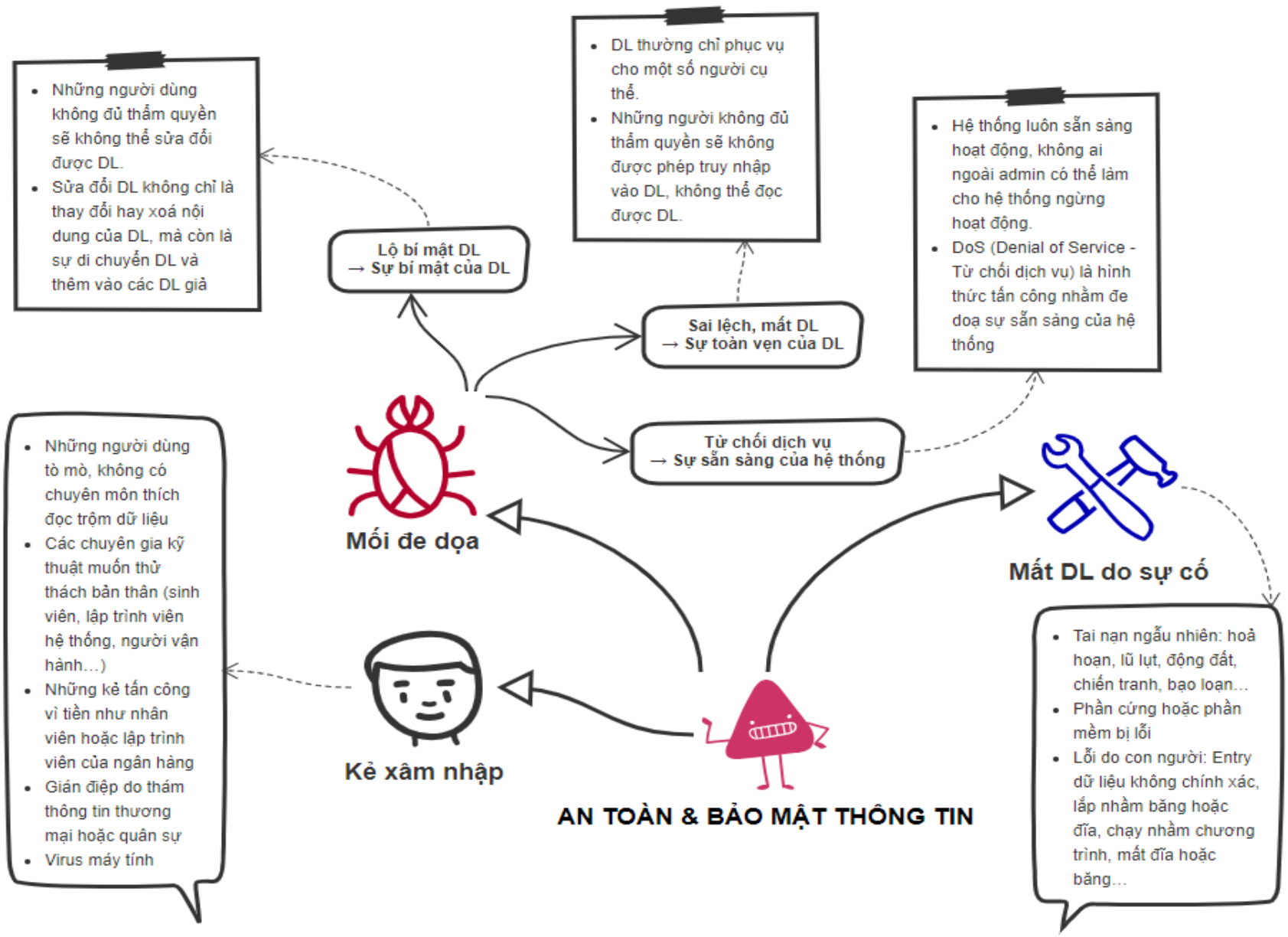
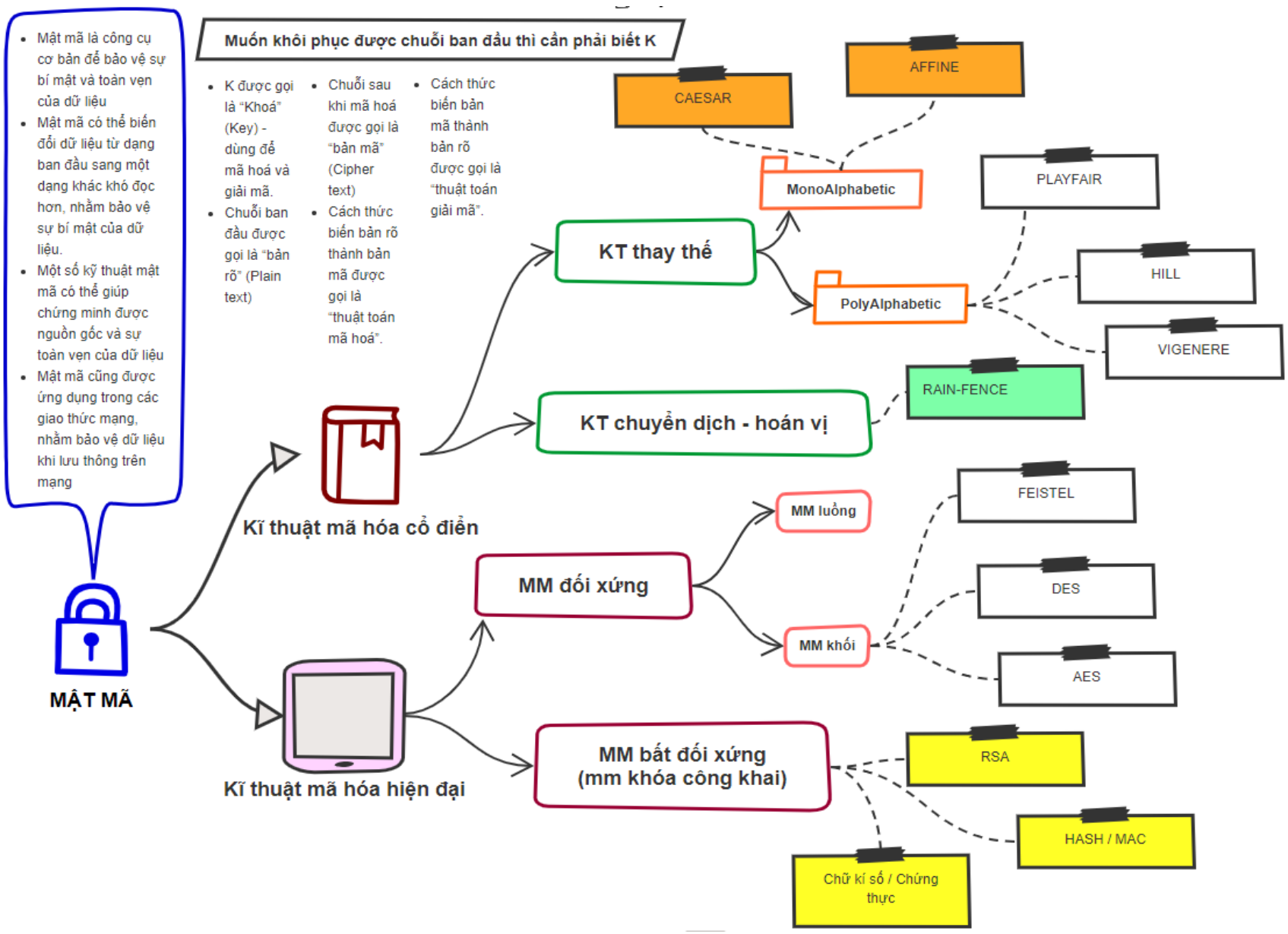
Mục Lục

