**MARS Introduction**

**And**

**Hybrid encryption**

第六組

1052990 黃智優

1052994 黃建鴻

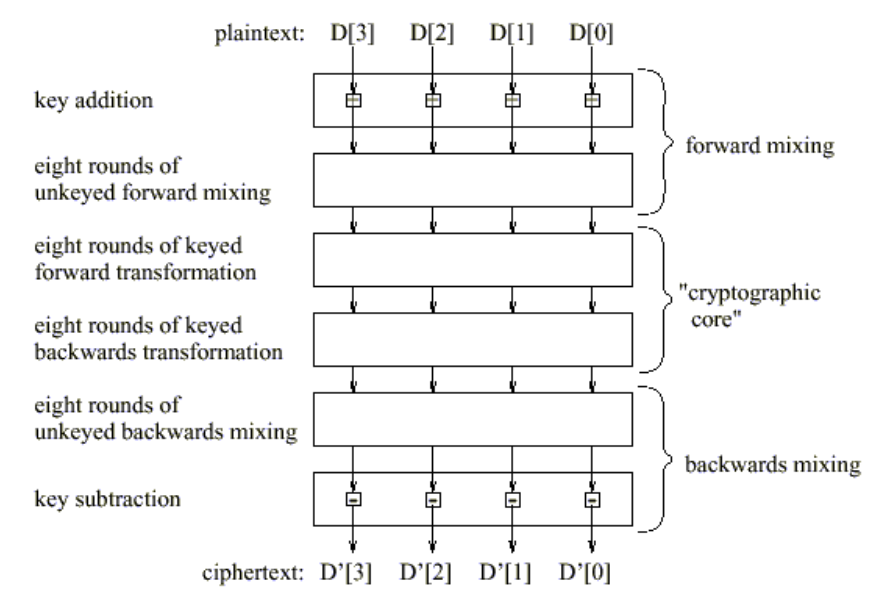
1052999 楊昆瀚

1053001 蔡翰霆

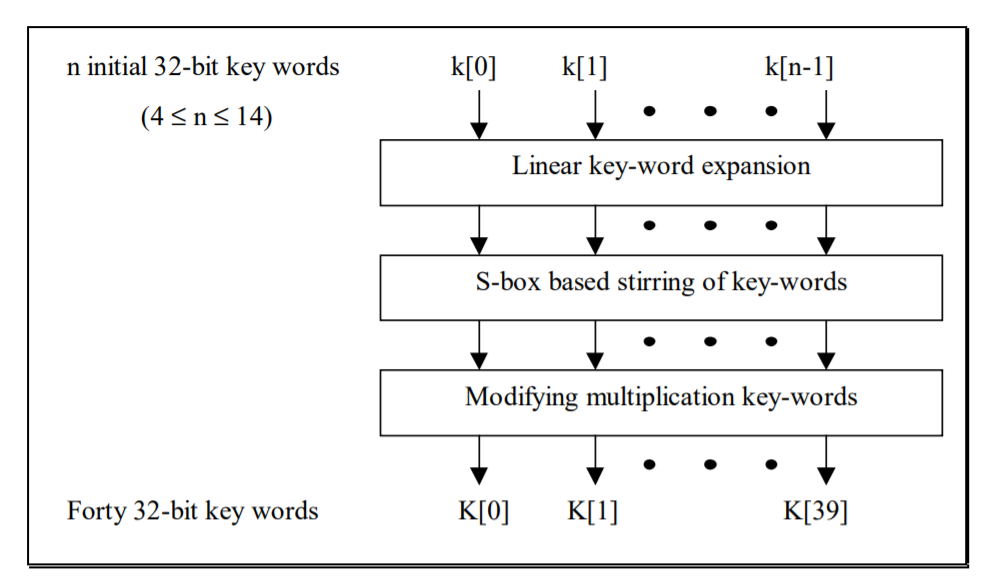
**MARS簡介:**

MARS是一種區塊加密法，這是IBM提交的高級加密標準程序(AES)。MARS在1999年8月進入AES決賽，並且為最後一輪的第五位入圍演算法。MARS的設計團隊包含Don Coppersmith，他曾經參加過數據加密標準(DES)的創立。他們專門設計了透過layered和compartmentalized approach來防止未來密碼學發展可能造成的威脅。

