1845479-HW02

Carmignani Federico

November 16, 2021

Abstract

The goal of this homework is to break the DES cipher.
The solution has to be sent to cns@diag.uniroma1.it from the institutional email.
The password to unzip the file with the ciphertext published by Classroom is: 'N1ty_^h'.
The deadline is November 17th at 11:59 pm.

1 What is the challenge

The challenge consists in trying to simulate an attack to a given ciphertext using the DES cipher. The objective is to determine the plaintext, a text in English, by bruteforcing the ciphertext trying to find the right password to retrieve it. The used cipher is DES, Data Encryption Standard, a block cipher that encrypts data in blocks of size of 64 bits each, so 64 bits of plain text goes as the input to DES, which produces 64 bits of ciphertext. The same algorithm and key are used for encryption and decryption (symmetric encryption) and the key length is 56 bits. Actually, the initial key consists of 64 bits. However, before the DES process starts, every 8th bit of the key is discarded to produce a 56-bit key, they are considered as parity bits, so many symbols are considered the same if they have the same first 7 bits. The ciphering of the plaintext can be done using modes of operation like CBC. In order to encode a file with DES, the OpenSSL library, a software for applications that secure communications over computer networks, provides a shell command to perform it: *openssl enc -des -pbkdf2 -in infile.txt -out outfile.txt.enc -k password.*
The decryption is similar, adding the '-d' flag option.

2 How to solve the challenge

The DES encryption usually needs a key and an initialization vector (IV) but it is possible to use also a single password through the PBKDF2 algorithm, a password based key derivation function that has been widely adopted in many security applications. In this way, we can perform the brute force attack just operating on the password, added with the '-k' flag option.
The brute force attack consist in trying all the possible combinations as password to be used by the PBKDF2 algorithm.
I did it using Python as programming language and the implementation will be added to the section 'How to use and run the program'.
It is a random brute force attack that creates possible passwords of a length given as input by the user and try to use them to decrypt the text. Since doing all the attempts using every possible symbol took a lot of time, I assumed that the password was composed by letters and numbers, and I used a random permutation generator of this symbols to assemble the passwords. Besides, I started trying to solve the decryption in parallel shells with password of increasing length until a maximum conjectured length of 8 symbols. So the program stopped when the decryption function was returning 0 and the output file was an ASCII text.

2.1 The steps towards the solution

In order to achieve the solution of the text, I started to generate the passwords taking the symbols from a char-set created from the ASCII table. In the program, a random function choses the symbols and creates the password, then the OpenSSL shell command is run and if the decryption is performed

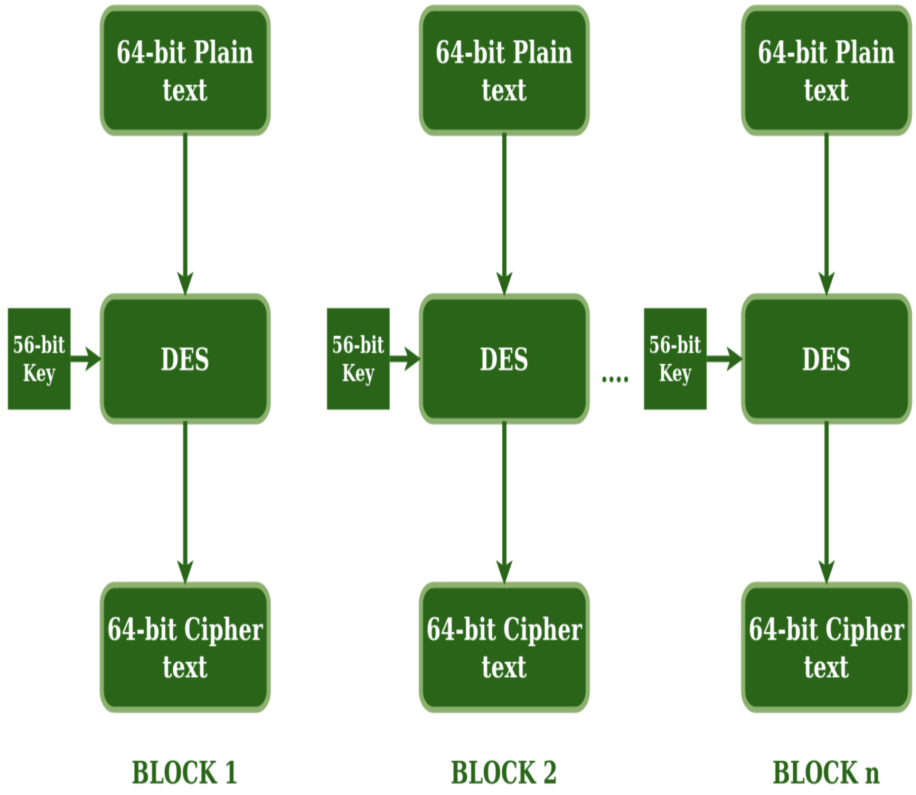


Figure 1: DES idea.

