

TCSS 487 Cryptography

Project Report

Group Members: Ethan Nesel, Cordel Hampshire, Qinyu Tao

App Menu:

```
Welcome to the 487 Project Menu, please type the number of an option
1 Compute Plain Crypto Hash of a File
2 Compute Plain Crypto Hash of a Text Input
3 Encrypt a Data File Symmetrically
4 Decrypt a Symmetric Cryptogram
5 Compute MAC of a given file
6 Generate Elliptic Key Pair
7 Encrypt a file under Schnorr/ECDHIES
8 Decrypt a file under Schnorr/ECDHIES
9 Create a signature for a file
10 Verify a signature
11 Encrypt input under Schnorr/ECDHIES
12 Decrypt input under Schnorr/ECDHIES

Enter 13 to exit
Enter a valid selection:
```

Our CLI cryptography project offers 11 different services to a user. The services are described in the above menu (the app menu) and are additionally described below (in order).

PART 1 Components

1. [10 points] Compute a plain cryptographic hash of a given file.
2. BONUS: [4 points] Compute a plain cryptographic hash of text input by the user directly to the app instead of having to be read from a file.
3. [10 points] Encrypt a given data file symmetrically under a given passphrase.
4. [10 points] Decrypt a given symmetric cryptogram under a given passphrase.

5. BONUS: [4 points] Compute an authentication tag (MAC) of a given file under a given passphrase.

Part 2 Components

6. [10 points] Generate an elliptic key pair from a given passphrase and write the public key to a file.
7. [10 points] Encrypt a data file under a given elliptic public key file.
8. [10 points] Decrypt a given elliptic-encrypted file from a given password.
9. [10 points] Sign a given file from a given password and write the signature to a file.
10. [10 points] Verify a given data file and its signature file under a given public key file.
11. BONUS: [4 points] Encrypt/decrypt text input by the user directly to the app instead of having to be read from a file.

Sources used in implementation:

(KECCAK/SHAKE methods)

https://github.com/mjosaarinen/tiny_sha3/blob/master/sha3.c

<https://github.com/XKCP/XKCP/blob/master/Standalone/CompactFIPS202/C/Keccak-readable-and-compact.c>

(SHAKE Suffix Guide)

<https://en.wikipedia.org/wiki/SHA-3>

(NIST.SP.800-185 for implementing KMACXOF256 and helpers)

<https://dx.doi.org/10.6028/NIST.SP.800-185>

(Byte to hex converter)

<https://stackoverflow.com/questions/9655181/how-to-convert-a-byte-array-to-a-hex-string-in-java>

(Hex to byte converter)

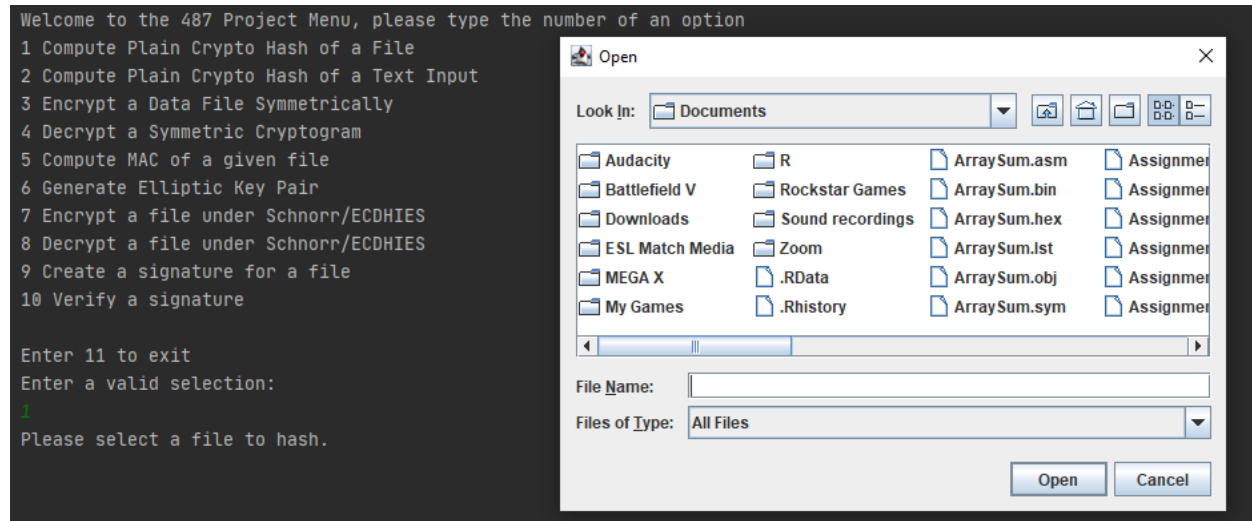
<https://stackoverflow.com/questions/140131/convert-a-string-representation-of-a-hex-dump-to-a-byte-array-using-java>

