**Introduction to AES and Symmetric Encryption**

**Introduction:**

The learning objective of this lab is for students to get familiar with the concepts in the secret-key encryption. After finishing the lab, students should have a first-hand experience with encryption algorithms, the AES algorithm, encryption modes. Moreover, students will be able to use tools and write programs to encrypt/decrypt messages. This lab covers the following topics:

* Secret-key encryption
* AES algorithm
* Encryption modes
* Hash and salt

*References:*

*Portions of this lab material come from the following sources:*

* *Handbook of Applied Cryptography by Menezes, van Oorschot and Vanstone  (http://cacr.uwaterloo.ca/hac/)*
* *Introduction to Cryptography by Trappe and Washington*
* *Introduction to Modern Cryptography by Katz and Lindell*
* *SEED security labs from Wenliang Du*

**Overview:**

Symmetric-key block ciphers are the most prominent and important elements in many cryptographic systems. Individually, they provide confidentiality. As a fundamental building block, their versatility allows construction of pseudorandom number generators, stream ciphers, MACs, and hash functions. They may furthermore serve as a central component in message authentication techniques, data integrity mechanisms, entity authentication protocols, and (symmetric-key) digital signature schemes. AES is a symmetric block cipher that is heavily used in wireless networking. We will explore this algorithm more in this lab.

No block cipher is ideally suited for all applications, even one offering a high level of security. This is a result of inevitable tradeoffs required in practical applications, including those arising from, for example, speed requirements and memory limitations (e.g., code size, data size, cache memory), constraints imposed by implementation platforms (e.g., hardware, software, chipcards), and differing tolerances of applications to properties of various modes of operation. In addition, efficiency must typically be traded off against security.

Block Ciphers:

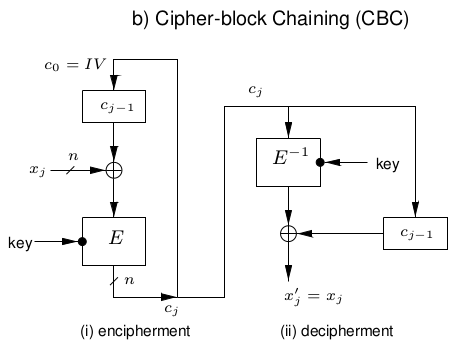
A block cipher is a function (see §1.3.1) which maps n-bit plaintext blocks to n-bit ciphertext blocks; n is called the blocklength. It may be viewed as a simple substitution cipher with large character size. The function is parameterized by a k-bit key K, taking values from a subset K (the key space) of the set of all k-bit vectors. It is generally assumed that the key is chosen at random. Use of plaintext and ciphertext blocks of equal size avoids  
data expansion. To allow unique decryption, the encryption function must be one-to-one (i.e., invertible). For n-bit plaintext and ciphertext blocks and a fixed key, the encryption function is a bijection, defining a permutation on n-bit vectors. Each key potentially defines a different bijection.

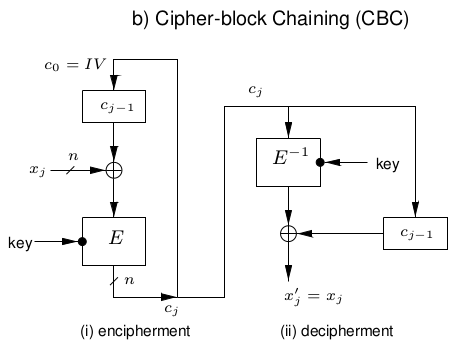
Modes of operation:

A block cipher encrypts plaintext in fixed-size n-bit blocks (often n = 64). For messages exceeding n bits, the simplest approach is to partition the message into n-bit blocks and  
encrypt each separately. This electronic-codebook (ECB) mode has disadvantages in most applications, motivating other methods of employing block ciphers (modes of operation)  
on larger messages. The four most common modes are ECB, CBC, CFB, and OFB. Let's explore two of these.

ECB:

The electronic codebook (ECB) mode of operation is given in the Algorithm and illustrated in the Figure (a):





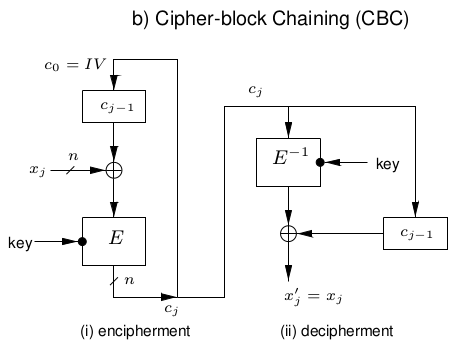
Properties of the ECB mode of operation:

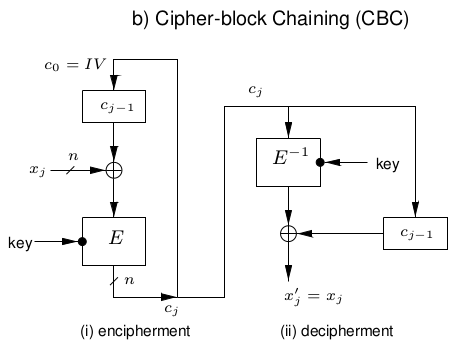
* Identical plaintext blocks (under the same key) result in identical ciphertext.
* Chaining dependencies: blocks are enciphered independently of other blocks. Reordering ciphertext blocks results in correspondingly re-ordered plaintext blocks.
* Error propagation: one or more bit errors in a single ciphertext block affect decipherment of that block only. For typical ciphers E, decryption of such a block is then random (with about 50% of the recovered plaintext bits in error).

We will look at how the properties of ECB in an algorithm such as AES can effect security later in this lab.

CBC:

The cipher-block chaining (CBC) mode of operation, specified in the following Algorithm and illustrated in Figure (b), involves use of an n-bit initialization vector, denoted IV.





Properties of the CBC mode of operation:

* Identical plaintexts: identical ciphertext blocks result when the same plaintext is enciphered under the same key and IV . **Changing the IV , key, or first plaintext block (e.g., using a counter or random field) results in different ciphertext.**
* Chaining dependencies: the chaining mechanism causes ciphertext c\_j to depend on x\_j and all preceding plaintext blocks (the entire dependency on preceding blocks is, however, contained in the value of the previous ciphertext block). Consequently, rearranging the order of ciphertext blocks affects decryption. Proper decryption of a correct ciphertext block requires a correct preceding ciphertext block.
* Error propagation: a single bit error in ciphertext block c\_j affects decipherment of  
  blocks c\_j and c\_j+1 (since x\_j depends on c\_j and c\_j−1 ).

Advanced Encryption Standard (AES)

The AES algorithm is a block cipher algorithm that can work with multiple key lengths, such as 128, 192, and 256, and multiple modes, such as ECB and CBC. The AES algorithm was selected to replace the DES algorithm when weaknesses were exposed in DES. We won't give an in depth review of the AES algorithm here. For those that are interested there are lots of great references with explanations ( for example: *Introduction to Cryptography by Trappe and Washington*). Given a key, let's assume 128 bits, and a message, the algorithm consists of ten rounds. Each round uses a key that is either equal to or derived from the initial key. Each round works on blocks of size equal to the key, 128 in this case. Padding can be used to ensure an even block size. There are four steps that are used to form the rounds:

* ByteSub Transformation (BS): this is used to form resistance to differential cryptoanalysis.
* ShiftRow Transformation (SR): a linear mixing step that causes diffusion of the bits over multiple rounds.
* MixColumn Transformation (MC): similar to shift row.
* AddRoundKey (ARK): the round key is XORed with the result of the previous step.

Combined we have a simplified version of the AES algorithm:

1. ARK, using the initial key
2. Nine rounds of BS, SR, MC, ARK using round keys 1 through 9.
3. A final round: BS, SR, ARK, using the 10th round key.

**Lab Tasks Overview:**

Now that we have an overview of the AES algorithm, block ciphers and modes we can start our lab tasks. There are two goals with these tasks:

1. The first explores how the AES algorithm may not work for every task depending on the key size and mode selected. This is explored more in part1 below.
2. The second is to explore that often the best way to defeat encryption is not to break the actual algorithm, like AES, but break a poorly chosen password. We explore this more in Part2.

**Part 1 Lab Tasks**: AES and Encryption Mode – ECB vs. CBC

In this task, we will play with various encryption algorithms and modes. You can use the following  
openssl enc command to encrypt/decrypt a file. To see the manuals, you can type *man openssl*  
and *man enc*.

