

## Cryptographic App Report and Manual

TCSS 487

June 4 2021

Andrew Josten

The following is a user guide for my Cryptographic app and written report of my solutions for each part.

### Implementation Report

The app implements KMACXOF256 derived from SHA3 and ECDHIES encryption and Schnorr signatures. The code base is designed in way so that the Main class acts as a driver for user input and calls functions compiled in two classes: KmacFunctions and EcurveFunctions.

KmacFunctions includes:

1. *cryptographicHash(msg)*- Computes cryptographic hash of a msg given as a byte array.
2. *authenticationTag(msg, pwd)*- Computes authentication tag of byte array msg given password pwd. This is one of the bonus services.
3. *encrypt(msg, pwd)*- encrypts a message symmetrically under the passphrase. Both are provided as a byte array. Returns a byte array of the cryptogram (z,c,t) as (z || c || t)
4. *decrypt(cryptogram, pwd)*- decrypts a cryptogram (provided in the form of z || c || t) and returns the decrypted byte array if the decrypted t matches t'.

The KmacFunctions class utilizes a class called KMAC which provides the KMACXOF256 function, cSHAKE256 (from which KMACXOF256 is derived from), as well as the supporting functions (bytepad, right\_encode, left\_encode, encode\_string), and a concatenation function (equivalent to ||). The implementation largely follows the NIST 800-185 publication's outline of the function.

Lastly, my Keccak functionality is included in the class SHA3. Originally, I had attempted to use the C example that we walked through in class, but I had troubles wrapping it all together. In the end, I used the Keccak team's examples in python and C to use as a guide as well as some parts from the mini SHA3. Using their website pseudocode ([https://keccak.team/keccak\\_specs\\_summary.html](https://keccak.team/keccak_specs_summary.html)) and their github examples (<https://github.com/XKCP/XKCP/blob/master/Standalone/CompactFIPS202/C/Keccak-readable-and-compact.c>) and (<https://github.com/XKCP/XKCP/blob/master/Standalone/CompactFIPS202/Python/CompactFIPS202.py>) I managed to get the Keccak functionalities to work. I left my unused, nonfunctional mini sha3 code for posterity.

As for the Elliptic Curve arithmetic functions, they are contained in EcurveFunctions:

1. *KeyPair(pwd)*- Generate a Schnorr/ECDHIES key pair from passphrase pwd (a byte array). This key pair is contained in the instance of EcurveFunctions and can be overwritten if KeyPair is called a second time.
2. *encrypt(msg)*- Encrypts a byte array under the (Schnorr/ECDHIES) public key V. This public key is the same one that was made by KeyPair. It returns an instance of CurveGram, a simple class that holds a cryptogram made via elliptic curve, (Z,c,t). CurveGram has a toBytes() function.
3. *decrypt(curvegram, pwd)*- Decrypts a given ecurve encryption under password pw. Takes a curve gram (Z,c,t) as its first argument. Returns a decoded message if  $t = t'$ .
4. *sigGenerator(pwd, msg)*- generates a signature for a given file (passed as a byte array) using a given password. Returns (h,z) as a concatenated byte array.
5. *verify(signature, msg)*- verifies a signature on a given message. The signature is passed as a 2d byte array of the byte arrays h and z. Returns true if verified.

EcurveFunctions derives its elliptic arithmetic functionality from the class Ecurve which represents an Edwards curve as two points (X,Y), and contains the added functionality of summing two curves, exponentiation, the various constructors, etc. I found this part was much easier to implement according to the provided pseudocode. Also included is toBytes and unBytes, which transform the instance into a byte array or back respectively.

Together, KmacFunctions and EcurveFunctions are used by the Main class which prompts the user. Main includes functions to read a file in as a byte array and write to a file as a byte array.

As a disclaimer, some of the calling of the high-level functions is a little clunky (eg decrypting a elliptic cryptogram requires the input byte array to be decompose into a CurveGram instance in the calling class). I would improve the abstraction of these functions but I simply ran out of time. Still, this solution functions.

