SSEF2011 Research Project

CRYPTOGRAPHY IN THE REAL WORLD

by

Tan Wei Lin[1], Mak Chi En Michelle[1], Tok Genevieve[1]
[1]NUS High School, 20 Clementi Avenue 1, Singapore 129957

Teacher Advisors

Tan Rui Feng[1]
[1]NUS High School, 20 Clementi Avenue 1, Singapore 129957

External Mentors

Choy Jia Li Valerie[2]
[2]DSO National Laboratories, 20 Science Park Drive, Singapore 118230

Date: 2nd Nov 2010 – 31st Dec 2010

Summary

Encryption is used to ensure that only authorised parties may have access to secret information. In this project, we integrated the 256-bit version of the Advanced Encryption Standard cipher (AES-256) and the Secure Hash Algorithm (SHA-256) into a secure system with the capability to encrypt and decrypt files. The program, written in C programming language, has 4 different encryption modes. It reads a user-defined input file and prompts the user to key in a password which will then be used in the encryption of this file. The encrypted file will then be written to another user-defined output file.

Abstract

The aim of this project is to implement a secure file encryption/decryption cryptosystem using C Programming language that takes a user-input password as the main parameter to generate the

encryption/decryption key. The cryptosystem is designed such that the ciphertext will be unintelligible to any adversary who does not possess the password. This program utilises the 256-bit Secure Hash Algorithm (SHA-256) to hash the password entered by the user and the hash result is then used as the cipher key. The 256-bit Advanced Encryption Standard (AES-256) is the cipher that carries out the encryption and decryption processes. The plaintext is first padded so that its total length is a multiple of 128 bits. The padded plaintext is then broken up into 128-bit blocks to be encrypted. The result of the encryption is written to a ciphertext file. As AES is a symmetric-key cipher, a legitimate user will need to use the same password to decrypt the file and recover the original plaintext.

Introduction

The main objective of encryption is to maintain confidentiality, that is, to ensure that all information transferred is kept secret from everyone except from those who are meant to view it. Symmetric-key encryption requires a key or password known only to authorised parties and the same key is used for decryption. Ideally, encrypted information sent across an insecure channel will be incomprehensible to a malicious adversary who does not have the key. This project utilises the cipher AES-256, a symmetric-key block cipher, and the cryptographic hash function SHA-256 to carry out encryption and decryption, with the main objective being the confidentiality in information transfer.

To make use of modular programming, a mini-crypto library was implemented. It contains several functions handling different operations. This includes implementations of AES-256 and SHA-256, as well as those for handling user input, error checking and program action. The main function is used to control how the rest of the functions interact with one another. There are four encryption modes that the user is able to select from – Cipher Block Chaining (CBC), Counter (CTR), Cipher Feedback (CFB) and Output Feedback (OFB). The final program allows the user to enter a password, select an Initialisation Vector (IV) from seven preset IVs, choose a mode of operation, and finally encrypts or decrypts the input it reads from a user-selected file. The output is then saved in a different, user-specified file.

A few assumptions were made with regards to the security of the system. Firstly, it was assumed that the AES-256 and SHA-256 algorithms are secure and their implementations are performing their intended task. Secondly, the modularisation of the respective functions is taken to have been done correctly and that each function performs its task as intended. It is also assumed that the user of the program knows how to use the system correctly. This includes him choosing a strong password for the encryption and having basic knowledge of the strengths and weaknesses of the four modes of operation of the cipher. The IVs are assumed to be randomly generated.

Furthermore, if the encryption and decryption of a file are carried out by two individuals, it is assumed that both the authorised sender and receiver of the ciphertext know about or agree on the password and IV beforehand. This is a reasonable assumption if the IV is transmitted together with the

encrypted file or if it was agreed during the key exchange phase. This key could also have been agreed upon offline, transmitted to the other party via public key encryption like RSA, or exchanged utilising an authentication protocol. We also assume that the user will choose a different combination of password and IV each time he encrypts a file to prevent inadvertently leaking out critical information about the plaintext.

Background

Encryption is the process of transforming original, understandable information (known as plaintext) into incomprehensible form (known as ciphertext) through a mathematical process. Decryption is the reverse of encryption – transforming the ciphertext into plaintext. Usually, both processes require the use of some secret information, known as the key. Symmetric-key algorithms utilise the same key for both encryption and decryption. The finite set of keys, plaintexts, ciphertexts and the encryption and decryption functions make up the cryptosystem.

In this project, we use AES, a symmetric-key encryption system that comprises of a family of block ciphers. They have a block size of 128 bits and key sizes of 128, 192, and 256 bits. For our program, we only made use of AES-256 as our encryption cipher.

The ciphers in AES are based on the design principle known as a Substitution-Permutation Network (SPN). SPN is a series of related mathematical operations that are used in many block ciphers including AES. A SPN takes in one block of the plaintext and the cipher key, and applies substitution boxes (S-boxes) and permutation boxes (P-boxes) to the input in each round. AES-256 uses 14 rounds. Each round in AES consists of 4 steps: SubBytes, ShiftRows, MixColumns and AddRoundKey. In the SubBytes step, each byte in the array is replaced by a corresponding byte-substitute as specified by the AES S-box. Following that, ShiftRows is executed, where the rows of the state are shifted by an offset x, and $x = row\ number - 1$. Thus, the first row is left untouched, the second row is shifted by an offset of 1, and the third and fourth rows are shifted by an offset of 2 and 3 respectively. The next step is MixColumns, where the 4 bytes in the columns of the array are multiplied with the appropriate matrix over the Galois Field $GF(2^8)$. The final step, known as AddRoundKey, is where bitwise XOR is performed on the subkey with the state. These four steps are repeated for 14 rounds in AES-256. However, the last round will not include the MixColumns step.

A cryptographic hash function takes in a data input of any length and outputs a bit string of fixed length. The output of the function is known as the hash. In this project, we used SHA-256 which has an output of 256 bits. The function is designed to run one-way so that it will be difficult to find collisions, that is, different passwords hashing to the same output.

