It-sikkerhed: A4

Silja Maria(dfv380) og Christian Jacobsen(wbr220)

October 2020

1 Resubmission

For this resubmission, we have completed task 5 in the seed labs, and overall fixed the rest of the report up. We also answer questions 20.7 and 20.17 correctly this time.

2 Review Questions

2.1 20.4

In this question, we wish to explain what the between a block cipher and a stream cipher is.

Block cipher and stream cipher are both methods for symmetric encryption. They differ primarily in the size of the data, which they apply the encryption to. Using a block cipher method, the data will first be grouped into blocks and then each of these blocks will be encrypted by an encrypting algorithm. Using a stream cipher method on the other hand, the encryption algorithm will be applied to every byte in the data. Thus, cipher streams performs much more encryptions operations, but on a smaller input data size than cipher blocks does.

2.2 20.11

In this question, we wish to explain what the difference between a session key and a master key is.

Both a session key and a master key are types of symmetric encryption keys, but have different purposes and different duration. A session key is used encrypted messages, when two end hosts wants to communicate. The key will then be used encrypt all messages send between the two hosts and when the session is stops, the key will be destroyed. A master key is used between a key distributor and a host and have two purposes. It can either be used to encrypt other keys or it can be used to generate other keys. An example of this is when

we wish to generate a session key for two hosts. In this case, the master key will be used to protect the session key, when it is send to the two hosts.

2.3 21.5

In this question, we wish to briefly explain Diffie-Hellman key exchange.

Diffie-Hellmans key exhange is a way to generate a session key between two users that want use encryption when communicate with each other. Before generating the session key, the two users need to agree on two numbers p and q, where p is a prime number and q is primitive root of p. Then both of the two hosts generate a secret randomly value, a and b. These values is then used to calculate a shared value of each of the hosts. These two shared values are called A and B and are calculated in this way

$$A = q^a \texttt{mod} p$$

$$B = q^b \texttt{mod} p$$

The two hosts now exhange these values and use them to calculate the secret key, which will be used to encrypt the messages they send to each other. The secret is calculated of both of the hosts and result in the same number:

$$K = B^a \texttt{mod} p$$

$$K = A^b \texttt{mod} p$$

Since the hosts are the only ones who knows the secret values, they will also be they only ones to know the value of the secret key.

2.4 22.10

In this question, we wish to explain the transport and tunnel modes of ESP.

When using ESP, the protocol can either operate in transport mode or in tunnel mode. If the protocol operates in transport mode, the protocol will only encrypt the data part in the packet. Using tunnel mode instead, the entire packet is encrypted.

2.5 23.6

In this question, we wish to answer what the role of a CA in X.509 is?

CA is a third party that is used to control the validation of a public key. By this we mean to check if a public key belongs to the user, which it claims to belong to. Users can obtain a certificate from this third for their public key. This certificate can then verify the public key, which other users can use as a indicator to trust this key.

2.6 23.10

In this question, we wish to answer how do most current X.509 implementations check the validity of signatures on a certificate?

Browsers, and the like, include a list of trusted Certificate Authorities, which are responsible for signing these certificates. For example there exists the widely used Lets Encrypt service, designed around making ssl and https connections available for everyone. If a website has their certificate signed by Lets Encrypt, then it can be trusted as much as the CAs signing process can be trusted.
It is also necessary to maintain a Certificate Revocation List(CRL) which contains information about which certificates are revoked, and therefore shouldn't be considered valid any longer. The book however notes, that these lists are often not checked because of the overhead required

3 Problems

3.1 20.7

In this problem, we wish to answer if we can perform encryption operations in parallel on multiple blocks of plaintext in any of the five modes? How about decryption?

ECB - using this approach, the data will be grouped into n blocks and then be encrypted using the same key. The blocks will then be put together into the total cipher text. When decrypting the cipher text, the same key will be used to decrypt all of the n blocks. Since there exists no dependency between each of the blocks when encrypted or decrypted, these operations can be performed in parallel.

