A Project Report

On

Secure Local File System

Submitted to

SHAHEED SUKHDEV COLLEGE OF BUSINESS STUDIES

In partial fulfillment of the requirements for the award of degree of

Post Graduate Diploma in Cyber Security and Law

Submitted by:

Shanti Kumar Deepak - 18704

SarahUlfat – 18730

Anushka Singh - 18756

Supervised by:

Sanjeev Multani

Infosec Trainer, Lucideus

UNIVERSITY OF DELHI

NEW-DELHI

DECLARATION

I hereby declare that the project work entitled **Secure Local File Storage** submitted to the Shaheed Sukhdev College of Business Studies, is a record of an original work done by me under the guidance of **Sanjeev Multani, _______ Team**, **Lucideus**, and this project work is submitted in the partial fulfillment of the requirements for the award of the degree of Post Graduate Diploma in Cyber Security and Law. The results embodied in this report have not been submitted to any other University or Institute for the award of any degree or diploma.

Signature of the Students (with date)

Signature of the Guide (with date)

ABSTRACT

Encryption is one of the best solutions to protect personal and sensitive information. File encryption can play an important role when it comes to securing of user’s sensitive personal information and lost or theft of devices from unauthorized access. The main objective is to keep data secured and encrypted even when not in use. With the help of this project, we aim to provide a user interface to encrypt all of the files in a specific directory by executing our program which would then encrypt all files in that directory and its subdirectories with a much faster and stronger encryption mechanism.

Table of Contents

1. Introduction

1.1 Cryptography

Cryptography is the science of using mathematics to encrypt and decrypt data. Cryptography enables you to store sensitive information or transmit it across insecure networks (like the Internet) so that it cannot be read by anyone except the intended recipient.

A cryptographic algorithm or cipher is a mathematical function used in the encryption and decryption process. A cryptographic algorithm works in combination with a key, a word, number or phrase to encrypt the plaintext. The same plaintext encrypts to different cipher text with different keys. The security of encrypted data is entirely dependent on two things: the strength of the cryptographic algorithm and the secrecy of the key.

2. Project Description

2.1 Objective

The main objective is to design a file encrypting application which would also act as a file encrypting ransomware such that our local file storage will be secured from unwanted access. At present, there are some loop holes which exist in various operating systems like in Windows or Linux OS, one can easily bypass user login when the machine is physically available. The application will prove to be useful in securing any sensitive data from being disclosed to unauthorized user(s) if the system goes into the wrong hands.

In order to gain access to information, the user would be asked for a key to decrypt the encrypted files but if anyone attempts to decrypt the files with the wrong entered keys, the files would then be unrecoverable.

2.2 Domain

The project is based on the principles of cryptography as we secure data at rest to hide personal information from intruders.

In order words, a special type of file encryptor lets us choose which directory needs to be sourced to the application which would then encrypt all the files of that directory and its subdirectories.

2.3 Scope

The scope of the project is any stand-alone machine running Linux, MAC or Windows Operating System. Once the execution starts, the application will encrypt all the files of the desired directory and its subdirectories.

2.4 Future Scope & Enhancements

In the current scenario, the application encrypts files of only a single machine on which it is being executed but with further improvements, it will also be able to encrypt any shared directories with respect to the permissions of the host machine.

Moreover, it can be extended to work upon the machines available in the network that is, network level file encryption.

The key management framework can also be improved as currently, we input one password and the application creates a valid key to encrypt all the files with that same key. In future, we can work upon automatic key generation and management algorithms to make the application run faster and without much user intervention.

2.5 Tools & Technology Used

Python 2.7 version on a **Kali Linux** machine has been used for the development of the application. Other tools including the usual text editor (nano) and Sublime Text editor are used for better coding and design experience.

2.6 Related Study

2.6.1 AES(Advanced Encryption Standard)

AES or Advanced Encryption Standards (also known as Rijndael) is one of the most widely used methods for encrypting and decrypting sensitive information.
This encryption method uses what is known as a block cipher algorithm to ensure that data can be stored securely.
The more popular and widely adopted symmetric encryption algorithm likely to be encountered nowadays is the Advanced Encryption Standard (AES). It is found at least six times faster than triple DES.
A replacement for DES was needed as its key size was too small. With increasing computing power, it was considered vulnerable against exhaustive key search attack. Triple DES was designed to overcome this drawback, but it was found slow.
The features of AES are as follows:

1. Symmetric key symmetric block cipher
2. 128-bit data (128/192/256-bit keys)
3. Stronger and faster than Triple-DES
4. Provide full specification and design details

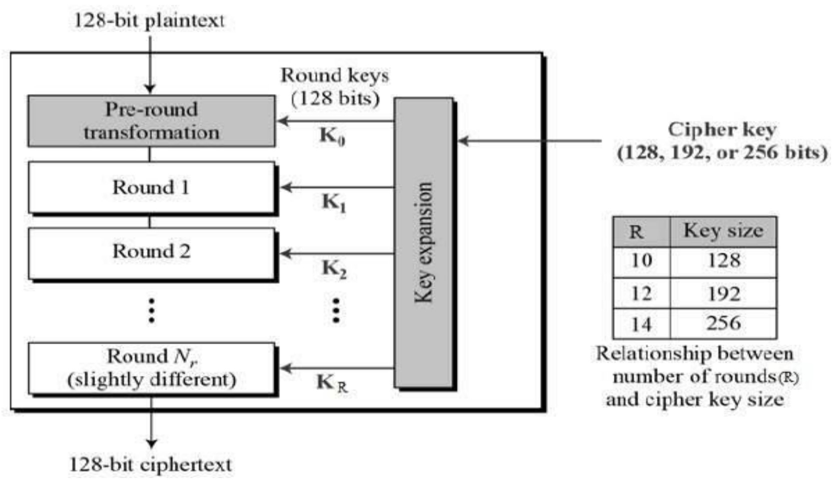
Operation of AES

AES is an iterative rather than Feistel cipher. It is based on ‘substitution–permutation network’. It comprises of a series of linked operations, some of which involve replacing inputs by specific outputs (substitutions) and others involve shuffling bits around (permutations).

Interestingly, AES performs all its computations on bytes rather than bits. Hence, AES treats the 128 bits of a plaintext block as 16 bytes. These 16 bytes are arranged in four columns and four rows for processing as a matrix.

Unlike DES, the number of rounds in AES is variable and depends on the length of the key. AES uses 10 rounds for 128-bit keys, 12 rounds for 192-bit keys and 14 rounds for 256-bit keys. Each of these rounds uses a different 128-bit round key, which is calculated from the original AES key.

The schematic of AES structure is given in the following illustration:

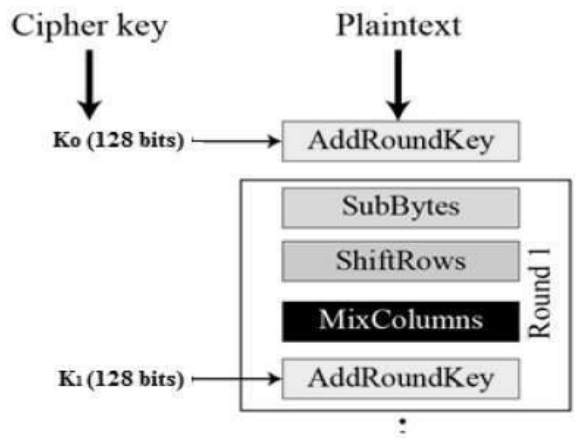


R	Key size
10	128
12	192
14	256

Encryption Process

Here, we restrict to description of a typical round of AES encryption. Each round comprise of four sub-processes. The first transformation in the AES encryption cipher is substitution of data using a substitution table; the second transformation shifts data rows, the third mixes columns. The last transformation is a simple exclusive or (XOR) operation performed on each column using a different part of the encryption key, longer keys need more rounds to complete.

The first-round process is depicted below:



Byte Substitution (SubBytes)

The 16 input bytes are substituted by looking up a fixed table (S-box) given in design. The result is in a matrix of four rows and four columns.

Shift Rows

Each of the four rows of the matrix is shifted to the left. Any entries that 'fall off' are re-inserted on the right side of row. Shift is carried out as follows −

- First row is not shifted.
- Second row is shifted one (byte) position to the left.
- Third row is shifted two positions to the left.
- Fourth row is shifted three positions to the left.
- The result is a new matrix consisting of the same 16 bytes but shifted with respect to each other.

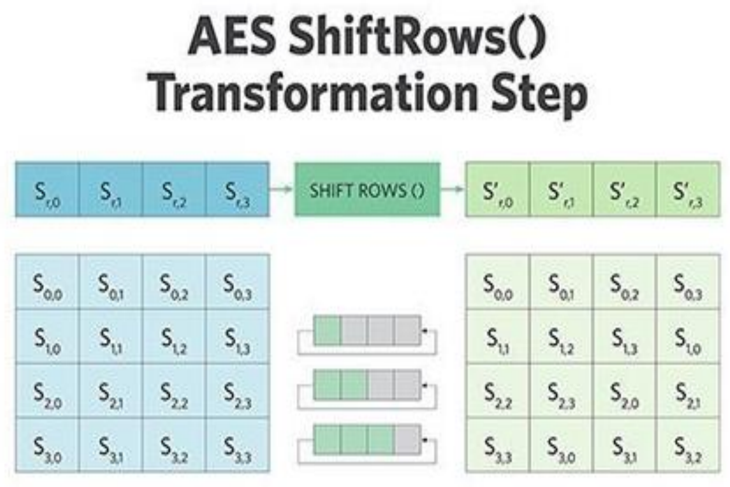


Fig- AES Shift Rows Transformation step

Mix Columns

Each column of four bytes is now transformed using a special mathematical function. This function takes as input the four bytes of one column and outputs four completely new bytes, which replace the original column. The result is another new matrix consisting of 16 new bytes. It should be noted that this step is not performed in the last round.

Add round key

The 16 bytes of the matrix are now considered as 128 bits and are XORed to the 128 bits of the round key. If this is the last round then the output is the cipher text. Otherwise, the resulting 128 bits are interpreted as 16 bytes and we begin another similar round.

Decryption Process

The process of decryption of an AES cipher text is similar to the encryption process in the reverse order. Each round consists of the four processes conducted in the reverse order −

- Add round key
- Mix columns
- Shift rows
- Byte substitution

Since sub-processes in each round are in reverse manner, unlike for a Feistel Cipher, the encryption and decryption algorithms need to be separately implemented, although they are very closely related.

2.6.2 Mode of Operations Used with AES

The block ciphers are schemes for encryption or decryption where a block of plaintext is treated as a single block and is used to obtain a block of cipher text with the same size.

Today, AES (Advanced Encryption Standard) which is one of the most used algorithms for block encryption encompasses the use of Cipher Block Chaining Mode (CBC) and Galois/Counter Mode (GCM).

2.6.3 Cipher Block Chaining Mode (CBC)

Invented by IBM in 1976, this mode of operation provides cryptographic security using an initialization vector (IV) which is of the same size as the block that is encrypted.

First, an XOR operation is applied to the plaintext block (P_1) with the IV and then an encryption with the key (K) is performed. Then, the results of the encryption performed on each block (C_1, C_2, …, C_{N-1}) is used in an XOR operation of the next plaintext block P_N which results in C_N as shown in Figure 2.7.1 (a). In this way, when identical plaintext blocks are encrypted, a different result is obtained. It should be emphasized that the same key K is used in each of the encryption blocks.

During decrypting of a cipher text block, the intended recipient should add XOR the output data received from the decryption algorithm to the previous cipher text block as shown in Figure 2.7.1 (b). As the receiver knows all the cipher text blocks just after obtaining the encrypted message, s/he can decrypt the message using many threads simultaneously.