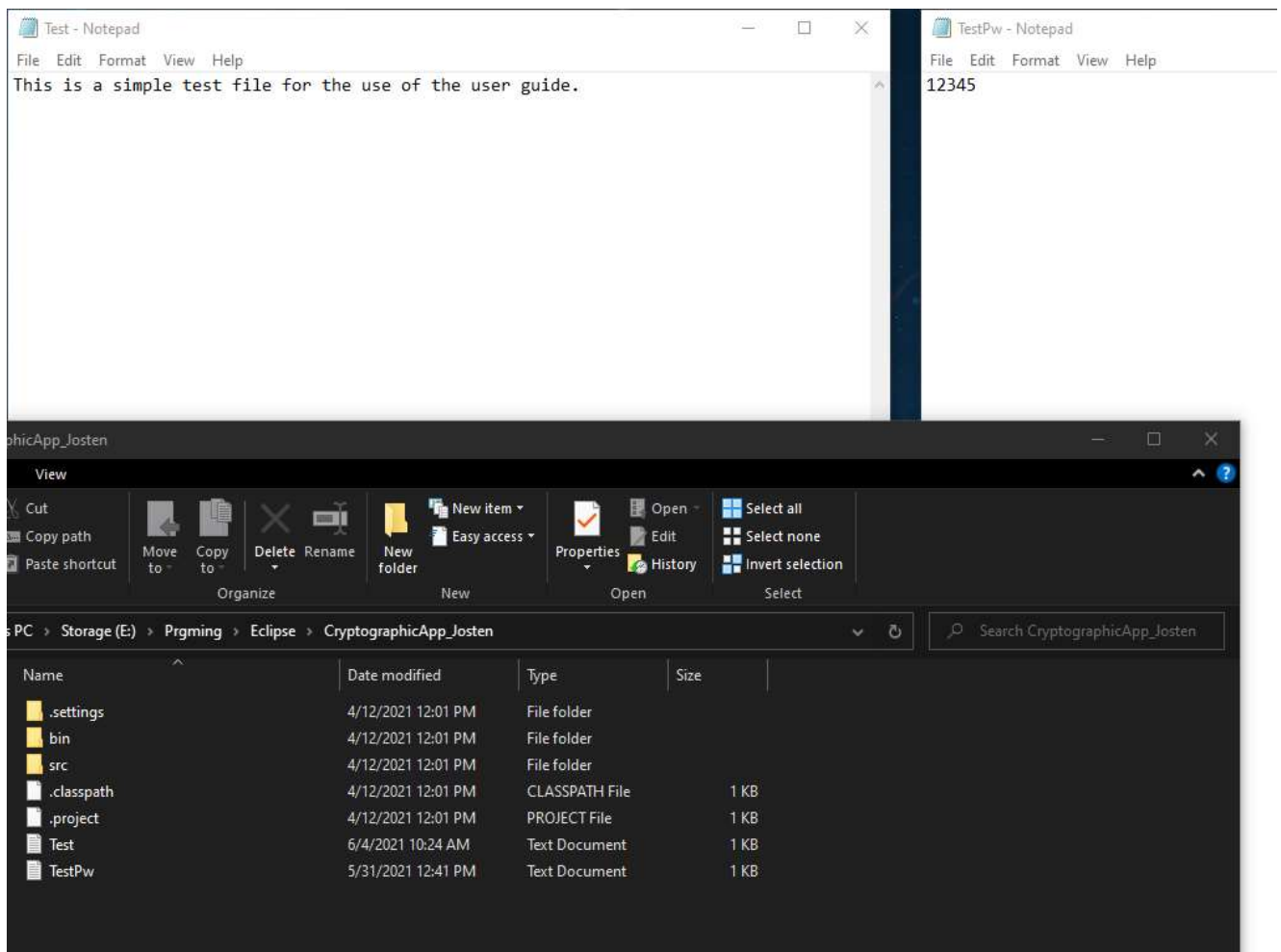
## User Guide

The cryptographic app takes input as .txt files and gives output to .txt files of a provided name. Files are read and written to using Java's `FileInputStream.read()` and `FileOutputStream.write()`.

The file searching is not particularly robust so provided files should be in the root of the project folder to be found. Though there is some validation for input, the app cannot handle every bad input so it can crash in certain conditions.

Also note that **all** input is given as files. I was unable to finish any kind of direct input into the console for messages and passwords.

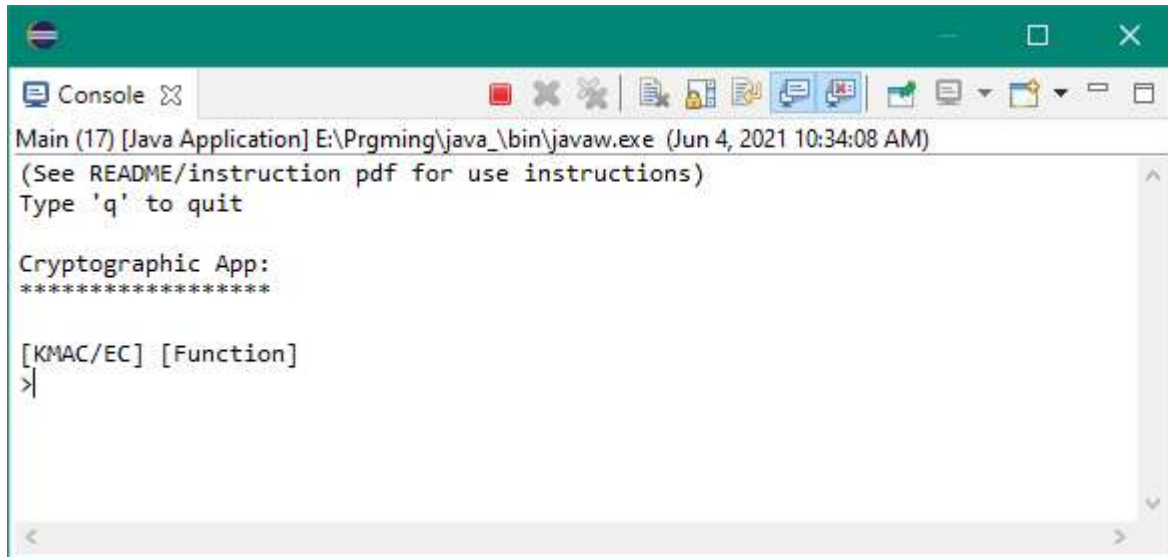
The guide will use this example input and passphrase:



Where Test.txt is general input (eg a message) and TestPw.txt is a passphrase.