MARS的區塊大小為128 bit，而MARS的KEY的大小為可變的，介於4到14個word之間 (一個word為32 bit)。其使用的算法為type-3 Feistel network，且與大多數區塊加密不同的地方是MARS擁有異構結構(heterogeneous structure)，透過幾輪的加密核心與unkeyed混合來達成key whiting，是一種旨在提高迭代區塊加密安全性的技術，包括在第一輪之前和最後一輪加密完之後將data與key利用加法和減法結合在一起的步驟。MARS幾乎將密碼學家所知的每種設計技術結合在一起，他使用了加法、減法、S-BOX和fixed/data dependent rotations與乘法。前後混合包裹的cryptographic core可以抵抗所有已知的密碼分析攻擊，而mixing round則提供了快速的雪崩(avalanche)並可以防止未知的、新的攻擊。

MARS主要的程序為首8輪的無密鑰forward mixing與中間層的cryptographic core，和最後的backwards mixing。MARS將128bit的data切成4個32-bit的word當作輸入。輸入首先通過前向混合階段進行，此階段包含一個key addition和八輪unkeyed前向混合。然後進入加密核心（type-3 Feistel網路），進行八輪有key正向變換和有key向後變換，最後是另一個後向混合階段，其中包含八輪unkeyed後向混合和一個key的減法。

**Key Expansion:**

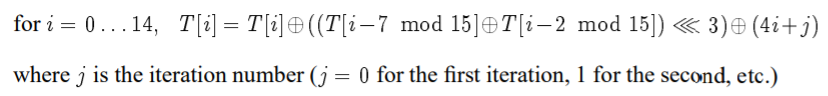
MARS讓使用者提供一把4到14 word的key，在經過了key expansion之後，將其變成了之後加密解密都會用到的40 word。以下為key expansion程序圖: 

首先第一步利用線性擴展，將原本的key擴展為40 word，這只是利用簡單的方法執行線性變換。第二步利用S-BOX執行7輪的type-1 Feistel network來破壞掉前面key的線性關係。最後在執行乘法key-word修改，檢查用於加密解密過程中用於乘法的key word。

最一開始我們將原始的key複製到15個word的臨時陣列T[ ]，



然後重複以下四次:

1.利用公式線性變換T陣列

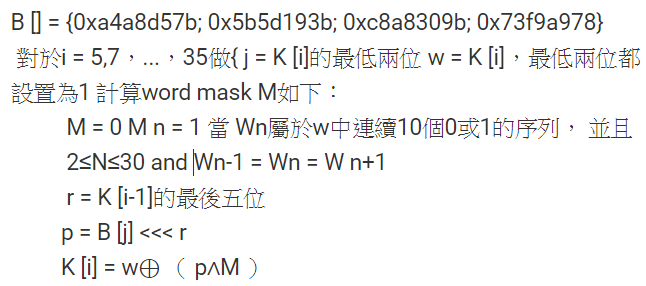
2.接下來使用4輪的type-1 Feistel network來打亂T陣列



3.從T陣列裡拿10個word重新排列到接下來擴展key陣列[ ]的10個word中

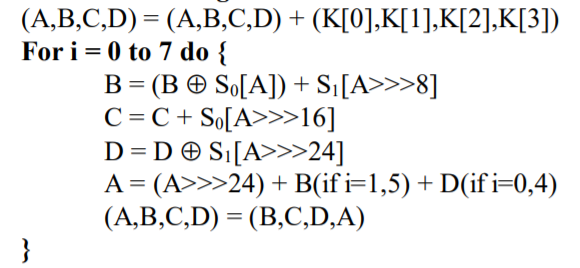


最後進行modifying multiplication key-words

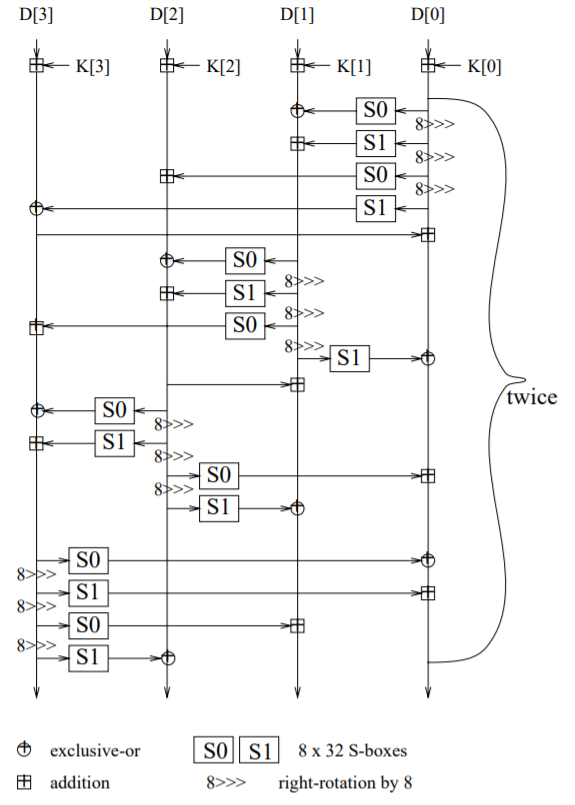


**Forward mixing:**

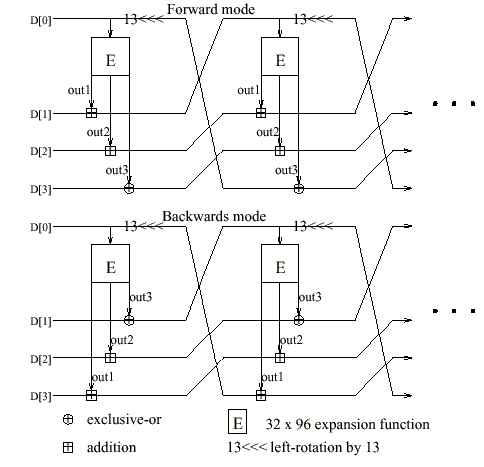
在forward mixing的步驟開始前，先將輸入的128bit明文分成四塊D[0]、D[1]、D[2]、D[3]，並且把每個data word加上key word K[0]、K[1]、K[2]、K[3]。將D[0]作為source data，另外三個作為target data。將source data D[0]再一次分成四塊b0,b1,b2,b3，其中b0是the lowest byte而b3是the highest byte。b0和b2作為S 0 (S-BOX)的索引值S 0 [b0]、S 0 [b2]，b1和b3作為S 1的索引值S 1 [b1]、S 1 [b3] (S-BOX由512個word組成，通常會將其分成兩個256 word的S 0、 S 1)。首先S 0 [b0]與第一個target word進行XOR，然後S 1 [b1]加到同一個word。再來將S 0 [b2]與第二個target word相加，S 1 [b3]和第三個target word進行XOR，最後將source word向右邊rotate 24個位置。在每輪結束後旋轉四個word，讓目前的第一個target word變成source word，目前的第二個target word變成第一個target word，以此類推。最後將D[0]、D[1]、D[2]、D[3]合併成一個128-bit的data，並且向左rotate 32個位置後作為下一輪的輸出。



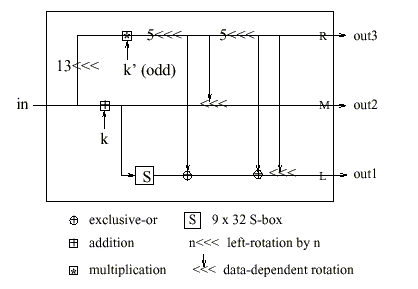
forward mixing pseudocode

**Cryptographic core(加密核心):**

加密核心是type-3 Feistel network ，而這裡的演算法使用的是有密鑰E-function來取代向前混和的無密鑰S-box，E-function的輸出也與其他的單詞相加或XOR，核心總共有16個round，8 round向前8 round 向後。



E-function



E-function是將兩組密鑰與input混合不同操作的組合，E-function是MARS演算法中設計最好的部分之一，能將不同的functions以最大化優點的方式組合。

E-function操作:

利用input – D，和兩密鑰K1、K2生成三個輸出數據

分別為L、M、R

密碼中使用操作應用於32位，0為最低有效位，31為最高有效位，均以2^32來表示，c <<< d、c >>> d分別表示32位字c向左或向右的d位置的循環。

1. 把D循環13位的結果賦給R
2. 把D和K1相加的結果賦給M
3. 取M的低9位做為S-box的索引找到替代數賦給L
4. 把R和K2相乘的結果循環左移5位後的值返回給R
5. 把L和R XOR的結果返回給L
6. 取R的低5位值，把M循環左移這個值後的結果返回給M
7. 把R循環位後的結果返回給R
8. 把L和R XOR的結果返回給L
9. 取R的低5位值，把L循環左移這個值後的結果返回給L
10. 把L、M、R作為E-function操作的第一、第二、第三輸出

