

# **CSCE 465 : Secret Key Encryption**

Due on 10 October, 2019

*Gu Fall 2019*

**M. Hunter Martin**  
**825002697**

## Written Problems

### 2.2 (3 pts)

Random J. has been told to design a scheme to prevent messages from being modified by an intruder. Random J. decided to append to each message a hash of that message. Does this solve the problem? Why?

If we assume that an attacker can modify the message, then we know that the attacker can also modify the hash. The attacker can simply change the message then recompute their own hash.

### 2.3 (4 pts)

Suppose Alice, Bob, and Carol want to use secret key technology to authenticate each other. If they all used the same secret key  $K$ , then Bob could impersonate Carol to Alicia (actually any of three can impersonate the other to the third). Suppose instead that each had their own secret key, so Alice uses  $K_A$ , Bob uses  $K_B$ , and Carol uses  $K_C$ . This means that each one, to prove his/her identity, responds to a challenge with a function of his/her secret key and the challenge. Is this more secure than having them all use the same secret key  $K$ ? (Hint: what does Alice need to know in order to verify Carol's answer to Alice's challenge?)

It is somewhat more secure, but not in a useful way. Because these are secret keys, it is again impossible to know that the person is who they say who they are, but in theory you would be able to tell messages from Bob and Carol apart. Essentially you know that respondent D and E are different people, but you don't know if D is Bob or if D is Carol.

### 2.4 (4 pts)

It is common, for performance reasons, to sign a message digest of a message rather than the message itself. Why is it so important that it be difficult to find two messages with the same message digest?

If we find two messages that map to the same message digest, then we can't verify the identity of the sender.

### 3.2 (7 pts)

Token cards display a number that changes periodically, perhaps every minute. Each such device has a unique secret key. A human can prove possession of a particular such device by entering the displayed number into a computer system. The computer system knows the secret keys of each authorized device. How would you design such a device?

The device can generate a hash between the current time and the secret key. The user will submit the generated hash "one-time passcode," and the computer system will recompute the hash of the time and the user's secret key. If the two hashes match, then the user will have verified their ownership.

**3.3 (7 pts)**

How many DES keys, on the average, encrypt a particular plaintext block to a particular ciphertext block? Please explain.

Each block of plaintext is 64 bits, and those bits are mapped to a 64 bit ciphertext output using a 56 bit key. This means that 1 in 256 of all possible keys can map a particular plaintext block to a particular ciphertext block.

**3.5 (7 pts)**

Suppose the DES mangler function mapped every 32-bit value to zero, regardless of the value of its input. What function would DES then compute?

All DES would do is an initial permutation, the LR block swap, and the ending permutation. The ciphertext would be a simple interchange of the odd and even bits.

**4.2 (7 pts)**

The pseudo-random stream of blocks generated by 64-bit OFB must eventually repeat (since at most 264 different blocks can be generated). Will KIV necessarily be the first block to be repeated?

Yes. Because we know that decryption in OFB is the inverse of encryption, we know that sequence must be repeating.

**4.4 (*Bonus* 7 pts)**

What is a practical method for finding a triple of keys that maps a given plaintext to a given ciphertext using EDE? Hint: It is like the meet-in-the-middle attack mentioned in the class (detailed in the KPS textbook).

This question refers to the fact that it is relatively simple to brute force the Triple DES (3DES) encryption. We can fix the first key, increment the second key, and increment the third key fully for each of the second key's values. We can expect to break the 3DES algorithm relatively quickly by simply trying all of the keys.

**4.6 (7 pts)**

Consider the following alternative method of encrypting a message. To encrypt a message, use the algorithm for doing a CBC decrypt. To decrypt a message, use the algorithm for doing a CBC encrypt. Would this work? What are the security implications of this, if any, as contrasted with the "normal" CBC?

This would work, but identical plaintext blocks would produce identical ciphertext blocks.