### First prompt

When the app starts it will prompt the user for what type of encryption they are using (symmetric or elliptic) and which function they want.



```
Console
Main (17) [Java Application] E:\Prgrming\java_\bin\javaw.exe (Jun 4, 2021 10:34:08 AM)
(See README/instruction pdf for use instructions)
Type 'q' to quit

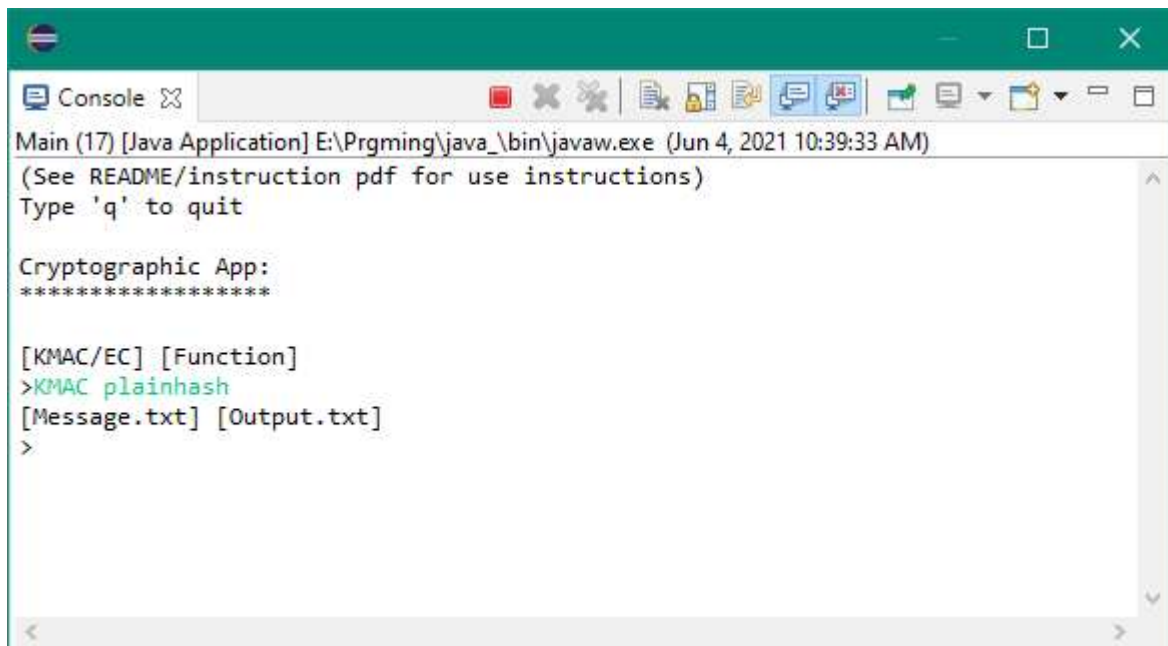
Cryptographic App:
*****

[KMAC/EC] [Function]
>
```

Input is space separated and expects either KMAC or EC as the first argument and a function as the second. (A full list of functions will be provided in a table in this report).

### Second Prompt

Once the function is known the app will ask for input, such as a message file and possibly a passphrase file and a name for the output file (should include .txt). This input will also be space separated.

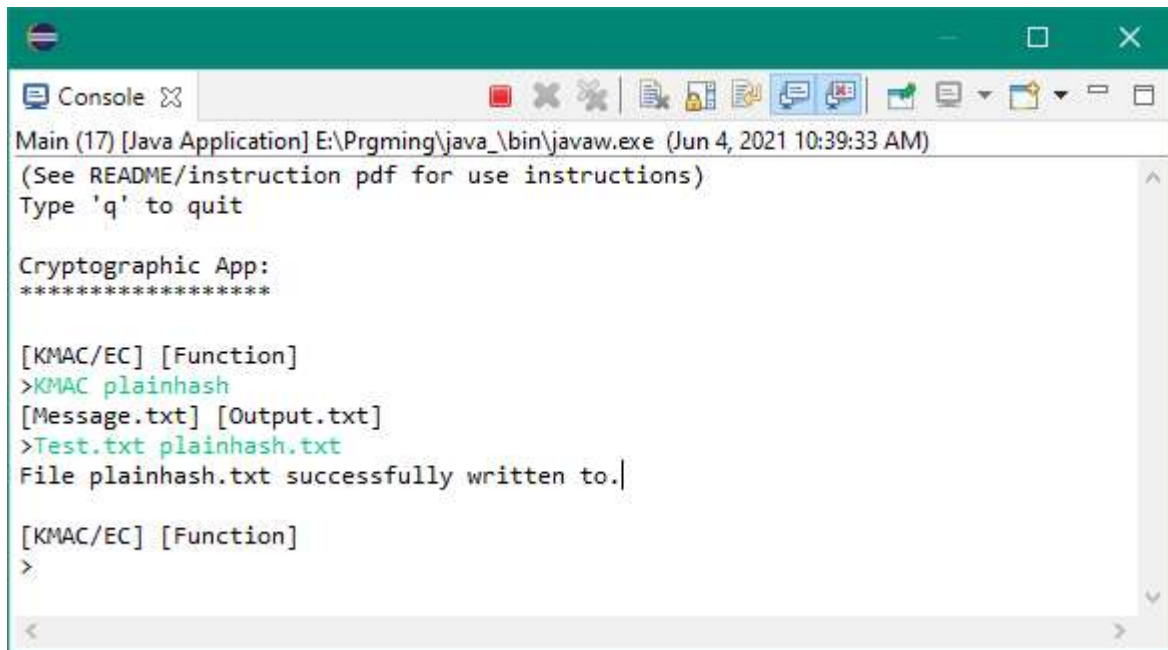


```
Console
Main (17) [Java Application] E:\Prgrming\java_\bin\javaw.exe (Jun 4, 2021 10:39:33 AM)
(See README/instruction pdf for use instructions)
Type 'q' to quit

Cryptographic App:
*****

[KMAC/EC] [Function]
>KMAC plainhash
[Message.txt] [Output.txt]
>
```

For this guide, we will input Test.txt and name our output file plainhash.txt. When we input these arguments, it should note that the file was written to and then prompt for the next function call.