(Modified version of double and add)

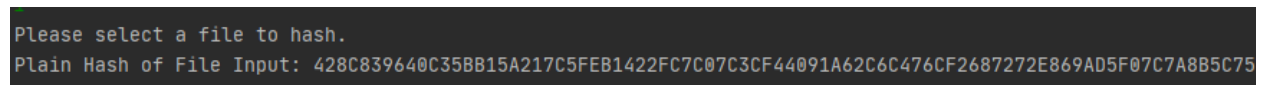
<https://andrea.corbellini.name/2015/05/17/elliptic-curve-cryptography-a-gentle-introduction/>

*All methods inspired by office hours visits, course slides, or course material are credited with a comment in the code.

Option 1: Computing a plain cryptographic hash of a file



By entering 1 in the menu and hitting enter a file chooser will pop up. (Please check behind the current tab if the file chooser does not appear in the foreground).



Once a file has been selected the hash will be printed below the file enter prompt message.

Option 2: Computing a cryptographic hash of user text input

```

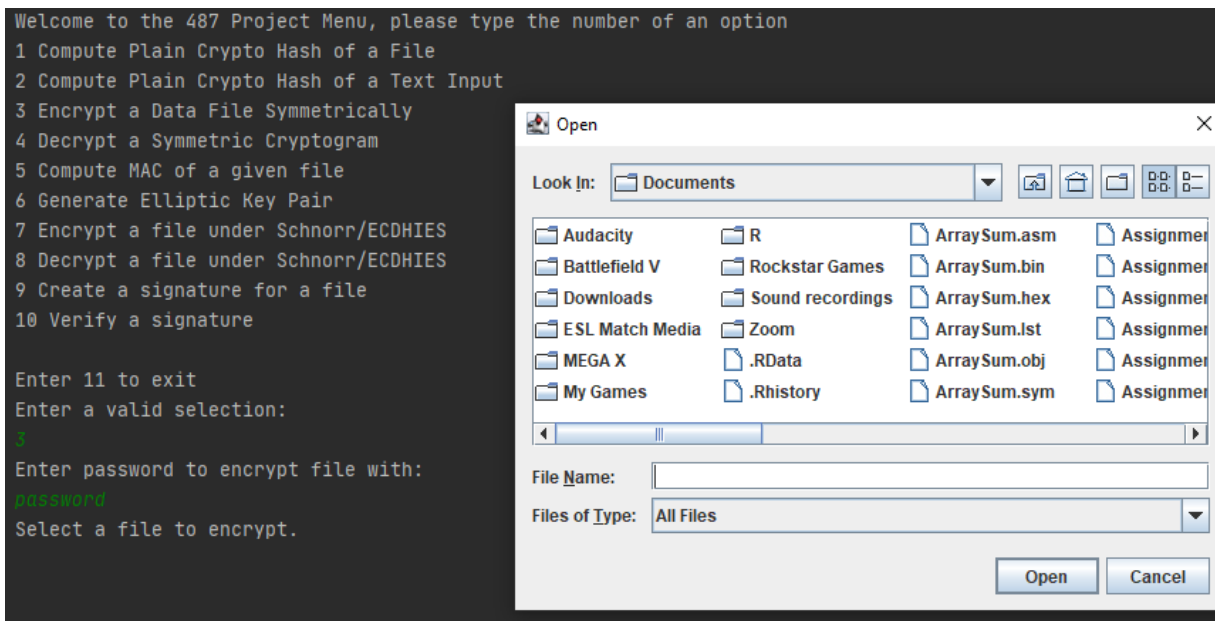
Welcome to the 487 Project Menu, please type the number of an option
1 Compute Plain Crypto Hash of a File
2 Compute Plain Crypto Hash of a Text Input
3 Encrypt a Data File Symmetrically
4 Decrypt a Symmetric Cryptogram
5 Compute MAC of a given file
6 Generate Elliptic Key Pair
7 Encrypt a file under Schnorr/ECDHIES
8 Decrypt a file under Schnorr/ECDHIES
9 Create a signature for a file
10 Verify a signature

Enter 11 to exit
Enter a valid selection:
2
Enter text to hash:
Testing the crypto program!
Plain Hash of Text Input: 428C839640C358B15A217C5FEB1422FC7C07C3CF44091A62C6C476CF2687272E869AD5F07C7A8B5C7574A5444F8BF9DEA2

```

By entering 2 in the menu (which reappears after completing an option) you are prompted to enter text to hash. The hash is then printed below.

Option 3: Encrypting a data file symmetrically



By entering 3 in the menu you can start the process of symmetrically encrypting a data file. A password and data file are requested.

