temp = output.toString();
while(temp.length() < blockSize){
    temp = "0" + temp;
}
//Build the outputString blockSize
// characters at a time. Each character
// in the inputString results in one
// character in the outputString.
outputString += temp;
}
return outputString;
}
}
```

3. Viết hàm xử lý sự kiện cho Frame DIGITAL_MD5

```
public class DIGITAL_MD5 extends javax.swing.JFrame {
    Keys keys = new Keys();
    int blockSize = 4;
    ThuVien obj = new ThuVien();
    // String signedMsg;
    /** Creates new form MD5HASH */
    public DIGITAL_MD5() {
        initComponents();
    }
}
```

3.1. Xử lý sự kiện "Đăng Ký"

```
private void bntDangKyActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        // TODO add your handling code here:
        String user = txtuser.getText();
        String pass = txtpass.getText();
        String bam = "";
        bam = user + pass;
        MessageDigest md = MessageDigest.getInstance("MD5");
        md.update(bam.getBytes());
        byte[] byteData = md.digest();
        //convert the byte to hex
```

```
StringBuffer hexString = new StringBuffer();
for (int i = 0; i < byteData.length; i++) {
    String hex = Integer.toHexString(0xff & byteData[i]);
    if (hex.length() == 1) {
        hexString.append('0');
    }
    hexString.append(hex);
}
System.out.println("Digest(in hex format):: " + hexString.toString());
txtgoc.setText(hexString.toString());
//Viết chức năng ghi File
BufferedWriter bw = null;
//ghi van ban da ma hoa
String fileName = "D:\\BamMD5.txt";
//luu van ban
bw = new BufferedWriter(new FileWriter(fileName));
// ghi van ban
bw.write(hexString.toString());
bw.close();
JOptionPane.showMessageDialog(null, "Bạn Đã Đăng Ký Thành Công.Vui
lòng Đăng nhập lại !!!");
} catch (Exception ex) {
System.out.println(" Loi bam username và password:" + ex);
}
}
```

3.2. Xử lý sự kiện đăng nhập

```
private void bntDangNhapActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
    user = txtuser.getText();  
    pass = txtpass.getText();  
    String bam = "";  
    bam = user + pass;  
    //mở File đã lưu username và password  
    BufferedReader br = null;  
    String fileName = "D:\\BamMD5.txt"; //GEN-  
    try{  
        br = new BufferedReader( new FileReader(fileName));  
        StringBuffer sb = new StringBuffer();  
        char[] ca = new char[5];  
        while (br.ready()) {  
            int len = br.read(ca);  
            sb.append(ca, 0, len);  
        }  
        br.close();  
        //hiển thi File đã lưu  
        System.out.println("chung thuc :" + " " + sb);  
        String chuoit = sb.toString();  
        // Thực hiện băm username và pass cho người dùng đăng nhập  
        MessageDigest md = MessageDigest.getInstance("MD5");  
        md.update(bam.getBytes());  
        byte[] byteData = md.digest();  
        StringBuffer hexString = new StringBuffer();  
        for (int i = 0; i < byteData.length; i++) {  
            String hex = Integer.toHexString(0xff & byteData[i]);  
            if (hex.length() == 1) {  
                hexString.append('0');  
            }  
            hexString.append(hex);  
        }  
        System.out.println("Bam username và password :" + " " + hexString.toString());  
        // Thực hiện so sánh username và password  
        Boolean k=hexString.toString().equals(chuoit);  
        if(k==true)  
        {  
            JOptionPane.showMessageDialog(null, " Đăng Nhập Thành Công");  
            // hiển thi username và password  
            txtgoc.setText("Username :" +user );  
        }  
        else  
            JOptionPane.showMessageDialog(null, " Đăng Nhập Thất bại");  
    } catch (Exception ex) {  
        System.out.println(" Lỗi Đăng Nhập :" + ex);  
    }  
}
```

3.3. Xử lý sự kiện Gửi Văn Bản

```

private void bntguivabanActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    //mo file dang ky user
    BufferedReader br = null;
    String fileName = "D:\\BamMDS.txt"; //GEN-
    try{
        br = new BufferedReader( new FileReader(fileName));
        StringBuffer sb = new StringBuffer();
        char[] ca = new char[5];
        while (br.ready()) {
            int len = br.read(ca);
            sb.append(ca, 0, len);
        }
        br.close();
        //hiển thị File đã lưu
        System.out.println("chung thuc :" + " " + sb);
        //lay ma bam user+ pass
        String message = sb.toString();
        // khai tao cac tham so RSA
        System.out.println("Alice's keys:");
        System.out.println("e: " + keys.e);
        System.out.println("d: " + keys.d);
        System.out.println("n: " + keys.n);
        //xu ly 1 lan 4 ky tu
        while(message.length()% (blockSize/2) != 0){
            //Append a visible character on the end.
