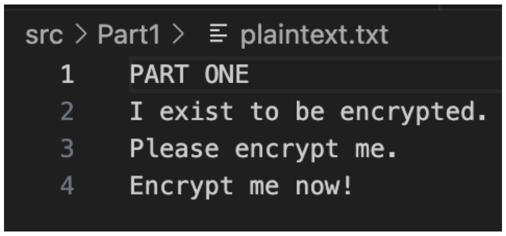
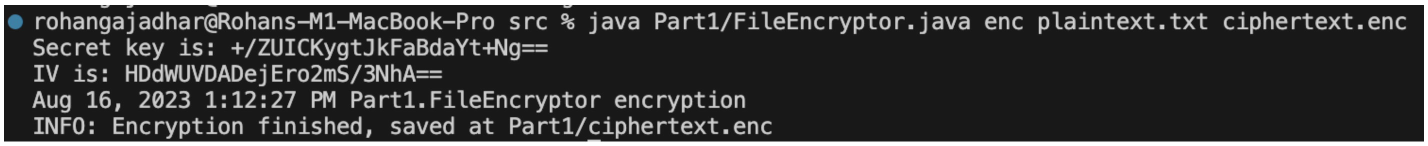
## <u>Part 1 - Output</u>



```
src > Part1 > plaintext.txt
PART ONE
I exist to be encrypted.
Please encrypt me.
Encrypt me now!
```

To encrypt a file ‘plaintext.txt’ to ‘ciphertext.enc’ we can run the following command in the terminal, from the <u>Assignment1/src directory</u>.

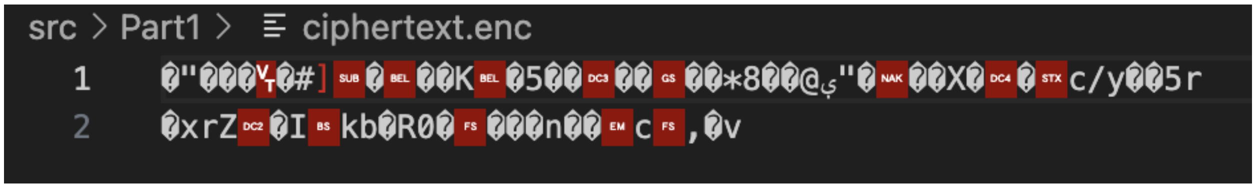
*The plaintext.txt file to be encrypted.*



```
rohangajadhar@Rohans-M1-MacBook-Pro src % java Part1/FileEncryptor.java enc plaintext.txt ciphertext.enc
Secret key is: +/ZUICKygtJkFaBdaYt+Ng==
IV is: HDdWUVDADejEro2mS/3NhA==
Aug 16, 2023 1:12:27 PM Part1.FileEncryptor encryption
INFO: Encryption finished, saved at Part1/ciphertext.enc
```

*The command to encrypt a file, and the resulting output.*

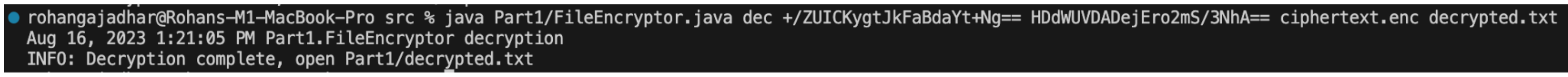
Once the file is successfully encrypted the secret key and initialization vector (IV) are printed to the terminal, as the user requires these to decrypt the file.



```
src > Part1 > ciphertext.enc
�"���V�#]SUB�BEL��KBEL�5��DC3��GS��*8��@ي"�NAK��X�DC4�STXc/y��5r
�xrZDC2�IBSkb�R0�FS���n��EMcFS,�v
```

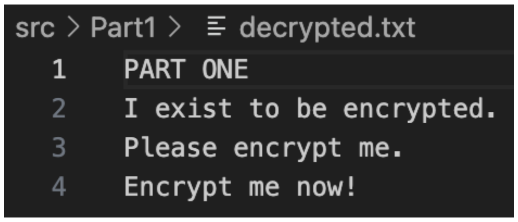
*The contents of the encrypted file ‘ciphertext.enc’, containing non-readable gibberish.*

