Tutorial #2
Security in Computing COSC2356/2357

Q1. a) Use exclusive or (XOR) to "add" the bit strings 11001010 $\oplus$ 10011010.

b) Convert the decimal numbers 8734 and 5177 into binary numbers, combine them using XOR, and convert the result back into a decimal number.

Ans:

a) 11001010 $\oplus$ 10011010 = 01010000

b) 8734 = '10001000011110',

5177 = '01010000111001',

8734 $\oplus$ 5177 = 10001000011110 $\oplus$ 01010000111001 = 11011000100111,

'11011000100111' = 13863

Q2. A stream cipher can be viewed as a generalization of a one-time pad. Recall that the one-time pad is provably secure. Why can't we prove that a stream cipher is secure using the same argument that was used for the one-time pad?

Ans:

The keystreams are not chosen uniformly at random, since a relatively small number of keys generate a much larger number of possible keystreams.

One of the requirement of the perfect secrecy is the key space has to have the same size as message space. Hence the key and the message must have the same length. In the One Time Pad Cipher, this is the case. In a Stream cipher, given that the key is used to generate a stream, whose length is greater, it does not meet the Perfect Secrecy requirement (while still being reasonably secure).

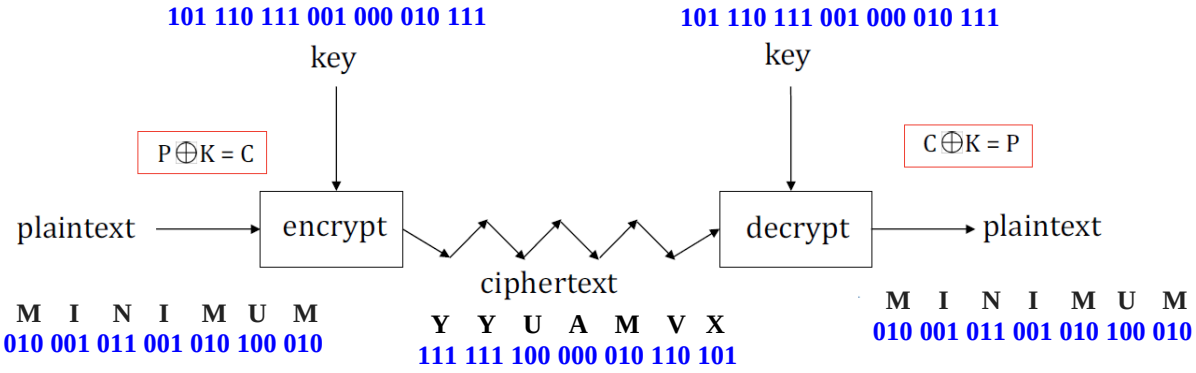
Q3. Using **One-Time Pad** encryption algorithm, encrypt the plaintext "**MINIMUM**". The **key** is: **101 110 111 001 000 010 111**

[*Hints:* Convert each alphabet of the plaintext in to binary using the following dictionary (Table-Q3).

Table-Q3: Dictionary

A	I	M	N	U	X	V	Y
000	001	010	011	100	101	110	111

Ans:



Encryption:

Step-1: Convert the plaintext "**MINIMUM**" in to binary string using the dictionary given in Table-Q3 as follows:

M	**I**	**N**	**I**	**M**	**U**	**M**
010	001	011	001	010	100	010

Therefore, the binary string of the plaintext is: **010 001 011 001 010 100 010**

Step-2: Perform an XOR operation on binary plaintext **010 001 011 001 010 100 010** and key **101 110 111 001 000 010 111** to find out the binary ciphertext as follows:

Plaintext	**010 001 011 001 010 100 010**
⊕ **Key**	**101 110 111 001 000 010 111**
Ciphertext	**111 111 100 000 010 110 101**

Step-3: Convert the binary ciphertext in to English ciphertext using the dictionary given in Table-Q3 as follows:

111	**111**	**100**	**000**	**010**	**110**	**101**
Y	Y	U	A	M	V	X

Therefore, the English ciphertext is: **YYUAMVX**

Decryption:

Step-1: Convert the English ciphertext in to binary ciphertext using the dictionary given in Table-Q3 as follows:

Y	Y	U	A	M	V	X
111	**111**	**100**	**000**	**010**	**110**	**101**

Therefore, the binary ciphertext is: **YYUAMVX**

Step-2: Perform an XOR operation on binary ciphertext **010 001 011 001 010 100 010** and key **101 110 111 001 000 010 111** as follows:

Ciphertext **111 111 100 000 010 110 101**

$\oplus$ *Key* **101 110 111 001 000 010 111**

Plaintext **010 001 011 001 010 100 010**

Step-3: Convert the binary plaintext in to English plaintext using the dictionary given in Table-Q3 as follows:

010	**001**	**011**	**001**	**010**	**100**	**010**
M	***I***	***N***	***I***	***M***	***U***	***M***

Therefore, the English plaintext is: **MINIMUM**

Q4. Recall the online bid method discussed in the lecture

a) What property or properties of a secure hash function *h* does this scheme rely on to prevent cheating?

b) Suppose that Charlie is certain that Alice and Bob will both submit bids between $10,000 and $20,000. Describe a forward search attack that Charlie can use to determine Alice's bid and Bob's bid from their respective hash values.

c) Is the attack in part (**b)** a practical security concern?

d) How can the bidding procedure be modified to prevent a forward search such as that in part (**b**)?