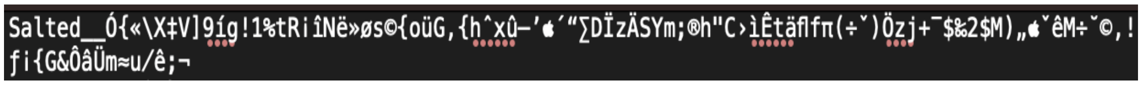
correctly and the file generated is formed by ASCII characters, the program ends printing the password. If had worked on the key (hex), it would have been 2^{56} attempts in the worst case. I could be done working on the bits of the key, but also the IV had to be considered and brute forced. In order to understand if a decryption is done well we can decrypt, then re-encrypt and compare the ciphertexts; this is what I did. In order to do it faster, I exploited the parallelism of shells supported by the OS, running in parallel different programs and passing different lengths of password to be used.

3 How to use and run the program

The program asks for the length of the passwords to generate to the user and then performs what described before automatically. It has to be executed using Spider or another IDE to run the program or also in the terminal using Python interpreter with this command: ‘python 1845479-HW02.py’, the important thing is that the file with the ciphertext is in the same folder of the Python script.

3.1 The code

```
#libraries
from subprocess import PIPE, run
import os
import random

#in order to execute shell commands and save their output
def out(command):
    result = run(command, stdout=PIPE, stderr=PIPE, universal_newlines=True, shell=True)
```



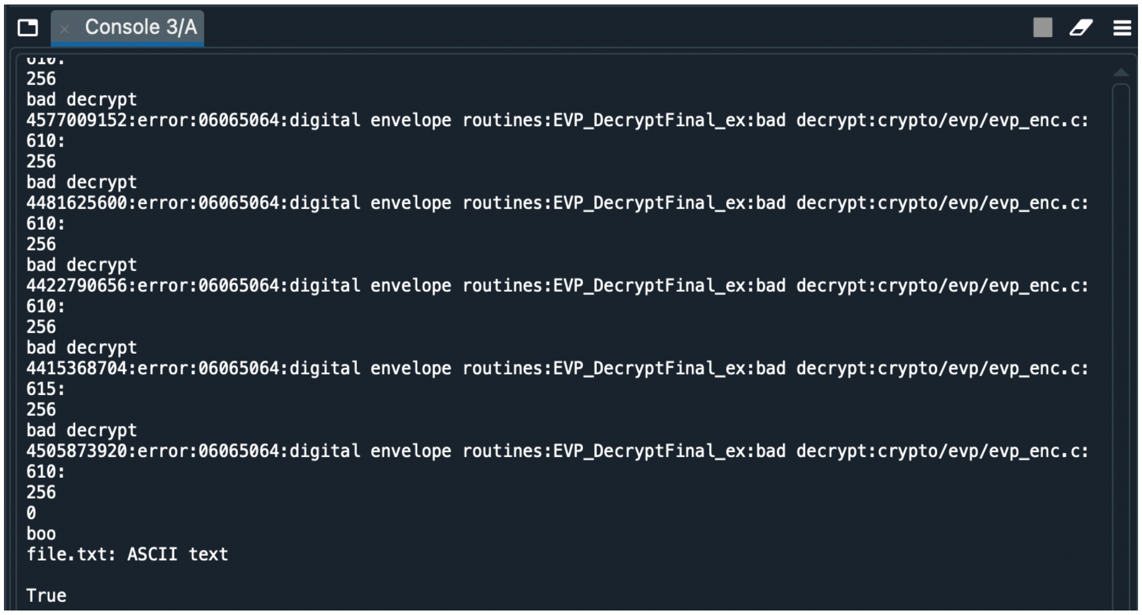
```
Salted__Ó{«\X‡V]9íg!1%tRiîNë»øs©{oüG,{h^xû–'´"∑DÏzÄSYm;®h"C›ìÊtäflfπ(÷ˇ)Özj+¯$‰2$M)„ˇêM÷˘©,!
fi{G&ÔâÜm≈u/ê;¬
```

Figure 2: Demo 1.

```
    return result.stdout

#the command to decrypt and the length as input
i=False
command0="openssl enc -des -d -pbkdf2 -in outfile.txt.enc -out file.txt -k "
stop=0
length=input("Insert the length of the password with whom you want to try brute-force: ")
length=int(length)

#until an ASCII text is found
while(i==False):
    stop=0
    #creating random passwords of the length chosen
    while (stop==0):
        ListOfChars=[]
        codes=list(range(48,57))+list(range(65,90))+list(range(97,122))
        for a in range(length):
            ASCIIcode=random.choice(codes)
            ListOfChars.append(chr(ASCIIcode))
        password=''.join(ListOfChars)
        #run the command appending that password
        command= command0.__add__(password)
        res=os.system(command)
        print(res)
        #if the decrypting process is ok then exit the cycle
        if(res!=256):
            print(password)
            stop=1

    #see if the file is ASCII text and in this case exit the program, otherwise
    #a new cycle is started
    my_output = out("file file.txt")
    print(my_output)
    if("ASCII" in my_output):
        i=True
    else:
        i=False
    print(i)
```

