Cryptography with Liberty Basic

102: Classical Cryptography: DES

By Onur Alver (*CryptoMan*)

INTRODUCTION

Table of Contents

Cryptography with Liberty Basic

102: Classical Cryptography: DES

By Onur Alver (CryptoMan)

INTRODUCTION

TRANSPOSITION

ENCRYPTION KEY

DES : DATA ENCRYPTION STANDARD

56 BITS DES KEY LENGTH

3DES: TRIPLE DES

MAC : MESSAGE AUTHENTICATION CODE

CONCLUSION

We will continue in our series of articles about Cryptography from where we left off at our first article. We have talked about the basics of cryptography and historical cryptography which still influences modern cryptography. You can see these simple and seemingly complex ideas are finding their way into various articles, code snippets and sample code as 'good' or 'sufficient' cryptography. Unfortunately, such simple methods may serve as good entertainment but not are really cryptography. So, let's continue our study of cryptography and understand what is good cryptography.

In our first article we have focused on substitution technique. Remember that ***substitution*** technique is a system where the letters of plaintext (unciphered data) are replaced by other letters or numbers or symbols. This is like substituting X for A, K for B, etc. XORing plaintext with another character effectively moves the alphabet by as many characters of the ASCII value of the character it is XOR'ed with.

Now, let's continue with the **transposition** technique.

TRANSPOSITION

Transposition is changing the position of letters in a predetermined way. For example, you can divide the text into two halves by taking every other character and assigning that character to left half or right half as shown below:

```
PLAINTEXT: ATTACKENEMYONAUGUST22AT 330GMT
    LEFT : ATCEEYNUUT2T30M
    RIGHT: TAKNMOAGS2A03GT
TRANSPOSED:ATCEEYNUUT2T30MTAKNMOAGS2A03GT
```

Of course, you can invent more complex division schemes than the simple example given above. What is usually done after transposition is to pass the resultant transposed text from a number of substitution functions which will translate characters into other characters following a predetermined schedule. The following example, coded in Liberty Basic language, uses the simple **`left - right transposition`** followed by a **`polyalphabetic`** substitution.

```
    REM TRANSPOSITION + SUBSTITUTION EXAMPLE
    REM FIRST TRANSPOSED THEN POLYALPHABETICALLY SUBSTITUTED
    REM AND EVERYTHING DONE IN REVERSE TO OBTAIN ORIGINAL MESSAGE.
    POLYDEPTH=5:DIM CRYPTKEY$(POLYDEPTH)
    PLAINKEY$= "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ:. "
    CRYPTKEY$(1)="LNOP4KQ6RSTFVEWZ:M.01X2H3G5A789BY CUDIJ"
    CRYPTKEY$(2)="4KQ6RSTFVEWL7IJ89BYCU DNOPZ:M.01X2H3G5A"
    CRYPTKEY$(3)="59B YCUDIJLA78NOP4KQ6RSTFVEWZ:M.01X2H3G"
    CRYPTKEY$(4)="VEWZ:M.01X2H3LNOP4KQ6RSTFG5A789BY CUDIJ"
    CRYPTKEY$(5)="4KQL7IJ89BYCU DNO6RSTFVEWPZ:M.01X2H3G5A"
    PLAINTEXT$="ATTACK AT 23:45 25 JUNE 2001"
    TRANSPOSEDTEXT$=""
    LEFTSIDE$=""
    RIGHTSIDE$=""
    FOR I=1 TO INT(LEN(PLAINTEXT$)) STEP 2
        LEFTSIDE$ =LEFTSIDE$+MID$(PLAINTEXT$,I,1)
        RIGHTSIDE$=RIGHTSIDE$+MID$(PLAINTEXT$,I+1,1)
    NEXT I
    TRANSPOSEDTEXT$=LEFTSIDE$+RIGHTSIDE$
```

```
    CIPHERTEXT$=""
    FOR I=1 TO LEN(TRANSPOSEDTEXT$)
        FOR J=1 TO LEN(PLAINKEY$)
            CurrentChar$=UPPER$(MID$(TRANSPOSEDTEXT$,I,1))
            CurrentPos$ =MID$(PLAINKEY$,J,1)
            IF CurrentChar$=CurrentPos$ THEN

          CI
PHERTEXT$ = CIPHERTEXT$ + MID$( CRYPTKEY$( (I MOD POLYDEPTH)+1), J, 1)
                EXIT FOR
            END IF
        NEXT J
    NEXT I
    DECIPHEREDTEXT$=""
    FOR I=1 TO LEN(CIPHERTEXT$)
        FOR J=1 TO LEN(CRYPTKEY$(1))
            CurrentChar$=UPPER$(MID$(CIPHERTEXT$,I,1))
            CurrentPos$ =MID$(CRYPTKEY$( (I MOD POLYDEPTH)+1),J,1)
            IF CurrentChar$=CurrentPos$ THEN
                DECIPHEREDTEXT$ = DECIPHEREDTEXT$ + MID$( PLAINKEY$, J
, 1)
                EXIT FOR
            END IF
        NEXT J
    NEXT I
    'AT THIS STAGE WHAT IS DECIPHERED IS TRANSPOSED TEXT
    'TO FIND CLEARTEXT WE MUST UN-TRANSPOSE IT
    SKIP=INT(LEN(PLAINTEXT$)/2)
    CLR$=""
    FOR I=1 TO SKIP
        CLR$=CLR$+MID$(DECIPHEREDTEXT$,I,1)+MID$(DECIPHEREDTEXT$,SKIP+
I,1)
    NEXT I
    PRINT "PLAIN TEXT :";PLAINTEXT$
    PRINT "TRANSPOSED TEXT:";TRANSPOSEDTEXT$
    PRINT "CIPHER TEXT :";CIPHERTEXT$
    PRINT "DECIPHERED :";CLR$
```

