SCHOOL OF COMPUTER SCIENCE ENGINEERING (SCOPE)

SYMMETRIC CRYPTOGRAPHY USING XOR CIPHER IN 8086

Submitted By:

Sri Shreya Volety (19BCB0036)
Sarthak Bajaj (19BCE0710)
Aparna Sahu (19BCE2125)

Project Report of Microprocessors and Interfacing (CSE2006)

Fall Semester 2021-22

Submitted to: **Signature:**
Faculty: Ragunath G
Date: 7-12-2021
Slot: E1

TABLE OF CONTENTS

1. Abstract

Cryptography is the process of encrypting or hiding information to protect data. Symmetric cryptography, which is the main focus of the project, uses the same key for both encryption and decryption, as opposed to asymmetric cryptography, which has a public and private key. The primary objective of our project is to use the XOR cipher to encrypt the contents of the file and store the output of the same in a new file. An XOR cipher is simple to configure, and is reasonably strong if the password length and the length of the string are close, so there is no discernible pattern. It is better than most substitution or transposition ciphers, and is often used in more complicated cryptography systems like DES. We intend to protect the files from malicious targets by using a private key that will only be known to the sender and the receiver. Each file will be encrypted separately, with different keys according to the choice of the user, and the output files will not be readable by anyone else except the authorized user.

2. Introduction

Securing data is an essential task in creating any file system, and cryptography is widely used in data security. The XOR cipher is a simple, inexpensive, robust cipher that is very useful in cryptography, because it has a balanced output, i.e., the ciphertext output can be very randomly 0 or 1, based on the plain text and key. This is not the case with other operations like AND, OR, etc. The XOR cipher has been used in computer malware, to prevent reverse engineering of the code. Further, it is frequently used in multiple different levels in algorithms like DES and AES. DES uses multiple rounds of XOR, along with other cryptographic mathematical functions, to secure data, and it is also a symmetric key cryptographic algorithm.

Using assembly language in the field of security is crucial, because as already mentioned, it is essential for security experts to understand how computer malware and file system security works, and train for it constantly. Further, assembly language interacts with the system on the architectural level, so developing security mechanisms in ALP is very important.

In our project, we have developed an algorithm to use the XOR cipher with a given key. As 8086 is a microprocessor and not a microcontroller, it does not have its own filing system, and hence we have to use DOS BOX, to access the input file and create the output file with the hashed output. The key that the user provides will be repeated until the length of the string is matched, and then the output shall be XORed. There will be a more extensive explanation of this in the implementation part of our study.

3. Literature Survey

Considering the main focus of our project is the XOR cipher used for encryption, our literature survey is centered around researching ALP implementations of the XOR cipher. The main point of the survey is to prove the robustness and scope of the XOR cipher.

3.1 XOR Cipher Based Cryptography and Authentication with Hardware Chip

Journal, Year: International Journal of Electronics Engineering, June - Dec 2018

This paper focuses mainly on access control, and user authentication, two fundamental principles in network and data security. Access control essentially defines who the administrator is giving access to, and who they are restricting, whereas user authentication is concerned with authenticating the user with their rank and credentials, hence the two ideas complement each other. This paper shows the hardware configuration of the XOR cipher, which is crucial to understand, even while simulating the cipher technique. For the chip design, Xilinx 14.2 software is used and it is simulated using Modelsim 10.0 software with VHDL programming.

3.2 Security Analysis of XOR Based Ciphered Image

Journal, Year: Asian Journal of Computer Science and Technology, 2018

This paper redirects the focus of XOR based cryptography in the image processing domain, and aims to analyze how effective the cipher is for the same. Scrambling is the process of breaking up an image, effectively making it unreadable, by severing the connection between neighboring pixels. It is often used as a step in data security, before a final measure like a watermark or a digital signature. This paper analysis the XOR cipher in the scrambling part of security. After using the cipher, the Unified Average Change Intensity (UACI- which effectively measures the difference between the original and ciphered image), and the Number of Pixels Change Rate (NPCR - a quantity used to understand for one pixel of the plain image, on apply the cipher, how many pixels of the ciphered image are changed) were calculated, and the values proved the robustness of the XOR cipher.

3.3 Cryptography Using Xor Cipher

Journal, Year: Research Journal of Science and Technology, 2017

This paper focuses on the encryption and decryption of ascii strings using the XOR cipher and the results of the algorithm were shown to be reasonably good. The three main steps of the procedure followed in the paper are as follows - (1) converting each character of the input text into extended ASCII code, which results in an input of 8N bits, if the original input had N bits, (2) constructing a string of 1's and 0's that match the length of the input string and (3) applying the XOR operation, and deconstructing the extended ASCII string back to its original form.

3.4 Arduino UNO and Android Based Digital Lock Using Combination of Vigenere Cipher and XOR Cipher

Journal, Year: Journal of Physics: Conference Series, June, 2020

This paper explores the efficiency of a combined Vigenere and XOR symmetric cipher in improving the working of a digital lock in Android based systems. Vigenere cipher is a polyalphabetic substitution cipher i.e it uses a series of interwoven caesar ciphers based on the keyword or the password chosen by the user. The Arduino UNO microcontroller is used for the digital lock prototyping, for it is easy and inexpensive to use, and the algorithm essentially uses 2 levels of cryptography, the first being the vigenere cipher, and the result of this is then XORed. A similar procedure is used for decryption, and this was shown to improve the secrecy in communication between two devices in a digital lock.