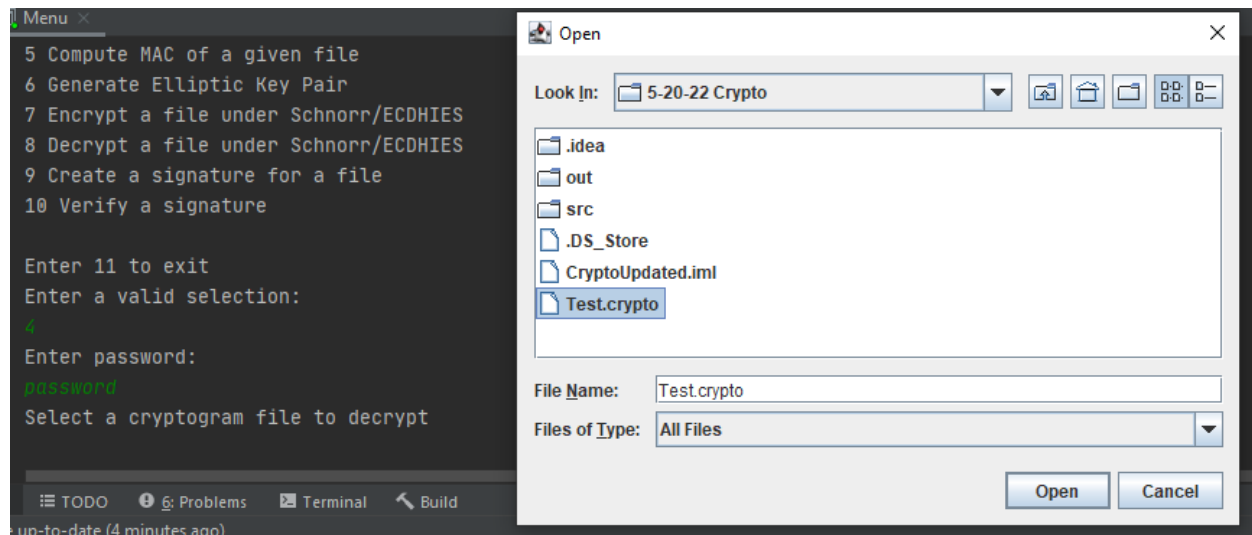
```

BF1F0114F929392E85EAF660C39BF83FD06175AC8368A93FB9571C30E3D055D0FB5CD86DA6B95B717DEC8D750E582A8F26406B7C44D
959E90AEC7F25BB00EB4C0EC0675AEC8F3E4002CC1C5914EAF78B
ECF5042C3FBF9C20AAB1079FE0E54DCF986C2792E1181829BDAAEE6624EECA10AEC3AAF11B560221F7E3B06377F89FD3E97A3E6BF62A

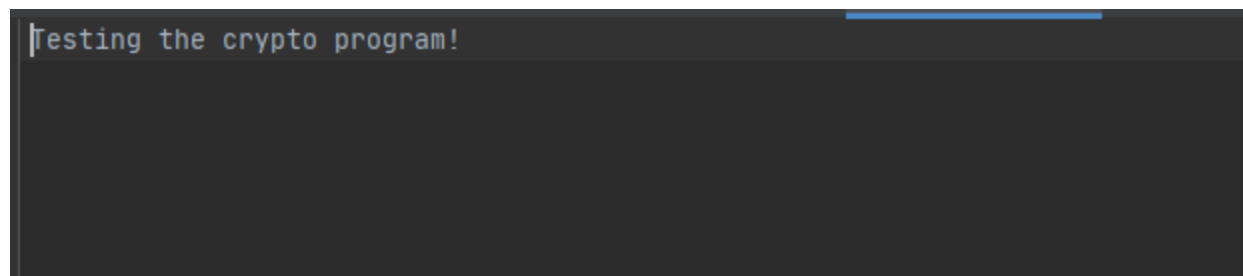
```

The cryptogram is saved in hexadecimal in a file titled (OriginalFileName).crypto.