The execution of this code results in:

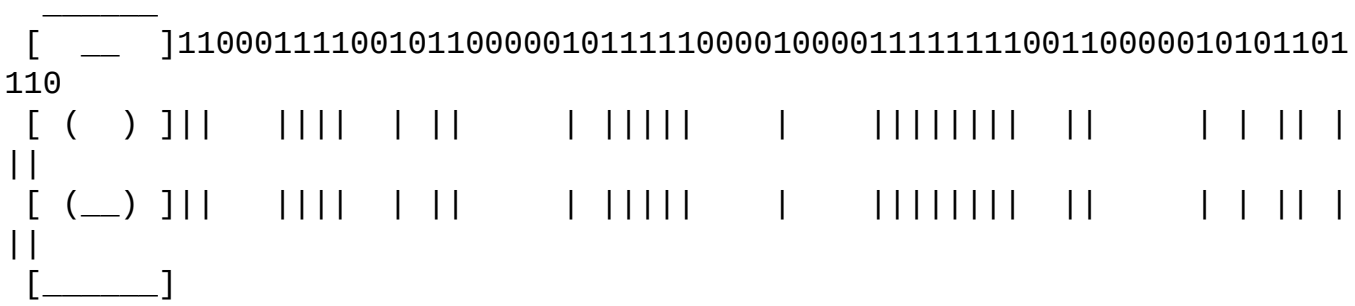
```
PLAIN TEXT.....:ATTACK AT 23:45 25 JUNE 2001
TRANSPOSED TEXT:ATC T2:52 UE20TAKA 34 5JN 01
CIPHER TEXT....:W:3A8QHMQJ0NW48W62APRGMSHA5E
DECIPHERED.....:ATTACK AT 23:45 25 JUNE 2001
```

So, you must be getting the idea by now. By creating more complex **transposition** and **substitution** formulas you can build very good crypto systems. The key idea is that your formula and design must be **`public`** so everyone can build the necessary software according to ***your*** algorithm. This allows others to test, analyze and critique your encryption to identify weaknesses and suggest improvements. What must be **`secret`** is the ***KEY*** to your encryption.

ENCRYPTION KEY

As a good physical key, the encryption key should be unique and should only open a specific lock or, in our case, decrypt the cipher text. This software equivalent of a physical key is a string of bits, say 64 bits.This is to say a chain of ones and zeroes like this:
1100011110010110000010111110000100001111111100110000010101101110

You can visualize the **`ones`** as the teeth and the **`zeroes`** as the rod of the key.



These **`ones`** and **`zeroes`** will instruct our algorithm how we want to ***transpose*** and ***substitute*** each bit of our data. This is done instead of making a simple schedule of taking every other character apart. To explain more fully, you take a block of data, usually 8 bytes, convert it into 64 bits and according to the complex schedule (as instructed by ones and zeroes in your key), mix all the bits successively with several iterations in such a way that at the end nobody should be able to form the original 64 bit order by looking at it or working at it without actually knowing the key.

The more bits you have in your key the more complex it becomes. The idea is to make it so complex and costly in computer time and effort that it must be unfeasible to find the solution in a reasonable timeframe, that reasonable time frame being a few hours, days, weeks or years. The number of bits in the key is known as the ***key length***
.
However, you must not confuse the key lengths of block ciphers like DES, 3DES, IDEA or AES with public key ciphers like RSA, Diffie - Helman, etc. A 128 bit **3DES** (**Triple DES**) key has equivalent strength to the 2048 bit **RSA** key. So, if someone claims that he has a super 512 bit cipher based on state

of the **RSA** which admittedly is much better than strong 128 bit ciphers, you should take this as an entertainment. This is like comparing apples and oranges. Key size should be considered in the context of algorithm. As a general rule, 112 bit to 256 bit block ciphers can be considered fairly strong and unbreakable. Number theoric public key algorithms like **RSA** must have 1024 or more bits. In fact for public key lengths, estimated secure key lengths are given by years. Currently it must be equal to or more than 1024 bits. By the year 2010, you should raise that standard to 2048 bits and beyond. We will discuss **public key cryptography** and **RSA** in the next article.