3.5 Securing Data at rest using Hill Cipher and XOR based operations

Journal, Year: IEEE 2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA) - Coimbatore (2018.3.29-2018.3.31)

This paper focuses on security of data in databases, and key management for the same. The combination of Hill Cipher and repeated XOR operations is used in this paper. Hill Cipher is also a substitution cipher that follows the logic of using modulo 26 on every letter in the given string. Owing to the simplicity of the algorithm and the increased pace of XOR operations, the paper concluded that the combination of the two was a fast and inexpensive way of securing data at rest (within static databases).

3.6 When an attacker meets a cipher-image in 2018

Journal, Year: Journal of Information Security and Applications, 2018

This is another paper that focuses on security of images, however, the XOR operations are reviewed in the context of a more complicated cryptographic algorithm called AES, which uses S-boxes (substitution boxes) to encrypt the images. It also reviews some permutations without substitution algorithms, and the comparative results of all of these XOR based systems is summarized in the paper.

4. Drawback of existing work and Proposed work

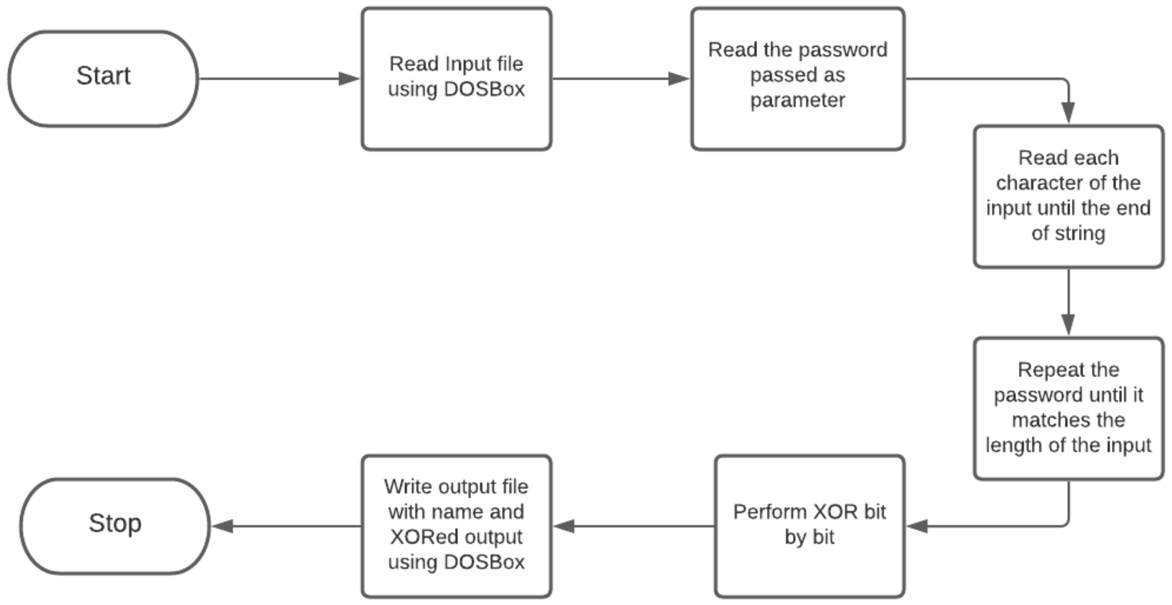
Of all the literature that was surveyed, one of the main problems was the time taken by the system to compute the cipher and write it to the output file. In our project, we have tried to rectify the same by choosing a medium length password to XOR the output with. Further, while the algorithm is discussed in most of the literature, implementation in ALP is provided only by a few papers, which is crucial to understanding how the algorithm would work in an assembly language context, considering most attacks also happen at the architectural level. As 8086 is a microprocessor, and requires interfacing with an external filing system, most literature has not used the processor in its implementation.

In our project, we are using the DOSBox software along with the 8086 microprocessor emulator, to access the input files, XOR the output and write the file to the default drive. This process is simple, robust, and greatly accelerated.

Our code essentially works on two levels, the first where the emulator is using the DOSBox to access the input files and write the output, and the second where functions are written to XOR the text and encrypt it. The project is neatly structured, and can be scaled for larger databases and also allows for levels of security in the levels, because the users are free to select passwords for each file.

5. Flowchart

The following is the flowchart of our project:


Start
Read Input file using DOSBox
Read the password passed as parameter
Read each character of the input until the end of string
Repeat the password until it matches the length of the input
Perform XOR bit by bit
Write output file with name and XORed output using DOSBox
Stop

6. Implementation

The detailed implementation of the project can be summarized by the following steps.

6.1 Reading the arguments passed in DOSBox

The first function essentially reads the arguments by the user. The arguments are the input file, the name of the output file to be created, and the password needed for XOR encryption and decryption. In our implementation, the input file was input.txt, the output was output.txt, and the password chosen was “MICRO”. The trial input text in our file was “this is just a trial file”.