## Task 1: Encryption using different ciphers and modes [6 pts]

I ran the following commands to encrypt the input.txt into 3 different ciphers and 3 different modes. (AES, DES, RC2, GOST89 for the ciphers, and CBC, OFB and ECB for the modes)

```
openssl enc -aes-128-cbc -e -in input.txt -out aes128cbc_output.bin \  
-K 00112233445566778889aabbccddeeff \  
-iv 0102030405060708
```

```
openssl enc -aes-128-ofb -e -in input.txt -out aes128ofb_output.bin \  
-K 00112233445566778889aabbccddeeff \  
-iv 0102030405060708
```

```
openssl enc -aes-128-ecb -e -in input.txt -out aes128ecb_output.bin \  
-K 00112233445566778889aabbccddeeff \  
-iv 0102030405060708
```

```
openssl enc -des-cbc -e -in input.txt -out descbc_output.bin \  
-K 00112233445566778889aabbccddeeff \  
-iv 0102030405060708
```

```
openssl enc -RC2-ECB -e -in input.txt -out rc2ecb_output.bin \  
-K 00112233445566778889aabbccddeeff \  
-iv 0102030405060708
```

```
openssl enc -gost89-ecb -e -in input.txt -out gost89ecb_output.bin \  
-K 00112233445566778889aabbccddeeff \  
-iv 0102030405060708
```

The input and outputs are included in Task 1 subdirectory.

## Task 2: Encryption Mode – ECB vs. CBC [8 pts]

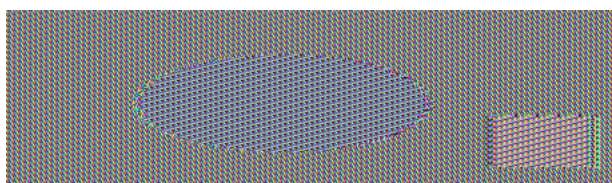
In this task I will encrypt two images using ECB and CBC.

### Trial 1: pic\_original.bmp

This is the original picture.



This is encrypted using ECB. You can notice that the shapes are still mostly visible.



This is encrypted using CBC. In this picture, we can't make out any of the shapes.



### Trial 1: test2.bmp

This is the original picture.

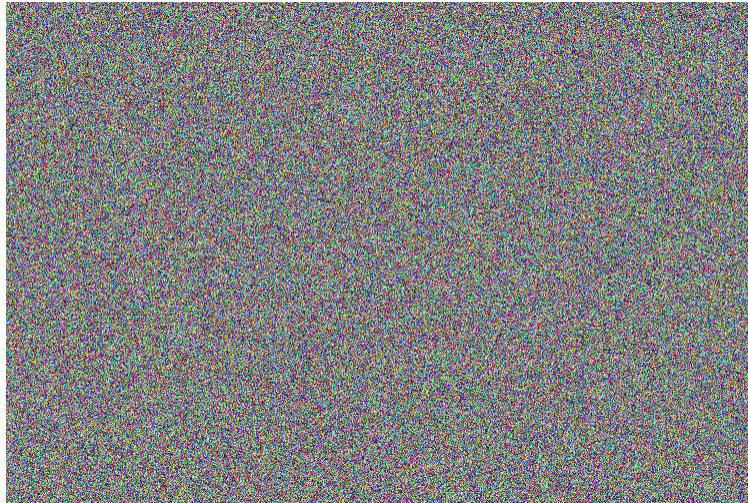


This is encrypted using ECB. You can still see the outline of a flower in the lower left hand corner.



This is the same picture again encrypted using CBC. In this picture, we again are unable to make out any of the shapes.





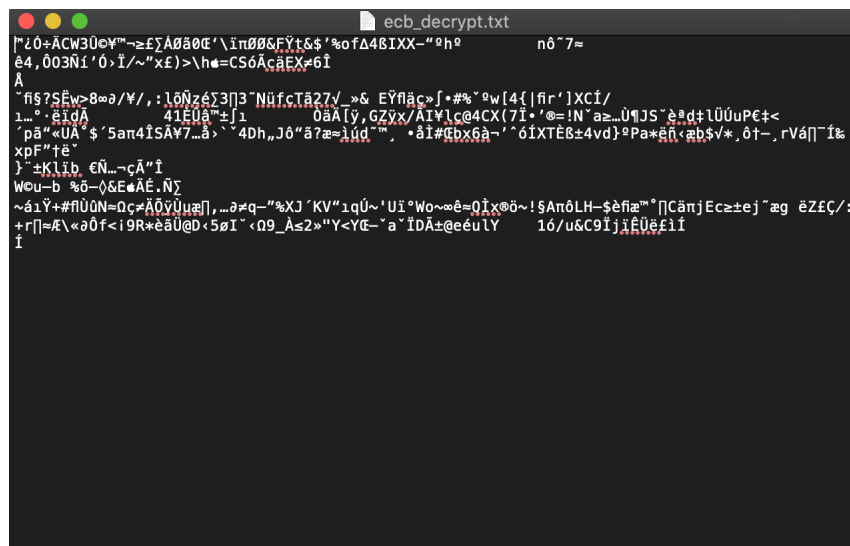
All of the pictures are stored in the Task 2 subdirectory.