Block ciphers are used in a mode of operation that breaks up a plaintext message into blocks of a fixed length before encrypting each block one at a time. In most modes of encryption, an

Initialisation Vector (IV) is required as part of the encryption of the first block of plaintext. It is a block of bits that is randomly generated. It is used to prevent repetitions in the ciphertext that would cause it to be more susceptible to an attack as well as to make it unique from other keystreams produced by the same cipher key. A nonce is a random or pseudo random number that can serve the same function as an IV. However, it can be used only once.

The four modes of block cipher operation used in the program were CBC, CFB, CTR and OFB mode. In Cipher-block chaining (CBC) mode, the first block of the plaintext is first read, then XORed with the IV. Each subsequent block of plaintext is XORed with the previous ciphertext before being encrypted with AES-256. During decryption, the first ciphertext block is decrypted with the decryption function of AES. The output will then be XORed with the IV, producing the first plaintext block. Each subsequent ciphertext block will undergo a similar process. However, the output of decryption will be XORed with the previous ciphertext block to produce the plaintext block.

In Cipher Feedback (CFB) mode, the IV is the first input for RijndaelEncrypt. The output is then XORed with the plaintext to produce the ciphertext. The ciphertext block is used as input for RijndaelEncrypt for the next block. During decryption, the output is XORed with the ciphertext to produce the plaintext. Like encryption, the ciphertext block is used as input for RijndaelEncrypt for the decryption of the next ciphertext block.

In Counter (CTR) mode, a counter is encrypted using RijndaelEncrypt to produce the output block. The output block is then XORed with the plaintext block, producing the final encrypted text known as the ciphertext. The starting value of the counter is taken to be the IV. With each block, the counter increases by one to a maximum of $2^{128}-1$ times before cycling back to zero. Decryption is done in a similar manner, with the output blocks being XORed with the ciphertext blocks to produce the plaintext blocks.

Output Feedback (OFB) mode, like CFB mode, utilises the IV chosen as the input for RijndaelEncrypt. For the first plaintext block, the chosen IV is encrypted with RijndaelEncrypt. The output is then XORed with the plaintext to produce the ciphertext block. The process will continue for p number of times, with the output of the previous encryption being used as the input of the cipher for the next block. This process is similar for decryption, with the output being XORed with the ciphertext.

There are existing programs that provide encryption and decryption of files available in the market. However, there is no detailed information about how these programs work, especially with regards to the hash function used. Thus, it is unknown whether the information is truly confidential after encryption. As such, there is a need to create a secure file encryption cryptosystem.

Specification and Design

In this system, the user is allowed to choose between encrypt or decrypt, enter a password of his choice with a length of no more than 20 characters, select between seven IVs and choose a particular block cipher mode of operation in that order. The program allows for the selection of CBC, CFB, CTR or OFB mode for encryption and decryption and checks at each input step to ensure that user input is valid. The program first checks if the user had three command line arguments in the following format: <program executable><space><source file><space><output file>

Figure 1.

It also checks if the source file can be opened. If neither requirement is fulfilled, the user will be alerted and can re-run the program immediately. The user will then choose between encryption and decryption. His choice will determine if an additional password verification step is needed before the password is hashed. Before encryption or decryption can take place, the IV which is used in the functions must be selected. Lastly, the user selects the mode of operation he wants.

The program was developed using both Dev C++ and Microsoft Visual C++ IDEs. The original program involved having all codes except for the cipher and hash in the main function. However, during the course of re-development, we employed modular programming instead, that is, we placed similar functions into separate .c files and called them from the main. This has enabled us to debug the program with greater ease.

The code was compartmentalised based on the functions it was supposed to carry out. These functions include the password input, password hashing, error checking mechanisms, and the different encryption and decryption codes for the four different modes of block cipher operation. Functions that had similar purposes were placed together in a file. For example, the functions encryptCBC(), encryptCFB(), encryptCTR() and encryptOFB() were placed in encrypt.c. Figure 2 shows a list of files used in the program.

action.c clic.c cmp.c compare.c decrypt.c encrypt.c hashPword.c inputPword.c iv.c main.c
rijndael-alg-fast.c rijndael-alg-fast.h sha256.c sha256.h

Figure 2.

Implementation

The program begins with an input check using the function, clic, to ensure that the user has input exactly three command line arguments in the format of Figure 1. If the user input contains more than or less than three command line arguments, an error message will appear and the program will automatically exit.

If there is no error, the program will progress to ask the user if he would like to encrypt or decrypt his file.

If he were to select 'encrypt', the program will ask for a password input, followed by a password verification step. Unless the two passwords match, the user will be prompted to input his password again. The program will then proceed to hash the user-input password with SHA-256 and

store the hash in cipherKey[32]. Following this, the user is asked to choose his mode of block cipher operation.

If the user chooses to decrypt a file, there will not be an additional password verification step. Instead, after typing the password, the program will proceed to ask the user for his choice of mode of block cipher operation.

The user would be given four modes to choose from, namely CBC, CTR, OFB, or CFB mode. The program accepts any permutation of capitals and non-capitals letters for the name of the four modes. The functions compare_CBC, compare_CFB, compare_CTR and compare_OFB check if the letters the user entered correspond to any of the modes of operation. If they do not, the user will have to re-enter his choice until he keys in an acceptable input that is recognised and received by the program. These functions also proceed to call the corresponding action function, that is, compare_CBC would call actionCBC.

```
printf("\nWhich mode do you wish to use to encrypt or decrypt your file?
         \nEnter 'CBC', 'CFB', 'CTR' or 'OFB' to choose encrypt/decrypt mode.\n");
scanf ("%s", c);
 for (;;)
 {
        if (strlen(c)!=3 || (!compare_CBC(c) && !compare_CFB(c) && !compare_CTR(c)
          && !compare_OFB(c))){
              printf ("\nERROR! Enter 'CBC', 'CFB', 'CTR' or 'OFB' to
                      choose encrypt/decrypt mode.\n");
              printf ("Which mode do you wish to use to encrypt or decrypt
                       your file?\nEnter 'CBC', 'CFB', 'CTR' or 'OFB'
                       to choose encrypt/decrypt mode.\n");
              fflush (stdin);
              scanf("%s", c);
        }
        else {
              fflush (stdin);
              break;
        }
}
```