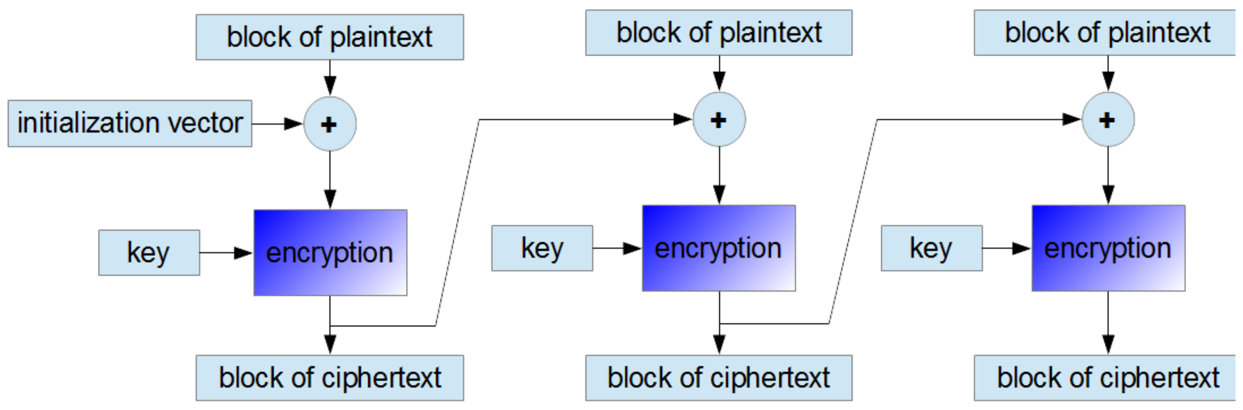


Figure 2.7 (a) Encryption in the CBC Mode

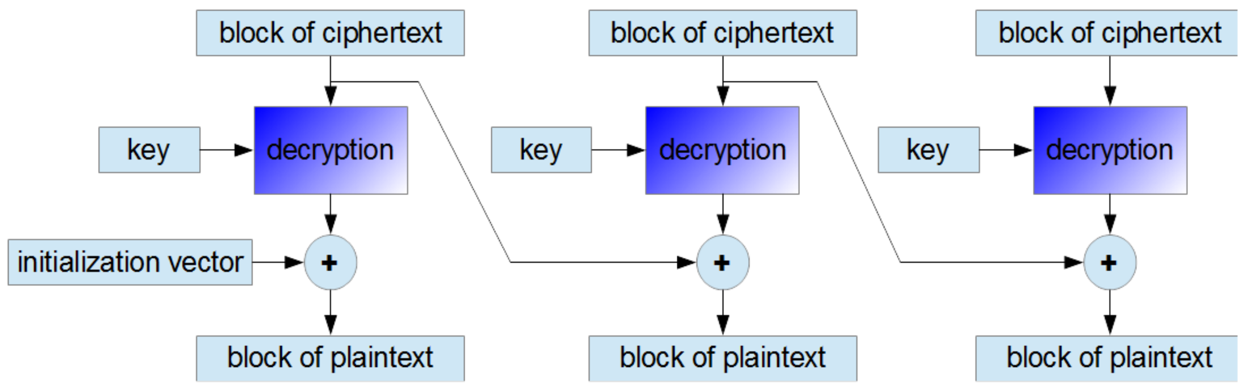
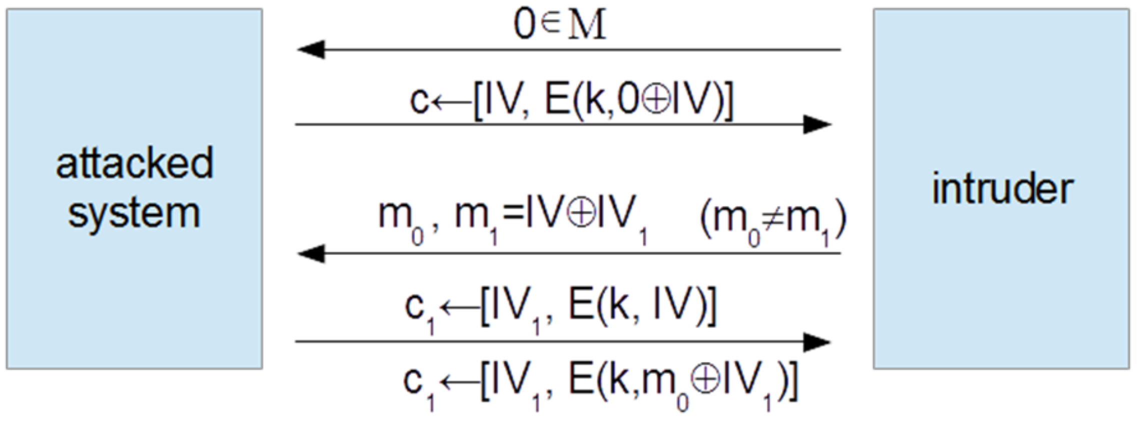


Figure 2.7.1 (b) Decryption in the CBC Mode

Security of the CBC Mode

The initialization vector (IV) should be created randomly by the sender. In order to allow decryption of the message by the receiver, it should be concatenated with the cipher text blocks during transmission.



attacked system
intruder
$0 \in M$
$c \leftarrow [IV, E(k, 0 \oplus IV)]$
$m_0, m_1 = IV \oplus IV_1 \quad (m_0 \neq m_1)$
$c_1 \leftarrow [IV_1, E(k, IV)]$
$c_1 \leftarrow [IV_1, E(k, m_0 \oplus IV_1)]$

Figure 2.7.1 (c)

In the example presented above, if the intruder is able to predict that the vector IV_1 will be used by the attacked system to produce the response c_1, they can guess which one of the two encrypted messages $m_0$or m_1 is carried by the response c_1. This situation breaks the rule that the intruder shouldn't be able to distinguish between two cipher texts even if they have chosen both plaintexts. Therefore, the attacked system is vulnerable to chosen-plaintext attacks.

If the vector IV is generated based on non-random data, for example the user password, it should be encrypted before use. One should use a separate secret key for this activity.

The initialization vector IV should be changed after using the secret key a number of times. It can be shown that even properly created IV used too many times, makes the system vulnerable to chosen-plaintext attacks. For AES cipher it is estimated to be 248 blocks while for 3DES it is about 216 plaintext blocks.

Demerits:

1. If one bit of a plaintext message is damaged (for example because of some earlier transmission error), all subsequent cipher text blocks will be damaged and it will be never possible to decrypt the cipher text received from this plaintext. As opposed to that, if one cipher text bit is damaged, only two received plaintext blocks will be damaged.

2. It is vulnerable to Chosen-Cipher text Attacks (CCA).
3. CBC mode suffers from a serialization problem during encryption. The cipher text C[n] cannot be computed until C[n–1] is known which prevents the cipher from being invoked in parallel.
4. If an intruder could predict what vector would be used, then the encryption would not be resistant to Chosen-Plaintext Attacks (CPA).

2.6.4 Galois/Counter Mode (GCM)

It is a Block Cipher Mode of operation that uses universal hashing over a binary Galois field to provide authenticated encryption. The GCM mode uses an initialization vector (IV) in its processing. This mode is used for authenticated encryption with associated data. GCM provides confidentiality and authenticity for the encrypted data and authenticity for the additional authenticated data (AAD). The AAD is not encrypted. GCM mode requires that the IV is a nonce that is, the IV must be unique for each execution of the mode under the given key.

GCM has two operations namely, authenticated encryption and authenticated decryption.

The authenticated encryption operation has four inputs, each of which is a bit string:

1. A secret key K whose length is appropriate for the underlying block cipher.
2. An initialization vector IV that can have any number of bits between 1 and 2^{64}. For a fixed value of the key, each IV value must be distinct, but need not have equal lengths.
3. A plaintext P which can have any number of bits between 0 and $2^{39} - 256$.
4. Additional authenticated data (AAD), which is denoted as A. This data is authenticated, but not encrypted, and can have any number of bits between 0 and 2^{64}.

The two outputs of the authenticated encryption function are:

1. A cipher text C whose length is exactly that of the plaintext P.
2. An authentication tag T whose length can be any value between 0 and 128.

The authenticated decryption function has five inputs: K, C, IV, A and T. The single output of the authenticated decryption function is either the plaintext P which corresponds to the input cipher text, C or a special symbol FAIL which means the inputs are not authentic. If the inputs were not generated by the encrypt operation with the identical key, it would return FAIL with high probability. The additional authenticated data A is used to protect information that needs to be authenticated but which must be left unencrypted.

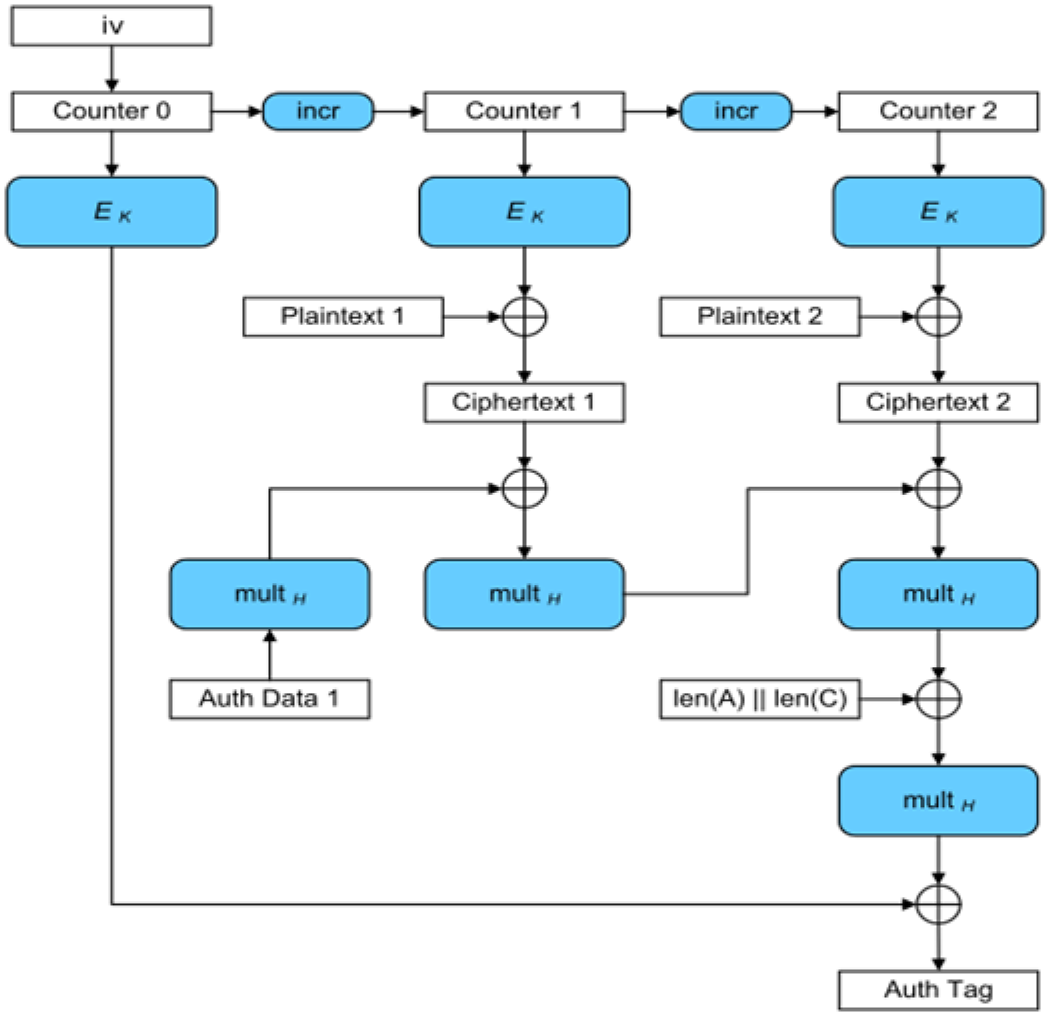


Figure 2.7.2

Merits:

1. It can be implemented in hardware to achieve high speeds with low cost and low latency.
2. It is capable of acting as a stand-alone MAC, authenticating messages when there are no data to encryptwith no modifications.
3. It accepts initialization vectors of arbitrary length, which makes it easier for applications to meet the requirement that all IVs be distinct.