To decrypt the file ‘ciphertext.enc’ to ‘decrypted.txt’ we can run the following command in the terminal.



```
rohangajadhar@Rohans-M1-MacBook-Pro src % java Part1/FileEncryptor.java dec +/ZUICKygtJkFaBdaYt+Ng== HDdWUVDADejEro2mS/3NhA== ciphertext.enc decrypted.txt
Aug 16, 2023 1:21:05 PM Part1.FileEncryptor decryption
INFO: Decryption complete, open Part1/decrypted.txt
```

*The command to decrypt a file, and the resulting output.*



```
src > Part1 > decrypted.txt
PART ONE
I exist to be encrypted.
Please encrypt me.
Encrypt me now!
```

Once the file is successfully decrypted, the file ‘decrypted.txt’ is generated. This contains the decrypted contents of ‘ciphertext.enc’, identical to ‘plaintext.txt’ proving that the encryption/decryption process preserved the contents of the file.

*The contents of ‘decrypted.txt’, the output of the decrypted ‘ciphertext.enc’.*

## Part 1 - Discussion

The base code is extended by creating two methods, for encryption and decryption respectively. This allows the user to specify whether to encrypt or decrypt a file. The main method takes command line arguments to determine which mode to run in.

For encryption mode; the user must enter 3 arguments:
[1] “enc” (to initiate encryption mode)
[2] the file to encrypt
[3] the name of the file to generate that will contain the encrypted contents of [2]

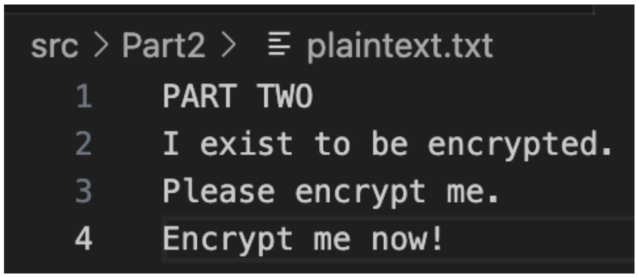
The generated secret key and iv (initialization vector) are converted to base64 and are then printed to the terminal, as the user requires these as command line arguments when decrypting.

For decryption mode; the user must enter 5 arguments:
[1] “dec” (to initiate decryption mode)
[2] the base64 secret key
[3] the base64 iv
[4] the encrypted file to decrypt
[5] the name of the file to generate that will contain the decrypted contents of [4]

### Util.java

Note that the Util.java file is not included in any of the parts. This was causing a multitude of issues when trying to run the encryption/decryption program. As it only contained one method that was used for formatting bytes I decided that it was not necessary to include.

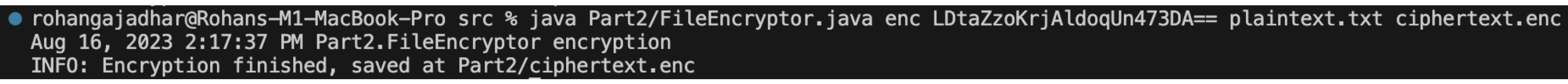
## Part 2 - Output



```
src > Part2 > plaintext.txt
PART TWO
I exist to be encrypted.
Please encrypt me.
Encrypt me now!
```

To encrypt a file ‘plaintext.txt’ to ‘ciphertext.enc’ we can run the following command in the terminal, from the Assignment1/src directory.

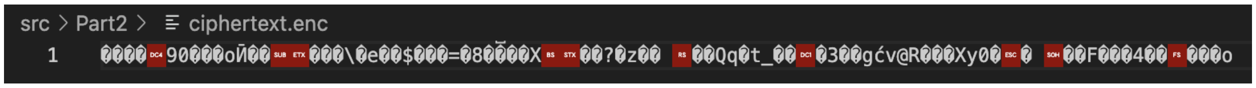
*The ‘plaintext.txt’ file to be encrypted.*



```
rohangajadhar@Rohans-M1-MacBook-Pro src % java Part2/FileEncryptor.java enc LDtaZzoKrjAldoqUn473DA== plaintext.txt ciphertext.enc
Aug 16, 2023 2:17:37 PM Part2.FileEncryptor encryption
INFO: Encryption finished, saved at Part2/ciphertext.enc
```

