# HandyCipher- Part 1

## CS-265 Cryptography

“Cryptography is typically bypassed, not penetrated”

-Adi Shamir

A project by

Ravee Khandagale (011028654)

## About the HandyCipher

* The HandyCipher is a challenge authored by Bruce Kallick.
* After a quick glance at the challenge, the immediate conclusion we draw is about the level of confusion this challenge provides!
* This is one of the reasons, that, in two years only two people have even solved this challenge.
* The description talks about a pen-and-paper sort of encryption and decryption.
* However, a simple pen-and-paper just won’t do the trick here!
* We shall talk about a few things in this report aggregated as follows:

1. Description of the cipher
2. Encryption
3. Decryption
4. Key
5. My implementation of the cipher
6. Possible Attacks on the cipher

## Description of the Cipher

The principle of this cipher is homophonic substitution.

This means that one letter can be mapped to more than one character (specifically, one to five characters).

In short, a plaintext of 100 words will result in a cipher text which might range anywhere from 100-500 characters.

There is also a twist here, though.

After substituting, there are nulls which are introduced in the cipher text.

This further increases confusion.

The encryption algorithm is purposely made complicated.

The decryption process is also simple, but, all-in-all, this all makes handy cipher a very complicated cipher to crack.

Both the encryption and decryption processes are based on the encryption and decryption ways of a Core Cipher.

## Encryption

* This is a stream cipher.
* A value is derived for each character by looking at the subkey.
* The value is further broken down into its binary equivalent.
* For example,

for a key value of A= 10,

the binary value= 01010.

* Now, randomly alternating between the rows and columns and diagonals of the key matrix, according to the on and of bits of the diary equivalent, the corresponding elements are chosen as ciphertext.
* Furthermore, a random combination of these values is chosen, so as to add to the confusion of ciphertext.
* You will notice that there can be 1-5 substitute characters for each letter of the plaintext. And any combination of these characters too.
* That is to say, a binary equivalent with one 1 is going to account for only one character as its encrypted equivalent.
* Now nulls are randomly added to all this confusion.

The inclusion of nulls makes it more difficult for a hill climb attack to take place, which the original core cipher was susceptible to.

## Decryption

* The same key table is used as that used for encryption.
* Each cipher text characters are scanned though, one by one and the nulls are removed.
* The cipher text values are located and traced back to the original decimal value from the positions located by the binary representation.
* Practically implementing a decryption, though, one should keep in mind, that this is a homophonic substitution, which is why one should look for characters in the range 1-5.

## Key

* First of all, lets define the useable characters for a key.
* These are a permutation of the characters from A-Z, numbers 0-9 and special characters like ,.-? ^
* There is a 5 x 5 table which is extended to an 8 x 5 table.
* The first five columns are used for encrypting characters.
* The characters in the last three columns are used as padding characters.
* After mapping this into a matrix/ table, we can generate a sub-key of 31 characters. While counting the value of the plaintext character to convert it into a binary equivalent, the numerical values of the 41-character key is omitted.