2.7 Comaparison: AES-CBC & AES-GCM

AES-CBC operates by XOR'ing (eXclusive OR) each block with the previous block and cannot be written in parallel thereby affecting performance due to complex mathematics involved requiring serial encryption. It is also vulnerable to padding oracle attacks which exploit the tendency of block ciphers to add arbitrary values onto the end of the last block in a sequence in order to meet the specified block size.

On the other hand, Galois/Counter mode (GCM) of operation (AES-128-GCM) combines Galois field multiplication with the Counter Mode of operation for block ciphers. The Counter Mode of operation is designed to change block ciphers into stream ciphers where each block is encrypted with a pseudo-random value from a ‘keystream’. This concept achieves this by using successive values of an incrementing ‘counter’ such that every block is encrypted with a unique value that is unlikely to re-occur. The Galois field multiplication component takes this to the next level by conceptualizing each block as its own finite field for the use of encryption on the basis of the AES standard.

Galois/Counter Mode can take full advantage of parallel processing and implementing GCM can make efficient use of an instruction pipeline or a hardware pipeline unlike the Cipher Block Chaining (CBC) mode of operation which incurs significant pipeline stalls that hamper its efficiency and performance. The AES-GCM mode of operation can actually be carried out in parallel both for encryption and decryption. Furthermore, this method also allows VPN to only use a 128-bit key whereas AES-CBC typically requires a 256-bit key to be considered secure.

Therefore, AES-GCM is a more secure cipher than AES-CBC.

3 Project Implementation and Design

3.1 Python Library and Modules:

3.1.1 PyCrypto:

PyCrypto is a library, which provides secure hash functions and various encryption algorithms. It supports Python version 2.1 through 3.3.

3.1.2 Crypto.Hash Package:

Cryptographic hash functions take arbitrary binary strings as input, and produce a random-like fixed-length output (called digest or hash value).

It is practically infeasible to derive the original input data from the digest. In other words, the cryptographic hash function is one-way (pre-image resistance).

Given the digest of one message, it is also practically infeasible to find another message (second pre-image) with the same digest (weak collision resistance).

Finally, it is infeasible to find two arbitrary messages with the same digest (strong collision resistance).

Hash functions can be simply used as integrity checks. In combination with a public-key algorithm, you can implement a digital signature.

API principles

Every time you want to hash a message, you have to create a new hash object with the new() function in the relevant algorithm module (e.g. Crypto.Hash.SHA256.new()).

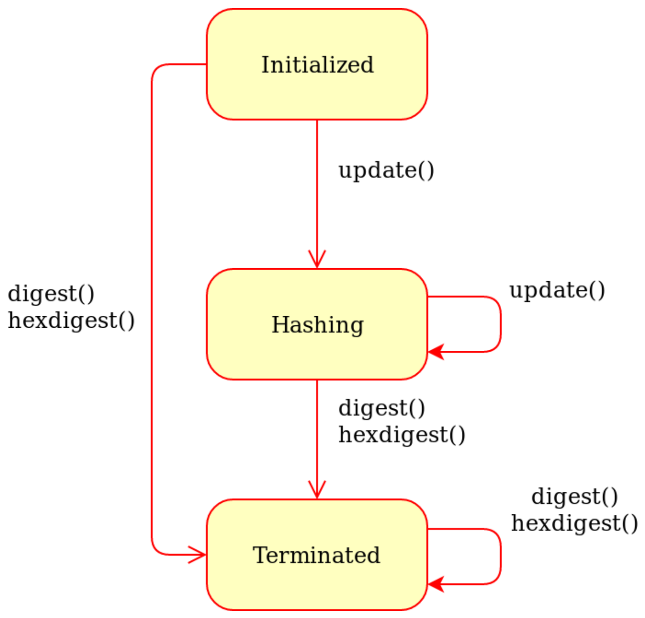


Fig-1: Generic state diagram for a hash object

A first piece of message to hash can be passed to new() with the data parameter:

```
>> from Crypto.Hash import SHA256
>> hash_object = SHA256.new(data=b'First')
```

Attributes of hash objects

Every hash object has the following attributes:

Attribute	Description
digest_size	Size of the digest in bytes, that is, the output of the **digest()** method. It does not exist for hash functions with variable digest output (such as **Crypto.Hash.SHAKE128**). This is also a module attribute.
block_size	The size of the message block in bytes, input to the compression function. Only applicable for algorithms based on the Merkle-Damgard construction (e.g. **Crypto.Hash.SHA256**). This is also a module attribute.
Oid	A string with the dotted representation of the ASN.1 OID assigned to the hash algorithm.

3.1.3 Crypto.Cipher Package:

The Crypto.Cipher package contains algorithms for protecting the confidentiality of data.

There are three types of encryption algorithms:

Symmetric ciphers: all parties use the same key, for both decrypting and encrypting data. Symmetric ciphers are typically very fast and can process very large amount of data.

Asymmetric ciphers: senders and receivers use different keys. Senders encrypt with public keys (non-secret) whereas receivers decrypt with private keys (secret). Asymmetric ciphers are typically very slow and can process only very small payloads. Example: PKCS#1 OAEP (RSA).

Hybrid ciphers: the two types of ciphers above can be combined in a construction that inherits the benefits of both. An asymmetric cipher is used to protect a short-lived symmetric key, and a symmetric cipher (under that key) encrypts the actual message.

API principles

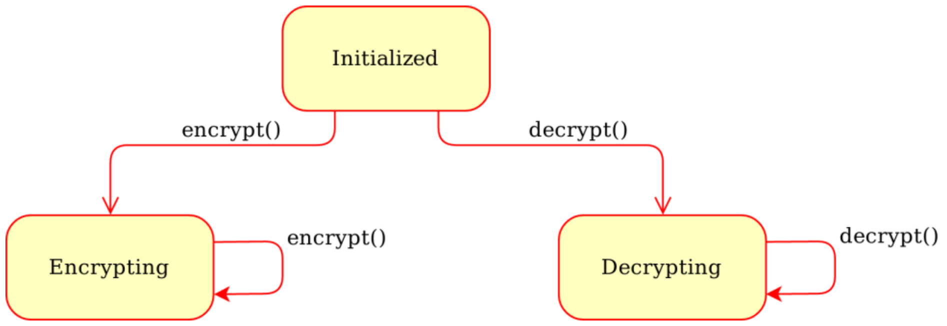


Fig-2: Generic state diagram for a cipher object

The base API of a cipher is fairly simple:

You instantiate a cipher object by calling the new() function from the relevant cipher module (e.g. Crypto.Cipher.AES.new()). The first parameter is always the cryptographic key; its length depends on the particular cipher. You can (and sometimes must) pass additional cipher- or mode-specific parameters to new() (such as a nonce or a mode of operation).

For encrypting data, you call the encrypt() method of the cipher object with the plaintext. The method returns the piece of cipher text. Alternatively, with the output parameter you can specify a pre-allocated buffer for the result.

For most algorithms, you may call encrypt() multiple times (i.e. once for each piece of plaintext).

For decrypting data, you call the decrypt() method of the cipher object with the cipher text. The method returns the piece of plaintext. The output parameter can be passed here too.
For most algorithms, you may call decrypt() multiple times

3.1.4 import os:

The OS module in python provides functions for interacting with the operating system. OS, comes under Python's standard utility modules. This module provides a portable way of using operating system dependent functionality. The *os* and *os.path* modules include many functions to interact with the file system.

Following are some functions in OS module:

1. os.name: This function gives the name of the operating system dependent module imported.
2. os.getcwd(): Function os.getcwd(), returns the Current Working Directory(CWD) of the file used to execute the code, can vary from system to system.
3. os.error: All functions in this module raise OSError in the case of invalid or inaccessible file names and paths, or other arguments that have the correct type, but are not accepted by the operating system. os.error is an alias for built-in OSError exception.
4. os.close(): Close file descriptor fd. A file opened using open(), can be closed by close()only. But file opened through os.popen(), can be closed with close() or os.close(). If we try closing a file opened with open(), using os.close(), Python would throw TypeError.
5. os.rename(): A file old.txt can be renamed to new.txt, using the function os.rename(). The name of the file changes only if, the file exists and user has sufficient privilege permission to change the file.

6. os.walk(top, topdown=True, onerror=None, followlinks=False)
Generate the file names in a directory tree by walking the tree either top-down or bottom-up. For each directory in the tree rooted at directory top (including top itself), it yields a 3-tuple (dirpath, dirnames, filenames).

dirpath is a string, the path to the directory. dirnames is a list of the names of the subdirectories in dirpath (excluding '.' and '..'). filenames is a list of the names of the non-directory files in dirpath. Note that the names in the lists contain no path components. To get a full path (which begins with top) to a file or directory in dirpath, do os.path.join(dirpath, name).

7. os.remove(path)

Remove (delete) the file path. If path is a directory, OSError is raised; see rmdir() below to remove a directory. This is identical to the unlink() function documented below. On Windows, attempting to remove a file that is in use causes an exception to be raised; on Unix, the directory entry is removed but the storage allocated to the file is not made available until the original file is no longer in use.

8. os.path.join(path, *paths)

Join one or more path components intelligently. The return value is the concatenation of path and any members of *paths with exactly one directory separator (os.sep) following each non-empty part except the last, meaning that the result will only end in a separator if the last part is empty. If a component is an absolute path, all previous components are thrown away and joining continues from the absolute path component.

9. os.path.dirname(path)

Return the directory name of pathname path. This is the first element of the pair returned by passing path to the function split().

3.1.5 import random:

Sometimes we want the computer to pick a random number in a given range, pick a random element from a list, pick a random card from a deck, flip a coin, etc. The random module provides access to functions that support these types of operations. The random module is another library of functions that can extend the basic features of python.

The random module provides access to functions that support many operations.

Perhaps the most important thing is that it allows you to generate random numbers.

3.1.6 import sys

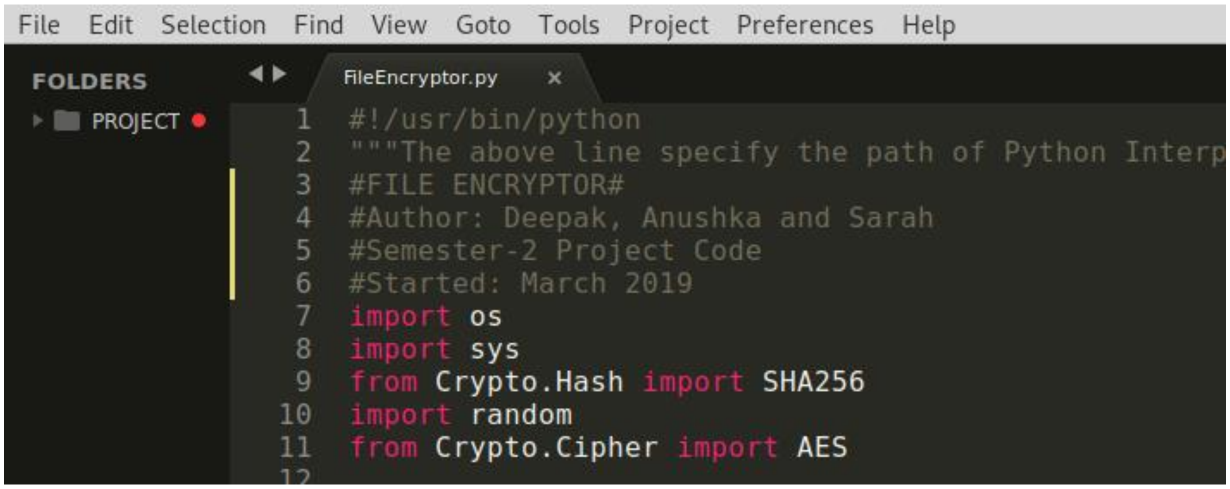
The sys module provides information about constants, functions and methods of the Python interpreter. dir(system) gives a summary of the available constants, functions and methods. Another possibility is the help() function. Using help(sys) provides valuable detail information.