6.2 Creating the Output file

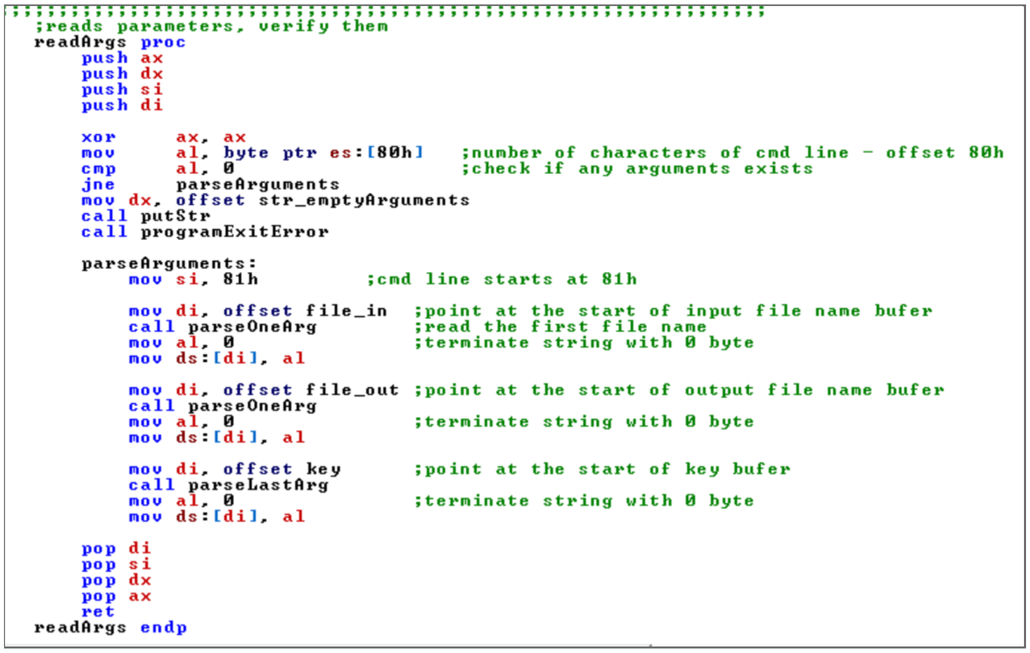
The function checks if the output file mentioned by the user already exists, or if it needs to be created. If the file doesn’t exist, the function saves the file descriptor of the output file, and creates the file. If the file already exists, a message is shown to the user explaining that the output file already exists, and asks if the user wants the contents of the user to be overwritten. The user can either choose to overwrite the file, or just provide a new file name for the output file.

6.3 Reading the input file piece by piece, XOR it, and save it to the output file

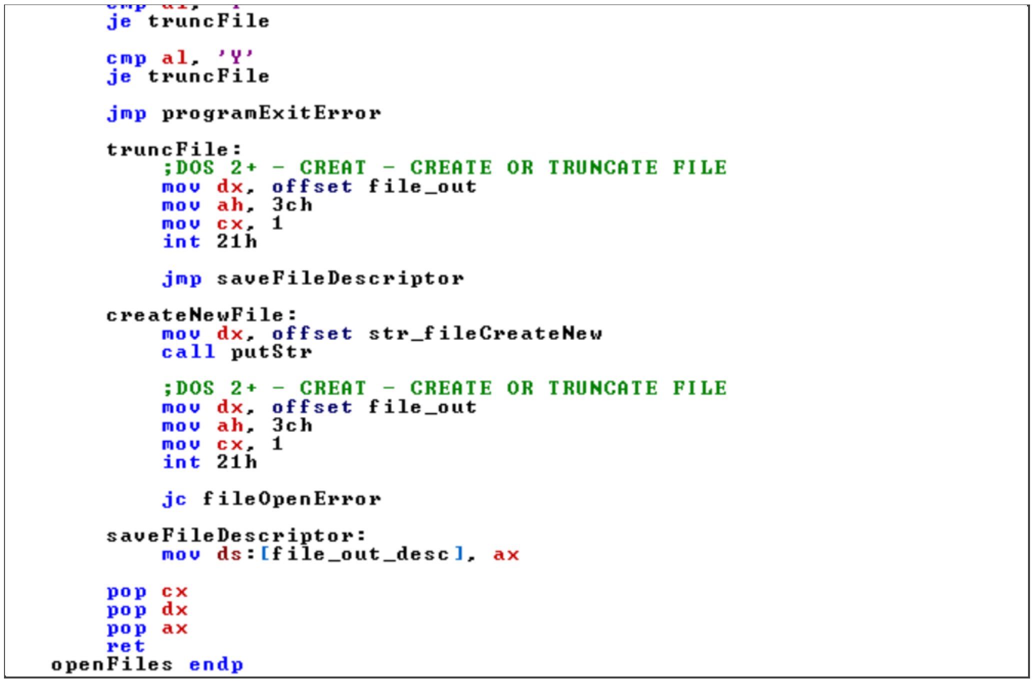
The function repeats the password to fill out the length of the password, performs bit by bit XOR on the input string and saves the encrypted character to the output file.

On execution, when the output file is created, the contents of the file are illegible and illogical and can only be decrypted by the secret key.