Option 4: Decrypting a cryptogram from option 3

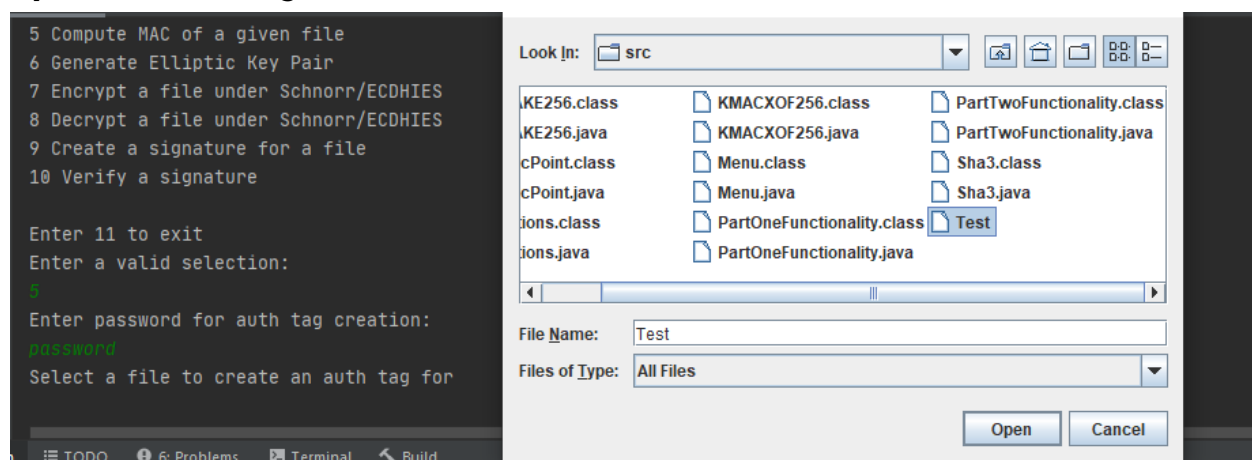


By entering a 4 in the menu you can decrypt a symmetric cryptogram. You will need the password used to encrypt, and the cryptogram.



The plaintext is then printed to a file called (CryptogramFileName). decrypted. If an incorrect password is used the output will state such and return to the menu.

Option 5: Creating a MAC for a file



By entering 5 into the menu, you are prompted for a password and file. The MAC will then be printed as is shown below.

```
Enter password for auth tag creation:
password
Select a file to create an auth tag for
Tag: 1BFCE52CE279AEC2631E5BFFF89EA813E728773E7C9645E7BA1371F3D28E85E0C00C8C840AD49FE8CC
```

Option 6: Generate an elliptic key pair

```
6 Generate Elliptic Key Pair
7 Encrypt a file under Schnorr/ECDHIES
8 Decrypt a file under Schnorr/ECDHIES
9 Create a signature for a file
10 Verify a signature

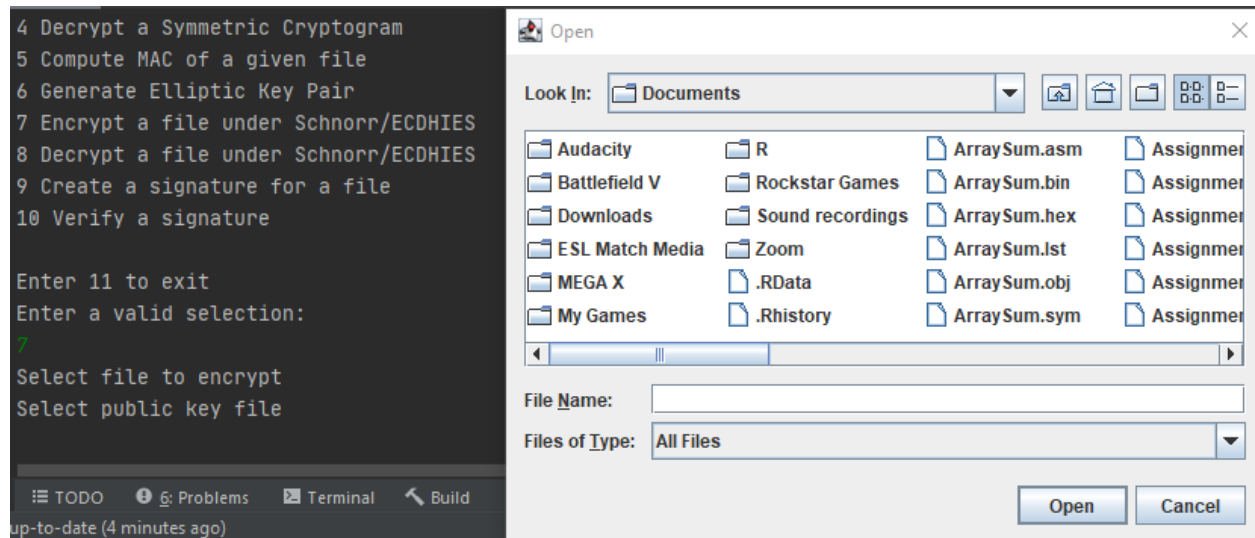
Enter 11 to exit
Enter a valid selection:
6
Enter password to create keys:
password
Keys created
```

To generate an elliptic key pair, just enter 6 into the menu and then type in a password. This password will be used to generate the pair and the public keys will be saved to a .pub_key_file with the password as the prefix. This is for convenience but the password would not normally be present in the file name as that would ruin it!

```
k69445024829909403913186295037962501818861980005532976579673030707492124423766619296727663880249
425505573523011691060106978902984069495940403476198140104852172340123445246048918443408797437832
```

Note: The output of the keys is in big integer format and thus should not be interpreted as hex like the other outputs.

