
# Delhi Technological University

*Department of Applied Physics*

---

# Cryptography using Assembly Language

*Mid-term Evaluation Project Report*

---

## (EP-206) Microprocessor & Interfacing

A Project by:-
Dhruv Tyagi (2K19/EP/032)
Ayush Kumar (2K19/EP/030)

# Acknowledgement

We would like to express our sincere gratitude towards our respected professor, Dr. Rishu Chaujar for encouraging us and providing us with the opportunity to expand our subject knowledge by working on this project.

Her continued support & valuable criticism along with her guidance have been huge contributions towards the successful completion of this project.

# Table Of Contents

# An Introduction to Cryptography

From the very dawn of mankind's existence forming networks among communities & exchanging information has been a key component of society. In today's world exchanging information securely to protect the privacy of users has become a crucial concern. This concern gave rise to the art of coding the messages in such a way that only the intended people could have access to the information. Unauthorized people could not extract any information, even if the scrambled messages fell into their hands.

**Cryptography** is the practice or study of secure communication by transforming information into a form that only the intended recipients are able to understand. Essentially, cryptography is the science of encrypting & decrypting data to make it unintelligible to malicious third-parties—known as adversaries. Applications of cryptography include digital currencies, computer passwords, military communications, chip-based payment cards and electronic commerce.


Perhaps the most common example of Cryptography we see today is in banks such as SBI, which use 256-bit encryption to protect their data.

## Cryptography techniques

Cryptography is closely related to the disciplines of cryptology & cryptanalysis. It includes techniques such as microdots, merging words with images, and other ways to hide information in storage or transit. However, in today's

computer-centric world, cryptography is most often associated with scrambling plaintext (ordinary text, sometimes referred to as clear text) into cipher text (a process called encryption), then back again (known as decryption). Individuals who practice this field are known as cryptographers.

The 2 main components of any cryptographic method are encryption of the original text into cipher text & then decryption of the cipher text to recover the original text.

## Encryption:

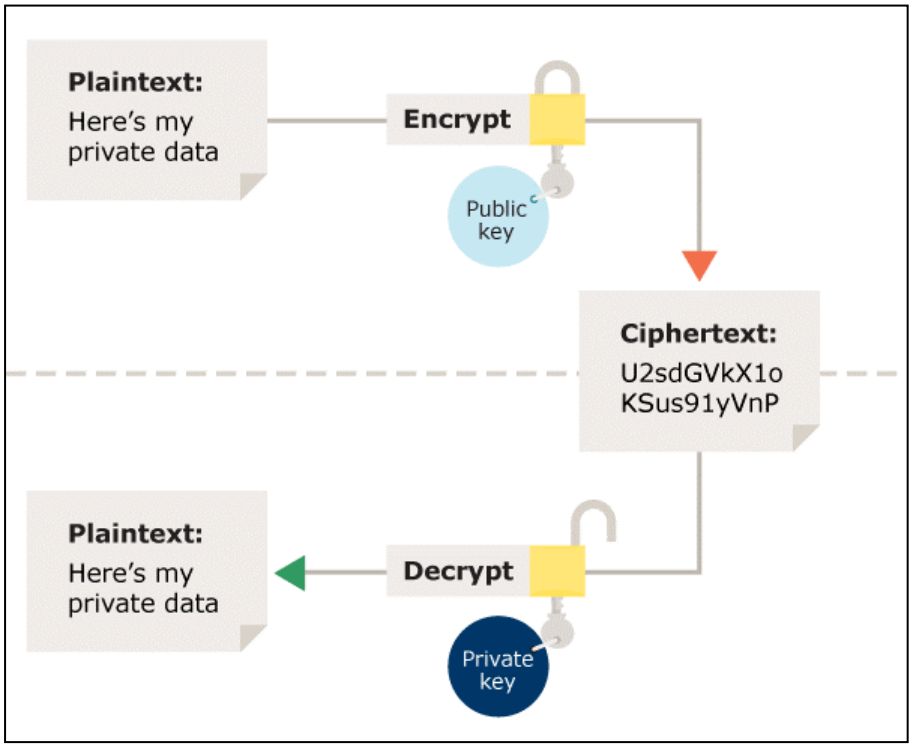
In cryptography, **encryption** is the process of encoding information. This process converts the original representation of the information, known as plaintext, into an alternative form known as cipher text. Usually, encryption alone isn't enough to protect data.


Although encrypted data appears random, encryption proceeds in a logical, predictable way, allowing a party that receives the encrypted data and possesses the right key to decrypt the data, turning it back into plaintext. Truly secure encryption will use keys complex enough that a third party is highly unlikely to decrypt or break the ciphertext

## Decryption:

Decryption is the process of decoding encrypted information. This process is used to recover the original plaintext back from the encrypted cipher text. It decodes the encrypted information given that the user trying to access the information is authorized & possesses the secret key or password required to decrypt & interpret the ciphertext.


Another important aspect to note is that decryption process usually requires a different key than the one used in the encryption process, so as to further increase the security of the message being conveyed.



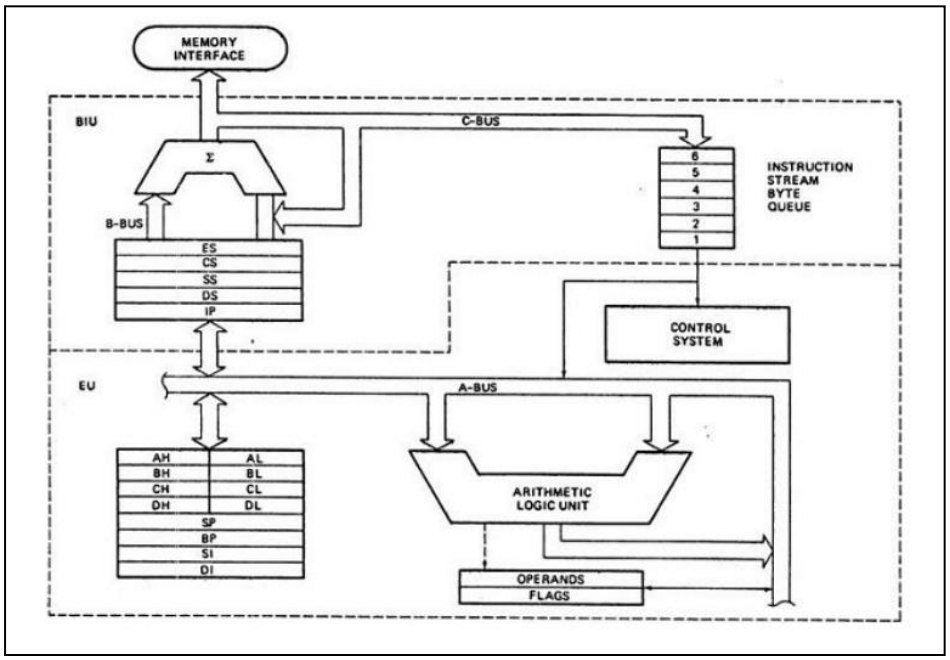

The graphic shown above is a block diagram of the entire process of a cryptographic transfer of any given message. It gives a general intuition of how both the encryption process at the transmitters end & the decryption process at the receivers end tie in with one another to complete the entire process.

# 8086 Microprocessor

In this section, we take a brief look at the 8086 microprocessor designed & released by Intel on June 8, 1978. Intel 8086 Microprocessor was designed to provide an order of magnitude increase in processing throughput over the older 808x
Intel 8086 is a 16-bit microprocessor with 16-bit Data bus/ALU, 20-bit address bus and Maximum clock frequency is 5 MHz.

