

## ĐỀ CƯƠNG THỰC HÀNH MÔN AT&BMTT

### MỤC TIÊU

Nắm vững kiến thức về an toàn bảo mật thông tin

### NỘI DUNG

Buổi 1: Lập Trình Python

Buổi 2: Trình bày chi tiết các bước mã hóa, giải mã kỹ thuật mã hóa Des, Xây dựng chương trình Demo

Buổi 3: Cài Đặt Thư Viện Pycrypto – Sử Dụng Mã Hóa Đối Xứng

Buổi 4: Cài Đặt Thư Viện Pycrypto – Sử Dụng Mã Hóa Bất Đối Xứng

Buổi 5: Cài Đặt Thư Viện Pycrypto – Ứng Dụng Băm Và Chữ Ký Số - Truy Tìm Nguồn Gốc Tập Tin

Buổi 6: Xây Dựng Ứng Dụng Demo Các Giải Thuật Mã Hóa và Giải Mã – Nộp Bài

Tài liệu tham khảo:

Hướng dẫn lập trình ngôn ngữ python [www.cit.ctu.edu.vn/~dtnghe/oss/python.pdf](http://www.cit.ctu.edu.vn/~dtnghe/oss/python.pdf)

# BUỔI 1: LẬP TRÌNH PYTHON

## Giới thiệu PYTHON

### 1. Giới thiệu

Python là một ngôn ngữ lập trình dạng thông dịch do Guido Van Rossum tạo ra năm 1990. Python được viết từ nhiều ngôn ngữ lập trình khác nhau và tạo ra những bản hiện thực khác nhau. Trong đó bản hiện thực chính là CPython được viết bằng C.

Python có tính tương thích lớn trên nhiều môi trường phát triển và cũng được ứng dụng rộng rãi trên nhiều lĩnh vực.

Điểm khác biệt đặc trưng giữa C và Python là Python sử dụng cơ chế cấp phát bộ nhớ tự động và hoàn toàn tạo kiểu động (khác với C phải khai báo biến và cấp phát bộ nhớ tương ứng với kiểu của biến). Điều này giúp tối thiểu hóa số lần gõ phím để viết mã lệnh và giúp cho cấu trúc có hình thức gọn gàng sáng sủa rất thuận tiện cho người mới học lập trình.

### 2. Cài đặt Python và trình soạn thảo

Download Python : Truy cập vào trang web : <https://www.python.org/> để download Python phiên bản 2.7.13

#### Chú ý : Cài đặt phiên bản 2.7.13 chứ không cài phiên bản 3.6

Sau khi cài đặt python ta cần thêm Python vào PATH của hệ thống bằng các bước sau : (chỉ thay đổi khi cài đặt trên môi trường window)

Mở Control Panel


- Chọn System
- Chọn Advanced System Setting
- Chọn tab Advance > Environment Variable
- Trong mục Systeme Variable, tìm chọn variable PATH, copy paste dòng sau vào value của variable này :

`;C:\Python27;C:\Python27\Lib\site-packages\;C:\Python27\Scripts\`

- Chọn Ok và kết thúc việc thêm python vào PATH của hệ thống

Download trình soạn thảo : Truy cập vào trang web : <https://atom.io/> để download trình soạn thảo Atom

Cách sử dụng Atom để viết một chương trình đơn giản bằng Python :

- Tạo một thư mục trong ổ D và đặt tên là Python
- Mở Atom, từ menu File chọn New File, ta thấy một cửa sổ mới được tạo ra.
- Bấm tổ hợp Ctrl + S để save file đó vào thư mục Python vừa tạo ở ổ đĩa D với tên gọi hello.py
- Trên cửa sổ soạn thảo của Atom viết vào dòng lệnh sau : `print "Hello Python"` và bấm Ctrl + S để save lại
- Mở cửa sổ terminal (bấm tổ hợp  + R, gõ vào `cmd`).
- Trên cửa sổ terminal, gõ vào dòng lệnh `cd D:\Python` để di chuyển đến thư mục mà ta vừa tạo
- (Nếu báo lỗi thì thực hiện theo 2 bước, ta gõ `D:` để di chuyển qua ổ đĩa D, sau đó ta gõ

### 3. Cài đặt thư viện trong python

- Nâng cấp PIP bằng lệnh python get-pip.py
- Cài đặt các thư viện bằng lệnh pip install <tên thư viện>

Yêu cầu. Viết chương trình để thực hiện các công việc

Cài đặt mật mã affine bằng ngôn ngữ python

Áp dụng trên chương trình giao diện đồ họa cơ bản

Ví dụ



Code

```
# -*- coding: utf-8 -*-
from Tkinter import *
import ttk

window = Tk()

window.title("Welcome to Demo An Toan Bao Mat Thong Tin")

lb0 = Label(window, text="CHƯƠNG TRÌNH DEMO", font=("Arial Bold", 10))
lb0.grid(column=0, row=0)

lb1 = Label(window, text="CHƯƠNG TRÌNH DEMO", font=("Arial Bold", 20))
lb1.grid(column=1, row=1)

lb2 = Label(window, text="MẬT MÃ AFFINE", font=("Arial Bold", 15))
lb2.grid(column=0, row=2)

plainlb3 = Label(window, text="PLAIN TEXT", font=("Arial", 14))
plainlb3.grid(column=0, row=3)

plaintxt = Entry(window, width=20)
```

```

plaintxt.grid(column=1, row=3)
KEYlb4 = Label(window, text="KEY PAIR",font=("Arial", 14))
KEYlb4.grid(column=2, row=3)
KEYA1 = Entry(window,width=3)
KEYA1.grid(column=3, row=3)
KEYB1 = Entry(window,width=5)
KEYB1.grid(column=4, row=3)
lb5 = Label(window, text="CIPHER TEXT",font=("Arial", 14))
lb5.grid(column=0, row=4)
ciphertxt3 = Entry(window,width=20)
ciphertxt3.grid(column=1, row=4)
denctxt3 = Entry(window,width=20)
denctxt3.grid(column=3, row=4)
def Char2Num(c): return ord(c)-65
def Num2Char(n): return chr(n+65)
def xgcd(b,a):
    tmp=a
    x0, x1, y0, y1 = 1, 0, 0, 1
    while a!=0:
        q, b, a = b // a, a, b % a
        x0, x1 = x1, x0 - q * x1
        y0, y1 = y1, y0 - q * y1
    if x0<0:x0=tmp+x0
    return x0
def encryptAF(txt,a,b,m):
    r=""
    for c in txt:
        e=(a*Char2Num(c)+b )%m

```

```

        r=r+Num2Char(e)

    return r

def decryptAF(txt,a,b,m):
    r=""
    a1=xgcd(a,m)
    for c in txt:
        e=(a1*(Char2Num(c)-b ))%m
        r=r+Num2Char(e)
    return r

def clicked():
    a,b,m=int(KEYA1.get()),int(KEYB1.get()),26
    entxt=encryptAF(plaintext.get(),a,b,m)
    ciphertxt3.delete(0,END)
    #a=int(KEYA1.get())
    ciphertxt3.insert(INSERT,entxt)

def giaima():
    a,b,m=int(KEYA1.get()),int(KEYB1.get()),26
    detxt=decryptAF(ciphertxt3.get(),a,b,m)
    denctxt3.delete(0,END)
    #a=int(KEYA1.get())
    denctxt3.insert(INSERT,etxt)

    AFbtn = Button(window, text="Mã Hóa", command=clicked)

AFbtn.grid(column=5, row=3)
DEAFbtn = Button(window, text="Giải Mã ", command=giaima)
DEAFbtn.grid(column=2, row=4)
window.geometry('800x600')
window.mainloop()

```

## BUỔI 2: CÀI ĐẶT THƯ VIỆN PYCRYPTO – SỬ DỤNG MÃ HÓA ĐỐI XỨNG

### Cơ sở lý thuyết

Bản rõ:  $P \in X$ , bản mật:  $C \in Y$  và khóa:  $k, k' \in K$ .

Mật mã là một cặp đơn ánh và ánh xạ ngược của nó với các tham số tương ứng ( $fk, f^{-1}k'$ ).

Tạo mật mã  $fk: X \rightarrow Y$ , hay  $P \xrightarrow{k} C$ .

Giải mật mã  $f^{-1}k': Y \rightarrow X$ , hay  $C \xrightarrow{k'} P$ .

### Phân loại

Mật mã khóa đối xứng (Symmetric Key Cryptography):  $k=k'$ .

Mật mã khóa bất đối xứng (Asymmetric Key Cryptography):  $k \neq k'$ . Tên khác Mật mã khóa công khai (Public Key Cryptography).

Hàm băm (Hash Function): hàm đặc biệt một chiều, thu nhỏ dữ liệu và không có ánh xạ ngược. Ý nghĩa: tạo vết riêng cho tập dữ liệu được trực quan thành Dấu vân tay (Fingerprint).

### Mục tiêu:

Sinh viên hiểu rõ về cách mã hóa – giải mã dữ liệu đối xứng

### Mật mã DES

DES được công nhận vào năm 1977 bởi Viện nghiên cứu quốc gia về chuẩn của Mỹ (NIST – National Institute of Standards and Technology)

### Nguyên lý:

Sử dụng một khóa  $K$  tạo ra  $n$  khóa con  $K_1, K_2, \dots, K_n$

Hoán vị dữ liệu (Initial Permutation)

Thực hiện  $n$  vòng lặp, ở mỗi vòng lặp

Dữ liệu được chia thành hai phần

Áp dụng phép toán thay thế lên một phần, phần còn lại giữ nguyên

Hoán vị 2 phần cho nhau (trái / phải)

Hoán vị dữ liệu (Final Permutation)

DES – Tóm tắt giải thuật

- Tạo 16 khóa con

$C[0]D[0] = PC-1(KEY)$

for  $i = 1$  to 16

$C[i] = \text{LeftShift}[i](C[i-1])$

$D[i] = \text{LeftShift}[i](D[i-1])$

$K[i] = \text{PC-2}(C[i]D[i])$

end for

- Mã hóa khối dữ liệu  $L[0]R[0] = \text{IP}(\text{plain block})$  for  $i=1$  to 16

$L[i] = R[i-1]$

$R[i] = L[i-1] \text{ XOR } F(R[i-1], K[i])$

end for

cipher block =  $\text{FP}(R[16]L[16])$

- Giải mã khối dữ liệu  $R[16]L[16] = \text{IP}(\text{cipher block})$  for  $i=1$  to 16

$R[i-1] = L[i]$

$L[i-1] = R[i] \text{ xor } f(L[i], K[i])$

end for

plain block =  $\text{FP}(L[0]R[0])$

## Mật mã AES

AES sử dụng khối 128 bit và khóa 128 bit được phát triển bởi cơ quan NIST (National Institute of Standards and Technology, USA) vào năm 2001, từ mật mã Rijndael của hai nhà toán học Joan Daemen and Vincent Rijmen người Hà lan. Khóa mật AES có thể nâng cấp tùy sự phát triển của máy tính. Hiện nay khóa an toàn nhất của AES là 256 bit.

## Giải thuật AES đơn giản hóa

### 1. Trạng thái xử lý

Dữ liệu xử lý dài 2 byte (16 bit) được gọi là trạng thái xử lý (State). Trạng thái 2 byte được cấu trúc lại thành ma trận nibble 2x2 và được xếp thứ tự theo cột. Bản rõ:  $P=p_0p_1\dots p_{15}$  và bản mật:  $C=c_0c_1\dots c_{15}$  là các dãy 16 bit được xếp thành trạng thái ban đầu và kết thúc.

$$\text{Bản rõ có trạng thái ban đầu: } S_p = \begin{bmatrix} p_0p_1p_2p_3 & p_8p_9p_{10}p_{11} \\ p_4p_5p_6p_7 & p_{12}p_{13}p_{14}p_{15} \end{bmatrix}$$

$$\text{Bản mật có trạng thái kết thúc: } S_c = \begin{bmatrix} c_0c_1c_2c_3 & c_8c_9c_{10}c_{11} \\ c_4c_5c_6c_7 & c_{12}c_{13}c_{14}c_{15} \end{bmatrix}$$

$$\text{Trạng thái trung gian: } S = \begin{bmatrix} N_0 & N_2 \\ N_1 & N_3 \end{bmatrix}$$

### 2. Các hàm xử lý trạng thái

(1) Hàm Cộng khóa (Add Key):  $A_{K_i} = S \oplus S_{K_i}$ , trong đó  $S_{K_i}$  xếp khóa  $K_i$  (16 bit) theo cấu trúc trạng thái. Do hàm  $A_{K_i}$  có tính chất đặc biệt:  $A_{K_i} \circ A_{K_i} = S \oplus S_{K_i} \oplus S_{K_i} = S$ , nên  $A_{K_i}^{-1} = A_{K_i}$ .