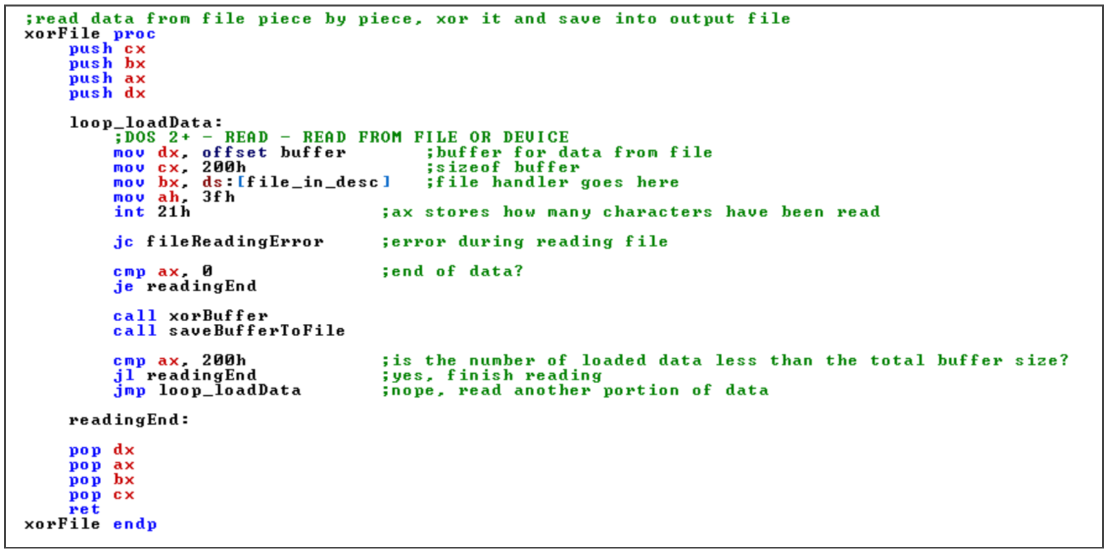
7. Screenshots of the Prototype



```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
  ;reads parameters, verify them
  readArgs proc
      push ax
      push dx
      push si
      push di

      xor     ax, ax
      mov     al, byte ptr es:[80h]    ;number of characters of cmd line - offset 80h
      cmp     al, 0                    ;check if any arguments exists
      jne     parseArguments
      mov dx, offset str_emptyArguments
      call putStr
      call programExitError

      parseArguments:
          mov si, 81h           ;cmd line starts at 81h

          mov di, offset file_in  ;point at the start of input file name bufer
          call parseOneArg        ;read the first file name
          mov al, 0               ;terminate string with 0 byte
          mov ds:[di], al

          mov di, offset file_out ;point at the start of output file name bufer
          call parseOneArg
          mov al, 0               ;terminate string with 0 byte
          mov ds:[di], al

          mov di, offset key      ;point at the start of key bufer
          call parseLastArg
          mov al, 0               ;terminate string with 0 byte
          mov ds:[di], al

      pop di
      pop si
      pop dx
      pop ax
      ret
  readArgs endp
```