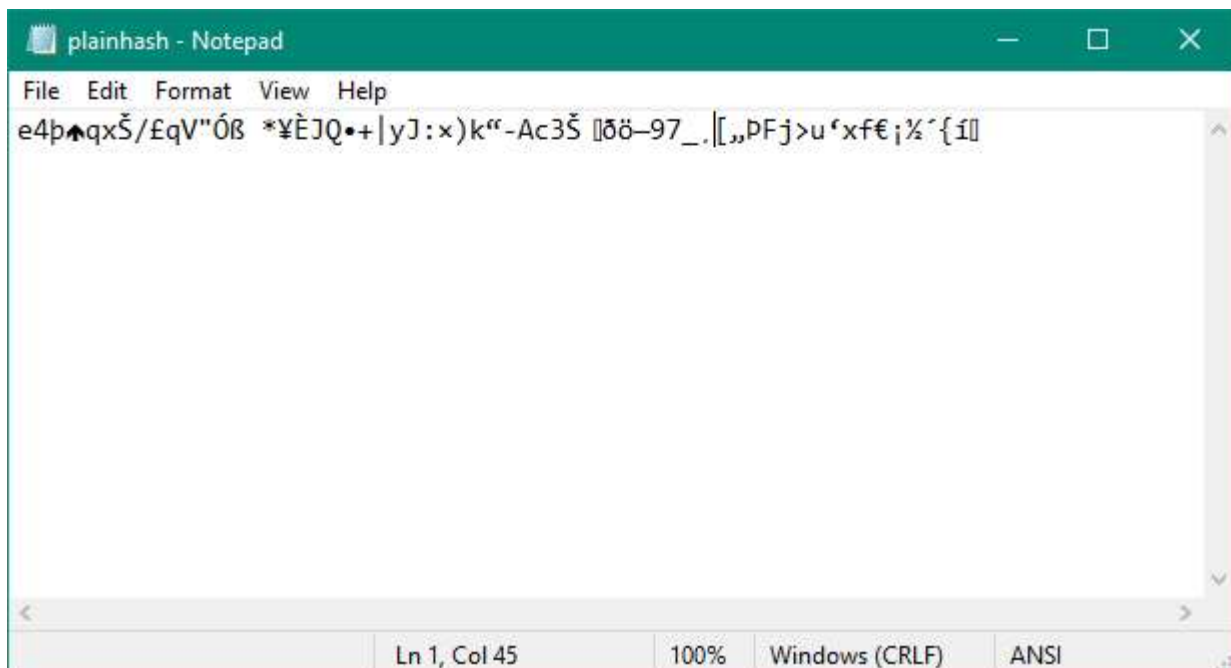
```
Console
Main (17) [Java Application] E:\Prgming\java\_bin\javaw.exe (Jun 4, 2021 10:39:33 AM)
(See README/instruction pdf for use instructions)
Type 'q' to quit

Cryptographic App:
*****

[KMAC/EC] [Function]
>KMAC plainhash
[Message.txt] [Output.txt]
>Test.txt plainhash.txt
File plainhash.txt successfully written to.

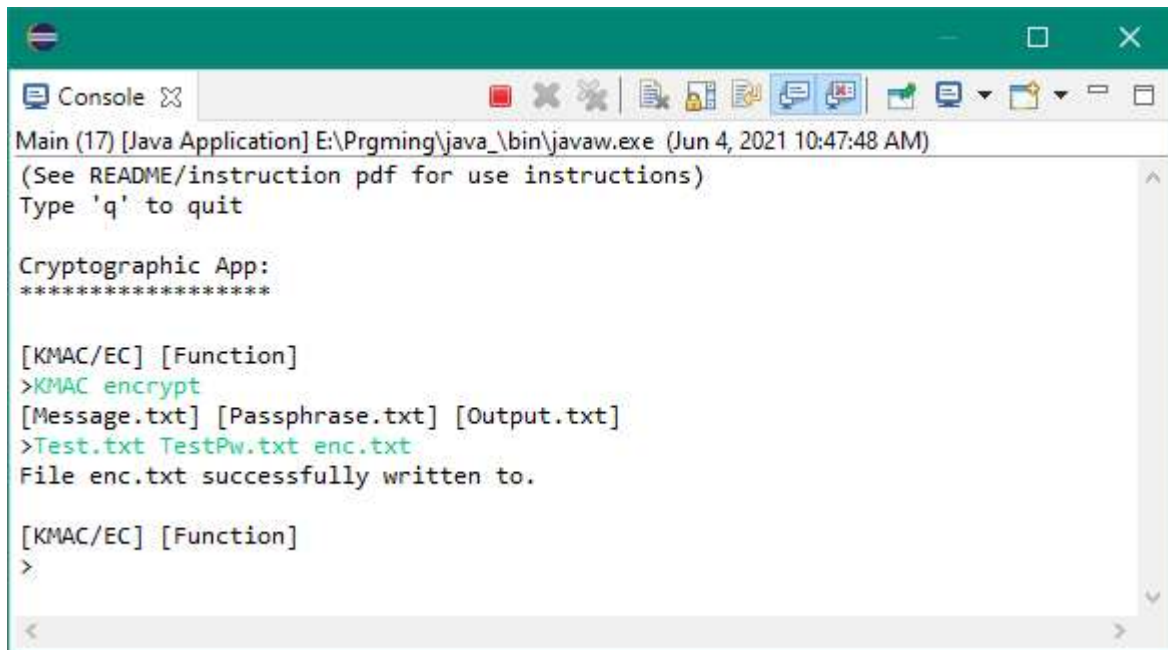
[KMAC/EC] [Function]
>
```

The file plainhash.txt should be:



```
plainhash - Notepad
File Edit Format View Help
e4p▲qxŠ/£qV"Óß *¥ÈJQ•+|yJ:x)k“-Ac3Š 0ðö-97_.|[„PFj>u‘xf€;¼`{1
```

For an encryption/decryption example we will encrypt Test.txt via KMACXOF256. Input is:



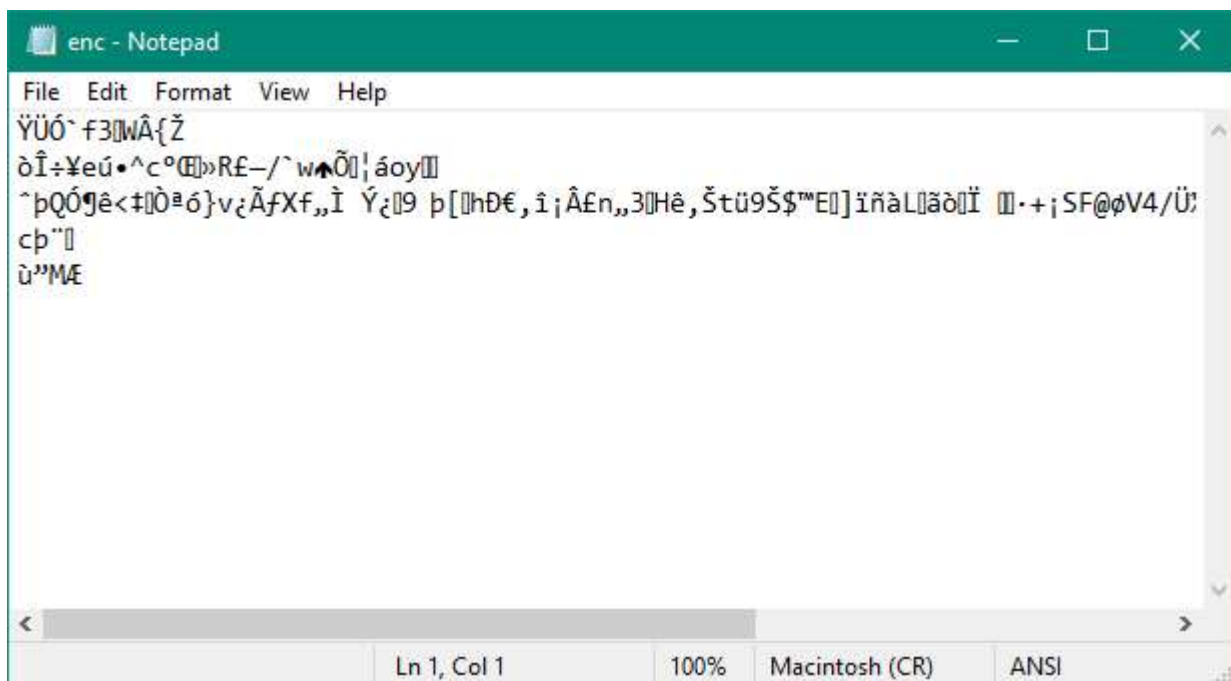
```
Main (17) [Java Application] E:\Prgming\java\_bin\javaw.exe (Jun 4, 2021 10:47:48 AM)
(See README/instruction pdf for use instructions)
Type 'q' to quit

Cryptographic App:
*****

[KMAC/EC] [Function]
>KMAC encrypt
[Message.txt] [Passphrase.txt] [Output.txt]
>Test.txt TestPw.txt enc.txt
File enc.txt successfully written to.

[KMAC/EC] [Function]
>
```

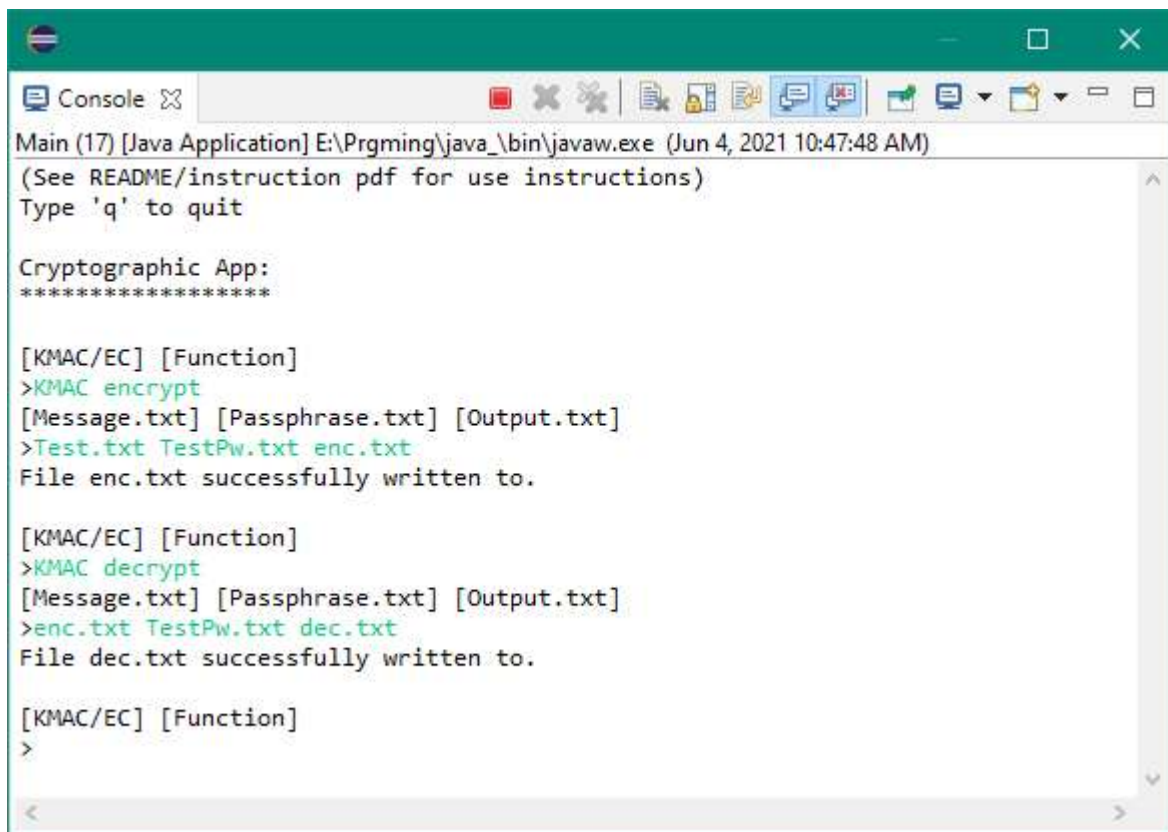
Output file is



```
enc - Notepad
File Edit Format View Help
ÿÜÓ`f3WÂ{Ž
òÎ÷¥eú•^c°»R£-/`w▲Õ!áoy
^pQÓgê<#0°ó}vçÄfXf,,İ Ÿç9 p[0hĐ€,î;ÂEn,,3Hè,Štü9Š$™E]iñàLäòİ 00.+;SF@øV4/Ü;
cþ"
ù"ME
```

A symmetric cryptogram in the form (z || c || t) written as a byte array to a file.

