

For Part 1, the input and output files can be specified. I have used plaintext.txt, ciphertext.enc and out.txt when testing but it works with any file. As demonstrated the user can chose to encrypt or decrypt by using enc or dec and the key and initvector as both encoded in Base64

This testing shows that it meets the requirements of what is required for part 1.

Design Decisions.

For Part 1, not many design decisions were made beyond extending the existing code that was provided. In the provided code encryption and decryption were done in the main method and the user couldn't choose which one they wanted to do. A file (not chosen by the user) was encrypted and then it was decrypted and placed in a random folder.

I chose to move the encryption out of the main method and create a method to encrypt and a different method to decrypt. This meant that one could be called without doing the other. The main method now reads in the arguments from the user and provides them to the appropriate method depending on what the user wants to do. This also allows the user to specify the input and output files that they want to use. I also changed the key and initvector to be being encoded with Base64.

Part 2

Testing to ensure it functions correct.

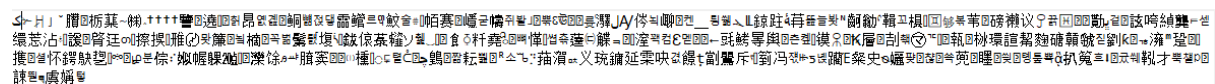
When testing Part 2, I have used the same plaintext file as above.

For Part 2, the program needs to be able to encrypt by being given a key and using that to encrypt.

To do this the command is `java FileEncryptor enc (Base 64 key) (file to encrypt) (file to save to)`

```
$ java FileEncryptor enc 7scC357IBSX0TyT49UIfxA== plaintext.txt ciphertext.enc
Random key=7scC357IBSX0TyT49UIfxA==
initVector=BS8DQi1w3lJFTGWA0j9QoQ==
Encryption finished, saved at ciphertext.enc
```

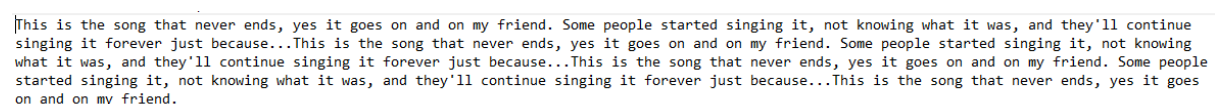
As shown, the key that is provided is the key that is used to encrypt the file. The encrypted file when opened with notepad looks like this;



This can then be decrypted with the same command as Part 1.

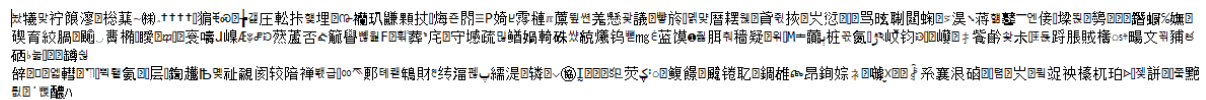
```
$ java FileEncryptor dec 7scC357IBSX0TyT49UIfxA== BS8DQi1w3lJFTGWA0j9QoQ== ciphertext.enc out.txt
Decryption complete, open out.txt
```

The output file then looks like this;

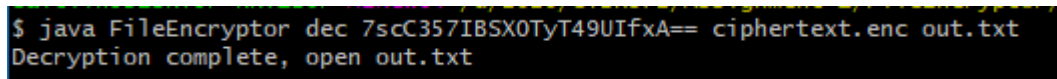


This means that the program had use the key provided and then it can decrypt that file.

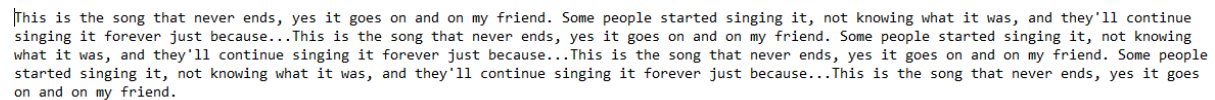
For Part 2 the program also needs to be able to decrypt without being provided the IV value. To test this I am going to use a ciphertext.enc file that looks like this;



The command to decrypt without an initvector is java FileEncryptor dec (Base64 key) (file to decrypt) (file to save to)