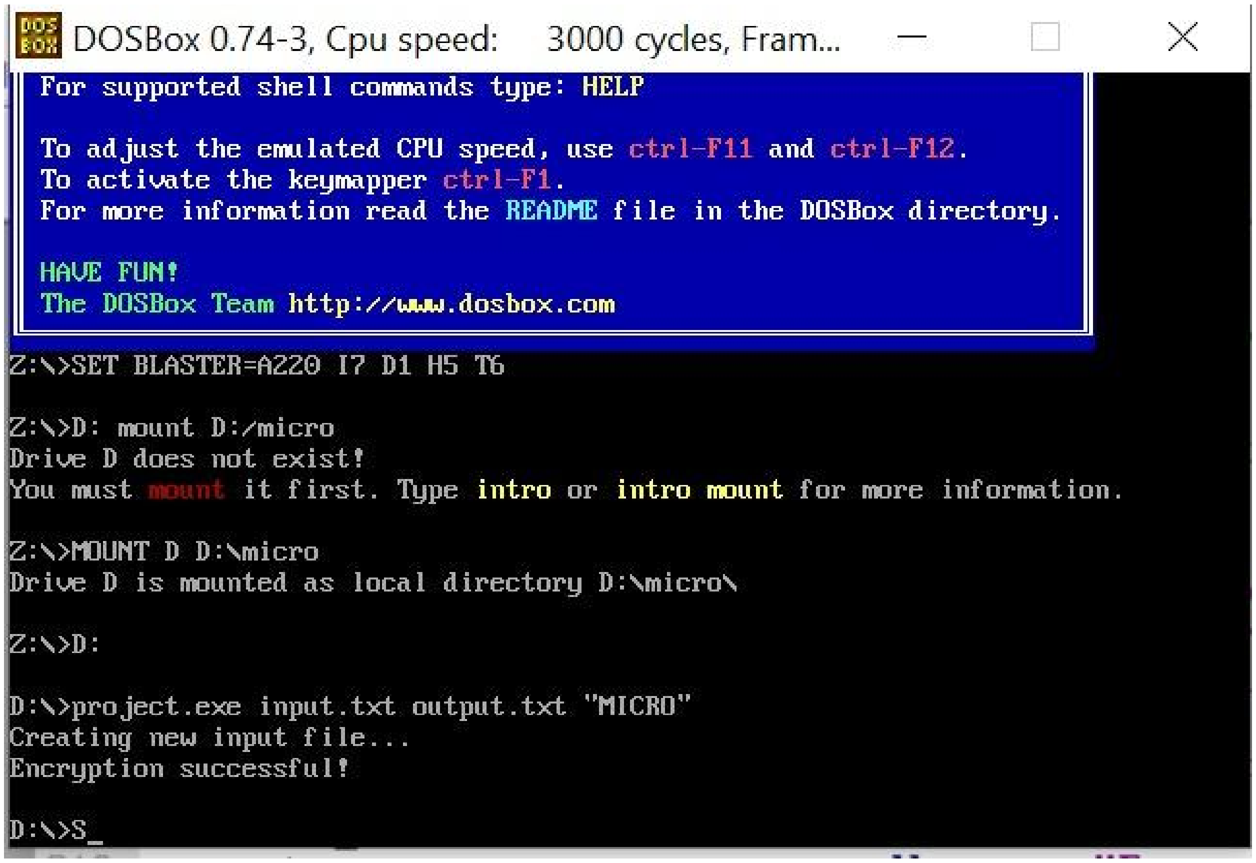
```
        je truncFile

        cmp al, 'Y'
        je truncFile

        jmp programExitError

        truncFile:
            ;DOS 2+ - CREAT - CREATE OR TRUNCATE FILE
            mov dx, offset file_out
            mov ah, 3ch
            mov cx, 1
            int 21h

            jmp saveFileDescriptor

        createNewFile:
            mov dx, offset str_fileCreateNew
            call putStr

            ;DOS 2+ - CREAT - CREATE OR TRUNCATE FILE
            mov dx, offset file_out
            mov ah, 3ch
            mov cx, 1
            int 21h

            jc fileOpenError

        saveFileDescriptor:
            mov ds:[file_out_desc], ax

        pop cx
        pop dx
        pop ax
        ret
    openFiles endp
```



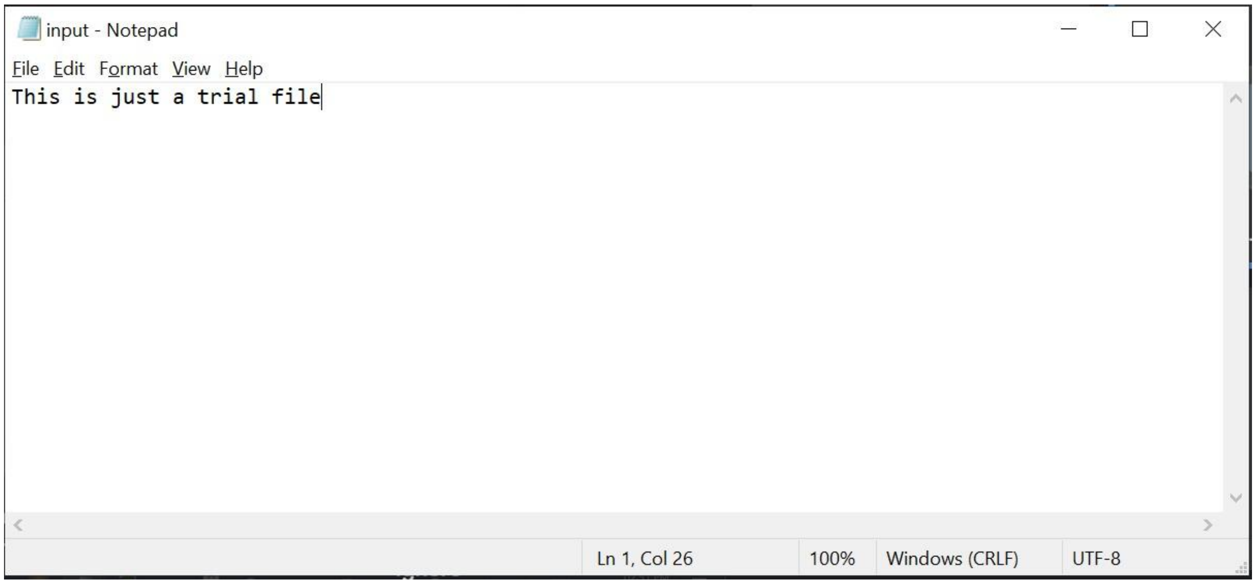
```
;read data from file piece by piece, xor it and save into output file
xorFile proc
    push cx
    push bx
    push ax
    push dx

    loop_loadData:
        ;DOS 2+ - READ - READ FROM FILE OR DEVICE
        mov dx, offset buffer        ;buffer for data from file
        mov cx, 200h                 ;sizeof buffer
        mov bx, ds:[file_in_desc]    ;file handler goes here
        mov ah, 3fh
        int 21h                  ;ax stores how many characters have been read

        jc fileReadingError      ;error during reading file

        cmp ax, 0                ;end of data?
        je readingEnd

        call xorBuffer
        call saveBufferToFile

        cmp ax, 200h             ;is the number of loaded data less than the total buffer size?
        jl readingEnd            ;yes, finish reading
        jmp loop_loadData        ;nope, read another portion of data

    readingEnd:

    pop dx
    pop ax
    pop bx
    pop cx
    ret
xorFile endp
```

