FACULTY OF INFORMATICS

COURSEWORK COVERSHEET

SUBJECT'S INFORMATION:			
Subject:	CSCI361 Cryptography and Secure Applications		
Session:	July 2019		
Programme / Section:	BCS (S1)		
Lecturer:	**Mohamad Faizal Alias**		
Coursework Type *(tick appropriate box)*	❑ Individual Assessment		
Coursework Title:	Assessment 3	Coursework Percentage:	10%
Hand-out Date:	Week 9	Received By : (signature)	
Due Date:	Week 13	Received Date :	
STUDENT'S INFORMATION:			
Student's Name & ID:	**TEH WIN SAM** **J16021533** **6306196**		
Contact Number / Email:	**0133696298 /** **ME@TEHWINSAM.COM**		
STUDENT'S DECLARATION			

By signing this, I / We declare that:

1. This assignment meets all the requirements for the subject as detailed in the relevant Subject Outline, which I/ we have read.
2. It is my / our own work and I / we did not collaborate with or copy from others.
3. I / we have read and understand my responsibilities under the University of Wollongong's policy on plagiarism.
4. I / we have not plagiarised from published work (including the internet). Where I have used the work from others, I / we have referenced it in the text and provided a reference list at the end of the assignment.

I am / we are aware that late submission without an authorised extension from the subject co-ordinator may incur a penalty. *(See your subject outline for further information).*

Name & Signature:	TEH WI N SAM		

-----------✂ ------------------✂ -----------------✂ ------------------✂ -------------------✂ --------------------✂ --

COURSEWORK SUBMISSION RECEIPT

Subject:	CSCI361 Cryptography and Secure Application	Session:	July 2019
Programme / Section:	BCS (S1)	Lecturer:	**Mohamad Faizal Alias**
Coursework Type: (Tick appropriate box)	❑ Individual Assessment		
Coursework Title:	Assessment 3	Coursework Percentage:	10%
Hand-out Date:	Week 9	Received By: (Signature)	
Due date:	Week 13	Received Date:	
STUDENT'S INFORMATION:			
Student's Name & ID:			
Contact Number / Email:			

	Assessment Criteria	**Total Marks**	**Given Marks**
1.	**Part 1** – Online Quiz 2 - AES	2	
2.	**Part 2** – Implementation of RSA Digital Signature Scheme and Hashing Tool (MD-5 & SHA-1)		
	RSA Digital Signature – Signing and Verifying	3	
	MD-5 and SHA-1 tool applying on a file	3	
	Report	2	
		10	

	Penalty	
Marked by: ______________________ Date: _____________	**Final Mark (10 %)**	

Lecturer’s Comments
Penalty for late submission:
1 day – minus 20% of total mark awarded 2 days – minus 50% of total mark awarded 3 days – 0 mark for this piece of coursework

University of Wollongong

CSCI361: Cryptography and Secure Applications

July 2019 Session

Individual Assessment 3

Aims

This assessment consists of TWO parts. Part 1 is to assess the students on the understanding of Advance Encryption Standard (AES) via Online Quiz 2. Part 2 is a program development to simulate the implementation of RSA Digital Signature scheme and Hashing tool.

Objectives

The assessment includes the following objectives:

1. Fully understand the implementation of AES and the internal algorithm of AES
2. Able to apply the concept of Digital Signature Scheme using RSA algorithm.
3. Implementing Key Generation code
4. Implementing RSA Signature Generation and Verification code
5. Using MD-5 and SHA-1 library to create tool for File Hashing

Assessment 3 Part 1: (2%) – Online Quiz 2

Part 1 of Assessment 3 is consists of an Online Quiz 2 covering topic of AES.

You are expected to study thoroughly Chapter 7 – Advance Encryption Standard (AES)

This Quiz 2 contribute 2% of the overall 10% of Assessment 3.

Assessment 3 Part 2 Specifications: (8%) - Individual

Specifications:

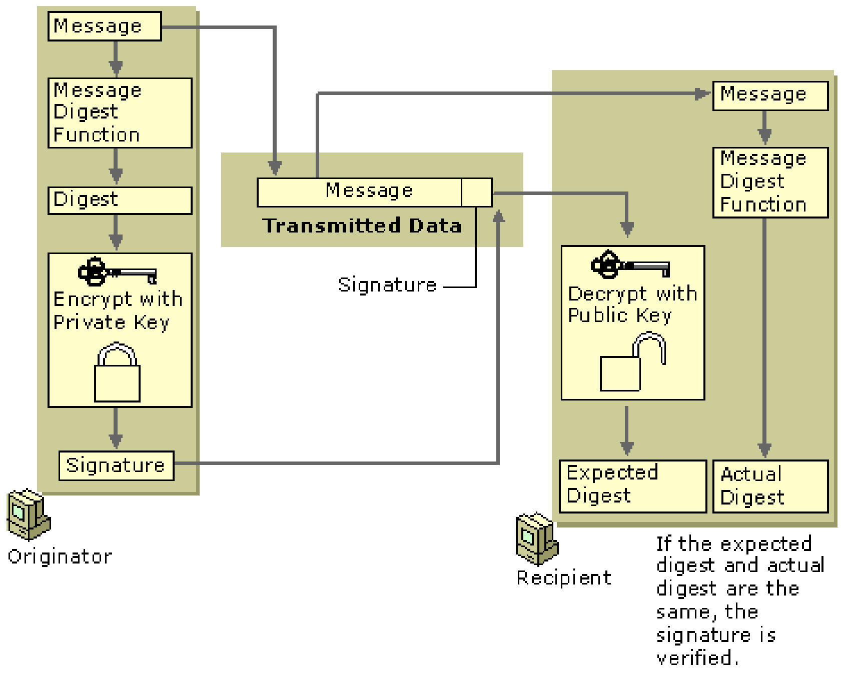
In this Assignment 3 part 2 you are going to create a simple program (a tool) to sign and verify a file using RSA Digital Signature scheme. Apart from that, a user also can use your program to generate MD-5 and SHA-1 hash values.

Your program should use a menu based for the options. Use the following sample output as guidelines:

```
******************************
*** Welcome to My Crypto Tool ***
******************************
Choose an option below:
1.   Generate RSA Signature for a file
2.   Verify a Signature of a file
3.   Create MD-5 Hash for a file
4.   Create SHA-1 Hash for a file
5.   Quit
```

The following is a guideline on how RSA Signature and Verification on a File works. A file can be of any type of file, while the Signature File stores the generated RSA Signature (done by your program).

During Verification process, the Generated Hash will be compared with the Hash value stored in the Signature file.



Notes:

a. Message is stored in a file.

b. Signature is stored in signature file. **No transmission of data required in this assignment.**

c. Message digest is using SHA-1

d. Public (PU)and Private key (PR) are generated using RSA algorithm

e. Originator – referring to the creation process of message with signature

f. Recipient – referring to the validation process of the message with the signature

As for the Menu options for Generating MD-5 and SHA-1 Hash, these options will request the user to enter a filename to generate the respective hash values. The output of the Hash a.k.a. Checksum (in Hex format) is directly appear on the screen.

You may use the Crypto++ library for this assignment.

Submission requirement:

You need to submit the following files for assessment:

- report.pdf, which contains flow chart of the overall working program, explanation of code segments according to menu options used during testing, generated output and screen captures should also be included in the report.
- Your source code should be submitted together with the report.pdf file in one folder. ZIP the folder and named it <your-name-CSC361-Assign3>.zip and submit it. Remember to put your name and student number in all source codes (comments header).
- Your source code folder should also include *.exe file for easier testing by the markers on Windows platform.

Assignments must be submitted electronically via Moodle submission link available. The hardcopy of report is **NOT** required.

Plagiarism

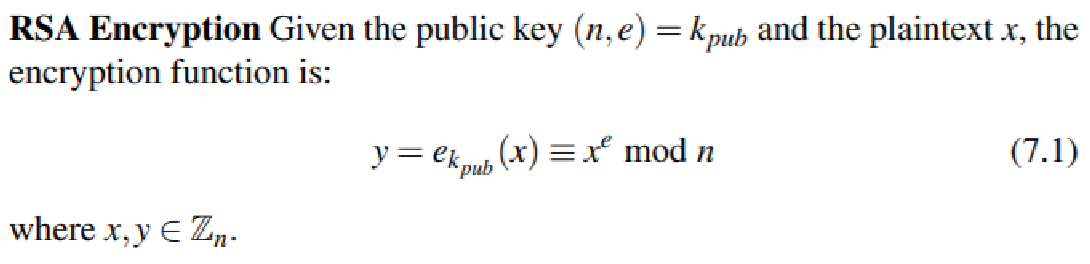
A plagiarised assignment will receive a zero mark (and penalised according to the university rules). Plagiarism detection software will be used.

Table of Contents

Basic overview of RSA

RSA is in Public Key infrastructure. It’s an asymmetric algorithm it will generate a pair of key which separate into Public Key and Private Key. The size of the encryption is at the range of “1024 – 4096” which is very secure.

RSA Encryption



RSA Encryption Given the public key $(n,e) = k_{pub}$ and the plaintext x, the encryption function is:

$$y = e_{k_{pub}}(x) \equiv x^e \bmod n \tag{7.1}$$

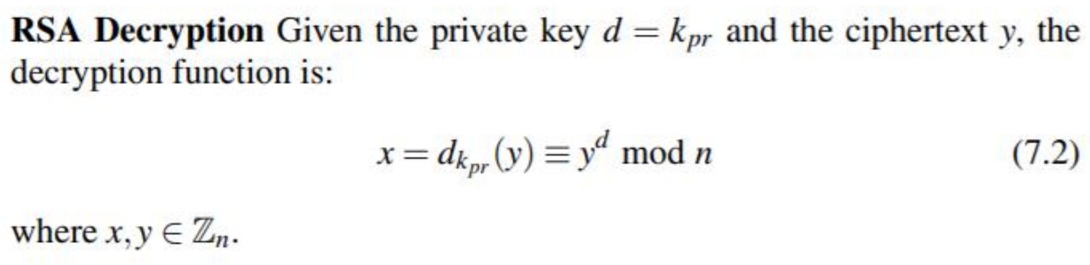
where $x, y \in \mathbb{Z}_n$.

Y is equivalent to the ciphertext.

$e_{k_{pub}}$ as you can see it uses “public key ” to perform encryption on the plaintext(x). Whereby it contains 2 elements in public key (n , e)

*Will discuss later about the values ‘n’ & ‘e’ *

RSA Decryption



RSA Decryption Given the private key $d = k_{pr}$ and the ciphertext y, the decryption function is:

$$x = d_{k_{pr}}(y) \equiv y^d \bmod n \tag{7.2}$$

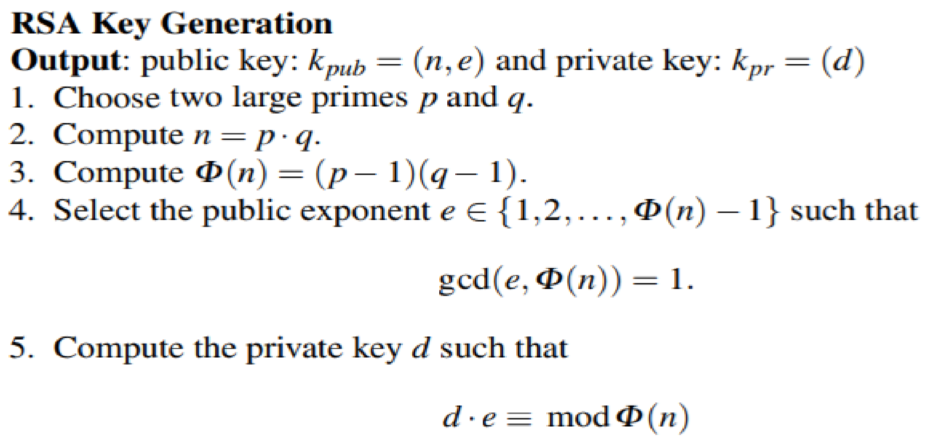
where $x, y \in \mathbb{Z}_n$.

X is equivalent to plaintext

$d_{k_{pr}}$ as you can see it uses “private key” to perform decryption on the ciphertext(y). Whereby it contains inverse of publickey which is ‘d’.

*Will discuss later about the value ‘d’ *

Key Generation on ‘n’ , ‘e’ and ‘d’



RSA Key Generation

Output: public key: $k_{pub} = (n, e)$ and private key: $k_{pr} = (d)$

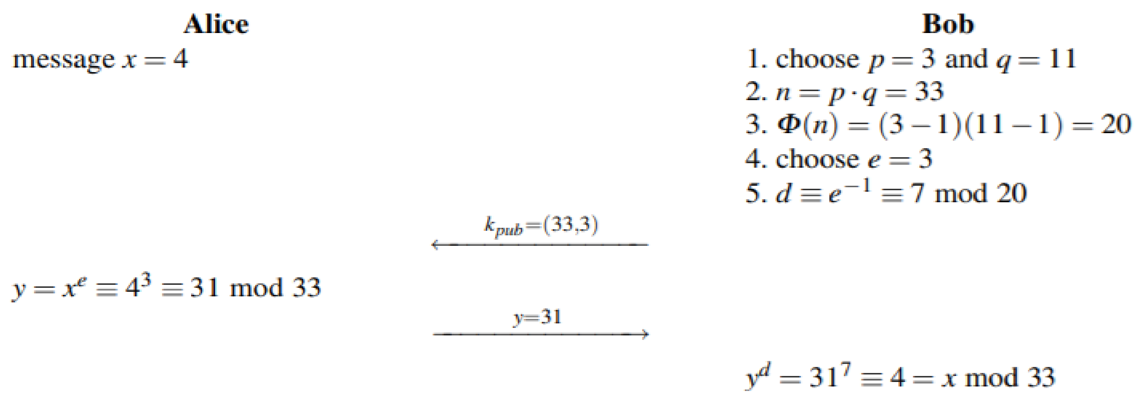
1. Choose two large primes p and q.
2. Compute $n = p \cdot q$.
3. Compute $\Phi(n) = (p-1)(q-1)$.
4. Select the public exponent $e \in \{1, 2, \ldots, \Phi(n) - 1\}$ such that

$$\gcd(e, \Phi(n)) = 1.$$

5. Compute the private key d such that

$$d \cdot e \equiv \bmod \Phi(n)$$

Sample of Encryption and Decryption in small value of ‘p’ & ’q’



Alice		**Bob**
message $x = 4$		1. choose $p = 3$ and $q = 11$
		2. $n = p \cdot q = 33$
		3. $\Phi(n) = (3-1)(11-1) = 20$
		4. choose $e = 3$
		5. $d \equiv e^{-1} \equiv 7 \bmod 20$
	$\xleftarrow{k_{pub}=(33,3)}$	
$y = x^e \equiv 4^3 \equiv 31 \bmod 33$		
	$\xrightarrow{y=31}$	
		$y^d = 31^7 \equiv 4 = x \bmod 33$

What happen if uses different pair of Private Key to perform the decryption ?