Option 7: Encrypt a file under Schnorr/ECDHIES

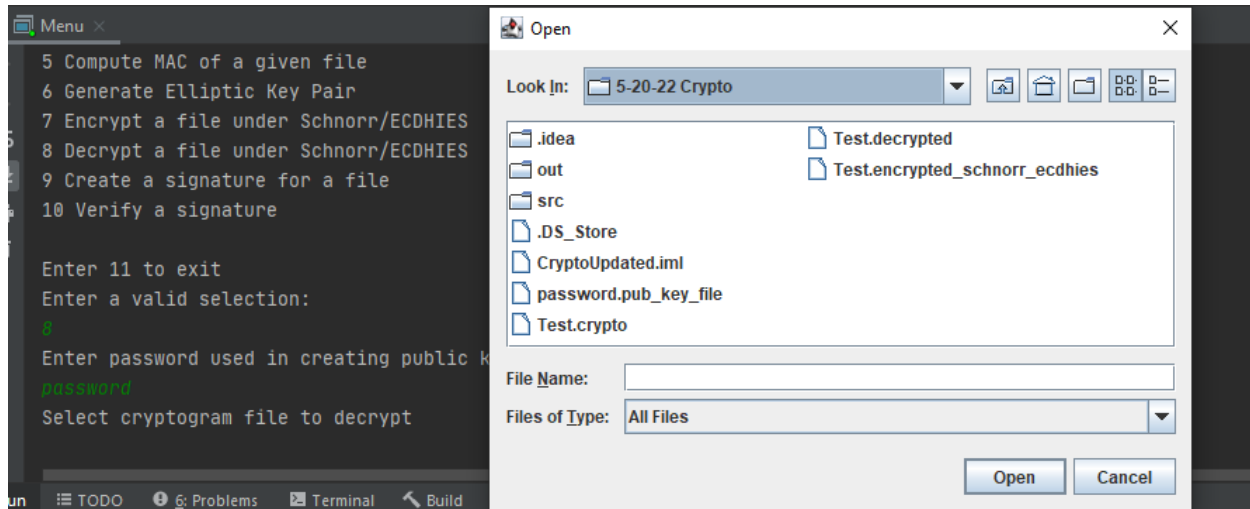


To encrypt a file under Schnorr/ECDHIES, press 7 and then select a file to encrypt and the corresponding public key file you would like to use. Note: The first file chooser appears separate and is omitted in the guide screenshots for this option.

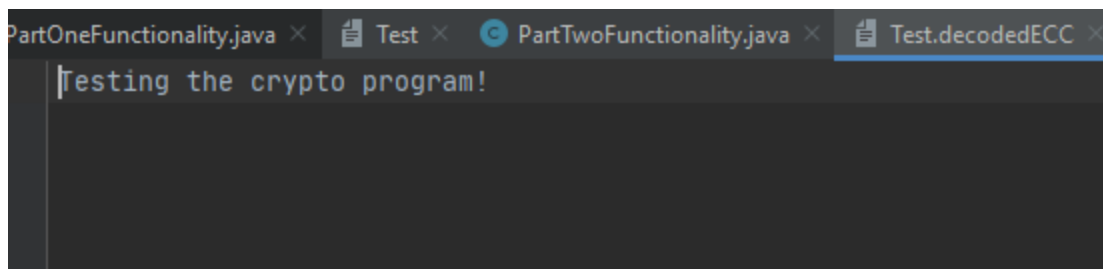
```
46615560029524610037533073103191101790385153561276511776465653062996900123676723846608824812955101779115
17466322490700207061900100179336929123601161504620362409229736179027203510096576633402371609458208243551
F6701BEA88BF4B0B027492005274B21D1E79B46510011B3D404B15
08F94C52A2C0C9F3026309FA90438AF44F018F5236D98624B50C2CCD426CD1564EBBEEC30826A1A71B827A82CEA7D794C2CD5CF4
```

The cryptogram is saved to a file ending in `.encrypted_schnorr_ecdhies`. The first two values are read as big integers and the last two are read as hexadecimal.