*The command to encrypt a file, and the resulting output.*

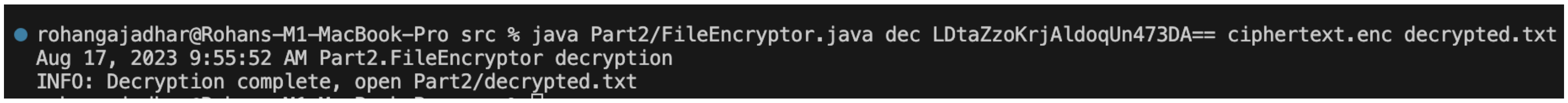
Once the file is successfully encrypted, the ‘ciphertext.enc’ file is generated.



```
src > Part2 > ciphertext.enc
����DC490���oЙ��SUBETX���\�e��$���=�80̆���XBSSTX��?�z��  RS��Qq�t_��DC1�30�gćv@R���Xy��ESC�  SOH��F���40�FS���o
```

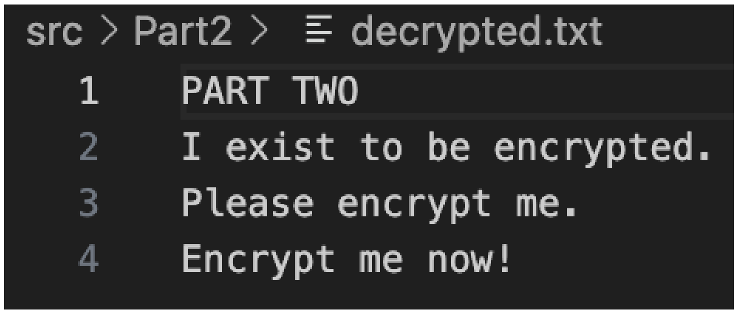
*The contents of the encrypted file ‘ciphertext.enc’, containing non-readable gibberish.*

To decrypt the file ‘ciphertext.enc’ to ‘decrypted.txt’ we can run the following command in the terminal.



```
● rohangajadhar@Rohans-M1-MacBook-Pro src % java Part2/FileEncryptor.java dec LDtaZzoKrjAldoqUn473DA== ciphertext.enc decrypted.txt
  Aug 17, 2023 9:55:52 AM Part2.FileEncryptor decryption
  INFO: Decryption complete, open Part2/decrypted.txt
```

*The command to decrypt a file, and the resulting output.*



```
src > Part2 > decrypted.txt
PART TWO
I exist to be encrypted.
Please encrypt me.
Encrypt me now!
```

Once the file is successfully decrypted, the file ‘decrypted.txt’ is generated. This contains the decrypted contents of ‘ciphertext.enc’, identical to ‘plaintext.txt’ proving
that the encryption/decryption process preserved the contents of the file.