The action function checks whether the user entered ‘e’ or‘d’ at the beginning and calls the appropriate function. If the user had selected CBC mode and entered ‘e’ to encrypt, actionCBC would call encryptCBC.

```
void actionCBC (char en[10], u8 IV[16], u8 cipherKey[32], int keyBits,
                         FILE *infp, FILE *outfp){
       if (en[0] == 'E' || en[0] == 'e')
             encryptCBC (IV, cipherKey, keyBits, infp, outfp);
       else if (en[0] == 'D' || en[0] == 'd')
            decryptCBC (IV, cipherKey, keyBits, infp, outfp);
}
```

The program will then proceed to carry out the encryption/decryption of the user-input file utilising AES-256. The contents of cipherKey[32] is used as the key in this encryption/decryption step. The program will read the plaintext (resp. ciphertext) from the user-specified input file. The output of the encryption (resp. decryption) step, known as the ciphertext (resp. plaintext) will be written to the user-specified output file. The program will return a confirmation message and exit once this has been completed.

One of the challenges we faced in implementing the system was to make the code as concise as possible. Initially, the code was not modularised and considerably longer. It took some time for the program to be properly modularised as there were some problems in determining the input of each function while we were doing the modularisation. Another major challenge was the implementation of CTR mode as there were difficulties in getting the counter to increment properly.

Results and Evaluation

The program works as intended and allows a user to select a password of not more than 20 characters and encrypt a file of their choice in CBC, CFB, CTR or OFB mode. They are also able to decrypt the ciphertext file using the same password that they have selected. In addition, they are able to select from seven different preset IVs within the program to encrypt their file with, hence providing additional security. The program prevents the user from keying wrong inputs by the inclusion of infinite loops, the main error-catching mechanism in the program. The following table summarises the similarities and differences between the four modes of block cipher operation.

	CBC	**CFB**	**CTR**	**OFB**
Decryption Function	Rijndael-Decrypt	Rijndael-Encrypt	Rijndael-Encrypt	Rijndael-Encrypt
Chaining Dependencies	Yes	Yes	No	Yes
Error Propagation	Single bit error on n^{th} ciphertext block may flip corresponding bit on $(n+1)^{th}$ plaintext block, but affect the n^{th} plaintext block significantly.	Single bit error on n^{th} ciphertext block may change corresponding bit on n^{th} plaintext block, but affect the $(n+1)^{th}$ plaintext block significantly.	Single bit error on n^{th} ciphertext block is localized to corresponding bit in the n^{th} plaintext block after decryption	Single bit error on n^{th} ciphertext block is localized to corresponding bit in the n^{th} plaintext block after decryption
Error Recovery	Will recover 2 blocks later	Will recover 2 blocks later	Will recover 1 block later	Will recover 1 block later
Parallelisability of encryption	Not parallelisable	Not parallelisable	Parallelisable	Not parallelisable
Parallelisability of decryption	Parallelisable	Parallelisable	Parallelisable	Not parallelisable
Wrong IV Selection	Only first 16 bytes of decrypted ciphertext will be unintelligible.	Only first 16 bytes of decrypted ciphertext will be unintelligible.	The entire decrypted ciphertext will be unintelligible	The entire decrypted ciphertext will be unintelligible

All modes except CBC mode use the RijndaelEncrypt function in the decryption process. In certain constrained environments, it is better that decryption requires just RijndaelEncrypt so that the same code can be used for both encryption and decryption, thus saving on code space. On the other hand, all modes except CTR mode have chaining dependencies. Ciphertext j j will depend on plaintext j and all the previous plaintext blocks. As such, a one bit change in a plaintext block will

affect all subsequent ciphertext blocks. Thus, it will be better not to utilise CTR mode if the user is concerned with ensuring that the same plaintext block will not result in the same ciphertext block.

Parallelisability refers to the property of the mode of operation where different blocks of plaintext (resp. ciphertext) can be encrypted (resp. decrypted) at the same time. CTR mode for both encryption and decryption is parallelisable since the counter can be encrypted in a pre-processing phase and the result stored before the plaintext/ciphertext is received. Thus, it is faster when encrypting a plaintext as compared to the other block cipher modes of operation and it is more advisable to select that mode of block cipher operation if the user desires to encrypt a large file in a short span of time.

When operating in CTR or OFB mode, using the wrong IV would render an attacker unable to decrypt the entire file to the original even if he has the correct password. Thus, if the user is concerned with ensuring that the security of the file is not compromised even with the attacker knowing the password, CTR or OFB mode will be better choices. However, since there are only seven IVs provided in this program, the attacker with the correct password would be able to decrypt the file after multiple tries with all the available IVs.

In terms of error propagation, any error in one block will affect all four modes of operation by a maximum of two blocks. However, there is no ‘best’ mode of block cipher operation for such a case. All four modes are able to recover from errors in their ciphertext blocks. This property is important when faced with random errors that might occur.

Having different IVs can enhance the security of the system as different keystreams can be generated with the same password using different IVs.

AES is much faster than asymmetric-key ciphers of equivalent security where the encryption and decryption keys are different. However, it also depends on the mode of operation. For instance, when implemented in CBC mode, AES may be slower due to CBC mode’s inability to process blocks of plaintext in parallel.

The program requires prior knowledge of command line arguments to operate. Thus, it is not as user friendly as we would like it to be. Although a help file is included, the implementation of a Graphical User Interface (GUI) would make the program more user-friendly.

This program was written using the C programming language. We feel that it is appropriate as C is comparatively more function-oriented as compared to object-oriented programming such as Java or C#. As such, writing the program is easier and the code is more streamlined.

Future Works

Our program is not yet perfect and more functions could be added. These include giving the users the option to delete source files and encryption of entire folders instead of individual files. We

can also allow users the choice to run the program via DOS windows, visual interface, or merely call up the functions.

Also, we can generate IVs that are derived from a cryptographically secure pseudo-random number generator (CSPRNG). The IVs in our current program were generated by randomly typing in valid hexadecimal input into the program. This is not a good method as there might be a discernible mathematical pattern in the way the IVs are generated. However, using a CSPRNG will be difficult to implement as the user has to decrypt the file with the same IV. To mitigate that problem, we can first generate random IVs then store them into the program database before compilation.