DES : DATA ENCRYPTION STANDARD

The most widely used encryption system in the world is **DES: Data Encryption Standard.** The DES system was adopted by the U.S. government as the standard for encryption in 1977. **DES** was designed by **IBM** engineers in 1973, in response to a call from the US government requesting an encryption system for all unclassified government data. The **DES** system is based on an earlier cipher system called LUCIFER and was the only system acceptable by the US National Security Agency (NSA). Originally designed as a 128-bit system, it was reduced to 56-bits following the advice of **NSA.**

There has been significant controversy over this reduction which **NSA** claims that was made due to hardware design limitations. Opponents say this significantly reduced the strength of the algorithm to a level so that any encryption can be easily broken by **NSA**. Nevertheless, **DES** has been widely adopted by many sectors, chiefly the financial sector which has decided to use it for the protection of **`PIN`** numbers on ATMs and EFT/POS terminals. **DES** is also used for encrypting sensitive data (ECB mode) on host to host communications, checking for message integrity with a special mode of DES called **cipher block chaining** (CBC mode).

DES is a **block cipher** operating on 64 bit data blocks (8 bytes) using 56 bit keys implementing successfully the following concepts:

***Diffusion*:** the statistical structure of cleartext data is mapped into a long range statistics of ciphertext. This means that every clear to cipher text transformation affects significantly following transformations making statistical frequency analysis efforts unfeasible

***Confusion*:** makes the relationship between ciphertext and the key very complex to make discovery of the key extremely difficult.

***Avalanche Effect*:** assures big changes in ciphertext with even 1 bit change in the key.

The **DES** algorithm has **16 rounds** with a specific **S-box** scheduling **substitutions, shifting and shuffling bits** with permutations with a resultant avalanche effect so huge that the statistical relation between the cleartext and ciphertext becomes nearly impossible to decipher.

DES effectively employs the techniques of substitutions and transpositions in a very systematic and

foolproof way.

DES design has classified secrets such as the nature of S-boxes. Why are those numbers used? These questions were never answered but only said that those are indeed very good numbers, and that any arbitrary change of these number could result in significant security losses. This is to say, these numbers are so designed that they optimize the bit shuffling in such a way as to make sure no trace of original data can be deduced from the output without knowing the key.

DES is a symmetric cipher, so the same key is used for both encryption and decryption. To decrypt, the algorithm is executed in reverse order to obtain the cleartext . Therefore, the ***encryptor and decryptor must share a common secret key***. The Algorithm is a public key, but is is secret. If the key became known than all of the encrypted data can be decrypted.
This brings the following significant problems:

o Key Generation
o Key Distribution
o Key Exchange
o Key Protection
o Key Verification

We will discuss these problems in the next article, as well as discussing how public key algorithms like **RSA** are solving these problems.