3.1.7 import pkg_resource

The pkg_resources module provides runtime facilities for finding, introspecting, activating and using installed Python distributions. The pkg_resources module distributed with setup tools provides an API for Python libraries to access their resource files and for extensible applications and frameworks to automatically discover plugins.

3.2 Code Description:

3.2.1 import library modules

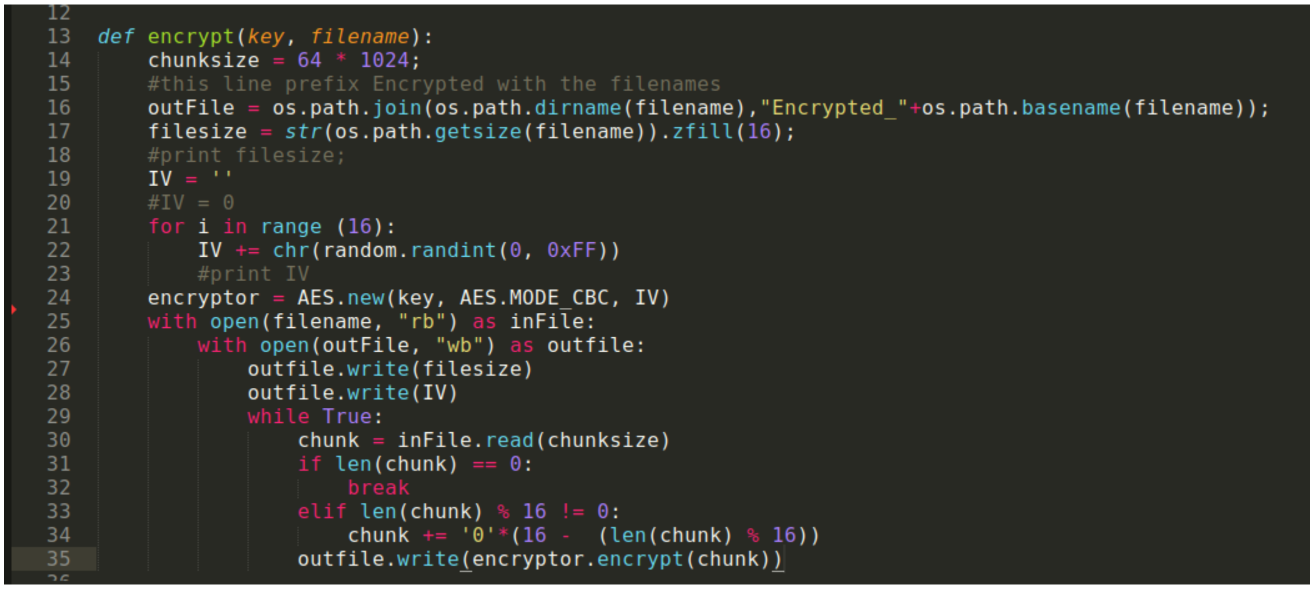

File Edit Selection Find View Goto Tools Project Preferences Help
FOLDERS
PROJECT
FileEncryptor.py

```
#!/usr/bin/python
"""The above line specify the path of Python Interp
#FILE ENCRYPTOR#
#Author: Deepak, Anushka and Sarah
#Semester-2 Project Code
#Started: March 2019
import os
import sys
from Crypto.Hash import SHA256
import random
from Crypto.Cipher import AES
```

Fig – Import Library Modules

Here we have imported important library and modules that we will be going to use in our code.

3.2.2 defining the encrypt function

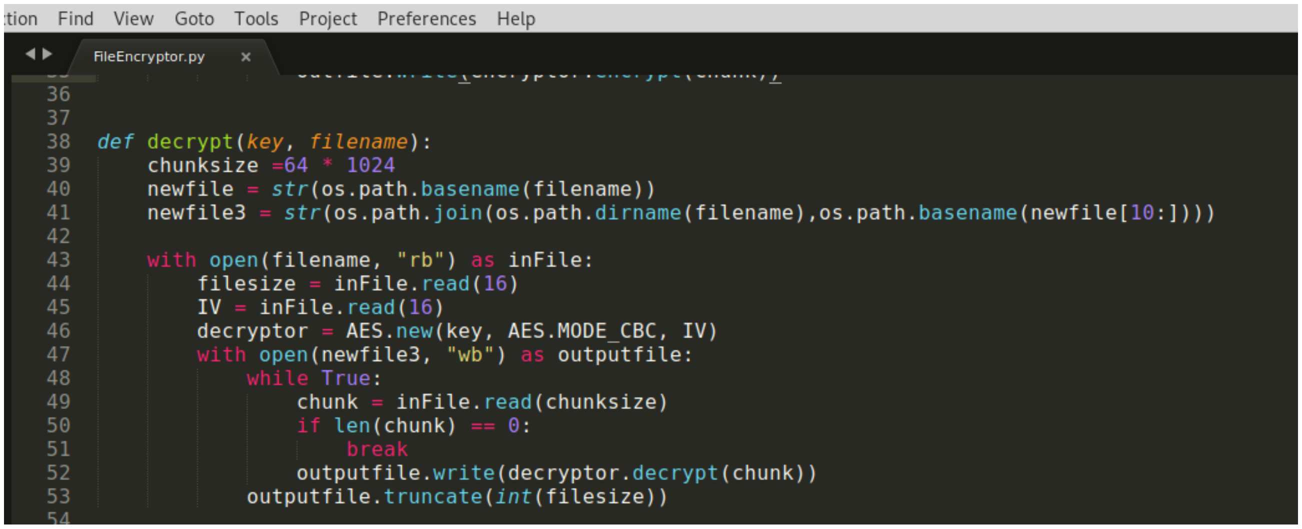


```python
def encrypt(key, filename):
    chunksize = 64 * 1024;
    #this line prefix Encrypted with the filenames
    outFile = os.path.join(os.path.dirname(filename),"Encrypted_"+os.path.basename(filename));
    filesize = str(os.path.getsize(filename)).zfill(16);
    #print filesize;
    IV = ''
    #IV = 0
    for i in range (16):
        IV += chr(random.randint(0, 0xFF))
        #print IV
    encryptor = AES.new(key, AES.MODE_CBC, IV)
    with open(filename, "rb") as inFile:
        with open(outFile, "wb") as outfile:
            outfile.write(filesize)
            outfile.write(IV)
            while True:
                chunk = inFile.read(chunksize)
                if len(chunk) == 0:
                    break
                elif len(chunk) % 16 != 0:
                    chunk += '0'*(16 -  (len(chunk) % 16))
                outfile.write(encryptor.encrypt(chunk))
```

Fig – Encrypt Function

In the above snippet we are defining the process of encrypting a file. This function will be called from the main method repeatedly for each file to be encrypted.

3.2.3 defining the decrypt function

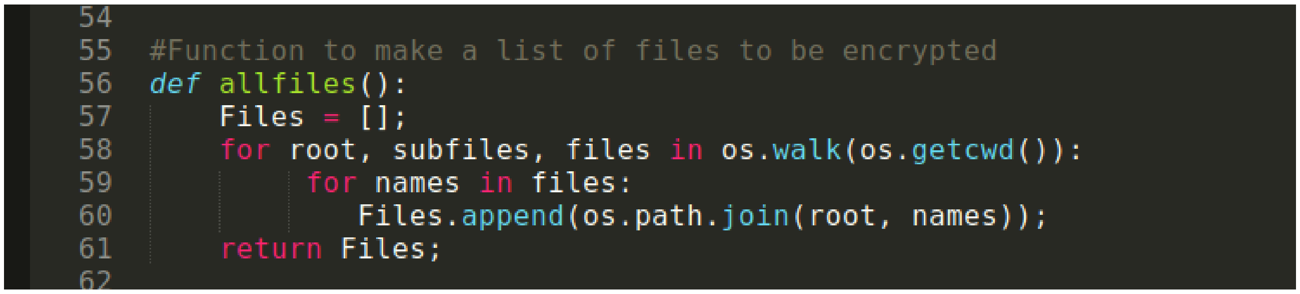


```python
def decrypt(key, filename):
    chunksize =64 * 1024
    newfile = str(os.path.basename(filename))
    newfile3 = str(os.path.join(os.path.dirname(filename),os.path.basename(newfile[10:])))

    with open(filename, "rb") as inFile:
        filesize = inFile.read(16)
        IV = inFile.read(16)
        decryptor = AES.new(key, AES.MODE_CBC, IV)
        with open(newfile3, "wb") as outputfile:
            while True:
                chunk = inFile.read(chunksize)
                if len(chunk) == 0:
                    break
                outputfile.write(decryptor.decrypt(chunk))
            outputfile.truncate(int(filesize))
```

Fig – Decrypt Function

In the above snippet we are defining the process of decrypting a file. This function will be called from the main method repeatedly for each file to be decrypted.

3.2.4 defining the allfile function

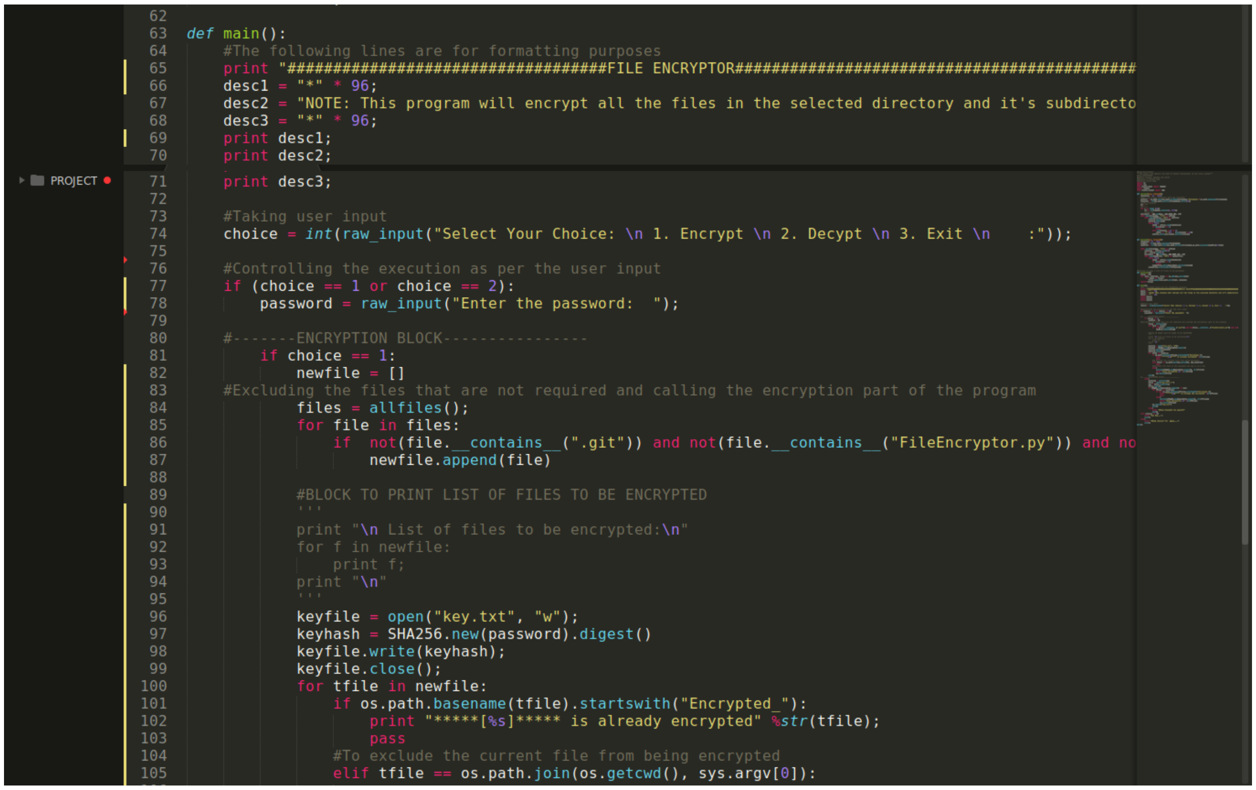


```

#Function to make a list of files to be encrypted
def allfiles():
    Files = [];
    for root, subfiles, files in os.walk(os.getcwd()):
        for names in files:
            Files.append(os.path.join(root, names));
    return Files;

```

Fig – allfile() Function

In the above snippet we are defining the function allfiles() which will list down all the files in the current directory and all of its subdirectories and put them in a list called Files.