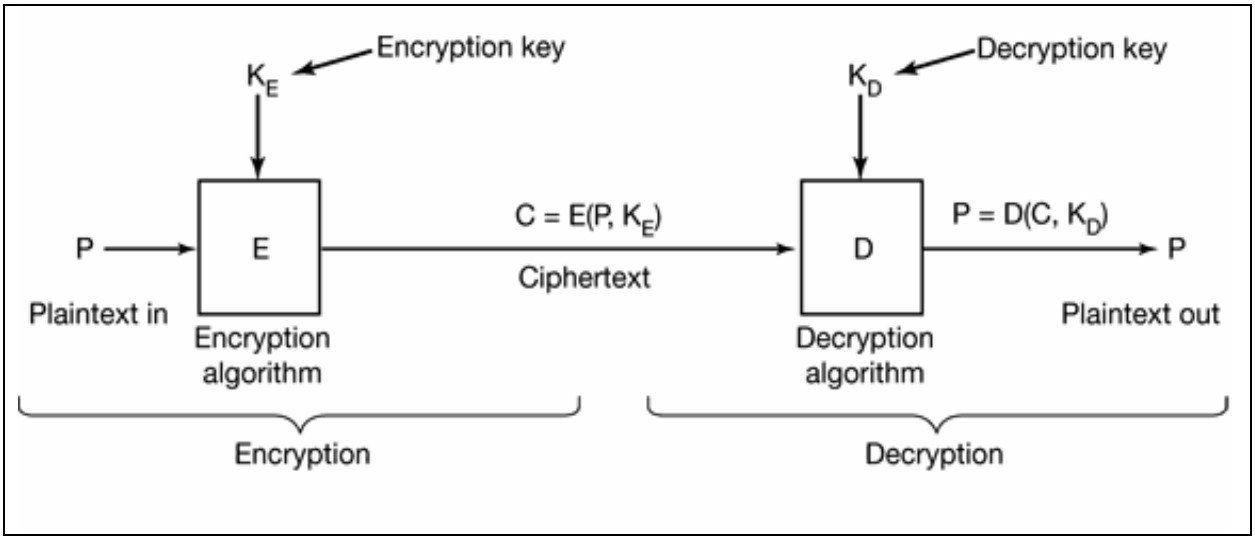
BẢNG MÃ ASCII

Mã thập phân	Mã hexa	Mã nhị phân	Kí tự	Mã thập phân	Mã hexa	Mã nhị phân	Kí tự
000	000	00000000	NUL	064	040	01000000	@
001	001	00000001	SOH	065	041	01000001	A
002	002	00000010	STX	066	042	01000010	B
003	003	00000011	ETX	067	043	01000011	C
004	004	00000100	EOT	068	044	01000100	D
005	005	00000101	ENQ	069	045	01000101	E
006	006	00000110	ACK	070	046	01000110	F
007	007	00000111	BEL	071	047	01000111	G
008	008	00001000	BS	072	048	01001000	H
009	009	00001001	HT	073	049	01001001	I
010	00A	00001010	LF	074	04A	01001010	J
011	00B	00001011	VT	075	04B	01001011	K
012	00C	00001100	FF	076	04C	01001100	L
013	00D	00001101	CR	077	04D	01001101	M
014	00E	00001110	SO	078	04E	01001110	N
015	00F	00001111	SI	079	04F	01001111	O
016	010	00010000	DLE	080	050	01010000	P
017	011	00010001	DC1	081	051	01010001	Q
018	012	00010010	DC2	082	052	01010010	R
019	013	00010011	DC3	083	053	01010011	S
020	014	00010100	DC4	084	054	01010100	T
021	015	00010101	NAK	085	055	01010101	U
022	016	00010110	SYN	086	056	01010110	V
023	017	00010111	ETB	087	057	01010111	W
024	018	00011000	CAN	088	058	01011000	X
025	019	00011001	EM	089	059	01011001	Y
026	01A	00011010	SUB	090	05A	01011010	Z
027	01B	00011011	ESC	091	05B	01011011	[
028	01C	00011100	FS	092	05C	01011100	\
029	01D	00011101	GS	093	05D	01011101	]
030	01E	00011110	RS	094	05E	01011110	^
031	01F	00011111	US	095	05F	01011111	_
032	020	00100000	SP	096	060	01100000	`
033	021	00100001	!	097	061	01100001	a
034	022	00100010	"	098	062	01100010	b
035	023	00100011	#	099	063	01100011	c
036	024	00100100	$	100	064	01100100	d
037	025	00100101	%	101	065	01100101	e
038	026	00100110	&	102	066	01100110	f
039	027	00100111	'	103	067	01100111	g
040	028	00101000	(	104	068	01101000	h
041	029	00101001	)	105	069	01101001	i
042	02A	00101010	*	106	06A	01101010	j
043	02B	00101011	+	107	06B	01101011	k
044	02C	00101100	,	108	06C	01101100	l
045	02D	00101101	-	109	06D	01101101	m
046	02E	00101110	.	110	06E	01101110	n
047	02F	00101111	/	111	06F	01101111	o
048	030	00110000	0	112	070	01110000	p
049	031	00110001	1	113	071	01110001	q
050	032	00110010	2	114	072	01110010	r
051	033	00110011	3	115	073	01110011	s
052	034	00110100	4	116	074	01110100	t
053	035	00110101	5	117	075	01110101	u
054	036	00110110	6	118	076	01110110	v
055	037	00110111	7	119	077	01110111	w
056	038	00111000	8	120	078	01111000	x
057	039	00111001	9	121	079	01111001	y
058	03A	00111010	:	122	07A	01111010	z
059	03B	00111011	;	123	07B	01111011	{
060	03C	00111100	<	124	07C	01111100	|
061	03D	00111101	=	125	07D	01111101	}
062	03E	00111110	>	126	07E	01111110	~
063	03F	00111111	?	127	07F	01111111	DEL


AN TOÀN & BẢO MẬT THÔNG TIN
Mối đe dọa
Kẻ xâm nhập
Mất DL do sự cố
Lộ bí mật DL
→ Sự bí mật của DL
Sai lệch, mất DL
→ Sự toàn vẹn của DL
Từ chối dịch vụ
→ Sự sẵn sàng của hệ thống
Những người dùng không đủ thẩm quyền sẽ không thể sửa đổi được DL.
Sửa đổi DL không chỉ là thay đổi hay xoá nội dung của DL, mà còn là sự di chuyển DL và thêm vào các DL giả
DL thường chỉ phục vụ cho một số người cụ thể.
Những người không đủ thẩm quyền sẽ không được phép truy nhập vào DL, không thể đọc được DL.
Hệ thống luôn sẵn sàng hoạt động, không ai ngoài admin có thể làm cho hệ thống ngừng hoạt động.
DoS (Denial of Service - Từ chối dịch vụ) là hình thức tấn công nhằm đe doạ sự sẵn sàng của hệ thống
Những người dùng tò mò, không có chuyên môn thích đọc trộm dữ liệu
Các chuyên gia kỹ thuật muốn thử thách bản thân (sinh viên, lập trình viên hệ thống, người vận hành…)
Những kẻ tấn công vì tiền như nhân viên hoặc lập trình viên của ngân hàng
Gián điệp do thám thông tin thương mại hoặc quân sự
Virus máy tính
Tai nạn ngẫu nhiên: hoả hoạn, lũ lụt, động đất, chiến tranh, bạo loạn…
Phần cứng hoặc phần mềm bị lỗi
Lỗi do con người: Entry dữ liệu không chính xác, lắp nhầm băng hoặc đĩa, chạy nhầm chương trình, mất đĩa hoặc băng…


