Problem #2 (Multi Time Pad is not Secure)

Introduction

OTP

The one time pad(OTP) is the encryption technique that can't be cracked, until the key is not known, or is not True random key. The size of the plaintext and key is same.

```
    message⊕key=cipher
```

MTP

The MTP is same as that of OTP, but the same key is used for encrypting several messages. This makes the encryption weaker.

```
    message_1⊕key=cipher_1
    message_2⊕key=cipher_2
    ..
    ..
    message_n⊕key=cipher_n
```

encrypt_mtp.py

This script will take the plain text and encrypts it, by automatically generating the key of size of plain text.

There are 2 ways by which you can provide the plain text, either by command line or by file(separated by newline if more plaintexts.)

The command line input is as follows:

```
$ python encrypt_mtp.py
```

```
Where is your plain Text available?
1. In file(plaintext seperated by newlines)
2. I will provide here only.
Enter choice:
```

If you choose choice 1:

```
Enter choice: 1
Enter file name: plaintext.txt
Enter file name of cipher text, where you want to store: ciphertext.txt
Your cipher texts is written into file ciphertext.txt

You can get the key from key.txt
```

And if you choose choice 2:

```
Enter choice: 2
Input plaintext here: This is problem 2
Your cipher text is:  e90f11b1d2a65aceb8daa13ced6f09b4a3
Your key is: bd6778c2f2cf29eec8a8ce5e810a649491
```

decrypt_mtp.py

This script will take the cipher text, key and then decrypts it.
There are 2 ways by which you can provide the plain text, either by command line or by file(separated by newline if more plaintexts.)

The command line input is as follows:

```
$ python decrypt_mtp.py
```

```
Where is your cipher Text available?
1. In file(ciphertext seperated by newlines)
2. I will provide here only.
Enter choice:
```

If you choose choice 1:

```
Enter choice: 1
Enter file name where cipher stored: ciphertext.txt
Enter file name of key: key.txt
Enter file name where you want to store plaintext: plaintext.txt
Your cipher texts is written into file plaintext.txt
```

If you choose choice 2:

```
Enter choice: 2
Input ciphertext here: 14edd5365175610be217
Input key here: 4784bb553455087f9137
Your plaintext is: Since its
```

key_mtp.py

`generateKey(keysize)`
Generates the key of given length. It uses two algorithm in combine, which are:

- TOTP: Time-Based One-Time Password Algorithm: https://tools.ietf.org/html/rfc6238
- HOTP: An HMAC-Based One-Time Password Algorithm: https://tools.ietf.org/html/rfc4226

The following algorithm is:

1. Create some random message key.
2. Note the current time, encode it into string.
3. create `HMAC` of key as secret and msg as encoded time, and hash it with `SHA-512`.
4. create `BitVector` of the hash created in step 3.

crack_mtp.py

Run the script by,

```
$ python crack_mtp.py
```

Enter the ciphertext file name, the cracked plaintext is written into `recoveredtext.txt`.

```
Enter filname of cipher text to crack: ciphertext.txt
Crack started!!
Please wait for crack to finish, this will take few seconds...

Plain text is written into recoveredtext.txt file.

Accuracy is: xx.xx%
```

In the crack MTP, the following XOR property is being used.

```
Space XOR [.] = [.]
Space XOR Space = 0
**X XOR Space = x; where X = [A-Z], x = [a-z] and vice versa**
```

Assumptions and Limitations

- The characters for which key character is not found, I have putted `*` at that position in the `recoveredtext.txt` file. These `*` means these characters are not revealed in this attack. However many of the words can be completed by obvious guessing.
- The attack is dependent on number of cipher text available, more is the number of ciphers and length of ciphers, more the characters of plain text is revealed. In simple words, more will be the space in the plain text, more characters get reviled.
- I am taking THRESHOLD value as 0.75, which is 75% of the total cipher text. That means if space was found **75%** out of total cipher text, then I have assumed that position as space, and then calculate the key at that position.