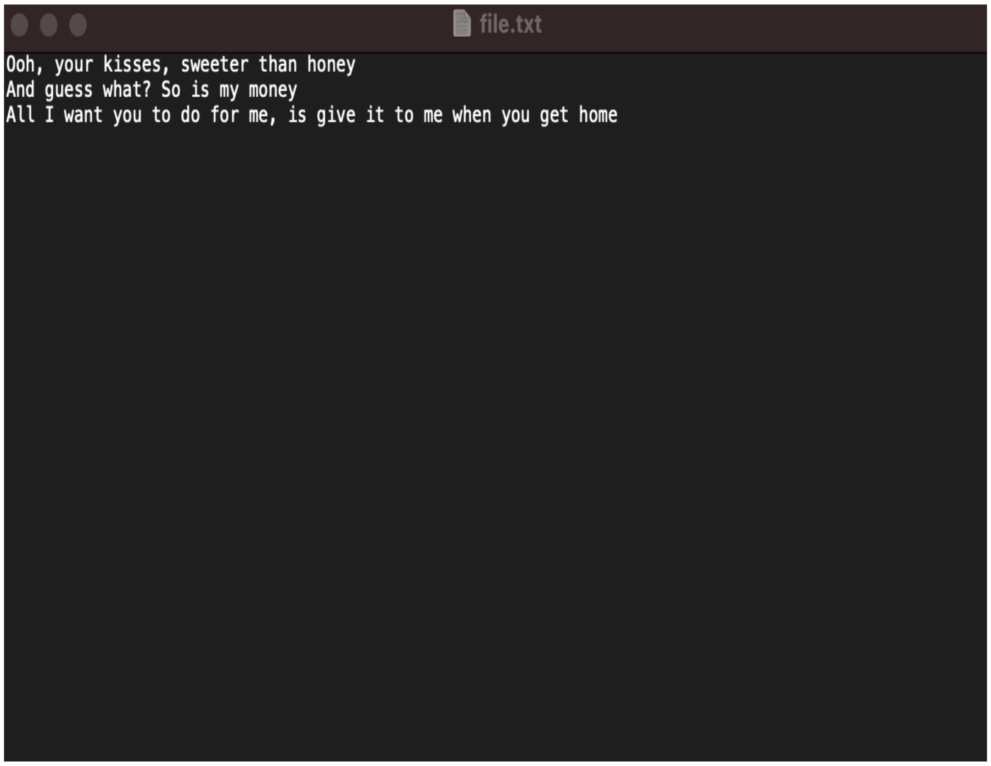
3.2 The program in use

As told before, the program asks to insert the length, I tried with 0 without success, then with 1 without success too and finally with 3 where I found the password to decrypt it: *'boo'*.
An illustration is given in Figure 3, where the Python console shows the moment when the password was found, after many attempts. We can see how the OpenSSL command returns 0 and not 256 (error in decrypting), and the file is ASCII text, in fact it passes the check and prints 'True'.
The ciphertext is shown in Figure 2.
Instead, in Figure 4, it is shown the plaintext produced by the OpenSSL command: *openssl enc -des -d -pbkdf2 -in outfile.txt.enc -out file.txt -k boo.*



```
610:
256
bad decrypt
4577009152:error:06065064:digital envelope routines:EVP_DecryptFinal_ex:bad decrypt:crypto/evp/evp_enc.c:
610:
256
bad decrypt
4481625600:error:06065064:digital envelope routines:EVP_DecryptFinal_ex:bad decrypt:crypto/evp/evp_enc.c:
610:
256
bad decrypt
4422790656:error:06065064:digital envelope routines:EVP_DecryptFinal_ex:bad decrypt:crypto/evp/evp_enc.c:
610:
256
bad decrypt
4415368704:error:06065064:digital envelope routines:EVP_DecryptFinal_ex:bad decrypt:crypto/evp/evp_enc.c:
615:
256
bad decrypt
4505873920:error:06065064:digital envelope routines:EVP_DecryptFinal_ex:bad decrypt:crypto/evp/evp_enc.c:
610:
256
0
boo
file.txt: ASCII text

True
```

Figure 3: Demo 1.



```
Ooh, your kisses, sweeter than honey
And guess what? So is my money
All I want you to do for me, is give it to me when you get home
```

Figure 4: Demo 2.