CBC, CFB and OFB - using each of these approaches, the encryption of one block will depend on the cipher text of the previous block. Thus, this cannot be performed in parallel. The applies to decryption using OFB, since the decryption of one block append on the previous block. However, using CBC or CFB the decryption can be done in parallel, since the calculation only uses the cipher text and thus, none of the calculation of the previous blocks.

CTR - using this approach, we choose a counter value, which will be used to encrypt the data. The counter value will be increased by one for each of the blocks. The encryption of each block can be performed in parallel as long as the know the number of the block. The same with decryption.

3.2 20.17

In this problem, we wish to answer the following question:

Suppose someone suggests the following way to confirm that the two of you are both in possession of the same secret key. You create a random bit string the length of the key, XOR it with the key, and send the result over the channel. Your partner XORs the incoming block with the key (which should be the same as your key) and sends it back. You check, and if what you receive is your original random string, you have verified that your partner has the same secret key, yet neither of you has ever transmitted the key. Is there a flaw in this scheme?

We will be able to use the method to verify that both parts have the same key, since XOR'ing a bit string with another bit string twice will result in the original bit string. However, the method is not very safe. If an attacker captures first the encrypted string, and then captures the original string, he/she is able to calculate the key by XOR'ing the two.

3.3 21.7

In a public-key system using RSA, you intercept the ciphertext C = 61 sent to a user whose public key is e = 11, n = 91. What is the plaintext M?
We use prime factorization on n: $n = 7 \cdot 13$. We calculate $\phi(n) = (p-1)(q-1) = 6 \cdot 12 = 72$. We know $e = 11$. We determine $de\%72 = 1, \quad d < 72, \quad d = 59$
Knowing the private key, we present the decryption of the given ciphertext:

$$M = C^d\%n = 61^59\%7 = 3$$

3.4 22.2

Consider the following threats to Web security and describe how each is countered by a particular feature of SSL:

3.4.1 Man-in-the-middle attack: An attacker interposes during key exchange, acting as the client to the server and as the server to the client.

This kind of attack is thwarted by the certificate and chain of trust ingrained in the SSL protocols. If an attacker sets up between the server and the client, the client will quickly realise that the certificate used for the domain isn't signed by anyone in its trust store.

3.4.2 Password sniffing: Passwords in HTTP or other application traffic are eavesdropped.

Password-sniffing is prevented by encrypting requests before sending them out. Thus noone without the key-pairs generated in the handshakes, will not be able to read the sniffed packets.

3.4.3 IP spoofing: Uses forged IP addresses to fool a host into accepting bogus data.

While a connection might be spoofed, the SSL protocol demands encrypted communication, and if a session is hijacked, the host will notice, and either abort the session or try to renegotiate a session.

3.4.4 IP hijacking: An active, authenticated connection between two hosts is disrupted and the attacker takes the place of one of the hosts.

The attacker will in this case not know the MAC shared in the handshake, and even if they recieve/send packets to the server/client, the recieving party will instantly know that the session has been broken.

3.4.5 SYN flooding: An attacker sends TCP SYN messages to request a connection but does not respond to the final message to establish the connection fully. The attacked TCP module typically leaves the "half-open connection" around for a few minutes. Repeated SYN messages can clog the TCP module.

SSL has no direct feature to prevent this kind of flooding, especially since it sits on top of the tcp layer.

4 Seed labs

For this lab session, we're working with encryption.