Let imagine that a PAIR of RSA (public and private key separate into a puzzle)

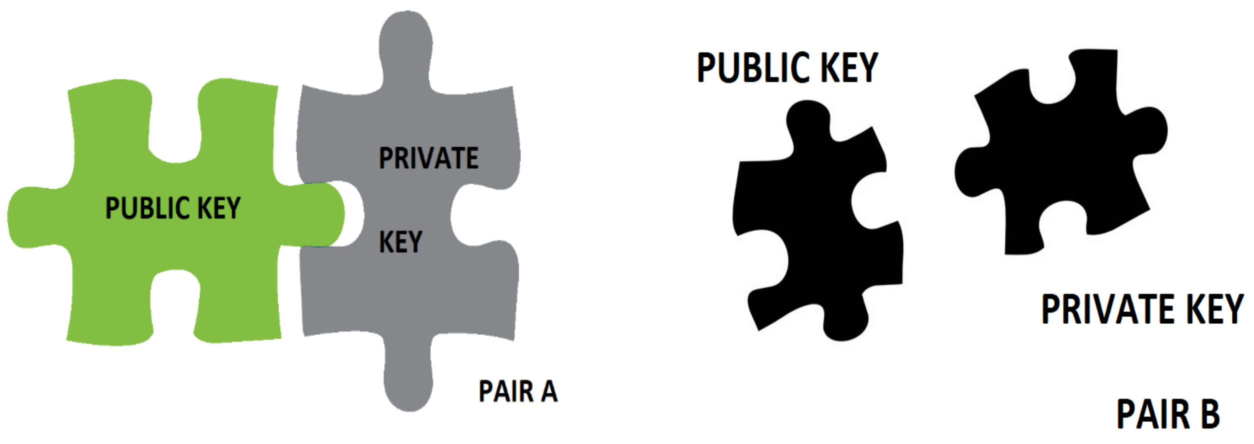
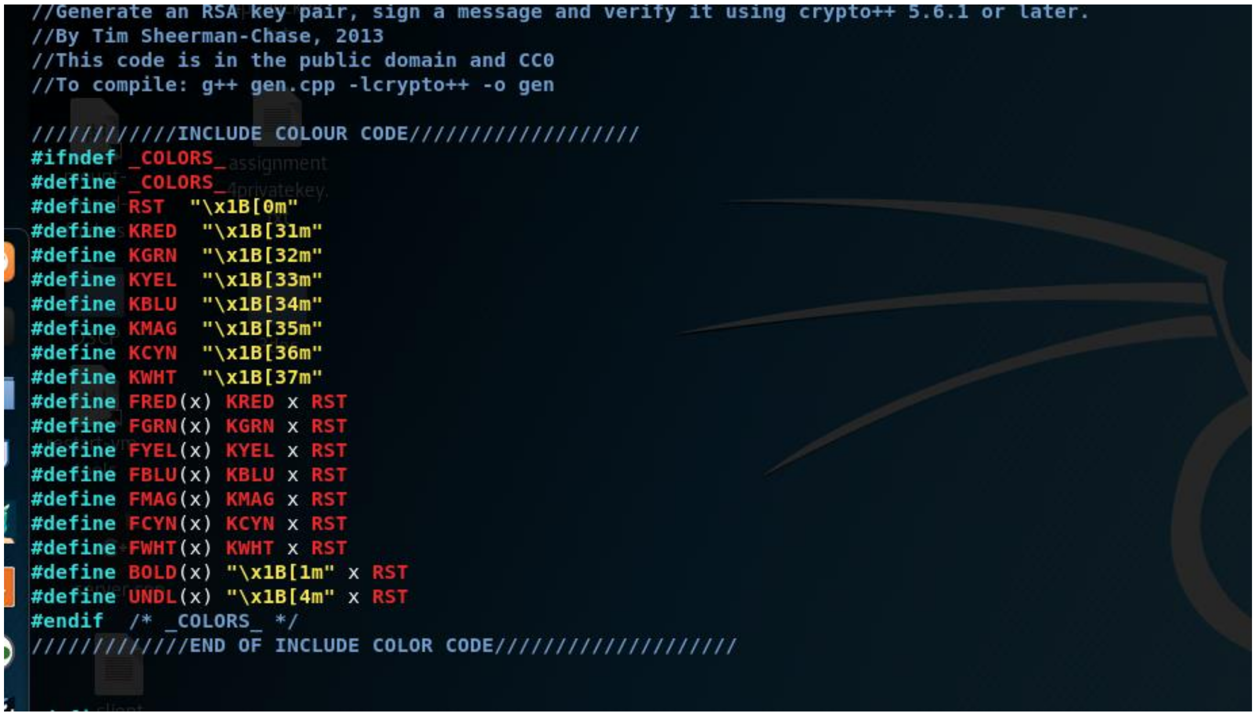


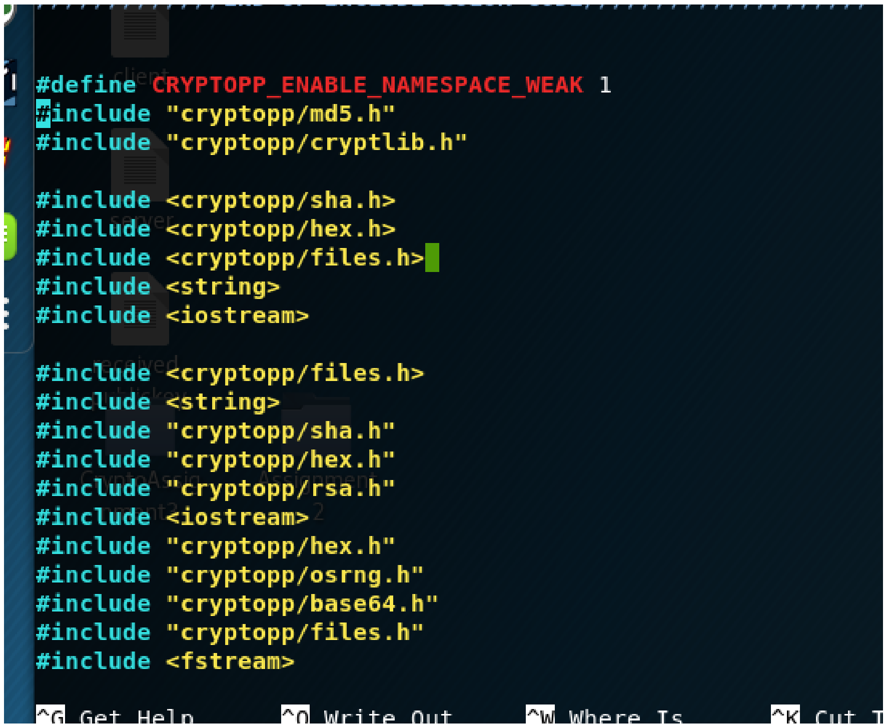
Image the Alice using (Public Key PAIR A) to encrypt the plaintext and using Private Key of PAIR B to perform the decryption the puzzle will not matched and it failed to perform the decryption.

Include Color code



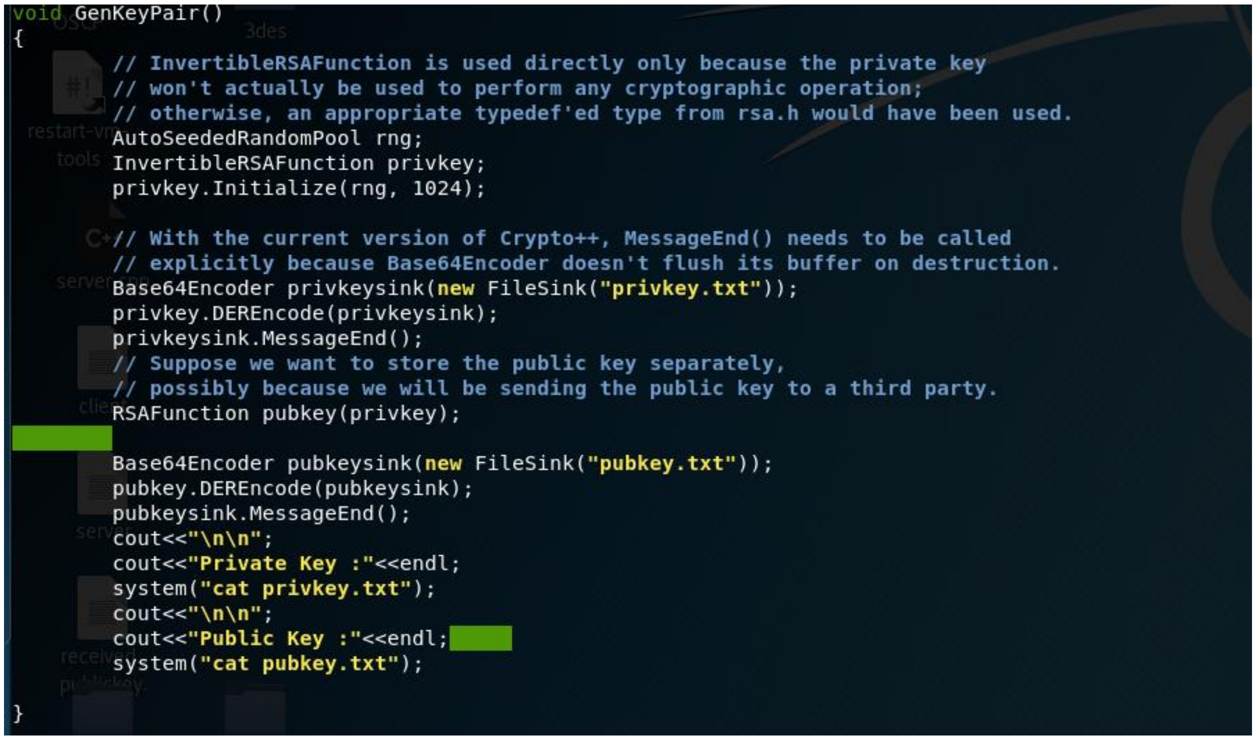
```
//Generate an RSA key pair, sign a message and verify it using crypto++ 5.6.1 or later.
//By Tim Sheerman-Chase, 2013
//This code is in the public domain and CC0
//To compile: g++ gen.cpp -lcrypto++ -o gen

////////////INCLUDE COLOUR CODE//////////////////
#ifndef _COLORS_
#define _COLORS_
#define RST  "\x1B[0m"
#define KRED  "\x1B[31m"
#define KGRN  "\x1B[32m"
#define KYEL  "\x1B[33m"
#define KBLU  "\x1B[34m"
#define KMAG  "\x1B[35m"
#define KCYN  "\x1B[36m"
#define KWHT  "\x1B[37m"
#define FRED(x) KRED x RST
#define FGRN(x) KGRN x RST
#define FYEL(x) KYEL x RST
#define FBLU(x) KBLU x RST
#define FMAG(x) KMAG x RST
#define FCYN(x) KCYN x RST
#define FWHT(x) KWHT x RST
#define BOLD(x) "\x1B[1m" x RST
#define UNDL(x) "\x1B[4m" x RST
#endif  /* _COLORS_ */
////////////END OF INCLUDE COLOR CODE//////////////////
```

Include library and crypto++ library



```
#define CRYPTOPP_ENABLE_NAMESPACE_WEAK 1
#include "cryptopp/md5.h"
#include "cryptopp/cryptlib.h"

#include <cryptopp/sha.h>
#include <cryptopp/hex.h>
#include <cryptopp/files.h>
#include <string>
#include <iostream>

#include <cryptopp/files.h>
#include <string>
#include "cryptopp/sha.h"
#include "cryptopp/hex.h"
#include "cryptopp/rsa.h"
#include <iostream>
#include "cryptopp/hex.h"
#include "cryptopp/osrng.h"
#include "cryptopp/base64.h"
#include "cryptopp/files.h"
#include <fstream>

^G Get Help      ^O Write Out      ^W Where Is      ^K Cut T
```