### Architecture of 8086

The 8086 processor architecture is described in terms of its memory structure, register structure, instruction set, and external interface. The architecture of a 8086 microprocessor may be represented by the figure shown below:




Architecture of the 8086 microprocessor

The 8086 has eight more or less general 16-bit registers (including the stack pointer but excluding the instruction pointer, flag register and segment registers). Four of them, AX, BX, CX, DX, can also be accessed as twice as many 8-bit registers (see figure) while the other four, SI, DI, BP, SP, are 16-bit only.

The registers are shown in the table below. Also, nine flags record the processor state and control its operation: The status register (flag register) is a 16-bit register, 9 out of these 16 bits are active and indicate the current state of the processor. These bits include: Carry flag (CF), Parity flag (PF), Auxiliary flag (AF), Zero flag (ZF), Sign flag (SF), Trap flag (TF), Interrupt flag (IF), Direction flag (DF) and Overflow flag (OF). The layout of the flag register is also shown in the table below.

| | | | |
|---|---|---|---|
| *General Purpose Registers* | AH | AL | AX (primary accumulator) |
| | BH | BL | BX (base, accumulator) |
| | CH | CL | CX (counter, accumulator) |
| | DH | DL | DX (accumulator, other functions) |
| *Index and base registers* | SI | | Source Index |
| | DI | | Destination Index |
| | BP | | Base Pointer |
| | SP | | Stack Pointer |
| *Status register* | 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 | | (bit position) |
| | - - - - O D I T S Z - A - P - C | | Flags |
| *Segment register* | CS | | Code Segment |
| | DS | | Data Segment |
| | ES | | Extra Segment |
| | SS | | Stack Segment |
| *Instruction pointer* | IP | | Instruction Pointer |

A table displaying the register layout of the 8086 microprocessor

8086 has approximately 117 different instructions [1, 3, 4] with about 300 op-codes with three instruction formats: no-operand, single-operand and two-operand instructions as well as the string instructions that involve array operations. Intel 8086 instructions classified into 8 groups: Data transfer instructions, Arithmetic instructions, Bit Manipulation instructions, String instructions, Unconditional Transfer instructions, Conditional Branch instructions, Interrupt instructions, and Processor Control instructions. Intel 8086 provides various 12 different addressing modes to access instruction operands. The operand may be contained in: register, immediate, memory or I/O ports. The addressing Modes are classified into 5 groups: Register and

immediate modes (two modes), Memory addressing modes (six modes), Port addressing mode (two modes), Relative addressing mode (one mode) and Implied addressing mode (one mode).

# Simulation

For the simulation aspect of this project we have developed a few programs in assembly language using the emu8086 software that simulate the encryption & decryption processes but for different parameters and with variable flexibilities of the range of characters that may be used in the original text message.

## 8086 subroutine to encrypt/decrypt lower case characters using xlat

The first program that we developed using emu8086 software is a subroutine that encrypts/decrypts any message written in lower case characters using the 'xlat' function.

The reason behind choosing emu8086 as our choice of simulation software was that functions like 'xlat' are easily accessible which allow for short & concise codes to do something that may seem as complicated as encryption & decryption.

The code for the subroutine is displayed below:-

```
1. ; Microprocessor Interfacing Midterm Project
2.
3. ; Dhruv Tyagi (2K19/EP/032) & Ayush Kumar (2K19/EP/030)
4.
5. ; 8086 subroutine to encrypt/decrypt lower case characters using xlat
6.
7. name "crypt"
8.
9. org 100h
10.
11.
12.       jmp start
13.
14.       ; string has '$' in the end:
15.       string1 db 'dhruv', 0Dh,0Ah, '$' ; '$' used to denote end of
   string
```

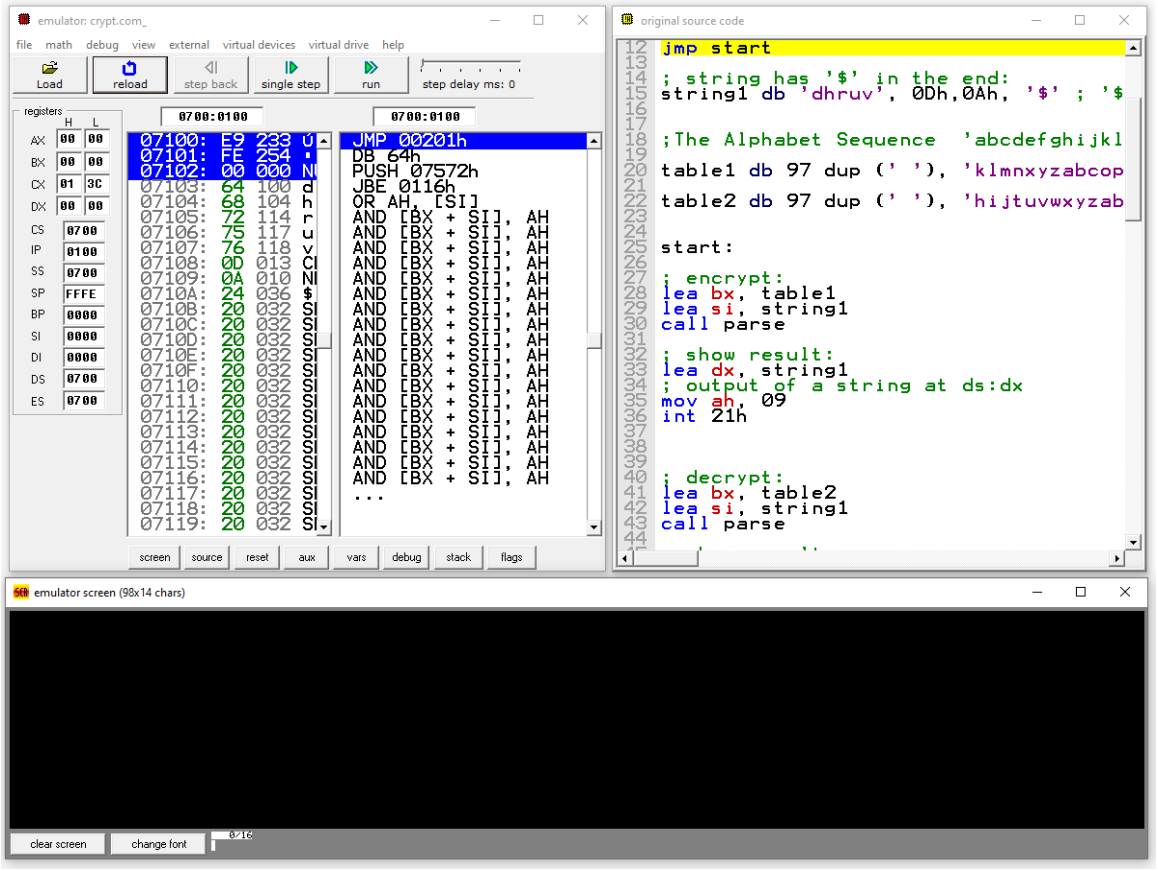
```
16.
17.
18.       ;The Alphabet Sequence  'abcdefghijklmnopqrstuwxyz'
19.
20.       table1 db 97 dup (' '), 'klmnxyzabcopqrstvuwdefghij'
21.
22.       table2 db 97 dup (' '), 'hijtuvwxyzabcdklmnoprqsefg'
23.
24.
25.       start:
26.
27.       ; encrypt:
28.       lea bx, table1
29.       lea si, string1
30.       call parse
31.
32.       ; show result:
33.       lea dx, string1
34.       ; output of a string at ds:dx
35.       mov ah, 09
36.       int 21h
37.
38.
39.
40.       ; decrypt:
41.       lea bx, table2
42.       lea si, string1
43.       call parse
44.
45.       ; show result:
46.       lea dx, string1
47.       ; output of a string at ds:dx
48.       mov ah, 09
49.       int 21h
50.
51.       ; wait for any key...
52.       mov ah, 0
53.       int 16h
54.
55.
56.       ret    ; exit to operating system.
57.
58.
59.
```

```
60.
61.       ; subroutine to encrypt/decrypt
62.       ; parameters:
63.       ;             si - address of string to encrypt
64.       ;             bx - table to use.
65.       parse proc near
66.
67.       next_char:
68.             cmp [si], '$'       ; end of string denoted by "$"
69.             je end_of_string
70.
71.             mov al, [si]
72.             cmp al, 'a'
73.             jb  skip
74.             cmp al, 'z'
75.             ja  skip
76.             ; xlat algorithm: al = ds:[bx + unsigned al]
77.             xlatb      ; encrypt using table1 & decrypt using table2
78.             mov [si], al
79.       skip:
80.             inc si
81.             jmp next_char
82.
83.       end_of_string:
84.
85.
86.       ret
87.       parse endp
88.
89.
90.
91. end
```