4.1 Task 2

We use the "openssl-enc" tool which encrypts/decrypts text using symmetric key pairs. We try three different ciphers:

```
seed@VM:~$ cat plain.txt
Hello world

seed@VM:~$ openssl enc -aes-256-cbc -in plain.txt -out cipher.bin \
    -K 00112233445566778899aabbccddeeff \
    -iv 0 102030405060708
seed@VM:~$ xxd cipher.bin
00000000: 9b69 d212 5c5f cac4 6942 7214 dab8 5522  .i..\_..iBr...U"
seed@VM:~$ openssl enc -des3 -in plain.txt -out cipher.bin \
    -K 00112233445566778899aabbccddeeff -iv 0102030405060708
seed@VM:~$ xxd cipher.bin
00000000: 61c4 c596 3e2b dce8 c02b b7f5 e26c f470  a...>+...+...l.p
seed@VM:~$ openssl enc -aes-256-cbc -in plain.txt -out cipher.bin \
```

```
    -pass pass:encryption_is_fun
seed@VM:~$ xxd cipher.bin
00000000: 5361 6c74 6564 5f5f 4d88 db81 3e6f d9af  Salted__M...>o..
00000010: 001d eb05 b943 4767 8892 7abc 96e2 7979  .....CGg..z...yy
```

First we encode using the aes-256-cbc cipher, then the des3 cipher, and finally i show using a passphrase from which the keys and ivs can be generated, allowing for easier decryption at a later point.

4.2 Task 3

Task 3 of this lab, has us encrypting a simple photo using the ECB (Electronic Code Book) and CBC (Cipher Block Chaining) modes. As seen by the picture in figure 1, the chain-block mode is much better at obscuring the image, while the ecb mode still leaves a noticable outline of the shapes. This is what we expected with the chain-block depending on each previous block for randomness as opposed to the ecb only having the current block to use.

It should be noted that this wouldn't be the same for any format. In a bitmap the pixel values are encoded one after the other, and the colors in the given picture are very uniform. When the ECB looks through the image, each equivalent block of the image will be encrypted the same, and the bitmap will thus look like the original. If one took a photo and encrypted it as a jpg, the image would be hidden much better, since there aren't as many repeating blocks of pixels. To test this, we demonstrate with a picture with both large sets of randomness, and more uniform data - A subset of the (colored)mandelbrot set.[1] The reason this has been chosen is because it represents both large "random" sets of data, like a photograph would, and large uniform sets of data, like our previous example. The original image is seen on figure 2, and a side-by-side can be seen on figure 3.
As should be obvious the large contiguous areas of color exhibit the previously defined behaviour, *but* the more complex structures, arms and sattelites, show the randomness that is to be expected of e.g. a photo. The cbc of course is still completely unable to be read, and should be considered boring.

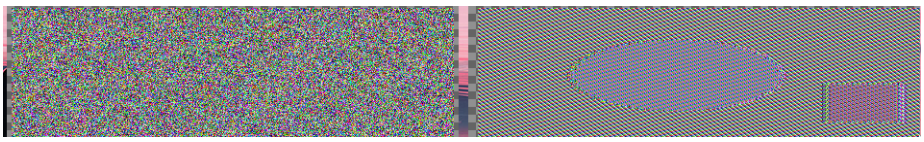

Figure 1: The side-by-side comparison. Left is the chain-block mode, and right is the electronic code block