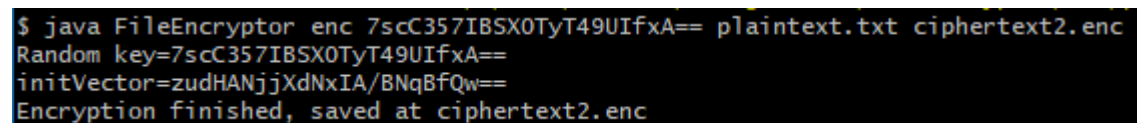
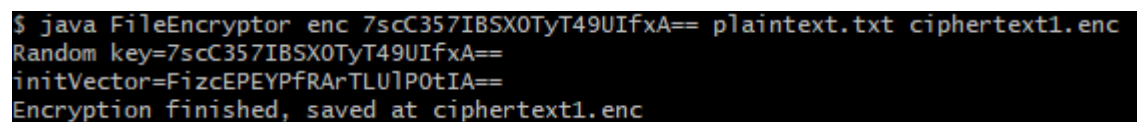
It has managed to decrypt the file without the IV and the out file looks like this;



This means that it was successful at managing to decrypt the file without the key. This also means that it meets all the requirements of Part 2. It also shows that I can decrypt a file that was previous encrypted as this was a file that previous encrypted.

Prove that CipherText is Different

I have encrypted 2 files with the same key;



Using Hexdump ciphertext1.enc looks like this;

file name: ciphertxt1.enc
mime type:

```
0000-0010: 16 2c dc 10-f1 18 3d f4-40 ad 32 d4-94 fd 2d 20
0000-0020: 41 45 53 20-31 32 38 20-20 20 20 20-20 20 20 20
0000-0030: 4e bf 38 af-9c ca 6b 95-23 34 fd 81-59 30 25 3d
0000-0040: be b1 ef 61-06 95 99 0e-c8 cf 89 af-ca 24 87 9b
0000-0050: 53 e5 6f 3f-aa b3 6d 8b-31 94 ca 68-f4 db 39 54
0000-0060: fb 81 b2 d7-1d 50 0d c8-e0 fb 77 be-10 ef cc 2c
0000-0070: 1a 25 76 a6-39 68 a8 30-7d 76 58 eb-a6 15 e7 8f
0000-0080: fd 51 06 e8-73 07 7f 1f-3d c4 2c c3-77 c5 55 4e
0000-0090: cb 58 6b 78-bb 07 75 cc-db be 48 30-f1 5f f3 50
0000-00a0: 35 47 9b 91-17 06 40 cf-3c 6f fa 38-1b 4c 5b aa
0000-00b0: 7a d3 10 ed-1a a6 6d 48-84 c9 47 9a-bb c6 a1 6a
0000-00c0: dd 0c 2d 72-6f 42 d9 d7-68 42 27 e3-b1 76 14 4e
0000-00d0: 3c b1 a2 36-84 dc 08 c3-be 16 4f cc-18 80 f9 31
0000-00e0: 0c f6 0e 85-4f 92 c7 9a-87 a0 09 3b-2f 71 40 77
0000-00f0: 52 da e3 6e-1b de a6 a6-f3 d3 22 23-a3 e8 82 9f
0000-0100: b2 8e cf 47-53 d3 39 b5-65 c9 22 68-ed 68 f3 eb
0000-0110: 2f b7 a2 8c-ba d6 49 95-d0 50 a9 52-f3 4a 7b a7
0000-0120: 4a a3 67 92-d4 fa 49 8c-ea 61 1f c8-2e 7e 99 9e
0000-0130: 86 18 6a 1c-7b 2d 23 96-c4 17 ef 5d-4c e0 09 cb
0000-0140: 61 86 b0 49-11 7e da 59-5c e4 8a c6-ed b7 cf 94
0000-0150: c0 be 7f da-07 e2 84 66-81 98 d0 10-ee 9e 45 cd
0000-0160: 49 da 40 13-95 5a 1f 47-fc 9a a1 ff-b1 c2 b8 51
0000-0170: 7c 21 f9 37-60 00 db fd-13 81 c0 35-c5 a4 d2 eb
0000-0180: 9e bf 83 21-c4 3d db 1d-9c c7 9c 71-2c 11 fe cd
0000-0190: 6c 0c b7 42-ab 20 d5 a8-d5 bd a1 a3-9a 6c 69 9e
0000-01a0: a4 01 a1 92-17 c8 63 8a-15 ba 70 6c-ec 76 b3 c1
0000-01b0: df dc eb 9b-66 b6 9b 02-65 af 40 e4-58 37 7a 95
0000-01c0: de bb 0f fc-6f 44 39 0a-2d df 17 fe-a4 04 25 9a
0000-01d0: 0d 99 9b 03-0b d6 01 ce-e4 27 09 d4-77 59 db bd
0000-01e0: 14 68 56 99-46 dd 81 0f-85 a8 e5 bf-40 33 a4 08
0000-01f0: ba 44 87 fa-dc b5 9b 03-3d 32 10 ef-9f 8a 20 f9
0000-0200: 00 cd 90 5b-c0 8c 9f 3a-5b 11 69 af-87 62 a8 f8
0000-0210: 9e 5a d5 5e-93 e6 9f 54-2d e0 a0 af-b6 eb a0 30
0000-0220: 59 23 38 a3-fd b5 af b5-4f b0 3e 15-2d 14 1a c9
0000-0230: c9 95 8b 22-3f 1a 8d a5-51 a2 42 5a-05 67 ab bd
0000-0240: b1 ac 3a fe-b4 b4 04 77-0b 6c 99 3e-56 36 b2 6b
0000-0250: 96 d7 21 d5-c7 8d 73 61-0b e2 37 d7-c7 8e 64 d9
0000-0260: 98 58 33 1c-df 84 de 5f-96 bd 4c c6-4d c0 4d 60
0000-0270: 27 f4 b4 55-65 05 cb f2-00 dc 02 7b-d7 d1 50 b0
0000-0280: e8 50 e8 9a-80 a6 55 9b-7d 4a c7 ef-bb a0 42 2f
```