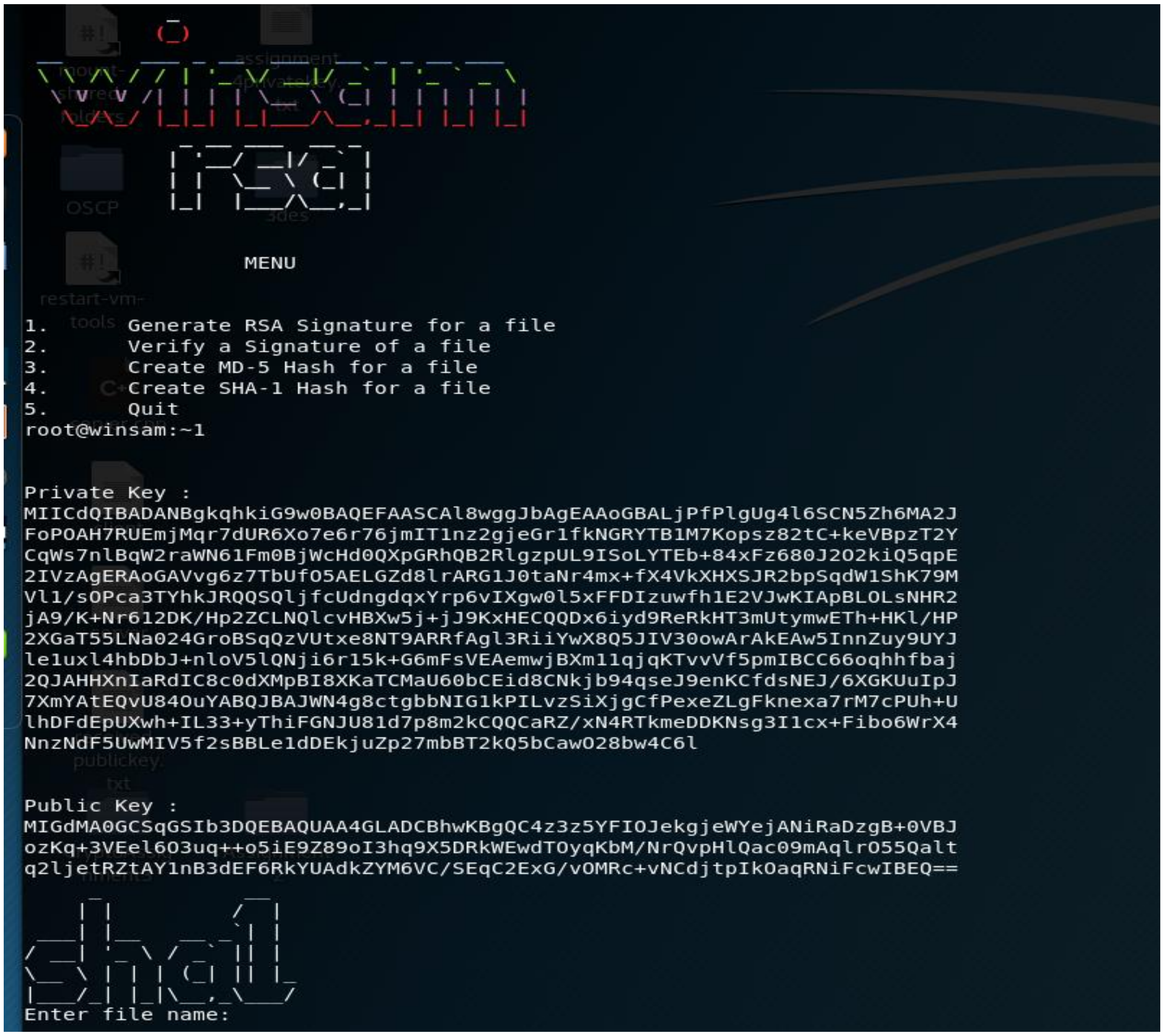
GenKeyPair() Function



```
void GenKeyPair()
{
        // InvertibleRSAFunction is used directly only because the private key
        // won't actually be used to perform any cryptographic operation;
        // otherwise, an appropriate typedef'ed type from rsa.h would have been used.
        AutoSeededRandomPool rng;
        InvertibleRSAFunction privkey;
        privkey.Initialize(rng, 1024);

        // With the current version of Crypto++, MessageEnd() needs to be called
        // explicitly because Base64Encoder doesn't flush its buffer on destruction.
        Base64Encoder privkeysink(new FileSink("privkey.txt"));
        privkey.DEREncode(privkeysink);
        privkeysink.MessageEnd();
        // Suppose we want to store the public key separately,
        // possibly because we will be sending the public key to a third party.
        RSAFunction pubkey(privkey);

        Base64Encoder pubkeysink(new FileSink("pubkey.txt"));
        pubkey.DEREncode(pubkeysink);
        pubkeysink.MessageEnd();
        cout<<"\n\n";
        cout<<"Private Key :"<<endl;
        system("cat privkey.txt");
        cout<<"\n\n";
        cout<<"Public Key :"<<endl;
        system("cat pubkey.txt");

}
```

It generate a 1024 bits of RSA pair and encode it using base64 and save as base64(publickey) and base64(privatekey) into 2 files which are “pubkey.txt” and “privkey.txt” using ‘FileSink’. After that it will display the content of ‘privkey.txt’ and ‘pubkey.txt’ files.

Sample output GenKeyPair()



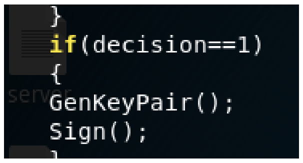
```
                MENU

1.      Generate RSA Signature for a file
2.      Verify a Signature of a file
3.      Create MD-5 Hash for a file
4.      Create SHA-1 Hash for a file
5.      Quit
root@winsam:~1


Private Key :
MIICdQIBADANBgkqhkiG9w0BAQEFAASCAl8wggJbAgEAAoGBALjPfPlgUg4l6SCN5Zh6MA2J
FoPOAH7RUEmjMqr7dUR6Xo7e6r76jmIT1nz2gjeGr1fkNGRYTB1M7Kopsz82tC+keVBpzT2Y
CqWs7nlBqW2raWN61Fm0BjWcHd0QXpGRhQB2RlgzpUL9ISoLYTEb+84xFz680J2O2kiQ5qpE
2IVzAgERAoGAVvg6z7TbUfO5AELGZd8lrARG1J0taNr4mx+fX4VkXHXSJR2bpSqdW1ShK79M
Vl1/sOPca3TYhkJRQQSQljfcUdngdqxYrp6vIXgw0l5xFFDIzuwfh1E2VJwKIApBLOLsNHR2
jA9/K+Nr612DK/Hp2ZCLNQlcvHBXw5j+jJ9KxHECQQDx6iyd9ReRkHT3mUtymwETh+HKl/HP
2XGaT55LNa024GroBSqQzVUtxe8NT9ARRfAgl3RiiYwX8Q5JIV30owArAkEAw5InnZuy9UYJ
le1uxl4hbDbJ+nloV5lQNji6r15k+G6mFsVEAemwjBXm11qjqKTvvVf5pmIBCC66oqhhfbaj
2QJAHHXnIaRdIC8c0dXMpBI8XKaTCMaU60bCEid8CNkjb94qseJ9enKCfdsNEJ/6XGKUuIpJ
7XmYAtEQvU84OuYABQJBAJWN4g8ctgbbNIG1kPILvzSiXjgCfPexeZLgFknexa7rM7cPUh+U
lhDFdEpUXwh+IL33+yThiFGNJU81d7p8m2kCQQCaRZ/xN4RTkmeDDKNsg3I1cx+Fibo6WrX4
NnzNdF5UwMIV5f2sBBLe1dDEkjuZp27mbBT2kQ5bCawO28bw4C6l


Public Key :
MIGdMA0GCSqGSIb3DQEBAQUAA4GLADCBhwKBgQC4z3z5YFIOJekgjeWYejANiRaDzgB+0VBJ
ozKq+3VEel6O3uq++o5iE9Z89oI3hq9X5DRkWEwdTOyqKbM/NrQvpHlQac09mAqlrO55Qalt
q2ljetRZtAY1nB3dEF6RkYUAdkZYM6VC/SEqC2ExG/vOMRc+vNCdjtpIkOaqRNiFcwIBEQ==

Enter file name:
```

When user click selection '1' on the menu page which is the GenKeyPair() function it automatically create a pair of key and encode with Base64 and 'dump' the value of public key and private key from these 2 different files 'privkey.txt' and 'pubkey.txt'.

Following command 'cat pubkey.txt' and 'cat privkey.txt' which to display the value stored in the files.

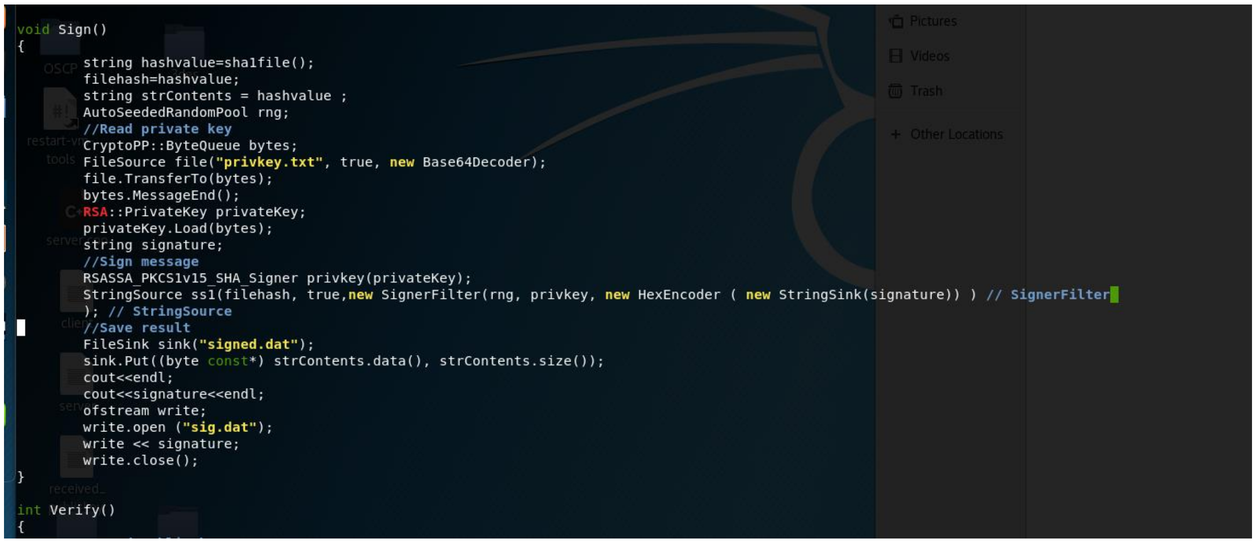
Files	VALUE
privkey.txt	Private Key
Pubkey.txt	Public Key

After finished the GenKeyPair() it will run Sign() , it will see this in main() function



```
}
if(decision==1)
{
GenKeyPair();
Sign();
```

Sign() function



```
void Sign()
{
        string hashvalue=sha1file();
        filehash=hashvalue;
        string strContents = hashvalue ;
        AutoSeededRandomPool rng;
        //Read private key
        CryptoPP::ByteQueue bytes;
        FileSource file("privkey.txt", true, new Base64Decoder);
        file.TransferTo(bytes);
        bytes.MessageEnd();
        RSA::PrivateKey privateKey;
        privateKey.Load(bytes);
        string signature;
        //Sign message
        RSASSA_PKCS1v15_SHA_Signer privkey(privateKey);
        StringSource ss1(filehash, true,new SignerFilter(rng, privkey, new HexEncoder ( new StringSink(signature)) ) // SignerFilter
        ); // StringSource
        //Save result
        FileSink sink("signed.dat");
        sink.Put((byte const*) strContents.data(), strContents.size());
        cout<<endl;
        cout<<signature<<endl;
        ofstream write;
        write.open ("sig.dat");
        write << signature;
        write.close();
}

int Verify()
{
```

At the beginning of the Sign() function it will ‘perform a sha1file() function’ to grab the return return into a string variable which is our ‘plaintext/message’. In our scenario the plaintext/message is the SHA-1 checksum value from any files input by user.

Next will grab the privatekey from ‘privkey.txt’ , and decode it using Base64Decoder

```
        FileSource file("privkey.txt", true, new Base64Decoder);
```

And load it into ‘bytes’

```
file.TransferTo(bytes);
        bytes.MessageEnd();
        RSA::PrivateKey privateKey;
        privateKey.Load(bytes);
        string signature;
        //Sign message
        RSASSA_PKCS1v15_SHA_Signer privkey(privateKey);
```

And it start the sign process and store into a variable ‘signature’ and encode using HexEncoder

```
StringSource ss1(filehash, true,new SignerFilter(rng, privkey, new
HexEncoder ( new StringSink(signature)) ) // SignerFilter
);
// StringSource
//Save result
```

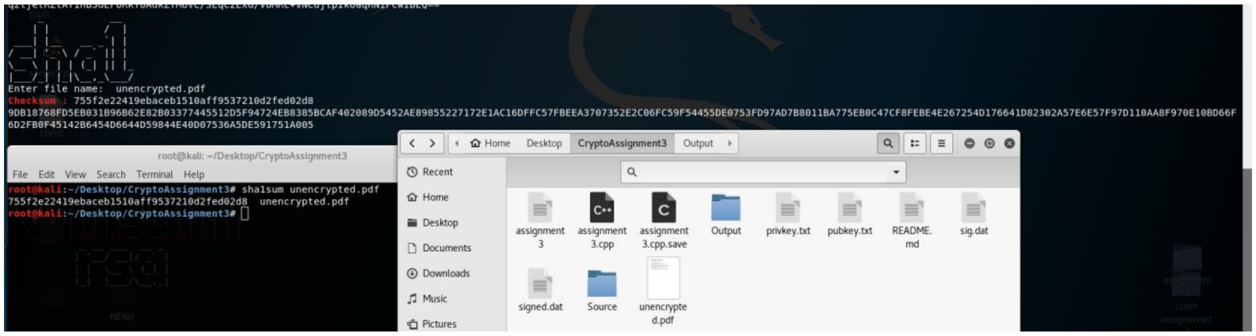
and save the ‘plaintext/message && signature’ into 2 different ‘.dat’ files

```
        FileSink sink("signed.dat");
         sink.Put((byte const*) strContents.data(), strContents.size());
         cout<<endl;
         cout<<signature<<endl;
         ofstream write;
         write.open ("sig.dat");
         write << signature;
         write.close();
```

First strContents.data() is the ‘plaintext’ which is the 40 lengths character (SHA-1) which will store in signed.dat

Sample output of Sign()

Whereby the ‘signature’ is store into ‘sig.dat’ .



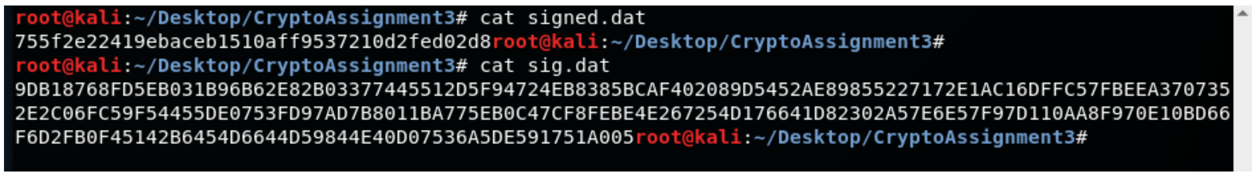
```
Enter file name:  unencrypted.pdf
Checksum : 755f2e22419ebaceb1510aff9537210d2fed02d8
9DB18768FD5EB031B96B62E82B03377445512D5F94724EB8385BCAF402089D5452AE89855227172E1AC16DFFC57FBEEA3707352E2C06FC59F54455DE0753FD97AD7B8011BA775EB0C47CF8FEBE4E267254D176641D82302A57E6E57F97D110AA8F970E10BD66F
6D2FB0F45142B6454D6644D59844E40D07536A5DE591751A005
```

```
root@kali:~/Desktop/CryptoAssignment3# sha1sum unencrypted.pdf
755f2e22419ebaceb1510aff9537210d2fed02d8  unencrypted.pdf
root@kali:~/Desktop/CryptoAssignment3#
```

After user have input the filename ‘unencrypted.pdf’ it will show the signature in HEX format and it checksum . I do have include another terminal just to show that the checksum value is “matched” using tool that embedded in Kali Linux called *sha1sum* .