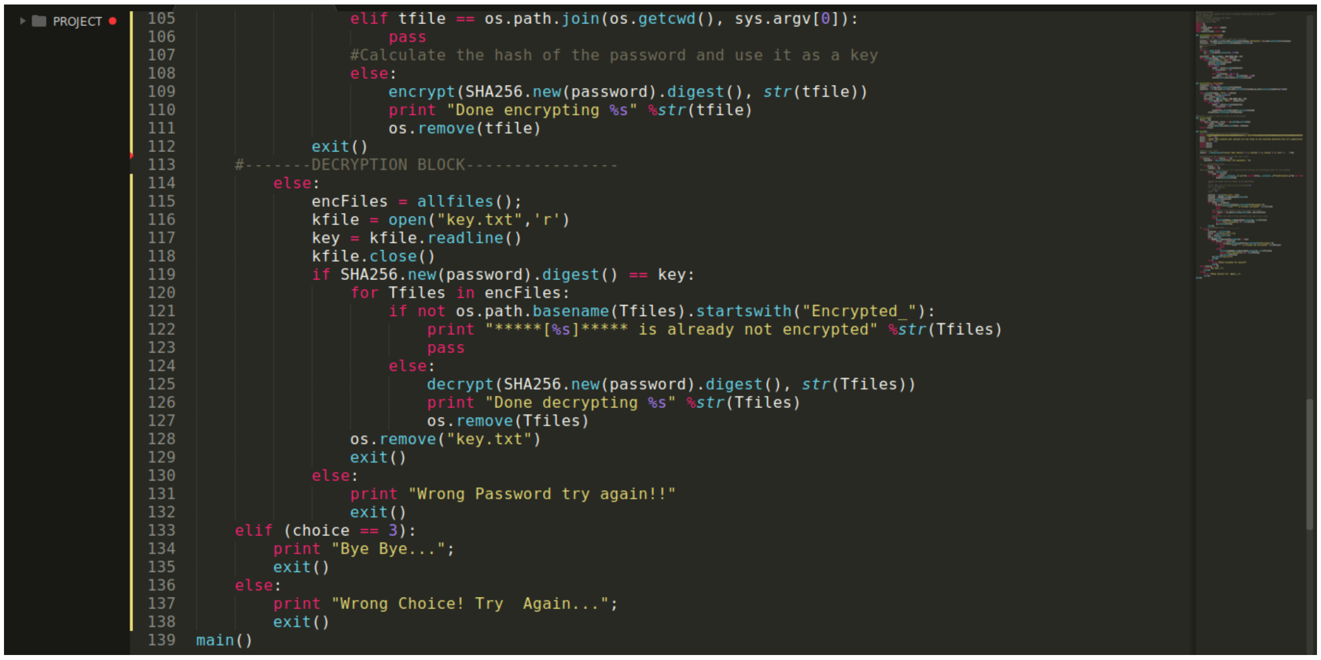
3.2.5 defining the main function and calling encrypt and decrypt functions



```
 62
 63  def main():
 64      #The following lines are for formatting purposes
 65      print "##################################FILE ENCRYPTOR###########################################
 66      desc1 = "*" * 96;
 67      desc2 = "NOTE: This program will encrypt all the files in the selected directory and it's subdirecto
 68      desc3 = "*" * 96;
 69      print desc1;
 70      print desc2;
 71      print desc3;
 72
 73      #Taking user input
 74      choice = int(raw_input("Select Your Choice: \n 1. Encrypt \n 2. Decypt \n 3. Exit \n    :"));
 75
 76      #Controlling the execution as per the user input
 77      if (choice == 1 or choice == 2):
 78          password = raw_input("Enter the password:  ");
 79
 80      #-------ENCRYPTION BLOCK----------------
 81          if choice == 1:
 82              newfile = []
 83      #Excluding the files that are not required and calling the encryption part of the program
 84              files = allfiles();
 85              for file in files:
 86                  if  not(file.__contains__(".git")) and not(file.__contains__("FileEncryptor.py")) and no
 87                      newfile.append(file)
 88
 89              #BLOCK TO PRINT LIST OF FILES TO BE ENCRYPTED
 90              '''
 91              print "\n List of files to be encrypted:\n"
 92              for f in newfile:
 93                  print f;
 94              print "\n"
 95              '''
 96              keyfile = open("key.txt", "w");
 97              keyhash = SHA256.new(password).digest()
 98              keyfile.write(keyhash);
 99              keyfile.close();
100              for tfile in newfile:
101                  if os.path.basename(tfile).startswith("Encrypted_"):
102                      print "*****[%s]***** is already encrypted" %str(tfile);
103                      pass
104                  #To exclude the current file from being encrypted
105                  elif tfile == os.path.join(os.getcwd(), sys.argv[0]):
```

Fig – main Function

In the above snippet we are defining the main function which will control the complete flow of the encryption and decryption process with proper checking whether a files needs to be encrypted or not whether it is already encrypted or not. Here from line 84 to 87 we have also designed the logic to exclude the important files for example the running application FileEncryptor.py and the key file.



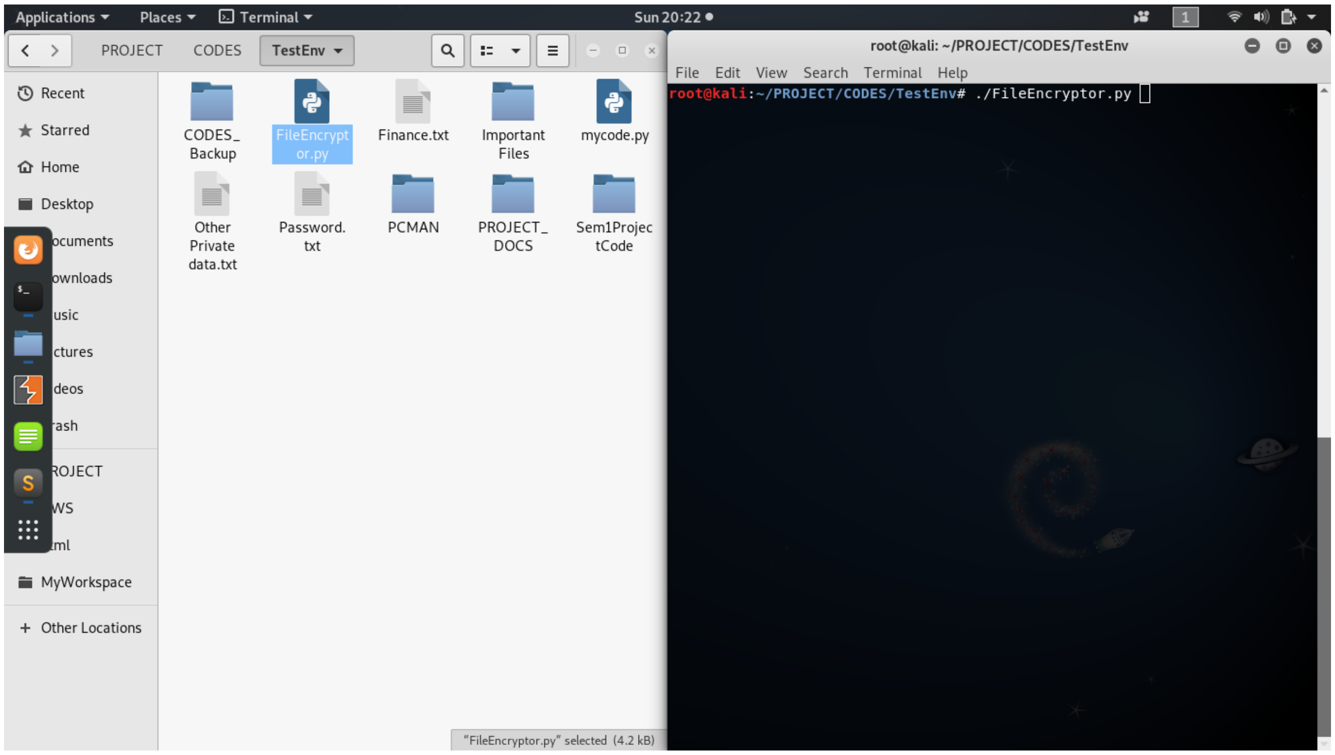
```
                elif tfile == os.path.join(os.getcwd(), sys.argv[0]):
                    pass
                #Calculate the hash of the password and use it as a key
                else:
                    encrypt(SHA256.new(password).digest(), str(tfile))
                    print "Done encrypting %s" %str(tfile)
                    os.remove(tfile)
            exit()
    #-------DECRYPTION BLOCK----------------
        else:
            encFiles = allfiles();
            kfile = open("key.txt",'r')
            key = kfile.readline()
            kfile.close()
            if SHA256.new(password).digest() == key:
                for Tfiles in encFiles:
                    if not os.path.basename(Tfiles).startswith("Encrypted_"):
                        print "*****[%s]***** is already not encrypted" %str(Tfiles)
                        pass
                    else:
                        decrypt(SHA256.new(password).digest(), str(Tfiles))
                        print "Done decrypting %s" %str(Tfiles)
                        os.remove(Tfiles)
                os.remove("key.txt")
                exit()
            else:
                print "Wrong Password try again!!"
                exit()
    elif (choice == 3):
        print "Bye Bye...";
        exit()
    else:
        print "Wrong Choice! Try  Again...";
        exit()
main()
```

Fig – main Function Continue

In the main function we have two blocks one for calling the encryption process/function and one for calling the decryption process. In each block we prepare the list of files that are actually need to be passed in the encrypt or decrypt function.