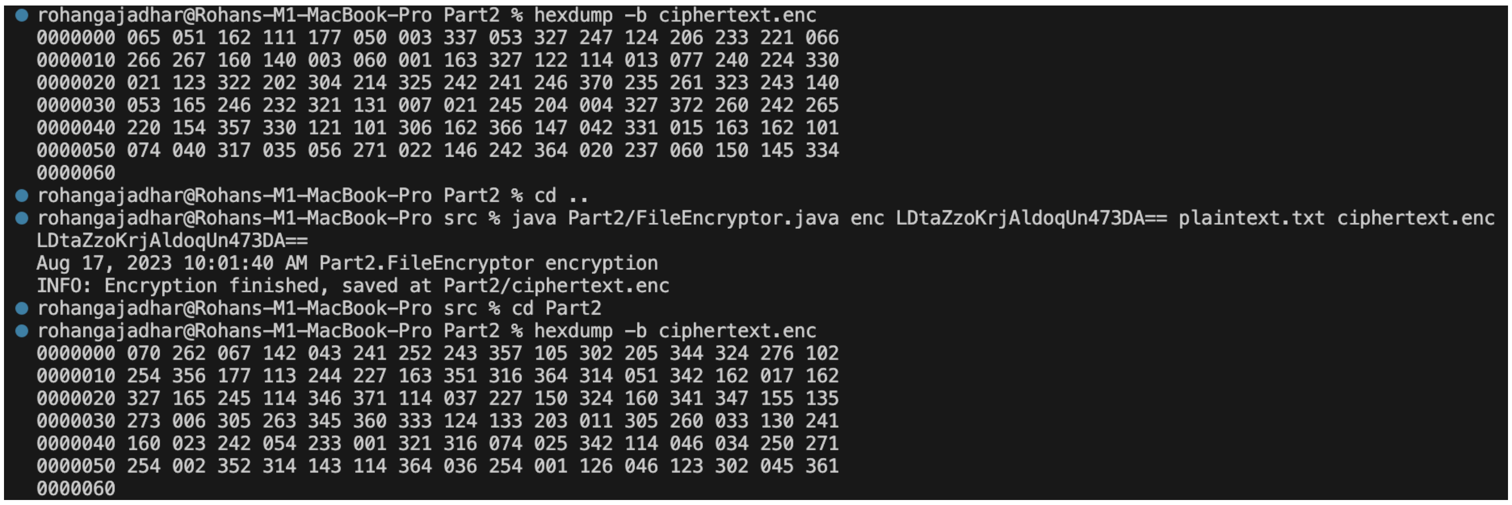
*The contents of ‘decrypted.txt’, the output of the decrypted ‘ciphertext.txt’.*

## <u>Part 2 - Discussion</u>

Part 2 extends part 1 by not requiring the iv to be specified when decrypting the file. The user is also able to specify the base64 secret key. It is important to note that the secret key must be 128 bit, to conform to the AES secret key requirements. The iv is generated when encrypting the file, however it is not printed to the console. As we are operating in CBC mode, the program writes the iv to the first block of the plaintext before encryption.

For encryption mode; the user must enter 4 arguments:
[1] “enc” (to initiate encryption mode)
[2] the base64 secret key
[3] the file to encrypt
[4] the name of the file to generate that will contain the encrypted contents of [3]

For decryption mode; the user must enter 4 arguments:
[1] “dec” (to initiate decryption mode)
[2] the base64 secret key
[3] the encrypted file to decrypt
[4] the name of the file to generate that will contain the decrypted contents of [3]



```
● rohangajadhar@Rohans-M1-MacBook-Pro Part2 % hexdump -b ciphertext.enc
  0000000 065 051 162 111 177 050 003 337 053 327 247 124 206 233 221 066
  0000010 266 267 160 140 003 060 001 163 327 122 114 013 077 240 224 330
  0000020 021 123 322 202 304 214 325 242 241 246 370 235 261 323 243 140
  0000030 053 165 246 232 321 131 007 021 245 204 004 327 372 260 242 265
  0000040 220 154 357 330 121 101 306 162 366 147 042 331 015 163 162 101
  0000050 074 040 317 035 056 271 022 146 242 364 020 237 060 150 145 334
  0000060
● rohangajadhar@Rohans-M1-MacBook-Pro Part2 % cd ..
● rohangajadhar@Rohans-M1-MacBook-Pro src % java Part2/FileEncryptor.java enc LDtaZzoKrjAldoqUn473DA== plaintext.txt ciphertext.enc
  LDtaZzoKrjAldoqUn473DA==
  Aug 17, 2023 10:01:40 AM Part2.FileEncryptor encryption
  INFO: Encryption finished, saved at Part2/ciphertext.enc
● rohangajadhar@Rohans-M1-MacBook-Pro src % cd Part2
● rohangajadhar@Rohans-M1-MacBook-Pro Part2 % hexdump -b ciphertext.enc
  0000000 070 262 067 142 043 241 252 243 357 105 302 205 344 324 276 102
  0000010 254 356 177 113 244 227 163 351 316 364 314 051 342 162 017 162
  0000020 327 165 245 114 346 371 114 037 227 150 324 160 341 347 155 135
  0000030 273 006 305 263 345 360 333 124 133 203 011 305 260 033 130 241
  0000040 160 023 242 054 233 001 321 316 074 025 342 114 046 034 250 271
  0000050 254 002 352 314 143 114 364 036 254 001 126 046 123 302 045 361
  0000060
```