Conclusion

In this project, a secure system utilising both AES-256 and SHA-256 has been successfully implemented using C. The program successfully encrypts and decrypts all types of files. Due to the individual weaknesses of both SHA-256 and AES, we cannot conclude with certainty that the encrypted files are indeed of a high level of security. The strength of the system is limited but we believe it to be sufficient for this time.

Acknowledgements

We would like to thank our mentor, Ms Choy, for helping us with this project and providing invaluable insights into the world of programming. We would also like to thank our substitute mentor, Mr Teo, and our teacher mentor, Mr Tan, for their help.

Statement of Contribution

All student authors contributed equally to this project. Ms Choy helped us in debugging the program as well as vetting our report. Mr Teo taught us some programming skills as well as aided in the modularising of the program. Mr. Tan provided valuable insights into doing the report and poster.

References

1. Kantarcioglu, Murat. (n.d). *Modes of Operation.* Retrieved from www.utdallas.edu/~muratk/courses/crypto07_files/modes.pdf
2. The University of Sydney. (2004). *Math3024 Elementary Cryptography and Protocols, Exercises and Solutions for Week 7*. Retrived from http://echidna.maths.usyd.edu.au/kohel/tch/MATH3024/Tutorials/Solutions/tutorial07.pdf
3. National Institute of Standards and Technology (NIST). (October 20, 2000). *Report on the Symmetric Key Block Cipher Modes of Operation Workshop*. Retrived from http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/workshop1/workshop-report.pdf
4. Lars R. Knudsen. (n.d.). *Block Cipher Chaining Modes of Operation.* Retrived from http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/workshop1/presentations/slides-knudsen-modesp.pdf
5. Menezes, P. van Oorschot, S. Vanstone. (1996). *Handbook on Applied Cryptography.* Retrived from http://www.slideshare.net/nathanurag/chap7-presentation
6. *mcrypt(1) - Linux man page.* (n.d.). Retrived from http://linux.die.net/man/1/mcrypt

Appendix

Full Source Codes:

Main program

```
/* Main program v2.10*/

#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <string.h>

#include "sha256.h"
#include "rijndael-alg-fst.h"

int inputPword (u8 pword[20]);
int checkPword (u8 pword[20]);
int checkED (u8 pword[20], char en[10]);
void chooseIV (u8 IV[16], int ivchoose);
char clic (int argc, char *argv[]);
void hashPword (u8 pword[20], u8 cipherKey[32]);
int compare_CBC (char *str);
int compare_CFB (char *str);
int compare_CTR (char *str);
int compare_OFB (char *str);
void actionCBC (char en[10], u8 IV[16], u8 cipherKey[32], int keyBits, FILE *infp,
                FILE *outfp);
void actionCFB (char en[10], u8 IV[16], u8 cipherKey[32], int keyBits, FILE *infp,
                FILE *outfp);
void actionCTR (char en[10], u8 IV[16], u8 cipherKey[32], int keyBits, FILE *infp,
                FILE *outfp);
void actionOFB (char en[10], u8 IV[16], u8 cipherKey[32], int keyBits, FILE *infp,
                FILE *outfp);
void encryptCBC (u8 IV[16], u8 cipherKey[32], int keyBits, FILE *infp, FILE *outfp);
void decryptCBC (u8 IV[16], u8 cipherKey[32], int keyBits, FILE *infp, FILE *outfp);
void encryptCFB (u8 IV[16], u8 cipherKey[32], int keyBits, FILE *infp, FILE *outfp);
void decryptCFB (u8 IV[16], u8 cipherKey[32], int keyBits, FILE *infp, FILE *outfp);
void encryptCTR (u8 IV[16], u8 cipherKey[32], int keyBits, FILE *infp, FILE *outfp);
void decryptCTR (u8 IV[16], u8 cipherKey[32], int keyBits, FILE *infp, FILE *outfp);
void encryptOFB (u8 IV[16], u8 cipherKey[32], int keyBits, FILE *infp, FILE *outfp);
void decryptOFB (u8 IV[16], u8 cipherKey[32], int keyBits, FILE *infp, FILE *outfp);

int main (int argc, char *argv[] )

{
        int ivchoose, keyBits=256;
        char en[10] = {0}, c[10] = {0};
        u8 IV[16], cipherKey[32], cipherKey1[32], pword[20] = {0}, pword1[20]= {0};
        FILE *infp, *outfp;

        if (clic (argc, argv))
        {
                printf ("\nPlease run the program again.\n");
                system ("pause");
                return 0;
        }

        printf ("Files selected: <%s><%s>\n", argv[1], argv[2]);

        printf ("\nDo you want to encrypt or decrypt your file?\n
                 Enter 'E' to encrypt and 'D' to decrypt your file:\n");
        scanf ("%s", en);

        checkED (pword, en);
        /* Checks whether the user wishes to encrypt or decrypt their file. */
```

```
        hashPword (pword, cipherKey);
        /* Hashes the password to generate the cipherkey. */
        chooseIV (IV, ivchoose);
        /* Selects the Initialisation Vector. */

        printf("\nWhich mode do you wish to use to encrypt or decrypt your file?
                \nEnter 'CBC', 'CFB', 'CTR' or 'OFB' to choose encrypt/decrypt mode.\n");

        scanf ("%s", c);

        for (;;)
        {
                if (strlen(c)!=3 || (!compare_CBC(c) && !compare_CFB(c) && !compare_CTR(c)
                  && !compare_OFB(c)))
                {
                        printf ("\nERROR! Enter 'CBC', 'CFB', 'CTR' or 'OFB' to
                                choose encrypt/decrypt mode.\n");
                        printf ("Which mode do you wish to use to encrypt or decrypt
                                 your file?\nEnter 'CBC', 'CFB', 'CTR' or 'OFB'
                                 to choose encrypt/decrypt mode.\n");
                        fflush (stdin);
                        scanf("%s", c);
                }

                else
                {
                        fflush (stdin);
                        break;
                }
        }

        infp = fopen (argv[1], "rb");
        outfp = fopen (argv[2], "wb");

        if (compare_CBC (c))
                  actionCBC (en, IV, cipherKey, keyBits, infp, outfp);
    /* Either encrypts or decrypts the file in CBC mode based on user input. */

        if (compare_CFB (c))
                  actionCFB (en, IV, cipherKey, keyBits, infp, outfp);
    /* Either encrypts or decrypts the file in CFB mode based on user input. */

        if (compare_CTR (c))
                  actionCTR (en, IV, cipherKey, keyBits, infp, outfp);
    /* Either encrypts or decrypts the file in CTR mode based on user input. */


        if (compare_OFB (c))
                  actionOFB (en, IV, cipherKey, keyBits, infp, outfp);
    /* Either encrypts or decrypts the file in OFB mode based on user input. */

        printf ("\n");
        printf ("Output printed to file <%s>.", argv[2]);
        printf ("\n");

        fclose (infp);
        fclose (outfp);

        system ("pause");

        return 0;

}
```