I believe this is a sufficient overview of **DES** for an introductory article. Those readers who want to go deeper can find many articles on Internet as well as several books in their bookstores. So, let’s go directly into an example **implementation of DES written purely in Liberty Basic**.

```
    REM
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''''''''''''''''''''''''''

'DES ENCRYPTION / DECRYPTION DEMO SOFTWARE WRITTEN PURELY IN LIBERTY B
ASIC
    'Copyright(c) 2006, Verisoft CryptoMan
    'www.verisoft.com onur@verisoft.com

'---------------------------------------------------------------------
-------------------------

'This software and source code is provided for educational private use
 and may not be used

'for commercial purposes without express written consent of VERISOFT.
```

```
'-------------------------------------------------------------------
-------------------------
    'LIABILITY CLAUSE:

'Verisoft will not accept any liabilities or claims due to use, non-
use, misuse of this software

'or damages to property or life under any circumstances, direct or ind
irect. Use it at your own

'risk. There are no claims to performance, correctness and fitness of
this software.

'Use of or exportation of cryptographic software to certain countries;
 and disclosure thereof
    'may violate local laws and regulations.

'-------------------------------------------------------------------
-------------------------

' LESSON 1: : THIS EXAMPLE ONLY ENCRYPTS AND DECRYPTS PLAINTEXT FILES
    ' DES : IT IS NOT FAST, IT IS NOT PERFECT, IT IS NOT COMPLETE !

' SYMETRIC CRYPTO : IT ONLY SHOWS, HOW REAL ANSI/ISO DATA ENCRYPTION S
TANDARD
    ' WORKS: SINGLE DES, 56-BIT KEYS, ECB MODE.

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''''''''''''''''''''''''''

' WARNING: IF YOU ENCRYPT AND IMPORTANT FILE, AND THEN ERASE IT, AND T
HEN FORGET

' THE KEY AND YOU HAVE ONLY ENCRYPTED FILE; YOU WILL BE IN TROUBLE. PL
EASE, DON'T

' CALL US OR ANYONE. IF YOU HAVE GOOD FRIENDS AT NSA, THEY MAY DO SOME
THING BUT

' NOT US. SORRY. DON'T ENCRYPT YOUR IMPORTANT FILES AND FORGET YOUR KE
Y !!!!!!!!

'_____
_____
    GLOBAL INITIALTR$,FINALTR$,SWAP$,KEYTR1$,KEYTR2$,ETR$,PTR$,CR$,LF$
```

```
    CALL Initialize

    NoMainWin
    WindowWidth = 800
    WindowHeight = 500
    UpperLeftX=int((DisplayWidth-WindowWidth)/2)
    UpperLeftY=int((DisplayHeight-WindowHeight)/2)

    statictext #main.statictext6, "Source File", 15, 2, 108, 20
    statictext #main.statictext7, "Destination File", 15, 57, 192, 20
    statictext #main.statictext8, "Encryption Key", 15, 112, 200, 20
    textbox #main.source, 15, 24, 280, 25
    button #main.browseSource, "Browse"
, [browseSource], UL, 230, 2, 60, 20
    textbox #main.destination, 15, 80, 280, 25
    button #main.browseDestination
, "Browse", [browseDestination], UL, 230, 57, 60, 20
    textbox #main.key, 15, 132, 110, 25
    radiobutton #main.selectDecry
pt, "Decrypt Source to Destination"
, [selectDecrypt], [reset], 15, 192, 280, 25
    radiobutton #main.selectEncry
pt, "Encrypt Source to Destination"
, [selectEncrypt], [reset], 15, 172, 280, 25
    button #main.go, "Go", [go], UL, 10, 222, 85, 25
    texteditor #main.tesr
c, 300,10, 490, 200 'The handle for our texteditor is #window.te
    texteditor #main.tede
s, 300,230,490, 200 'The handle for our texteditor is #window.te
    graphicbox #main.gb, 800, 1, 10, 10

    open "DES Crypto Demo V1.00"+space$(80)+
"Verisoft(c)2006, www.verisoft.com " for window as #main
    print #main, "font Courier 10"
    print #main, "trapclose [quit]"

    MODE = 1
    #main.selectEncrypt, "set"
    wait

[browseSource] 'Search for source file
    filedialog "Select source file:","*.*",INFILE$
    if INFILE$ <> "" then #main.source, INFILE$
    wait

[browseDestination] 'Search for destination file
```

```
    filedialog "Select destination file:","*.*",OUTFILE$
    if OUTFILE$ <> "" then #main.destination, OUTFILE$
    wait

[selectDecrypt] 'Set mode to decryption
    MODE = 2
    wait

[selectEncrypt] 'Set mode to encryption
    MODE = 1
    wait

[reset] 'Perform action for the radiobutton named 'selectDecrypt'
    wait

[go] 'Do it!
    #main.key, "!contents? KEY$"