Mật mã là công cụ cơ bản để bảo vệ sự bí mật và toàn vẹn của dữ liệu
Mật mã có thể biến đổi dữ liệu từ dạng ban đầu sang một dạng khác khó đọc hơn, nhằm bảo vệ sự bí mật của dữ liệu.
Một số kỹ thuật mật mã có thể giúp chứng minh được nguồn gốc và sự toàn vẹn của dữ liệu
Mật mã cũng được ứng dụng trong các giao thức mạng, nhằm bảo vệ dữ liệu khi lưu thông trên mạng
MẬT MÃ
Muốn khôi phục được chuỗi ban đầu thì cần phải biết K
K được gọi là "Khoá" (Key) - dùng để mã hoá và giải mã.
Chuỗi ban đầu được gọi là "bản rõ" (Plain text)
Chuỗi sau khi mã hoá được gọi là "bản mã" (Cipher text)
Cách thức biến bản rõ thành bản mã được gọi là "thuật toán mã hoá".
Cách thức biến bản mã thành bản rõ được gọi là "thuật toán giải mã".
Kĩ thuật mã hóa cổ điển
KT thay thế
CAESAR
AFFINE
MonoAlphabetic
PolyAlphabetic
PLAYFAIR
HILL
VIGENERE
KT chuyển dịch - hoán vị
RAIN-FENCE
Kĩ thuật mã hóa hiện đại
MM đối xứng
MM luồng
MM khối
FEISTEL
DES
AES
MM bất đối xứng (mm khóa công khai)
RSA
HASH / MAC
Chữ kí số / Chứng thực

Quá trình mã hoá và giải mã



Trong đó:

- P: bản rõ
- K_E: khoá mã hoá (Encryption Key)
- C: bản mã
- E: thuật toán mã hoá (hay hàm mã hoá)
- $C = E(P, K_E)$: công thức định nghĩa sự mã hoá. Bản mã được tạo ra nhờ áp dụng thuật toán mã hoá E, tác động vào bản rõ P, với khoá mã hoá K_E làm tham số
- K_D: khoá giải mã (Decryption Key)
- D: thuật toán giải mã (hay hàm giải mã)
- $P = D(C, K_D)$: công thức định nghĩa sự giải mã. Thuật toán D sẽ tác động lên bản mã C với K_D là tham số, nó sẽ biến bản mã C trở về dạng ban đầu là bản rõ P

Note:

- Nếu $K_D = K_E$ (tức là khoá giải mã và khoá mã hoá giống nhau)
  → *"mật mã đối xứng"*. Khoá này phải được giữ bí mật.
- Nếu $K_D \neq K_E$
  → *"mật mã bất đối xứng"*.

MẬT MÃ MONOALPHABETIC

- Quy tắc thay thế:
  Mỗi kí tự trong bảng chữ cái được thay thế bởi một kí tự bất kì khác cùng bảng

Ưu điểm	Nhược điểm
Số lượng khoá rất lớn, khó bẻ khoá bằng phương pháp Brute –force: ▪ Số lượng hoán vị của chuỗi dài 26 chữ cái là: ***26! ≈ 4.1026 (trên 4.1026 khoá!)*** ▪ Giả sử một máy tính cá nhân tốc độ 4 GHz (thực hiện 4 tỷ phép tính/1 giây) có thể kiểm tra được 1 khoá trong 1 phép tính. ▪ Để kiểm tra hết 4.1026 khoá sẽ cần tới: $\frac{4.10^{26}}{4.10^{9}.3600.24.365} \approx 3\ tỉ\ năm$	Có thể bẻ khoá mật mã này dựa trên các thống kê về các đặc điểm tự nhiên của ngôn ngữ

- Bảng thống kê xác suất xuất hiện của các kí tự trong tiếng Anh

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

- Chia 26 chữ cái thành 5 nhóm sau:
  - ***E ~ 0.127***
  - ***T, A, O, I, N, S, H, R ~ 0.060 đến 0.090***
  - ***D, L ~ 0.04***
  - ***C, U, M, W, F, G, Y, P, B ~ 0.015 đến 0.028.***
  - ***V, K, J, X, Q, Z ~ 0.01***
- 30 cặp kí tự tiếng Anh có xác suất xuất hiện cao nhất (từ cao xuống thấp)
  TH, HE, IN, ER, AN, RE, ED, ON, ES, ST, EN, AT, TO, NT, HA, ND, OU, EA, NG, AS, OR, TI, IS, ET, IT, AR, TE, SE, HI, OF
- 12 bộ 3 kí tự tiếng Anh có xác suất xuất hiện cao nhất (từ cao xuống thấp)
  THE, ING, AND, HER, ERE, ENT, THA, NTH, WAS, ETH, FOR, DTH

MẬT MÃ CAESAR

- Đây là một trong những mật mã ra đời sớm nhất, do Julius Caesar phát minh.
- Quy tắc thay thế:
  Mỗi kí tự trong bảng chữ cái được thay thế bởi một kí tự khác cùng bảng, cách sau nó ba vị trí.
  - Plaintext:
    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
  - Ciphertext:
    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
- **Ta gán:**

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

→ Như vậy, mọi phép toán mã hoá và giải mã đều được thực hiện trong một tập hợp 26 số nguyên từ 0 đến 25 (gọi là "Vành 26" hay Z_{26}).
Công thức mã hoá:

$$C = E(P, 3) = (P + 3) \bmod 26$$

Công thức giải mã:

$$P = D(C, 3) = (C - 3) \bmod 26$$

- Tổng quát hoá mật mã Caesar bằng cách thay thế một kí tự ban đầu bởi kí tự đứng sau nó *K* vị trí.
- **Công thức mã hoá:**

$$\boldsymbol{C = E(P, K) = (P + K) \bmod 26}$$

- **Công thức giải mã:**

$$\boldsymbol{P = D(C, K) = (C - K) \bmod 26}$$

Ưu điểm	**Nhược điểm**
▪ Đơn giản, dễ thực hiện	▪ Độ an toàn không cao, dễ bị bẻ khoá bởi tấn công Brute-force do số lượng khoá quá ít (chỉ có 25 khoá)

(Brute-force là hình thức tấn công bằng cách thử tất cả các khả năng của khoá để tìm ra khoá đúng)

MẬT MÃ AFFINE

- **Công thức mã hóa:**

$$C = E(P, \{a,b\}) = (aP + b) \bmod 26$$

- **Công thức giải mã:**

$$P = D(C, \{a,b\}) = a^{-1}(C - b) \bmod 26$$

Trong đó:

- P: kí tự ban đầu
- C: kí tự thay thế
- K: cặp 2 số nguyên $\{a,b\}$ thuộc Z_{26}

- **Note:**
  - Nếu a =1 thì mật mã Affine sẽ trở thành mật mã Caesar tổng quát
  - Để có thể giải mã được thì a phải nguyên tố với 26, tức là ước số chung lớn nhất của a và 26 bằng 1: $\textit{ƯCLN(a, 26) = 1}$
  - a^{-1} là một số thuộc Z_{26} thoả mãn:

$$a.a^{-1} = a^{-1}.a = 1 \text{ (trong } Z_{26})$$

  - VD:
    Các giá trị của a:
    1, 3, 5, 7, 9, 11, 15, 17, 19, 21, 23, 25
    Các a-1 tương ứng:
    1, 9, 21, 15, 3, 19, 7, 23, 11, 5, 17, 25

Ưu điểm	Nhược điểm
Độ phức tạp lớn hơn mật mã Caesar tổng quát, số lượng khoá cũng nhiều hơn	Độ an toàn chưa cao, dễ bị phá bởi tấn công Brute-force do số lượng khoá chưa nhiều (chỉ có 312 khoá): ▪ Có 12 giá trị của a ▪ Có 26 giá trị của b ▪ Tổng cộng có 12 x 26 = 312 cặp số {a,b}

MẬT MÃ POLYALPHABETIC

Mỗi kí tự plain text có thể được thay thế bởi nhiều kí tự khác nhau, dựa trên các khoá thay thế khác nhau.