Error Checking

```
/* File Name: clic.c
 * Checks for valid file input
 */
```

```
#include <assert.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <string.h>

char clic (int argc, char *argv[])
{
        FILE *infp, *outfp;

        if (argc != 3)
        {
                printf ("Place in cmd.exe:
                        <this file><space><sourcefile><space><destinationfile>\n");
                return 1;
        }

        infp = fopen (argv[1], "rb");
        outfp = fopen (argv[2], "wb");

        if (infp == NULL)
        {
                printf ("\nCan't open source file.\n");
                return 1;
        }

        return 0;
}


/* File Name: checkED.c
 * This function checks whether the user inputs 'e' or 'd' to either encrypt or
 * decrypt their files.
 * Also checks if passwords are similar if 'e' is the user input
 */

#include <assert.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

#include "rijndael-alg-fst.h"

void checkED (u8 pword[20], char en[10])
{
        u8 pword1[20] = {0};

        for (;;)
        {
                if (en[0] != 'E' && en[0] != 'e' && en[0] != 'D' && en[0] != 'd' ||
                    strlen(en) !=1)
                {
                        printf ("\nERROR! Enter 'E' to encrypt or 'D' to decrypt.\n");
                        printf ("Do you want to encrypt or decrypt your file?\nEnter 'E' to
                                 encrypt and 'D' to decrypt your file:\n");
                        fflush (stdin);
                        scanf ("%s", en);
                }

                else
                {
                        break;
                        fflush (stdin);
                }
        }

        if (en[0] == 'E' || en[0] == 'e')
        {
```

```
                for (;;)
                {
                        inputPword (pword);
                        checkPword (pword1);
                        if (memcmp (pword, pword1, 20) != 0)
                                printf ("ERROR! Your passwords do not match. \nPlease try
                                        again.\n\n");

                        else break;
                }
        }

        else if (en[0] == 'D' || en[0] == 'd')
                inputPword (pword);
}


/* File Name: compareMode.c
 * This file contains the block cipher mode selection.
 * It checks if the user input for the choice of block cipher mode
 * of operation is correct
 */

#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <string.h>

int compare_C (char c)
{
        if (c == 'C' || c == 'c') return 1;
        else return 0;
}

int compare_B (char b)
{
        if (b == 'B' || b == 'b') return 1;
        else return 0;
}

int compare_F (char f)
{
        if (f == 'F' || f == 'f') return 1;
        else return 0;
}

int compare_O (char o)
{
        if (o == 'O' || o == 'o') return 1;
        else return 0;
}

int compare_R (char r)
{
        if (r == 'R' || r == 'r') return 1;
        else return 0;
}

int compare_T (char t)
{
        if (t == 'T' || t == 't') return 1;
        else return 0;
}


int compare_CBC (char *str)
{
```

```
        if (compare_C (str[0]) && compare_B (str[1]) && compare_C (str[2])) return 1;

        else return 0;
}

int compare_CFB (char *str)
{
        if (compare_C (str[0]) && compare_F (str[1]) && compare_B (str[2])) return 1;

        else return 0;
}

int compare_CTR (char *str)
{
        if (compare_C (str[0]) && compare_T (str[1]) && compare_R (str[2])) return 1;

        else return 0;
}

int compare_OFB (char *str)
{
        if (compare_O (str[0]) && compare_F (str[1]) && compare_B (str[2])) return 1;

        else return 0;
}
```

Password Input

```
/* File Name: Pword.c
 * This file contains the functions for entering and hashing the password.
 */

#include <assert.h>
#include <ctype.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <string.h>

#include "sha256.h"
#include "rijndael-alg-fst.h"

int inputPword(u8 pword[20])
{
        int ch, k = 0;

        memset (pword, 0, 20);

        puts ("\nEnter your password of 20 characters max then press <Enter>.");

        fflush (stdout);

        while ((ch = getch()) != EOF
                && ch != '\n'
                && ch != '\r')
        {
                if (k < 20||ch == '\b')
                {
                        if (ch == '\b' && k > 0)
                        {
                                printf("\b \b");           //Backspace
                                fflush (stdout);
                                --k;
                                pword[k] = '\0';
                        }

                        else if (isalnum (ch))
```

```
                        {
                                printf("*");
                              //Prints '*' instead of the character entered.
                                pword[k++] = (char)ch;
                        }
                }
        }

        pword[k] = '\0';
        printf ("\n");

        return k;
}

int checkPword(u8 pword1[20])
{
        int ch, k = 0;

        memset (pword1, 0, 20);

        puts ("Please reenter your password, then press <Enter>.");
        fflush (stdout);

        while ((ch = getch()) != EOF
                && ch != '\n'
                && ch != '\r')
        {
                if (k < 20||ch == '\b')
                {
                        if (ch == '\b' && k > 0)
                        {
                                printf ("\b \b");
                                fflush (stdout);
                                --k;
                                pword1[k] = '\0';
                        }

                        else if (isalnum (ch))
                        {
                                printf("*");
                                pword1[k++] = (char)ch;
                        }
                }
        }

        pword1[k] = '\0';
        printf("\n");

        return k;
}

void hashPword (u8 pword[20], u8 cipherKey[32])
{
        int plen;

        memset (cipherKey, 0, 32);
        plen = strlen (pword);
        sha256_context ctx;

        sha256_starts (&ctx);
        sha256_update (&ctx, pword, plen);
        sha256_finish (&ctx, cipherKey);

}
```