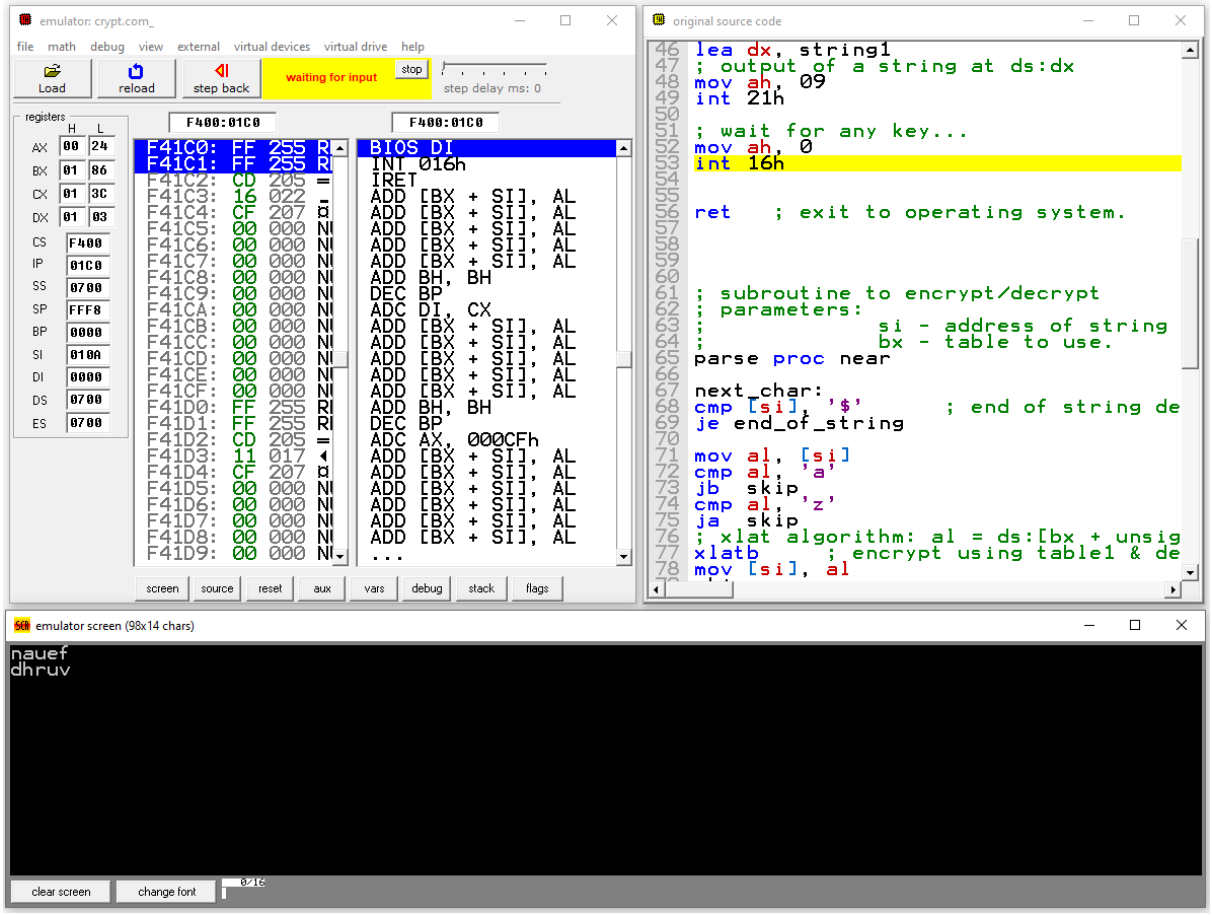
Note: - The string being currently encoded & then decoded is ‘dhruv’. The results as seen from screenshot of the output window for string ‘dhruv’ are as shown below

## Screenshots of the Output windows for plaintext ‘dhruv’:

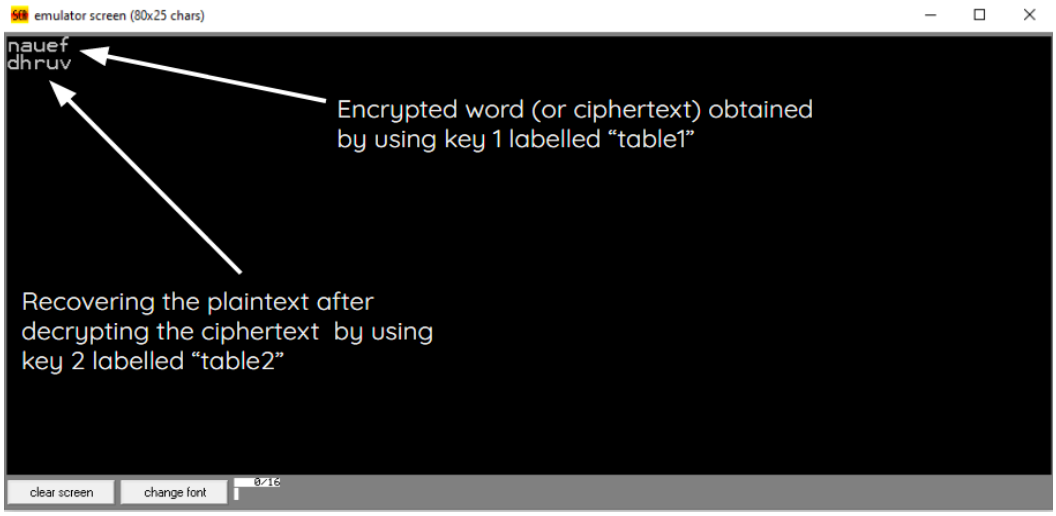
<u>Before Execution</u>




## After Execution



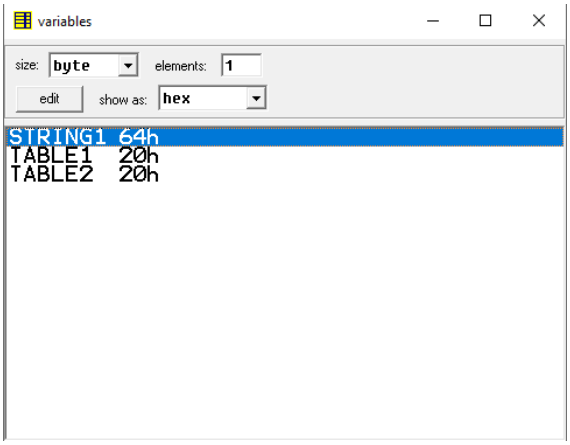

As seen from the after execution snapshot above, the plaintext ‘dhruv’ is first encoded into the string ‘nauef’ as per the array table1. Thus in this particular example ‘nauef’ is the cipher text. Consequently, the ciphertext ‘nauef’ is decrypted to recover the original plaintext ‘dhruv’.