MẬT MÃ PLAYFAIR

- Được phát minh bởi nhà khoa học người Anh - Charles Wheatstone - năm 1854
- Được sử dụng rộng rãi trong quân đội Anh, Mỹ và đồng minh trong chiến tranh thế giới thứ II
- Quy tắc thay thế:
  Thay thế từng cặp 2 kí tự trong bản rõ bởi 2 kí tự tương ứng trong ma trận khoá 5 x 5.
- Công thức mã hóa:
  - Lần lượt viết từng kí tự của plaintext vào ma trận, từ trái sang phải, từ trên xuống dưới, bỏ các kí tự trùng lặp
  - Viết các ký tự còn lại trong bảng chữ cái vào ma trận theo thứ tự, I và J được coi như một ký tự
  - **Mỗi ký tự trong cặp plaintext** sẽ được mã hoá bằng ký tự nằm **cùng hàng với nó**, nhưng **cùng cột với ký tự kia**
  - Nếu cặp ký tự plaintext rơi vào **cùng một hàng** của ma trận thì mỗi ký tự **được thay thế bởi ký tự bên phải** nó, **cùng một cột** của ma trận thì mỗi ký tự được **thay thế bởi ký tự ngay sát dưới**
  - Nếu ký tự plaintext rơi vào **hàng cuối cùng** thì ciphertext của nó là ký tự **cùng cột ở hàng đầu tiên**, **cột cuối cùng** thì ciphertext của nó là ký tự **cùng hàng ở cột đầu tiên**
  - Nếu hai kí tự trong plaintext giống nhau thì chúng sẽ được được cách ly bằng một ký tự đại diện, chẳng hạn là **x**

Ưu điểm	Nhược điểm
▪ Có không gian khoá lớn tương tự mật mã Monoalphabetic nên khó bẻ được khoá bằng phương pháp Brute –force ▪ Có khả năng che giấu một phần thông tin về tần suất xuất hiện các chữ cái, nhờ thực hiện mã hoá từng cặp hai kí tự	

MẬT MÃ HILL

- Được phát minh bởi nhà toán học Lester Hill vào năm 1929
- Quy tắc thay thế:
  *Thay thế từng nhóm **m** kí tự trong plaintext bởi **m** kí tự ciphertext*
- Công thức mã hóa:
  - Hệ pt:

$$C_1 = (k_{11}P_1 + k_{12}P_2 + \ldots + k_{1m}P_m) \bmod 26$$
$$C_2 = (k_{21}P_1 + k_{22}P_2 + \ldots + k_{2m}P_m) \bmod 26$$
$$C_3 = (k_{31}P_1 + k_{32}P_2 + \ldots + k_{3m}P_m) \bmod 26$$
$$\ldots$$
$$C_m = (k_{m1}P_1 + k_{m2}P_2 + \ldots + k_{mm}P_m) \bmod 26$$

  - Ma trận:

$$\begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} k_{11} & k_{12} & k_{13} \\ k_{21} & k_{22} & k_{23} \\ k_{31} & k_{32} & k_{33} \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix} \bmod 26$$

  - Công thức:

$$\mathbf{C} = \mathbf{KP} \bmod 26$$

  Trong đó:
  - C: ma trận cột biểu diễn ciphertext
  - P: ma trận cột biểu diễn plaintext
  - K: ma trận khoá
- Công thức giải mã:

$$\mathbf{P} = \mathbf{K^{-1}C} \bmod 26$$

  Trong đó:
  - K^{-1} là ma trận nghịch đảo của ma trận khoá K
    $\rightarrow K.K^{-1} = K^{-1}.K = I$ (I là ma trận đơn vị)

Ưu điểm	Nhược điểm
▪ Độ an toàn của Hill sẽ càng lớn khi sử dụng ma trận K càng lớn ▪ Có khả năng che dấu hoàn toàn tần suất xuất hiện các kí tự đơn	▪ Rất mạnh khi chống lại tấn công chỉ biết ciphertext, nhưng nó lại dễ dàng bị bẻ gãy với một tấn công biết plaintext, do có thể dễ dàng xác định ma trận K từ các cặp P-C đã biết

MẬT MÃ VIGENERE

- Quy tắc thay thế:
  *Thay thế từng nhóm **m** kí tự trong plaintext bởi **m** kí tự ciphertext*
- Công thức mã hóa:

$$C_1 = (P_1 + k_1) \bmod 26$$
$$C_2 = (P_2 + k_2) \bmod 26$$
$$C_3 = (P_3 + k_3) \bmod 26$$
$$\dots$$
$$C_m = (P_m + k_m) \bmod 26$$

- Công thức giải mã:

$$P_1 = (C_1 - k_1) \bmod 26$$
$$P_2 = (C_2 - k_2) \bmod 26$$
$$P_3 = (C_3 - k_3) \bmod 26$$
$$\dots$$
$$P_m = (C_m - k_m) \bmod 26$$

  Trong đó:
  - C_i: các kí tự ciphertext
  - P_i: các kí tự plaintext
  - k_i: các giá trị của khoá
- **VD:** Giả sử lấy *m=6* và khoá *K* là CIPHER, plaintext là: WEWILLMEETATMIDNIGHT. Hãy xác định ciphertext.
  - Do m=6, ta sẽ tách plaintext thành từng nhóm 6 kí tự:
    WEWILL/MEETAT/MIDNIG/HT
    Viết theo dạng số là: 22 4 22 8 11 11 / 12 4 4 19 0 19 / 12 8 3 13 8 6 / 7 19
  - Từ khoá CIPHER tương ứng với: *K* = (2, 8, 15, 7, 4, 17)
  - Cộng từng nhóm 6 kí tự của plaintext với *K* ta có:
    22 4 22 8 11 11/ 12 4 4 19 0 19/ 12 8 3 13 8 6/ 7 19
    2 8 15 7 4 17/ 2 8 15 7 4 17/ 2 8 15 7 4 17/ 2 8
    24 12 11 15 15 2 / 14 12 19 0 4 10/ 14 16 18 20 12 23/ 9 1
  - Ciphertext tương ứng là: YMLPPCOMTAEKOQSUMXJB

Ưu điểm	Nhược điểm
	▪ Không giấu được hoàn toàn tần suất xuất hiện các kí tự, và vẫn có thể bị phân tích → Giải pháp khắc phục là sử dụng hệ thống biểu diễn thông tin không mang tính thống kê của ngôn ngữ - đó là hệ nhị phân

MẬT MÃ VERNMAN

- Được phát minh bởi kỹ sư Gilbert Vernman của AT&T - năm 1918
- Hệ thống này làm việc trên dữ liệu nhị phân chứ không phải các ký tự.
- Quy tắc thay thế:
  - *Plaintext được biểu diễn dưới dạng một chuỗi bít nhị phân*
  - *Khoá K cũng được biểu diễn dưới dạng một chuỗi bít nhị phân (càng dài càng tốt, càng ngẫu nhiên càng tốt)*
  - *Ciphertext được sinh ra bởi phép XOR giữa plaintext với khóa K*
- Công thức mã hóa: $c_i = p_i \oplus k$
- Công thức giải mã: $p_i = c_i \oplus k_i$