### Task 3: Encryption Mode – Corrupted Cipher Text [10 pts]

Input.txt:

The FitnessGram Pacer Test is a multistage aerobic capacity test that progressively gets more difficult as it continues. The 20 meter pacer test will begin in 30 seconds. Line up at the start. The running speed starts slowly but gets faster each minute after you hear this signal bodeboop. A sing lap should be completed every time you hear this sound. ding Remember to run in a straight line and run as long as possible. The second time you fail to complete a lap before the sound, your test is over. The test will begin on the word start. On your mark. Get ready, Start.

In ECB, I had guessed that we'd be able to recover no data, and that was proven correct because of the dependence of the blocks.



In CBC, I had guessed that we'd be able to also recover no data, and I was again correct. This was again because of the block dependence (errors propagate).

```

h_{f ΔsεâL0fēfzðEi|)°xzdò
ENIS¥°ē8"iē'<·0v
ôlo> 0?69AñΩ=zGē@[ÆlñSyn]tSb/i/V}h[]ieē"ÜN=>!cŸ
.-'sMF€W'~0xâ1iΩ-J0iΩc+ēb;ýtæ{ðd$Ÿyē°I;aē°{ i3"~Q'Üc
|C' l0Ωfiâ>)Ytôâg"
iôm"=ÆkΩ~^ÜS' 0I-i0. „ÆIcwyH-ú>àEi"ÄεTñ?jî10[]
v~Vgt&ε-~%ç` °z.ñf°c'â†üœRâ{2nI'  ÆR$ {IñÄ+ÿet+f/ÄQ'n-d0ñ-Éoæ#K$6dÜ«RqWDÜε-S]Æ
~>+AE"EK_ei'~tÜ"~œJ0úv+0ô_jâ¥.=T[]è`Æ0$
Ä%b
>>ÜçRÉ"εä-C•l¥9°ôE
~"εAÜBðIÉ•Ä•ēİgl'fRÉoÚΔó°#≈-fı~*Ÿf_~ðÄ~·Ä7T-/0E~_ynÄΔÜñHÜTrİDNCÉ~N
.0İÄy-èA«
ü!ç
|SKZ|Kœâfı!É#°b@Rç{0≤~â
y°Jc=âz}~"°p;üv¥$C'âÜ°ð-âo>u,ŸK<H,Ä4Üvqðz.TœœÄİdæ"~YJô$Vf"~«S.PÊôic,

```

In CFB, I had guessed that we'd be able to recover no data, but this was only partially true. We could recover everything before the block with the error in it. In this way, errors propagate forward but not backwards.

```

The FitnessGram s,ôK'-újxCâ6ê°$Ä2`~2W"8Ä0Ä'ôJ0$GŸçLİÜ?AèLİ0%.ä[>...º,CÄ-Cg±"/VÜ%±ıú$Ÿİİ/ø%?
Än0Δf;çsÄciÜ"i"~ú4/MÇÜ[]üB.G•':HÄvYΔ-}üv#Æ[]0g»SÇEw...X
Dk'ß2JvbmSS[μ¶,f\<"¶vN.İÉ2Ä7?ðNVV£.PÜ%)İn}'fV...Ü±â;4N>Æē~W-N'frÉ
tÊgX|ÆHÿaÜgçÜÜ'-hLÇÄhêÉÉİF{I8RE+6,=Ne°ôKëv$ð/ç90ô·œFÉİy,Ä@câiçŸ'ñð3
j/
ðfiNÖU'·Ééœ=NfiA@1n6..."Çs~Lég.=ôRİéœ=V.8=rç5$[8[èÜDT"18æ[fi]»wİð0Ézİ
%0Æâ*:â:Ä-Érf'êç>7dyôô;/[]00Äâİ]j;Ä~<l@ñX?ôâ>i /0k*Ü,òQê+)ēø Δd'ÊÊÆ#Äİf&£ ~p£fHNß?
ô""~*%EY+æâİ
EÉp-z#EÉQ¶¶EİX""Fçqçffİ£•Ÿ_Yb.0n1+ÿ> :Æ•0~èÄPÜôÄ°Zf~"EÜVJo~*R¶¶øŸÉ¶&kº±ð