(2) Hàm Thay thế nibble (Nibble Substitution): NS thay thế các nibble trong trạng thái  $S$  bởi các  $Sbox(N)$  và có dạng:

$$S = \begin{bmatrix} N_0 & N_2 \\ N_1 & N_3 \end{bmatrix} \text{ chuyển thành } Sbox = \begin{bmatrix} Sbox(N_0) & Sbox(N_2) \\ Sbox(N_1) & Sbox(N_3) \end{bmatrix}$$

Hàm này có hàm ngược tính theo giải thuật LUT trên là S-box hay  $Sbox(N) = g(f(N))$  suy ra:  $N = f^{-1}(g^{-1}(Sbox))$ .

(3) Hàm Trượt hàng (Shift Row): SR trượt hàng thứ 2 sang phải 1 nibble tương đương với hoán vị các nibble trên hàng 2 như hình dưới đây:

$$S = \begin{bmatrix} N_0 & N_2 \\ N_1 & N_3 \end{bmatrix} \text{ chuyển thành } SR = \begin{bmatrix} N_0 & N_2 \\ N_3 & N_1 \end{bmatrix}$$

Hàm ngược  $SR^{-1}$  trượt hàng 2 của SR ngược lại sang trái 1 nibble quay về  $S$ .

(4) Hàm Trộn cột (Mix Column): MC biến đổi mỗi cột của trạng thái  $S$  bằng cách nhân với  $c(z)=x^2z+1$  trong vành  $R(2^2)$  modulo đa thức  $z^2+1$ . Phép nhân đã được giải thích ở phần trên. Kết quả phép nhân có thể tóm gọn thành bảng kết quả:



$$\begin{bmatrix} b_0 & b_1 & b_2 & b_3 \\ b_4 & b_5 & b_6 & b_7 \end{bmatrix} \text{thành} \begin{bmatrix} b_0 \oplus b_6 & b_1 \oplus b_4 \oplus b_7 & b_2 \oplus b_4 \oplus b_5 & b_3 \oplus b_5 \\ b_2 \oplus b_4 & b_0 \oplus b_3 \oplus b_5 & b_0 \oplus b_1 \oplus b_6 & b_1 \oplus b_7 \end{bmatrix}$$

Hàm ngược của MC như đã phân tích ở phần trên chỉ là kết quả nhân với  $c^{-1}(z) = x \cdot z + x^3 + 1$ . Như vậy,  $MC^{-1} = c^{-1}(z)S$ . Kết quả này cũng có thể tóm gọn thành bảng như trên.

### 3. Giải thuật Tạo mật mã

Giải thuật AES đơn giản hóa có thể tóm gọn thành hàm hợp:

$$A_{K_2} \circ SR \circ NS \circ A_{K_1} \circ MC \circ SR \circ NS \circ A_{K_0}$$

biến đổi trạng thái bản rõ về trạng thái bản mật.

Giải thuật được thực hiện qua 2 vòng tính theo chỉ số của khóa vòng tính  $K_i$ ,  $i=0,1,2$

Vòng tính khởi tạo chỉ đơn giản cộng bit  $A_{K_0}$  với  $K_0$ .

Vòng tính 1 thực hiện các phép biến đổi trạng thái NS, SR, MC và cuối cùng là cộng bit  $A_{K_1}$  với  $K_1$ .

Vòng tính 2 thực hiện tương tự với  $K_2$  nhưng thiếu MC.

### 4. Giải thuật Giải mật mã

Giải mật là quy trình ngược sử dụng các hàm ngược như đã nêu ở phần trên và có dạng:

$$A_{K_0} \circ NS^{-1} \circ SR^{-1} \circ MC^{-1} \circ A_{K_1} \circ NS^{-1} \circ SR^{-1} \circ A_{K_2}$$

Các bước giải mật này gây khó cho cài đặt do quy trình thuận và ngược không sử dụng cùng loại hàm xử lý. Vấn đề: có thể đổi vị trí các hàm cùng loại về cùng thứ tự xử lý ở hai quy trình này không?

Trước hết dễ thấy  $NS^{-1} \circ SR^{-1} = SR^{-1} \circ NS^{-1}$  do các hàm ngược này xử lý tuần tự: Trượt hàng rồi Thay thế hay ngược lại đều như nhau.

Tiếp theo,  $MC^{-1} \circ A_{K_i}(S) = MC^{-1}(A_{K_i}(S)) = MC^{-1}(K_i \oplus S) = \alpha(z)^{-1}(K_i \oplus S) = \alpha(z)^{-1}(K_i) \oplus \alpha(z)^{-1}(S) = \alpha(z)^{-1}(K_i) \oplus MC^{-1}(S) = A_{\alpha(z)^{-1}K_i}(MC^{-1}(S))$ .

Vậy quy trình ngược của giải mật được viết lại một cách tương đương:

$$A_{K_0} \circ SR^{-1} \circ NS^{-1} \circ A_{\alpha(z)^{-1}K_i} \circ MC^{-1} \circ SR^{-1} \circ NS^{-1} \circ A_{K_2}$$

Quy trình này cho phép cài đặt AES dễ dàng hơn rất nhiều do các bước đều sử dụng cùng loại hàm khả nghịch.

### Yêu Cầu

Sinh viên viết chương trình trên để thực hiện các yêu cầu sau:

1. Sinh khóa và lưu khóa
2. Đọc khóa và tạo mật mã / giải mật mã DES (ECB, CBC)
3. Đọc khóa và tạo mật mã / giải mật mã AES (ECB)
4. Xây dựng Demo – giao diện đồ họa (tham khảo buổi 1)

Tham khảo

<https://www.dlitz.net/software/pycrypto/api/2.6/>

## BUỔI 3: : CÀI ĐẶT THƯ VIỆN PYCRYPTO – SỬ DỤNG MÃ HÓA BẤT ĐỐI XỨNG

Sinh khóa, lưu khóa, đọc khóa mà mật mã Diffie–Hellman

Sinh khóa, lưu khóa, đọc khóa và mật mã RSA

- Được phát triển bởi Rivest, Shamir và Adleman.
- Mật mã hóa và giải mật mã được tính theo công thức:
- $C = M^e \bmod n$
- $M = C^d \bmod n$
- Các yêu cầu:
- Có thể tìm được các giá trị  $e$ ,  $d$ ,  $n$  sao cho
- $M^e d \equiv M \pmod{n}$  với mọi  $M < n$
- Dễ dàng tính được  $M^e$  và  $C^d$  với mọi  $M < n$
- Không thể tính được  $d$  từ  $e$  và  $n$
- Giải thuật:
- Chọn 2 số nguyên tố lớn  $p$  và  $q$
- Tính  $n = p * q$
- Tính  $\phi(n) = (p-1) * (q-1)$
- Chọn  $e$  sao cho  $\text{USCLN}(e, \phi(n)) = 1$  với  $1 < e < \phi(n)$
- Tính  $d$  sao cho  $ed \equiv 1 \pmod{\phi(n)}$

### Yêu Cầu

Sinh viên viết chương trình trên để thực hiện các yêu cầu sau:

1. Sinh khóa và lưu khóa
2. Đọc khóa và tạo mật mã / giải mật Diffie–Hellman
3. Đọc khóa và tạo mật mã / giải mật RSA
4. Xây dựng Demo – giao diện đồ họa (tham khảo buổi 1)

Tham khảo

<https://www.dlitz.net/software/pycrypto/api/2.6/>



## BUỔI 4: CÀI ĐẶT THƯ VIỆN PYCRYPTO – ỨNG DỤNG BĂM VÀ CHỮ KÝ SỐ - TRUY TÌM NGUỒN GỐC TẬP TIN

Băm là một giải pháp tạo ra một đặc trưng cho một file dữ liệu. Tương tự như mỗi một con người có một dấu vân tay đặc trưng. Vì vậy Băm còn được gọi dấu vân tay (Fingerprint) của file dữ liệu.

Hàm băm (Hash Function) là một dạng mật mã tạo bản mật không cần giải mật mà đáp ứng yêu cầu kiểm tra tính toàn vẹn của một dữ liệu dựa trên đặc trưng vân tay của nó.

Băm (Hash) là một khối dữ liệu thu nhỏ lại từ một file dữ liệu và được tạo ra bởi hàm băm (Hash Function).

Hàm băm là một phép biến đổi một chiều (One-Way) có đầu vào là dãy  $m+t$  bit và đầu ra là dãy  $t$  bit, trong đó  $t < m$ . Đầu vào của hàm băm gồm file dữ liệu với  $m$  bit và IV (Input Value) với  $t$  bit. Đầu ra chính là giá trị băm với  $t$  bit.

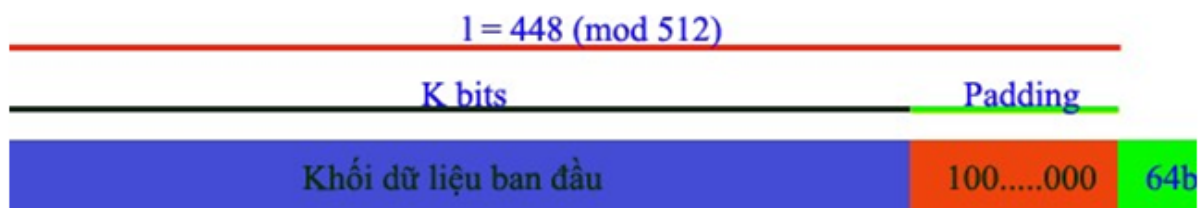
Hàm băm  $H(x)$  có khả năng bảo mật tốt, nếu thỏa 3 tính chất: Một chiều (One Way), Tự do liên kết yếu (Weakly Collision Free) và Tự do liên kết mạnh (Strong Collision Free).

### Giải thuật MD5

- Phát triển bởi Ron Rivest tại đại học MIT
- Input: thông điệp với độ dài bất kỳ
- Output: giá trị băm (message digest) 128 bits
- Giải thuật gồm 5 bước thao tác trên khối 512 bits

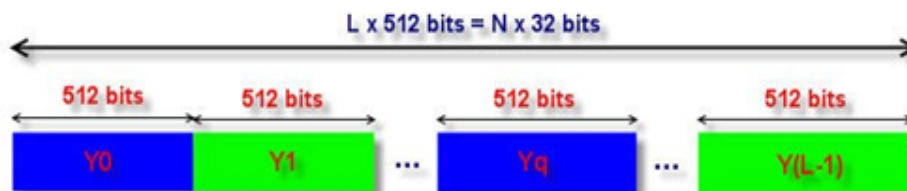
• Bước 1: **nhồi dữ liệu**

- Nhồi thêm các bits sao cho dữ liệu có độ dài  $l \equiv 448 \bmod 512$  hay  $l = n * 512 + 448$  ( $n, l$  nguyên)
- Luôn thực hiện nhồi dữ liệu ngay cả khi dữ liệu ban đầu có độ dài mong muốn. Ví dụ, dữ liệu có độ dài 448 được nhồi thêm 512 bits để được độ dài 960 bits.
- Số lượng bit nhồi thêm nằm trong khoảng 1 đến 512
- Các bit được nhồi gồm 1 bit "1" và các bit 0 theo sau.