This concludes task 3

[1] Courtesy of `https://en.wikipedia.org/wiki/Mandelbrot_set`

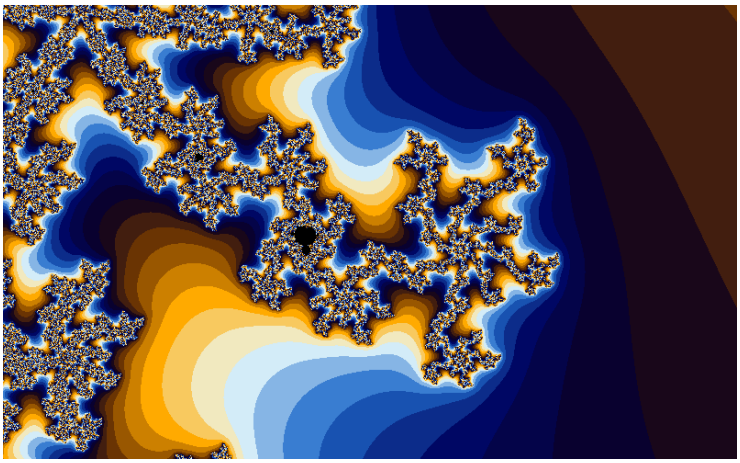

Figure 2: An image showing both large random blobs and large uniform color

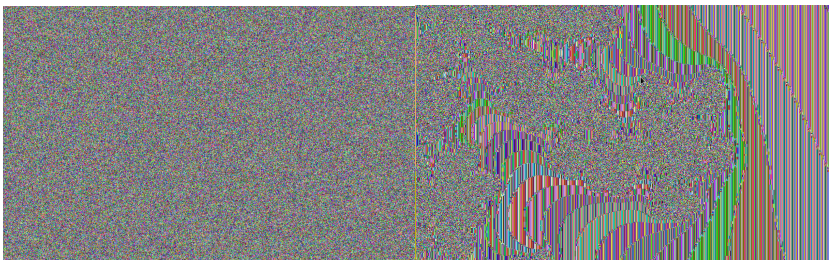

Figure 3: Another side-by-side, this time of mandelbrot, with CBC on the left and ECB on the right.

4.3 Task 5

For this assignment we look at the error propagation of the various AES encryption modes: ECB, CBC, CFB, and OFB. We do this by creating a text file that is at least 1000 bytes long, encrypting it and corrupting a single bit. We then decrypt the ciphertext and explain the results. The task also asks us to think about what will happen before we produce the results.

We encrypt the following text:

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer lectus quam, tempor quis maximus quis, laoreet a enim. In at ligula est. Proin rhoncus aliquet varius. Nunc vulputate elit sed velit tincidunt pharetra. Nam nunc mi, vehicula ut neque nec, tincidunt maximus magna. Sed consequat iaculis risus, laoreet molestie metus ornare in. Vestibulum commodo luctus felis eget scelerisque. Etiam quis tempus massa, molestie vestibulum felis. Cras quis sollicitudin metus. Vivamus sollicitudin eros nec enim tincidunt faucibus. Maecenas laoreet ultricies convallis. Aliquam suscipit egestas neque ut efficitur. Pellentesque at neque fringilla, tincidunt urna dictum, tempus lectus. Cras a nisi dui.

Aenean suscipit, massa id maximus posuere, ex risus commodo quam, sed dignissim lorem dolor at lorem. Fusce bibendum, libero nec dictum semper, risus ante maximus nisl, in tincidunt lorem metus vel mi. Proin pretium orci sit amet ornare congue. Morbi nec libero

vitae diam tristique dictum dapibus nam.
Which rounds out to 1004 bytes.

With the commands shown below:

```
$ ciphs=(ecb cbc cfb ofb)
$ for ciph in $ciphs; do
openssl enc -aes-128-$ciph -in document.txt -out "$ciph.bin" \
-k 00112233445566778899aabbccddeeff -iv 0102030405060708;
done
```

To save space in this report, the files, and code to generate these will be included in our code.zip hand-in.

4.3.1 ECB

As mentioned in the previous section, in ECB every block will be encrypted and decrypted without regard to the previous and next blocks. Thus the final decryption with a corrupted block will be the correct text, with only one small part of the text corrupted.

4.3.2 CBC

In CBC, the decryption of a block, relies on the ciphertext of the previous block. In the case of a corrupted block, the block *and* its following block, is going to be corrupted, while the rest of the blocks are decrypted as normal.

4.3.3 CFB

CFB Suffers from the same issues, with the broken ciphertext corrupting both the current and the next block, while letting the rest of the text remain functional.

4.3.4 OFB

OFB is very comparable to cfb, with the exception that OFB carries the encryption result through, instead of the ciphertext. Thus a broken ciphertext only corrupts the one block, and leaves the rest of the decoded string looking good.