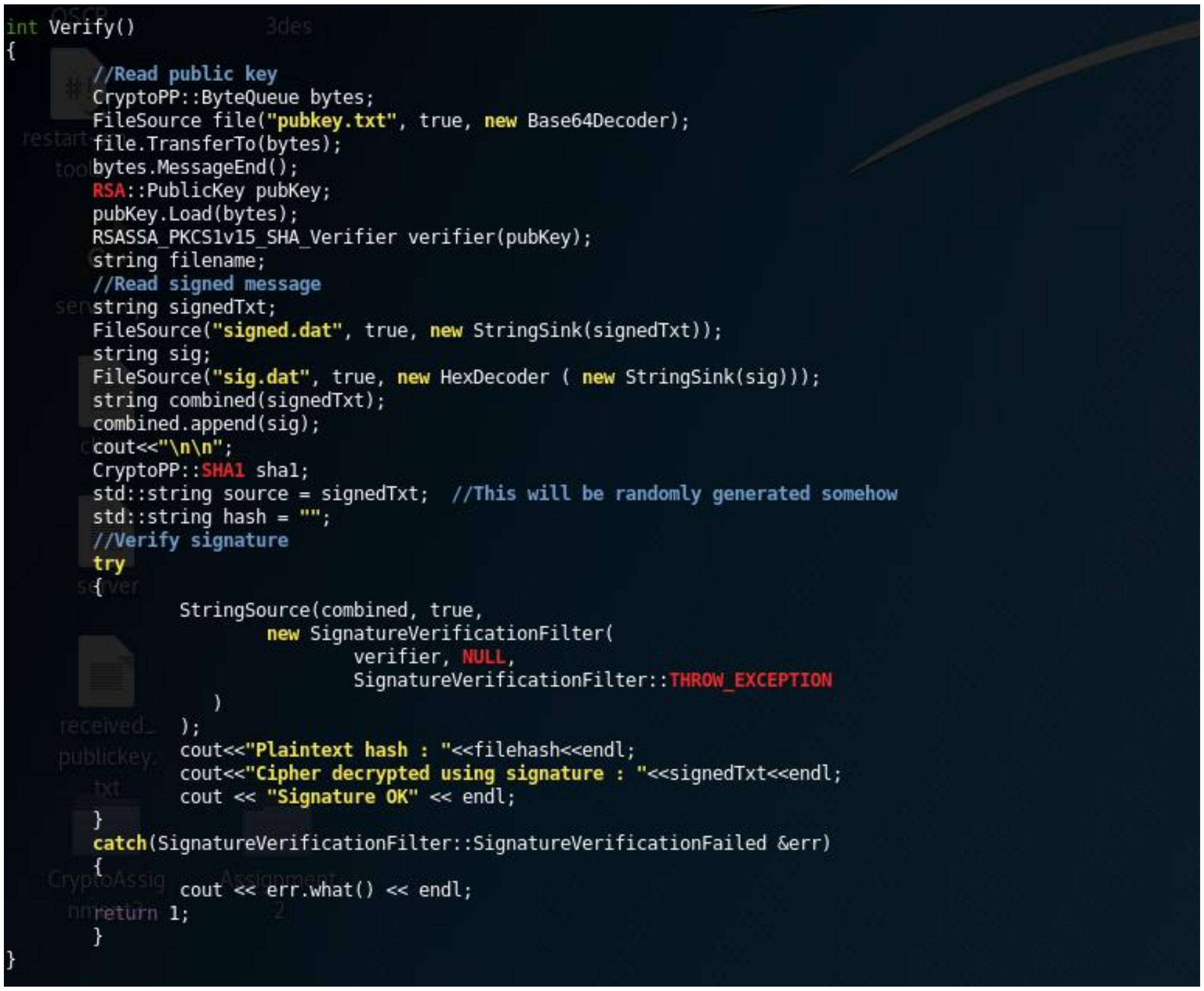
As you can see on the “Files” it have created 4 different files ‘privkey.txt’ , ‘pubkey.txt’ , ‘sig.dat’ and ‘signed.dat’.

Files	VALUE
privkey.txt	Private Key
Pubkey.txt	Public Key
Sig.dat	Signature value
Signed.dat	Plaintext/message in our scenario it’s a 40length of checksum value created from SHA1FILE()



```
root@kali:~/Desktop/CryptoAssignment3# cat signed.dat
755f2e22419ebaceb1510aff9537210d2fed02d8root@kali:~/Desktop/CryptoAssignment3#
root@kali:~/Desktop/CryptoAssignment3# cat sig.dat
9DB18768FD5EB031B96B62E82B03377445512D5F94724EB8385BCAF402089D5452AE89855227172E1AC16DFFC57FBEEA370735
2E2C06FC59F54455DE0753FD97AD7B8011BA775EB0C47CF8FEBE4E267254D176641D82302A57E6E57F97D110AA8F970E10BD66
F6D2FB0F45142B6454D6644D59844E40D07536A5DE591751A005root@kali:~/Desktop/CryptoAssignment3#
```

Verify() function



```
int Verify()
{
        //Read public key
        CryptoPP::ByteQueue bytes;
        FileSource file("pubkey.txt", true, new Base64Decoder);
        file.TransferTo(bytes);
        bytes.MessageEnd();
        RSA::PublicKey pubKey;
        pubKey.Load(bytes);
        RSASSA_PKCS1v15_SHA_Verifier verifier(pubKey);
        string filename;
        //Read signed message
        string signedTxt;
        FileSource("signed.dat", true, new StringSink(signedTxt));
        string sig;
        FileSource("sig.dat", true, new HexDecoder ( new StringSink(sig)));
        string combined(signedTxt);
        combined.append(sig);
        cout<<"\n\n";
        CryptoPP::SHA1 sha1;
        std::string source = signedTxt;  //This will be randomly generated somehow
        std::string hash = "";
        //Verify signature
        try
        {
                StringSource(combined, true,
                        new SignatureVerificationFilter(
                                verifier, NULL,
                                SignatureVerificationFilter::THROW_EXCEPTION
                    )
                );
                cout<<"Plaintext hash : "<<filehash<<endl;
                cout<<"Cipher decrypted using signature : "<<signedTxt<<endl;
                cout << "Signature OK" << endl;
        }
        catch(SignatureVerificationFilter::SignatureVerificationFailed &err)
        {
                cout << err.what() << endl;
        return 1;
        }
}
```

Firstly, it use public key to perform the decryption process.

So it will grab the publickey using FileSource and after it grab it perform the decode using Base64Decoder.

```
FileSource file("pubkey.txt", true, new Base64Decoder);
```

And store into bytes

```
bytes.MessageEnd();
        RSA::PublicKey pubKey;
        pubKey.Load(bytes);
        RSASSA_PKCS1v15_SHA_Verifier verifier(pubKey);
```

And create an object of verifier to perform RSASSA_PKCS1v15_SHA_Verifier.

And it grab the signature and ‘plaintext/message’ from the ‘signed.dat and sig.dat’ and store into 2 variables ‘signedTxt and sig’ and combined together.

```
//Read signed message
        string signedTxt;
        FileSource("signed.dat", true, new StringSink(signedTxt));
        string sig;
        FileSource("sig.dat", true, new HexDecoder ( new
StringSink(sig)));
        string combined(signedTxt);
        combined.append(sig);
```

Next, after we have combine the “message/plaintext + signature” it will go to next lines of code which is to perform process of verifying

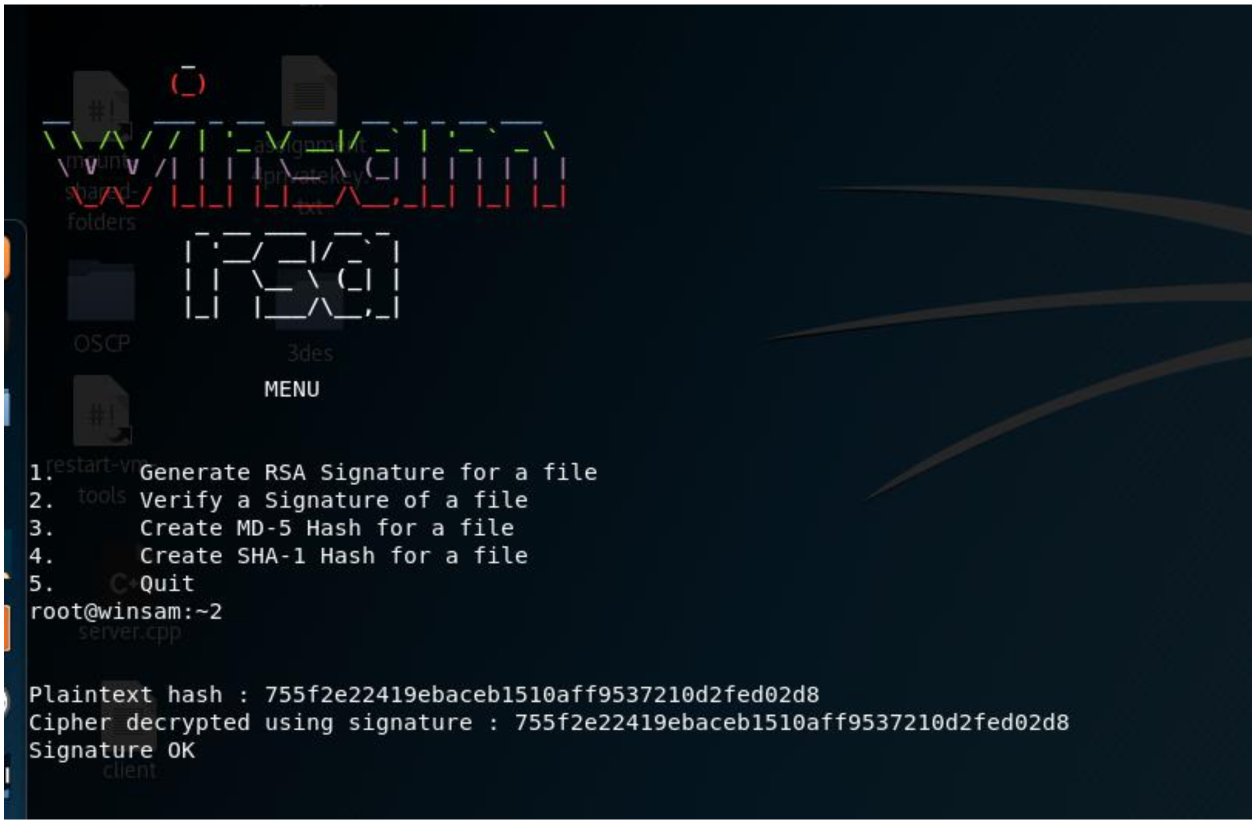
```
       CryptoPP::SHA1 sha1;
        std::string source = signedTxt;  //This will be randomly generated
somehow
        std::string hash = "";
        //Verify signature
        try
        { StringSource(combined, true,new
SignatureVerificationFilter(verifier, NULL,
SignatureVerificationFilter::THROW_EXCEPTION));
                cout<<"Plaintext hash : "<<filehash<<endl;
                cout<<"Cipher decrypted using signature :
"<<signedTxt<<endl;
                cout << "Signature OK" << endl;
        }
        catch(SignatureVerificationFilter::SignatureVerificationFailed
&err)
        {      cout << err.what() << endl;
        return 1;
        }
```

Try will try to use the “combined” variable to do verify process

```
   StringSource(combined, true,
                         new SignatureVerificationFilter(
                                 verifier, NULL,
SignatureVerificationFilter::THROW_EXCEPTION
                      )
                   );
```

StringSource take variable ‘combined’ which is the value of “message/plaintext + signature” with verifier object ‘in our scenario the verifier is a publickey that we generate from “GenKeyPair()” ’.

Validate the publickey



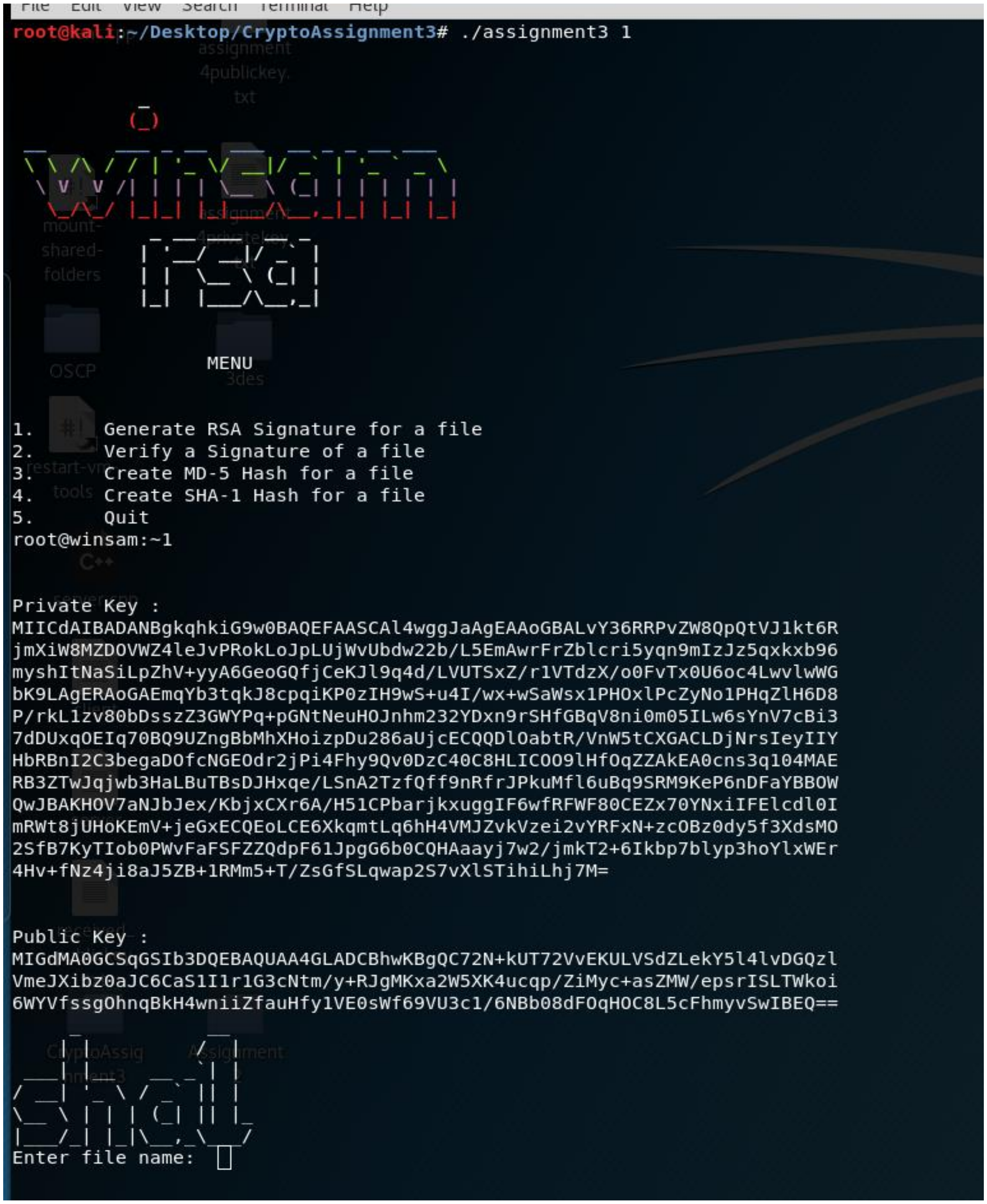
```
                 MENU


1.      Generate RSA Signature for a file
2.      Verify a Signature of a file
3.      Create MD-5 Hash for a file
4.      Create SHA-1 Hash for a file
5.      Quit
root@winsam:~2


Plaintext hash : 755f2e22419ebaceb1510aff9537210d2fed02d8
Cipher decrypted using signature : 755f2e22419ebaceb1510aff9537210d2fed02d8
Signature OK
```

If using the correct public key to decrypt the signature it will allow us to decrypt, but since this is an assignment will have to prove that what if the public key is incorrect.