# Algorithm behind the code

The working of the 8086 subroutine to encrypt/decrypt lower case characters using xlat has been in the previous section as illustrated by the output window snapshots. In this section, the algorithm behind the code used is briefly discussed.

- To begin , the subroutine is labeled ‘crypt’ & the program control is sent straight to the instruction line labeled ‘start’ where the program execution begins
- The line “string1 db 'dhruv', 0Dh,0Ah, '$'” basically uses the db directive to declare byte type variable & set aside 1 or more storage locations for the string ‘dhruv’ labeled “string1”.
- ‘0Dh’ in hex is 10 & corresponds to the ASCII code for linefeed (‘\n’) which is used to move the cursor to the next line but same column & ‘0Ah’ in hex is 13 & corresponds to the ASCII code for carriage return (‘\r’) which is used to move the cursor to the beginning of the next line.
- Lines 20 & 22 are used to create 2 arrays labeled ‘table1’ & ‘table2’ respectively. Both of these are allocated a space of 97 bytes of storage & are initialized with the 2 keys used in encryption & decryption processes. table1 array is initialized with key ‘klmnxyzabcopqrstvuwdefghij’ & table2 array is initialized with key ‘hijtuvwxyzabcdklmnoprqsefg’.
- Hence our 3 initially defined variables that will stay constant for the remainder of the program are as shown in the variable list provided by emu8086:




- The ‘start’ section of the code begins, & to start the encryption process effective address of table1 is loaded into the bx register & effective address of string1 is loaded into the source index. After which loop ‘parse’ is called.
- Program control shifts to instruction line ‘parse’ (code line 65). The loop parse forms the basis for both the encryption & the decryption processes in the program. What ‘parse’ effectively does is, it encrypts the string as per table1 & then later on also decrypts it using table2.
- The exact functioning of **parse loop** is as follows:
    - Initally, next_char instruction line begins & the parse loop effectively checks for the ‘$’ which denotes the end of the string. It does so by comparing current value pointed to by source index with ‘$’. If values are equal program control is sent to end_of_string instruction line & the encryption/decryption process is over.
    - Then the current value pointed to by source index is moved into the al register & value in al register is compared with ‘a’. If the current value pointed to by source index is below ‘a’ the current character is skipped. Similarly if the current value pointed to by source index is above ‘z’ it is also skipped. The purpose behind these lines is to make sure that the string currently being encypted/decrypted contains only lower case letters as this code is designed to work for messages composed of only lowercase letters. So for example if any

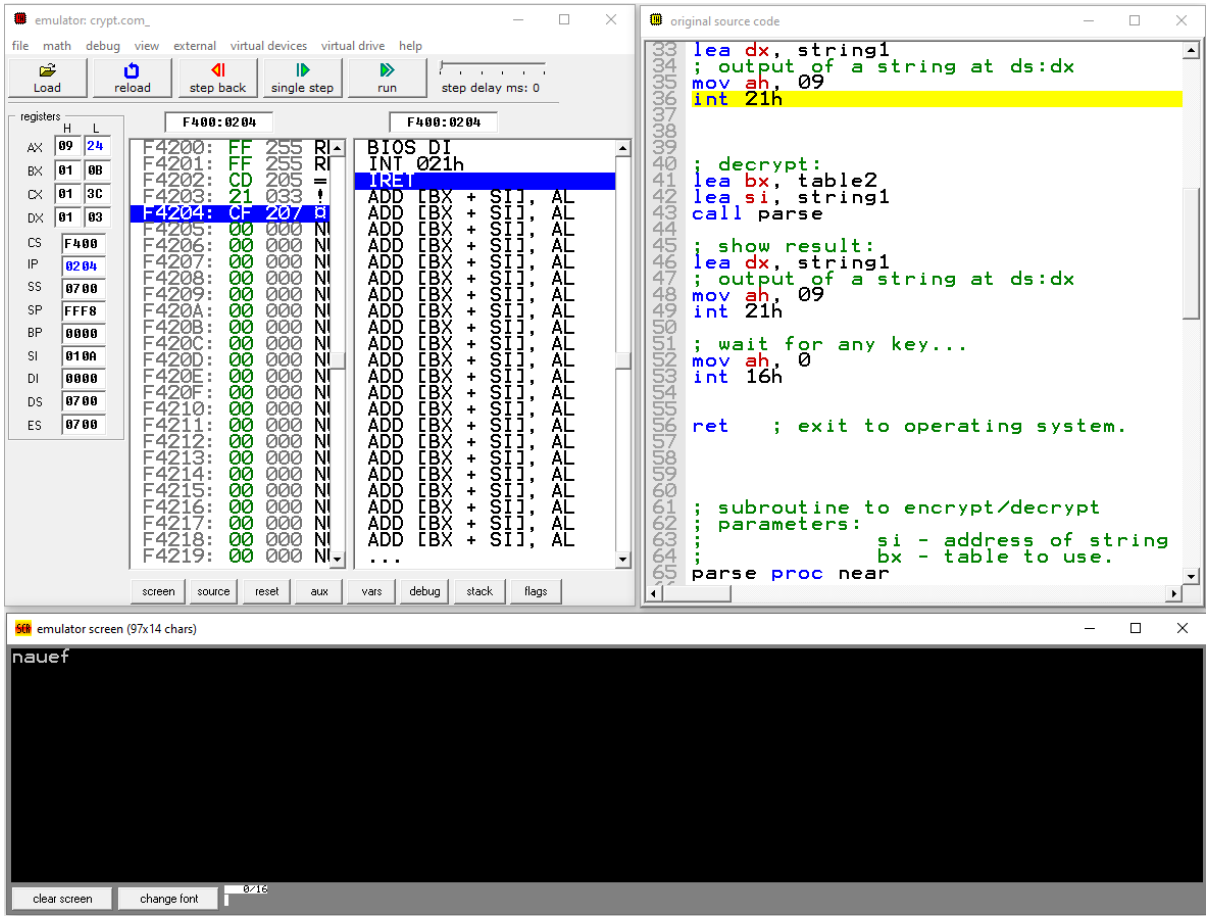
numbers are added in string1 they are not encrypted or decrypted i.e. they remain as they are as seen below


As seen above the program does not encypt/decrypt upper case characters, special symbols or numbers such as 1

- Then 'xlatb' is used to basically encrypt/decrypt the current character of the string pointed to by the source index. '**xlat**' is a commonly used instruction in assembler; the escape code instruction, also called table lookup instruction is basically used to move into AL, the contents of memory location in the data segment (DS) which forms the segment address whose offset address is the sum (BX+ unsigned AL). '**xlatb**' is the no operand form used when BX will always reside in the data segment.
- Then the original character from the plaintext will be replaced by the encrypted/decrypted character as per table1/table2 respectively.
- It then increments the value in source index so that the source index points to the next character in the string sequence & sends program control back to next_char (starting of loop)
- For a string like 'dhruv', the loop 'parse' will run for (n+2) iterations where n=no. of letters in message & it runs for additional 2 times since the loop it also runs for the '0Dh' & '0Ah' commands.