• Bước 2: **thêm vào độ dài**

- Độ dài của khối dữ liệu ban đầu được biểu diễn dưới dạng nhị phân 64-bit và được thêm vào cuối chuỗi nhị phân kết quả của bước 1
- Nếu độ dài của khối dữ liệu ban đầu  $> 2^{64}$ , chỉ 64 bits thấp được sử dụng, nghĩa là giá trị được thêm vào bằng  $K \bmod 2^{64}$
- Kết quả có được từ 2 bước đầu là một khối dữ liệu có độ dài là bội số của 512. Khối dữ liệu được biểu diễn:
  - Bằng một dãy  $L$  khối 512-bit  $Y_0, Y_1, \dots, Y_{L-1}$
  - Bằng một dãy  $N$  từ (word) 32-bit  $M_0, M_1, \dots, M_{N-1}$ . Vậy  $N = L \times 16$  ( $32 \times 16 = 512$ )



- Bước 3: **khởi tạo bộ đệm MD** (MD buffer)
  - Một bộ đệm 128-bit được dùng lưu trữ các giá trị băm trung gian và kết quả. Bộ đệm được biểu diễn bằng 4 thanh ghi 32-bit với các giá trị khởi tạo ở dạng little-endian (byte có trọng số nhỏ nhất trong từ nằm ở địa chỉ thấp nhất) như sau:
    - A = 67 45 23 01
    - B = EF CD AB 89
    - C = 98 BA DC FE
    - D = 10 32 54 76
  - Các giá trị này tương đương với các từ 32-bit sau:
    - A = 01 23 45 67
    - B = 89 AB CD EF
    - C = FE DC BA 98
    - D = 76 54 32 10

#### Bước 4: xử lý các khối dữ liệu 512-bit

- Trọng tâm của giải thuật là **hàm nén** (compression function) gồm 4 "vòng" xử lý. Các vòng này có cấu trúc giống nhau nhưng sử dụng các hàm luận lý khác nhau gồm F, G, H và I
- $F(X,Y,Z) = X \wedge Y \vee \neg X \wedge Z$
- $G(X,Y,Z) = X \wedge Z \vee Y \wedge \neg Z$
- $H(X,Y,Z) = X \text{ xor } Y \text{ xor } Z$
- $I(X,Y,Z) = Y \text{ xor } (X \vee \neg Z)$
- Mảng 64 phần tử được tính theo công thức:  $T[i] = 2^{32} \times \text{abs}(\sin(i))$ ,  $i$  được tính theo radian.
- Kết quả của 4 vòng được cộng (theo modulo  $2^{32}$  với đầu vào  $CV_q$  để tạo  $CV_{q+1}$

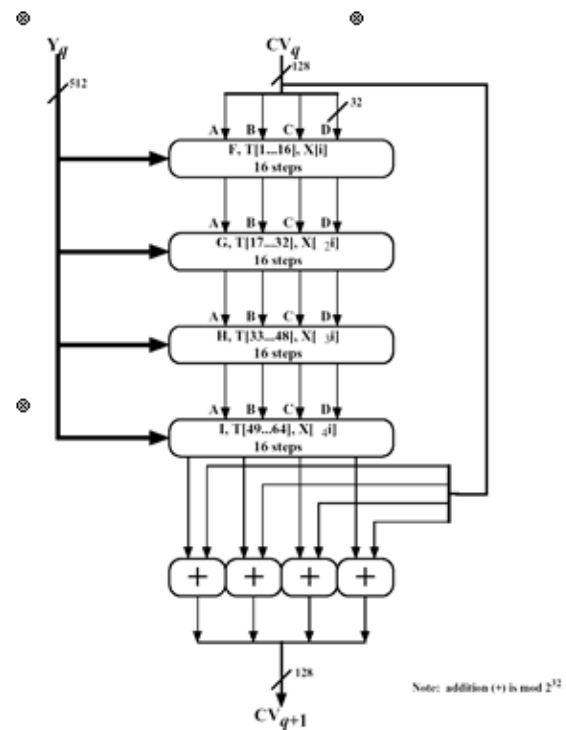


Figure 9.2 MD5 Processing of a Single 512-bit Block (MD5 Compression Function)

#### Bước 5: Xuất kết quả

- Sau khi xử lý hết L khối 512-bit, đầu ra của lần xử lý thứ L là giá trị băm 128 bits.

#### Giải thuật SHA-1

1. Secure Hash Algorithm (SHA) phát triển bởi National Institute of Standard and Technology (NIST)
2. Đầu vào: thông điệp với độ dài tối đa 264 bits
3. Đầu ra: giá trị băm (message digest) có độ dài 160 bits
4. Giải thuật gồm 5 bước thao tác trên các khối 512 bits

### Bước 1: **nhồi thêm dữ liệu**

- Thông điệp được nhồi thêm các bits sao cho độ dài  $l \equiv 448 \pmod{512}$  hay  $l = n * 512 + 448 (n, l \text{ nguyên})$
- Thông điệp luôn luôn được nhồi thêm dữ liệu
- Số bits nhồi thêm nằm trong khoảng 1 đến 512
- Phần dữ liệu nhồi thêm bao gồm một bit 1 và theo sau là các bit 0

### Bước 2: **thêm vào độ dài**

- Độ dài của khối dữ liệu ban đầu được biểu diễn dưới dạng nhị phân 64-bit và được thêm vào cuối chuỗi nhị phân kết quả của bước 1
- Độ dài được biểu diễn dưới dạng nhị phân 64-bit không dấu
- Kết quả có được từ 2 bước đầu là một khối dữ liệu có độ dài là bội số của 512. Khối dữ liệu được biểu diễn:
  - Bằng một dãy  $L$  khối 512-bit  $Y_0, Y_1, \dots, Y_{L-1}$
  - Bằng một dãy  $N$  từ (word) 32-bit  $M_0, M_1, M_{N-1}$ . Vậy  $N = L \times 16$

### Bước 3: **khởi tạo bộ đệm MD** (MD buffer)

- Một bộ đệm 160-bit được dùng lưu trữ các giá trị băm trung gian và kết quả. Bộ đệm được biểu diễn bằng 5 thanh ghi 32-bit với các giá trị khởi tạo ở dạng big-endian (byte có trọng số lớn nhất trong từ nằm ở địa chỉ thấp nhất) như sau:
  - $A = 01\ 23\ 45\ 67$
  - $B = 89\ AB\ CD\ EF$
  - $C = FE\ DC\ BA\ 98$
  - $D = 76\ 54\ 32\ 10$
  - $E = C3\ D2\ E1\ F0$
- Các giá trị này tương đương với các từ 32-bit sau:
  - $A = 01\ 23\ 45\ 67$
  - $B = 89\ AB\ CD\ EF$
  - $C = FE\ DC\ BA\ 98$
  - $D = 76\ 54\ 32\ 10$
  - $E = C3\ D2\ E1\ F0$



#### Bước 4: xử lý các khối dữ liệu 512-bit

- Trọng tâm của giải thuật bao gồm 4 lặp thực hiện tất cả 80 bước.
- 4 vòng lặp có cấu trúc như nhau, chỉ nhau ở các hàm logic  $f_1, f_2, f_3, f_4$
- Mỗi vòng có đầu vào gồm khối 512-thời và một bộ đệm 160-bit ABCDE. Mỗi thao tác sẽ cập nhật giá trị bộ đệm
- Mỗi bước sử dụng một hằng số  $K_t$  (0 ≤ t ≤ 79)
  - $K_t = 5A827999$  (0 ≤ t ≤ 19)
  - $K_t = 6ED9EBA1$  (20 ≤ t ≤ 39)
  - $K_t = 8F1BBCDC$  (40 ≤ t ≤ 59)
  - $K_t = CA62C1D6$  (60 ≤ t ≤ 79)
- Đầu ra của 4 vòng (bước 80) được cộng đầu ra của bước  $CV_q$  để tạo ra  $CV_{q+1}$

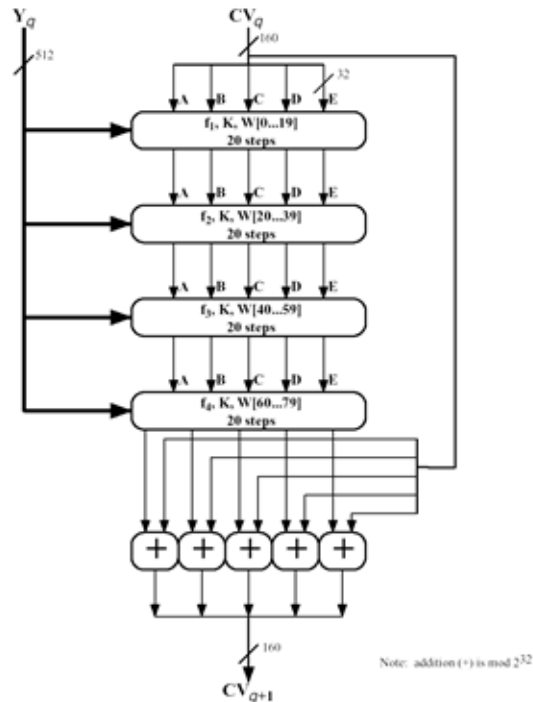


Figure 9.5 SHA-1 Processing of a Single 512-bit Block (SHA-1 Compression Function)

#### Bước 5: xuất kết quả

- Sau khi thao tác trên toàn bộ L blocks. Kết quả của khối thứ L là bảng băm 160-bit

##### Yêu Cầu

Sinh viên viết chương trình trên để thực hiện các yêu cầu sau:

1. Băm file dữ liệu: MD5 và SHA 1 và kiểm tra toàn vẹn
2. Sử dụng RSA tạo chữ ký số cho file dữ liệu
3. Kiểm tra chữ ký số và xác nhận nguồn gốc file dữ liệu
4. Xây dựng Demo – giao diện đồ họa (tham khảo buổi 1)

## BUỔI 5: XÂY DỰNG CHƯƠNG TRÌNH TẠO CHỮ KÝ SỐ BẰNG NGÔN NGỮ PYTHON – SỬ DỤNG RSA VÀ HASH 256

Yêu Cầu

Sinh viên viết chương trình – làm bài tập nhóm

## ĐỀ CƯƠNG THỰC HÀNH MÔN AT&BMTT

### MỤC TIÊU

Nắm vững kiến thức về an toàn bảo mật thông tin

### NỘI DUNG

Buổi 1: Lập Trình Python

Buổi 2: Trình bày chi tiết các bước mã hóa, giải mã kỹ thuật mã hóa Des, Xây dựng chương trình Demo

Buổi 3: Cài Đặt Thư Viện Pycrypto – Sử Dụng Mã Hóa Đối Xứng

Buổi 4: Cài Đặt Thư Viện Pycrypto – Sử Dụng Mã Hóa Bất Đối Xứng

Buổi 5: Cài Đặt Thư Viện Pycrypto – Ứng Dụng Băm Và Chữ Ký Số - Truy Tìm Nguồn Gốc Tập Tin

Buổi 6: Xây Dựng Ứng Dụng Demo Các Giải Thuật Mã Hóa và Giải Mã – Nộp Bài

Tài liệu tham khảo:

Hướng dẫn lập trình ngôn ngữ python [www.cit.ctu.edu.vn/~dtngchi/oss/python.pdf](http://www.cit.ctu.edu.vn/~dtngchi/oss/python.pdf)

# BUỔI 1: LẬP TRÌNH PYTHON

## Giới thiệu OpenSSL

### 4. Giới thiệu

Python là một ngôn ngữ lập trình dạng thông dịch do Guido Van Rossum tạo ra năm 1990. Python được viết từ nhiều ngôn ngữ lập trình khác nhau và tạo ra những bản hiện thực khác nhau. Trong đó bản hiện thực chính là CPython được viết bằng C.

Python có tính tương thích lớn trên nhiều môi trường phát triển và cũng được ứng dụng rộng rãi trên nhiều lĩnh vực.

Điểm khác biệt đặc trưng giữa C và Python là Python sử dụng cơ chế cấp phát bộ nhớ tự động và hoàn toàn tạo kiểu động (khác với C phải khai báo biến và cấp phát bộ nhớ tương ứng với kiểu của biến). Điều này giúp tối thiểu hóa số lần gõ phím để viết mã lệnh và giúp cho cấu trúc có hình thức gọn gàng sáng sủa rất thuận tiện cho người mới học lập trình.

### 5. Cài đặt Python và trình soạn thảo

Download Python : Truy cập vào trang web : <https://www.python.org/> để download Python phiên bản 2.7.13

#### Chú ý : Cài đặt phiên bản 2.7.13 chứ không cài phiên bản 3.6

Sau khi cài đặt python ta cần thêm Python vào PATH của hệ thống bằng các bước sau : (chỉ thay đổi khi cài đặt trên môi trường window)

Mở Control Panel


- Chọn System
- Chọn Advanced System Setting
- Chọn tab Advance > Environment Variable
- Trong mục Systeme Variable, tìm chọn variable PATH, copy paste dòng sau vào value của variable này :

`;C:\Python27;C:\Python27\Lib\site-packages\;C:\Python27\Scripts\`

- Chọn Ok và kết thúc việc thêm python vào PATH của hệ thống

Download trình soạn thảo : Truy cập vào trang web : <https://atom.io/> để download trình soạn thảo Atom

Cách sử dụng Atom để viết một chương trình đơn giản bằng Python :

- Tạo một thư mục trong ổ D và đặt tên là Python
- Mở Atom, từ menu File chọn New File, ta thấy một cửa sổ mới được tạo ra.
- Bấm tổ hợp Ctrl + S để save file đó vào thư mục Python vừa tạo ở ổ đĩa D với tên gọi hello.py
- Trên cửa sổ soạn thảo của Atom viết vào dòng lệnh sau : `print "Hello Python"` và bấm Ctrl + S để save lại
- Mở cửa sổ terminal (bấm tổ hợp  + R, gõ vào `cmd`).
- Trên cửa sổ terminal, gõ vào dòng lệnh `cd D:\Python` để di chuyển đến thư mục mà ta vừa tạo
- (Nếu báo lỗi thì thực hiện theo 2 bước, ta gõ `D:` để di chuyển qua ổ đĩa D, sau đó ta gõ

## 6. Cài đặt thư viện trong python

- Nâng cấp PIP bằng lệnh python get-pip.py
- Cài đặt các thư viện bằng lệnh pip install <tên thư viện>