Option 8: Decrypting a file under Schnorr/ECDHIES

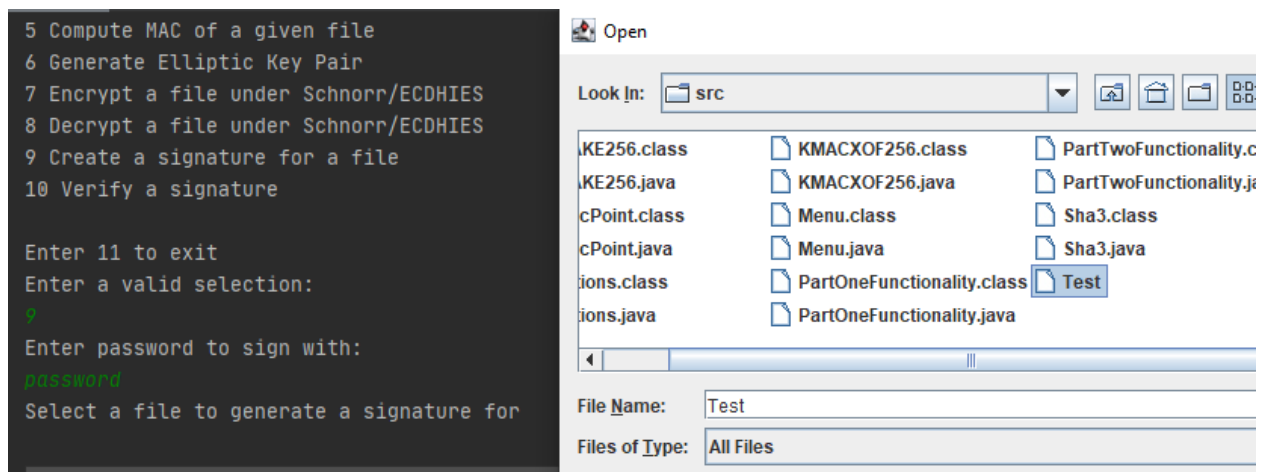


To decrypt under Schnorr/ECDHIES we type 8, and enter the password used to create the public key associated with the cryptogram. Then we select the cryptogram itself. If the password matches that used to build the cryptogram, it will output a .decodedECC file with the name of the cryptogram as the prefix. If not, the program will report a no match.

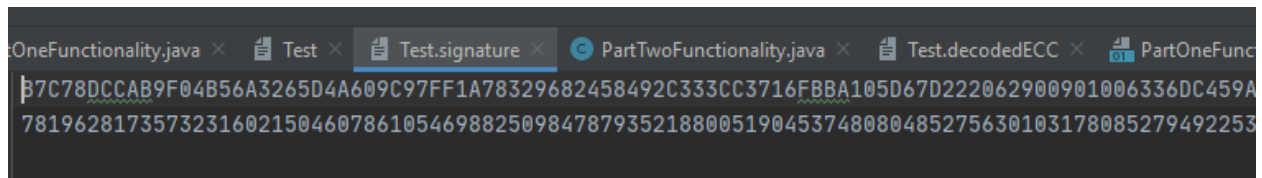


The output is shown above.

Option 9: Create a signature for a file



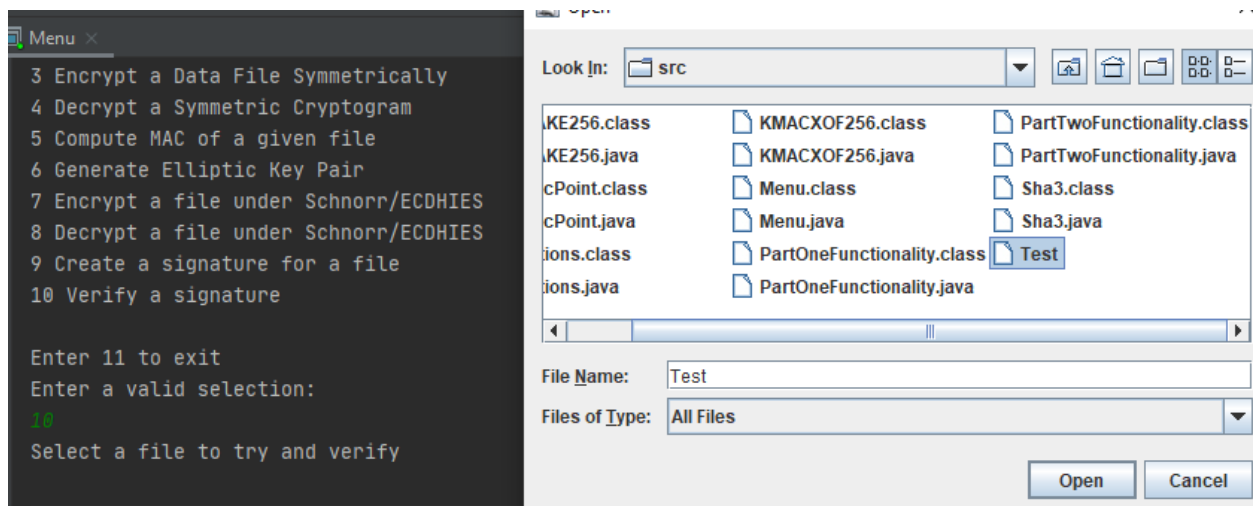
To create a signature for a file, first, enter 9. Then enter the password and select a file you would like to create a signature for. The corresponding output will be a file created with the same prefix and a .signature extension.