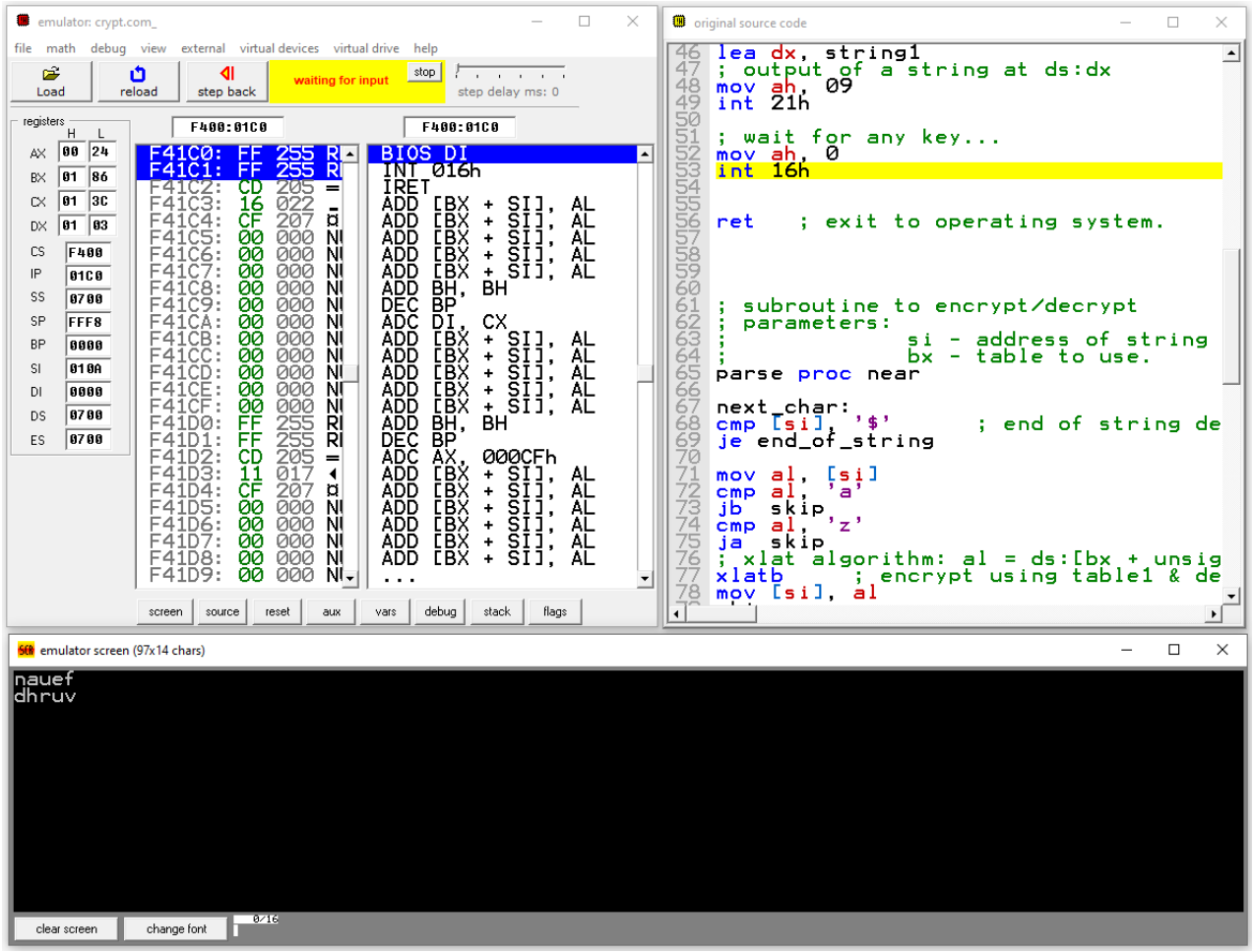
- Once loop 'parse' has ended it sends program control back to the next instruction line in the code sequence, i.e. to line 33 (under comment '; show result')

- The section of code used to show the result begins by loading the encrypted string1 into dx register. Then ‘mov ah, 09’ is used to call the interrupt 9H in AH register, followed by ‘int 21h’ which basically is used to show the result i.e. the encrypted cipher message stored in string1 in the output BIOS window as follows:




Stage in program execution where only result of encrypted cipher message has been output in the BIOS window

- After the result of the encrypted cipher message has been output, the decryption process & result of the decrypted plain text is then output in the same manner, as the whole process for the encryption process has been observed.




Final result obtained after the 'parse' loop has run for the decryption protocol & the output of the decrypted plaintext is printed in the BIOS output window.

This was the basic outline behind the algorithm implemented for a program to encrypt/decrypt any given plain text composed of only lowercase characters.

# Limitations & Improvements

In this section we address some of the limitations of the model used for encryption/decryption protocols & we also consider potential improvements that could be made to further enhance the program.

Some of the limitations faced by our implementation of the encryption/decryption program are as follows:

- The cipher text we obtain is unable to mask the no. of characters in the original plain text. This fact might compromise the security of the original plaintext & lead to unintended recipients guessing the plain text.
- The cipher text is unable to encrypt upper case letters, numbers and special symbols. This reduces the degree of freedom that the composer of the original message has while writing his/her message to be encrypted.
- The cipher text only masks the plain text using lower case alphabets, again this makes it easier for any third party viewers of the cipher text to figure out the pattern behind the encryption.
- The keys used for encryption & decryption are not sufficiently complicated enough to alter the plain text enough to make it completely unintelligible.