8. Results

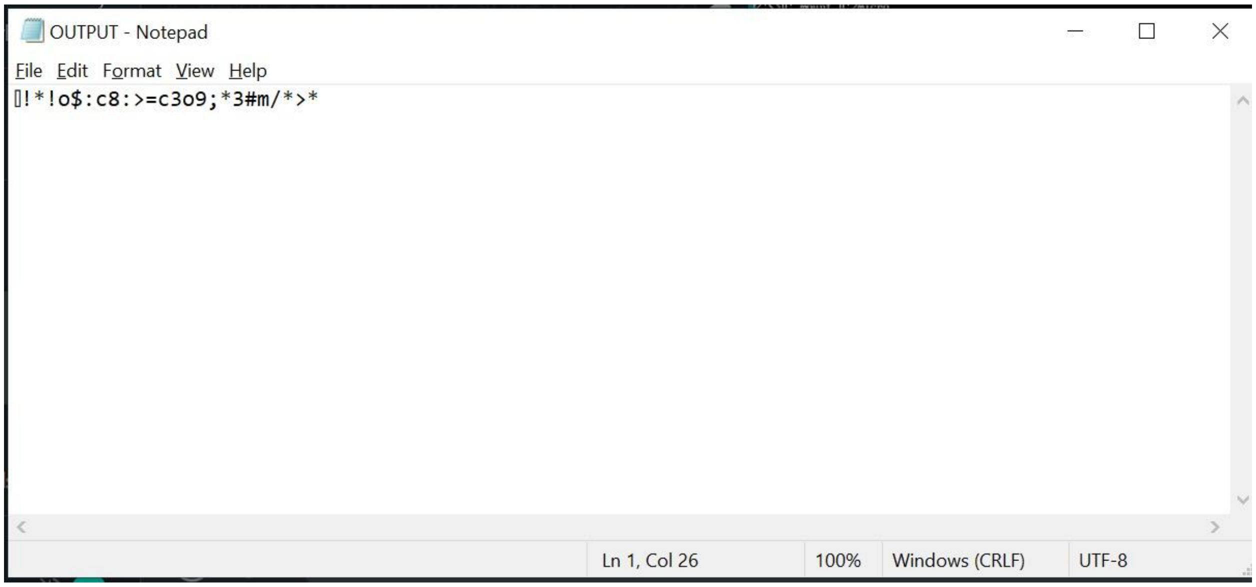
The results of our code were fast, and accurate. On executing the code, the result was an output file with the XORed output. The code runs accurately for multiple test cases.



```
DOSBox 0.74-3, Cpu speed:     3000 cycles, Fram...
 For supported shell commands type: HELP

 To adjust the emulated CPU speed, use ctrl-F11 and ctrl-F12.
 To activate the keymapper ctrl-F1.
 For more information read the README file in the DOSBox directory.

 HAVE FUN!
 The DOSBox Team http://www.dosbox.com
Z:\>SET BLASTER=A220 I7 D1 H5 T6

Z:\>D: mount D:/micro
Drive D does not exist!
You must mount it first. Type intro or intro mount for more information.

Z:\>MOUNT D D:\micro
Drive D is mounted as local directory D:\micro\

Z:\>D:

D:\>project.exe input.txt output.txt "MICRO"
Creating new input file...
Encryption successful!

D:\>S_
```



```
input - Notepad
File  Edit  Format  View  Help
This is just a trial file

Ln 1, Col 26    100%    Windows (CRLF)    UTF-8
```



```
▯!*!o$:c8:>=c3o9;*3#m/*>*
```

9. Conclusion

Over the course of the project, we have gained extensive knowledge of the use of the XOR cipher and its various applications in real world scenarios, and the importance of assembly language in data security and cryptography operations. We have learnt how to use the DOSBox and interface it with the emu8086 simulator, and understood how to perform basic XOR encryption operations. We have also developed familiarity with file handling in ALP using the 8086 microprocessor, and the results of our project were promising. Therefore, it is fair to say we have successfully completed this project, fulfilling the requirements of the course, to the best of our knowledge and capacity.

10. References

1. XOR Cipher Based Cryptography and Authentication with Hardware Chip | International Journal of Electronics Engineering, June - Dec 2018
2. Security Analysis of XOR Based Ciphered Image | Asian Journal of Computer Science and Technology, 2018
3. Cryptography Using Xor Cipher | Research Journal of Science and Technology, 2017
4. Arduino UNO and Android Based Digital Lock Using Combination of Vigenere Cipher and XOR Cipher | Journal of Physics: Conference Series, June, 2020

5. Securing Data at rest using Hill Cipher and XOR based operations | IEEE 2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA) - Coimbatore (2018.3.29-2018.3.31)
6. When an attacker meets a cipher-image in 2018 | Journal of Information Security and Applications, 2018