  Trong đó:
  - p_i = bit thứ *i* của plaintext
  - k_i = bit thứ *i* của mật khóa *K*
  - $\oplus$ là phép XOR (cùng bit = 0, khác bit = 1)

Ưu điểm	Nhược điểm
▪ Với khoá *K* đủ dài và ngẫu nhiên, các thông tin mang tính thống kê của ngôn ngữ có thể được che giấu hoàn toàn ▪ Sự ra đời của mật mã hệ nhị phân là tiền đề cho sự ra đời của các mật mã hiện đại	▪ Nếu *K* ngắn thì sẽ phải sử dụng *K* lặp đi lặp lại, làm giảm tính ngẫu nhiên, và có thể làm lộ một phần thông tin về thống kê tần suất → Việc sinh ra được một khoá *K* dài và thực sự ngẫu nhiên như vậy sẽ đòi hỏi nhiều công sức

Một số phép toán thao tác với bít trong C

Phép toán	Trong C	Ví dụ
AND	&	a & b
OR	|	a | b
XOR	^	a ^ b
NOT	~	~a
Dịch trái	<<	a << 4
Dịch phải	>>	a >> 4

MẬT MÃ RAIN – FENCE

- Quy tắc thay thế:
  Plaintext được viết dịch xuống tuần tự theo các đường chéo rồi đọc trình tự theo các hàng
- VD: Giả sử plaintext là *"meet me after the toga party"*. Ciphertext:

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

Ưu điểm	Nhược điểm
	▪ Mật mã hoán vị thuần túy rất dễ nhận ra bởi nó giữ nguyên tần suất xuất hiện ký tự đơn (và làm thay đổi tần suất của các cặp, các bộ kí tự của plaintext) → Để tăng độ phức tạp, người ta có thể tiến hành đổi chỗ nhiều lần, hoặc kết hợp với các thuật toán mã hoá khác.

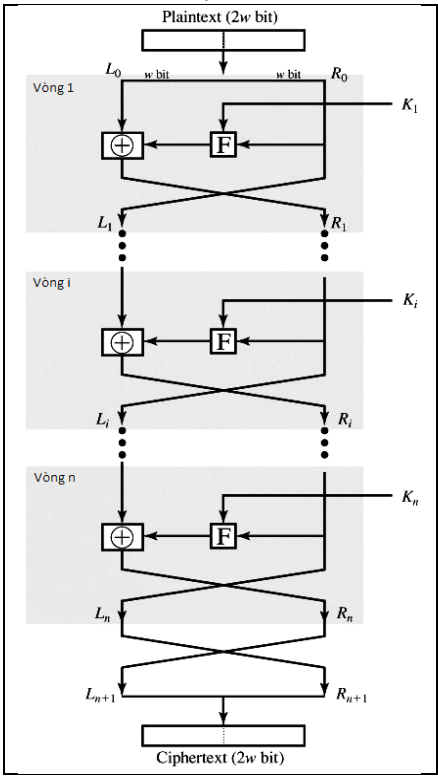
MẬT MÃ KHỐI

- Mật mã luồng (Stream cipher) là loại mật mã lần lượt mã hóa từng bit hay từng byte dữ liệu số của một luồng dữ liệu.
- Mật mã khối (Block cipher) mã hóa mỗi lần cả một khối plaintext và thường kết xuất ra một khối ciphertext có chiều dài như khối plaintext.
- Độ dài tiêu biểu của một khối thường là 64 bit hay 128 bit.

MẬT MÃ KHỐI FEISTEL

- Mật mã Feistel được coi là cơ sở của các mật mã khối hiện đại
- Quy tắc thay thế:
  - Đầu vào là một khối plaintext dài *m* bít và khoá *K* dài *k* bit
  - Sau đó liên tục tác động lên plaintext bằng các kỹ thuật thay thế và hoán vị, lặp lại nhiều lần, để thu được ciphertext ở đầu ra cũng có chiều dài *m* bit.
- Note:
  - Kỹ thuật thay thế có tác dụng làm *xáo trộn* thông tin thống kê của plaintext, làm phức tạp hóa mối quan hệ thống kê giữa ciphertext và mật khoá, nhằm ngăn cản nỗ lực tìm khoá.
  - Kỹ thuật hoán vị có tác dụng làm *khuếch tán* thông tin thống kê của plaintext, pha loãng các cấu trúc thống kê của plaintext ra một phạm vi rộng hơn, làm phức tạp hóa mối quan hệ thống kê giữa ciphertext và plaintext.

→ Có thể lặp lại phép thay thế và hoán vị nhiều lần để tăng độ phức tạp, nâng cao tính bảo mật



- Plaintext có chiều dài $m = 2w$ bit, được chia thành hai nửa, L_0 và R_0.
- Hai nửa dữ liệu đi qua n vòng xử lý rồi được tổ hợp lại thành khối ciphertext.
- Mỗi vòng xử lý i có các đầu vào L_{i-1} và R_{i-1} lấy từ vòng trước cùng với khóa con K_i, lấy từ khóa K gốc.
- Các khóa con K_i đều khác nhau và khác khóa K.
- Cấu trúc 1 vòng xử lí:
  - Đầu tiên, phép thay thế được thực hiện trên L_0, bằng cách XOR L_0 với $F(R_0, K_1)$. Hàm F là một phép biến đổi, càng phức tạp càng tốt
  - Sau đó thực hiện phép hoán vị bằng cách hoán đổi giá trị của hai nửa dữ liệu
  - Kết quả thu được: $R_1 = L_0 \oplus F(R_0, K_1)$
    $L_1 = R_0$
  - Lấy L_1, R_1 làm đầu vào của vòng thứ 2, lặp lại n vòng.

- Công thức tổng quát mỗi vòng:

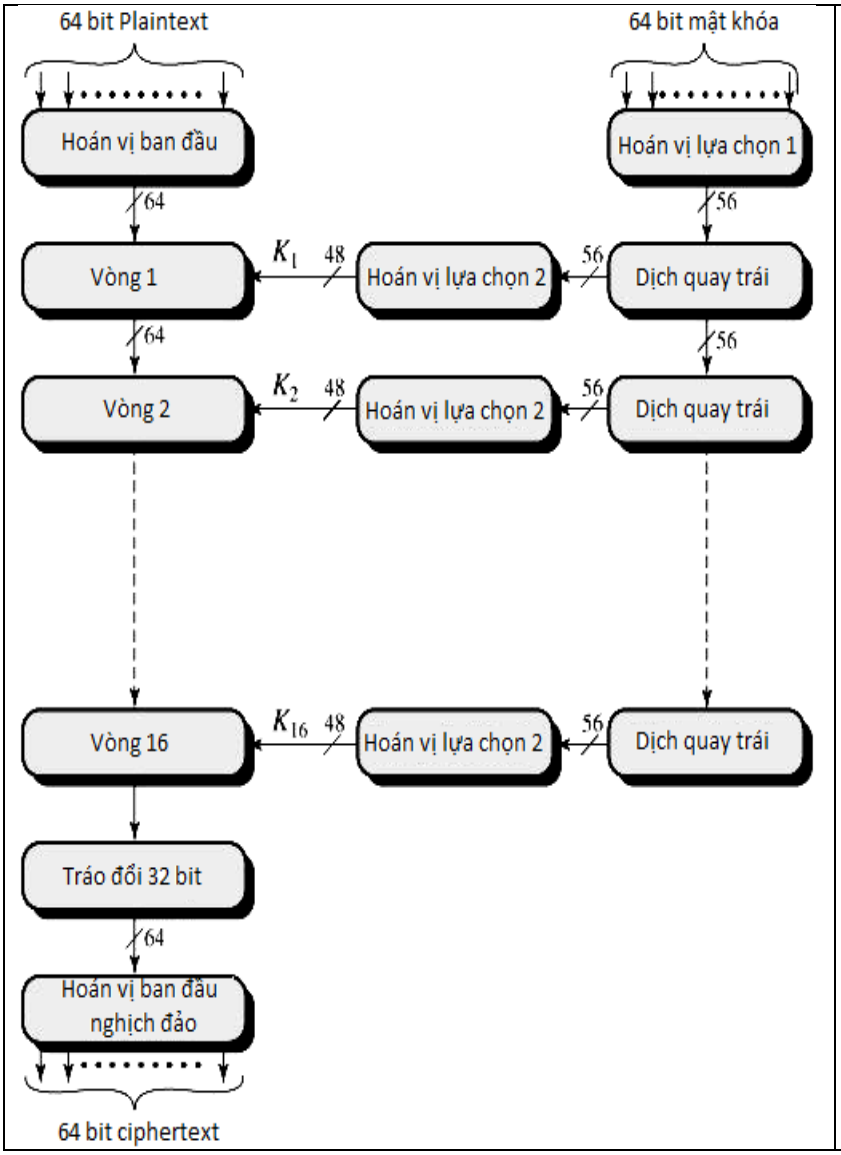
$$R_i = L_{i-1} \oplus F(R_{i-1}, K_i)$$
$$L_i = R_{i-1}$$