To address these limitations an alternate version of the previous program shown might be implemented as shown through the source code given below:-

```
1.  .model small
2.
3.  org 100h
4.  .data
5.      ; name type initializer
6.      sum DW 0
7.      delta DW 02ACh
8.      v0 DW ?
9.      v1 DW ?
10.     k0 DW ?
11.     k1 DW ?
```

```
    k2 DW ?
    k3 DW ?
    msgV DB 'Enter text of 4 letters: $'
    msgK DB 0Dh,0Ah,'Enter password of 4 keys: $'
    encrypting DB 0Dh,0Ah,'Encrypting... $'
    decrypting DB 0Dh,0Ah,'Decrypting... $'
    encryptedMsg DB 0Dh,0Ah,'Encrypted text: $'
    decryptedMsg DB 0Dh,0Ah,'Decrypted text again: $'

; DB 8-bit integer
; DW 16-bit integer
; DD 32-bit integer or real
; DQ 64-bit integer or real
; DT 80-bit integer (10 byte)

.code

main proc

    ; "Enter text of 4 letters: "
    mov ah, 09h       ; write string (from "dx")
    lea dx, msgV      ; lea for LoadEffectiveAddress
    int 21h           ; Dos interrupt "do it"

    ; reading v0
    mov ah, 01h       ; read char (stored in "al")
    int 21h
    mov bh, al
    int 21h
    mov bl, al
    mov v0, bx

    ; reading v1
    mov ah, 01h       ; read char (stored in "al")
    int 21h
    mov bh, al
    int 21h
    mov bl, al
    mov v1, bx

    ; "Enter password of 4 keys: "
    mov ah, 09h
    lea dx, msgK
    int 21h
```

```
56.
57.     ; reading k[0]
58.     mov ah, 01h
59.     int 21h
60.     xor bx, bx
61.     mov bl, al
62.     mov k0, bx
63.
64.     ; reading k[1]
65.     mov ah, 01h
66.     int 21h
67.     xor bx, bx
68.     mov bl, al
69.     mov k1, bx
70.
71.     ; reading k[2]
72.     mov ah, 01h
73.     int 21h
74.     xor bx, bx
75.     mov bl, al
76.     mov k2, bx
77.
78.     ; reading k[3]
79.     mov ah, 01h
80.     int 21h
81.     xor bx, bx
82.     mov bl, al
83.     mov k3, bx
84.
85.     ; "Encrypting..."
86.     mov ah, 09h
87.     mov dx, offset encrypting
88.     int 21h
89.
90.     ; encrypt the text
91.     call encrypt
92.
93.     ; "Encrypted text: "
94.     mov ah, 09h
95.     mov dx, offset encryptedMsg
96.     int 21h
97.
98.     mov ah, 02h
99.     ; print v0
```

```
100.            mov bx, v0
101.            mov dl, bh
102.            int 21h
103.            mov dl, bl
104.            int 21h
105.
106.            ; print v1
107.            mov bx, v1
108.            mov dl, bh
109.            int 21h
110.            mov dl, bl
111.            int 21h
112.
113.            ; "Decrypting..."
114.            mov ah, 09h
115.            mov dx, offset decrypting
116.            int 21h
117.
118.            ; decrypt the text
119.            call decrypt
120.
121.            ; "Decrypted text: "
122.            mov ah, 09h
123.            mov dx, offset decryptedMsg
124.            int 21h
125.
126.            mov ah, 02h
127.            ; print v0
128.            mov bx, v0
129.            mov dl, bh
130.            int 21h
131.            mov dl, bl
132.            int 21h
133.
134.            ; print v1
135.            mov bx, v1
136.            mov dl, bh
137.            int 21h
138.            mov dl, bl
139.            int 21h
140.
141.            ; stop the program
142.            mov ah, 4ch
143.            int 21h
```

```
144.
145.            endp
146.
147.
148.        ; ============================================== encryption procedure
    ============================================== ;
149.            encrypt proc
150.
151.                mov cx, 8       ; counter for "loop" instruction
152.
153.        encLoop:
154.                ; sum += delta
155.                mov bx, delta
156.                mov ax, sum
157.                add ax, bx
158.                mov sum, ax
159.
160.                ; v0 += ((v1<<4) + k0) ^ (v1 + sum) ^ ((v1>>5) + k1)
161.                ; dx = ((v1<<4) + k0)
162.                mov ax, v1
163.                shl ax, 4
164.                mov bx, k0
165.                add ax, bx
166.                mov dx, ax
167.                ; dx ^= (v1 + sum)
168.                mov ax, v1
169.                mov bx, sum
170.                add ax, bx
171.                xor dx, ax
172.                ; dx ^= ((v1>>5) + k1)
173.                mov ax, v1
174.                shr ax, 5
175.                mov bx, k1
176.                add ax, bx
177.                xor dx, ax
178.                ; v0 += dx
179.                mov ax, v0
180.                add ax, dx
181.                mov v0, ax
182.
183.                ; v1 += ((v0<<4) + k2) ^ (v0 + sum) ^ ((v0>>5) + k3)
184.                ; dx = ((v0<<4) + k2)
185.                mov ax, v0
186.                shl ax, 4
```

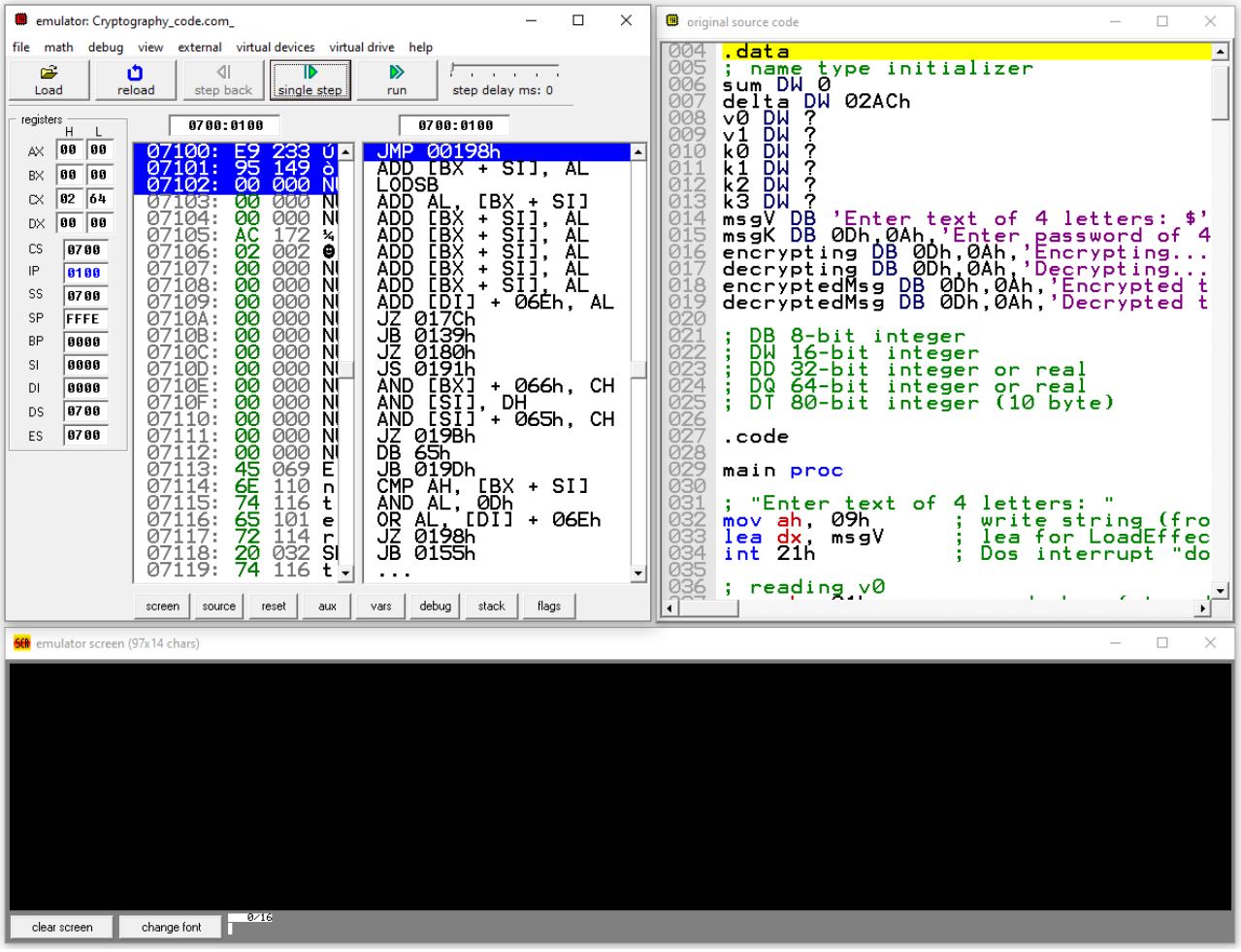
```
187.                mov bx, k2
188.                add ax, bx
189.                mov dx, ax
190.                ; dx ^= (v0 + sum)
191.                mov ax, v0
192.                mov bx, sum
193.                add ax, bx
194.                xor dx, ax
195.                ; dx ^= ((v0>>5) + k3)
196.                mov ax, v0
197.                shr ax, 5
198.                mov bx, k3
199.                add ax, bx
200.                xor dx, ax
201.                ; v1 += dx
202.                mov ax, v1
203.                add ax, dx
204.                mov v1, ax
205.
206.        loop encLoop          ; "loop" use "cx" as its counter
207.
208.                ret
209.            encrypt endp
210.        ; ============================================= END of encryption proc
    ============================================= ;
211.
212.
213.
214.
215.        ; ============================================= decryption procedure
    ============================================= ;
216.            decrypt proc
217.
218.                mov cx, 8       ; counter for "loop" instruction
219.
220.        decLoop:
221.                ; v1 -= ((v0<<4) + k2) ^ (v0 + sum) ^ ((v0>>5) + k3)
222.                ; dx = ((v0<<4) + k2)
223.                mov ax, v0
224.                shl ax, 4
225.                mov bx, k2
226.                add ax, bx
227.                mov dx, ax
228.                ; dx ^= (v0 + sum)
```

```
229.                mov ax, v0
230.                mov bx, sum
231.                add ax, bx
232.                xor dx, ax
233.                ; dx ^= ((v0>>5) + k3)
234.                mov ax, v0
235.                shr ax, 5
236.                mov bx, k3
237.                add ax, bx
238.                xor dx, ax
239.                ; v1 -= dx
240.                mov ax, v1
241.                sub ax, dx
242.                mov v1, ax
243.
244.                ; v0 -= ((v1<<4) + k0) ^ (v1 + sum) ^ ((v1>>5) + k1)
245.                ; dx = ((v1<<4) + k0)
246.                mov ax, v1
247.                shl ax, 4
248.                mov bx, k0
249.                add ax, bx
250.                mov dx, ax
251.                ; dx ^= (v1 + sum)
252.                mov ax, v1
253.                mov bx, sum
254.                add ax, bx
255.                xor dx, ax
256.                ; dx ^= ((v1>>5) + k1)
257.                mov ax, v1
258.                shr ax, 5
259.                mov bx, k1
260.                add ax, bx
261.                xor dx, ax
262.                ; v0 -= dx
263.                mov ax, v0
264.                sub ax, dx
265.                mov v0, ax
266.
267.                ; sum -= delta
268.                mov bx, delta
269.                mov ax, sum
270.                sub ax, bx
271.                mov sum, ax
272.
```