3.3 Program Execution and Outputs for AES CBC mode:

3.3.1 Prepare the execution file (by making it executable)

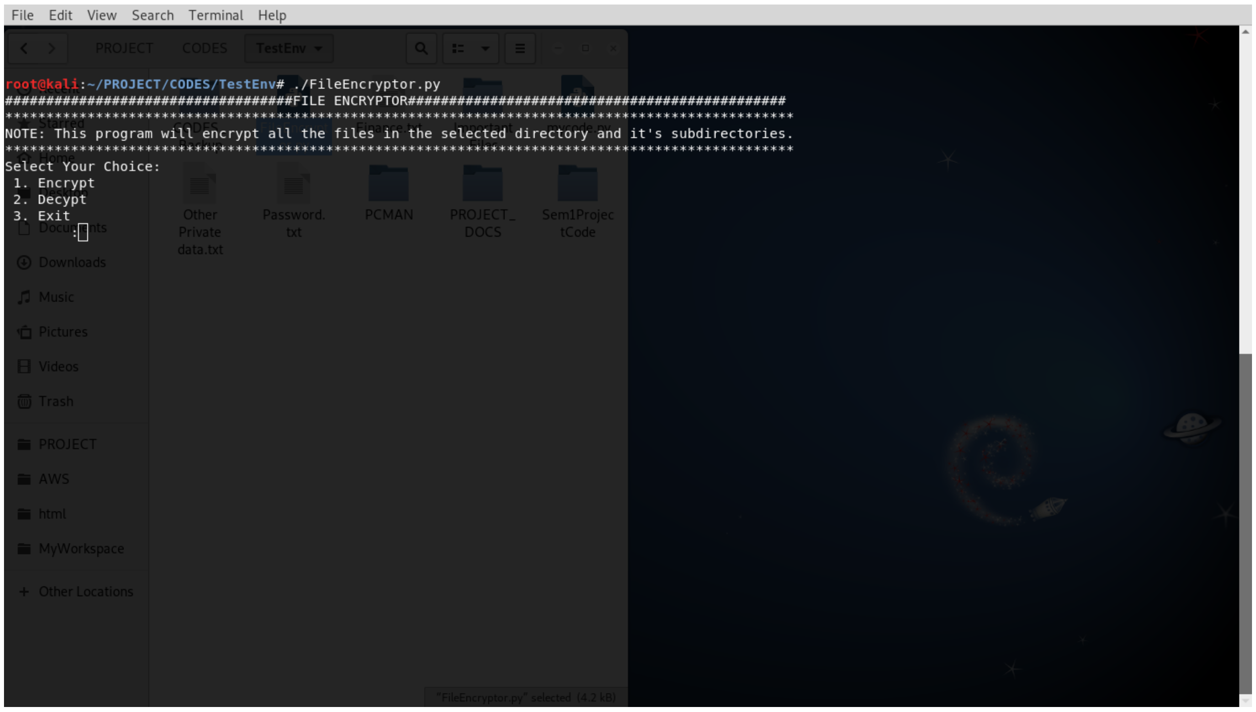


```
root@kali:~/PROJECT/CODES/TestEnv# ./FileEncryptor.py
```

Fig – Start of execution

In the above snippet the **FileEncryptor.py** file is our main application that will encrypt or decrypt the files. We have just opened the directory where this file is placed and a terminal pointing to the same directory.

3.3.2 Start execution of the application



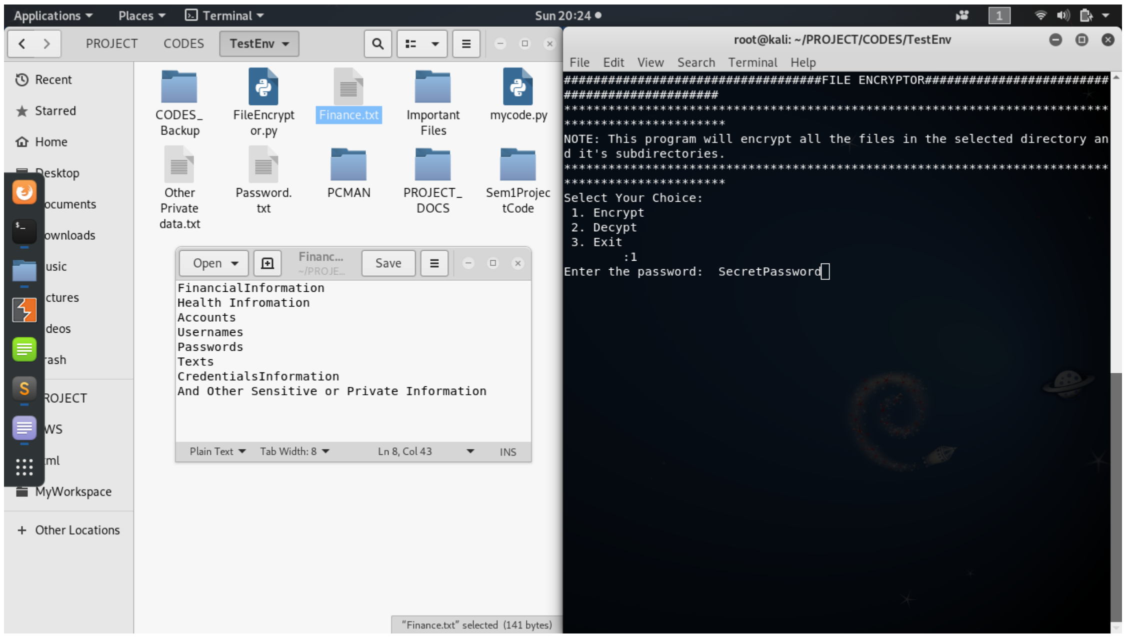
```
root@kali:~/PROJECT/CODES/TestEnv# ./FileEncryptor.py
###################################FILE ENCRYPTOR############################################
*************************************************************************************************
NOTE: This program will encrypt all the files in the selected directory and it's subdirectories.
*************************************************************************************************
Select Your Choice:
 1. Encrypt
 2. Decypt
 3. Exit
         :
```

Fig – welcome screen

The above snippet is showing the welcome screen when we start the execution of our application, it displays three options:

1. **Encrypt** to start the encryption process,
2. **Decrypt** to start the decryption process if one has already used the encrypt option
3. **Exit** if the application launched by mistake.

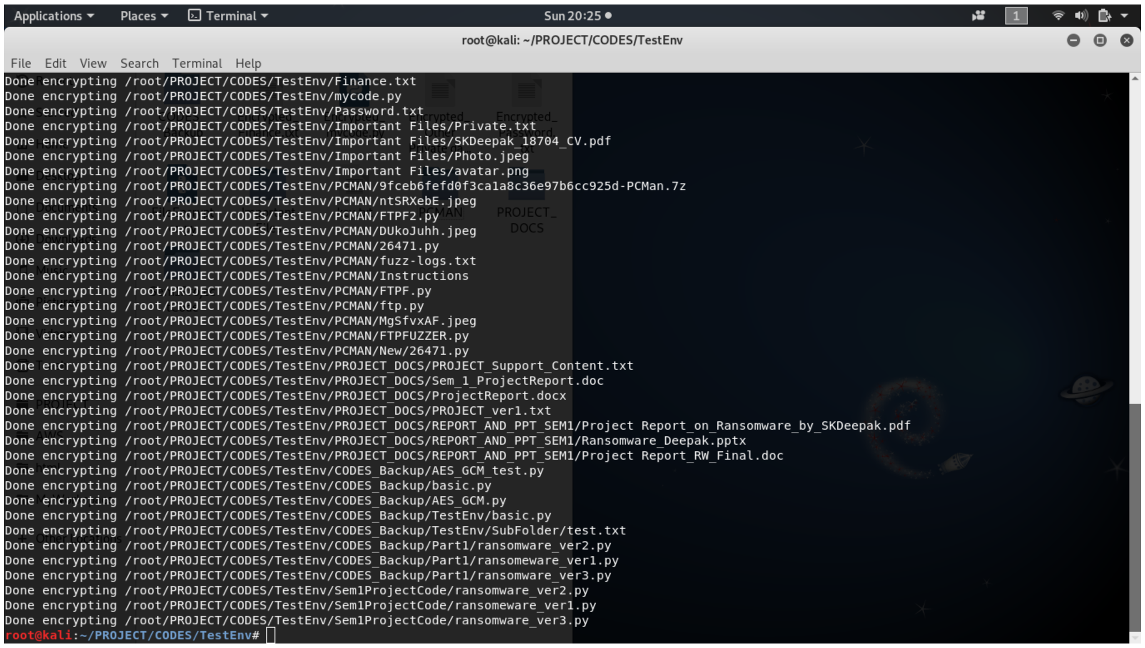
3.3.3 Encryption Process



```
FinancialInformation
Health Infromation
Accounts
Usernames
Passwords
Texts
CredentialsInformation
And Other Sensitive or Private Information
```

```
##################################FILE ENCRYPTOR##########################
#####################
***************************************************************************
*********************
NOTE: This program will encrypt all the files in the selected directory an
d it's subdirectories.
***************************************************************************
*********************
Select Your Choice:
 1. Encrypt
 2. Decypt
 3. Exit
        :1
Enter the password:  SecretPassword
```

Fig – encryption process



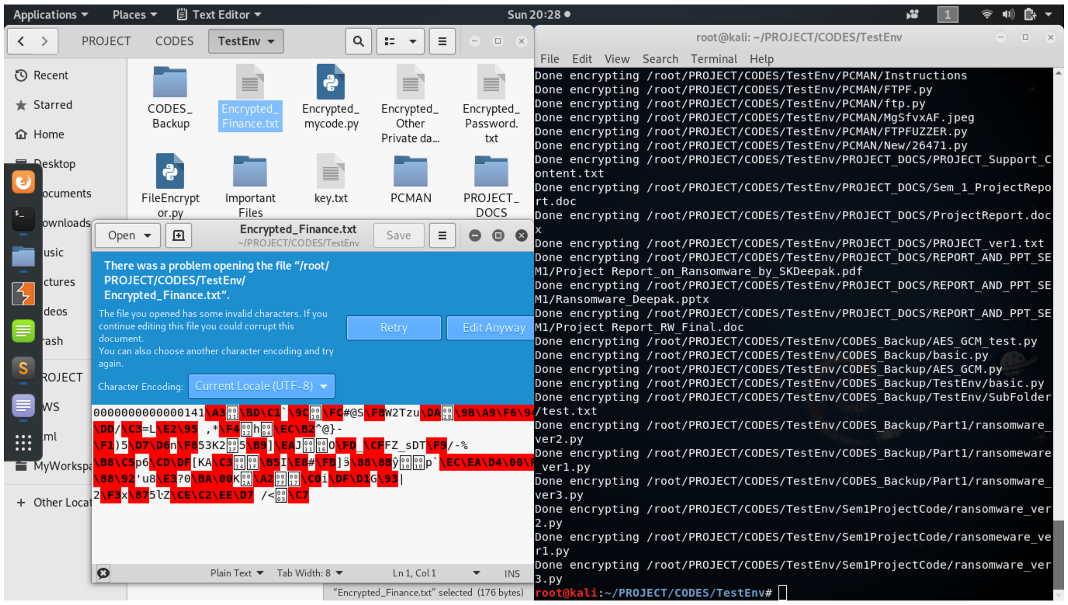
```
Done encrypting /root/PROJECT/CODES/TestEnv/Finance.txt
Done encrypting /root/PROJECT/CODES/TestEnv/mycode.py
Done encrypting /root/PROJECT/CODES/TestEnv/Password.txt
Done encrypting /root/PROJECT/CODES/TestEnv/Important Files/Private.txt
Done encrypting /root/PROJECT/CODES/TestEnv/Important Files/SKDeepak_18704_CV.pdf
Done encrypting /root/PROJECT/CODES/TestEnv/Important Files/Photo.jpeg
Done encrypting /root/PROJECT/CODES/TestEnv/Important Files/avatar.png
Done encrypting /root/PROJECT/CODES/TestEnv/PCMAN/9fceb6fefd0f3ca1a8c36e97b6cc925d-PCMan.7z
Done encrypting /root/PROJECT/CODES/TestEnv/PCMAN/ntSRXebE.jpeg
Done encrypting /root/PROJECT/CODES/TestEnv/PCMAN/FTPF2.py
Done encrypting /root/PROJECT/CODES/TestEnv/PCMAN/DUkoJuhh.jpeg
Done encrypting /root/PROJECT/CODES/TestEnv/PCMAN/26471.py
Done encrypting /root/PROJECT/CODES/TestEnv/PCMAN/fuzz-logs.txt
Done encrypting /root/PROJECT/CODES/TestEnv/PCMAN/Instructions
Done encrypting /root/PROJECT/CODES/TestEnv/PCMAN/FTPF.py
Done encrypting /root/PROJECT/CODES/TestEnv/PCMAN/ftp.py
Done encrypting /root/PROJECT/CODES/TestEnv/PCMAN/MgSfvxAF.jpeg
Done encrypting /root/PROJECT/CODES/TestEnv/PCMAN/FTPFUZZER.py
Done encrypting /root/PROJECT/CODES/TestEnv/PCMAN/New/26471.py
Done encrypting /root/PROJECT/CODES/TestEnv/PROJECT_DOCS/PROJECT_Support_Content.txt
Done encrypting /root/PROJECT/CODES/TestEnv/PROJECT_DOCS/Sem_1_ProjectReport.doc
Done encrypting /root/PROJECT/CODES/TestEnv/PROJECT_DOCS/ProjectReport.docx
Done encrypting /root/PROJECT/CODES/TestEnv/PROJECT_DOCS/PROJECT_ver1.txt
Done encrypting /root/PROJECT/CODES/TestEnv/PROJECT_DOCS/REPORT_AND_PPT_SEM1/Project Report_on_Ransomware_by_SKDeepak.pdf
Done encrypting /root/PROJECT/CODES/TestEnv/PROJECT_DOCS/REPORT_AND_PPT_SEM1/Ransomware_Deepak.pptx
Done encrypting /root/PROJECT/CODES/TestEnv/PROJECT_DOCS/REPORT_AND_PPT_SEM1/Project Report_RW_Final.doc
Done encrypting /root/PROJECT/CODES/TestEnv/CODES_Backup/AES_GCM_test.py
Done encrypting /root/PROJECT/CODES/TestEnv/CODES_Backup/basic.py
Done encrypting /root/PROJECT/CODES/TestEnv/CODES_Backup/AES_GCM.py
Done encrypting /root/PROJECT/CODES/TestEnv/CODES_Backup/TestEnv/basic.py
Done encrypting /root/PROJECT/CODES/TestEnv/CODES_Backup/TestEnv/SubFolder/test.txt
Done encrypting /root/PROJECT/CODES/TestEnv/CODES_Backup/Part1/ransomware_ver2.py
Done encrypting /root/PROJECT/CODES/TestEnv/CODES_Backup/Part1/ransomeware_ver1.py
Done encrypting /root/PROJECT/CODES/TestEnv/CODES_Backup/Part1/ransomware_ver3.py
Done encrypting /root/PROJECT/CODES/TestEnv/Sem1ProjectCode/ransomware_ver2.py
Done encrypting /root/PROJECT/CODES/TestEnv/Sem1ProjectCode/ransomeware_ver1.py
Done encrypting /root/PROJECT/CODES/TestEnv/Sem1ProjectCode/ransomware_ver3.py
root@kali:~/PROJECT/CODES/TestEnv#
```