Step 1: Backup current “pubkey.txt”, we copy pubkey.txt and saved as “backupkey.txt”

```
cp pubkey.txt backupkey.txt
```

Step 2 : Generate a new pair key, using selection ‘1’ from the menu page



```
root@kali:~/Desktop/CryptoAssignment3# ./assignment3 1

                 MENU

1.      Generate RSA Signature for a file
2.      Verify a Signature of a file
3.      Create MD-5 Hash for a file
4.      Create SHA-1 Hash for a file
5.      Quit
root@winsam:~1


Private Key :
MIICdAIBADANBgkqhkiG9w0BAQEFAASCAl4wggJaAgEAAoGBALvY36RRPvZW8QpQtVJ1kt6R
jmXiW8MZDOVWZ4leJvPRokLoJpLUjWvUbdw22b/L5EmAwrFrZblcri5yqn9mIzJz5qxkxb96
myshItNaSiLpZhV+yyA6GeoGQfjCeKJl9q4d/LVUTSxZ/r1VTdzX/o0FvTx0U6oc4LwvlwWG
bK9LAgERAoGAEmqYb3tqkJ8cpqiKP0zIH9wS+u4I/wx+wSaWsx1PHOxlPcZyNo1PHqZlH6D8
P/rkL1zv80bDsszZ3GWYPq+pGNtNeuHOJnhm232YDxn9rSHfGBqV8ni0m05ILw6sYnV7cBi3
7dDUxqOEIq70BQ9UZngBbMhXHoizpDu286aUjcECQQDlOabtR/VnW5tCXGACLDjNrsIeyIIY
HbRBnI2C3begaDOfcNGEOdr2jPi4Fhy9Qv0DzC40C8HLICOO9lHfOqZZAkEA0cns3q104MAE
RB3ZTwJqjwb3HaLBuTBsDJHxqe/LSnA2TzfQff9nRfrJPkuMfl6uBq9SRM9KeP6nDFaYBBOW
QwJBAKHOV7aNJbJex/KbjxCXr6A/H51CPbarjkxuggIF6wfRFWF80CEZx70YNxiIFElcdl0I
mRWt8jUHoKEmV+jeGxECQEoLCE6XkqmtLq6hH4VMJZvkVzei2vYRFxN+zcOBz0dy5f3XdsMO
2SfB7KyTIob0PWvFaFSFZZQdpF61JpgG6b0CQHAaayj7w2/jmkT2+6Ikbp7blyp3hoYlxWEr
4Hv+fNz4ji8aJ5ZB+1RMm5+T/ZsGfSLqwap2S7vXlSTihiLhj7M=


Public Key :
MIGdMA0GCSqGSIb3DQEBAQUAA4GLADCBhwKBgQC72N+kUT72VvEKULVSdZLekY5l4lvDGQzl
VmeJXibz0aJC6CaS1I1r1G3cNtm/y+RJgMKxa2W5XK4ucqp/ZiMyc+asZMW/epsrISLTWkoi
6WYVfssgOhnqBkH4wniiZfauHfy1VE0sWf69VU3c1/6NBb08dFOqHOC8L5cFhmyvSwIBEQ==

Enter file name:
```

Now we have generate a new pair of RSA key

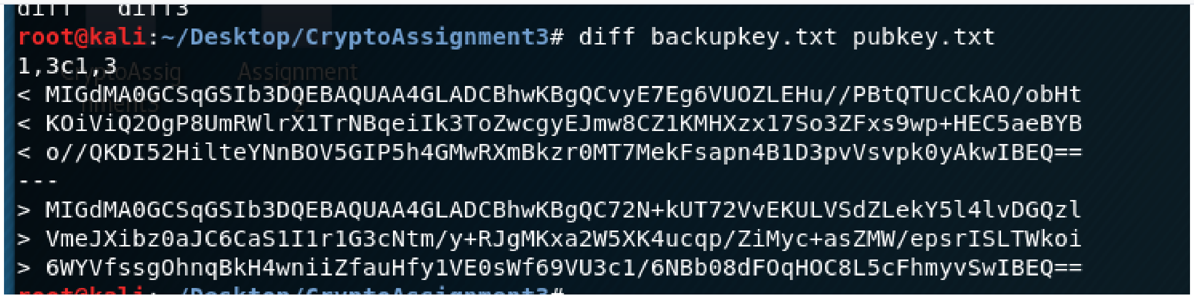
Step 3:

```
Enter file name:  ^C
root@kali:~/Desktop/CryptoAssignment3#
```

We quit the program using CTRL + C to quit the program.

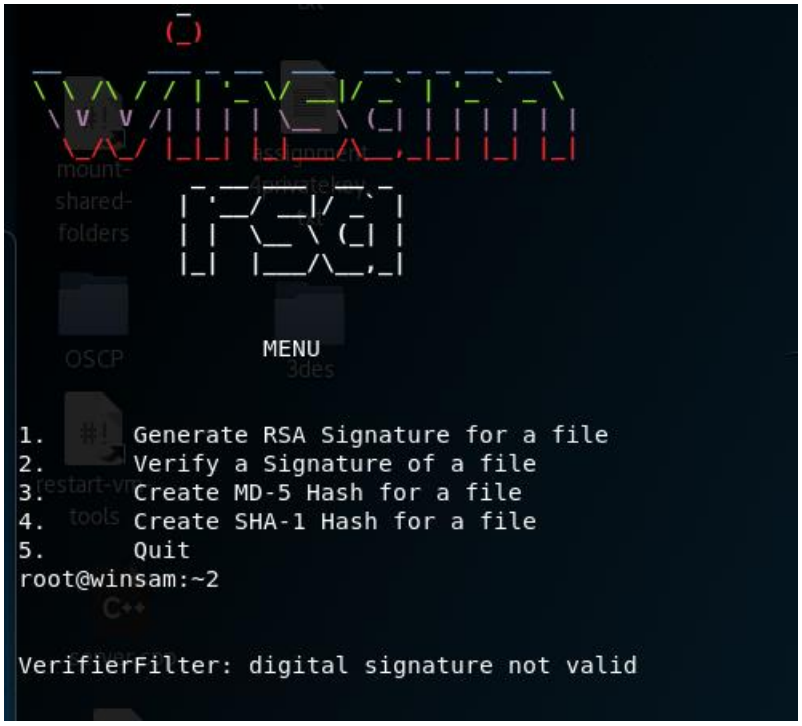
Step 4: Look at the different between the 2 files ‘pubkey.txt’ and ‘backupkey.txt’

```
diff backupkey.txt pubkey.txt
```



```
root@kali:~/Desktop/CryptoAssignment3# diff backupkey.txt pubkey.txt
1,3c1,3
< MIGdMA0GCSqGSIb3DQEBAQUAA4GLADCBhwKBgQCvyE7Eg6VUOZLEHu//PBtQTUcCkAO/obHt
< KOiViQ2OgP8UmRWlrX1TrNBqeiIk3ToZwcgyEJmw8CZ1KMHXzx17So3ZFxs9wp+HEC5aeBYB
< o//QKDI52HilteYNnBOV5GIP5h4GMwRXmBkzr0MT7MekFsapn4B1D3pvVsvpk0yAkwIBEQ==
---
> MIGdMA0GCSqGSIb3DQEBAQUAA4GLADCBhwKBgQC72N+kUT72VvEKULVSdZLekY5l4lvDGQzl
> VmeJXibz0aJC6CaS1I1r1G3cNtm/y+RJgMKxa2W5XK4ucqp/ZiMyc+asZMW/epsrISLTWkoi
> 6WYVfssgOhnqBkH4wniiZfauHfy1VE0sWf69VU3c1/6NBb08dFOqHOC8L5cFhmyvSwIBEQ==
```

Step 5 : Launch the program , and select ‘2’ from the menu page



```
                MENU


1.      Generate RSA Signature for a file
2.      Verify a Signature of a file
3.      Create MD-5 Hash for a file
4.      Create SHA-1 Hash for a file
5.      Quit
root@winsam:~2


VerifierFilter: digital signature not valid
```

Now it state that the **digital signature not valid** , because we’re using the different pubkey to decrypt .

Quick Revision on PAIR_A and PAIR_B on RSA Decryption and why it failed using different key

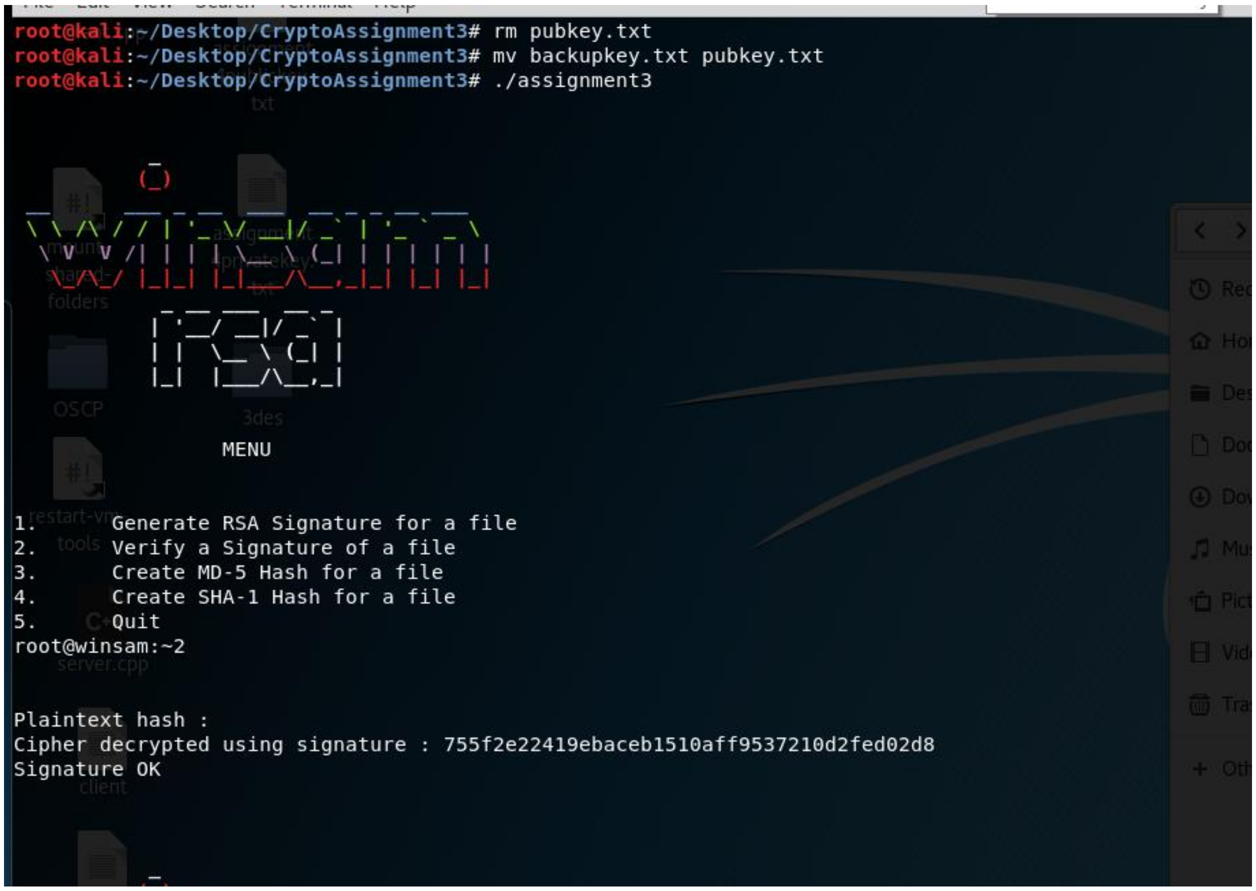
a.pdf is sign using PRIVATEKEY_A and it able to decrypt with PUBLICKEY_A **ONLY** , but now the 'unencrypted.pdf' is signed with 'PRIVATEKEY_A' but decrypt with 'PUBLICKEY_B' therefore it failed to perform the decryption process.