IV Selection

```
/* File Name: iv.c
 * This file contains the functions for Initialisation Vector selection.
 */

#include <stdio.h>
#include <conio.h>

#include "sha256.h"
#include "rijndael-alg-fst.h"

void allocateIV(u8 IV[16], u8 iv[16]);

u8 iv1[16] = {0x00, 0x01, 0x02, 0x03,
              0x04, 0x05, 0x06, 0x07,
              0x08, 0x09, 0x0a, 0x0b,
              0x0c, 0x0d, 0x0e, 0x0f,
             };

u8 iv2[16] = {0x1d, 0xe8, 0xe1, 0x2f,
              0x23, 0xe4, 0x9d, 0xd4,
              0xa7, 0xff, 0xc5, 0xf4,
              0xe9, 0xe8, 0xfc, 0xa5,
             };

u8 iv3[16] = {0xdd, 0xdf, 0x2d, 0x3f,
              0x0d, 0x94, 0x9e, 0x3a,
              0x2c, 0xf9, 0xf4, 0xf4,
              0xdc, 0xda, 0xf1, 0xd3,
             };

u8 iv4[16] = {0x2d, 0x58, 0x35, 0x6f,
              0x3d, 0x34, 0x0d, 0x3a,
              0x2c, 0xf9, 0x8e, 0x10,
              0x19, 0xc7, 0xf0, 0x3b,
             };

u8 iv5[16] = {0x5a, 0xd6, 0x2a, 0x5f,
              0x88, 0x55, 0x42, 0x4a,
              0xb7, 0xd4, 0xae, 0x0f,
              0x3d, 0xc3, 0xa7, 0xab,
             };

u8 iv6[16] = {0x53, 0xa6, 0xb0, 0x0d,
              0x25, 0x93, 0x21, 0x4f,
              0xf6, 0xf7, 0xf9, 0x2d,
              0xca, 0xe0, 0xdd, 0xaa,
             };

u8 iv7[16] = {0x45, 0x3d, 0x8e, 0x1b,
              0x7e, 0x1f, 0x69, 0x93,
              0xe4, 0xda, 0x9e, 0x1f,
              0xe5, 0x11, 0x33, 0xc7,
             };


void chooseIV(u8 IV[16], int ivchoose)
{
        printf("\nEnter IV (A number from 1-7): ");

        for (;;)
        {

                if (scanf("%d", &ivchoose) != 1)
                {
                        printf("\nIV should be a number from 1 to 7.");

                        printf("\nEnter IV (A number from 1-7): ");
```

```
                        fflush(stdin);
                }

                else if (ivchoose != 1 && ivchoose != 2 && ivchoose != 3 && ivchoose != 4
                         &&ivchoose != 5 && ivchoose != 6 && ivchoose != 7)
                {
                        printf("\nIV should be a number from 1 to 7.\n");
                        printf("Enter IV (A number from 1-7): ");
                        fflush(stdin);
                }

                else break;

        }

        switch (ivchoose)
        {
               case 1: allocateIV( IV, iv1 ); break;
               case 2: allocateIV( IV, iv2 ); break;
               case 3: allocateIV( IV, iv3 ); break;
               case 4: allocateIV( IV, iv4 ); break;
               case 5: allocateIV( IV, iv5 ); break;
               case 6: allocateIV( IV, iv6 ); break;
               case 7: allocateIV( IV, iv7 ); break;
       }
}

void allocateIV(u8 IV[16], u8 iv[16])
{
        int i;

        for (i=0; i<16; i++)
               IV[i] = iv[i];
}
```

Action Functions

```
/* File Name: action.c
 * This file contains the action functions for the various modes that either
 * encrypts or decrypts based on user input.
 * File Name: action.c
 */

#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <string.h>

#include "rijndael-alg-fst.h"

void encryptCBC (u8 IV[16], u8 cipherKey[32], int keyBits, FILE *infp, FILE *outfp);
void decryptCBC (u8 IV[16], u8 cipherKey[32], int keyBits, FILE *infp, FILE *outfp);
void encryptCFB (u8 IV[16], u8 cipherKey[32], int keyBits, FILE *infp, FILE *outfp);
void decryptCFB (u8 IV[16], u8 cipherKey[32], int keyBits, FILE *infp, FILE *outfp);
void encryptCTR (u8 IV[16], u8 cipherKey[32], int keyBits, FILE *infp, FILE *outfp);
void decryptCTR (u8 IV[16], u8 cipherKey[32], int keyBits, FILE *infp, FILE *outfp);
void encryptOFB (u8 IV[16], u8 cipherKey[32], int keyBits, FILE *infp, FILE *outfp);
void decryptOFB (u8 IV[16], u8 cipherKey[32], int keyBits, FILE *infp, FILE *outfp);

void actionCBC (char en[10], u8 IV[16], u8 cipherKey[32], int keyBits, FILE *infp,
               FILE *outfp)
{
        if (en[0] == 'E' || en[0] == 'e')
                encryptCBC (IV, cipherKey, keyBits, infp, outfp);

        else if (en[0] == 'D' || en[0] == 'd')
                decryptCBC (IV, cipherKey, keyBits, infp, outfp);
```

```
}

void actionCFB (char en[10], u8 IV[16], u8 cipherKey[32], int keyBits, FILE *infp,
              FILE *outfp)
{
        if (en[0] == 'E' || en[0] == 'e')
                encryptCFB (IV, cipherKey, keyBits, infp, outfp);

        else if (en[0] == 'D' || en[0] == 'd')
                decryptCFB (IV, cipherKey, keyBits, infp, outfp);
}

void actionCTR (char en[10], u8 IV[16], u8 cipherKey[32], int keyBits, FILE *infp,
              FILE *outfp)
{
        if (en[0] == 'E' || en[0] == 'e')
                encryptCTR (IV, cipherKey, keyBits, infp, outfp);

        else if (en[0] == 'D' || en[0] == 'd')
                decryptCTR (IV, cipherKey, keyBits, infp, outfp);
}

void actionOFB (char en[10], u8 IV[16], u8 cipherKey[32], int keyBits, FILE *infp,
              FILE *outfp)
{
        if (en[0] == 'E' || en[0] == 'e')
                encryptOFB (IV, cipherKey, keyBits, infp, outfp);

        else if (en[0] == 'D' || en[0] == 'd')
                decryptOFB (IV, cipherKey, keyBits, infp, outfp);
}
```