    #main.source, "!contents? INFILE$"
    #main.destination, "!contents? OUTFILE$"

    cursor hourglass
    RESULT = ENCRYPTION(MODE,INFILE$,OUTFILE$,KEY$)
    cursor normal
    if RESULT = 1 then notice "DES Crypto Result"+CR$+
"Action completed successfully"
    if RESULT = 2 then notice
"Encrypt/decrypt mode not properly selected"
    if RESULT = 3 then notice "Source file does not exist"
    if RESULT = 4 then notice "Destination file already exists"
    if RESULT = 5 then notice "No de/encryption key specified"
    wait

[quit] 'End the program
    close #main
    end

FUNCTION ENCRYPTION(MODE,INFILE$,OUTFILE$,KEY$)

    IF MODE=1 THEN

        OPEN INFILE$ FOR INPUT AS #1
        IF OUTFILE$="" THEN OUTFILE$="ENCTEMP";TIME$("ms");".TXT"
        OPEN OUTFILE$ FOR OUTPUT AS #2
        #main.tesrc,""
        #main.tesrc,DATE$();" ";TIME$()
        #main.tesrc,"-----------------------------------------------"
```

```
        #main.tedes,""
        #main.tedes,DATE$();" ";TIME$()
        #main.tedes,"------------------------------------------------"


        WHILE EOF(#1)=0
            LINE INPUT #1,TEXT$
            TEXT$=TEXT$+CR$
            #main.tesrc,TEXT$;
            DO

''''''''''''''''PAD WITH NUL WHEN BLOCK LESS THAN 8 BYTES''''''''''''
'''
            DX$=LEFT$(TEXT$,8)
            PDX$=chr$(0)+chr$(0)+chr$(0)+chr$(0)+chr$(0)+chr$(0)+chr$(
0)+chr$(0)
            PDX$=DX$+LEFT$(PDX$,8-LEN(DX$))
            ENC$=DESencrypt$(PDX$,KEY$)
            HX$=SpecHex$(ENC$)
            #main.tedes,HX$
            PRINT #2,HX$
            TEXT$=MID$(TEXT$,9)
            SCAN
            LOOP WHILE LEN(TEXT$)>0
        WEND

        CLOSE #1
        CLOSE #2
        #main.tesrc,"------------------------------------------------"
        #main.tedes,"------------------------------------------------"

        ENCRYPTION = 1

    ELSE

        OPEN INFILE$ FOR INPUT AS #1
        IF OUTFILE$="" THEN OUTFILE$="DECTEMP";TIME$("ms");".TXT"
        OPEN OUTFILE$ FOR OUTPUT AS #2
        #main.tesrc,""
        #main.tesrc,DATE$();" ";TIME$()
        #main.tesrc,"------------------------------------------------"
        #main.tedes,""
        #main.tedes,DATE$();" ";TIME$()
        #main.tedes,"------------------------------------------------"

        WHILE EOF(#1)=0
```

```
            LINE INPUT #1,TEXT$
            #main.tesrc,TEXT$
            DX$=PackHex$(LEFT$(TEXT$,16))
            DEC$=DESdecrypt$(DX$,KEY$)


''''''''''''''''''REMOVE PADDING ADDED DURING ENCRYPTION''''''''''''''''
''
            TDEC$=""
            FOR I=1 TO 8
                CH$=MID$(DEC$,I,1)
                IF CH$>CHR$(0) THEN TDEC$=TDEC$+CH$
            NEXT
            #main.tedes,TDEC$;
            PRINT #2,TDEC$;

        WEND

        CLOSE #1
        CLOSE #2
        #main.tesrc,"----------------------------------------------"
        #main.tedes,"----------------------------------------------"

        ENCRYPTION = 1

    END IF
END FUNCTION

END

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'''''''''
' COPY GLOBALS AND ALL BELOW THIS LINE UNDER YOUR SOFTWARE AND YOU WIL
L
' DESencrypt AND DESdecrypt FUNCTIONS IN YOUR SOFTWARE.  DES WORKS WIT
H
' 8 BYTE KEYS ON 8 BYTE DATA. IT ENCRYPTS TO 8 BYTE BLOCK AND WILL DEC
RYPT
' FROM 8 BYTE BLOCK TO PLAIN TEXT. IF YOU HAVE LONGER DATA THEN YOU MU
ST
' DIVIDE IT INTO 8 BYTE BLOCKS. SHORTER THAN 8 BYTE BLOCKS MUST BE PAD
DED
' AND UNPADDED BY YOUR SOFTWARE LOGIC. ABOVE, DEMO SHELL SHOWS THIS NI
CELY.
' HOWEVER, DOWN BELOW THE GUI SHELL, IT CALLS THE FUNCTIONS BELOW. DES
 NEEDS
```

```
' ALL THOSE COMPLEX FUNCTIONS AND NUMBERS.  YOU SHOULD NOT TOUCH OR WO
RRY
' ABOUT THEM. JUST USE DESencrypt( DATA$,KEY$ ) OR DESdecrypt( ENCDATA
$,KEY$ ).
' YOU CAN DECRYPT WITH THE EXACT SAME KEY YOU USED FOR ENCRYPTION.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'''''''''

FUNCTION PackHex$( y$ )
    z$=""
    for j=1 to len(y$) step 2
        n=HEXDEC(mid$(y$,j,2))
        z$=z$+chr$(n)
    next j
    PackHex$=z$
END FUNCTION

FUNCTION SpecHex$( y$ )
    z$=""
    FOR j=1 TO LEN(y$)
        n=ASC(MID$(y$,j,1))
        IF n<16 THEN z$=z$+"0"
        z$=z$+DECHEX$(n)
    NEXT j
    SpecHex$=z$
END FUNCTION

FUNCTION RANDOMKEY$()
    X$=""
    FOR I=1 TO 8