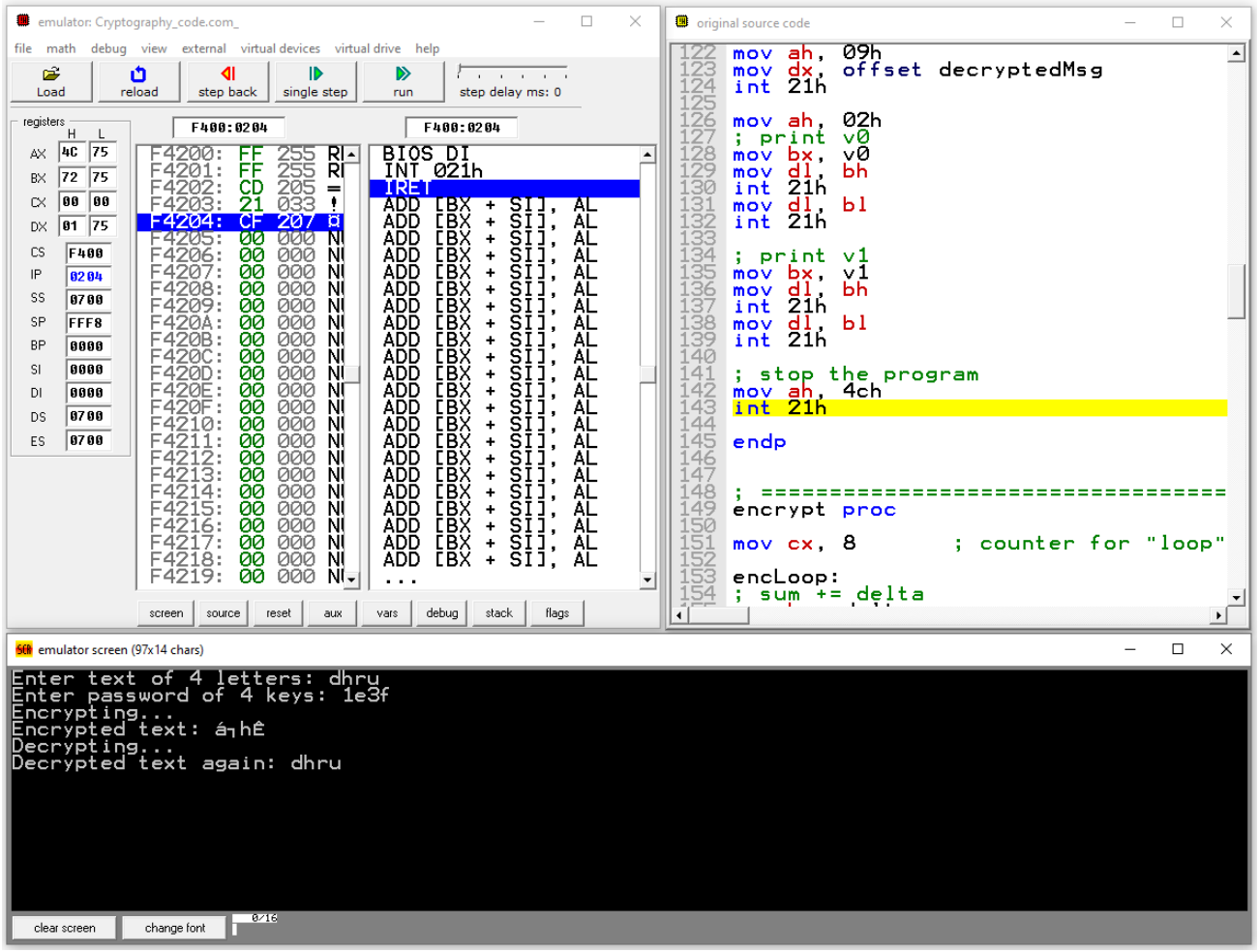
```
273.        loop decLoop          ; "loop" use "cx" as its counter
274.
275.            ret
276.        decrypt endp
277.        ; ============================================== END of decryption proc
    ============================================== ;
278.
279.
280.        end
```

## Screenshots of Output windows for plaintext ‘dhru’ & password ‘1e3f’

Before Execution



## After Execution




As seen this model overcomes the limitations faced by the previous model however due to its extended functionality it is observed that the lines of code stretch to almost 280 as compared to the miniscule 91 lines of code that it required from our implementation. Also the execution time of the simpler model was far less than the execution time required by the more advance program. Hence each model of implementation of encryption/decryption procedures possess their own pros & cons, in the end it is upto the user to determine which program is of their preference.

# Applications of Cryptography

As mentioned earlier, cryptography is an important aspect of secure digital communication & data security in today's world. As such, there is a wide range of applications of cryptography in a multitude of fields. Some of the prevalent applications of cryptography & cryptographic techniques have been briefly discussed under this section.

A few important applications of cryptography may be listed as:

## Complete Data Protection

Data breaches are a real problem for businesses of all sizes, although many smaller businesses wrongly assume that it is a problem that mostly affects larger brands. There are many benefits that encryption could bring to your company, outside of those that you might already suspect. If you are considering encryption for your business, you should take these benefits into account before making your decision.

With the right encryption solution, you can go through each day knowing that your data is safe and that there is no reasonable way in which hackers could potentially get their hands on the raw data. It would take a brute-force style program more than a lifetime to successfully decode all of the information.

Although there are other means in which data could potentially be accessed, taking the simple step toward encryption helps to make the job too difficult for hackers that might ordinarily be interested in targeting your business.