Yêu cầu. Viết chương trình để thực hiện các công việc

Cài đặt mật mã affine bằng ngôn ngữ python

Áp dụng trên chương trình giao diện đồ họa cơ bản

Ví dụ



Code

```
# -*- coding: utf-8 -*-
from Tkinter import *
import ttk

window = Tk()

window.title("Welcome to Demo An Toan Bao Mat Thong Tin")

lb0 = Label(window, text="CHƯƠNG TRÌNH DEMO", font=("Arial Bold", 10))
lb0.grid(column=0, row=0)

lb1 = Label(window, text="CHƯƠNG TRÌNH DEMO", font=("Arial Bold", 20))
lb1.grid(column=1, row=1)

lb2 = Label(window, text="MẬT MÃ AFINE", font=("Arial Bold", 15))
lb2.grid(column=0, row=2)

plainlb3 = Label(window, text="PLAIN TEXT", font=("Arial", 14))
plainlb3.grid(column=0, row=3)

plaintxt = Entry(window, width=20)
```

```

plaintxt.grid(column=1, row=3)
KEYlb4 = Label(window, text="KEY PAIR",font=("Arial", 14))
KEYlb4.grid(column=2, row=3)
KEYA1 = Entry(window,width=3)
KEYA1.grid(column=3, row=3)
KEYB1 = Entry(window,width=5)
KEYB1.grid(column=4, row=3)
lb5 = Label(window, text="CIPHER TEXT",font=("Arial", 14))
lb5.grid(column=0, row=4)
ciphertxt3 = Entry(window,width=20)
ciphertxt3.grid(column=1, row=4)
denctxt3 = Entry(window,width=20)
denctxt3.grid(column=3, row=4)
def Char2Num(c): return ord(c)-65
def Num2Char(n): return chr(n+65)
def xgcd(b,a):
    tmp=a
    x0, x1, y0, y1 = 1, 0, 0, 1
    while a!=0:
        q, b, a = b // a, a, b % a
        x0, x1 = x1, x0 - q * x1
        y0, y1 = y1, y0 - q * y1
    if x0<0:x0=tmp+x0
    return x0
def encryptAF(txt,a,b,m):
    r=""
    for c in txt:
        e=(a*Char2Num(c)+b )%m

```

```

        r=r+Num2Char(e)

    return r

def decryptAF(txt,a,b,m):
    r=""
    a1=xgcd(a,m)
    for c in txt:
        e=(a1*(Char2Num(c)-b ))%m
        r=r+Num2Char(e)
    return r

def clicked():
    a,b,m=int(KEYA1.get()),int(KEYB1.get()),26
    entxt=encryptAF(plaintext.get(),a,b,m)
    ciphertxt3.delete(0,END)
    #a=int(KEYA1.get())
    ciphertxt3.insert(INSERT,entxt)

def giaima():
    a,b,m=int(KEYA1.get()),int(KEYB1.get()),26
    detxt=decryptAF(ciphertxt3.get(),a,b,m)
    denctxt3.delete(0,END)
    #a=int(KEYA1.get())
    denctxt3.insert(INSERT,detxt)

    AFbtn = Button(window, text="Mã Hóa", command=clicked)

AFbtn.grid(column=5, row=3)
DEAFbtn = Button(window, text="Giải Mã ", command=giaima)
DEAFbtn.grid(column=2, row=4)
window.geometry('800x600')
window.mainloop()

```

## BUỔI 2: CÀI ĐẶT THƯ VIỆN PYCRYPTO – SỬ DỤNG MÃ HÓA ĐỐI XỨNG

### Cơ sở lý thuyết

Bản rõ:  $P \in X$ , bản mật:  $C \in Y$  và khóa:  $k, k' \in K$ .

Mật mã là một cặp đơn ánh và ánh xạ ngược của nó với các tham số tương ứng ( $fk, f^{-1}k'$ ).

Tạo mật mã  $fk: X \rightarrow Y$ , hay  $P \xrightarrow{k} C$ .

Giải mật mã  $f^{-1}k': Y \rightarrow X$ , hay  $C \xrightarrow{k'} P$ .

### Phân loại

Mật mã khóa đối xứng (Symmetric Key Cryptography):  $k=k'$ .

Mật mã khóa bất đối xứng (Asymmetric Key Cryptography):  $k \neq k'$ . Tên khác Mật mã khóa công khai (Public Key Cryptography).

Hàm băm (Hash Function): hàm đặc biệt một chiều, thu nhỏ dữ liệu và không có ánh xạ ngược. Ý nghĩa: tạo vết riêng cho tập dữ liệu được trực quan thành Dấu vân tay (Fingerprint).

### Mục tiêu:

Sinh viên hiểu rõ về cách mã hóa – giải mã dữ liệu đối xứng

### Mật mã DES

DES được công nhận vào năm 1977 bởi Viện nghiên cứu quốc gia về chuẩn của Mỹ (NIST – National Institute of Standards and Technology)

### Nguyên lý:

Sử dụng một khóa  $K$  tạo ra  $n$  khóa con  $K_1, K_2, \dots, K_n$

Hoán vị dữ liệu (Initial Permutation)

Thực hiện  $n$  vòng lặp, ở mỗi vòng lặp

Dữ liệu được chia thành hai phần

Áp dụng phép toán thay thế lên một phần, phần còn lại giữ nguyên

Hoán vị 2 phần cho nhau (trái / phải)

Hoán vị dữ liệu (Final Permutation)

DES – Tóm tắt giải thuật

- Tạo 16 khóa con

$C[0]D[0] = PC-1(KEY)$



for  $i = 1$  to 16

$C[i] = \text{LeftShift}[i](C[i-1])$

$D[i] = \text{LeftShift}[i](D[i-1])$

$K[i] = \text{PC-2}(C[i]D[i])$

end for

- Mã hóa khối dữ liệu  $L[0]R[0] = \text{IP}(\text{plain block})$  for  $i=1$  to 16

$L[i] = R[i-1]$

$R[i] = L[i-1] \text{ XOR } F(R[i-1], K[i])$

end for

cipher block =  $\text{FP}(R[16]L[16])$

- Giải mã khối dữ liệu  $R[16]L[16] = \text{IP}(\text{cipher block})$  for  $i=1$  to 16

$R[i-1] = L[i]$

$L[i-1] = R[i] \text{ xor } f(L[i], K[i])$

end for

plain block =  $\text{FP}(L[0]R[0])$

## Mật mã AES

AES sử dụng khối 128 bit và khóa 128 bit được phát triển bởi cơ quan NIST (National Institute of Standards and Technology, USA) vào năm 2001, từ mật mã Rijndael của hai nhà toán học Joan Daemen and Vincent Rijmen người Hà lan. Khóa mật AES có thể nâng cấp tùy sự phát triển của máy tính. Hiện nay khóa an toàn nhất của AES là 256 bit.

## Giải thuật AES đơn giản hóa

### 1. Trạng thái xử lý

Dữ liệu xử lý dài 2 byte (16 bit) được gọi là trạng thái xử lý (State). Trạng thái 2 byte được cấu trúc lại thành ma trận nibble 2x2 và được xếp thứ tự theo cột. Bản rõ:  $P=p_0p_1\dots p_{15}$  và bản mật:  $C=c_0c_1\dots c_{15}$  là các dãy 16 bit được xếp thành trạng thái ban đầu và kết thúc.

$$\text{Bản rõ có trạng thái ban đầu: } S_p = \begin{bmatrix} p_0p_1p_2p_3 & p_8p_9p_{10}p_{11} \\ p_4p_5p_6p_7 & p_{12}p_{13}p_{14}p_{15} \end{bmatrix}$$

$$\text{Bản mật có trạng thái kết thúc: } S_c = \begin{bmatrix} c_0c_1c_2c_3 & c_8c_9c_{10}c_{11} \\ c_4c_5c_6c_7 & c_{12}c_{13}c_{14}c_{15} \end{bmatrix}$$

$$\text{Trạng thái trung gian: } S = \begin{bmatrix} N_0 & N_2 \\ N_1 & N_3 \end{bmatrix}$$

### 2. Các hàm xử lý trạng thái

(1) Hàm Cộng khóa (Add Key):  $A_{K_i} = S \oplus S_{K_i}$ , trong đó  $S_{K_i}$  xếp khóa  $K_i$  (16 bit) theo cấu trúc trạng thái. Do hàm  $A_{K_i}$  có tính chất đặc biệt:  $A_{K_i} \circ A_{K_i} = S \oplus S_{K_i} \oplus S_{K_i} = S$ , nên  $A_{K_i}^{-1} = A_{K_i}$ .

(2) Hàm Thay thế nibble (Nibble Substitution): NS thay thế các nibble trong trạng thái  $S$  bởi các Sbox(N) và có dạng:

$$S = \begin{bmatrix} N_0 & N_2 \\ N_1 & N_3 \end{bmatrix} \text{ chuyển thành } Sbox = \begin{bmatrix} Sbox(N_0) & Sbox(N_2) \\ Sbox(N_1) & Sbox(N_3) \end{bmatrix}$$

Hàm này có hàm ngược tính theo giải thuật LUT trên là S-box hay  $Sbox(N) = g(f(N))$  suy ra:  $N = f^{-1}(g^{-1}(Sbox))$ .

(3) Hàm Trượt hàng (Shift Row): SR trượt hàng thứ 2 sang phải 1 nibble tương đương với hoán vị các nibble trên hàng 2 như hình dưới đây:

$$S = \begin{bmatrix} N_0 & N_2 \\ N_1 & N_3 \end{bmatrix} \text{ chuyển thành } SR = \begin{bmatrix} N_0 & N_2 \\ N_3 & N_1 \end{bmatrix}$$

Hàm ngược  $SR^{-1}$  trượt hàng 2 của SR ngược lại sang trái 1 nibble quay về  $S$ .

(4) Hàm Trộn cột (Mix Column): MC biến đổi mỗi cột của trạng thái  $S$  bằng cách nhân với  $c(z)=x^2z+1$  trong vành  $R(2^2)$  modulo đa thức  $z^2+1$ . Phép nhân đã được giải thích ở phần trên. Kết quả phép nhân có thể tóm gọn thành bảng kết quả:

$$\begin{bmatrix} b_0 & b_1 & b_2 & b_3 \\ b_4 & b_5 & b_6 & b_7 \end{bmatrix} \text{thành} \begin{bmatrix} b_0 \oplus b_6 & b_1 \oplus b_4 \oplus b_7 & b_2 \oplus b_4 \oplus b_5 & b_3 \oplus b_5 \\ b_2 \oplus b_4 & b_0 \oplus b_3 \oplus b_5 & b_0 \oplus b_1 \oplus b_6 & b_1 \oplus b_7 \end{bmatrix}$$

Hàm ngược của MC như đã phân tích ở phần trên chỉ là kết quả nhân với  $c^{-1}(z) = x \cdot z + x^3 + 1$ . Như vậy,  $MC^{-1} = c^{-1}(z)S$ . Kết quả này cũng có thể tóm gọn thành bảng như trên.

### 3. Giải thuật Tạo mật mã

Giải thuật AES đơn giản hóa có thể tóm gọn thành hàm hợp:

$$A_{K_2} \circ SR \circ NS \circ A_{K_1} \circ MC \circ SR \circ NS \circ A_{K_0}$$

biến đổi trạng thái bản rõ về trạng thái bản mật.

Giải thuật được thực hiện qua 2 vòng tính theo chỉ số của khóa vòng tính  $K_i$ ,  $i=0,1,2$

Vòng tính khởi tạo chỉ đơn giản cộng bit  $A_{K_0}$  với  $K_0$ .

Vòng tính 1 thực hiện các phép biến đổi trạng thái NS, SR, MC và cuối cùng là cộng bit  $A_{K_1}$  với  $K_1$ .

Vòng tính 2 thực hiện tương tự với  $K_2$  nhưng thiếu MC.

### 4. Giải thuật Giải mật mã

Giải mật là quy trình ngược sử dụng các hàm ngược như đã nêu ở phần trên và có dạng:

$$A_{K_0} \circ NS^{-1} \circ SR^{-1} \circ MC^{-1} \circ A_{K_1} \circ NS^{-1} \circ SR^{-1} \circ A_{K_2}$$

Các bước giải mật này gây khó cho cài đặt do quy trình thuận và ngược không sử dụng cùng loại hàm xử lý. Vấn đề: có thể đổi vị trí các hàm cùng loại về cùng thứ tự xử lý ở hai quy trình này không?

Trước hết dễ thấy  $NS^{-1} \circ SR^{-1} = SR^{-1} \circ NS^{-1}$  do các hàm ngược này xử lý tuần tự: Trượt hàng rồi Thay thế hay ngược lại đều như nhau.