To decrypt the encoded file:



The screenshot shows a Java application window titled "Main (17) [Java Application] E:\Prgming\java\_\bin\javaw.exe (Jun 4, 2021 10:47:48 AM)". The console output is as follows:

```

(See README/instruction pdf for use instructions)
Type 'q' to quit

Cryptographic App:
*****

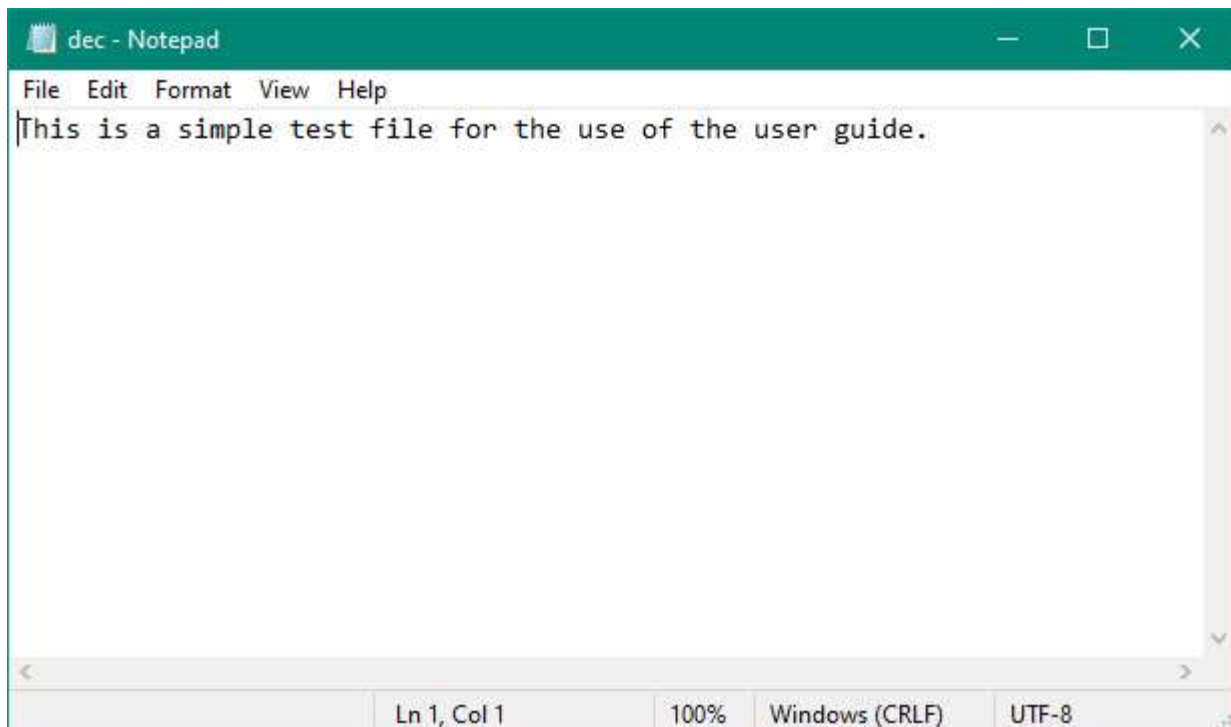
[KMAC/EC] [Function]
>KMAC encrypt
[Message.txt] [Passphrase.txt] [Output.txt]
>Test.txt TestPw.txt enc.txt
File enc.txt successfully written to.

[KMAC/EC] [Function]
>KMAC decrypt
[Message.txt] [Passphrase.txt] [Output.txt]
>enc.txt TestPw.txt dec.txt
File dec.txt successfully written to.

[KMAC/EC] [Function]
>

```

The decrypted output matches the original input:



The screenshot shows a Notepad window titled "dec - Notepad". The text content of the file is:

```

File Edit Format View Help
This is a simple test file for the use of the user guide.

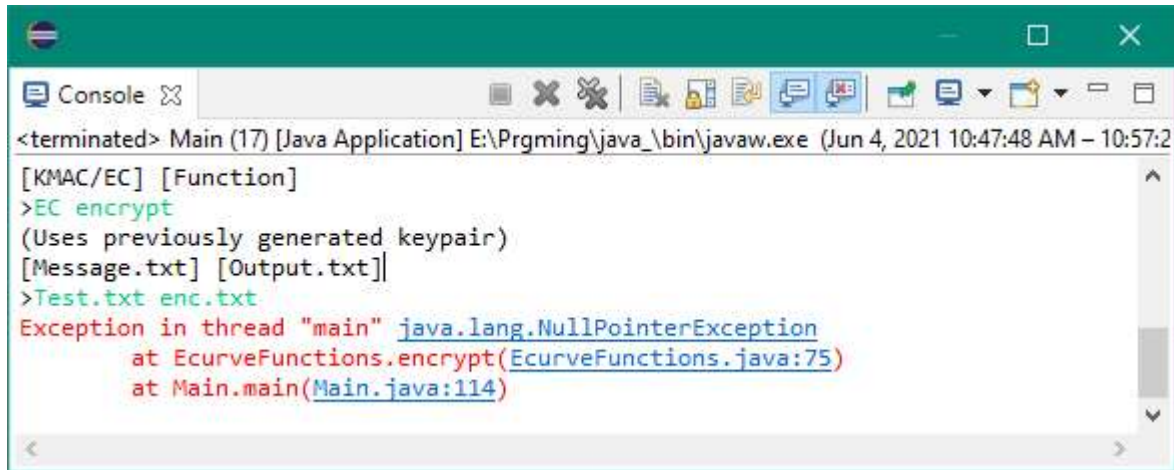
```

The status bar at the bottom indicates the cursor is at "Ln 1, Col 1", the zoom is "100%", the line ending is "Windows (CRLF)", and the encoding is "UTF-8".