- **Kích thước khối (*m*)**: Kích thước khối càng lớn càng bảo mật tốt, nhưng làm giảm tốc độ của thuật toán. Thường thì *m = 64* bít.
- **Kích thước khóa (*k*)**: Kích thước khóa càng lớn càng bảo mật tốt, nhưng làm giảm tốc độ của thuật toán. Trước đây DES sử dụng *k=56* bít, nhưng hiện không còn an toàn nữa. Ngày nay *k = 128* bít là phổ biến.
- Thời gian trung bình để dò tìm khóa theo thuật toán brute-force:

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

- **Số vòng xử lý (*n*)**: Một vòng đơn thì tính bảo mật không cao, nhưng nhiều vòng kết hợp sẽ làm tăng khả năng an ninh. Số vòng thường là 16.
- **Thuật toán sinh khóa con**: Thuật toán càng phức tạp thì càng khó bị phân tích phá mã.
- **Hàm F**: Hàm càng phức tạp thì càng khó bị phân tích phá mã.

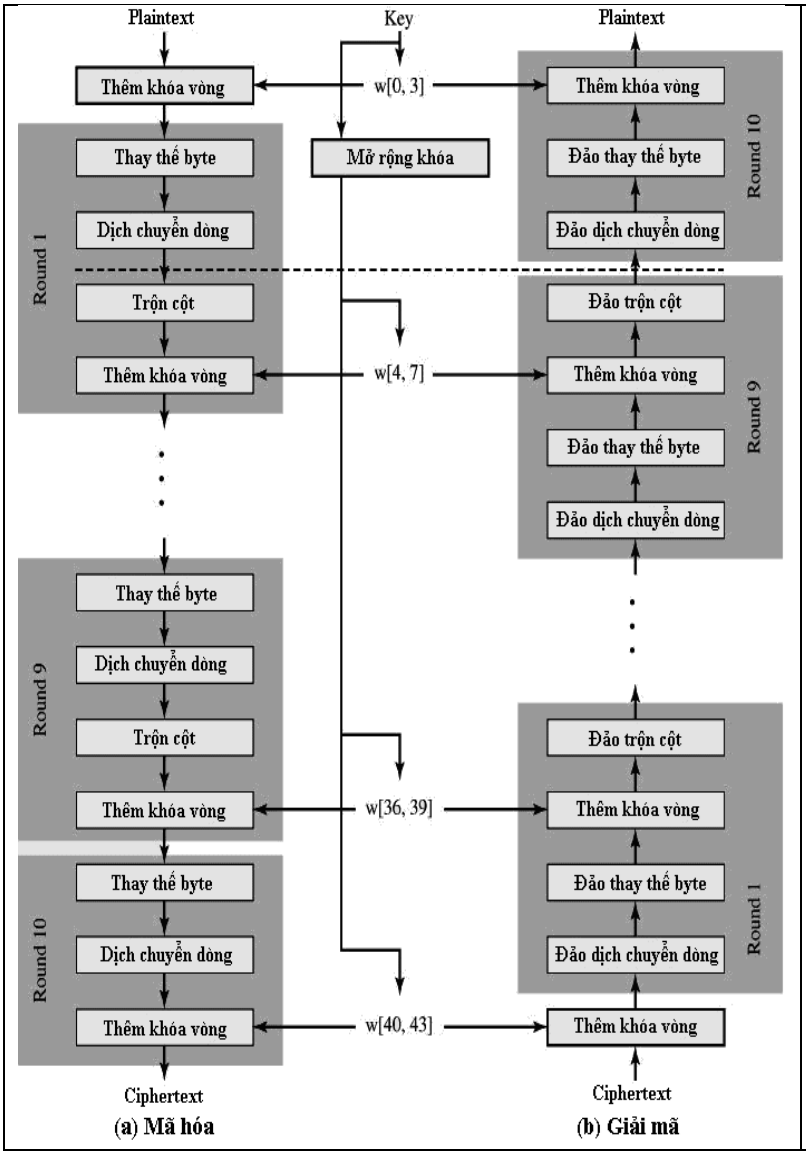
CHUẨN MÃ HÓA DỮ LIỆU DES



- Chuẩn mã hoá dữ liệu DES (Data Encryption Standard) được thông qua năm 1977 bởi National Bureau of Standards (NBS).
- DES sử dụng cấu trúc Feistel với độ dài khối $m = 64$ bít, và độ dài khóa $k = 56$ bít
- DES được sử dụng rộng rãi trong một thời gian dài (hơn 20 năm), cho tới khi nó bị bẻ gãy bởi tấn công brute-force năm 1998.

CHUẨN MÃ HÓA CẢI TIẾN AES

- Chuẩn mã hoá cải tiến AES (*Advanced Encryption Standard*) được ban hành bởi NIST (*National Institute of Standards and Technology*) vào năm 2001.
- AES là mật mã khối để thay thế cho DES trong các ứng dụng thương mại. Nó sử dụng kích thước khối 128 bit và kích thước mật khoá 128, 192 hay 256 bit
- *AES không mang cấu trúc Feistel. Mỗi vòng của nó chứa bốn hàm phân biệt:*
    - Thay thế byte
    - Hoán vị
    - Các phép số học trên một trường hữu hạn
    - XOR với mật khoá



- **Thay thế byte**: Thực hiện một phép thay thế từng byte cho khối.
- **Dịch chuyển hàng**: Chỉ là một hoán vị đơn giản.
- **Trộn cột**: Một phép thay thế sử dụng số học
- **Thêm khoá vòng**: Một phép XOR bít đơn giản của khối hiện tại với một phần của khoá mở rộng.
- Các đặc trưng của AES:
    - Có khả năng chống lại tất cả các tấn công đã được biết đến.
    - Đảm bảo tốc độ thực hiện trên nhiều nền tảng kiến trúc.
    - Có thiết kế đơn giản