Tiếp theo,  $MC^{-1} \circ A_{K_i}(S) = MC^{-1}(A_{K_i}(S)) = MC^{-1}(K_i \oplus S) = \alpha(z)^{-1}(K_i \oplus S) = \alpha(z)^{-1}(K_i) \oplus \alpha(z)^{-1}(S) = \alpha(z)^{-1}(K_i) \oplus MC^{-1}(S) = A_{\alpha(z)^{-1}K_i}(MC^{-1}(S))$ .

Vậy quy trình ngược của giải mật được viết lại một cách tương đương:

$$A_{K_0} \circ SR^{-1} \circ NS^{-1} \circ A_{\alpha(z)^{-1}K_i} \circ MC^{-1} \circ SR^{-1} \circ NS^{-1} \circ A_{K_2}$$

Quy trình này cho phép cài đặt AES dễ dàng hơn rất nhiều do các bước đều sử dụng cùng loại hàm khả nghịch.

### Yêu Cầu

Sinh viên viết chương trình trên để thực hiện các yêu cầu sau:

5. Sinh khóa và lưu khóa
6. Đọc khóa và tạo mật mã / giải mật mã DES (ECB, CBC)
7. Đọc khóa và tạo mật mã / giải mật mã AES (ECB)
8. Xây dựng Demo – giao diện đồ họa (tham khảo buổi 1)

Tham khảo

<https://www.dlitz.net/software/pycrypto/api/2.6/>

## BUỔI 3: : CÀI ĐẶT THƯ VIỆN PYCRYPTO – SỬ DỤNG MÃ HÓA BẤT ĐỐI XỨNG

Sinh khóa, lưu khóa, đọc khóa mà mật mã Diffie–Hellman

Sinh khóa, lưu khóa, đọc khóa và mật mã RSA

- Được phát triển bởi Rivest, Shamir và Adleman.
- Mật mã hóa và giải mật mã được tính theo công thức:
- $C = M^e \bmod n$
- $M = C^d \bmod n$
- Các yêu cầu:
- Có thể tìm được các giá trị  $e$ ,  $d$ ,  $n$  sao cho
- $M^e d \equiv M \pmod{n}$  với mọi  $M < n$
- Dễ dàng tính được  $M^e$  và  $C^d$  với mọi  $M < n$
- Không thể tính được  $d$  từ  $e$  và  $n$
- Giải thuật:
- Chọn 2 số nguyên tố lớn  $p$  và  $q$
- Tính  $n = p * q$
- Tính  $\phi(n) = (p-1) * (q-1)$
- Chọn  $e$  sao cho  $\text{USCLN}(e, \phi(n)) = 1$  với  $1 < e < \phi(n)$
- Tính  $d$  sao cho  $ed \equiv 1 \pmod{\phi(n)}$

### Yêu Cầu

Sinh viên viết chương trình trên để thực hiện các yêu cầu sau:

5. Sinh khóa và lưu khóa
6. Đọc khóa và tạo mật mã / giải mật Diffie–Hellman
7. Đọc khóa và tạo mật mã / giải mật RSA
8. Xây dựng Demo – giao diện đồ họa (tham khảo buổi 1)

Tham khảo

<https://www.dlitz.net/software/pycrypto/api/2.6/>



## BUỔI 4: CÀI ĐẶT THƯ VIỆN PYCRYPTO – ỨNG DỤNG BĂM VÀ CHỮ KÝ SỐ - TRUY TÌM NGUỒN GỐC TẬP TIN

Băm là một giải pháp tạo ra một đặc trưng cho một file dữ liệu. Tương tự như mỗi một con người có một dấu vân tay đặc trưng. Vì vậy Băm còn được gọi dấu vân tay (Fingerprint) của file dữ liệu.

Hàm băm (Hash Function) là một dạng mật mã tạo bản mật không cần giải mật mà đáp ứng yêu cầu kiểm tra tính toàn vẹn của một dữ liệu dựa trên đặc trưng vân tay của nó.

Băm (Hash) là một khối dữ liệu thu nhỏ lại từ một file dữ liệu và được tạo ra bởi hàm băm (Hash Function).

Hàm băm là một phép biến đổi một chiều (One-Way) có đầu vào là dãy  $m+t$  bit và đầu ra là dãy  $t$  bit, trong đó  $t < m$ . Đầu vào của hàm băm gồm file dữ liệu với  $m$  bit và IV (Input Value) với  $t$  bit. Đầu ra chính là giá trị băm với  $t$  bit.

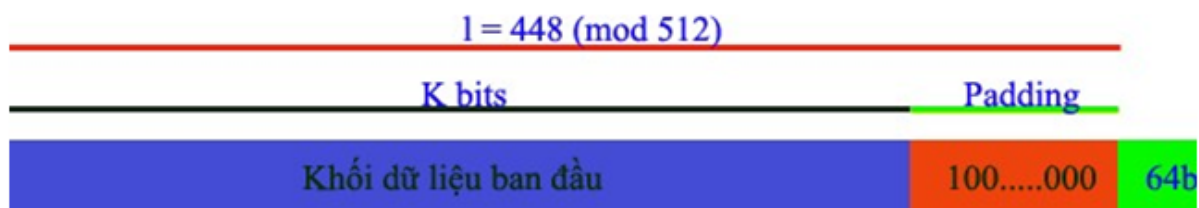
Hàm băm  $H(x)$  có khả năng bảo mật tốt, nếu thỏa 3 tính chất: Một chiều (One Way), Tự do liên kết yếu (Weakly Collision Free) và Tự do liên kết mạnh (Strong Collision Free).

### Giải thuật MD5

- Phát triển bởi Ron Rivest tại đại học MIT
- Input: thông điệp với độ dài bất kỳ
- Output: giá trị băm (message digest) 128 bits
- Giải thuật gồm 5 bước thao tác trên khối 512 bits

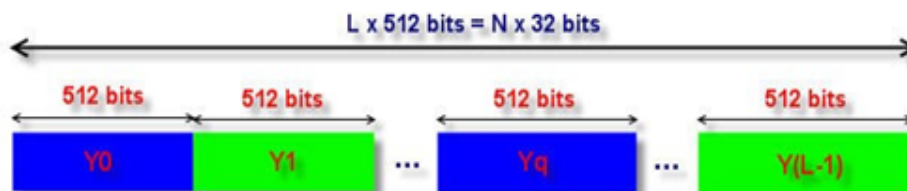
• Bước 1: **nhồi dữ liệu**

- Nhồi thêm các bits sao cho dữ liệu có độ dài  $l \equiv 448 \bmod 512$  hay  $l = n * 512 + 448$  ( $n, l$  nguyên)
- Luôn thực hiện nhồi dữ liệu ngay cả khi dữ liệu ban đầu có độ dài mong muốn. Ví dụ, dữ liệu có độ dài 448 được nhồi thêm 512 bits để được độ dài 960 bits.
- Số lượng bit nhồi thêm nằm trong khoảng 1 đến 512
- Các bit được nhồi gồm 1 bit "1" và các bit 0 theo sau. |



• Bước 2: **thêm vào độ dài**

- Độ dài của khối dữ liệu ban đầu được biểu diễn dưới dạng nhị phân 64-bit và được thêm vào cuối chuỗi nhị phân kết quả của bước 1
- Nếu độ dài của khối dữ liệu ban đầu  $> 2^{64}$ , chỉ 64 bits thấp được sử dụng, nghĩa là giá trị được thêm vào bằng  $K \bmod 2^{64}$
- Kết quả có được từ 2 bước đầu là một khối dữ liệu có độ dài là bội số của 512. Khối dữ liệu được biểu diễn:
  - Bằng một dãy  $L$  khối 512-bit  $Y_0, Y_1, \dots, Y_{L-1}$
  - Bằng một dãy  $N$  từ (word) 32-bit  $M_0, M_1, \dots, M_{N-1}$ . Vậy  $N = L \times 16$  ( $32 \times 16 = 512$ )



- Bước 3: **khởi tạo bộ đệm MD** (MD buffer)
  - Một bộ đệm 128-bit được dùng lưu trữ các giá trị băm trung gian và kết quả. Bộ đệm được biểu diễn bằng 4 thanh ghi 32-bit với các giá trị khởi tạo ở dạng little-endian (byte có trọng số nhỏ nhất trong từ nằm ở địa chỉ thấp nhất) như sau:
    - A = 67 45 23 01
    - B = EF CD AB 89
    - C = 98 BA DC FE
    - D = 10 32 54 76
  - Các giá trị này tương đương với các từ 32-bit sau:
    - A = 01 23 45 67
    - B = 89 AB CD EF
    - C = FE DC BA 98
    - D = 76 54 32 10



#### Bước 4: xử lý các khối dữ liệu 512-bit

- Trọng tâm của giải thuật là **hàm nén** (compression function) gồm 4 "vòng" xử lý. Các vòng này có cấu trúc giống nhau nhưng sử dụng các hàm luận lý khác nhau gồm F, G, H và I
- $F(X,Y,Z) = X \wedge Y \vee \neg X \wedge Z$
- $G(X,Y,Z) = X \wedge Z \vee Y \wedge \neg Z$
- $H(X,Y,Z) = X \text{ xor } Y \text{ xor } Z$
- $I(X,Y,Z) = Y \text{ xor } (X \vee \neg Z)$
- Mảng 64 phần tử được tính theo công thức:  $T[i] = 2^{32} \times \text{abs}(\sin(i))$ ,  $i$  được tính theo radian.
- Kết quả của 4 vòng được cộng (theo modulo  $2^{32}$  với đầu vào  $CV_q$  để tạo  $CV_{q+1}$

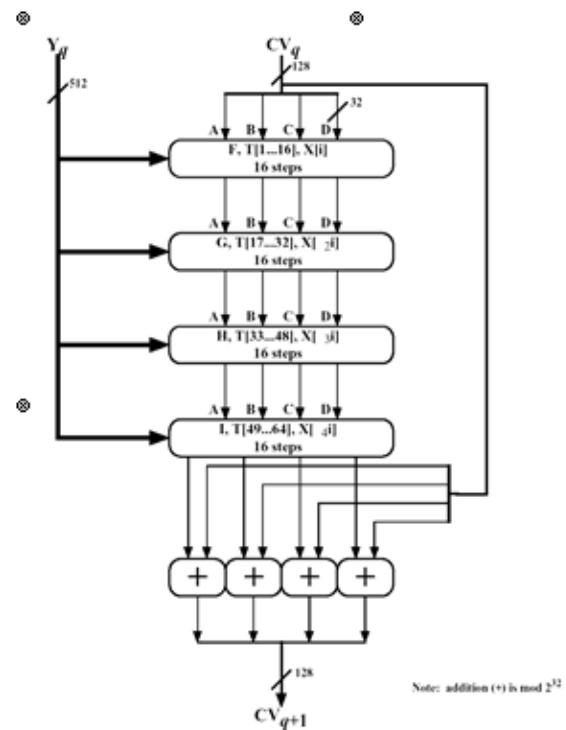


Figure 9.2 MD5 Processing of a Single 512-bit Block (MD5 Compression Function)

#### • Bước 5: Xuất kết quả

- Sau khi xử lý hết L khối 512-bit, đầu ra của lần xử lý thứ L là giá trị băm 128 bits.