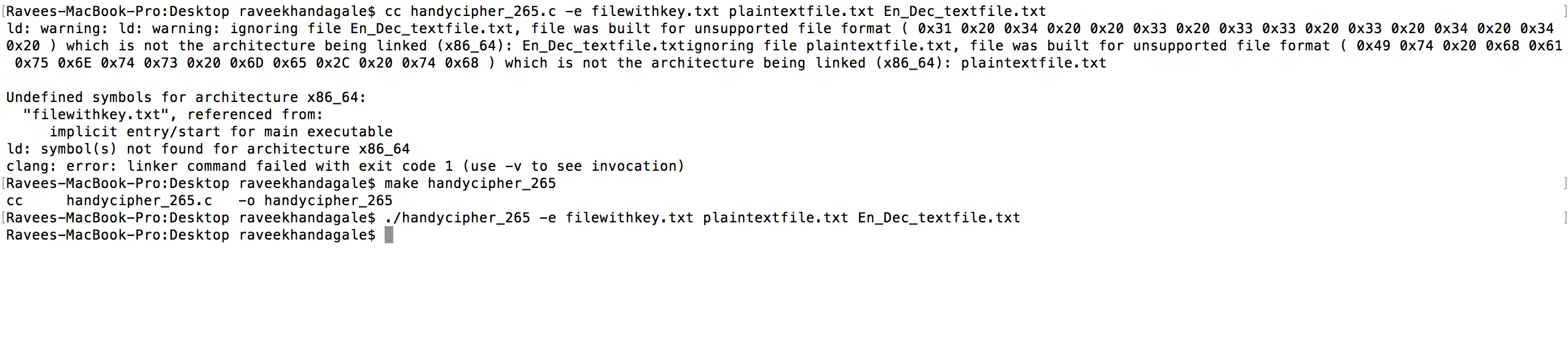
## My Implementation of the Key

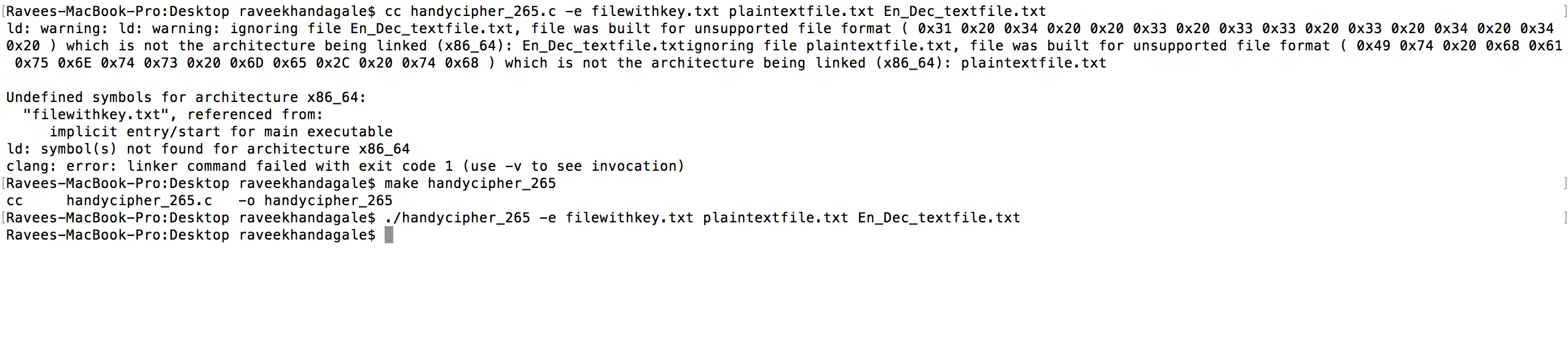
* I have coded a Homophonic substitution cipher, an idea on which handycipher is based- handycipher\_265.c.

Homophonic Substitution cipher increases the property of confusion in the encrypted text.

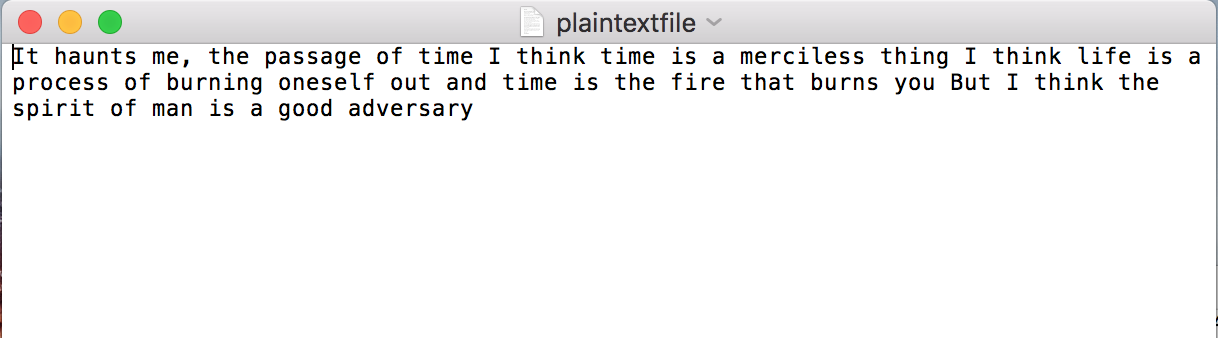
* It contains the following important functions

Results of implementation n terminal:



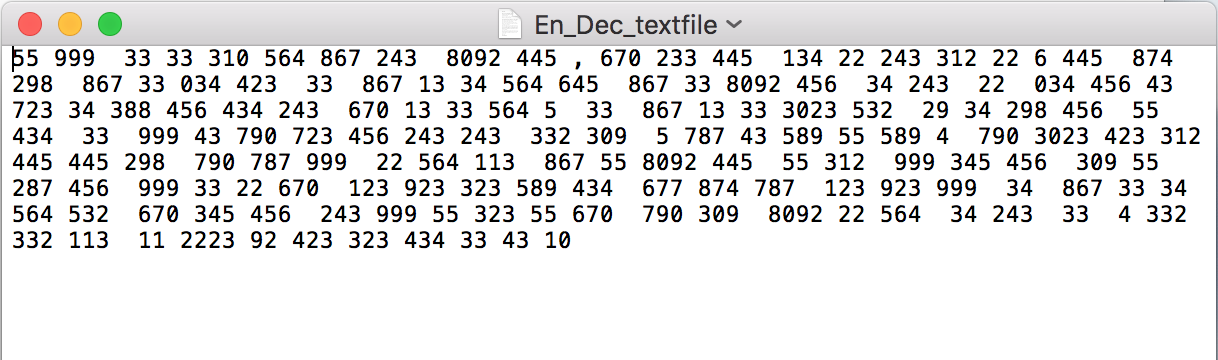


**encryptFile():**

****

* This function takes a streaming character and encrypts it into any of the choices laid out for ciphering that particular character.
* Now, this choice is given in a file called FileWithKey.
* There is a pseudo random generator function which uses a seed value to help randomly select the cipher-character to substitute.
* One of the many choices is chosen as the cipher text.
* As you can see, the cipher text looks obfuscated due to the homophonic property of the cipher.
* The streaming character is encrypted one by one.
* The program shows the representation of such a cipher.

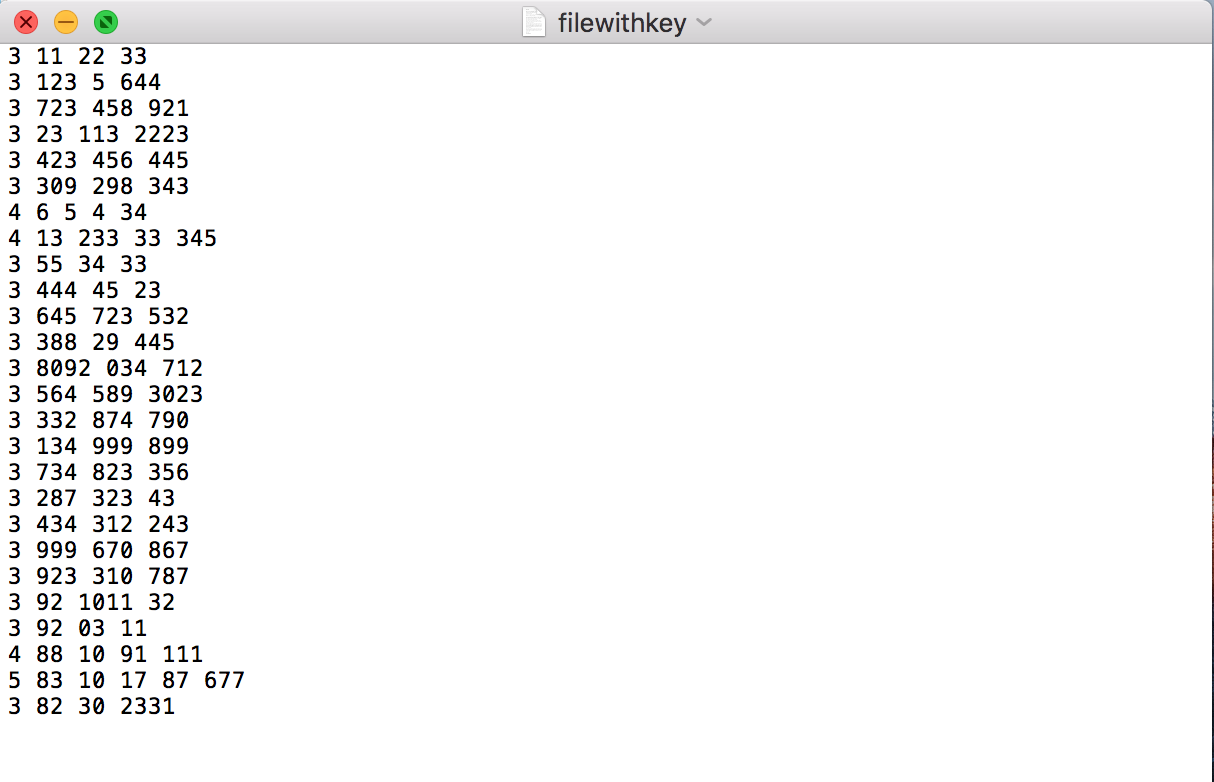
Following is a screenshot of such an encryption