*Generating two different ciphertexts and performing the hexdump -b command on each ciphertext.*

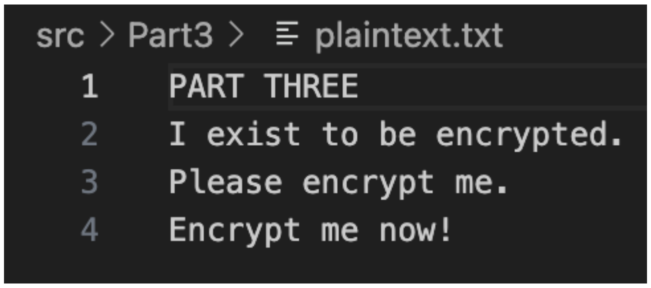
To show that the ciphertext is different each time the file is encrypted, we are able to use the 'hexdump -b' command on 'ciphertext.enc'. This will generate the input offset in hexadecimal format for the contents of the file. The above screenshot illustrates the use of this command on a pre-existing ciphertext/encoded file 'ciphertext.enc' to generate the first hexdump. We can then erase the encoded file 'ciphertext.enc', encrypt a file 'plaintext.txt' to 'ciphertext.enc' once again. We are then able to run the hexdump command on the new ciphertext to show that the hexdump is different each time a file is encrypted, whether the contents of the plaintext change in between encryptions or not.

## **<u>Part 2 - Secure Design</u>**

When encrypting a file in CBC mode, the iv is added to the first block of the plaintext before encryption. The ciphertext generated from this plaintext block is then added to the next block of plaintext before encryption. This process continues until the entire plaintext has been encrypted. The iv does not need to be kept secret, as the intention of an iv is to introduce randomness into the encryption process. If a plaintext were to be encrypted several times, the resulting ciphertext would be different each time due to the unique iv used for each encryption. The security of CBC mode relies on the secret key being kept - *secret*, not the secrecy of the iv. The iv is assumed to be publicly known, therefore encrypting the iv would become redundant.

However, <u>it is</u> important that the iv is random and *unpredictable* for each encryption - as this is the whole reason for an iv. The program uses the *SecureRandom()* java class to generate a 16 byte, byte array which contains the iv. The generation of this array is done each time the program runs, ensuring a new iv is used with every encryption. The *SecureRandom()* class has 128 bits, making it more secure than the more commonly used *Random()* class, as it only has 48 bits. This makes the probability of repeating characters much smaller when using the *SecureRandom()* class.

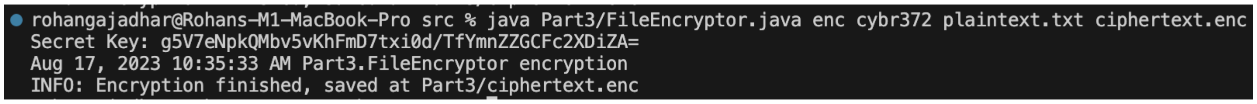
## Part 3 - Output



```
src > Part3 > plaintext.txt
PART THREE
I exist to be encrypted.
Please encrypt me.
Encrypt me now!
```

To encrypt a file ‘plaintext.txt’ to ‘ciphertext.enc’ we can run the following command in the terminal, from the Assignment1/src directory.

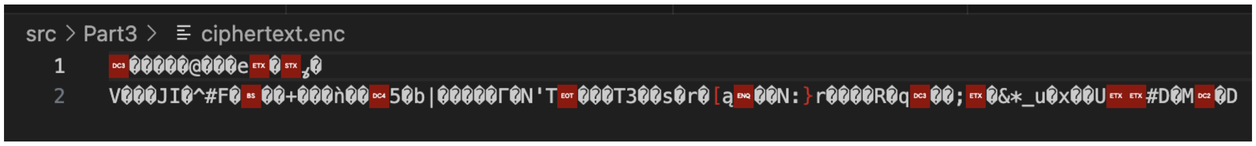
*The ‘plaintext.txt’ file to be encrypted.*



```
rohangajadhar@Rohans-M1-MacBook-Pro src % java Part3/FileEncryptor.java enc cybr372 plaintext.txt ciphertext.enc
Secret Key: g5V7eNpkQMbv5vKhFmD7txi0d/TfYmnZZGCFc2XDiZA=
Aug 17, 2023 10:35:33 AM Part3.FileEncryptor encryption
INFO: Encryption finished, saved at Part3/ciphertext.enc
```