#### Giải thuật SHA-1

5. Secure Hash Algorithm (SHA) phát triển bởi National Institute of Standard and Technology (NIST)
6. Đầu vào: thông điệp với độ dài tối đa 264 bits
7. Đầu ra: giá trị băm (message digest) có độ dài 160 bits
8. Giải thuật gồm 5 bước thao tác trên các khối 512 bits

### Bước 1: **nhồi thêm dữ liệu**

- Thông điệp được nhồi thêm các bits sao cho độ dài  $l \equiv 448 \pmod{512}$  hay  $l = n * 512 + 448 (n, l \text{ nguyên})$
- Thông điệp luôn luôn được nhồi thêm dữ liệu
- Số bits nhồi thêm nằm trong khoảng 1 đến 512
- Phần dữ liệu nhồi thêm bao gồm một bit 1 và theo sau là các bit 0

### Bước 2: **thêm vào độ dài**

- Độ dài của khối dữ liệu ban đầu được biểu diễn dưới dạng nhị phân 64-bit và được thêm vào cuối chuỗi nhị phân kết quả của bước 1
- Độ dài được biểu diễn dưới dạng nhị phân 64-bit không dấu
- Kết quả có được từ 2 bước đầu là một khối dữ liệu có độ dài là bội số của 512. Khối dữ liệu được biểu diễn:
  - Bằng một dãy  $L$  khối 512-bit  $Y_0, Y_1, \dots, Y_{L-1}$
  - Bằng một dãy  $N$  từ (word) 32-bit  $M_0, M_1, M_{N-1}$ . Vậy  $N = L \times 16$

### Bước 3: **khởi tạo bộ đệm MD** (MD buffer)

- Một bộ đệm 160-bit được dùng lưu trữ các giá trị băm trung gian và kết quả. Bộ đệm được biểu diễn bằng 5 thanh ghi 32-bit với các giá trị khởi tạo ở dạng big-endian (byte có trọng số lớn nhất trong từ nằm ở địa chỉ thấp nhất) như sau:
  - $A = 01\ 23\ 45\ 67$
  - $B = 89\ AB\ CD\ EF$
  - $C = FE\ DC\ BA\ 98$
  - $D = 76\ 54\ 32\ 10$
  - $E = C3\ D2\ E1\ F0$
- Các giá trị này tương đương với các từ 32-bit sau:
  - $A = 01\ 23\ 45\ 67$
  - $B = 89\ AB\ CD\ EF$
  - $C = FE\ DC\ BA\ 98$
  - $D = 76\ 54\ 32\ 10$
  - $E = C3\ D2\ E1\ F0$

#### Bước 4: xử lý các khối dữ liệu 512-bit

- Trọng tâm của giải thuật bao gồm 4 lặp thực hiện tất cả 80 bước.
- 4 vòng lặp có cấu trúc như nhau, chỉ nhau ở các hàm logic  $f_1, f_2, f_3, f_4$
- Mỗi vòng có đầu vào gồm khối 512-thời và một bộ đệm 160-bit ABCDE. Mỗi thao tác sẽ cập nhật giá trị bộ đệm
- Mỗi bước sử dụng một hằng số  $K_t$  (0 ≤ t ≤ 79)
  - $K_t = 5A827999$  (0 ≤ t ≤ 19)
  - $K_t = 6ED9EBA1$  (20 ≤ t ≤ 39)
  - $K_t = 8F1BBCDC$  (40 ≤ t ≤ 59)
  - $K_t = CA62C1D6$  (60 ≤ t ≤ 79)
- Đầu ra của 4 vòng (bước 80) được cộng đầu ra của bước  $CV_q$  để tạo ra  $CV_{q+1}$

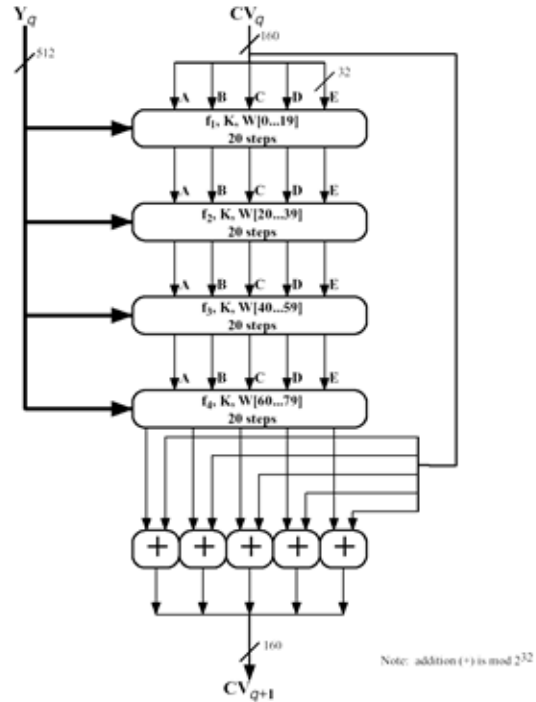


Figure 9.5 SHA-1 Processing of a Single 512-bit Block (SHA-1 Compression Function)

#### Bước 5: xuất kết quả

- Sau khi thao tác trên toàn bộ L blocks. Kết quả của khối thứ L là bảng băm 160-bit

##### Yêu Cầu

Sinh viên viết chương trình trên để thực hiện các yêu cầu sau:

5. Băm file dữ liệu: MD5 và SHA 1 và kiểm tra toàn vẹn
6. Sử dụng RSA tạo chữ ký số cho file dữ liệu
7. Kiểm tra chữ ký số và xác nhận nguồn gốc file dữ liệu
8. Xây dựng Demo – giao diện đồ họa (tham khảo buổi 1)

## BUỔI 5: XÂY DỰNG CHƯƠNG TRÌNH TẠO CHỮ KÝ SỐ BẰNG NGÔN NGỮ PYTHON – SỬ DỤNG RSA VÀ HASH 256

Yêu Cầu

Sinh viên viết chương trình – làm bài tập nhóm

## ĐỀ CƯƠNG THỰC HÀNH MÔN AT&BMTT

### MỤC TIÊU

Nắm vững kiến thức về an toàn bảo mật thông tin

### NỘI DUNG

Buổi 1: Lập Trình Python

Buổi 2: Trình bày chi tiết các bước mã hóa, giải mã kỹ thuật mã hóa Des, Xây dựng chương trình Demo

Buổi 3: Cài Đặt Thư Viện Pycrypto – Sử Dụng Mã Hóa Đối Xứng

Buổi 4: Cài Đặt Thư Viện Pycrypto – Sử Dụng Mã Hóa Bất Đối Xứng

Buổi 5: Cài Đặt Thư Viện Pycrypto – Ứng Dụng Băm Và Chữ Ký Số - Truy Tìm Nguồn Gốc Tập Tin

Buổi 6: Xây Dựng Ứng Dụng Demo Các Giải Thuật Mã Hóa và Giải Mã – Nộp Bài

Tài liệu tham khảo:

Hướng dẫn lập trình ngôn ngữ python [www.cit.ctu.edu.vn/~dtngchi/oss/python.pdf](http://www.cit.ctu.edu.vn/~dtngchi/oss/python.pdf)

# BUỔI 1: LẬP TRÌNH PYTHON

## Giới thiệu OpenSSL

### 7. Giới thiệu

Python là một ngôn ngữ lập trình dạng thông dịch do Guido Van Rossum tạo ra năm 1990. Python được viết từ nhiều ngôn ngữ lập trình khác nhau và tạo ra những bản hiện thực khác nhau. Trong đó bản hiện thực chính là CPython được viết bằng C.

Python có tính tương thích lớn trên nhiều môi trường phát triển và cũng được ứng dụng rộng rãi trên nhiều lĩnh vực.

Điểm khác biệt đặc trưng giữa C và Python là Python sử dụng cơ chế cấp phát bộ nhớ tự động và hoàn toàn tạo kiểu động (khác với C phải khai báo biến và cấp phát bộ nhớ tương ứng với kiểu của biến). Điều này giúp tối thiểu hóa số lần gõ phím để viết mã lệnh và giúp cho cấu trúc có hình thức gọn gàng sáng sủa rất thuận tiện cho người mới học lập trình.

### 8. Cài đặt Python và trình soạn thảo

Download Python : Truy cập vào trang web : <https://www.python.org/> để download Python phiên bản 2.7.13

#### Chú ý : Cài đặt phiên bản 2.7.13 chứ không cài phiên bản 3.6

Sau khi cài đặt python ta cần thêm Python vào PATH của hệ thống bằng các bước sau : (chỉ thay đổi khi cài đặt trên môi trường window)

Mở Control Panel


- Chọn System
- Chọn Advanced System Setting
- Chọn tab Advance > Environment Variable
- Trong mục Systeme Variable, tìm chọn variable PATH, copy paste dòng sau vào value của variable này :

`;C:\Python27;C:\Python27\Lib\site-packages\;C:\Python27\Scripts\`

- Chọn Ok và kết thúc việc thêm python vào PATH của hệ thống

Download trình soạn thảo : Truy cập vào trang web : <https://atom.io/> để download trình soạn thảo Atom

Cách sử dụng Atom để viết một chương trình đơn giản bằng Python :

- Tạo một thư mục trong ổ D và đặt tên là Python
- Mở Atom, từ menu File chọn New File, ta thấy một cửa sổ mới được tạo ra.
- Bấm tổ hợp Ctrl + S để save file đó vào thư mục Python vừa tạo ở ổ đĩa D với tên gọi hello.py
- Trên cửa sổ soạn thảo của Atom viết vào dòng lệnh sau : `print "Hello Python"` và bấm Ctrl + S để save lại
- Mở cửa sổ terminal (bấm tổ hợp  + R, gõ vào `cmd`).
- Trên cửa sổ terminal, gõ vào dòng lệnh `cd D:\Python` để di chuyển đến thư mục mà ta vừa tạo
- (Nếu báo lỗi thì thực hiện theo 2 bước, ta gõ `D:` để di chuyển qua ổ đĩa D, sau đó ta gõ

## 9. Cài đặt thư viện trong python

- Nâng cấp PIP bằng lệnh python get-pip.py
- Cài đặt các thư viện bằng lệnh pip install <tên thư viện>

Yêu cầu. Viết chương trình để thực hiện các công việc

Cài đặt mật mã affine bằng ngôn ngữ python

Áp dụng trên chương trình giao diện đồ họa cơ bản

Ví dụ



Code

```
# -*- coding: utf-8 -*-
from Tkinter import *
import ttk

window = Tk()

window.title("Welcome to Demo An Toan Bao Mat Thong Tin")

lb0 = Label(window, text="CHƯƠNG TRÌNH DEMO", font=("Arial Bold", 10))
lb0.grid(column=0, row=0)

lb1 = Label(window, text="CHƯƠNG TRÌNH DEMO", font=("Arial Bold", 20))
lb1.grid(column=1, row=1)

lb2 = Label(window, text="MẬT MÃ AFINE", font=("Arial Bold", 15))
lb2.grid(column=0, row=2)

plainlb3 = Label(window, text="PLAIN TEXT", font=("Arial", 14))
plainlb3.grid(column=0, row=3)

plaintxt = Entry(window, width=20)
```

```

plaintxt.grid(column=1, row=3)
KEYlb4 = Label(window, text="KEY PAIR",font=("Arial", 14))
KEYlb4.grid(column=2, row=3)
KEYA1 = Entry(window,width=3)
KEYA1.grid(column=3, row=3)
KEYB1 = Entry(window,width=5)
KEYB1.grid(column=4, row=3)
lb5 = Label(window, text="CIPHER TEXT",font=("Arial", 14))
lb5.grid(column=0, row=4)
ciphertxt3 = Entry(window,width=20)
ciphertxt3.grid(column=1, row=4)
denctxt3 = Entry(window,width=20)
denctxt3.grid(column=3, row=4)
def Char2Num(c): return ord(c)-65
def Num2Char(n): return chr(n+65)
def xgcd(b,a):
    tmp=a
    x0, x1, y0, y1 = 1, 0, 0, 1
    while a!=0:
        q, b, a = b // a, a, b % a
        x0, x1 = x1, x0 - q * x1
        y0, y1 = y1, y0 - q * y1
    if x0<0:x0=tmp+x0
    return x0
def encryptAF(txt,a,b,m):
    r=""
    for c in txt:
        e=(a*Char2Num(c)+b )%m

```



```

        r=r+Num2Char(e)

    return r

def decryptAF(txt,a,b,m):
    r=""
    a1=xgcd(a,m)
    for c in txt:
        e=(a1*(Char2Num(c)-b ))%m
        r=r+Num2Char(e)
    return r

def clicked():
    a,b,m=int(KEYA1.get()),int(KEYB1.get()),26
    entxt=encryptAF(plaintext.get(),a,b,m)
    ciphertxt3.delete(0,END)
    #a=int(KEYA1.get())
    ciphertxt3.insert(INSERT,entxt)

def giaima():
    a,b,m=int(KEYA1.get()),int(KEYB1.get()),26
    detxt=decryptAF(ciphertxt3.get(),a,b,m)
    denctxt3.delete(0,END)
    #a=int(KEYA1.get())
    denctxt3.insert(INSERT,detxt)

    AFbtn = Button(window, text="Mã Hóa", command=clicked)

AFbtn.grid(column=5, row=3)
DEAFbtn = Button(window, text="Giải Mã ", command=giaima)
DEAFbtn.grid(column=2, row=4)
window.geometry('800x600')
window.mainloop()

```

## BUỔI 2: CÀI ĐẶT THƯ VIỆN PYCRYPTO – SỬ DỤNG MÃ HÓA ĐỐI XỨNG

### Cơ sở lý thuyết

Bản rõ:  $P \in X$ , bản mật:  $C \in Y$  và khóa:  $k, k' \in K$ .

Mật mã là một cặp đơn ánh và ánh xạ ngược của nó với các tham số tương ứng ( $fk, f^{-1}k'$ ).

Tạo mật mã  $fk: X \rightarrow Y$ , hay  $P \xrightarrow{k} C$ .

Giải mật mã  $f^{-1}k': Y \rightarrow X$ , hay  $C \xrightarrow{k'} P$ .

### Phân loại

Mật mã khóa đối xứng (Symmetric Key Cryptography):  $k=k'$ .

Mật mã khóa bất đối xứng (Asymmetric Key Cryptography):  $k \neq k'$ . Tên khác Mật mã khóa công khai (Public Key Cryptography).