Encryption Functions

```
/* File Name: encrypt.c
 * This file contains all the enryption functions for CBC, CFB, CTR and OFB modes.
 */

#include <assert.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>


#include "sha256.h"
#include "rijndael-alg-fst.h"

void encryptCBC (u8 IV[16], u8 cipherKey[32], int keyBits, FILE *infp, FILE *outfp)
{
        char pword[20] = {0}, firstRound=1;
        int  i = 0, m, n, p, Nr, read;
        u8 buf[8192], pt[16] = {0}, ct[16] = {0}, input[16] = {0}, output[16] = {0};
        u32 rk[4*(MAXNR+1)];

        Nr = rijndaelKeySetupEnc (rk, cipherKey, keyBits);

        while ((read = fread (buf, sizeof(unsigned char), 8192, infp))>0)
        {
                m = read/16;       /* m is the no. of complete 16-byte blocks  */
                n = read%16;       /* n is the 'leftover' no. of bytes (i.e. read%16) */

                if (n != 0)
                {
                        for (i = m*16 + n; i<(m+1)*16; ++i)
                                buf[i]= 0x00;

                        m++;
```

```
                }

                for (p=1; p <= m; p++)
                {
                        for (i=0; i<16; ++i)
                        {
                                pt[i] = buf [(p-1)*16+i];
                                input[i]= pt[i] ^ ((firstRound)?IV[i]:ct[i]);
                        }

                        rijndaelEncrypt (rk, Nr, input, ct);
                        firstRound = 0;
                        fwrite (ct, sizeof(char), 16, outfp);

                }
        }
}

void encryptCFB (u8 IV[16], u8 cipherKey[32], int keyBits, FILE *infp, FILE *outfp)
{
        char pword[20] = {0}, firstRound=1;
        int  i = 0, m, n, p, Nr, read;
        u8 buf[8192], pt[16] = {0}, ct[16] = {0}, input[16]={0}, output[16]={0};
        u32 rk[4*(MAXNR+1)];

        Nr = rijndaelKeySetupEnc (rk, cipherKey, keyBits);

        while ((read = fread (buf, sizeof(unsigned char), 8192, infp))>0)
        {
                m = read/16;
                n = read%16;

                if (n != 0)
                {
                        for (i = m*16 + n; i<(m+1)*16; ++i)
                                buf[i]= 0x00;
                        m++;
                }

                for (p=1; p <= m; p++)
                {
                        for (i=0; i<16; ++i)
                                input[i] = ((firstRound)?IV[i]:ct[i]);

                        rijndaelEncrypt(rk, Nr, input, output);

                        for (i=0; i<16; ++i)
                        {
                                pt[i] = buf [(p-1)*16+i];
                                ct[i] = pt[i] ^ output[i];
                        }

                        firstRound = 0;
                        fwrite (ct, sizeof(char), 16, outfp);

                }
        }
}

void encryptCTR (u8 IV[16], u8 cipherKey[32], int keyBits, FILE *infp, FILE *outfp)
{
        char pword[20] = {0};
        int  i = 0, m, n, p, t, Nr, read;
        u8 buf[8192], pt[16] = {0}, ct[16] = {0}, input[16] = {0}, output[16] = {0};
        u32 rk[4*(MAXNR+1)];

        Nr = rijndaelKeySetupEnc (rk, cipherKey, keyBits);
```

```
        while ((read = fread(buf,sizeof(unsigned char), 8192, infp))>0)
        {
                m = read/16;
                n = read%16;

                if (n != 0)
                {
                        for (i = m*16 + n; i<(m+1)*16; ++i)
                                buf[i]= 0x00;
                        m++;
                }

                for (p=1; p <= m; p++)
                {
                        rijndaelEncrypt(rk, Nr, IV, output);

                        for (t=15; t>0; t--)
                        {
                                if (IV[t]==0xffU)
                                        IV[t]=0x00U;

                                else
                                {
                                        IV[t]++;
                                        break;
                                }
                        }

                        for(i=0; i<16; i++)
                        {
                                 pt[i] = buf [(p-1)*16+i];
                                 ct[i] = pt[i] ^ output[i];
                        }

                        fwrite (ct, sizeof(char), 16, outfp);
                }
        }
}

void encryptOFB (u8 IV[16], u8 cipherKey[32], int keyBits, FILE *infp, FILE *outfp)
{
        char pword[20]={0}, firstRound=1;
        int  i=0, m, n, p, Nr, KC=8, read;
        u8 buf[8192], pt[16] = {0}, ct[16] = {0}, input[16] = {0}, output[16] = {0};
        u32 rk[4*(MAXNR+1)];

        Nr = rijndaelKeySetupEnc (rk, cipherKey, keyBits);

        while ((read = fread(buf,sizeof(unsigned char), 8192, infp))>0)
        {
                m = read/16;
                n = read%16;

                if (n != 0)
                {
                        for (i = m*16 + n; i<(m+1)*16; ++i)
                        {
                                buf[i]= 0x00;
                        }
                        m++;
                }

                for (p=1; p <= m; p++)
                {
                        for (i=0; i<16; ++i)
                                input[i] = ((firstRound)?IV[i]:output[i]);

                        rijndaelEncrypt(rk, Nr, input, output);
```

```
                        for (i=0; i<16; ++i)
                        {
                                pt[i] = buf [(p-1)*16+i];
                                ct[i] = pt[i] ^ output[i];
                        }

                        firstRound = 0;
                        fwrite (ct, sizeof(char), 16, outfp);

                }
        }
}
```