11. Appendix

```
data1 segment
        file_in             db      40h             dup(0) ;63 bytes + 1 for '0'
        file_out            db      40h    dup(0) ;63 bytes + 1 for '0'
        key                         db      80h             dup(0) ;127 bytes + 1 for
'0'
        file_in_desc   dw                   ?
        file_out_desc dw                    ?
        buffer              db      200h   dup(0)          ;512 bytes


        str_emptyArguments                  db              "Usage: prog.exe input_file
output_file ""encryption key""",10,13,"$"
        str_argumentsError          db              "Arguments are
invalid!",10,13,"$"
        str_exitError               db              "Program finished with error
:(",10,13,"$"
        str_fileOpenErrorIn         db              "Input file open error!",10,13,"$"
        str_fileOpenErrorOut        db              "Output file open
error!",10,13,"$"
        str_fileCreateNew           db              "Creating new input
file...",10,13,"$"
        str_fileOverwrite           db              "Overwrite output file? [T/N]: $"
        str_fileReadError           db              "Error during reading from input
file",10,13,"$"
        str_fileWriteError          db              "Error during writing to output
file",10,13,"$"
        str_success                         db              "Encryption
successful!",10,13,"$"
data1 ends

code1 segment
start1:
        ;init stack
        mov    sp, offset topstack
        mov    ax, seg topstack
        mov    ss, ax

        ;init data segment
        mov ax, data1
        mov ds, ax

        call readArgs
        call openFiles
        call xorFile
        call closeFiles

        mov dx, offset str_success
        call putStr

        jmp programExit

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        ;reads parameters, verify them
        readArgs proc
                push ax
                push dx
                push si
                push di

                xor     ax, ax
                mov     al, byte ptr es:[80h]  ;number of characters of cmd line - offset
80h
                cmp             al, 0                           ;check if any
arguments exists
                jne             parseArguments
                mov dx, offset str_emptyArguments
                call putStr
                call programExitError

                parseArguments:
                        mov si, 81h                     ;cmd line starts at 81h

                        mov di, offset file_in          ;point at the start of input file
name bufer
                        call parseOneArg                ;read the first file name
                        mov al, 0                               ;terminate string with 0
byte
```

```
                        mov ds:[di], al

                        mov di, offset file_out         ;point at the start of output file
name bufer
                        call parseOneArg
                        mov al, 0                               ;terminate string with 0
byte
                        mov ds:[di], al

                        mov di, offset key              ;point at the start of key bufer
                        call parseLastArg
                        mov al, 0                               ;terminate string with 0
byte
                        mov ds:[di], al

                pop di
                pop si
                pop dx
                pop ax
                ret
        readArgs endp

        ;parses one argument into [di], requires set di address
        parseOneArg proc
                push ax
                push cx

                call skipSpaces
                mov ch, 0
                loop_copy:
                        mov al, es:[si]
                        cmp al, 0dh                     ;if data ends here then error
                        je argumentError

                        cmp ch, 40h-1                   ;overflow protection :)
                        jge copyNext            ;-1 because of string termination with 0

                        cmp al, ' '                     ;space = jump to next argument
                        je copyNext                     ;when the arguments are correct,
the loop should end here
                        mov ds:[di], al
                        inc si
                        inc di
                        inc ch

                        jmp loop_copy

                copyNext:
                pop cx
                pop ax
                ret
        parseOneArg endp

        ;parses last arg, similar to parseOneArg but accepts spaces
        parseLastArg proc
                push ax
                push cx

                call skipSpaces
                mov ch, 0
                loopcopy:
                        mov al, es:[si]         ;get next character from cmd line
                        cmp al, 0dh                     ;no more data
                        je exitLoop

                        cmp al, '"'                     ;skip quote
                        je skipQuote

                        cmp ch, 80h-1                   ;overflow protection :)
                        jge exitLoop            ;-1 because of string termination with 0

                        mov ds:[di], al
                        inc di
                        skipQuote:                      ;if we skip a character we dont
increase di
                        inc si

                        jmp loopcopy

                exitLoop:
                pop cx
                pop ax
                ret
        parseLastArg endp

        ;opens input and output file, saves descriptors