Hàm băm (Hash Function): hàm đặc biệt một chiều, thu nhỏ dữ liệu và không có ánh xạ ngược. Ý nghĩa: tạo vết riêng cho tập dữ liệu được trực quan thành Dấu vân tay (Fingerprint).

### Mục tiêu:

Sinh viên hiểu rõ về cách mã hóa – giải mã dữ liệu đối xứng

### Mật mã DES

DES được công nhận vào năm 1977 bởi Viện nghiên cứu quốc gia về chuẩn của Mỹ (NIST – National Institute of Standards and Technology)

### Nguyên lý:

Sử dụng một khóa  $K$  tạo ra  $n$  khóa con  $K_1, K_2, \dots, K_n$

Hoán vị dữ liệu (Initial Permutation)

Thực hiện  $n$  vòng lặp, ở mỗi vòng lặp

Dữ liệu được chia thành hai phần

Áp dụng phép toán thay thế lên một phần, phần còn lại giữ nguyên

Hoán vị 2 phần cho nhau (trái / phải)

Hoán vị dữ liệu (Final Permutation)

DES – Tóm tắt giải thuật

- Tạo 16 khóa con

$C[0]D[0] = PC-1(KEY)$

for  $i = 1$  to 16

$C[i] = \text{LeftShift}[i](C[i-1])$

$D[i] = \text{LeftShift}[i](D[i-1])$

$K[i] = \text{PC-2}(C[i]D[i])$

end for

- Mã hóa khối dữ liệu  $L[0]R[0] = \text{IP}(\text{plain block})$  for  $i=1$  to 16

$L[i] = R[i-1]$

$R[i] = L[i-1] \text{ XOR } F(R[i-1], K[i])$

end for

cipher block =  $\text{FP}(R[16]L[16])$

- Giải mã khối dữ liệu  $R[16]L[16] = \text{IP}(\text{cipher block})$  for  $i=1$  to 16

$R[i-1] = L[i]$

$L[i-1] = R[i] \text{ xor } f(L[i], K[i])$

end for

plain block =  $\text{FP}(L[0]R[0])$

## Mật mã AES

AES sử dụng khối 128 bit và khóa 128 bit được phát triển bởi cơ quan NIST (National Institute of Standards and Technology, USA) vào năm 2001, từ mật mã Rijndael của hai nhà toán học Joan Daemen and Vincent Rijmen người Hà lan. Khóa mật AES có thể nâng cấp tùy sự phát triển của máy tính. Hiện nay khóa an toàn nhất của AES là 256 bit.

## Giải thuật AES đơn giản hóa

### 1. Trạng thái xử lý

Dữ liệu xử lý dài 2 byte (16 bit) được gọi là trạng thái xử lý (State). Trạng thái 2 byte được cấu trúc lại thành ma trận nibble 2x2 và được xếp thứ tự theo cột. Bản rõ:  $P=p_0p_1\dots p_{15}$  và bản mật:  $C=c_0c_1\dots c_{15}$  là các dãy 16 bit được xếp thành trạng thái ban đầu và kết thúc.

$$\text{Bản rõ có trạng thái ban đầu: } S_p = \begin{bmatrix} p_0p_1p_2p_3 & p_8p_9p_{10}p_{11} \\ p_4p_5p_6p_7 & p_{12}p_{13}p_{14}p_{15} \end{bmatrix}$$

$$\text{Bản mật có trạng thái kết thúc: } S_c = \begin{bmatrix} c_0c_1c_2c_3 & c_8c_9c_{10}c_{11} \\ c_4c_5c_6c_7 & c_{12}c_{13}c_{14}c_{15} \end{bmatrix}$$

$$\text{Trạng thái trung gian: } S = \begin{bmatrix} N_0 & N_2 \\ N_1 & N_3 \end{bmatrix}$$

### 2. Các hàm xử lý trạng thái

(1) Hàm Cộng khóa (Add Key):  $A_{K_i} = S \oplus S_{K_i}$ , trong đó  $S_{K_i}$  xếp khóa  $K_i$  (16 bit) theo cấu trúc trạng thái. Do hàm  $A_{K_i}$  có tính chất đặc biệt:  $A_{K_i} \circ A_{K_i} = S \oplus S_{K_i} \oplus S_{K_i} = S$ , nên  $A_{K_i}^{-1} = A_{K_i}$ .

(2) Hàm Thay thế nibble (Nibble Substitution): NS thay thế các nibble trong trạng thái  $S$  bởi các  $Sbox(N)$  và có dạng:

$$S = \begin{bmatrix} N_0 & N_2 \\ N_1 & N_3 \end{bmatrix} \text{ chuyển thành } Sbox = \begin{bmatrix} Sbox(N_0) & Sbox(N_2) \\ Sbox(N_1) & Sbox(N_3) \end{bmatrix}$$

Hàm này có hàm ngược tính theo giải thuật LUT trên là S-box hay  $Sbox(N) = g(f(N))$  suy ra:  $N = f^{-1}(g^{-1}(Sbox))$ .

(3) Hàm Trượt hàng (Shift Row): SR trượt hàng thứ 2 sang phải 1 nibble tương đương với hoán vị các nibble trên hàng 2 như hình dưới đây:

$$S = \begin{bmatrix} N_0 & N_2 \\ N_1 & N_3 \end{bmatrix} \text{ chuyển thành } SR = \begin{bmatrix} N_0 & N_2 \\ N_3 & N_1 \end{bmatrix}$$

Hàm ngược  $SR^{-1}$  trượt hàng 2 của SR ngược lại sang trái 1 nibble quay về  $S$ .

(4) Hàm Trộn cột (Mix Column): MC biến đổi mỗi cột của trạng thái  $S$  bằng cách nhân với  $c(z)=x^2z+1$  trong vành  $R(2^2)$  modulo đa thức  $z^2+1$ . Phép nhân đã được giải thích ở phần trên. Kết quả phép nhân có thể tóm gọn thành bảng kết quả:

$$\begin{bmatrix} b_0 & b_1 & b_2 & b_3 \\ b_4 & b_5 & b_6 & b_7 \end{bmatrix} \text{thành} \begin{bmatrix} b_0 \oplus b_6 & b_1 \oplus b_4 \oplus b_7 & b_2 \oplus b_4 \oplus b_5 & b_3 \oplus b_5 \\ b_2 \oplus b_4 & b_0 \oplus b_3 \oplus b_5 & b_0 \oplus b_1 \oplus b_6 & b_1 \oplus b_7 \end{bmatrix}$$

Hàm ngược của MC như đã phân tích ở phần trên chỉ là kết quả nhân với  $c^{-1}(z)=x.z+x^3+1$ . Như vậy,  $MC^{-1}=c^{-1}(z)S$ . Kết quả này cũng có thể tóm gọn thành bảng như trên.

### 3. Giải thuật Tạo mật mã

Giải thuật AES đơn giản hóa có thể tóm gọn thành hàm hợp:

$$A_{K_2} \circ SR \circ NS \circ A_{K_1} \circ MC \circ SR \circ NS \circ A_{K_0}$$

biến đổi trạng thái bản rõ về trạng thái bản mật.

Giải thuật được thực hiện qua 2 vòng tính theo chỉ số của khóa vòng tính  $K_i$ ,  $i=0,1,2$

Vòng tính khởi tạo chỉ đơn giản cộng bit  $A_{K_0}$  với  $K_0$ .

Vòng tính 1 thực hiện các phép biến đổi trạng thái NS, SR, MC và cuối cùng là cộng bit  $A_{K_1}$  với  $K_1$ .

Vòng tính 2 thực hiện tương tự với  $K_2$  nhưng thiếu MC.

### 4. Giải thuật Giải mật mã

Giải mật là quy trình ngược sử dụng các hàm ngược như đã nêu ở phần trên và có dạng:

$$A_{K_0} \circ NS^{-1} \circ SR^{-1} \circ MC^{-1} \circ A_{K_1} \circ NS^{-1} \circ SR^{-1} \circ A_{K_2}$$

Các bước giải mật này gây khó cho cài đặt do quy trình thuận và ngược không sử dụng cùng loại hàm xử lý. Vấn đề: có thể đổi vị trí các hàm cùng loại về cùng thứ tự xử lý ở hai quy trình này không ?

Trước hết dễ thấy  $NS^{-1} \circ SR^{-1} = SR^{-1} \circ NS^{-1}$  do các hàm ngược này xử lý tuần tự: Trượt hàng rồi Thay thế hay ngược lại đều như nhau.

Tiếp theo,  $MC^{-1} \circ A_{K_i}(S) = MC^{-1}(A_{K_i}(S)) = MC^{-1}(K_i \oplus S) = \alpha(z)^{-1}(K_i \oplus S) = \alpha(z)^{-1}(K_i) \oplus \alpha(z)^{-1}(S) = \alpha(z)^{-1}(K_i) \oplus MC^{-1}(S) = A_{\alpha(z)^{-1}K_i}(MC^{-1}(S))$ .

Vậy quy trình ngược của giải mật được viết lại một cách tương đương:

$$A_{K_0} \circ SR^{-1} \circ NS^{-1} \circ A_{\alpha(z)^{-1}K_i} \circ MC^{-1} \circ SR^{-1} \circ NS^{-1} \circ A_{K_2}$$

Quy trình này cho phép cài đặt AES dễ dàng hơn rất nhiều do các bước đều sử dụng cùng loại hàm khả nghịch.

### Yêu Cầu

Sinh viên viết chương trình trên để thực hiện các yêu cầu sau:

9. Sinh khóa và lưu khóa
10. Đọc khóa và tạo mật mã / giải mật mã DES (ECB, CBC)
11. Đọc khóa và tạo mật mã / giải mật mã AES (ECB)
12. Xây dựng Demo – giao diện đồ họa (tham khảo buổi1)

Tham khảo

<https://www.dlitz.net/software/pycrypto/api/2.6/>

## BUỔI 3: : CÀI ĐẶT THƯ VIỆN PYCRYPTO – SỬ DỤNG MÃ HÓA BẤT ĐỐI XỨNG

Sinh khóa, lưu khóa, đọc khóa mà mật mã Diffie–Hellman

Sinh khóa, lưu khóa, đọc khóa và mật mã RSA

- Được phát triển bởi Rivest, Shamir và Adleman.
- Mật mã hóa và giải mật mã được tính theo công thức:
- $C = M^e \pmod n$
- $M = C^d \pmod n$
- Các yêu cầu:
- Có thể tìm được các giá trị  $e, d, n$  sao cho
- $M^e d \equiv M \pmod n$  với mọi  $M < n$
- Dễ dàng tính được  $M^e$  và  $C^d$  với mọi  $M < n$
- Không thể tính được  $d$  từ  $e$  và  $n$
- Giải thuật:
- Chọn 2 số nguyên tố lớn  $p$  và  $q$
- Tính  $n = p * q$
- Tính  $\phi(n) = (p-1) * (q-1)$
- Chọn  $e$  sao cho  $\text{USCLN}(e, \phi(n)) = 1$  với  $1 < e < \phi(n)$
- Tính  $d$  sao cho  $ed \equiv 1 \pmod{\phi(n)}$

### Yêu Cầu

Sinh viên viết chương trình trên để thực hiện các yêu cầu sau:

9. Sinh khóa và lưu khóa
10. Đọc khóa và tạo mật mã / giải mật Diffie–Hellman
11. Đọc khóa và tạo mật mã / giải mật RSA
12. Xây dựng Demo – giao diện đồ họa (tham khảo buổi 1)

Tham khảo

<https://www.dlitz.net/software/pycrypto/api/2.6/>



## BUỔI 4: CÀI ĐẶT THƯ VIỆN PYCRYPTO – ỨNG DỤNG BĂM VÀ CHỮ KÝ SỐ - TRUY TÌM NGUỒN GỐC TẬP TIN

Băm là một giải pháp tạo ra một đặc trưng cho một file dữ liệu. Tương tự như mỗi một con người có một dấu vân tay đặc trưng. Vì vậy Băm còn được gọi dấu vân tay (Fingerprint) của file dữ liệu.

Hàm băm (Hash Function) là một dạng mật mã tạo bản mật không cần giải mật mà đáp ứng yêu cầu kiểm tra tính toàn vẹn của một dữ liệu dựa trên đặc trưng vân tay của nó.

Băm (Hash) là một khối dữ liệu thu nhỏ lại từ một file dữ liệu và được tạo ra bởi hàm băm (Hash Function).

Hàm băm là một phép biến đổi một chiều (One-Way) có đầu vào là dãy  $m+t$  bit và đầu ra là dãy  $t$  bit, trong đó  $t < m$ . Đầu vào của hàm băm gồm file dữ liệu với  $m$  bit và IV (Input Value) với  $t$  bit. Đầu ra chính là giá trị băm với  $t$  bit.