Decryption Functions

```
/* File Name: decrypt.c
 * This file contains all the decryption functions for CBC, CFB, CTR and OFB modes.
 */

#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <string.h>

#include "sha256.h"
#include "rijndael-alg-fst.h"

void decryptCBC (u8 IV[16], u8 cipherKey[32], int keyBits , FILE *infp, FILE *outfp)
{
        char pword[20]= {0}, firstRound=1;
        int i, k = 0, Nr, ch, plen = 0, read, m, p;
        u8 buf[8192], pt[16] = {0}, ct[16] = {0}, ct1[16] = {0}, output[16] = {0};
        u32 Eqrk[4*(MAXNR+1)];
        sha256_context ctx;

        Nr = rijndaelKeySetupDec (Eqrk, cipherKey, keyBits);

        while ((read = fread (buf, sizeof(unsigned char), 8192, infp))>0)
        {
                m = read/16; /* m is the no. of complete 16-byte blocks  */

                for (p=1; p <= m; p++)
                {

                        for (i=0; i<16; ++i)
                                ct[i] = buf [(p-1)*16+i];


                        rijndaelDecrypt (Eqrk, Nr, ct, pt);

                        for (i=0; i<16; ++i)
                                output[i]= pt[i] ^ ((firstRound)?IV[i]:ct1[i]);

                        firstRound = 0;
                        memcpy(ct1, ct ,16);
                        fwrite (output, sizeof(char), 16, outfp);

                }
        }
}

void decryptCFB (u8 IV[16], u8 cipherKey[32], int keyBits , FILE *infp, FILE *outfp)
{
        char pword[20] = {0} , firstRound=1;
        int i, k = 0, Nr, ch, plen = 0, read, m, p;
```

```
        u8 buf[8192], pt[16] = {0}, ct[16]={0}, ct1[16] = {0}, input[16] = {0}, output[16]=
{0};
        u32 rk[4*(MAXNR+1)];
        sha256_context ctx;

        Nr = rijndaelKeySetupEnc (rk, cipherKey, keyBits);

        while ((read = fread (buf, sizeof(unsigned char), 8192, infp))>0)
        {
                m = read/16;

                for (p=1; p <= m; p++)
                {

                        for (i=0; i<16; ++i)
                                input[i] = ((firstRound)?IV[i]:ct[i]);

                        rijndaelEncrypt (rk, Nr, input, output);

                        for (i=0; i<16; ++i)
                        {
                                ct[i] = buf [(p-1)*16+i];
                                pt[i] = ct[i] ^ output[i];
                        }

                        firstRound = 0;
                        fwrite (pt, sizeof(char), 16, outfp);
                }
        }
}

void decryptCTR (u8 IV[16], u8 cipherKey[32], int keyBits , FILE *infp, FILE *outfp)
{
        char pword[20] = {0};
        int  i = 0, m, n, p, t, Nr, read;
        u8 buf[8192], pt[16] = {0}, ct[16] = {0}, input[16] = {0}, output[16] = {0};
        u32 rk[4*(MAXNR+1)];

        Nr = rijndaelKeySetupEnc(rk, cipherKey, keyBits);

        while ((read=fread (buf, sizeof(unsigned char), 8192, infp))>0)
        {
                m = read/16;

                for (p=1; p <= m; p++)
                {
                        rijndaelEncrypt (rk, Nr, IV, output);

                        for (t=15; t>0; t--)
                        {
                                if (IV[t]==0xffU)
                                        IV[t]=0x00U;

                                else
                                {
                                        IV[t]++;
                                        break;
                                }
                         }
                         for(i=0; i<16; i++)
                         {
                                 ct[i] = buf [(p-1)*16+i];
                                 pt[i] = ct[i] ^ output[i];
                         }

                         fwrite (pt, sizeof(char), 16, outfp);

                }
```

```
        }
}

void decryptOFB (u8 IV[16], u8 cipherKey[32], int keyBits , FILE *infp, FILE *outfp)
{
        char pword[20] = {0} , firstRound=1;
        int i, k = 0, Nr, ch, plen = 0, read, m, p;
        u8 buf[8192], pt[16] = {0}, ct[16] = {0}, ct1[16] = {0}, input[16] = {0}, output[16
] = {0};
        u32 rk[4*(MAXNR+1)];
        sha256_context ctx;

        Nr = rijndaelKeySetupEnc (rk, cipherKey, keyBits);

        while ((read = fread (buf, sizeof(unsigned char), 8192, infp))>0)
        {
                m = read/16;

                for (p=1; p <= m; p++)
                {
                        for (i=0; i<16; ++i)
                                input[i] = ((firstRound)?IV[i]:output[i]);

                        rijndaelEncrypt (rk, Nr, input, output);

                        for (i=0; i<16; ++i)
                        {
                                ct[i] = buf [(p-1)*16+i];
                                pt[i]= ct[i] ^ output[i];
                        }

                        firstRound = 0;
                        fwrite (pt, sizeof(char), 16, outfp);
                }
        }
}
```

README file

```
File Encryption and Decryption Software
Version 2.101 (3 d.p.)
Released 13/12/2010

--------------------------------------------------
Copyright (C) 2010 Choy Jia Li Valerie, Mak Michelle, Tan Wei Lin,
Tok Genevieve, DSO National Laboratories.
All Rights Reserved.


DESCRIPTION
---------------
This program is designed to assist you in encrypting a file that you want to keep
secure. It is not designed to have a visual interface.


INSTALLATION
---------------
No installation is needed.


USAGE
---------------
```

To use, open Command Prompt. Drag and drop or type the path of the executable file into Command Prompt window, followed by the source file that you wish to encrypt/decrypt and the destination file, separating them by a space as such:

<executable file.exe><SPACE><source file><SPACE><destination file>

The source file cannot be the same as the destination file. If the destination file does not exist, it will automatically be created for you.

Press enter to begin the encryption or decryption process. Follow the instructions given in the program.

Passwords chosen should be secure or there is no guarantee your files will be safe. Different IVs should be chosen when encrypting different files with the same password.

All file types can be encrypted and decrypted with this program (as far as we know).