Fig – encryption process output

Above in the first snippet it is showing on the left hand side the current state of the files and on the right side start of the execution process where if a user selects encrypt(1) option then he/she will be required to input a password. The second screen shows the results once the encrypt process finishes its part.

3.3.4 Status of files after encryption process

Once the encryption process completes the all the files in same directory and its sub directories will become unreadable as shown below:



Encrypted_Finance.txt
~/PROJECT/CODES/TestEnv

There was a problem opening the file "/root/PROJECT/CODES/TestEnv/Encrypted_Finance.txt".

The file you opened has some invalid characters. If you continue editing this file you could corrupt this document.
You can also choose another character encoding and try again.

Character Encoding: Current Locale (UTF-8)

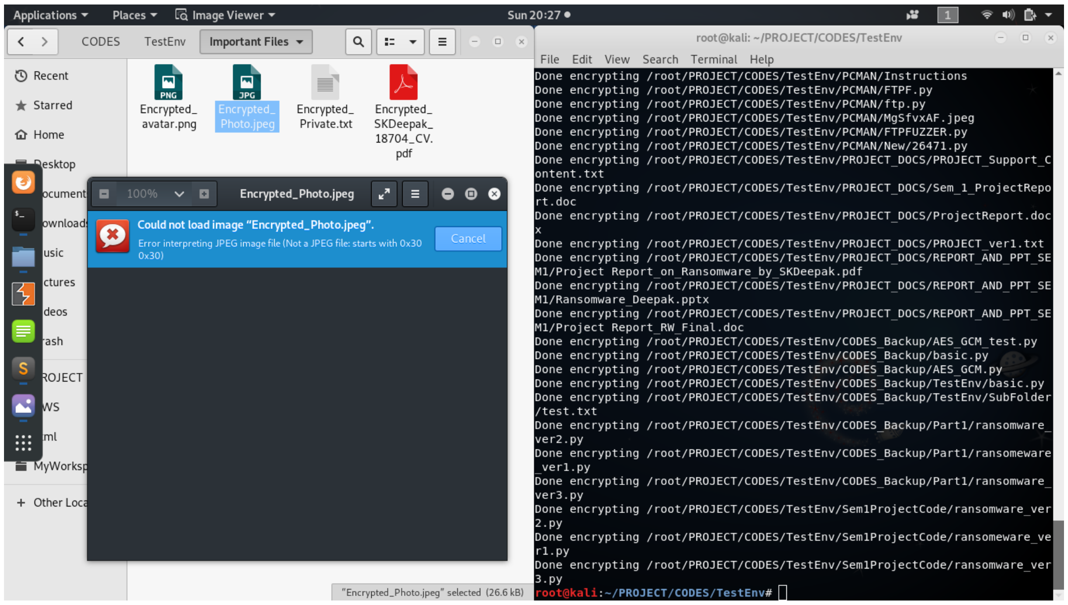
```
0000000000000141\A3\BD\C1`\9C\FC#@S\F8W2Tzu\DA\9B\A9\F6\94
\DD/\C3=L\E2\95 ,*\F4h\EC\B2^@}-
\F1)5\D7\D6n\F853K25\B9]\EAJ0\FD_\CFFZ_sDT\F9/-%
\B8\C5p6\CD\DF[KA\C3\B5I\E8#\FB]ӛ\88\8Bŷp`\EC\EA\D4\00\F
\88\92'u8\E3?0\BA\00K\A2\C0i\DF\D1G\93|
2\F3x\875l·Z\CE\C2\EE\D7 /<\C7
```

"Encrypted_Finance.txt" selected (176 bytes)

root@kali: ~/PROJECT/CODES/TestEnv

```
Done encrypting /root/PROJECT/CODES/TestEnv/PCMAN/Instructions
Done encrypting /root/PROJECT/CODES/TestEnv/PCMAN/FTPF.py
Done encrypting /root/PROJECT/CODES/TestEnv/PCMAN/ftp.py
Done encrypting /root/PROJECT/CODES/TestEnv/PCMAN/MgSfvxAF.jpeg
Done encrypting /root/PROJECT/CODES/TestEnv/PCMAN/FTPFUZZER.py
Done encrypting /root/PROJECT/CODES/TestEnv/PCMAN/New/26471.py
Done encrypting /root/PROJECT/CODES/TestEnv/PROJECT_DOCS/PROJECT_Support_C
ontent.txt
Done encrypting /root/PROJECT/CODES/TestEnv/PROJECT_DOCS/Sem_1_ProjectRepo
rt.doc
Done encrypting /root/PROJECT/CODES/TestEnv/PROJECT_DOCS/ProjectReport.doc
x
Done encrypting /root/PROJECT/CODES/TestEnv/PROJECT_DOCS/PROJECT_ver1.txt
Done encrypting /root/PROJECT/CODES/TestEnv/PROJECT_DOCS/REPORT_AND_PPT_SE
M1/Project Report_on_Ransomware_by_SKDeepak.pdf
Done encrypting /root/PROJECT/CODES/TestEnv/PROJECT_DOCS/REPORT_AND_PPT_SE
M1/Ransomware_Deepak.pptx
Done encrypting /root/PROJECT/CODES/TestEnv/PROJECT_DOCS/REPORT_AND_PPT_SE
M1/Project Report_RW_Final.doc
Done encrypting /root/PROJECT/CODES/TestEnv/CODES_Backup/AES_GCM_test.py
Done encrypting /root/PROJECT/CODES/TestEnv/CODES_Backup/basic.py
Done encrypting /root/PROJECT/CODES/TestEnv/CODES_Backup/AES_GCM.py
Done encrypting /root/PROJECT/CODES/TestEnv/CODES_Backup/TestEnv/basic.py
Done encrypting /root/PROJECT/CODES/TestEnv/CODES_Backup/TestEnv/SubFolder
/test.txt
Done encrypting /root/PROJECT/CODES/TestEnv/CODES_Backup/Part1/ransomware_
ver2.py
Done encrypting /root/PROJECT/CODES/TestEnv/CODES_Backup/Part1/ransomeware
_ver1.py
Done encrypting /root/PROJECT/CODES/TestEnv/CODES_Backup/Part1/ransomware_
ver3.py
Done encrypting /root/PROJECT/CODES/TestEnv/Sem1ProjectCode/ransomware_ver
2.py
Done encrypting /root/PROJECT/CODES/TestEnv/Sem1ProjectCode/ransomeware_ve
r1.py
Done encrypting /root/PROJECT/CODES/TestEnv/Sem1ProjectCode/ransomware_ver
3.py
root@kali:~/PROJECT/CODES/TestEnv#
```

Fig – encryption process output 1



Encrypted_Photo.jpeg

Could not load image "Encrypted_Photo.jpeg".
Error interpreting JPEG image file (Not a JPEG file: starts with 0x30 0x30)

"Encrypted_Photo.jpeg" selected (26.6 kB)