        X$=X$+CHR$(RND(1)*255)
    NEXT I
    RANDOMKEY$=X$
END FUNCTION

SUB Initialize
    CR$=CHR$(13):LF$=CHR$(10)

    INITIALTR$=""
    DATA 58,50,42,34,26,18,10,2,60,52,44,36,28,20,12,4,62,54,46,38,_
        30,22,14,6, 64,56,48,40,32,24,16,8,57,49,41,33,25,17,9,1,59,51
,43,_
        35,27,19,11,3,61,53,45,37,29,21,13,5,63,55,47,39,31,23,15,7
    FOR I=1 TO 64:READ X:INITIALTR$=INITIALTR$+CHR$(X): NEXT I

    FINALTR$=""
```

```
    DATA 40,8,48,16,56,24,64,32,39,7,47,15,55,23,63,31,38,6,46,14,54,_
        22,62,30,37,5,45,13,53,21,61,29,36,4,44,12,52,20,60,28,35,3,43
,11,_
        51,19,59,27,34,2,42,10,50,18,58,26,33,1,41,9,49,17,57,25
    FOR I=1 TO 64:READ X:FINALTR$=FINALTR$+CHR$(X): NEXT I

    SWAP$=""
    DATA 33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,_
        53,54,55,56,57,58,59,60,61,62,63,64,01,02,03,04,05,06,07,08,09
,10,_
        11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31
,32
    FOR I=1 TO 64:READ X:SWAP$=SWAP$+CHR$(X): NEXT I


    KEYTR1$=""
    DATA 57,49,41,33,25,17,9,1,58,50,42,34,26,18,10,2,59,51,43,35,_
        27,19,11,3,60,52,44,36,63,55,47,39,31,23,15,7,62,54,46,38,30,2
2,_
        14,6,61,53,45,37,29,21,13,5,28,20,12,4,0,0,0,0,0,0,0,0
    FOR I=1 TO 64:READ X:KEYTR1$=KEYTR1$+CHR$(X): NEXT I

    KEYTR2$=""
    DATA 14,17,11,24,1,5,3,28,15,6,21,10,23,19,12,4,26,8,16,7,27,20,_
        13,2,41,52,31,37,47,55,30,40,51,45,33,48,44,49,39,56,34,53,46,
42,_
        50,36,29,32,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
    FOR I=1 TO 64:READ X:KEYTR2$=KEYTR2$+CHR$(X): NEXT I

    ETR$=""
    DATA 32,1,2,3,4,5,4,5,6,7,8,9,8,9,10,11,12,13,12,13,14,15,_
        16,17,16,17,18,19,20,21,20,21,22,23,24,25,24,25,26,27,28,29,_
        28,29,30,31,32,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
    FOR I=1 TO 64:READ X:ETR$=ETR$+CHR$(X): NEXT I


    PTR$=""
    DATA 16,7,20,21,29,12,28,17,1,15,23,26,5,18,31,10,2,8,24,14,_
        32,27,3,9,19,13,30,6,22,11,4,25,0,0,0,0,0,0,0,0,0,0,0,0,0,0,_
        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
    FOR I=1 TO 64:READ X:PTR$=PTR$+CHR$(X): NEXT I


    DIM ROTS(16)
    DATA 1,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1
    FOR I=1 TO 16:READ X:ROTS(I)=X: NEXT I
```

```
DIM S(8,64)
DATA 14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7,_
    0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8,_
    4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0,_
    15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13

DATA 15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10,_
    3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5,_
    0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15,_
    13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9

DATA 10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8,_
    13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1,_
    13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7,_
    1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12

DATA 7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15,_
    13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9,_
    10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4,_
    3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14

DATA 2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9,_
    14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6,_
    4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14,_
    11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3

DATA 12, 1, 10, 15,  9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11,_
    10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8,_
    9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6,_
    4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0,  8, 13

DATA 4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1,_
    13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6,_
    1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2,_
    6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12

DATA 13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7,_
    1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2,_
    7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8,_