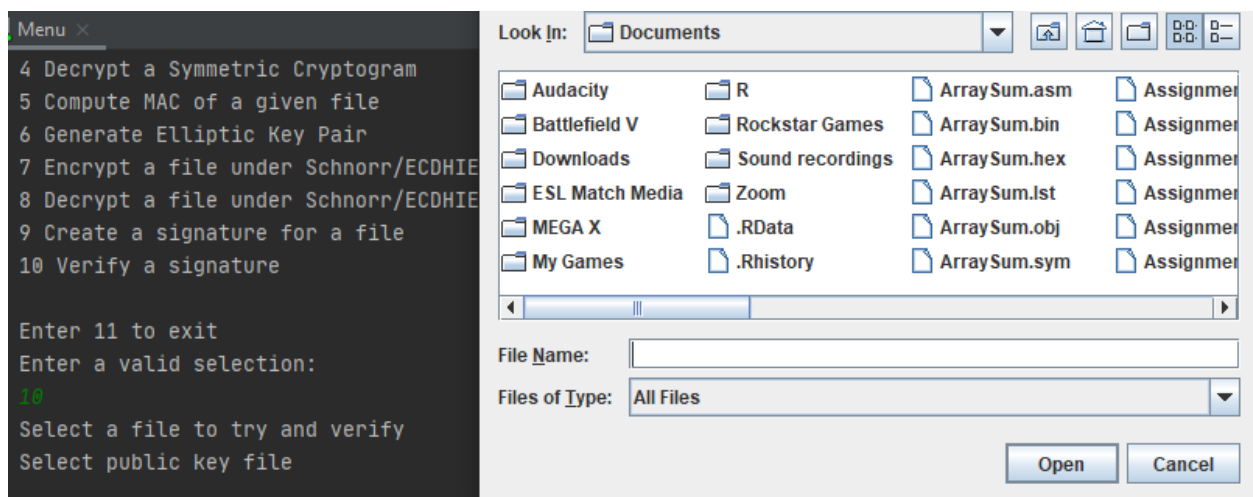
```
OneFunctionality.java × Test × Test.signature × PartTwoFunctionality.java × Test.decodedECC × PartOneFunc
p7C78DCCAB9F04B56A3265D4A609C97FF1A78329682458492C333CC3716FBBA105D67D222062900901006336DC459A
7819628173573231602150460786105469882509847879352188005190453748080485275630103178085279492253
```

The first part of the signature is in hex and the second is in big decimal as is shown in the signature file above.