And ciphertxt2.enc looks like this;

```
file name: ciphertext2.enc
mime type:
```

```
0000-0010: ce e7 47 00-d8 e3 5d d3-71 20 0f c1-36 a0 5f 43
0000-0020: 41 45 53 20-31 32 38 20-20 20 20 20-20 20 20 20
0000-0030: 59 95 20 1f-9f d3 bd eb-61 47 6e 3e-b0 30 48 60
0000-0040: 9b ae e7 d7-8d 6c f1 d0-50 fa 4c 16-f3 99 61 4a
0000-0050: 1a bf ea e8-9b bd f9 22-8d 94 75 57-64 b6 43 9d
0000-0060: 83 70 ea 90-0d ed c8 0b-79 f5 5b 09-68 0a 71 fe
0000-0070: 87 d2 7b b8-22 69 db 10-de 20 8f e6-01 20 2e 1c
0000-0080: cc 82 2d 0f-cf e1 0c 51-e5 49 66 b0-54 9c ea e5
0000-0090: 35 0e a4 58-42 06 f1 86-44 4e 07 52-77 01 f3 0a
0000-00a0: 4b 30 c1 e4-b3 a9 26 ef-8e 80 08 53-d6 b1 9d a4
0000-00b0: 60 8d c2 49-26 e7 4c 88-d5 b2 14 1d-fe e0 6b e0
0000-00c0: 91 f8 cf f7-a4 82 79 be-67 29 e2 79-63 2c b4 93
0000-00d0: 94 15 3b cb-c7 48 63 b9-3a f6 57 b0-3c 7f 50 78
0000-00e0: 22 13 22 19-0b e1 15 48-a3 1a 8b 51-14 36 59 2c
0000-00f0: ba 8c ad d6-09 85 b1 82-8f db d1 90-f3 e8 30 0b
0000-0100: 3a 40 22 0d-f6 7d fa 24-a6 3e e8 33-73 98 4b 72
0000-0110: 07 5d 82 e8-c6 84 f0 1a-92 fd c4 5c-17 04 a4 a1
0000-0120: 77 55 5f 2b-59 cc b1 8c-23 5d 1b 0f-fa 1f 49 98
0000-0130: 2c 6f c1 53-f3 e4 8f 3a-03 d6 b8 e0-9c d7 b5 d0
0000-0140: 65 b5 dd 94-6d bc 1e 09-64 96 d9 34-81 4e 78 12
0000-0150: 50 d6 15 24-ec 78 55 98-7b 29 5e 61-ee be 09 08
0000-0160: 90 b8 ac b0-38 39 98 24-b3 24 8b 23-85 52 c7 a4
0000-0170: dd be 45 a8-20 97 77 fa-98 6a bf 95-05 96 7c 49
0000-0180: 53 da f1 c0-54 a7 84 97-15 c1 39 2e-47 15 ac b4
0000-0190: 89 1f 04 77-d5 4d f5 2d-b7 94 63 b8-63 c7 51 84
0000-01a0: 43 4d 62 50-4a 3f 36 94-65 79 d8 6b-6b 6d 56 3d
0000-01b0: 94 6d c7 81-1a 40 12 96-ac b5 9d 9c-6a 22 b4 44
0000-01c0: 3c 5e a1 41-a5 93 bc 9d-01 c4 e9 51-4f 11 16 22
0000-01d0: f4 6c d5 23-1e d9 f6 26-c3 a6 73 5b-8c f3 6c da
0000-01e0: a7 3d db 03-67 de a5 0c-15 c3 f3 a4-8b c5 a6 fa
0000-01f0: 85 8b fd d7-c3 95 04 61-0f 3c df ff-52 68 02 3a
0000-0200: 13 4f 60 0c-96 46 d5 8b-e6 fc a4 53-c2 8b 4e 11
0000-0210: dc d0 1b cc-0b cf 0f ff-c9 36 61 b8-c2 e5 e4 32
0000-0220: e3 55 f0 6a-84 45 68 71-d0 6b cb c6-ce bf f9 b3
0000-0230: 37 ef b9 21-6e 8d 3f 5f-e4 6a 94 7f-30 3f 91 7e
0000-0240: 0d 60 7c 73-ae d8 f2 92-1b 8d f5 95-87 b8 dc 0c
0000-0250: a9 4f 0e d9-80 45 c9 6a-7b 62 b4 91-10 c1 48 4c
0000-0260: 6c 35 d9 4a-52 8d 1f 12-b8 54 ba 1e-91 a2 7e 3c
0000-0270: e2 77 4c 89-b9 97 67 97-e2 4f af d4-61 68 6f 81
0000-0280: 00 ae cd 74-12 b9 37 bd-71 d4 49 d1-11 4a be 48
```

