

The Project Report: Phase 2

Section 3

ID 3

Mohamed Ayman 900182267

AbdEl-Rahman Fawzy 900183004

Shamel Radwan 900183844

# The Project Report

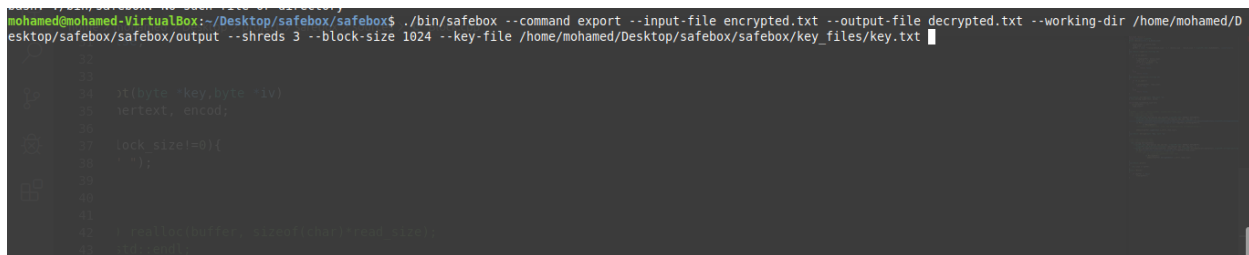
## How to run?

1. You need to put the data.txt in the input file.
2. Know the place you are in, and then put it in makefile.vars by pwd command
3. You need to make sure the parameters are right as the screenshot mentions(not divisible by 2)



```
mohamed@mohamed-VirtualBox: ~/Desktop/safebox/safebox
File Edit View Search Terminal Help
mohamed@mohamed-VirtualBox:~/Desktop/safebox/safebox$ ./bin/safebox --command import --input-file /home/mohamed/Desktop/safebox/safebox/data/data.txt --output-file encrypted.txt --working-dir /home/mohamed/Desktop/safebox/safebox/output --shreds 3 --block-size 1024 --key-file /home/mohamed/Desktop/safebox/safebox/key_files/key.txt
```

4. Make sure to write the right path in input and working directory.
5. Write what number of shreds you need
6. Type the size of each block
7. Run it in the terminal
8. After that you need to export: the screenshot elaborates on what I mean.



```
mohamed@mohamed-VirtualBox:~/Desktop/safebox/safebox$ ./bin/safebox --command export --input-file encrypted.txt --output-file decrypted.txt --working-dir /home/mohamed/Desktop/safebox/safebox/output --shreds 3 --block-size 1024 --key-file /home/mohamed/Desktop/safebox/safebox/key_files/key.txt
```

9. Input file path is the output of the import file. Number of shreds, block size and path of key must be the same
10. Open the decrypted.txt, you will see it is decrypted.

### **Multithreading process:**

In this phase, it is a must to develop what we have worked on in the first phase by creating various threads to handle importing or exporting file blocks to/from the various shreds at the same time. That will guarantee us a more effective process. In addition, we had to make the whole process more secure by spreading the blocks to the various shreds in a random way. Giving to each block an identity. I mean that each block will be known its shred name, block number inside the shred and the random block number in general compared to the other block numbers. It all starts in the SafeBoxImport as we created two objects from the classes FileSpooler, for opening the target file and getting its blocks, and MultiShredManager, calling its encrypt function. Reading the blocks from the file, each thread (which is shred) gets a random number from a Lotteryscheduler object -by withdrawing random numbers- that relates to a certain block and accesses it. Threads work on encrypting the accessed blocks and locating them into the shreds. It is guaranteed that blocks into the shreds are completely ordered randomly. To be able to know where each block relates to which location in the main file for the decryption purposes, we created the multiheaderqueue. By each thread accessing a block, encrypting it and putting it into a certain shred, data like the block number, shred number and its name are stored into a struct and saved in that queue. For more security, we worked on decrypting its data. Going through the functions of the queue, we solved the problem of padding using while loop in the decrypted queue to get the number of encrypted file in case the number of `read_size(sz)% BLOCKSIZE != 0`. Why? It is because the decrypted does not understand this. It needs the same size as the ciphered text. We have done ZERO PADDING to delete the padding idea in the decrypted function.