```

```
	openFiles proc
		push ax
		push dx
		push cx

		;DOS 2+ - OPEN - OPEN EXISTING FILE
		mov dx, offset file_in
		mov al, 0			;read
		mov	ah, 3dh
		int 21h				;errors if CF

		jc fileOpenError	;jump if carry (CF), CF is set when error
occured

		mov word ptr ds:[file_in_desc], ax

		;DOS 2+ - OPEN - OPEN EXISTING FILE
		mov dx, offset file_out
		mov al, 1			;save
		mov	ah, 3dh
		int 21h				;errors if CF

		jc createNewFile	;jump if carry (CF), open error - file does not
exists

		mov dx, offset str_fileOverwrite
		call putStr

		;DOS 1+ - READ CHARACTER FROM STANDARD INPUT,
WITH ECHO
		mov ah, 01h
		int 21h

		call putNewLine

		cmp al, 't'
		je truncFile

		cmp al, 'y'
		je truncFile

		cmp al, 'T'
		je truncFile
```

```
        cmp al, 'Y'
        je truncFile

        jmp programExitError

        truncFile:
                ;DOS 2+ - CREAT - CREATE OR TRUNCATE FILE
                mov dx, offset file_out
                mov ah, 3ch
                mov cx, 1
                int 21h

                jmp saveFileDescriptor

        createNewFile:
                mov dx, offset str_fileCreateNew
                call putStr

                ;DOS 2+ - CREAT - CREATE OR TRUNCATE FILE
                mov dx, offset file_out
                mov ah, 3ch
                mov cx, 1
                int 21h

                jc fileOpenError

        saveFileDescriptor:
                mov ds:[file_out_desc], ax

        pop cx
        pop dx
        pop ax
        ret
openFiles endp

;read data from file piece by piece, xor it and save into output file
xorFile proc
        push cx
        push bx
        push ax
        push dx
```

```
		loop_loadData:
			;DOS 2+ - READ - READ FROM FILE OR DEVICE
			mov dx, offset buffer			;buffer for data from file
			mov cx, 200h				;sizeof buffer
			mov bx, ds:[file_in_desc]	;file handler goes here
			mov ah, 3fh
			int 21h					;ax stores how many
characters have been read

			jc fileReadingError		;error during reading file

			cmp ax, 0				;end of data?
			je readingEnd

			call xorBuffer
			call saveBufferToFile

			cmp ax, 200h			;is the number of loaded data less
than the total buffer size?
			jl readingEnd			;yes, finish reading
			jmp loop_loadData		;nope, read another portion of data

		readingEnd:

		pop dx
		pop ax
		pop bx
		pop cx
		ret
	xorFile endp


	;xors buffer with key, stores output in buffer
	xorBuffer proc
		push si
		push di
		push ax
		push bx
		push cx

		;ax stores how many characters have been read
		mov di, offset buffer;buffer iterator
		mov si, offset key		;key iterator
		mov cx, ax				;counter - how many bytes are left
to read

		loop_xor:
			cmp cx, 0
			je endXoring

			mov al, byte ptr ds:[di]	;we want only one byte
			xor al, byte ptr ds:[si]	;xor - result in al
			mov byte ptr ds:[di], al	;update buffer

			dec cx
			inc di
			inc si

			mov al, byte ptr ds:[si]
			cmp al, 0					;if we reached zero
in key, we need to start from the beginning of the key
			jne loop_xor

			mov si, offset key			;rewind key

			jmp loop_xor

		endXoring:
		pop si
		pop di
		pop ax
		pop bx
		pop cx
		ret
	xorBuffer endp


	;saves data from buffer to output file
	saveBufferToFile proc
		push dx
		push ax
		push bx
		push cx

		;ax stores how many characters have been read
		mov cx, ax
		mov ah, 40h
		mov bx, ds:[file_out_desc]
		mov dx, offset buffer
		int 21h

		jc fileSavingError		;if CF then error

		pop cx
		pop bx
		pop ax
		pop dx
		ret
	saveBufferToFile endp


	;closes files
	closeFiles proc
		push ax
		push bx

		mov ah, 3eh
		mov bx, ds:[file_in_desc]
		int 21h

		mov ah, 3eh
		mov bx, ds:[file_out_desc]
		int 21h

		pop bx
		pop ax
		ret
	closeFiles endp

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
	;move si until we encountered non-space character
	skipSpaces proc
		push ax

		loop_Chars:
			mov		al, es:[si]
			cmp		al, ' '
			jne		skipSpacesExit
			inc		si
```

```
                jmp     loop_Chars

        skipSpacesExit:
                pop             ax
                ret
skipSpaces endp


;print new line
putNewLine proc
        push dx

        mov dl, 10
        call putChar

        mov dl, 13
        call putChar

        pop dx
        ret
putNewLine endp


;print character from dl
putChar proc
        push ax

        ;DOS 1+ - WRITE CHARACTER TO STANDARD OUTPUT
        mov ah, 02h
        int 21h

        pop ax
        ret
putChar endp



;print ds:dx
putStr proc
        push ax

        ;DOS 1+ - WRITE STRING TO STANDARD OUTPUT
        xor al, al
        mov ah, 09h
```

```
		int 21h

		pop ax
		ret
	putStr endp


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
	fileReadingError:
		mov dx, offset str_fileReadError
		call putStr

		call programExitError

	fileSavingError:
		mov dx, offset str_fileWriteError
		call putStr

		call programExitError

	argumentError:
		mov dx, offset str_argumentsError
		call putStr

		mov dx, offset str_emptyArguments
		call putStr

		call programExitError

	fileOpenError:
		mov dx, offset str_fileOpenErrorIn
		call putStr

		call programExitError

	programExitError:
		mov dx, offset str_exitError
		call putStr

		;DOS 2+ - EXIT - TERMINATE WITH RETURN CODE
		mov al, 1		;exit code, error
		mov	ah, 4ch		;terminate program
		int	21h
```

```
          programExit:
                    ;DOS 2+ - EXIT - TERMINATE WITH RETURN CODE
                    mov al, 0                 ;exit code, 0 means success
                    mov     ah, 4ch           ;terminate program
                    int     21h

code1 ends


stack1 segment stack

          dw 200h dup(?)
topstack          dw ?

stack1 ends


end start1
```