## Special EC Use Instructions

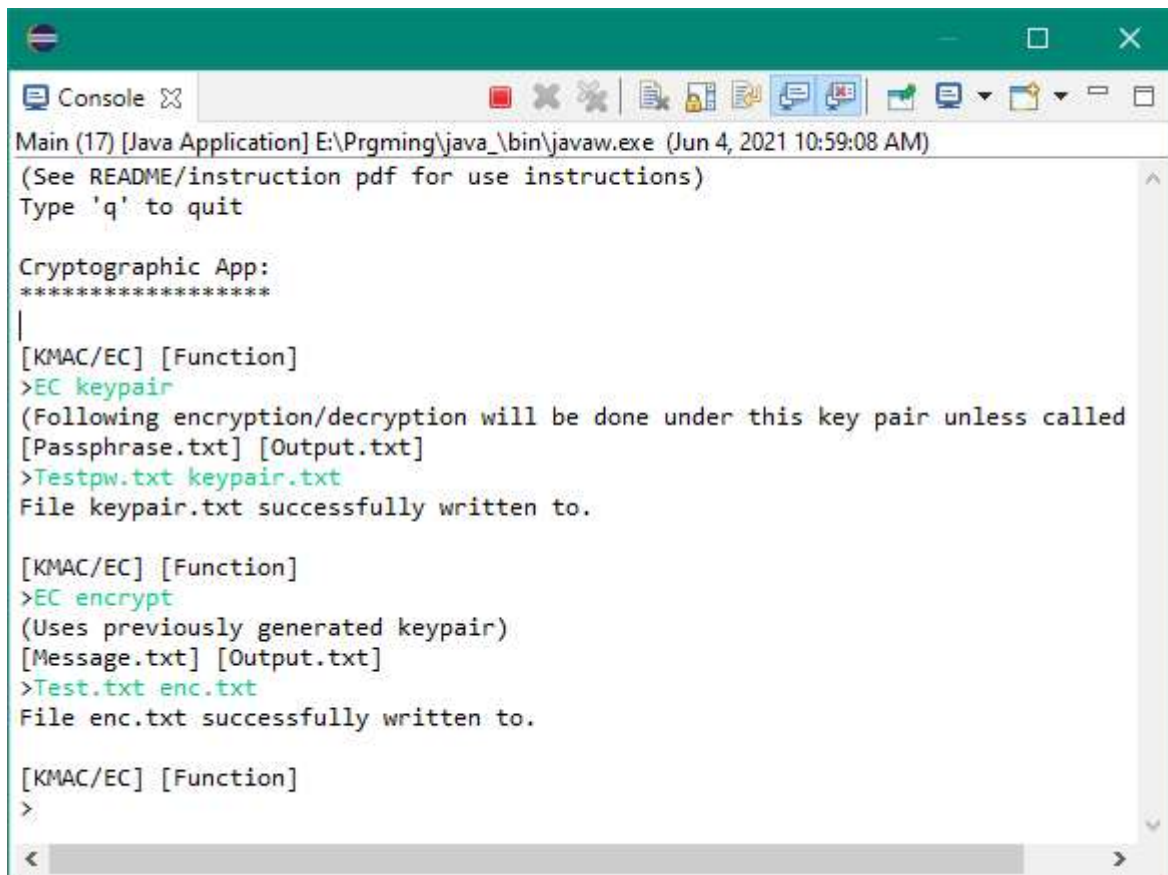
The Elliptic curve functionality depends on KeyPair, as in the public key V. Calling certain EC functions without first calling KeyPair on the chosen password file can lead to a crash:



The screenshot shows a Java console window titled "Main (17) [Java Application] E:\Prgrming\java\_\bin\javaw.exe (Jun 4, 2021 10:47:48 AM - 10:57:2)". The console output shows the user entering commands in a [KMAC/EC] [Function] prompt. The commands entered are >EC encrypt, (Uses previously generated keypair), [Message.txt] [Output.txt], and >Test.txt enc.txt. The output shows an exception in thread "main": java.lang.NullPointerException at EcurveFunctions.encrypt(EcurveFunctions.java:75) at Main.main(Main.java:114).

```
<terminated> Main (17) [Java Application] E:\Prgrming\java_\bin\javaw.exe (Jun 4, 2021 10:47:48 AM - 10:57:2)
[KMAC/EC] [Function]
>EC encrypt
(Uses previously generated keypair)
[Message.txt] [Output.txt]
>Test.txt enc.txt
Exception in thread "main" java.lang.NullPointerException
    at EcurveFunctions.encrypt(EcurveFunctions.java:75)
    at Main.main(Main.java:114)
```

As compared to:



The screenshot shows a Java console window titled "Main (17) [Java Application] E:\Prgrming\java\_\bin\javaw.exe (Jun 4, 2021 10:59:08 AM)". The console output shows the user entering commands in a [KMAC/EC] [Function] prompt. The commands entered are >EC keypair, (Following encryption/decryption will be done under this key pair unless called), [Passphrase.txt] [Output.txt], and >Testpw.txt keypair.txt. The output shows "File keypair.txt successfully written to." followed by >EC encrypt, (Uses previously generated keypair), [Message.txt] [Output.txt], and >Test.txt enc.txt. The output shows "File enc.txt successfully written to." followed by >.

```
Main (17) [Java Application] E:\Prgrming\java_\bin\javaw.exe (Jun 4, 2021 10:59:08 AM)
(See README/instruction pdf for use instructions)
Type 'q' to quit

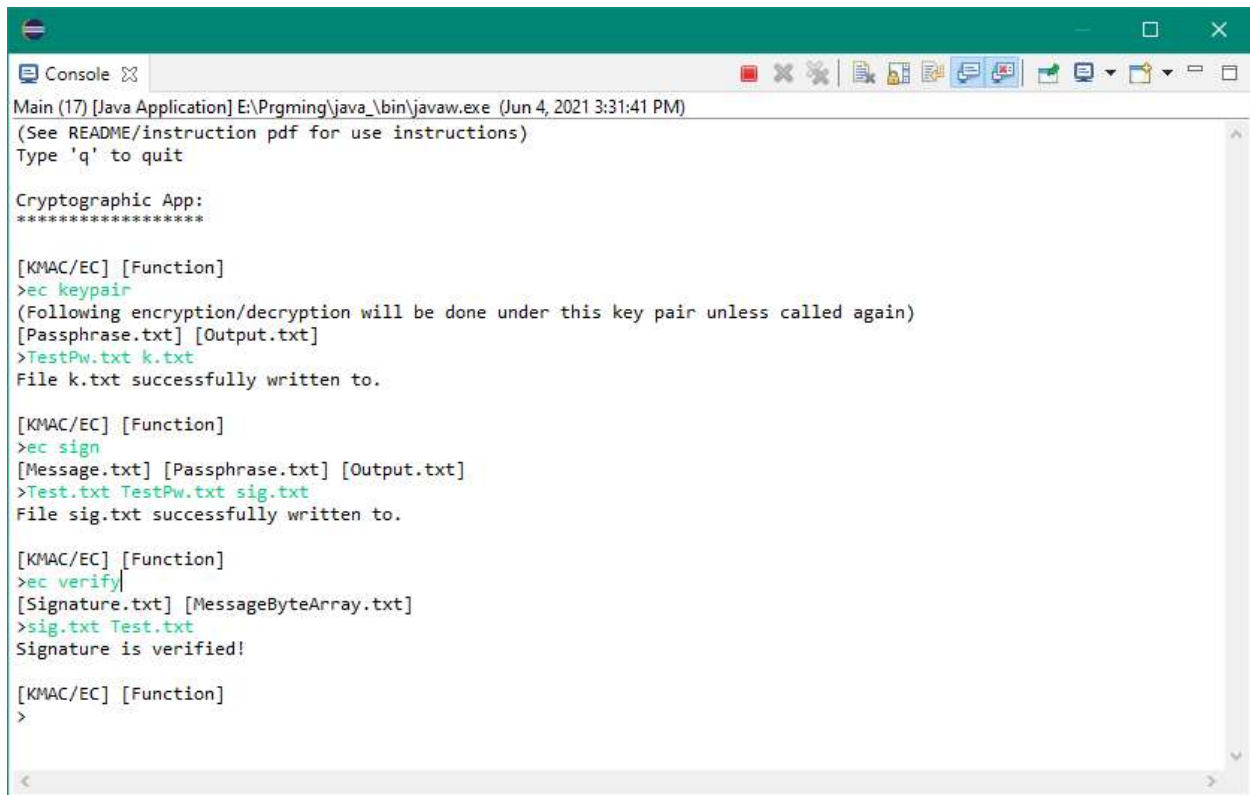
Cryptographic App:
*****
|
[KMAC/EC] [Function]
>EC keypair
(Following encryption/decryption will be done under this key pair unless called)
[Passphrase.txt] [Output.txt]
>Testpw.txt keypair.txt
File keypair.txt successfully written to.

[KMAC/EC] [Function]
>EC encrypt
(Uses previously generated keypair)
[Message.txt] [Output.txt]
>Test.txt enc.txt
File enc.txt successfully written to.

[KMAC/EC] [Function]
>
```



Also, signature verification is confirmed through the console. There is no output file.



```
Console
Main (17) [Java Application] E:\Prgming\java\bin\javaw.exe (Jun 4, 2021 3:31:41 PM)
(See README/instruction pdf for use instructions)
Type 'q' to quit

Cryptographic App:
*****

[KMAC/EC] [Function]
>ec keypair
(Following encryption/decryption will be done under this key pair unless called again)
[Passphrase.txt] [Output.txt]
>TestPw.txt k.txt
File k.txt successfully written to.

[KMAC/EC] [Function]
>ec sign
[Message.txt] [Passphrase.txt] [Output.txt]
>Test.txt TestPw.txt sig.txt
File sig.txt successfully written to.

[KMAC/EC] [Function]
>ec verify
[Signature.txt] [MessageByteArray.txt]
>sig.txt Test.txt
Signature is verified!

[KMAC/EC] [Function]
>
```

Function Summary Table

Function	Input(prompt 1)	Input(prompt 2)	Output
Plain Hash	KMAC plainhash	Msg.txt out.txt	Writes to out.txt
MAC tag	KMAC authentication	Msg.txt pw.txt out.txt	Writes to out.txt
Symmetric Encrypt	KMAC encrypt	Msg.txt pw.txt out.txt	Writes to out.txt
Symmetric Decrypt	KMAC decrypt	Cgram.txt pw.txt out.txt	Writes to out.txt
Make key pair	EC keypair	Pw.txt out.txt	Writes to out.txt, makes keypair
Curve Encrypt	EC encrypt	Msg.txt out.txt	Writes to out.txt
Curve Decrypt	EC decrypt	Cgram.txt out.txt	Writes to out.txt
Sign with Schnorr	EC sign	Msg.txt pw.txt out.txt	Writes to out.txt
Verify signature	EC verify	Signature.txt Msg.txt	Prints if verified or not

All inputs/outputs are as single byte arrays. (Eg, (Z,c,t) is written to out.txt as Z.bytes || c || t)