This shows that even though these two files were encrypted using the same key, they produce different cipher text each time.

Design Decisions

Since I was extending the program more, I added another method to encrypt with a key and decrypt without an IV. I know which encrypt and decrypt method to use by counting how many arguments there are.

The major design decision for Part 2 was how to store the IV and whether to keep it secret. I decided to store the IV at the beginning of the file that I was encrypting. For Part 1, I would just write the encrypted cipher text to the file but for Part 2 I write the IV as a byte array to the file before writing the encrypted cipher text. When I decrypt, I then read that first bytes out of the file and that is the IV to use.

I have chosen to do this for a couple of reason. It means that the file can decrypted at any time, even if it is much later. If the program stored all the key and IV pairs used then there would be issues as the program could lose them. It also allows the program to encrypt multiple files with the same key,

as the different IV values for each file are stored in the respective file and it means that the file and the program don't have to know that the same key was used multiple times.

I have not encrypted the IV in the file. Since it is written in bytes, if someone was to open the file with a text editor such as Notepad, it just looks like the rest of the file. Someone has to start digging further before being able to tell what the IV is. It also isn't all of the solution to decrypt the file. The key is still needed to decrypt the file and it isn't stored in the file. This means that even if someone got hold of the file and found the IV, they still couldn't decrypt it. This means that it is not as important to encrypt the IV so I have chosen not to.

Part 3

Testing to ensure it functions correct.

For Part 3, the program needs to be able to encrypt using a password and the decrypt using that password.

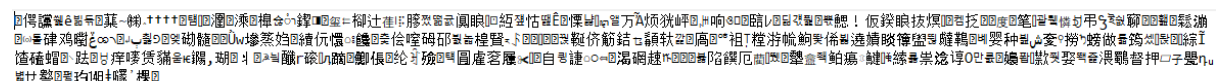
To test this, I am using the same plaintext as above.

The command to test this is `java FileEncryptor enc "password" plaintext.txt ciphertext.enc` (File to encrypt) (File to save to)

```
$ java FileEncryptor enc "thisismypassword" plaintext.txt ciphertext.enc
Random key=JX8YXB6E305CqjVw9tAdKw==
initVector=ofZUUJyLqcQHAK6+xrPW6Q==
Encryption finished, saved at ciphertext.enc
```

This shows that it has managed to encrypt the file using that password.