    2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11

FOR I=1 TO 8
    FOR J=1 TO 64
        READ X
        S(I,J)=X
```

```
        NEXT J
    NEXT I

END SUB

SUB TRANSPOSE BYREF DAT$, T$, N
    DIM iDAT(64),iT(64),iX(64)

    FOR I=1 TO 64
        IF MID$(DAT$,I,1)=CHR$(1) THEN iDAT(I)=1 ELSE iDAT(I)=0
        iX(I)=iDAT(I)
        iT(I)=ASC(MID$(T$,I,1))
    NEXT

    FOR I=1 TO N
        iDAT(I)=iX( iT(I) )
    NEXT
    ZAT$=""
    FOR I=1 TO 64: ZAT$=ZAT$+CHR$(iDAT(I)): NEXT I
    DAT$=LEFT$(ZAT$ ,64)
END SUB

SUB ROTATE BYREF KEY$
    X$=LEFT$(KEY$,56)
    X$=MID$(X$,2,55)
    X$=LEFT$(X$,27)+LEFT$(KEY$,1)+MID$(X$,29)
    X$=LEFT$(X$,55)+MID$(KEY$,29,1 )
    KEY$=LEFT$(X$,56)
END SUB

SUB UNROTATE BYREF KEY$
    X$=LEFT$(KEY$,56)
    X$=MID$(KEY$,28,1)+LEFT$(X$,55)
    X$=LEFT$(X$,28)+MID$(KEY$,56,1)+MID$(X$,30)
    KEY$=LEFT$(X$,56)
END SUB

SUB F I, BYREF KEY$, BYREF A$, BYREF X$
    DIM Z(64),Y(64)
    E$=LEFT$(A$,56)
    CALL TRANSPOSE E$,ETR$, 48
    FOR J=1 TO ROTS(I)
        CALL ROTATE KEY$
    NEXT
    IKEY$=LEFT$(KEY$,56)
    CALL TRANSPOSE IKEY$,KEYTR2$, 48
```

```
    FOR J=1 TO 48
        IF
 ASC(MID$(E$,J,1))+ASC(MID$(IKEY$,J,1))=1 THEN Y(J)=1 ELSE Y(J)=0
    NEXT
    FOR K=1 TO 64: Z(K)=ASC(MID$(X$,K,1)): NEXT
    FOR K=1 TO 8
        R=32*Y(6*K-5)+16*Y(6*K)+8*Y(6*K-4)+4*Y(6*K-3)+2*Y(6*K-2)+Y(6*K
-1)+1
        IF ODD(S(K,R) / 8) THEN Z(4*K-3)=1 ELSE Z(4*K-3)=0
        IF ODD(S(K,R) / 4) THEN Z(4*K-2)=1 ELSE Z(4*K-2)=0
        IF ODD(S(K,R) / 2) THEN Z(4*K-1)=1 ELSE Z(4*K-1)=0
        IF ODD(S(K,R)) THEN Z(4*K) =1 ELSE Z(4*K) =0
    NEXT
    X$=""
    FOR K=1 TO 64: X$=X$+CHR$(Z(K)): NEXT
    CALL TRANSPOSE X$,PTR$,32
END SUB

SUB F2 I, BYREF KEY$, BYREF A$, BYREF X$
    DIM Z(64),Y(64)
    E$=LEFT$(A$,64)
    CALL TRANSPOSE E$, ETR$, 48
    IKEY$=LEFT$(KEY$,64)
    CALL TRANSPOSE IKEY$, KEYTR2$ , 48
    FOR J=1 TO 48
        IF
 ASC(MID$(E$,J,1))+ASC(MID$(IKEY$,J,1))=1 THEN Y(J)=1 ELSE Y(J)=0
    NEXT J
    FOR J=1 TO ROTS(17-I)
        CALL UNROTATE KEY$
    NEXT J
    FOR K=1 TO 64: Z(K)=ASC(MID$(X$,K,1)): NEXT
    FOR K=1 TO 8
        R=32*Y(6*K-5)+16*Y(6*K)+8*Y(6*K-4)+4*Y(6*K-3)+2*Y(6*K-2)+Y(6*K
-1)+1
            IF ODD(S(K,R) / 8) THEN Z(4*K-3)=1 ELSE Z(4*K-3)=0
            IF ODD(S(K,R) / 4) THEN Z(4*K-2)=1 ELSE Z(4*K-2)=0
            IF ODD(S(K,R) / 2) THEN Z(4*K-1)=1 ELSE Z(4*K-1)=0
            IF ODD (S(K,R)) THEN Z(4*K)=1 ELSE Z(4*K)=0
    NEXT
    X$=""
    FOR K=1 TO 64: X$=X$+CHR$(Z(K)): NEXT
    CALL TRANSPOSE X$,PTR$,32
END SUB

FUNCTION ODD( N )
```

```
    IF INT(N) MOD 2 = 0 THEN ODD=0 ELSE ODD=1
END FUNCTION

FUNCTION DESencrypt$( PTEXT$, KY$ )
    PLAINTEXT$=BINARY$(PTEXT$)
    KEY$=BINARY$(KY$)
    A$=LEFT$( PLAINTEXT$,64 )
    CALL TRANSPOSE A$, INITIALTR$ ,64
    CALL TRANSPOSE KEY$, KEYTR1$ ,56
    FOR I=1 TO 16
        B$=LEFT$( A$,64 )
        A$=MID$(B$,33,32)
        CALL F I,KEY$,A$,X$
        FOR J=1 TO 32
            IF
 ASC(MID$(B$,J,1))+ASC(MID$(X$,
J,1))=1 THEN A$=A$+CHR$(1) ELSE A$=A$+CHR$(0)
        NEXT
    NEXT
    CALL TRANSPOSE A$,SWAP$,64
    CALL TRANSPOSE A$,FINALTR$,64
    DESencrypt$=ASCII$(A$)
END FUNCTION

FUNCTION DESdecrypt$( CTEXT$, KY$ )
    CRYPTEXT$=BINARY$(CTEXT$)
    KEY$=BINARY$(KY$)
    A$=LEFT$( CRYPTEXT$,64 )
    CALL TRANSPOSE A$, INITIALTR$ ,64
    CALL TRANSPOSE KEY$, KEYTR1$ ,56
    FOR I=1 TO 16
        B$=LEFT$( A$,64 )
        A$=MID$(B$,33,32)
        CALL F2 I,KEY$,A$,X$
        FOR J=1 TO 32
            IF
 ASC(MID$(B$,J,1))+ASC(MID$(X$,
J,1))=1 THEN A$=A$+CHR$(1) ELSE A$=A$+CHR$(0)
        NEXT
    NEXT
    CALL TRANSPOSE A$,SWAP$,64
    CALL TRANSPOSE A$,FINALTR$,64
    DESdecrypt$=ASCII$(A$)
END FUNCTION

FUNCTION BINARY$( BMP$ )
```

```
    BITMAP$=""
    FOR j=1 TO 8
        L1=ASC(MID$(BMP$,j,1))
        FOR i=7 TO 0 STEP -1
            IF ( 2^i AND L1 ) THEN
 BITMAP$=BITMAP$+CHR$(1) ELSE BITMAP$=BITMAP$+CHR$(0)
        NEXT i
    NEXT j
    BINARY$=BITMAP$
END FUNCTION

FUNCTION ASCII$( BMP$ )
    BITMAP$=""
    C=0
    N=7
    FOR j=1 TO 64
        C=C+ASC(MID$(BMP$,j,1))*2^N
        N=N-1
        IF ( j MOD 8 )=0 THEN
            BITMAP$=BITMAP$+CHR$(C)
            N=7
            C=0
        END IF
    NEXT j
    ASCII$=BITMAP$
END FUNCTION
```