*The command to encrypt a file, and the resulting output.*

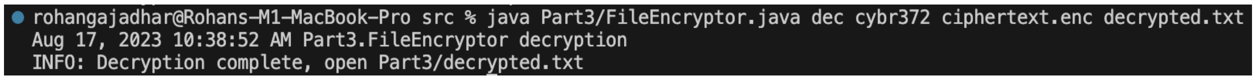
Once the file is successfully encrypted, the ‘ciphertext.enc’ file is generated.



```
src > Part3 > ciphertext.enc
DC3������@���eETX�STX ̧�
V���JI�^#F�BS��+���ǹ��DC45�b|�����Γ�N'TEOT���T3��s�r�[ą ENQ��N:}r����R�qDC3��;ETX�&*_u�x��UETXETX#D�MDC2�D
```

*The contents of the encrypted file ‘ciphertext.enc’, containing non-readable gibberish.*

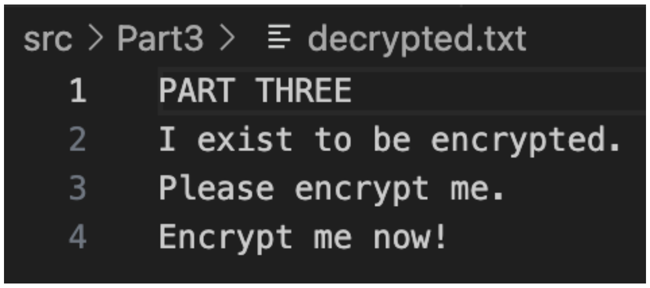
To decrypt the file ‘ciphertext.enc’ to ‘decrypted.txt’ we can run the following command in the terminal.



```
rohangajadhar@Rohans-M1-MacBook-Pro src % java Part3/FileEncryptor.java dec cybr372 ciphertext.enc decrypted.txt
Aug 17, 2023 10:38:52 AM Part3.FileEncryptor decryption
INFO: Decryption complete, open Part3/decrypted.txt
```

*The command to decrypt a file, and the resulting output.*

Once the file is successfully decrypted, the file ‘decrypted.txt’ is generated. This contains the decrypted contents of ‘ciphertext.enc’, identical to ‘plaintext.txt’ proving that the encryption/decryption process preserved the contents of the file.



```
src > Part3 > decrypted.txt
PART THREE
I exist to be encrypted.
Please encrypt me.
Encrypt me now!
```

*The contents of ‘decrypted.txt’, the output of the decrypted ‘ciphertext.txt’.*

## Part 3 - Discussion

Part 3 extends part 2 by allowing the user to specify a password. To account for the possibility that the user chooses a password with low entropy; a salt is added to the password, which is then hashed. When storing the password chosen by the user, a char array is used to increase security. String objects are immutable, meaning that you cannot overwrite the contents of a String after use. This proves insecure for storing the password, as the password determines the secret key, and keeping the secret key - *secret*, determines how secure the encryption process is.

The hashing process operates for 1000 iterations, which is the minimum iteration count required by the most recent *PBEKeySpec()* class. The *PBEKeySpec()* class generates a KeySpec using the password, salt, count and a key length (256). We can now generate the secret key using the PBEKeySpec, and perform the hashing using the “PBKDF2WithHmacSHA256” algorithm for our specified count (1000 iterations).

The salt is stored in a 16 byte, byte array. Much like the iv, it is generated using the *SecureRandom()* class. The salt is then added to the first block of the plaintext before encryption, as is the iv - much like the encryption in part 2. The decryption process is similar to part 2, except this time the password acts as the secret key. Meaning, we have to generate the secret key from the user entered password to authenticate that this is the same password used when the file was encrypted.