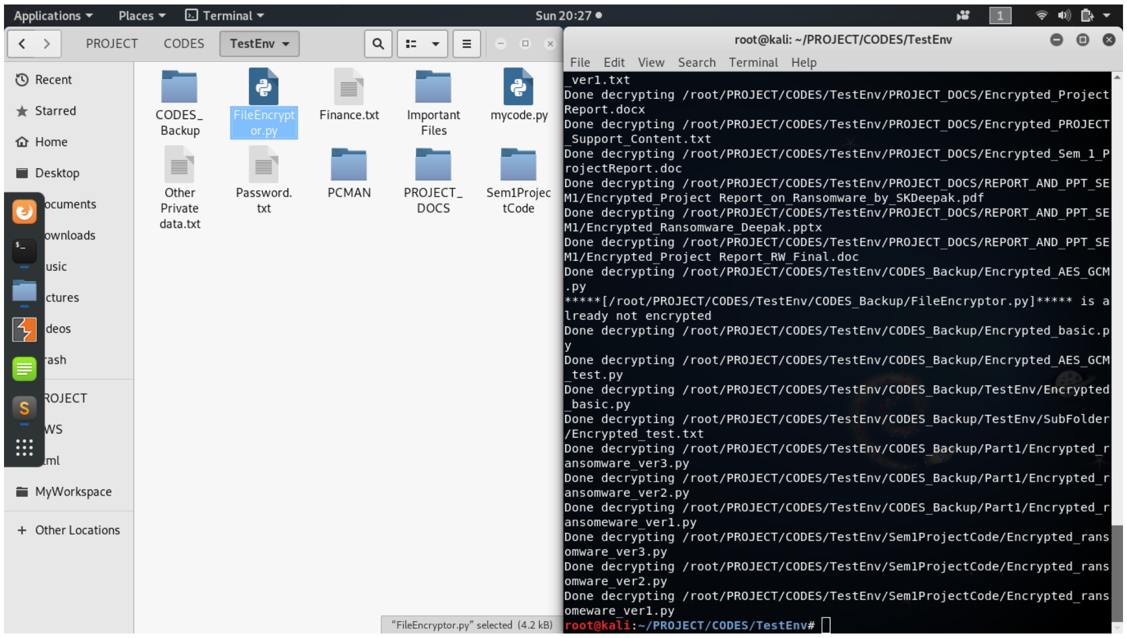
root@kali: ~/PROJECT/CODES/TestEnv

```
Done encrypting /root/PROJECT/CODES/TestEnv/PCMAN/Instructions
Done encrypting /root/PROJECT/CODES/TestEnv/PCMAN/FTPF.py
Done encrypting /root/PROJECT/CODES/TestEnv/PCMAN/ftp.py
Done encrypting /root/PROJECT/CODES/TestEnv/PCMAN/MgSfvxAF.jpeg
Done encrypting /root/PROJECT/CODES/TestEnv/PCMAN/FTPFUZZER.py
Done encrypting /root/PROJECT/CODES/TestEnv/PCMAN/New/26471.py
Done encrypting /root/PROJECT/CODES/TestEnv/PROJECT_DOCS/PROJECT_Support_C
ontent.txt
Done encrypting /root/PROJECT/CODES/TestEnv/PROJECT_DOCS/Sem_1_ProjectRepo
rt.doc
Done encrypting /root/PROJECT/CODES/TestEnv/PROJECT_DOCS/ProjectReport.doc
x
Done encrypting /root/PROJECT/CODES/TestEnv/PROJECT_DOCS/PROJECT_ver1.txt
Done encrypting /root/PROJECT/CODES/TestEnv/PROJECT_DOCS/REPORT_AND_PPT_SE
M1/Project Report_on_Ransomware_by_SKDeepak.pdf
Done encrypting /root/PROJECT/CODES/TestEnv/PROJECT_DOCS/REPORT_AND_PPT_SE
M1/Ransomware_Deepak.pptx
Done encrypting /root/PROJECT/CODES/TestEnv/PROJECT_DOCS/REPORT_AND_PPT_SE
M1/Project Report_RW_Final.doc
Done encrypting /root/PROJECT/CODES/TestEnv/CODES_Backup/AES_GCM_test.py
Done encrypting /root/PROJECT/CODES/TestEnv/CODES_Backup/basic.py
Done encrypting /root/PROJECT/CODES/TestEnv/CODES_Backup/AES_GCM.py
Done encrypting /root/PROJECT/CODES/TestEnv/CODES_Backup/TestEnv/basic.py
Done encrypting /root/PROJECT/CODES/TestEnv/CODES_Backup/TestEnv/SubFolder
/test.txt
Done encrypting /root/PROJECT/CODES/TestEnv/CODES_Backup/Part1/ransomware_
ver2.py
Done encrypting /root/PROJECT/CODES/TestEnv/CODES_Backup/Part1/ransomeware
_ver1.py
Done encrypting /root/PROJECT/CODES/TestEnv/CODES_Backup/Part1/ransomware_
ver3.py
Done encrypting /root/PROJECT/CODES/TestEnv/Sem1ProjectCode/ransomware_ver
2.py
Done encrypting /root/PROJECT/CODES/TestEnv/Sem1ProjectCode/ransomeware_ve
r1.py
Done encrypting /root/PROJECT/CODES/TestEnv/Sem1ProjectCode/ransomware_ver
3.py
root@kali:~/PROJECT/CODES/TestEnv#
```

Fig – encryption process output 2

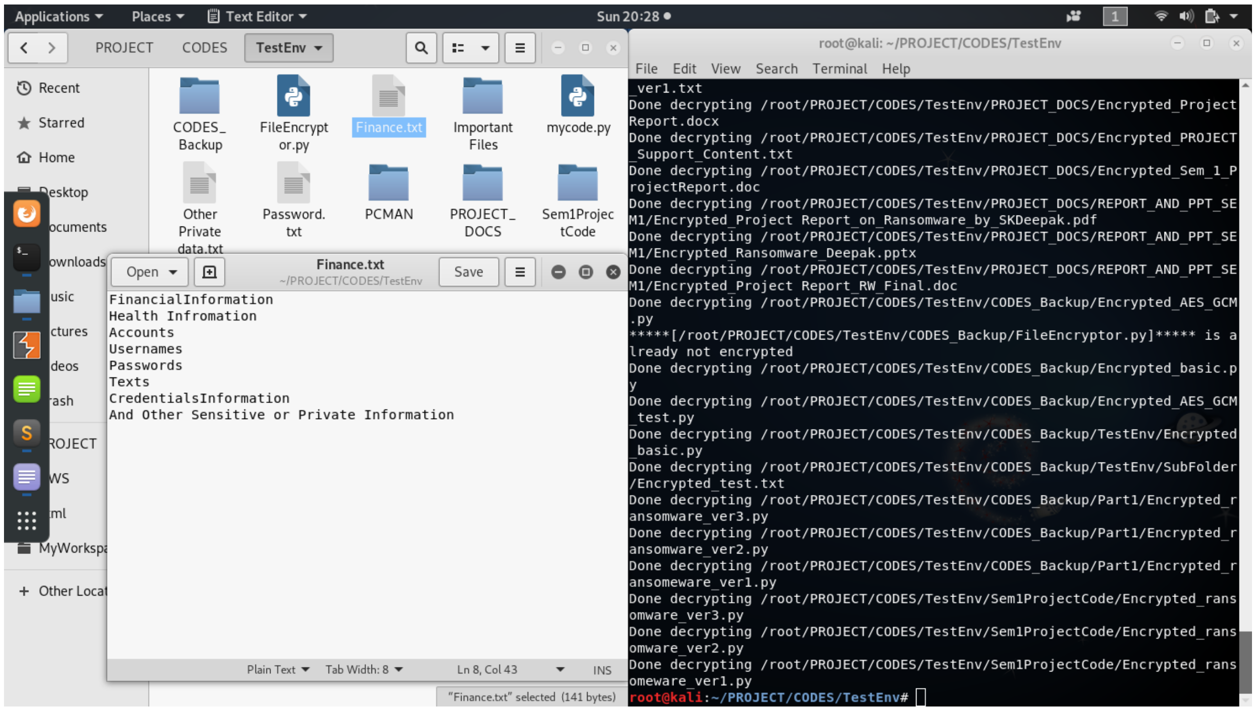
During the encryption process the application prefix the word **Encrypt_** with all the files to identify later in the decryption process that what all files are encrypted, so that the decryption process will only be performed on those files.

3.3.5 Decryption Process



```
_ver1.txt
Done decrypting /root/PROJECT/CODES/TestEnv/PROJECT_DOCS/Encrypted_Project
Report.docx
Done decrypting /root/PROJECT/CODES/TestEnv/PROJECT_DOCS/Encrypted_PROJECT
_Support_Content.txt
Done decrypting /root/PROJECT/CODES/TestEnv/PROJECT_DOCS/Encrypted_Sem_1_P
rojectReport.doc
Done decrypting /root/PROJECT/CODES/TestEnv/PROJECT_DOCS/REPORT_AND_PPT_SE
M1/Encrypted_Project Report_on_Ransomware_by_SKDeepak.pdf
Done decrypting /root/PROJECT/CODES/TestEnv/PROJECT_DOCS/REPORT_AND_PPT_SE
M1/Encrypted_Ransomware_Deepak.pptx
Done decrypting /root/PROJECT/CODES/TestEnv/PROJECT_DOCS/REPORT_AND_PPT_SE
M1/Encrypted_Project Report_RW_Final.doc
Done decrypting /root/PROJECT/CODES/TestEnv/CODES_Backup/Encrypted_AES_GCM
.py
*****[/root/PROJECT/CODES/TestEnv/CODES_Backup/FileEncryptor.py]***** is a
lready not encrypted
Done decrypting /root/PROJECT/CODES/TestEnv/CODES_Backup/Encrypted_basic.p
y
Done decrypting /root/PROJECT/CODES/TestEnv/CODES_Backup/Encrypted_AES_GCM
_test.py
Done decrypting /root/PROJECT/CODES/TestEnv/CODES_Backup/TestEnv/Encrypted
_basic.py
Done decrypting /root/PROJECT/CODES/TestEnv/CODES_Backup/TestEnv/SubFolder
/Encrypted_test.txt
Done decrypting /root/PROJECT/CODES/TestEnv/CODES_Backup/Part1/Encrypted_r
ansomware_ver3.py
Done decrypting /root/PROJECT/CODES/TestEnv/CODES_Backup/Part1/Encrypted_r
ansomware_ver2.py
Done decrypting /root/PROJECT/CODES/TestEnv/CODES_Backup/Part1/Encrypted_r
ansomeware_ver1.py
Done decrypting /root/PROJECT/CODES/TestEnv/Sem1ProjectCode/Encrypted_rans
omware_ver3.py
Done decrypting /root/PROJECT/CODES/TestEnv/Sem1ProjectCode/Encrypted_rans
omware_ver2.py
Done decrypting /root/PROJECT/CODES/TestEnv/Sem1ProjectCode/Encrypted_rans
omeware_ver1.py
root@kali:~/PROJECT/CODES/TestEnv#
```

Fig – Decryption process



```
FinancialInformation
Health Infromation
Accounts
Usernames
Passwords
Texts
CredentialsInformation
And Other Sensitive or Private Information
```

```
_ver1.txt
Done decrypting /root/PROJECT/CODES/TestEnv/PROJECT_DOCS/Encrypted_Project
Report.docx
Done decrypting /root/PROJECT/CODES/TestEnv/PROJECT_DOCS/Encrypted_PROJECT
_Support_Content.txt
Done decrypting /root/PROJECT/CODES/TestEnv/PROJECT_DOCS/Encrypted_Sem_1_P
rojectReport.doc
Done decrypting /root/PROJECT/CODES/TestEnv/PROJECT_DOCS/REPORT_AND_PPT_SE
M1/Encrypted_Project Report_on_Ransomware_by_SKDeepak.pdf
Done decrypting /root/PROJECT/CODES/TestEnv/PROJECT_DOCS/REPORT_AND_PPT_SE
M1/Encrypted_Ransomware_Deepak.pptx
Done decrypting /root/PROJECT/CODES/TestEnv/PROJECT_DOCS/REPORT_AND_PPT_SE
M1/Encrypted_Project Report_RW_Final.doc
Done decrypting /root/PROJECT/CODES/TestEnv/CODES_Backup/Encrypted_AES_GCM
.py
*****[/root/PROJECT/CODES/TestEnv/CODES_Backup/FileEncryptor.py]***** is a
lready not encrypted
Done decrypting /root/PROJECT/CODES/TestEnv/CODES_Backup/Encrypted_basic.p
y
Done decrypting /root/PROJECT/CODES/TestEnv/CODES_Backup/Encrypted_AES_GCM
_test.py
Done decrypting /root/PROJECT/CODES/TestEnv/CODES_Backup/TestEnv/Encrypted
_basic.py
Done decrypting /root/PROJECT/CODES/TestEnv/CODES_Backup/TestEnv/SubFolder
/Encrypted_test.txt
Done decrypting /root/PROJECT/CODES/TestEnv/CODES_Backup/Part1/Encrypted_r
ansomware_ver3.py
Done decrypting /root/PROJECT/CODES/TestEnv/CODES_Backup/Part1/Encrypted_r
ansomware_ver2.py
Done decrypting /root/PROJECT/CODES/TestEnv/CODES_Backup/Part1/Encrypted_r
ansomeware_ver1.py
Done decrypting /root/PROJECT/CODES/TestEnv/Sem1ProjectCode/Encrypted_rans
omware_ver3.py
Done decrypting /root/PROJECT/CODES/TestEnv/Sem1ProjectCode/Encrypted_rans
omware_ver2.py
Done decrypting /root/PROJECT/CODES/TestEnv/Sem1ProjectCode/Encrypted_rans
omeware_ver1.py
root@kali:~/PROJECT/CODES/TestEnv#
```

Fig – Decryption process output

The decryption process again asks for the password and only when the correct password is provided to the input, the application will start the decryption process and restores the files status as shown in above snapshots.