CIPHER'S EXERCISE

1. **Caesar**

```
int ktSo(char c)
{
return c - 'A';
}
char soKT(int n)
{
   return 'A' + n;
}

/* CAESAR
BT1: Nhap 1 chuoi plaintext, ma hoa = thuat toan CAESAR tong quat voi khoa K.
Hien chuoi ciphertext, giai ma de khoi phuc lai chuoi plaintext */
int main()
{
   string p, c;
   int k;
   cout << "Plaintext: ";    getline(cin, p);
   cout << "Key: ";          cin >> k;
   // Ma hoa
   for (int i = 0; i < p.size(); i++)
   {
      int m = (ktSo(p[i]) + k) % 26;
      c += soKT(m);
   }
   cout << "Ciphertext: " << c << endl;
   // Giai ma
   p = "";
   for (int i = 0; i < c.size(); i++)
   {
      int m = (ktSo(c[i]) - k + 26) % 26;
      p += soKT(m);
   }
   cout << "Plaintext: " << p;
}
```

```
/* BT2: Be khoa mm CAESAR bang pp Brute-force */
int main()
{
   string p, c;
   int k;
   cout << "Ciphertext: ";    getline(cin, c);
   for (k = 1; k < 26; k++)
   {
      p = "";
      for (int i = 0; i < c.size(); i++)
```

```
        {
            int m = (ktSo(c[i]) - k + 26) % 26;
            p += soKT(m);
        }
        cout << "Key: " << k << " - Plaintext: " << p << endl;
    }
}
```

2. **Affine**

```
/* AFFINE
BT1: Nhap 1 chuoi plaintext, ma hoa = thuat toan Affine voi cap so {a,b} nhap tu ban phim. Hien chuoi ciphertext, giai ma de khoi phuc lai chuoi plaintext */
int main()
{
    string p, c;
    int a, b, temp;
    cout << "Plaintext: ";    getline(cin, p);
    do
    {
        cout << "Key a = ";
        cin >> a;
    } while (a % 2 == 0 || a == 13);
    cout << "Key b = ";    cin >> b;
    // Ma hoa
    c = "";
    for (int i = 0; i < p.size(); i++)
    {
        int m = (a * ktSo(p[i]) + b) % 26;
        c += soKT(m);
    }
    cout << "Ciphertext: " << c << endl;
    // Giai ma
    p = "";
    for (int i = 1; i < 26; i += 2)
    {
        if (i * a % 26 == 1)
        {
            temp = i;
            break;
        }
    }
    for (int i = 0; i < c.size(); i++)
    {
        int m = temp * ktSo(c[i] - b + 26) % 26;
        p += soKT(m);
    }
    cout << "Plaintext: " << p << endl;
```

```
}
/* BT2: Be khoa mm Affine bang pp Brute-force */
int main()
{
   string p, c;
   int a, b;
   cout << "Ciphertext: ";    getline(cin, c);
   for (a = 1; a < 26; a += 2)
   {
      if (a != 13)
      {
         for (b = 0; b < 26; b++)
         {
            p = "";
            int temp;
            for (int i = 1; i < 26; i += 2)
            {
               if (i * a % 26 == 1)
               {
                  temp = i;
                  break;
               }
            }
            for (int i = 0; i < c.size(); i++)
            {
               int m = temp * ktSo(c[i] - b + 26) % 26;
               p += soKT(m);
            }
            cout << "(" << a << ", " << b << ") - p = " << p << endl;
         }
      }
   }
}
```

3. **MonoAlphabetic**

```
/* MonoAlphabetic
BT1: Nhap 1 chuoi plaintext, ma hoa = thuat toan MonoAlphabetic voi khoa K.
Hien chuoi ciphertext, giai ma de khoi phuc lai chuoi plaintext */
int main()
{
   string p, c, k;
   string alpha = "ABCDEFGHIJKLMNOPQSTUVWXYZ";
   cout << "Plaintext: ";      getline(cin, P);
   cout << "Key k = ";         getline(cin, K);
   // Ma hoa
   for (int i = 0; i < p.size(); i++)
   {
```

```
        // Tim ky tu p[i] trong chuoi alpha, tra ve vi tri xuat hien cua P[i] trong alpha
        int temp = alpha.find(p[i]);
        // Thay chuoi p thanh khoa cua k trong tung phan tu
        c += k[temp];
    }
    cout << "Chuoi ma hoa: " << c << endl;
    // Giai ma
    P = "";
    for (int i = 0; i < c.size(); i++)
    {
        int temp = k.find(c[i]);
        p += alpha[temp];
    }
    cout << "Chuoi giai ma: " << P << endl;
}
```

```
/* BT2: Dem ki tu dau tien trong chuoi xuat hien bao nhieu lan */
int demKT(string s)
{
    int count = 1;
    for (int i = 1; i < s.size(); i++)
    {
        if (s[i] == s[0]) count++;
    }
    return count;
}
```

```
/* BT3: Dem tan suat cac ki tu xuat hien trong chuoi */
int main()
{
    string p, n, temp;
    cout << "Plaintext: ";    getline(cin, p);
    // Chuan hoa chuoi
    for (int i = 0; i < p.length(); i++)
    {
        if (p[i] >= 'A' && p[i] <= 'Z') p[i] += 32;
        if (p[i] >= '0' && p[i] <= '9') n += p[i];
        if (p[i] >= 'a' && p[i] <= 'z') n += p[i];
    }
    temp = n;
    sort(n.begin(), n.end());
    // Cac ki tu co trong chuoi
    for (int i = 0; i < n.length() - 1; i++)
    {
        for (int j = i + 1; j < n.length(); j++)
        {
            if (n[i] == n[j])
            {
```

```
            n.erase(j, 1);
            j--;
        }
    }
}
// Dem so lan xuat hien cua cac ki tu
for (int i = 0; i < n.length() - 1; i++)
{
    int count = 0;
    for (int j = i + 1; j < temp.length(); j++)
    {
        if (n[i] == temp[j]) count++;
    }
    cout << n[i] << ": " << count << endl;
}
return 0;
}
```