            message += "-";
        }
        //end while
        System.out.println("chu ky doc :\n"+ message);
        String vanban= txtvanban.getText();

        // ma hoa
        String encodedMsg = obj.encode(message);
        String signature = obj.doRSA(encodedMsg,keys.d, keys.n,blockSize);
        System.out.println("Digital signature\n"+ signature);
        String signedMsg = vanban+"_"+ message + "_" + signature + user;
        System.out.println("Van Ban + Digital+ xac thuc:\n" + signedMsg);
        txtcipher.setText(signedMsg);
        // Ghi File

        BufferedWriter bw = null;
        //ghi van ban da ma hoa
        String fileNam1 = "D:\\Digital.txt";
        //luu van ban
        bw = new BufferedWriter(new FileWriter(fileNam1));
        // ghi van ban
        bw.write(signedMsg);
        bw.close();
        JOptionPane.showMessageDialog(null," Đã ghi file");
    }catch(Exception ex){}
}

```

3.4. Xử lý sự kiện Mở File

```

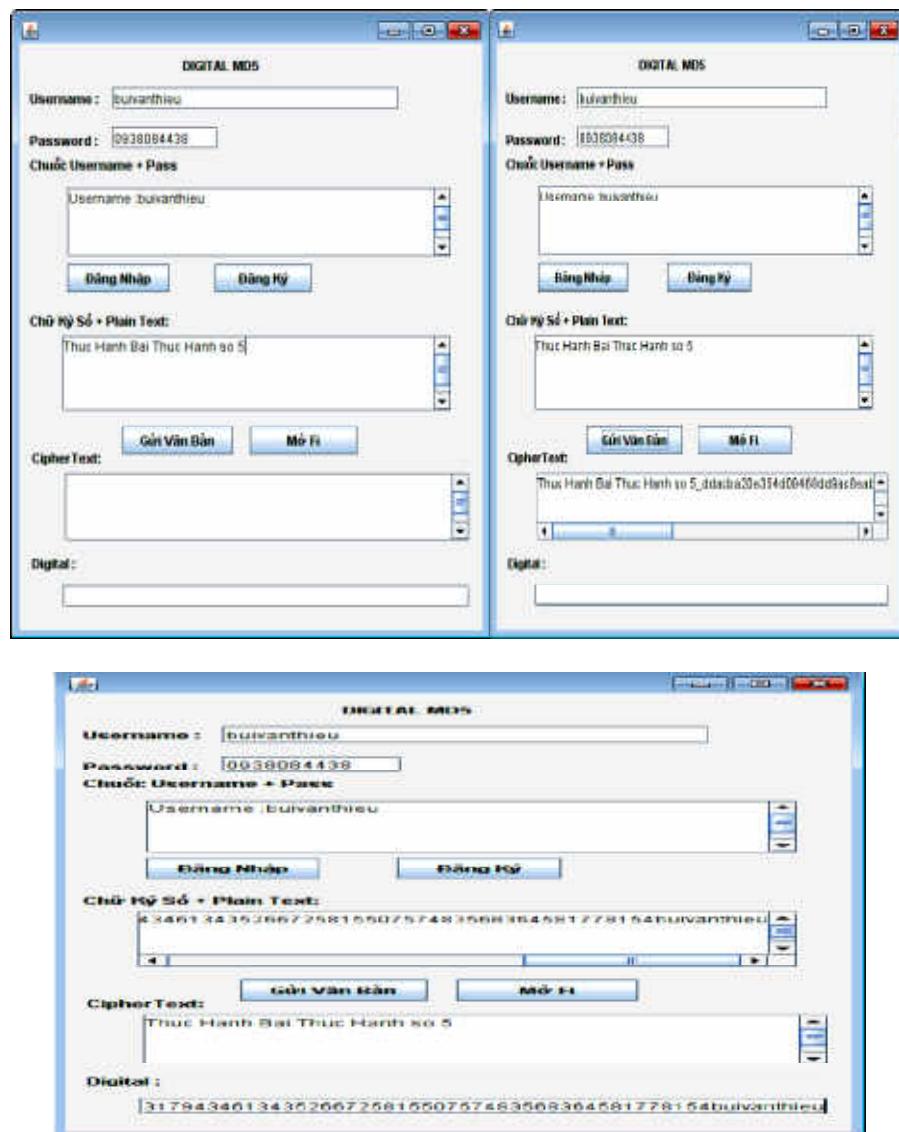
private void bntmovanbanActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    try{
        BufferedReader br = null;
        String fileName = "D:\\Digital.txt";
        br = new BufferedReader(new FileReader(fileName));
        StringBuffer sb = new StringBuffer();
        JOptionPane.showMessageDialog(null, " Đã mở file");
        char[] ca = new char[5];
    }
}

```

```
        while (br.ready()) {
            int len = br.read(ca);
            sb.append(ca, 0, len);
        }
        br.close();
        String chuoi = sb.toString();
        txtvanban.setText(chuoi);