Hàm băm  $H(x)$  có khả năng bảo mật tốt, nếu thỏa 3 tính chất: Một chiều (One Way), Tự do liên kết yếu (Weakly Collision Free) và Tự do liên kết mạnh (Strong Collision Free).

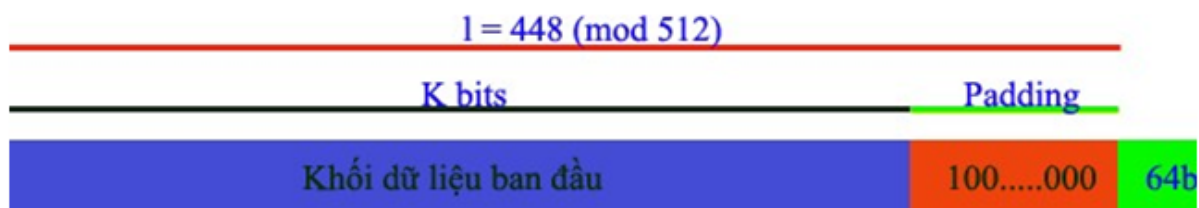
### Giải thuật MD5

- Phát triển bởi Ron Rivest tại đại học MIT
- Input: thông điệp với độ dài bất kỳ
- Output: giá trị băm (message digest) 128 bits
- Giải thuật gồm 5 bước thao tác trên khối 512 bits



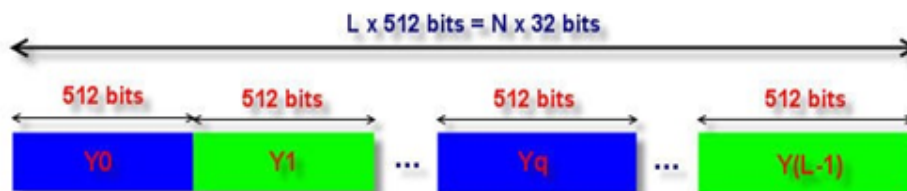
• Bước 1: **nhồi dữ liệu**

- Nhồi thêm các bits sao cho dữ liệu có độ dài  $l \equiv 448 \bmod 512$  hay  $l = n * 512 + 448$  ( $n, l$  nguyên)
- Luôn thực hiện nhồi dữ liệu ngay cả khi dữ liệu ban đầu có độ dài mong muốn. Ví dụ, dữ liệu có độ dài 448 được nhồi thêm 512 bits để được độ dài 960 bits.
- Số lượng bit nhồi thêm nằm trong khoảng 1 đến 512
- Các bit được nhồi gồm 1 bit "1" và các bit 0 theo sau. |



• Bước 2: **thêm vào độ dài**

- Độ dài của khối dữ liệu ban đầu được biểu diễn dưới dạng nhị phân 64-bit và được thêm vào cuối chuỗi nhị phân kết quả của bước 1
- Nếu độ dài của khối dữ liệu ban đầu  $> 2^{64}$ , chỉ 64 bits thấp được sử dụng, nghĩa là giá trị được thêm vào bằng  $K \bmod 2^{64}$
- Kết quả có được từ 2 bước đầu là một khối dữ liệu có độ dài là bội số của 512. Khối dữ liệu được biểu diễn:
  - Bằng một dãy  $L$  khối 512-bit  $Y_0, Y_1, \dots, Y_{L-1}$
  - Bằng một dãy  $N$  từ (word) 32-bit  $M_0, M_1, \dots, M_{N-1}$ . Vậy  $N = L \times 16$  ( $32 \times 16 = 512$ )



- Bước 3: **khởi tạo bộ đệm MD** (MD buffer)
  - Một bộ đệm 128-bit được dùng lưu trữ các giá trị băm trung gian và kết quả. Bộ đệm được biểu diễn bằng 4 thanh ghi 32-bit với các giá trị khởi tạo ở dạng little-endian (byte có trọng số nhỏ nhất trong từ nằm ở địa chỉ thấp nhất) như sau:
    - A = 67 45 23 01
    - B = EF CD AB 89
    - C = 98 BA DC FE
    - D = 10 32 54 76
  - Các giá trị này tương đương với các từ 32-bit sau:
    - A = 01 23 45 67
    - B = 89 AB CD EF
    - C = FE DC BA 98
    - D = 76 54 32 10

#### Bước 4: xử lý các khối dữ liệu 512-bit

- Trọng tâm của giải thuật là **hàm nén** (compression function) gồm 4 "vòng" xử lý. Các vòng này có cấu trúc giống nhau nhưng sử dụng các hàm luận lý khác nhau gồm F, G, H và I
- $F(X,Y,Z) = X \wedge Y \vee \neg X \wedge Z$
- $G(X,Y,Z) = X \wedge Z \vee Y \wedge \neg Z$
- $H(X,Y,Z) = X \text{ xor } Y \text{ xor } Z$
- $I(X,Y,Z) = Y \text{ xor } (X \vee \neg Z)$
- Mảng 64 phần tử được tính theo công thức:  $T[i] = 2^{32} \times \text{abs}(\sin(i))$ ,  $i$  được tính theo radian.
- Kết quả của 4 vòng được cộng (theo modulo  $2^{32}$  với đầu vào  $CV_q$  để tạo  $CV_{q+1}$

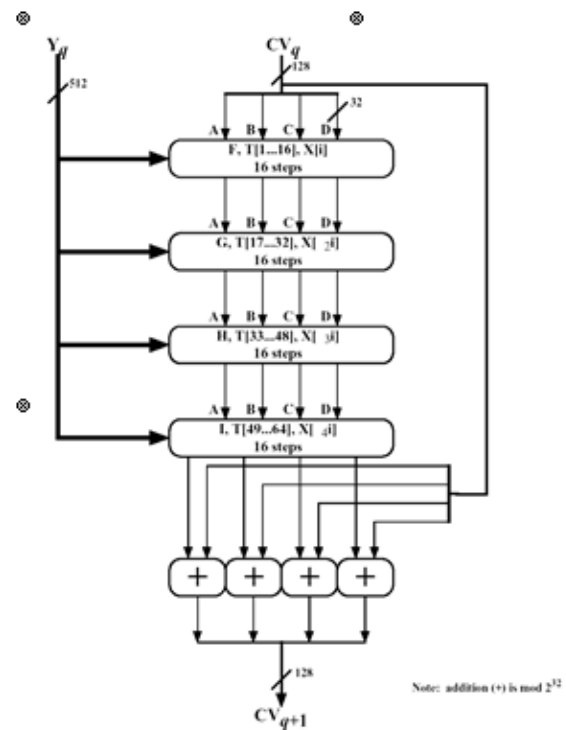


Figure 9.2 MD5 Processing of a Single 512-bit Block (MD5 Compression Function)

#### • Bước 5: Xuất kết quả

- Sau khi xử lý hết L khối 512-bit, đầu ra của lần xử lý thứ L là giá trị băm 128 bits.

#### Giải thuật SHA-1

- Secure Hash Algorithm (SHA) phát triển bởi National Institute of Standard and Technology (NIST)
- Đầu vào: thông điệp với độ dài tối đa 264 bits
- Đầu ra: giá trị băm (message digest) có độ dài 160 bits
- Giải thuật gồm 5 bước thao tác trên các khối 512 bits

### Bước 1: **nhồi thêm dữ liệu**

- Thông điệp được nhồi thêm các bits sao cho độ dài  $l \equiv 448 \pmod{512}$  hay  $l = n * 512 + 448 (n, l \text{ nguyên})$
- Thông điệp luôn luôn được nhồi thêm dữ liệu
- Số bits nhồi thêm nằm trong khoảng 1 đến 512
- Phần dữ liệu nhồi thêm bao gồm một bit 1 và theo sau là các bit 0

### Bước 2: **thêm vào độ dài**

- Độ dài của khối dữ liệu ban đầu được biểu diễn dưới dạng nhị phân 64-bit và được thêm vào cuối chuỗi nhị phân kết quả của bước 1
- Độ dài được biểu diễn dưới dạng nhị phân 64-bit không dấu
- Kết quả có được từ 2 bước đầu là một khối dữ liệu có độ dài là bội số của 512. Khối dữ liệu được biểu diễn:
  - Bằng một dãy  $L$  khối 512-bit  $Y_0, Y_1, \dots, Y_{L-1}$
  - Bằng một dãy  $N$  từ (word) 32-bit  $M_0, M_1, M_{N-1}$ . Vậy  $N = L \times 16$

### Bước 3: **khởi tạo bộ đệm MD** (MD buffer)

- Một bộ đệm 160-bit được dùng lưu trữ các giá trị băm trung gian và kết quả. Bộ đệm được biểu diễn bằng 5 thanh ghi 32-bit với các giá trị khởi tạo ở dạng big-endian (byte có trọng số lớn nhất trong từ nằm ở địa chỉ thấp nhất) như sau:
  - $A = 01\ 23\ 45\ 67$
  - $B = 89\ AB\ CD\ EF$
  - $C = FE\ DC\ BA\ 98$
  - $D = 76\ 54\ 32\ 10$
  - $E = C3\ D2\ E1\ F0$
- Các giá trị này tương đương với các từ 32-bit sau:
  - $A = 01\ 23\ 45\ 67$
  - $B = 89\ AB\ CD\ EF$
  - $C = FE\ DC\ BA\ 98$
  - $D = 76\ 54\ 32\ 10$
  - $E = C3\ D2\ E1\ F0$

#### Bước 4: xử lý các khối dữ liệu 512-bit

- Trọng tâm của giải thuật bao gồm 4 lặp thực hiện tất cả 80 bước.
- 4 vòng lặp có cấu trúc như nhau, chỉ nhau ở các hàm logic  $f_1, f_2, f_3, f_4$
- Mỗi vòng có đầu vào gồm khối 512-thời và một bộ đệm 160-bit ABCDE. Mỗi thao tác sẽ cập nhật giá trị bộ đệm
- Mỗi bước sử dụng một hằng số  $K_t$  (0 79)
  - $K_t = 5A827999$  ( $0 \leq t \leq 19$ )
  - $K_t = 6ED9EBA1$  ( $20 \leq t \leq 39$ )
  - $K_t = 8F1BBCDC$  ( $40 \leq t \leq 59$ )
  - $K_t = CA62C1D6$  ( $60 \leq t \leq 79$ )
- Đầu ra của 4 vòng (bước 80) được cộng đầu ra của bước  $CV_q$  để tạo ra  $CV_{q+1}$

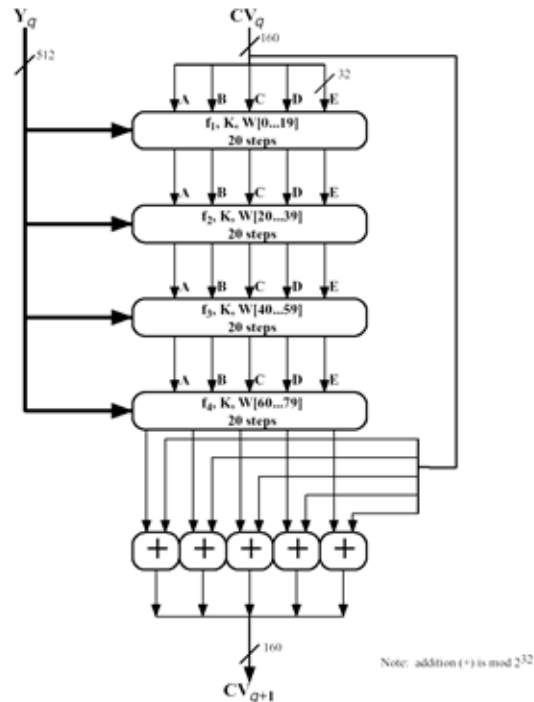


Figure 9.5 SHA-1 Processing of a Single 512-bit Block (SHA-1 Compression Function)

#### Bước 5: xuất kết quả

- Sau khi thao tác trên toàn bộ L blocks. Kết quả của khối thứ L là bảng băm 160-bit

##### Yêu Cầu

Sinh viên viết chương trình trên để thực hiện các yêu cầu sau:

9. Băm file dữ liệu: MD5 và SHA 1 và kiểm tra toàn vẹn
10. Sử dụng RSA tạo chữ ký số cho file dữ liệu
11. Kiểm tra chữ ký số và xác nhận nguồn gốc file dữ liệu
12. Xây dựng Demo – giao diện đồ họa (tham khảo buổi 1

## BUỔI 5: XÂY DỰNG CHƯƠNG TRÌNH TẠO CHỮ KÝ SỐ BẰNG NGÔN NGỮ PYTHON – SỬ DỤNG RSA VÀ HASH 256

Yêu Cầu

Sinh viên viết chương trình – làm bài tập nhóm

## BUỔI 6: KIỂM TRA