**Backward mixing:**

backward mixing和forward mixing的結構非常類似，他還包括了8 round的無密鑰type-3 Feistel network，是一個減法步驟。

Backward mixing操作:

把128位數據分成4塊，D0、D1、D2、D3

輸入的4塊數據，其中D0作為source，剩下3個當作目標數據，把32位的D0分成8位的4塊，b0、b1、b2、b3

第一輪 : D0D1D2D3b0b1b2b3firstsecondthird

b0和b2作為數組下標從S1中尋找S-box的替換數 : S1[b0]、S1[b2]

b1和b3作為數組下標從S0中尋找S-box的替換數 : S0[b1]、S0[b3]

對first的操作

first XOR S1[b0]後的結果返回給first

對second的操作

second XOR S0[b3]後的結果返回給second

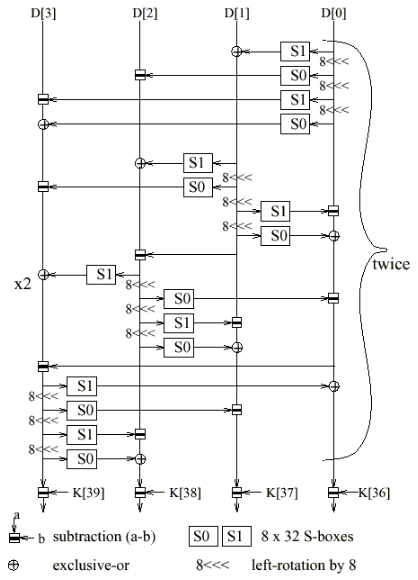
對third的操作

third減去S1[b2]後與S0[b1] XOR的結果返回給third

對source的操作

source循環左移24位後的結果返回給source

最後把D0、D1、D2、D3合併後的128位數據，循環左移32位後做為下一輪的輸入



**MARS 攻擊分析**

**1:** **Differential cryptanalysis**

由於cryptographic core本身16 round的費斯通結構已經夠複雜了，再加上外層8層的forwarding mixing和8層的Backward mixing 的阻力已經可以擋下現有技術的Differential attack了。

**2: Linear cryptanalysis**

若單論16輪cryptographic core就可以斷言Linear attack不存在，對於整體MARS也是如此。另一說法則是Linear attack已被證明使用S-box與XOR運算操作是最有效的。

**3: Exhaustive key search**

這種攻擊只需要幾個已知的明文 - 密文對。 攻擊者只需逐個嘗試所有密鑰，並檢查給定的明文是否加密給定的密文。 對於具有 *k* bits key和*n*個block確定密鑰所需的文本對的數量是近似的celling(k/n)

**4 The matching ciphertext attack**

這種攻擊出現在m bits block的DES 在加密2^(m/2)個block的時候，加密的block是可以被預測的。

**CODE : AES+DiffieHellman**

此段程式碼做簡單的介面，使用者先隨機選擇X做secreat number 接著算Y = a ^ X mod q 之後進行DiffieHellman交換彼此共同的鑰匙用這鑰匙做AES加密。由於AES鑰匙為256bit所以把運算出來的common key丟進sha256進行hash保證固定的長度，以及AES明文為128 bits block的block cipher當明文編碼長度不滿128 bits的倍數時進行padding 缺n個bytes 就n個 n的16進位(0x0n)，如果明文恰好為128bits的倍數時，同樣進行padding將 0x10 撲滿一個block方便進行加密解密。

# -\*- coding: utf-8 -\*-

"""

Created on Wed Dec 26 16:12:53 2018

@author: gordon

"""

import random

from gmpy2 import \*

from Crypto.Cipher import AES

import hashlib

a=mpz(11)

q=mpz(1040699159)

print('a :',a,', q :',q)

data = 'I love ncyucsie 噗嘻~~(≧Ꙫ≦)'

print('Plain :'+data+'\n')