        String signedMsg1=chuoi;
        String extractedMsgText =signedMsg1.substring(0,signedMsg1.indexOf('_'));
        System.out.println("Bob's Van Ban Goc:\n" + extractedMsgText);
        txtcipher.setText(extractedMsgText);
        String extractedSignature = signedMsg1.substring(signedMsg1.indexOf('_') + 1);
        System.out.println( "Bob's Digital signature:\n" + extractedSignature);
        txtdigital.setText(extractedSignature);
        String decipheredSignature =obj.doRSA(extractedSignature,keys.e,keys.n,blockSize);
        String decodedSignature = obj.decode(decipheredSignature);
        System.out.println("Bob's Decoded digital signature:\n" + decodedSignature);
        if(extractedMsgText.equals(decodedSignature))
        {
            System.out.println("KO: Valid signature");
        }
        else{
            System.out.println("KO: Invalid signature");
       }//end else }//end main

    }catch(Exception ex){}
}
```

1. Kiểm Tra Kết Quả

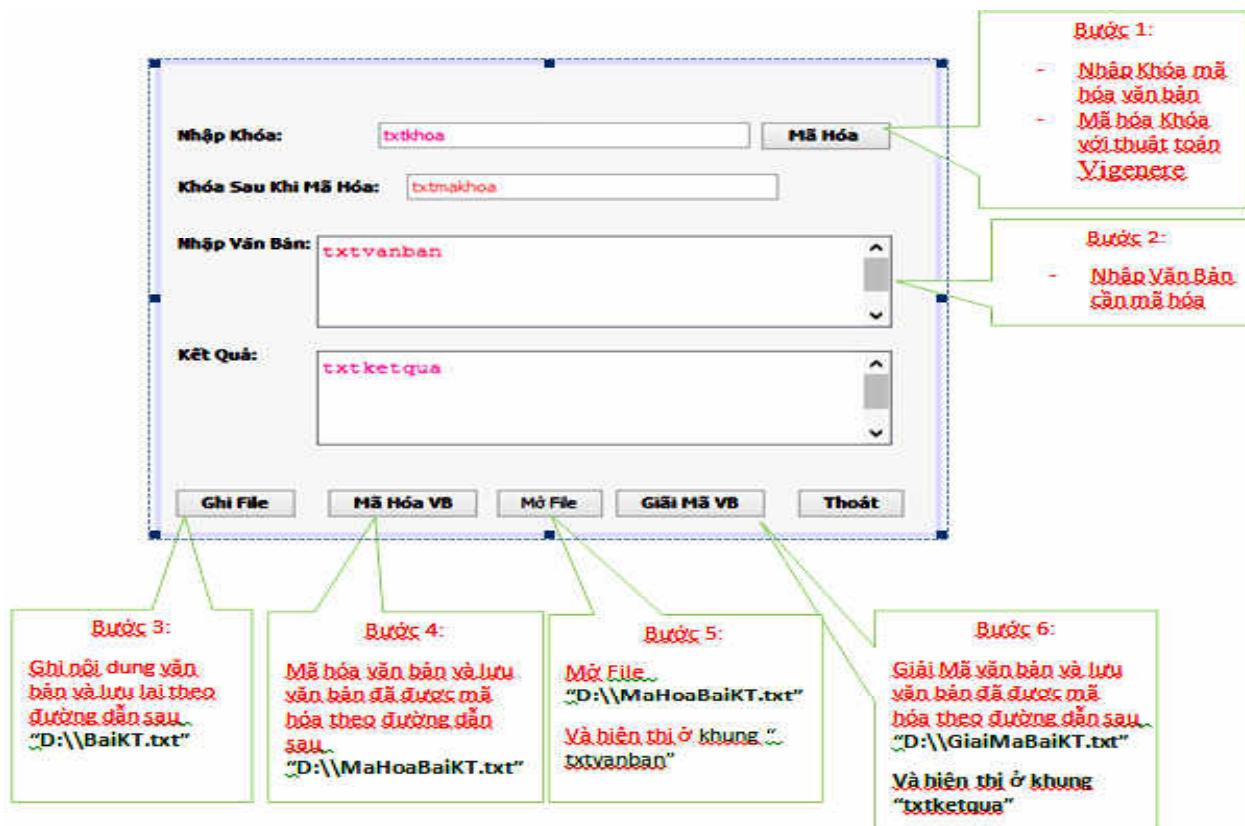


Bài thực hành số 9

8*8

Bài 1: Viết chương trình mã hóa và giải mã văn bản với yêu cầu sau:

- Khóa phải được mã hóa với thuật toán Vigenere.
- Văn bản phải được mã hóa với thuật toán DES.

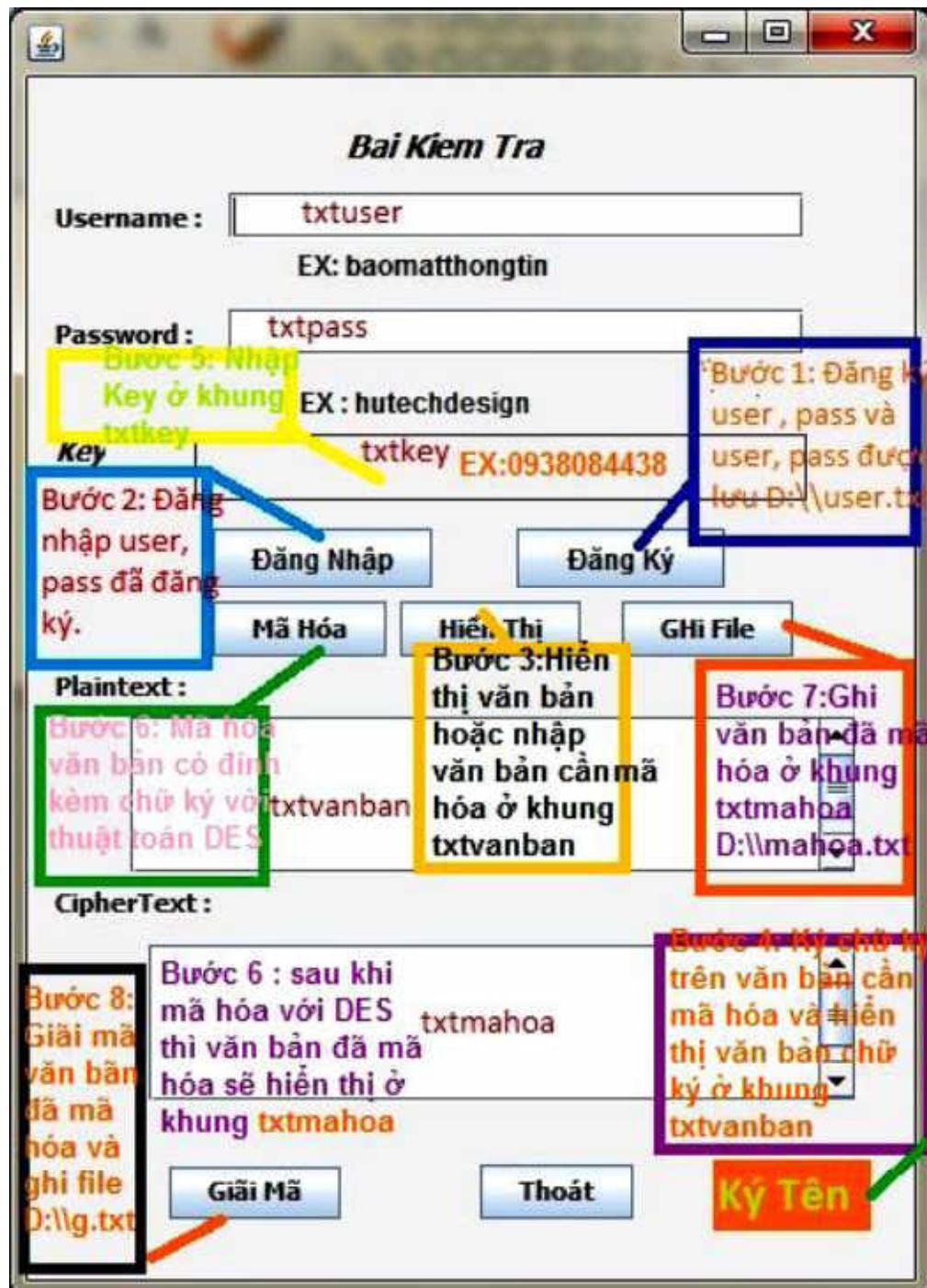


Bài 2: Viết chương trình thực hiện mã hóa và giải mã dữ liệu với các yêu cầu sau:

- Cho phép người dùng nhập và đăng ký username và password, username và password phải được băm với thuật toán MD5 và được lưu vào ổ đĩa C hoặc D với tên bam.txt (D:\\user.txt), mã băm này được dùng để ký vào văn bản.