4. **Playfair**

```
/* Playfair
BT1: Nhap 1 chuoi plaintext, ma hoa = thuat toan Playfair voi khoa K. Hien chuoi
ciphertext, giai ma de khoi phuc lai chuoi plaintext */
int main()
{
    // 1. Lap ma tran khoa 5x5
    string p, c, k, alpha = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    char matrix[5][5];
    cout << "Key k = ";    getline(cin, k);
    k += alpha;
    // Loai bo ki tu trung lap trong key
    for (int i = 0; i < k.size(); i++)
    {
        for (int j = i + 1; j < k.size(); j++)
        {
            if (k[i] == k[j] || k[j] = 'J')
            {
                k.erase(j, 1);
                j--;
            }
        }
    }
    // Ghi key vao mt theo hang, chen them cac ki tu khac vao o trong
    int index = 0;
    for (int i = 0; i < 5; i++)
    {
        for (int j = 0; j < 5; j++)
        {
```

```
            matrix[i][j] = k[index];
            index++;
        }
    }
    // Xua ma tran ra man hinh
    for (int i = 0; i < 5; i++)
    {
        for (int j = 0; j < 5; j++)
        {
            cout << matrix[i][j] << " ";
        }
        cout << "\n";
    }
    // 2. Chuan hoa chuoi plaintext (2 ki tu giong nhau thay = X)
    cout << "Plaintext: ";    getline(cin, p);
    for (int i = 0; i < p.size(); i += 2)
    {
        if (p[i] == p[i + 1]) p.insert(i + 1, 'X');
    }
    if (p.size() % 2 == 1) p += 'X';
    // 3. Ma hoa
    c = p;
    int r1, r2, c1, c2;
    for (int m = 0; m < p.size(); m += 2)
    {
        for (int i = 0; i < 5; i++)
        {
            for (int j = 0; j < 5; j++)
            {
                if (p[m] == matrix[i][j])
                {
                    r1 = i;
                    c1 = j;
                }
                if (p[m + 1] == matrix[i][j])
                {
                    r2 = i;
                    c2 = j;
                }
            }
        }
        if (r1 != r2 && c1 != c2)
        {
            c[m] = matrix[r1][c2];
            c[m + 1] = matrix[r2][c1];
        }
```

```
    else if (r1 == r2)
    {
        if (c1 + 1 >= 5) c1 = -1;
        if (c2 + 1 >= 5) c2 = -1;
        c[m] = matrix[r1][c1 + 1];
        c[m + 1] = matrix[r2][c2 + 1];
    }
    else if (c1 == c2)
    {
        if (r1 + 1 >= 5) r1 = -1;
        if (r2 + 1 >= 5) r2 = -1;
        c[m] = matrix[r1 + 1][c1];
        c[m + 1] = matrix[r2 + 1][c2];
    }
}
cout << "Ciphertext: " << c << endl;
// 4. Giai ma
for (int m = 0; m < c.size(); m += 2)
{
    for (int i = 0; i < 5; i++)
    {
        for (int j = 0; j < 5; j++)
        {
            if (c[m] == matrix[i][j])
            {
                r1 = i;
                c1 = j;
            }
            if (c[m + 1] == matrix[i][j])
            {
                r2 = i;
                c2 = j;
            }
        }
    }
    if (r1 != r2 && c1 != c2)
    {
        p[m] = matrix[r1][c2];
        p[m + 1] = matrix[r2][c1];
    }
    else if (r1 == r2)
    {
        if (c1 - 1 < 0) c1 = 5;
        if (c2 - 1 < 0) c2 = 5;
        p[m] = matrix[r1][c1 - 1];
        p[m + 1] = matrix[r2][c2 - 1];
```

```
        }
        else if (c1 == c2)
        {
            if (r1 - 1 < 0) r1 = 5;
            if (r2 - 1 < 0) r2 = -1;
            p[m] = matrix[r1 - 1][c1];
            p[m + 1] = matrix[r2 - 1][c2];
        }
    }
    cout << "Plaintext: " << p << endl;
}
```

5. **Vigenere**

```
/* Vigenere
BT1: Nhap 1 chuoi plaintext, ma hoa = thuat toan Vigenere voi khoa K. Hien chuoi ciphertext, giai ma de khoi phuc lai chuoi plaintext */
int main()
{
    string p, c, k;
    int index = 0;
    cout << "Plaintext: ";    getline(cin, p);
    cout << "Key k = ";       getline(cin, k);
    // Ma hoa
    for (int i = 0; i < p.size(); i++)
    {
        int P = KT_S(p[i]);
        int K = KT_S(k[index]);
        c += S_KT((P + K) % 26);
        index = (index + 1) % k.size();
    }
    cout << "Ciphertext: " << c << endl;
    // Giai ma
    p = "";
    index = 0;
    for (int i = 0; i < c.size(); i++)
    {
        int C = KT_S(c[i]);
        int K = KT_S(k[index]);
        p += S_KT((C - K + 26) % 26);
        index = (index + 1) % k.size();
    }
    cout << "Plaintext: " << p << endl;
}
```

6. **Vernman**

```
/* Vernman
BT1: Nhap 1 chuoi plaintext, ma hoa = thuat toan Vernman voi khoa K (su dung phep XOR). Hien chuoi ciphertext, giai ma de khoi phuc lai chuoi plaintext */
```

```
int main()
{
   string p, c, k;
   int index = 0;
   cout << "Plaintext: ";    getline(cin, p);
   cout << "Key k = ";       getline(cin, k);
   // Ma hoa
   for (int i = 0; i < p.size(); i++)
   {
      c[i] = p[i] ^ k[index];
      index++;
      if (index == k.size()) index = 0;
   }
   cout << "Ciphertext: " << c << endl;
   // Giai ma
   index = 0;
   for (int i = 0; i < c.size(); i++)
   {
      p[i] = c[i] ^ k[index];
      index++;
      if (index == k.size()) index = 0;
   }
   cout << "Plaintext: " << p << endl;
}
```

7. **Rain – fence**

```
/* Rain-fence
BT1: Nhap 1 chuoi plaintext, ma hoa = thuat toan Rain-fence. Hien chuoi ciphertext, giai ma de khoi phuc lai chuoi plaintext */
int main()
{
   string p, c, c1, c2;
   cout << "Plaintext: ";    getline(cin, p);
   // Ma hoa
   for (int i = 0; i < p.sixe(); i++)
   {
      if (i % 2 == 0) c1 += p[i];
      else { c2 += p[i]; }
   }
   c = c1 + c2;
   cout << "Ciphertext: " << c << endl;
   // Giai ma
   int n;
   c1 = c2 = p = "";
   if (c.size() % 2 == 0) n = 0;
   else { n = 1; }
   for (int i = 0; i < c.size(); i++)
```

```
    {
        if (i < c.size() / 2 + n) c1 += c[i];
        else { c2 += c[i]; }
    }
    for (int i = 0; i < c.size() / 2; i++)
    {
        p += c1[i] + c2[i];
    }
    if (n == 1) p += c1[c1.size() - 1];
    cout << "Plaintext: " << p << endl;
}
```

```
/* BT2: Nhap chuoi s = "huongcha", tang s len 30 ki tu. Ghi s vao ma tran 3x10, doc ma tran theo cot */
int main()
{
    string s, name = "huongcha";
    s = name;
    char matrix[3][10];
    int index = 0;
    while (s.length() < 30)
    {
        s += name[s.length() % name.length()];
    }
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 10; j++)
        {
            matrix[i][j] = s[index];
            index++;
        }
    }
    for (int j = 0; j < 10; j++)
    {
        for (int i = 0; i < 3; i++)
        {
            cout << matrix[i][j] << " ";
        }
        cout << "\n";
    }
    return 0;
}
```

8. **Feistel**

```
/* Feistel
BT1: Nhap 1 chuoi plaintext, ma hoa = thuat toan Feistel voi 2 vong lap:
- Plaintext dai m = 2 word = 16 bit
- Khoa k 8 bit, Ki sinh ra tu k nho phep dich trai k lan
```

```
- Ham F = R(i-1) + Ki */
char F(char r, char ki)
{
   return r + ki;
}
int main()
{
   string p, c;
   char k[3], l[3], r[3];
   int n = 2;
   cout << "Plaintext: ";    getline(cin, p);
   cout << "Key k = ";       cin >> k[0];
   // Ma hoa
   c = p;
   r[0] = p[0];
   l[0] = p[1];
   for (int i = 1; i <= n; i++)
   {
      k[i] = k[0] << i;
      r[i] = l[i - 1] ^ F(r[i - 1], k[i]);
      l[i] = r[i - 1];
   }
   c[0] = l[n];
   c[1] = r[n];
   cout << "Ciphertext: " << c << endl;
}
```

9. **Hash**

```
/* Hash
BT1: Nhap chuoi Plaintext, chia chuoi thanh cacs khoi 64 bit = 8 ki tu
(Su dung phep XOR) Tinh ma hash */
int main()
{
   string M;
   char H[] = {0, 0, 0, 0, 0, 0, 0, 0} cout << "Plaintext: ";
   getline(cin, M);
   // Them ki tu neu do dai chuoi khong phai boi so cua 8
   while (M.size() % 8 != 0)
   {
      M += 'X';
   }
   for (int i = 0; i < M.size(); i += 8)
   {
      for (int j = 0; j < 8; j++)
      {
         H[j] += H[j] ^ M[i + j];
      }
```

```
    }
    cout << "Hash's code: ";
    for (int i = 0; i < 8; i++)
    {
        cout << H[i];
    }
}
```