56 BITS DES KEY LENGTH

You may wonder why 56 bits used instead of 64 bits. This is because every byte has 7 data bits and 1 parity bit. Thus, 8 x 7 = 56 bits is the effective key size. Highest parity bit is generally ignored but if it is checked, it is then expected that DES keys must have odd parity.

3DES: TRIPLE DES

The **`56 bits` DES** algorithm, also known as **single DES**, has been cracked for some years now. This can be done with specially designed parallel computers with high speed hardware ***DES crypto processors***.

Another method used is a network effort made with tens of thousands of Pentium PCs connected over the Internet. A special screensaver uses the idle time of these PCs to search a key space assigned to each PC. This way, each PC is given a different key space from a master computer. After several days or months, eventually one of the PCs finds the key and reports this to the master computer via Internet. This attack is

possible if a known **plaintext attack** is possible. Otherwise this sort of brute force attack is difficult. However, most attractive target financial encrypted data such as enciphered **PIN** numbers have a known plaintext pattern and thus such an attack is possible. Due to all this, major credit card systems have since changed to 3DES:TRIPLE DES.

TRIPLE DES uses two **56 bit DES keys** in what is known as **EDE** mode, or ***Encrypt Decrypt Encrypt*** mode. So, effectively it is **112 bit** encryption. Even so, **EDE** remains generally termed as **128 bit** encryption. So, this is the so called ***strong 128 bit encryption***. **Single DES** run `three times` with `two keys`. Of course, you can also use `three keys` in **EDE** mode, but the classic **3DES** is **112 bit 3DES** which ***encrypts*** with **EDE** and ***decrypts*** with **DED** using two **56 bit keys**. **3DES is very strong** and we can say it is ***unbreakable*** until **quantum computers** become a reality. Otherwise, even with the world's fastest computer, the **112 bits key space** can not feasibly be searched using a brute force trial and error method to find all the right bits. There are ***2 to the power 112 different possible keys***. This is a big number.

How big? Let's say if every bit can be represented by one electron, which is not possible but still we say if this is possible, and we were to store every possibility, then you will have

- 5 192 296 858 534 827 628 530 496 329 220 096 electrons x 112 bits
- 1 electron is 1/1837 weight of an AMU, atomic mass unit
- 1 gram is equivalent to 1,675,000,000,000,000,000,000,000 AMU

According to this, the weight of our data only on our hard disk will be 188,996 kg (416,292 lb) or almost 200 metrictons. Which is to say you will have pretty heavy metal on your desktop. Not a good idea to put all this data in your laptop. In fact according to current hard disk technology your harddisk will instantly turn into a **black hole** and suck up everything around if you try to store all **3DES** keys possible.

So, don't mess around with 3DES without adult supervision.

MAC : MESSAGE AUTHENTICATION CODE

One of the useful applications of **DES** and **3DES** is **MAC: Message Authentication Code**. **MAC** is used to ensure ***INTEGRITY*** while **DES/3DES** ensures ***CONFIDENTIALITY.*** This means, using **MAC**, you can create a **digital signature** to assure that no part of the message, not even one bit has been changed in transit.

For example, if you are making a wire transfer from `account number 1029121` to `2771099` for $125.00, you will not want someone messing around with data to divert the funds to `account number 2499911` instead. Neither, do you want the amount of transfer changed from $125.00 to $12,500.00. Hence, the two banks exchange the ***MAC'ing keys*** used for transfers. Each transferred message is then divided into 8 byte blocks. These 8 byte blocks are then successively encrypted using either **DES** or **3DES**,

XORing the next block to the encrypted value of previous block until the end of the message is reached.

This procedure results in an 8 byte hexadecimal value. Usually you take the left half 4 bytes and call this the ***MAC of the message***. You append this **MAC** to the end of each message. Finally the recipient gets the message and **MAC**. The receiver then recomputes the **MAC** again using the agreed upon key and compares the transmitted **MAC**. If the two **MAC**'s don't match, then the data has been violated in transit.

Of course this same technique can be applied to files stored on the disk. You can put a **MAC** at the end of each line of a critical contract and one **MAC** for the entire document. If the document **MAC** doesn't match the expected **MAC**, then you can check every line with the **Line MAC**s to see which line of the document was altered. Likewise, you can put **MAC**s to every accounting record to detect any internal fraud by an employee messing with company accounts.

MAC is the backbone of many financial transactions like credit card systems. For example the latest **EMV** chip cards from **Visa** and **Mastercard** or **Contactless** cards like **Mastercard PayPass** or **Visa Wave** use variants of **MAC** like **CVV**, **ARQC**, **ARPC**, etc to provide end to end message integrity of credit and debit card transactions. From the date, time, amount, currency code, card number and random elements **ARQC (Authorization Request Cryptogram)** is generated by the chip card in a **POS** or **ATM**. During the transaction, the **MAC** is checked by the issuing bank online to determine if this is a valid transaction. If found to be 'good', the **ARPC (Authorization Response Cryptogram)** is sent back to the terminal which submits this code to the card. The card then decides whether this response is bona fide and accepts or rejects the transaction accordingly.

All of these global events happen in just a few seconds using a smartcard with a chip smaller than 1 millimeter square. More recently, these transactions are being made with contactless cards wirelessly over the air. Encryption is not science fiction but real life events happening behind the scenes things of such everday events as checking out from the supermarket with your credit card or using your digital mobile phone.

CONCLUSION

So you now have a **DES** algorithm written in Liberty Basic and a simple GUI application for experimentation. It is not a fast implementation. Such speed would require a DLL written in C. Perhaps this brings to mind a MATRIX style, Hollywood scene like those **`HEX numbers`** floating on the window; a cliche of our times when our hero or heroine sits in front of a display to watch these speeding numbers and, bingo, gets into the deepest secrets of the next ICBM launch.

Enjoy. It will get more interesting with public key cryptography in the next article.

CryptoMan

onur@verisoft.com