Ans:

a) **One way** (supposed to prevent anyone from determining a bid from the corresponding hash) and **collision resistance** (prevents anyone from changing their bid after submitting the hash).

b) In forward search attack, Charlie finds hashes of all reasonable bids and look for ones that give the same hash as Alice's and/or Bob's bid.

Charlie calculates hashes in advance *h(10000), h(11000), h(12000), ……., h(20000)* using SHA-256 hash algorithm

h(10000) = 39e5b4830d4d9c14db7368a95b65d5463ea3d09520373723430c03a5a453b5df

h(11000) = 8597f1e3043654da36e26467d59c6c6bbc9c44d498cc15ab2f91fc84440bbc35

h(12000) = c5d1866aabc15dda07995e73b08c4ccb514947dcd3a621cea851af5fe366f11b

h(13000) = d45f504deb6b2fe7df5b9efe1d652e08d0614df550d5e748cfb93c6877b12926

…….
…….
h(20000) = 876c9b16254e157d1eb645390dcfae6f29b9d3cd394e73a91de8ee5d0e67ee43

Say, **Alice** submits *h(11000),* and then **Bob** submits *h(12000)* using SHA-256

h(11000) = 8597f1e3043654da36e26467d59c6c6bbc9c44d498cc15ab2f91fc84440bbc35

h(12000) = c5d1866aabc15dda07995e73b08c4ccb514947dcd3a621cea851af5fe366f11b

Charlie observes from above list that $11000 and $12000 have been submitted. So, he simply submits h(13000) to be the winner
h(13000) = d45f504deb6b2fe7df5b9efe1d652e08d0614df550d5e748cfb93c6877b12926

c) Yes, most definitely.

d) Alice should select a random value $\boldsymbol{R_A}$ and submit $\boldsymbol{h(A, R_A)}$, and similarity for the other bidders. When submitting her bid, Alice must submit her bid $\boldsymbol{A}$ and the random padding $\boldsymbol{R_A}$.

For example, Alice chooses a Nonce (random number) $R_A=50000$ and uses it as one time pad. Alice adds the random number with original bid *11000* and performs hash as follows

h(11000+50000)=h(61000)=
FBF23EFCB4C015694C256068C3D196E9323F40A9993B3E9E4F29B118AD570FEB

Similarly Bob does the same (select another nonce=80000) and performs hash

h(12000+80000)=h(92000)=
AD16C1A6866C5887C5B59C1803CB1FC09769F1B403B6F1D9D0F10AD6AB4D5D50

Then they publish the hashes.

After publishing they reveal the bids 11000 and 12000 with the random numbers. This way Charlie cannot guess what the random numbers is, therefore cannot guess the range to produce the list of hash values. Because the random number can be anything and adding it with the original bid produces unpredictable value.

Task 1 (Symmetric Key Encryption and Decryption using OpenSSL).

It is assumed that you have OpenSSL installed in your computer. If you are using Microsoft Windows operating system, then download and install OpenSSl from the following link:

http://downloads.sourceforge.net/gnuwin32/openssl-0.9.8h-1-bin.zip

Unzip the file.

Note: If you are using linux or recent MacOS, then you already have OpenSSL.

Locate the "**openssl.exe**". Assume that the "**openssl.exe**" file is in **D:\OpenSSL\bin**. Open terminal (command prompt in Windows operating system) and run the following command from the directory mentioned above:

```
>openssl
```

You are ready to run OpenSSL command.

Task 1.1 (AES Algorithm) Assume that you have a plain-text file, called "**textFile.txt**", with your *name* and *student ID* in that file. The may look like the followings:

Student ID: S1234567
Name: ABCDEF

Apply Openssl's *AES algorithm* using the followings:

ECB mode and a 256-bit key to encrypt and decrypt. Choose a reasonable shared secret key (e.g 1234). Also try for different key size (e.g. 128-bit).

Solution:

Step-1 (Encryption): Encrypt a text file called "**textFile.txt"** and generates a binary ciphertext file called "**secret.txt**" using the following command.

```
aes-256-ecb -in textFile.txt -out secret.txt
```

Now a password will be asked. Enter **1234** as password.

Step-2 (Decryption): Decrypt the "**secret.txt**" file using the following command to obtain the plaintext file:

```
aes-256-ecb -d -in secret.txt -out decrypt.txt
```

Enter **1234** as password.

Check the file **decrypt.txt** where you should find the same content as in **textFile.txt.**

Task 1.2 (DES Algorithm) Assume that you have a plain-text file, called "**`textFile.txt`**", with your *name* and *student ID* in that file. Apply Openssl's *DES algorithm* using ECB mode to encrypt and decrypt. Choose a reasonable shared secret key.

Solution:

Step-1 (Encryption): Encrypt a text file called "**`textFile.txt`"** and generates a binary ciphertext file called "**`secret.txt`**" using the following command in the OpenSSL terminal.

```
des-ecb -in textFile.txt -out secret.txt
```

Now a password will be asked. Enter **1234** as password.

Step-2 (Decryption): Decrypt the "**`secret.txt`**" file using the following command to obtain the plaintext file:

```
des-ecb -d -in secret.txt -out decrypt.txt
```

Enter **1234** as password.

Check the file **`decrypt.txt`** where you should find the same content as in **`textFile.txt`**.