## Reliability in Transmission

A conventional approach that allows reliability is to carry out a checksum of the communicated information and then communicate the corresponding checksum in an encrypted format. When both the checksum and encrypted data

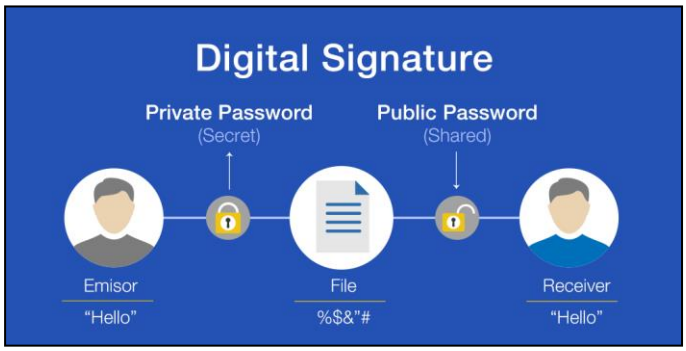
is received, the data is again checksummed and compared to the communicated checksum after the process of decryption. Thus, effective cryptographic mechanisms are more crucial to assure reliability in message transmission.

## Authentication of Identity

Cryptography is strongly linked to the approach of using passwords, and innovative systems probably make use of strong cryptographic methods together with the physical methods of individuals and collective secrets offering highly reliable verification of identity.

## Digital Signature

Digital signatures work by proving that a digital message or document was not modified—intentionally or unintentionally—from the time it was signed. Digital signatures do this by generating a unique hash of the message or document and encrypting it using the sender's private key.




Digital signatures employ asymmetric cryptography. In many instances they provide a layer of validation and security to messages sent through a non-secure channel: Properly implemented, a digital signature gives the receiver reason to believe the message was sent by the claimed sender. Digital signatures are equivalent to traditional handwritten signatures in many respects, but properly implemented digital signatures are more difficult to forge than the handwritten type.

# Benefits of Cryptography

Businesses that have already invested in traditional perimeter IT security firewalls and detection systems are continuously looking for something more to protect their data. Between external hackers and internal staff, protecting data in all states is too tall a task for most legacy solutions. To combat this trend, enterprises are considering data protection and specifically data encryption solutions as their next implementation. Encryption can be an excellent choice for businesses that are transferring large amounts of data and want to be certain that they are able to keep themselves safe from potential data theft. The issues with encryption solutions that are too heavy or negatively impact the user experience negatively are now in the past.

**1. Security across Multiple Devices**

With smart phones and other mobile devices gaining popularity in recent years, many companies have struggled to find a solution for keeping the data stored and passed through these devices safe from potential theft. Luckily, data encryption software will allow you to ensure that all data, across any device, is completely encrypted with the same protections in place that you would find in data stored on a desktop computer. Data encryption will help to take an untenable, stressful situation and make it manageable, while providing peace of mind. On top of that, device authentication can eliminate risk of infiltration from unwanted users.

**2. Move Data Securely**

One of the most vulnerable aspects of data emerges during the transport process. While SSL/TLS is the industry standard for data in motion it has many disadvantages for your data security. An effective encryption solution helps to ensure that data is protected at all times, at rest and in motion. Files that are shared or uploaded to cloud systems should be to ensure that the files remain safe throughout the transport process.

### 3. Integrity Maintained

One of the worries that many organizations of all sizes share is whether or not the encryption process will affect the integrity of their data. Although data theft is a very common problem, another way for hackers to commit data fraud is to knowingly alter the data that is available. Encryption keeps your data safe from alterations, and recipients of the data will be able to see if it has been tampered with. Alteration of data is something that many businesses often overlook when they are looking into ways to keep their data safe.

### 4. Ensure Compliance

Compliance is extremely important, and many IT departments must comply with legal, insurance and industry restrictions on how data can be handled and transmitted. Encryption provides one of the safest ways for a business to transmit and store data and comply with the restrictions that your business currently has in place such as FIPS, FISMA, HIPAA, PCI/DSS or Gramm-Leach Bliley.

Table 1: PKCS Specifications

| No. | PKCS title | Comments |
|---|---|---|
| 1 | RSA Cryptography Standard | |
| 2,4 | | incorporated into PKCS #1 |
| 3 | Diffie-Hellman Key Agreement Standard | superseded by IEEE 1363a etc. |
| 5 | Password-Based Cryptography Standard | |
| 6 | Extended-Certificate Syntax Standard | never adopted |
| 7 | Cryptographic Message Syntax Standard | superseded by RFC 3369 (CMS) |
| 8 | Private-Key Information Syntax Standard | |
| 9 | Selected Object Classes and Attribute Types | |
| 10 | Certification Request Syntax Standard | |
| 11 | Cryptographic Token Interface Standard | referred to as CRYPTOKI |
| 12 | Personal Information Exchange Syntax Standard | |
| 13 | (*reserved for ECC*) | never been published |
| 14 | (*reserved for pseudo random number generation*) | never been published |
| 15 | Cryptographic Token Information Syntax Standard | |

A few public key cryptography standards

# Conclusion

To conclude, through this project we were able to learn about different cryptographical methods such as secret-key & public key protocols. We explored the vast range of applications in fields of electronic commerce, chip-based payment cards, digital currencies, and military communications.

Through this project we also received the opportunity of further increasing our knowledge of Intel's 8086 microprocessor through the use of software's such as emu8086 which provided us a streamlined platform for assembly language coding. Designing encryption/decryption subroutines using assembly language allowed us to gain a deep intuition behind how these processes are being implemented in today's world.

Once again we would like to thank our professor Dr. Rishu Chaujar who gave us the opportunity to work on this project & further our knowledge on the topic to its current state.

# References

The references & resources used in this project have been cited below:

S.Morse, B. Ravenel, S. Mazor and W.Pohlman, " Intel Microprocessors: 8008 to 8086", IEEE Computer, Vol 13, No. 10, pages 42-60, October 1980.

[2017 Lina Gonga, Li Zhang, Wei Zhang, Xuhong Li, Xia Wang and Wenwen Pan, The Application of Data Encryption Technology in Computer Network Communication Security]

Crytography and algorithm by ELPROCUS.

Lice & Gibson, —Microcomputer System 8086 / 8088|| PHI, 2nd Edition

Benefits of Cryptography by James Salter CENTRI.

Max Caceres, Tim Robichaux, Dario V. Forte, Eric S. Seagren, Devin L. Ganger, Brad Smith, Wipul Jayawickrama, Christopher Stokes, Jan Kanclirz, Jr. :Next Generation SSH2 Implementation 2009, Pages 41-64

Uffenback, —The 8086 Family Design|| PHI, 2nd Edition.

Links:

https://community.jisc.ac.uk/library/advisory-services/introduction-cryptographic-techniques

https://www.semanticscholar.org/paper/Public-Key-Cryptography-Standards%3A-PKCS-Wang/4dc3387459dbe471689cc71096db79290287d299

https://en.wikipedia.org/wiki/Cryptography

https://ecestudy.files.wordpress.com/2015/02/mpi-unit-1.pdf

http://people.csail.mit.edu/rivest/crypto-security.html

https://old.amu.ac.in/emp/studym/99993168.pdf

https://www.sciencedirect.com/science/article/pii/B9781597492836000039

https://www.researchgate.net/publication/340163935_LECTURE_NINE_8086_MICROPROCESSOR_MEMORY_AND_IO_INTERFACING

https://www.educba.com/cryptography-techniques/