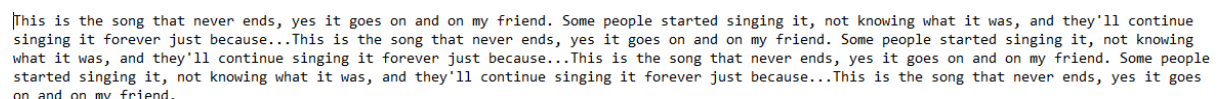
The encrypted file looks like this;



This shows that it has successfully managed to encrypt the file. However, to truly prove that this has encrypted correctly, I need to decrypt it with the same password. The command to test this is `java FileEncryptor dec "password" ciphertext.enc out.txt` (File to decrypt) (File to save to)

```
$ java FileEncryptor dec "thisismypassword" ciphertext.enc out.txt
Decryption complete, open out.txt
```

The program has managed to decrypt the file. And the out.txt file looks like this;



This means that it was successful at both encrypting and then decrypting using a password that the user has provided. This means that it meets the requirements for Part 3.

Design Decisions

For Part 3, I added another encrypt method and another decrypt method. Both methods take a password that is used to create a key that is used for encryption. The key is created by taking the password, adding salt, iterate and hash repeatedly. Since both the decrypt and encrypt method are using the same password, salt and iterating the same amount, they both produce the same key. This means that the encryption and decryption are successful. As with Part 2, the decryption method gets the salt that the encrypt method used by reading it in from the encrypted file.

Part 4

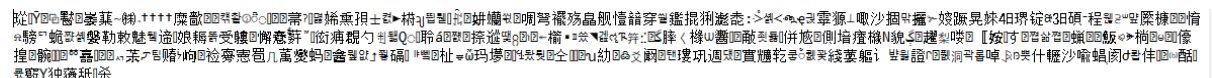
Testing to ensure it functions correct.

For Part 4, the user can chose what encryption algorithm they want to use, AES or Blowfish, as well as the key length to use. To test this, I am using the same plaintext as above.

To encryption with AES and a key length of 128, the command is, java FileEncryptor enc AES 128 "password" (File to encrypt) (File to save to)

```
$ java FileEncryptor enc AES 128 "Thegreatpassword" plaintext.txt ciphertext.enc
Random key=jMaJio8vzMBYwFpDazbw1w==
initVector=3D6f+ukfsuPJkX9APuUjXQ==
Encryption finished, saved at ciphertext.enc
```

This has managed to encrypt using AES and a key length of 128. This is proven as the encrypted file looks like this;



To prove that this has correctly worked the file can be decrypted. The command to do this is java FileEncryptor dec "password" (File to decrypt) (File to save to)

```
$ java FileEncryptor dec "Thegreatpassword" ciphertext.enc out.txt
Decryption complete, open out.txt
```

It successfully work, it doesn't need to be given the encryption algorithm or the key length.

This is the song that never ends, yes it goes on and on my friend. Some people started singing it, not knowing what it was, and they'll continue singing it forever just because...This is the song that never ends, yes it goes on and on my friend. Some people started singing it, not knowing what it was, and they'll continue singing it forever just because...This is the song that never ends, yes it goes on and on my friend. Some people started singing it, not knowing what it was, and they'll continue singing it forever just because...This is the song that never ends, yes it goes on and on my friend.

It has definitely worked as the out.txt file is the same as the plaintext.

To encryption with Blowfish and a key length of 256, the command is, java FileEncryptor enc Blowfish 256 "password" (File to encrypt) (File to save to)

```
$ java FileEncryptor enc Blowfish 256 "Thegreatpassword" plaintext.txt ciphertext.enc
Random key=IvSbPrh2+aIQRjhodcvI9tjtfoNgydmrg5MiZ+jur9w=
initVector=G1xmG72Uy1HCKPwPdILmIA==
Encryption finished, saved at ciphertext.enc
```

It has managed to successfully encrypt using Blowfish and 256 as the key length. As you can see the key for this encryption is longer than the key for the previous encryption which is only 128.