- Người dùng đăng nhập vào phần mềm và dùng chức năng hiển thị để mở văn bản cần mã hóa hoặc gõ trực tiếp vào khung Plaintext.
- Văn bản phải được ký với mã băm trên (user.txt) trước khi được mã hóa và giải mã với thuật toán DES/3DES.

- Khi mã hóa/giải mã thành công, văn bản đã mã hóa/giải mã cần lưu với tên và đường dẫn sau D:\\mahoa.txt (.txt,.doc,,.dat,...), D:\\vanban.txt.

❖ Các bước thực hiện

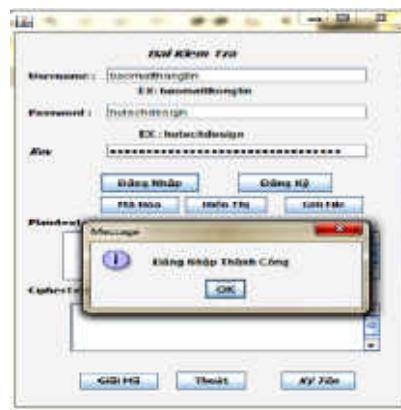


❖ Hướng dẫn chi tiết

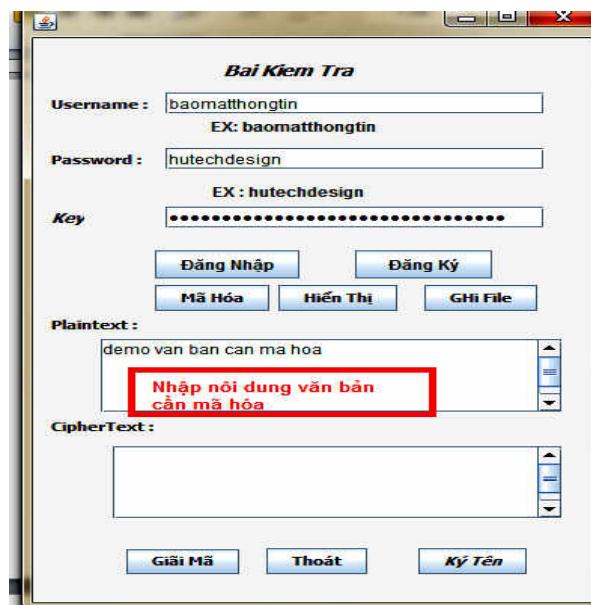
- B1: Xử lý sự kiện Đăng Ký



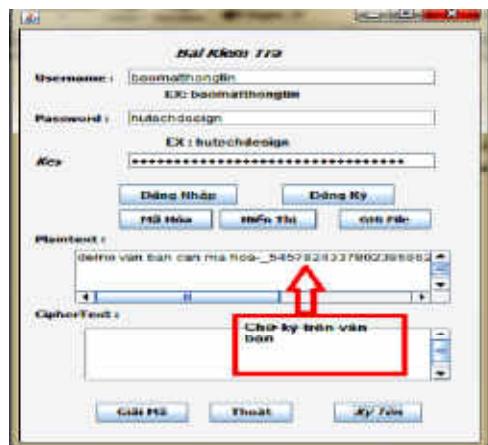
- B2: Xử lý sự kiện Đăng Nhập



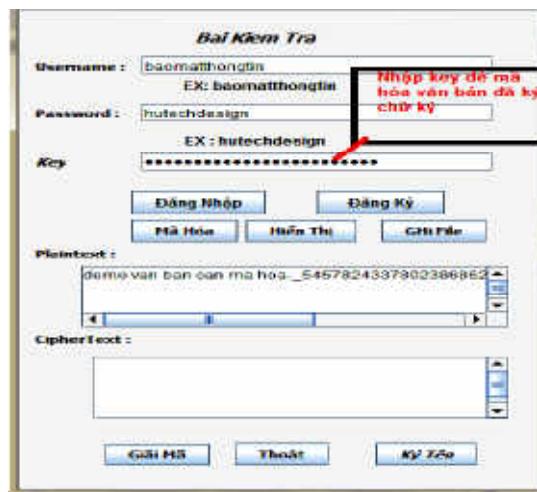
- B3: Nhập nội dung văn bản cần mã hóa ở ô txtvanban



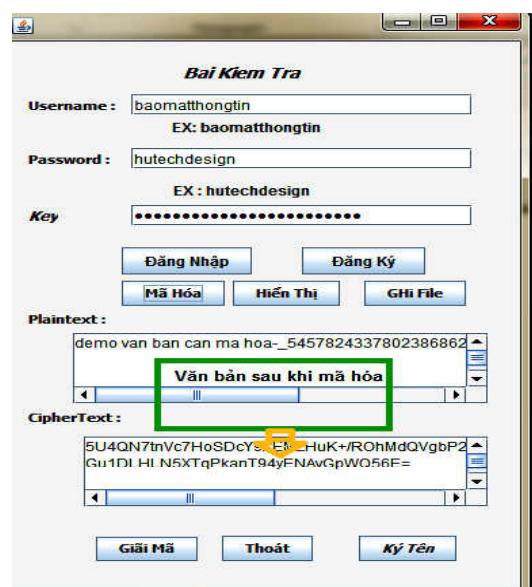
- B4: Xử lý sự kiện ký tên để tạo chữ ký trên văn bản