Step 5 : Replace back the "backupkey.txt" to "pubkey.txt"

```
rm pubkey.txt
mv backupkey.txt pubkey.txt
```

We deleted the pubkey.txt which is 'PUBLICKEY_B' and rename the backupkey.txt to pubkey.txt 'PUBLICKEY_A'

Step 6 : Launch the program and select '2' from the menu

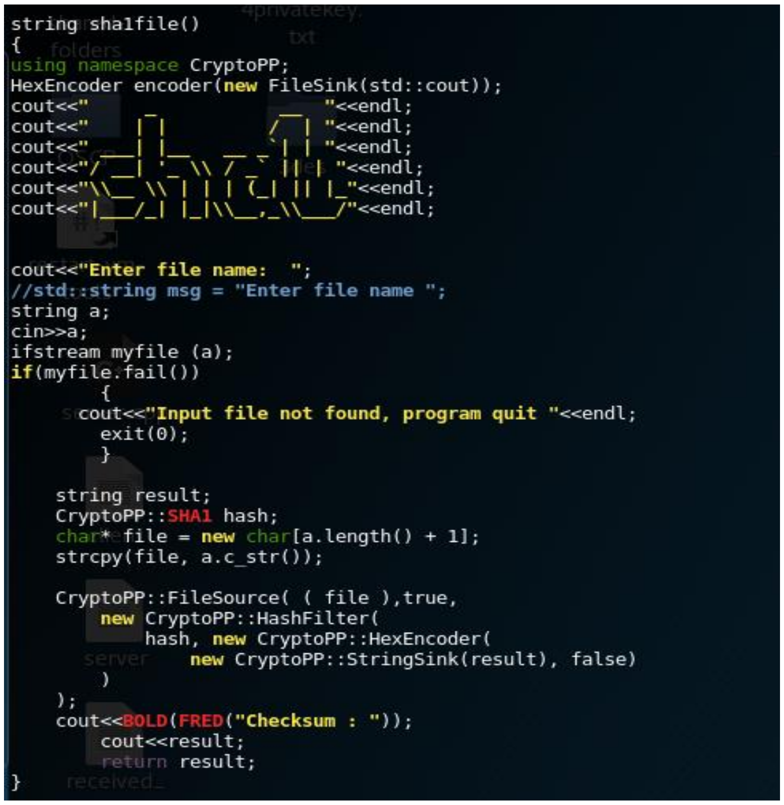


```
root@kali:~/Desktop/CryptoAssignment3# rm pubkey.txt
root@kali:~/Desktop/CryptoAssignment3# mv backupkey.txt pubkey.txt
root@kali:~/Desktop/CryptoAssignment3# ./assignment3

                    MENU

1.      Generate RSA Signature for a file
2.      Verify a Signature of a file
3.      Create MD-5 Hash for a file
4.      Create SHA-1 Hash for a file
5.      Quit
root@winsam:~2


Plaintext hash :
Cipher decrypted using signature : 755f2e22419ebaceb1510aff9537210d2fed02d8
Signature OK
```

Process of verifying is OK! Because we use the correct pair of RSA to decrypt.

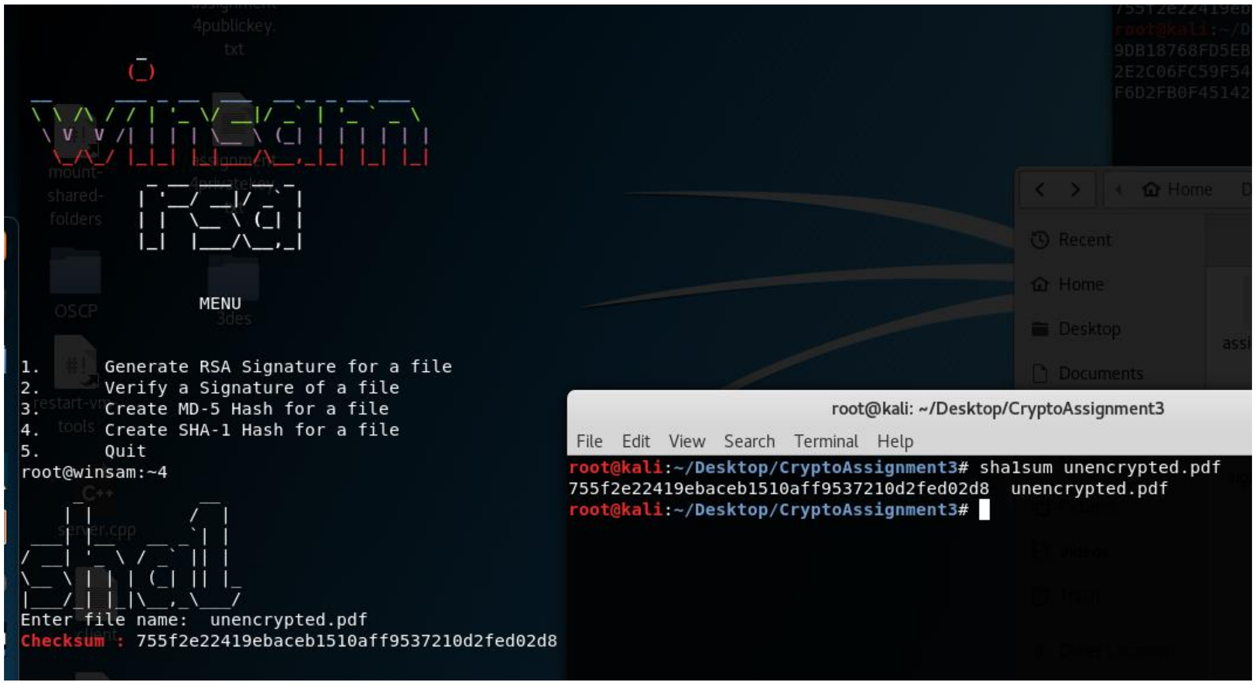
SHA1file()

Below function is the function to perform SHA-1 calculation on any input file



```cpp
string sha1file()
{
using namespace CryptoPP;
HexEncoder encoder(new FileSink(std::cout));
cout<<"       _          ___  "<<endl;
cout<<"      | |        /_ |  "<<endl;
cout<<"   ___| |__   __ _`| | "<<endl;
cout<<"/ ___| '_ \\ / _` ||| "<<endl;
cout<<"\\__ \\ | | | (_| || |_"<<endl;
cout<<"|___/_| |_|\\__,_\\___/"<<endl;


cout<<"Enter file name:  ";
//std::string msg = "Enter file name ";
string a;
cin>>a;
ifstream myfile (a);
if(myfile.fail())
        {
      cout<<"Input file not found, program quit "<<endl;
        exit(0);
        }

    string result;
    CryptoPP::SHA1 hash;
    char* file = new char[a.length() + 1];
    strcpy(file, a.c_str());

    CryptoPP::FileSource( ( file ),true,
        new CryptoPP::HashFilter(
            hash, new CryptoPP::HexEncoder(
                new CryptoPP::StringSink(result), false)
        )
    );
    cout<<BOLD(FRED("Checksum : "));
        cout<<result;
        return result;
}
```

Sample output of SHA1



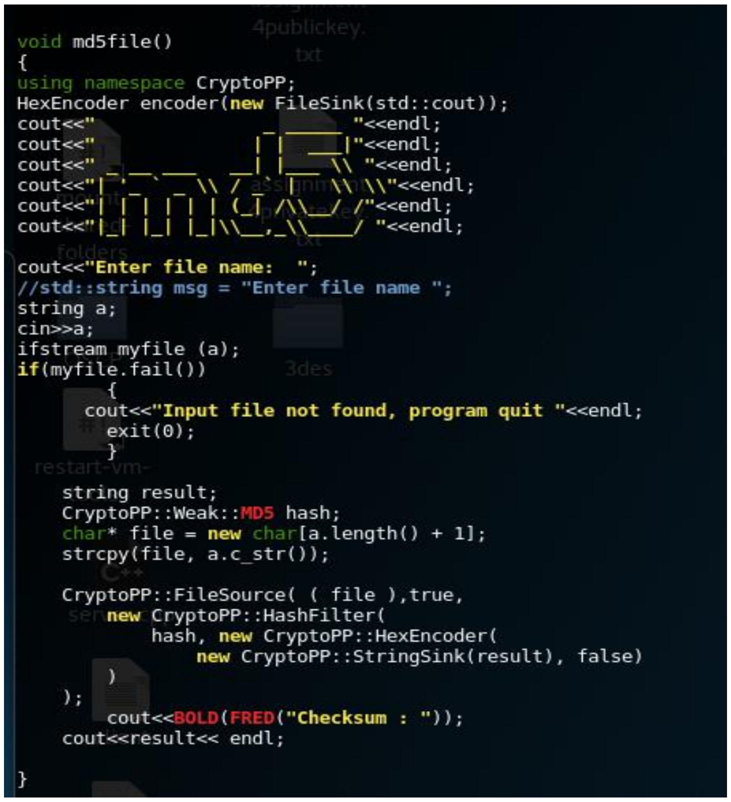
```
       _
      (_)
 __      __ _  _ __   ___   __ _  _ __ ___
 \ \ /\ / /| || '_ \ / __| / _` || '_ ` _ \
  \ V  V / | || | | |\__ \| (_| || | | | | |
   \_/\_/  |_||_| |_||___/ \__,_||_| |_| |_|
             _ __  ___   __ _
            | '__|/ __| / _` |
            | |   \__ \| (_| |
            |_|   |___/ \__,_|


                MENU


1.      Generate RSA Signature for a file
2.      Verify a Signature of a file
3.      Create MD-5 Hash for a file
4.      Create SHA-1 Hash for a file
5.      Quit
root@winsam:~4
       _          ___
      | |        /_ |
   ___| |__   __ _`| |
/ ___| '_ \ / _` ||| |
\__ \ | | | (_| || |_
|___/_| |_|\__,_\___/
Enter file name:  unencrypted.pdf
Checksum : 755f2e22419ebaceb1510aff9537210d2fed02d8
```

root@kali: ~/Desktop/CryptoAssignment3

```
root@kali:~/Desktop/CryptoAssignment3# sha1sum unencrypted.pdf
755f2e22419ebaceb1510aff9537210d2fed02d8  unencrypted.pdf
root@kali:~/Desktop/CryptoAssignment3#
```

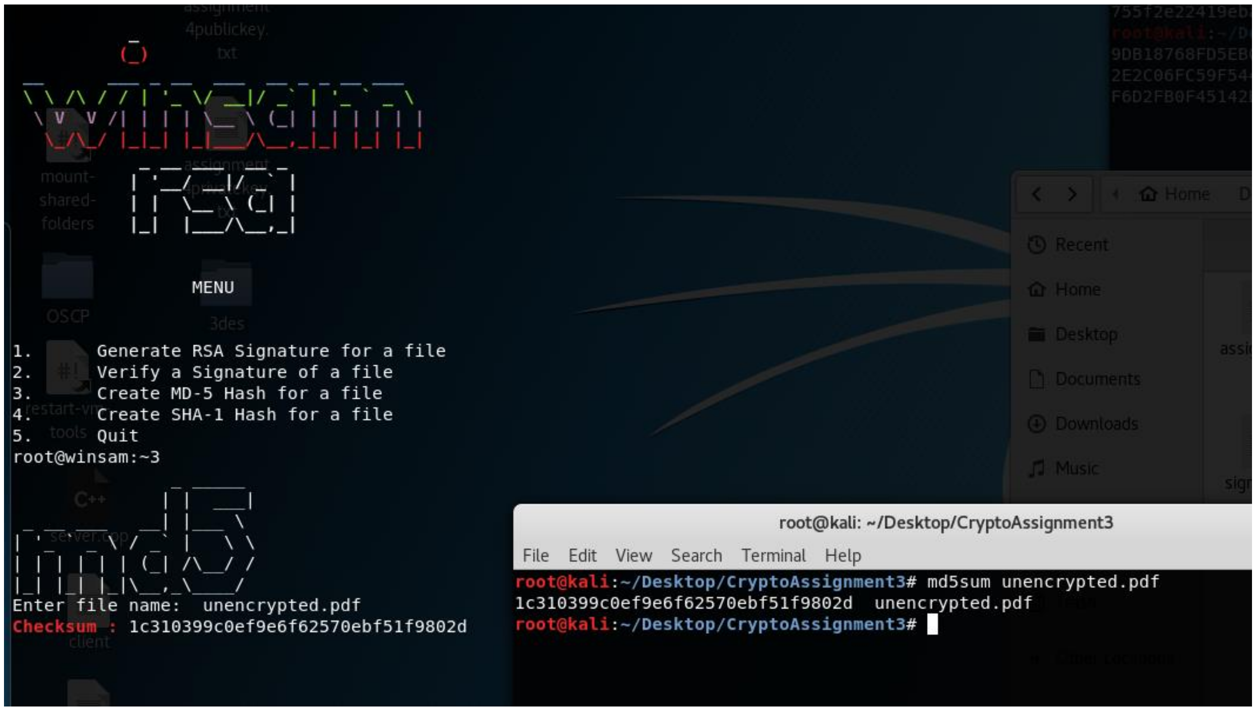
MD5file()

Below function is the function to perform MD5 calculation on any input file



```
void md5file()
{
using namespace CryptoPP;
HexEncoder encoder(new FileSink(std::cout));
cout<<"                    _ ____  "<<endl;
cout<<"                   | | ___|"<<endl;
cout<<"  _ __ ___   __| |___ \\ "<<endl;
cout<<"| '_ ` _ \\ / _` |___ \\ \\"<<endl;
cout<<"| | | | | | (_| /\\__/ /"<<endl;
cout<<"|_| |_| |_|\\__,_\\____/ "<<endl;

cout<<"Enter file name:  ";
//std::string msg = "Enter file name ";
string a;
cin>>a;
ifstream myfile (a);
if(myfile.fail())
        {
      cout<<"Input file not found, program quit "<<endl;
        exit(0);
        }

    string result;
    CryptoPP::Weak::MD5 hash;
    char* file = new char[a.length() + 1];
    strcpy(file, a.c_str());

    CryptoPP::FileSource( ( file ),true,
        new CryptoPP::HashFilter(
            hash, new CryptoPP::HexEncoder(
                new CryptoPP::StringSink(result), false)
        )
    );
        cout<<BOLD(FRED("Checksum : "));
    cout<<result<< endl;

}
```