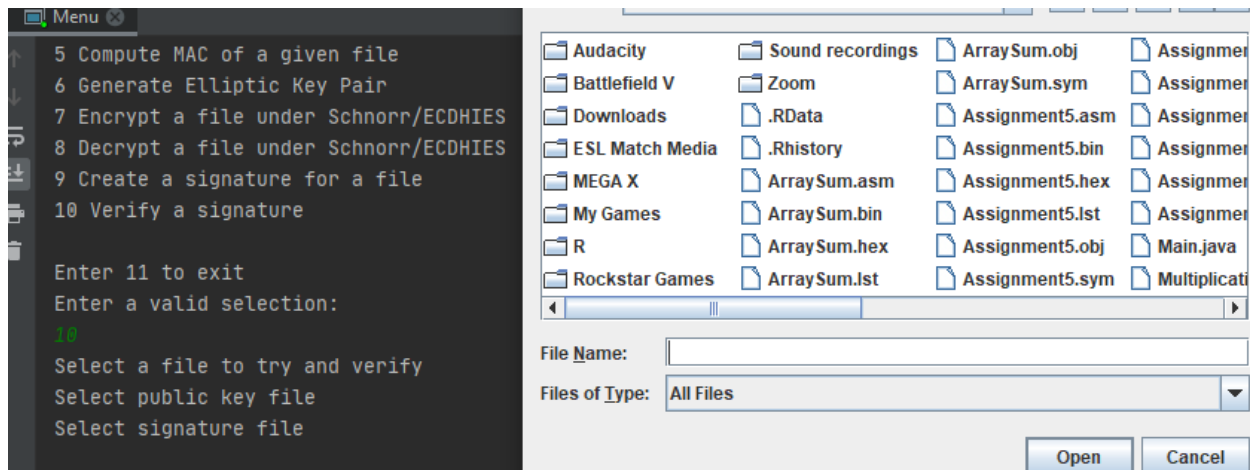
Option 10: Verifying a signature



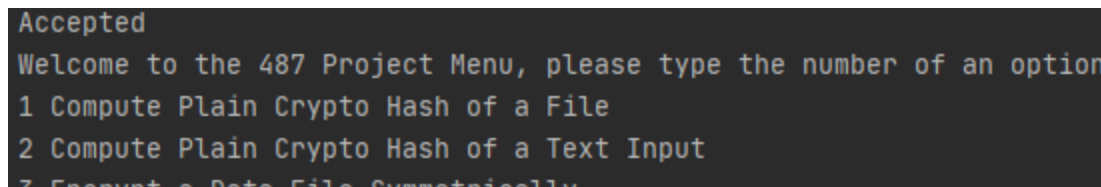
To verify a signature first enter 10, then enter the file you would like to verify.



Next, select the public key file. The public key file must have utilized the same password used to create the signature.

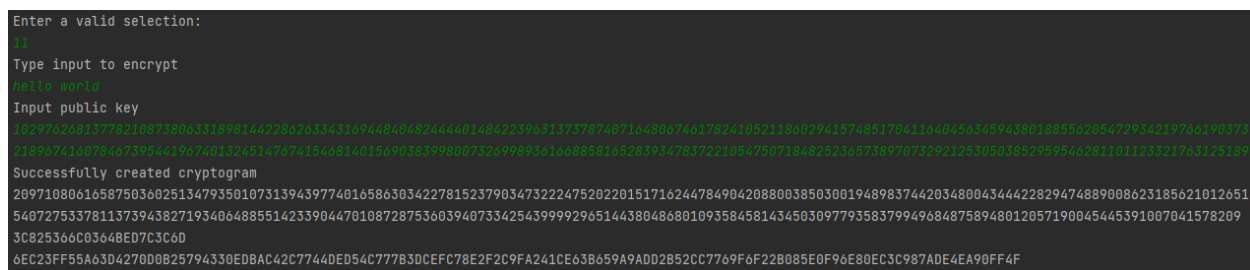


Finally, we select the signature file.



We see it is accepted and the menu is displayed again. Please note, if the signature does not check out, it will state “not accepted”

Option 11: Encrypt an input under Schnorr/ECDHIES



To encrypt an input under Schnorr/ECDHIES, press 11 and then input some text to encrypt and the corresponding public key you would like to use.

The cryptogram is printed to the console. The first two values are read as big integers and the last two are read as hexadecimal.

Option 12: Decrypting an input under Schnorr/ECDHIES

```
Enter a valid selection:
1
Enter password used in creating public key:
hello
Select cryptogram input to decrypt
100771000a1a50750302513679350107313943977401a50630342726123790367322212020703171624670490620000030500019409037442034000344422029474009000a2105621012633
40d72793137811373943027191a00400051423390467010072079360394073342543999290514430040001093004001434003097793033799490840750994001205719004646391007043578209
70c293a6c03a40f07c1c0d
0efc23ff59a03d42700001079433000a042c7744000046777030c0ff70e2f2c9fa241c630a99494007052cc77309f0f2000000f096000c1c98740c4e400ff4f
Successfully decoded cryptogram
hello world
```

To decrypt under Schnorr/ECDHIES we type 12, and enter the password used to create the public key associated with the cryptogram. Then we input the cryptogram itself. If the password matches that used to build the cryptogram, it will output the decoded text to the console. If not, the program will report a no match.

```

Enter a valid selection:
12
Enter password used in creating public key:
123456
Select cryptogram input to decrypt
00FF1000010075003020513479303073139439770010500303422701023790347032247502001017162447049042000003053001949937442034000034442003947009000023105421021051
00077753170113719430271934004005514213904470100720753003940734254399929051443004000109100450143503097793083799490004700940012005719004544331007041070209
30E25300C01040E007C3C00
HEC23F7304030427000005794330E00AC42C77440E0040777030CEFF70E2F2C0FA241CE0300949AD02002CC7704F4F220005EDF96E00EC3C0074DE40400FF4F
Mismatch detected, NOT ACCEPTED

```

A password mismatch is shown above.

Bugs/Issues/Comments:

- All options are working to the best of our knowledge, however, aside from some basic redirection back to the menu in the case of a closed file chooser, bad input will not be handled in most instances. Due to time constraints, it was necessary to focus on ensuring the graded elements worked. Future work would be to enforce pretreated input and more code cleanup.
- Additionally, saving the output as object files would have likely been better and something that could be improved upon.
- Ideally, we would reuse the same file reader and scanner as well.