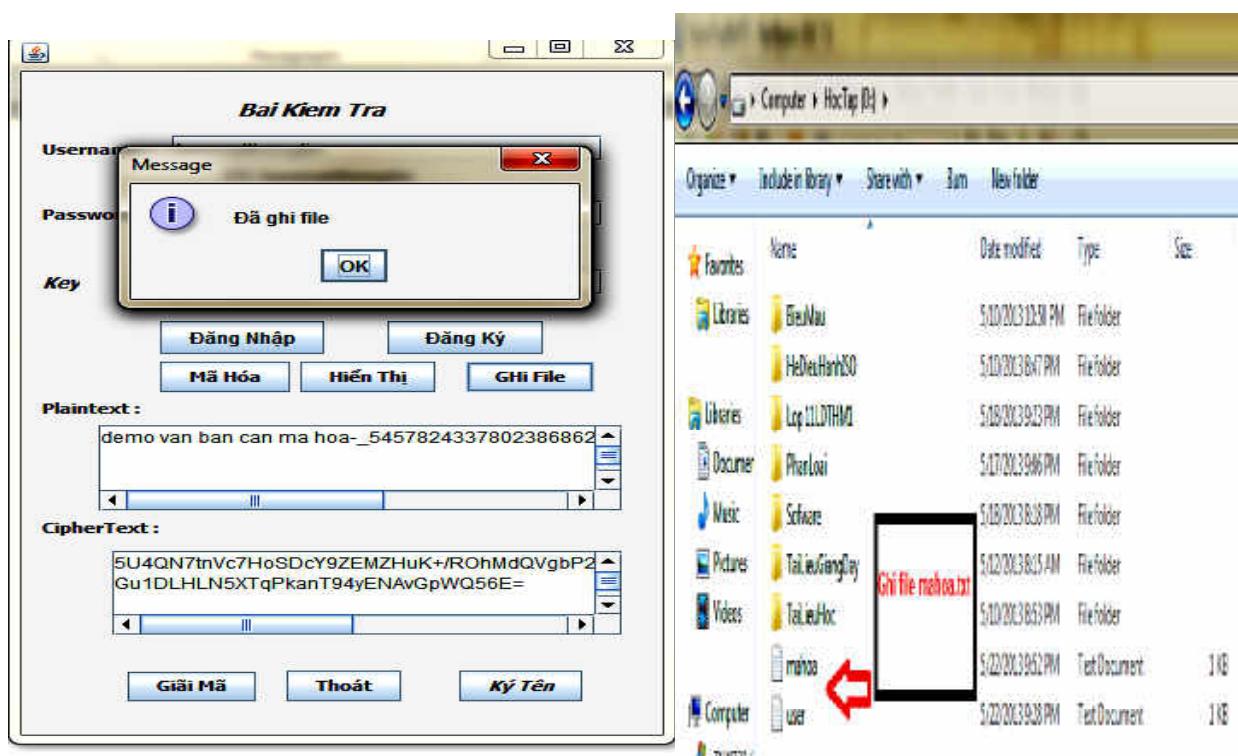
- B5: Nhập key ở khung txtkey để mã hóa văn bản



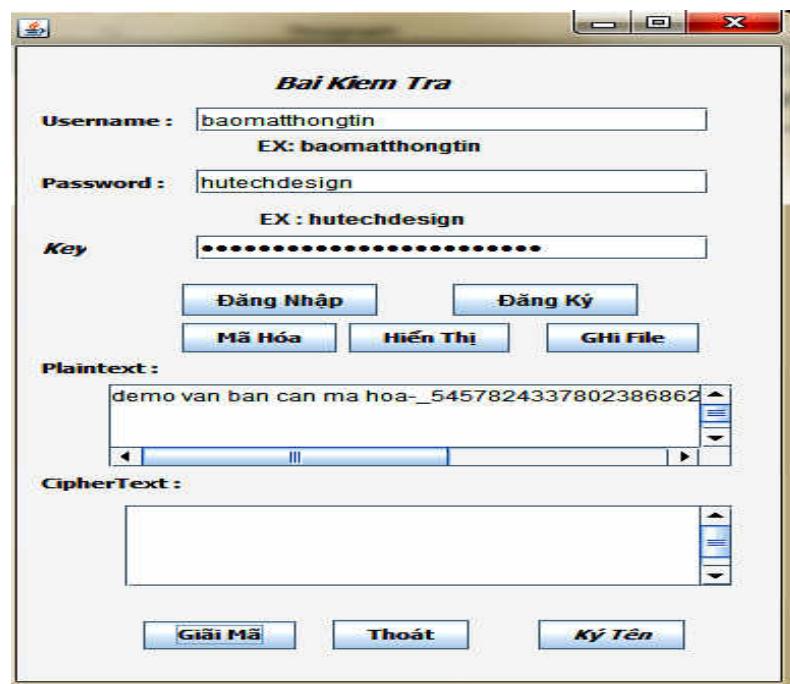
- B6: Xử lý sự kiện mã hóa văn bản



- B7: Ghi file văn bản đã mã hóa mahoa.txt



- B8: Giải mã văn bản



TÀI LIỆU THAM KHẢO

1. Giáo trình Bảo mật thông tin (TS. Trần Văn Dũng)
2. Sách: Bảo mật thông tin – mô hình và ứng dụng (Nguyễn Xuân Dũng).
3. Information Security Principles and Practices, Mark Stamp, John Wiley&Son, Inc, 2006.
4. Bảo mật thông tin, mô hình và ứng dụng, Nguyễn Xuân Dũng, Nhà xuất bản Thống Kê, 2007.
5. J. Michael Stewart, Network Security, Firewalls, and VPNs.
6. William Stallings, Network Security Esentials: Applications and Standards.
7. Comptia Security+, Deluxe study guide, Sybex, Wiley Publising, Inc.