***$ openssl enc -ciphertype -e -in plain.txt -out cipher.bin \***  
***-K 00112233445566778889aabbccddeeff \***  
***-iv 0102030405060708***

Please replace the ciphertype with a specific cipher type, such as -aes-128-cbc, -bf-cbc,  
-aes-128-cfb, etc. You can find the meaning  
of the command-line options and all the supported cipher types by typing "man enc". We include some  
common options for the openssl enc command in the following:

***-in <file> input file***  
***-out <file> output file***  
***-e encrypt***  
***-d decrypt***  
***-K/-iv key/iv in hex is the next argument***  
***-[pP] print the iv/key (then exit if -P)***

The file [pic\_original.bmp](https://canvas.umw.edu/courses/1352941/files/95456220/download?wrap=1)[Download pic\_original.bmp](https://canvas.umw.edu/courses/1352941/files/95456220/download?download_frd=1)can be downloaded on canvas, and it contains a simple picture.  
We would like to encrypt this picture, so people without the encryption keys cannot know what is in the  
picture. Please encrypt the file using the ECB (Electronic Code Book) and CBC (Cipher Block Chaining)  
modes, and then do the following:

1. Let us treat the encrypted picture as a picture, and use a picture viewing software to display it. How-  
   ever, For the .bmp file, the first 54 bytes contain the header information about the picture, we have  
   to set it correctly, so the encrypted file can be treated as a legitimate .bmp file. We will replace the  
   header of the encrypted picture with that of the original picture. We can use the **bless hex editor**  
   tool to directly modify binary files. We can also use the following  
   commands to get the header from p1.bmp, the data from p2.bmp (from offset 55 to the end of the  
   file), and then combine the header and data together into a new file.
2. ***$ head -c 54 p1.bmp > header***  
   ***$ tail -c +55 p2.bmp > body***  
   ***$ cat header body > new.bmp***
3. Display the encrypted picture using a picture viewing program.
4. Can you derive any useful information about the original picture from the encrypted picture? Please explain your observations. You should argue for one mode of encryption over the other based on how well the encryption mode did in concealing the information contained in the picture.
5. What causes the differences in the enciphered file between ECB and CBC?

**Part 2 Lab Tasks:**

In this homework you will write your own dictionary based password cracker. The program should be written in C++.

**Background**

Passwords are not stored in plain text; rather a hashed form of the password is kept on a system. When a user logs in, the password they enter is hashed and compared to the stored version. If the hashes match, the user is authenticated. However, if an attacker can obtain a copy of the hashed passwords they can try to recover the plain text passwords via a dictionary or rainbow table type method. The dictionary method uses a dictionary of common words/passwords and, using the same hash algorithm used on the passwords, computes the hashes of the known dictionary words and compares them against the password hashes. If they find a match they will know the plain text password. A rainbow table attack is very similar except that the hashes of the dictionary are all pre-computed and stored. This “rainbow” table can then be used multiple times, reducing the computational work for the attacker.

To improve security modern systems also “salt” the passwords. Salt is a relatively small random string, which is added to each password before hashing. The salt is unique for each password. The result is that even if two users happen to have the same password, their salts will differ and thus the resulting hashes will differ. This also makes using rainbow table type attacks more difficult. Since the salt is not part of the password it is stored in plaintext in the password file. Thus salting doesn't necessarily increase the security of any one password, if an attacker has the password file, but it does make brute force attacks on the entire password file more difficult.

**Tasks:**

I have included a simple Unix password hash generator (found here [our\_crypt.cpp](https://canvas.umw.edu/courses/1352941/files/95456221/download?wrap=1" \o "our_crypt.cpp)[Download our\_crypt.cpp](https://canvas.umw.edu/courses/1352941/files/95456221/download?download_frd=1)). The program takes a password, a 2 character salt, and generates the hash using the crypt() system call. Use the command "man crypt" for more information on this system call. Notice that the resulting hash has the salt as its prefix. This is important as the salt is needed to compare the hash and the user entered password.

1. Download the code, compile it, and run the program on a few passwords and salts. Make sure you understand what the program is doing and how the crypt function is used. E-mail me if you have any questions. Note that you need to include the linking command -lcrypt when you compile and link.
2. Write a dictionary-based program to break hashes produced by the program in part1 above. That is, your program should do the following:
   1. Take a hash produced by the program in step 1.
   2. Break the hash into the salt and true hash.
   3. Open a dictionary file, run the words in the dictionary file through the same hash function using the salt, comparing each one with the hash we want to break.
   4. If it finds a match produce the plaintext password.
3. Make sure your program is written in C++ and runs on the Linux lab machines.

**Example:**

Here is an example output of hash-cracking program I wrote running on the hash 1vBDNxjQ72c1g

*Enter the hash to break:*  
*1vBDNxjQ72c1g*

*Enter the dictionary file name:*  
*words.txt*  
*Got the salt: 1v*  
*Found the password: pass*

**What to turn in:**

You need to submit your solutions to both part 1 and part 2 via canvas. You will need to submit a **zip file** containing the following:

1. A lab report for part 1. The report should be a**pdf** document that includes your answers to the questions from part one along with evidence for each task. The evidence should be screen shots included in the pdf.
2. Code for part 2. You should include a your hash breaking C++ code along with a makefile to compile your program.