# 使用者都共用a、q ， 選擇任意小於q的數當X當私鑰並求出準備給對方的Y

# 其中Y=(a^X)mod q

class user:

def \_\_init\_\_(self,name,a,q):

self.name = name

self.a = a

self.q = q

temp = sum([ord(i) for i in name])

self.\_\_X = mpz\_random(random\_state(temp),q)

self.Y = powmod(self.a,self.\_\_X,self.q)

self.\_\_key = {} #DiffieHellman為1對1的通訊故需要紀錄與多個使用者的common key

print(self.name+'\'s master key :',self.\_\_X)

#接收到對方的Y後開始計算雙方的common key，key = (Yb^X)mod q ; Yb為對方的Y

def commonKey(self, \_user, Yb):

key = powmod(Yb,self.\_\_X,self.q)

key = int(key).to\_bytes(len(key)//4+1,byteorder='big') #由數字進行編碼

key = hashlib.sha256(key) #為保證AES的KEY長度固定所以塞進SHA256

key = key.digest()

print('\nCommon key for '+ self.name+' and ' + \_user.name +': \n'+ key.hex())

self.\_\_key[\_user.name] = key #放進key storage

#AES 加密

def AES\_encrypt(self, \_user, data):

data = data.encode('utf-8') #編碼將字串轉bytes

padding = 16 - len(data) % 16 #AES為block ciper，須將明文鋪滿為128 bits 的倍數

if padding == 0 : #缺3個bytes 就補3個 b'/x03'

padding = 16

P = bytes([padding])

data += padding\*(P)

cryptor = AES.new(self.\_\_key[\_user.name],AES.MODE\_CBC , 16\*'\x00') #選擇block的模式和設定IV

cip = cryptor.encrypt(data)

print('\ncipher : \n'+ cip.hex())

return [cip , self.name] #回傳秘文和加密者的名

#AES 解密

def AES\_decrypt(self, cip):

data,name = cip[0],cip[1] #解析分出密文和加密者

cryptor = AES.new(self.\_\_key[name], AES.MODE\_CBC,16\*'\x00') #使用與加密者的common key

plain = cryptor.decrypt(data)

plain = plain[:(len(plain)-plain[-1])] #解密時將當初的padding給去掉

plain = plain.decode('utf-8')

print('\nplain from ' +name+' : \n'+ plain)

return plain

**#以下為簡易介面 :**

Alice = user('Alice', a , q) #建立Alice

Bob = user('Bob', a , q) #建立Bob

Alice.commonKey(Bob,Bob.Y) #收到彼此的Y之後算K = Y^X

Bob.commonKey(Alice,Alice.Y)

cip = Alice.AES\_encrypt(Bob,data) #Alice對Bob送出 "你白癡喔" 並進行加密

plain = Bob.AES\_decrypt(cip) #Bob從Alice收到密文進行解密



**上面程式使用的a=mpz(11)，q=mpz(1040699159)數字為以下程式求的，大約運算了4個小時**

from gmpy2 import \*

#隨機生成一個數rand

rand=mpz\_rrandomb(random\_state(),30)

print("random number ",end='')

print(rand)

#rand的下一個質數

q=next\_prime(rand)

print("prime number ",end='')

print(q)

def find\_a():

for i in range(1,q): #i的j次方找原根

for j in range(1,q):

if powmod(i,j,q)==1 and j<q-1:

break

elif powmod(i,j,q)==1 and j==q-1:

a=i

return a

print(find\_a())

**程式碼環境 : 使用anaconda CMD輸入conda install Gmpy2 建立**

Requirement:

Gmpy2

Hashlib

Crypto

MPVR

MPC

Reference:

<http://www.jiamisoft.com/blog/3863-marsjiamisuanfa.html>

<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.35.5887&rep=rep1&type=pdf>

<http://reto.orgfree.com/us/projectlinks/MARSReport.html#AlgKeyExp>

<https://pdfs.semanticscholar.org/0267/4d670aec83002c4710a5743465bc3082b8f2.pdf>

<http://reto.orgfree.com/us/projectlinks/keyexp.html>

<http://www.jiamisoft.com/blog/3863-marsjiamisuanfa.html>

<https://www.cryptrec.go.jp/exreport/cryptrec-ex-1085-2000.pdf>

<http://cryptography.wikia.com/wiki/MARS_(cryptography)>