Sample output of MD5

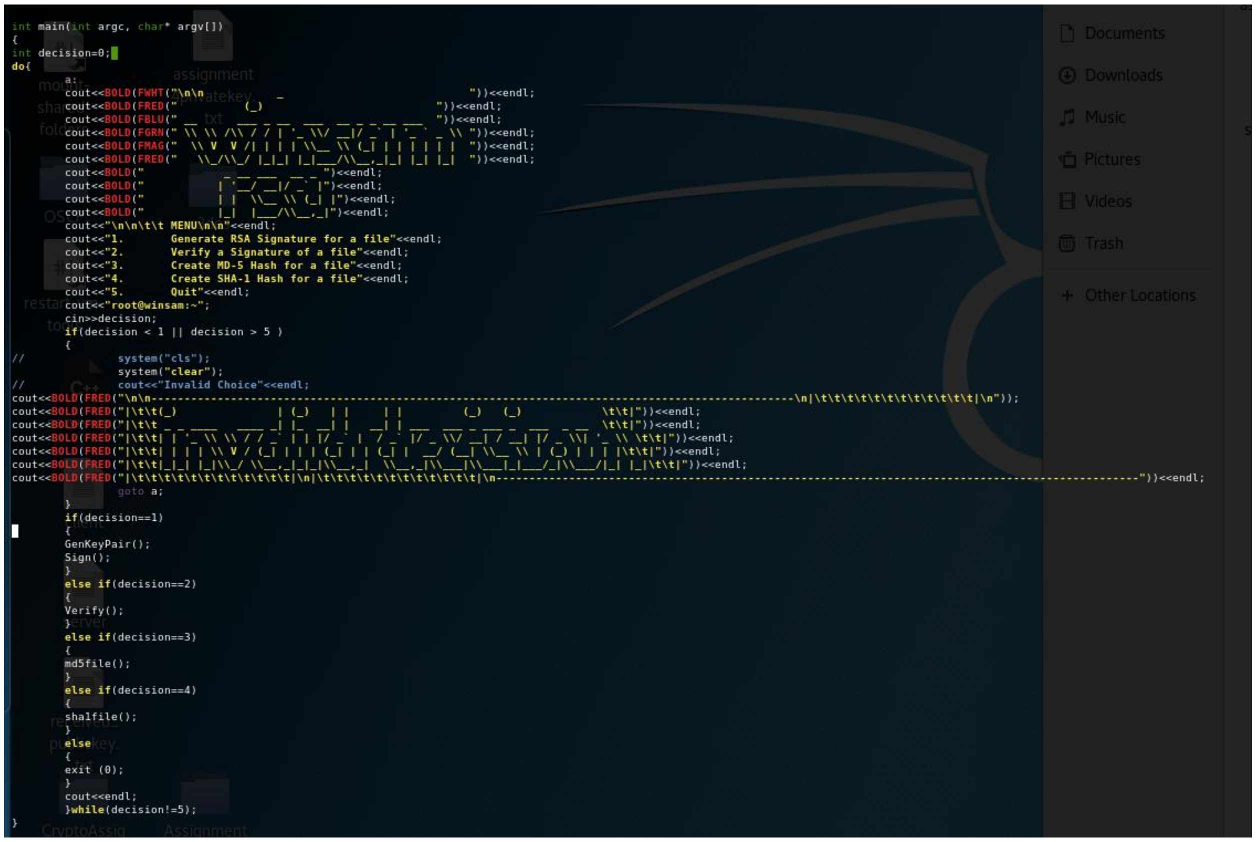


```
                  MENU

1.    Generate RSA Signature for a file
2.    Verify a Signature of a file
3.    Create MD-5 Hash for a file
4.    Create SHA-1 Hash for a file
5.    Quit
root@winsam:~3
                _ ____
               | |  ___|
 _ __ ___   ___| |___ \
| '_ ` _ \ / _` |    \ \
| | | | | | (_| /\__/ /
|_| |_| |_|\__,_\____/
Enter file name:  unencrypted.pdf
Checksum : 1c310399c0ef9e6f62570ebf51f9802d
```

```
root@kali:~/Desktop/CryptoAssignment3# md5sum unencrypted.pdf
1c310399c0ef9e6f62570ebf51f9802d  unencrypted.pdf
root@kali:~/Desktop/CryptoAssignment3#
```

root@kali: ~/Desktop/CryptoAssignment3
File Edit View Search Terminal Help
Recent
Home
Desktop
Documents
Downloads
Music

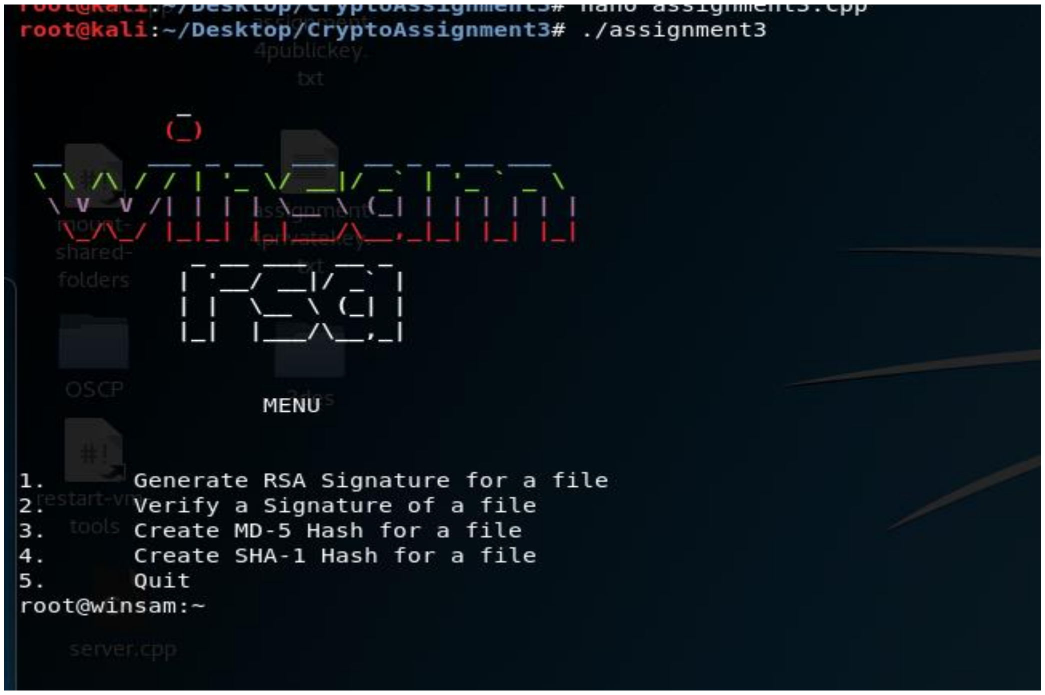
Main()



```
int main(int argc, char* argv[])
{
int decision=0;
do{
        a:
        cout<<BOLD(FWHT("\n\n                  _                              "))<<endl;
        cout<<BOLD(FRED("              (_)                              "))<<endl;
        cout<<BOLD(FBLU(" __      ___ _ __  ___  __ _ _ __ ___   "))<<endl;
        cout<<BOLD(FGRN(" \\ \\ /\\ / / | '_ \\/ __|/ _` | '_ ` _ \\ "))<<endl;
        cout<<BOLD(FMAG("  \\ V  V /| | | | \\__ \\ (_| | | | | | | "))<<endl;
        cout<<BOLD(FRED("   \\_/\\_/ |_|_| |_|___/\\__,_|_| |_| |_| "))<<endl;
        cout<<BOLD("          _ __ ___  __ _ ")<<endl;
        cout<<BOLD("         | '__/ __|/ _` |")<<endl;
        cout<<BOLD("         | |  \\__ \\ (_| |")<<endl;
        cout<<BOLD("         |_|  |___/\\__,_|")<<endl;
        cout<<"\n\n\t\t MENU\n\n"<<endl;
        cout<<"1.       Generate RSA Signature for a file"<<endl;
        cout<<"2.       Verify a Signature of a file"<<endl;
        cout<<"3.       Create MD-5 Hash for a file"<<endl;
        cout<<"4.       Create SHA-1 Hash for a file"<<endl;
        cout<<"5.       Quit"<<endl;
        cout<<"root@winsam:~";
        cin>>decision;
        if(decision < 1 || decision > 5 )
        {
//              system("cls");
                system("clear");
//              cout<<"Invalid Choice"<<endl;
cout<<BOLD(FRED("\n\n------------------------------------------------------------------------------------------------\n|\t\t\t\t\t\t\t\t\t\t\t\t|\n"));
cout<<BOLD(FRED("|\t\t(_)               | (_)   | |     | |         (_)   (_)            \t\t|"))<<endl;
cout<<BOLD(FRED("|\t\t _ _ ____   ____ _| |_  __| |   __| | ___  ___ _ ___ _  ___  _ __  \t\t|"))<<endl;
cout<<BOLD(FRED("|\t\t| | '_ \\ \\ / / _` | | |/ _` |  / _` |/ _ \\/ __| / __| |/ _ \\| '_ \\ \t\t|"))<<endl;
cout<<BOLD(FRED("|\t\t| | | | \\ V / (_| | | | (_| | | (_| |  __/ (__| \\__ \\ | (_) | | | |\t\t|"))<<endl;
cout<<BOLD(FRED("|\t\t|_|_| |_|\\_/ \\__,_|_|_|\\__,_|  \\__,_|\\___|\\___|_|___/_|\\___/|_| |_|\t\t|"))<<endl;
cout<<BOLD(FRED("|\t\t\t\t\t\t\t\t\t\t\t\t|\n|\t\t\t\t\t\t\t\t\t\t\t\t|\n--------------------------------------------------------------------------------------------------"))<<endl;
                goto a;
        }
        if(decision==1)
        {
        GenKeyPair();
        Sign();
        }
        else if(decision==2)
        {
        Verify();
        }
        else if(decision==3)
        {
        md5file();
        }
        else if(decision==4)
        {
        sha1file();
        }
        else
        {
        exit (0);
        }
        cout<<endl;
        }while(decision!=5);
}
```

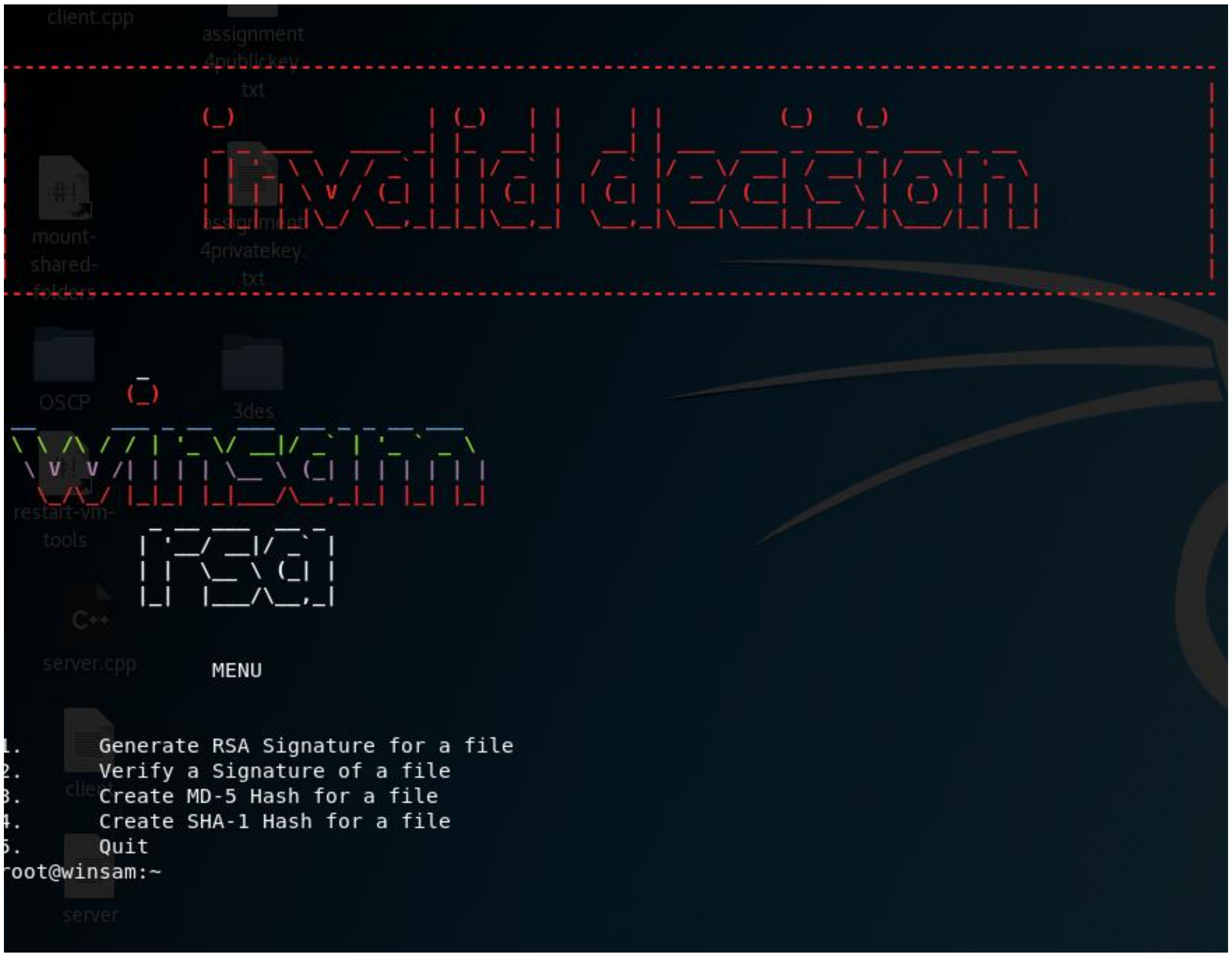
It’s just show the menu and some if else statement .

Sample output on MAIN()



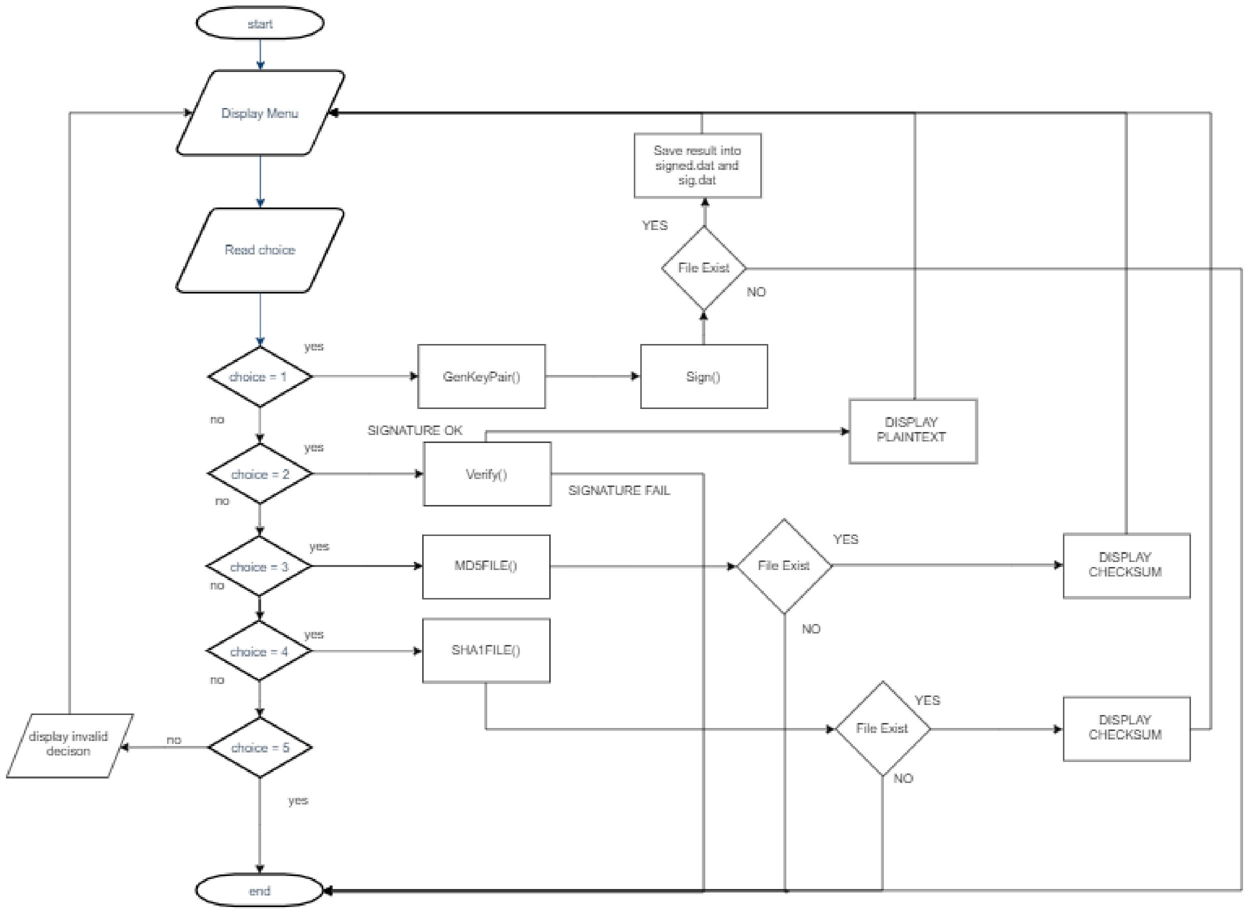
```
root@kali:~/Desktop/CryptoAssignment3# ./assignment3


                  _
                 (_)
 __      ___ _ __  ___  __ _ _ __ ___
 \ \ /\ / / | '_ \/ __|/ _` | '_ ` _ \
  \ V  V /| | | | \__ \ (_| | | | | | |
   \_/\_/ |_|_| |_|___/\__,_|_| |_| |_|
          _ __ ___  __ _
         | '__/ __|/ _` |
         | |  \__ \ (_| |
         |_|  |___/\__,_|


                MENU


1.      Generate RSA Signature for a file
2.      Verify a Signature of a file
3.      Create MD-5 Hash for a file
4.      Create SHA-1 Hash for a file
5.      Quit
root@winsam:~
```