!Eß7^7^7^7^_2cIÜM+.,ç!âü«tþb;1ZÞj»°-d-ZÄ ð!d/»YÿÖ°K&i>S_D³Yh+gy µtë¶¶,vcþ1-R«¹VR1ð1ðsk&öñ⁹4ÝíÊä
 ZÞD0ð8weYwYwXWvjjj1ÜîÉ,éÖDY+A[~°PpAl.#g.,<#⁺F¶¶¶qal=ʼN¹iÊAšêá1Ü.BüÊ²pêbsk|/;8r8R[R«³·b-ò-to)Fäv¶¶¶0²æ7þbµ¶.,@·V8X⁹a8üZüpp¶¶r⁹³J5Ü:TŠÜx«||:øð!ðÜM).3|
 c<tIKS1ik 1¶¶,ip|cl-DpgÖð*D9sñ<|ÉÉ+Fvâû&Y|¶¶=-ÎZ[z˙%KÐ3³0CDU<ÜÇJÜ);E|eÜ0n6j+oÅ
 lPⁱ-i-.O!tþB
 tþTz1ÉÁ;HJD⁴T.<{b|ñ|÷-}i=m¶Y+m³ Ål¶⁹⁹Kegð.-ÔÂQKÖ
 Ünå0.[0IÑ⁷75KÜq¡AôYd.s{«\$1ª¹1¶⁻⁻#°cë UðâH,sPxH⁹âó-è¶¶¶Npïð¹

To prove that this has correctly worked the file can be decrypted. The command to do this is `java FileEncryptor dec "password" (File to decrypt) (File to save to)`

```
$ java FileEncryptor dec "Thegreatpassword" ciphertext.enc out.txt
Decryption complete, open out.txt
```

It has managed to decrypt the file, again without telling it the encryption algorithm or key length and it prove that it worked, the out.txt file looks like this,

[This is the song that never ends, yes it goes on and on my friend. Some people started singing it, not knowing what it was, and they'll continue singing it forever just because...This is the song that never ends, yes it goes on and on my friend. Some people started singing it, not knowing what it was, and they'll continue singing it forever just because...This is the song that never ends, yes it goes on and on my friend. Some people started singing it, not knowing what it was, and they'll continue singing it forever just because...This is the song that never ends, yes it goes on and on my friend.

This proves that the program can encrypt with either algorithm and a given key length.

The other Part 4 that needs to be done is allow the user to query the algorithm and key length that has been used for an encryption file. The command to do this is, `java FileEncryptor info (encrypted file)`

```
$ java FileEncryptor info ciphertext.enc
Blowfish 256
```

This was for a file that I had previous encrypted and it successfully tells the user that the algorithm used was Blowfish and that the length was 256.

This shows that the program meets all the requirements of Part 4.

Design Decisions

Like with the previous parts, I have added additional methods to be able to encrypt with different algorithms and key lengths as well as provide the metadata of an encrypted file to the user.

I have embedded the metadata in a similar way to the way that I stored the IV value for Part 2. I combined the algorithm name and key length into one string and then stored that as a byte array before writing to the file. This means that it can be read out of the file in the same way as the IV. This means that the decrypt method can get access to the algorithm used and the key length. This

means that the program can decrypt the file without the user telling it what algorithm and key length were used because it is in the file.

As discussed above, I had stored it in this way so that the decrypt method knows that algorithm and key length to use and it means that the file can be decrypted at any time later as the information that the program requires are stored in the file. It is also secure because it is embedded in the file and it still doesn't give an outside attacker all the information that they need to decrypt the file themselves. The algorithm and key length don't tell the attacker exactly what the key is which means that they aren't going to be able to decrypt it themselves. This means that it is secure.

Overall Design Decisions

The program originally used `Log.INFO` to provide information to the user. The issue with this is that it gives some information away as to what method is being called and what is being done. In order to now release this information, I have changed all of these statements to `System.out.println` statements. It provides the same functionality to the user as it tells them what they need to know but it doesn't tell them what method was called which is not something that the user requires. This makes the program slightly more secure.

There are a number of errors that could be thrown throughout the program if the user doesn't provide valid arguments. If an error isn't handled it can provide a stack trace which shows what method it originated in and what caused the error. This is not something that I want a user to see, so I have handled all the errors with catch statements and provided nicer print statements to the user. This has two benefits. The first is that it no longer prints the stack trace and it is more understandable for the user so they can understand what they need to change in order to get the program to run correctly.

Sensible default values are also provided. The default encryption algorithm is AES and if the key length isn't specified then a default of 128 is used. This means that the user isn't required to include all that information if they don't want to and the program will still function correctly.

Code is also included to prevent data duplication. This means that the `FileEncryptor` class can't be cloned, serialized or de-serialized.