```

In OFB, I had guessed that we'd be able to recover all the data not directly affected, and that was true. Only one character was affected in the string.

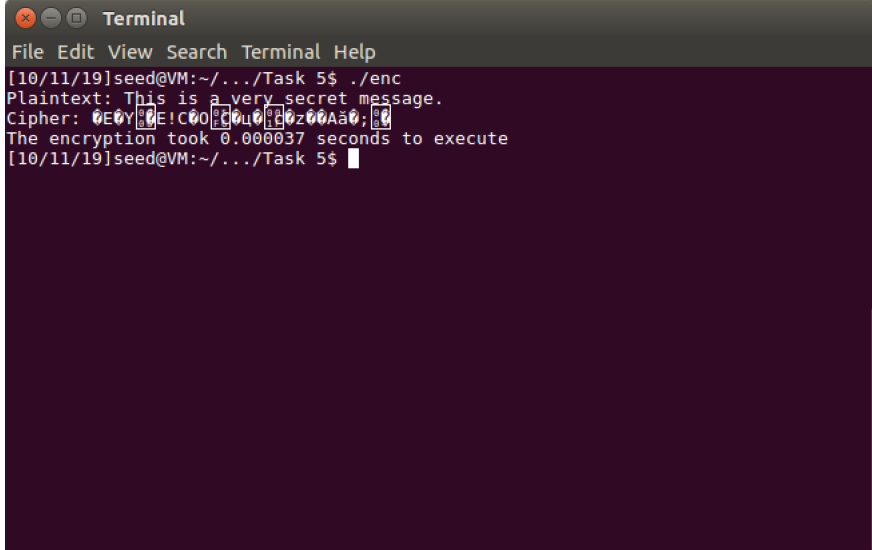
The FitnessGram Pacer Test is ` multistage aerobic capacity test that progressively gets more difficult as it continues. The 20 meter pacer test will begin in 30 seconds. Line up at the start. The running speed starts slowly but gets faster each minute after you hear this signal bodeboop. A sing lap should be completed every time you hear this sound. ding Remember to run in a straight line and run as long as possible. The second time you fail to complete a lap before the sound, your test is over. The test will begin on the word start. On your mark. Get ready, Start.

## Task 4: Programming using the Crypto Library [15 pts]

```
[10/10/19]seed@VM:~/.../Task 4$ make
gcc -I/usr/local/ssl/include/ -L/usr/local/ssl/lib/ -o enc task4.c -lcrypto
[10/10/19]seed@VM:~/.../Task 4$ ./enc
Key Found: median
[10/10/19]seed@VM:~/.../Task 4$
```

Using the EVP api, I was able to determine the encryption key was "median."

## Task 5: Write your own DES encryption code (one round) [15 pts]



```
Terminal
File Edit View Search Terminal Help
[10/11/19]seed@VM:~/.../Task 5$ ./enc
Plaintext: This is a very secret message.
Cipher: 0E6Y!C00E0z00Aa0:
The encryption took 0.000037 seconds to execute
[10/11/19]seed@VM:~/.../Task 5$
```

My DES took on average 0.000033 seconds to complete. I used the EVP API functions like TA Mendoza said was allowed on Oct. 6th in the comments about Task 5.

## References

- [1] Gu. Homework 3: Secret-Key Encryption.