Input number out of the selection choice



```
        MENU

1.      Generate RSA Signature for a file
2.      Verify a Signature of a file
3.      Create MD-5 Hash for a file
4.      Create SHA-1 Hash for a file
5.      Quit
root@winsam:~
```

Flowchart


start
Display Menu
Read choice
choice = 1
yes
no
GenKeyPair()
Sign()
File Exist
YES
NO
Save result into signed.dat and sig.dat
choice = 2
yes
no
SIGNATURE OK
Verify()
SIGNATURE FAIL
DISPLAY PLAINTEXT
choice = 3
yes
no
MD5FILE()
File Exist
YES
NO
DISPLAY CHECKSUM
choice = 4
yes
no
SHA1FILE()
File Exist
YES
NO
DISPLAY CHECKSUM
choice = 5
no
display invalid decison
yes
end

USER MANUAL (GITHUB)

README.MD

```
README.MD
```

GIT CLONE FROM GITHUB

To download the program in Ubuntu or Linux environment

```
# git clone https://github.com/Applebois/CryptoAssignment3
Cloning into 'CryptoAssignment3'...
remote: Enumerating objects: 20, done.
remote: Counting objects: 100% (20/20), done.
remote: Compressing objects: 100% (13/13), done.
remote: Total 20 (delta 1), reused 20 (delta 1), pack-reused 0
Unpacking objects: 100% (20/20), done.
```

INSTALLATION

Install the crypto++ library or else might not able to build/compile in later.

```
#sudo apt-get update
```

Then, issue next command to install crypto++

```
#sudo apt-get install libcrypto++-dev libcrypto++-doc libcrypto++-utils
```

COMMAND TO COMPILE/BUILD

How to compile the source code after issued "git clone" command, and will save as seed_cfb using parameter -o

```
# cd CryptoAssignment3
# cd Source
# g++ -g3 -ggdb -O0 -Wall -Wextra -Wno-unused -o assignment3
assignment3.cpp -lcryptopp
```

ADD PRIVILEGE TO THE BINARY FILE

Change program privilege to allow execution after issued "git clone" command

```
# cd CryptoAssignment3
# cd Output
# pwd
/root/CryptoAssignment3/Output
# chmod u+x assignment3
# ./assignment3
           _
          (_)
 __      ___ _ __  ___  __ _ _ __ ___
 \ \ /\ / / | '_ \/ __|/ _` | '_ ` _ \
  \ V  V /| | | | \__ \ (_| | | | | | |
   \_/\_/ |_|_| |_|___/\__,_|_| |_| |_|
            _ __ ___  __ _
           | '__/ __|/ _` |
           | |  \__ \ (_| |
           |_|  |___/\__,_|


               MENU


1.     Generate RSA Signature for a file
2.     Verify a Signature of a file
3.     Create MD-5 Hash for a file
4.     Create SHA-1 Hash for a file
5.     Quit
root@winsam:~
```

USAGE DEMONSTRATION

Option 1

```


1.      Generate RSA Signature for a file
2.      Verify a Signature of a file
3.      Create MD-5 Hash for a file
4.      Create SHA-1 Hash for a file
5.      Quit
root@winsam:~1



Private Key :
MIICdgIBADANBgkqhkiG9w0BAQEFAASCAmAwggJcAgEAAoGBAK7oxadqKq6Lu2zsZ1Wdj5IM
HDT3YkYv5ORR3BJMbcAnb5A+wjYPNSXFXHhYo37U+lMni/8UU9VEzdZ5avfqwlvft+i/tPtk
cjoP5lAVJ1xAxOnPlVlJS30vNL3HbOBWhINp+n12rPOJyL+TtpJ3g97xzSQSkhdYDdQmdI2u
vdhZAgERAoGAD27kQ3pPD2aup7qBlpz2FGp69Z1bfqnffY7AmDPrkPRruedrfT2Tv5FqCp5o
x20lJXRuPCdwzw2ZsQq2nWeJnoxxFj4dPO0Mi2g22yYv7WL5FCS8ssN3c6WKQMjfg3lq3DQ0
kbO86gGQnAea3E1sNNX1Ifqf5Jpu2nHoqrav+x8CQQDhUf4TgQkqwnfh8RBKvR2SdcoR+2bz
LrNuI5zNxdH04N1rIr8G68F8LCH31ZUwbzSeGgpWrmncz+W4vwitNAj/AkEAxrmZQZM/uyGW
O4eMGaBm90g7cpSzBChVwkeYFRc8VuiRBIP10A7V++Bf8Fs8hEpThYOYo3o5T6rtYr32QGoG
pwJBAJHLpGb5JAyb8zfYN7fjx9c9KGX9BmEeN93azt+AAFMoFszaPl7UyH2F99yZQmqiTzkf
6JJw2xZoZ3eKq0LlbzsCQQCX91cUBzC8RtxLo+ObIE69GR5mrfJOeTKFggri86acz/Zs3WGf
GmdmQis/VNPssU7toN4iqMJqGUweVQeaq25/AkEAt2yIYujFInQ5U9aR8VdkV70tley2nybJ
y8HznLr8qloD/l61BSh6t42xcdrZPhQ2v3iqDRYcLLBKqui4d8RmQw==



Public Key :
MIGdMA0GCSqGSIb3DQEBAQUAA4GLADCBhwKBgQCu6MWnaiqui7ts7GdVnY+SDBw092JGL+Tk
UdwSTG3AJ2+QPsI2DzUlxVx4WKN+1PpTJ4v/FFPVRM3WeWr36sJb37fov7T7ZHI6D+ZQFSdc
QMTpz5VZSUt9LzS9x2zgVoSDafp9dqzzici/k7aSd4Pe8c0kEpIXWA3UJnSNrr3YWQIBEQ==
     _           __
    | |         /  |
 ___| |__   __ _`| |
/ __| '_ \ / _` || |
\__ \ | | | (_| || |_
|___/_| |_|\__,_\___/
Enter file name:  sample.pdf
Checksum : 755f2e22419ebaceb1510aff9537210d2fed02d8
971F9AD92F48080A6E266E0F5D0C165822F12995C9A74488D142019327D86071EA95B5E748
1066D3E816B09B6901044ADD1789DE62D25E86D9B2CEB09850968CFA8CA7737F3749F045E0
```

```
80F02DEC833DEBC19E2ECEA6AA00F32685446859E7936BB7AC03002794749EFF39FF7D7D44
9AACE7D3CB9F4C0CE2896C7C5334064C07
```

Content of Public Key

```
root@kali:~/CryptoAssignment3/Output# cat pubkey.txt
MIGdMA0GCSqGSIb3DQEBAQUAA4GLADCBhwKBgQCu6MWnaiqui7ts7GdVnY+SDBw092JGL+Tk
UdwSTG3AJ2+QPsI2DzUlxVx4WKN+1PpTJ4v/FFPVRM3WeWr36sJb37fov7T7ZHI6D+ZQFSdc
QMTpz5VZSUt9LzS9x2zgVoSDafp9dqzzici/k7aSd4Pe8c0kEpIXWA3UJnSNrr3YWQIBEQ==
```

Content of PrivateKey

```
root@kali:~/CryptoAssignment3/Output# cat privkey.txt
MIICdgIBADANBgkqhkiG9w0BAQEFAASCAmAwggJcAgEAAoGBAK7oxadqKq6Lu2zsZ1Wdj5IM
HDT3YkYv5ORR3BJMbcAnb5A+wjYPNSXFXHhYo37U+lMni/8UU9VEzdZ5avfqwlvft+i/tPtk
cjoP5lAVJ1xAxOnPlVlJS30vNL3HbOBWhINp+n12rPOJyL+TtpJ3g97xzSQSkhdYDdQmdI2u
vdhZAgERAoGAD27kQ3pPD2aup7qBlpz2FGp69Z1bfqnffY7AmDPrkPRruedrfT2Tv5FqCp5o
x20lJXRuPCdwzw2ZsQq2nWeJnoxxFj4dPO0Mi2g22yYv7WL5FCS8ssN3c6WKQMjfg3lq3DQ0
kbO86gGQnAea3E1sNNX1Ifqf5Jpu2nHoqrav+x8CQQDhUf4TgQkqwnfh8RBKvR2SdcoR+2bz
LrNuI5zNxdH04N1rIr8G68F8LCH31ZUwbzSeGgpWrmncz+W4vwitNAj/AkEAxrmZQZM/uyGW
O4eMGaBm90g7cpSzBChVwkeYFRc8VuiRBIP10A7V++Bf8Fs8hEpThYOYo3o5T6rtYr32QGoG
pwJBAJHLpGb5JAyb8zfYN7fjx9c9KGX9BmEeN93azt+AAFMoFszaP17UyH2F99yZQmqiTzkf
6JJw2xZoZ3eKq0LlbzsCQQCX91cUBzC8RtxLo+ObIE69GR5mrfJOeTKFggri86acz/Zs3WGf
GmdmQis/VNPssU7toN4iqMJqGUweVQeaq25/AkEAt2yIYujFInQ5U9aR8VdkV70tley2nybJ
y8HznLr8qloD/l61BSh6t42xcdrZPhQ2v3iqDRYcLLBKqui4d8RmQw==
```

CONTENT OF SIG.DAT

```
root@kali:~/CryptoAssignment3/Output# cat sig.dat
971F9AD92F48080A6E266E0F5D0C165822F12995C9A74488D142019327D86071EA95B5E748
1066D3E816B09B6901044ADD1789DE62D25E86D9B2CEB09850968CFA8CA7737F3749F045E0
80F02DEC833DEBC19E2ECEA6AA00F32685446859E7936BB7AC03002794749EFF39FF7D7D44
9AACE7D3CB9F4C0CE2896C7C5334064C07r
```

CONTENT OF SIGNED.DAT

```
root@kali:~/CryptoAssignment3/Output# cat signed.dat
```

```
755f2e22419ebaceb1510aff9537210d2fed02d8
```

Option 2

IF CORRECT PAIR OF PUBLIC KEY

```
1.      Generate RSA Signature for a file
2.      Verify a Signature of a file
3.      Create MD-5 Hash for a file
4.      Create SHA-1 Hash for a file
5.      Quit
root@winsam:~2



Plaintext hash :
Cipher decrypted using signature :
755f2e22419ebaceb1510aff9537210d2fed02d8
Signature OK
```

IF INCORRECT PAIR OF PUBLIC KEY

```
1.      Generate RSA Signature for a file
2.      Verify a Signature of a file
3.      Create MD-5 Hash for a file
4.      Create SHA-1 Hash for a file
5.      Quit
root@winsam:~2



VerifierFilter: digital signature not valid
```

Option 3

```
1.      Generate RSA Signature for a file
2.      Verify a Signature of a file
3.      Create MD-5 Hash for a file
4.      Create SHA-1 Hash for a file
5.      Quit
root@winsam:~3
               _ _____
              | |  ___|
 _ __ ___   __| |___ \
| '_ ` _ \ / _` |   \ \
| | | | | | (_| /\__/ /
|_| |_| |_|\__,_\____/
Enter file name:  sample.pdf
Checksum : 1c310399c0ef9e6f62570ebf51f9802d
```

|PROOF OF CONCEPT SHOW CHECKSUM VALUE IS MATCHES ON MD5 |

```
root@kali:~/CryptoAssignment3/Output# md5sum sample.pdf
1c310399c0ef9e6f62570ebf51f9802d  sample.pdf
```

Option 4

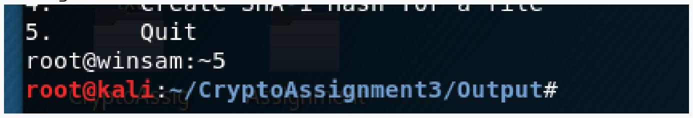
```
1.      Generate RSA Signature for a file
2.      Verify a Signature of a file
3.      Create MD-5 Hash for a file
4.      Create SHA-1 Hash for a file
5.      Quit
root@winsam:~4

     _           __
    | |         /  |
 ___| |__   __ _`| |
/ __| '_ \ / _` || |
\__ \ | | | (_| || |_
|___/_| |_|\__,_\___/
Enter file name:  sample.pdf
Checksum : 755f2e22419ebaceb1510aff9537210d2fed02d8
```

|PROOF OF CONCEPT SHOW CHECKSUM VALUE IS MATCHES ON SHA1 |

```
root@kali:~/CryptoAssignment3/Output# sha1sum sample.pdf
755f2e22419ebaceb1510aff9537210d2fed02d8  sample.pdf
```

Option 5

```
Program QUITED
```



```
5.      Quit
root@winsam:~5
root@kali:~/CryptoAssignment3/Output#
```