**decryptFile():**

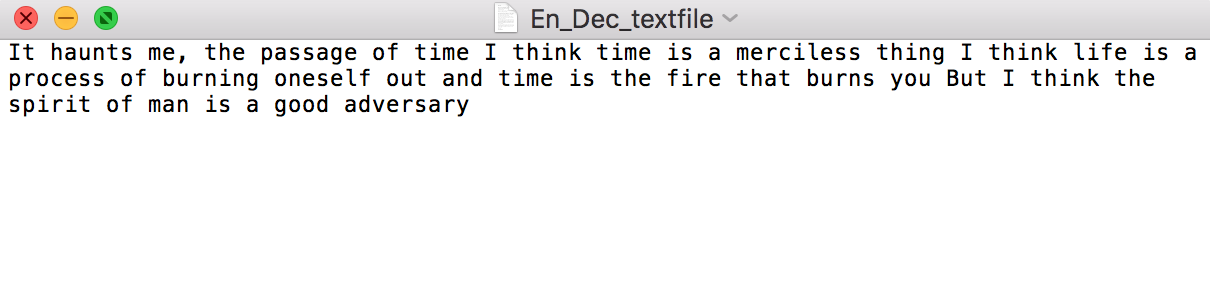
* This function takes as input the cipher text in the encrypted text file.
* This cipher text, which is seemingly confusing, is decrypted by this function.
* It takes each cipher character and decrypts it using the key mentioned in the key file.

Here is the key file



* There are multiple keys for every letter.
* After decrypting, it is written back to the output file called En\_Dec\_textfile.

Following is the screenshot of such a decryption



## Possible attacks on the Cipher

*My version of an attack:*

* The first question that comes to mind of an attacker is how to get rid of the nulls, rather how to identify the null characters in the cipher text.
* The possible nulls are going to be (31 choose the number of nulls employed in the cipher text). One can easily brute force this number in a matter of minutes using a script.
* After the nulls are identified, the rest of the non-null characters of the key can be played with.
* Using probability and a very slight idea of frequency analysis, we can get a break on the key and break the cipher.

All this will take a matter of an hour to break, if proceeded in a right way. Otherwise, an averagely scripted code will take about a few hours to break the code.

*A Known Plaintext attack*

* A plaintext attack is based on the fact that there is a known plaintext in the cipher text that is given.
* The known plaintext is known as ‘crib’.
* After knowing the plaintext-ciphertext pairs, the attacker can narrow down the brute force attack possible.
* The attacker can therefore, with the help of probability speculate the combinations possible of the plaintext and cipher text.
* For example,
* For the known plaintext letter a, which is the starting letter of the known plaintext,
* The attacker can speculate the probability and brute force the combination from the cipher text characters ranging from 1-5.
* He can then map and re-map using a similar way with the following plaintext and cipher text.
* Eventually, the attacker will, hence, figure out the entire plaintext or at least, to start with, the key.
* This form of plaintext attack, however, will consume a lot of time and if conducted by pen and paper, will take decades. With the help of a script, however, it should take about 41/5= 8! units. This multiplied by the number of cipher text characters is the work factor. This goes to say further that this kind of an attack will take time, but it is certainly feasible.
* For a longer cipher text, and a longer known plaintext, obtaining the key in this manner should not be difficult.

## References

1. <https://en.wikipedia.org/wiki/>
2. https://www.mysterytwisterc3.org/en/