Introduction to Information Security
Project #3 Cryptography

Goals

- Understand how the DES block cipher works and implement CBC mode encryption and decryption using DES block cipher
- Collectively crack a DES key
- Estimate how much time it takes for a single user to brute-force keys

Grading

Project submissions are going to be graded using scripts, so please make sure you follow the specified format of submission.

Task 1: CBC mode encryption using DES block cipher (40 points)

Implement CBC mode based on the provided DES block cipher. The DES block cipher is provided in three languages, i.e. C++/Java/Python (**des.cpp/java/py**), and you can choose any of them. We provide a template, **task1.cpp/java/py**. The template contains an example usage of the DES block cipher in function ***test_des***, and two TODO functions (***cbc_encrypt/cbc_decrypt***). Your job is to implement the two TODO functions.

Your program takes five arguments in order, which are described as follows:

1. *mode*: string, "enc" or "dec", performs either encryption or decryption according to the argument
2. *input file*: string, the path to the input file, where the content is stored in bytes.
3. *key file*: string, the path to the key file, where a 64-bit key string is stored in hex format, for example, the key in Table 1 is stored as 0191019101910191.
4. *initial vector file*: string, the path to the initial vector file, where the 64-bit initial vector is stored in hex format.
5. *output file*: string, the path to the output file, where the content is stored in bytes.

Follow the following guidelines and steps for ***cbc_encrypt*** (***cbc_decrypt*** is very similar and not described separately).

Step 1: Padding

DES block cipher uses 64-bit (8 bytes) input/output, so you need to divide messages into a multiple of 8 bytes. Since messages can be variable length, you need to apply bit padding to the message.

1. Consider the original message as binary string, bit padding is to first append a '1' to the message, and then append '0's to make it a multiple of 8 bytes.
2. For instance, if the original message has 8 bytes, you need to append one '1' and sixty-three '0's, that is, append an 8-byte padding. If the original message is 6 bytes, you need to append one '1' and fifteen '0's, i.e. 0x8000, to make it 8 bytes. More about bit padding at [this link (https://en.wikipedia.org/wiki/Padding_(cryptography))](https://en.wikipedia.org/wiki/Padding_(cryptography)).
3. Correspondingly, you need to remove padding after decryption.

Step 2: Cipher Block Chaining
You have 8-byte message blocks after padding. You are then required to encrypt these blocks in the CBC mode, with the key and the initial vector loaded from the corresponding file. The CBC mode is explained in **Appendix**. After encryption, write the ciphertext in bytes to the *output file*.
Step 3: Verification
We provide two sets of plaintext file/key file/iv file/ciphertext file in the project and a script **verify.py** to help you verify your program works.
We provide two sets of plaintext file/key file/iv file/ciphertext file in the project. You can use them to verify your program works.

1. Set 1: plaintext_1, key, iv, ciphertext_1
2. Set 2: plaintext_2, key, iv, ciphertext_2
3. To run verify.py:
    a. *cd task1*
    b. *python verify.py $LANGUAGE*

Deliverable: task1.cpp/java/py
Note: If you use C++, submit a Makefile (you can reuse the Makefile provided in project folder).

Task 2: Collectively Crack a DES key (40 points)

In this task, you are given an 8-byte plaintext and an 8-byte ciphertext. You are required to write your program (**task2.cpp/java/py**) to find the key using the provided block cipher (**des.c/java/py, or des_wrapper.py**). In *Task 1*, you get a 64-bit key, but internally only 56-bit is used in encryption. You need to enumerate these 2^{56} keys. Since exploring the entire 56-bit key space is infeasible for individual students, we divide up the task to each student in the class. That is, each student gets a 28-bit key space and the range assignments can be found in the project folder (**task2/keyrange.csv**). Each student is assigned a Minimum Key and a Maximum Key (inclusive). The correct key is in one of these ranges. You are required to enumerate the assigned keys until you find the correct one or have tried all of them.

We provide a template, **task2.cpp/java/py**. The template contains two TODO parts (**enum_key, crack**). Your job is to implement the two TODOs.
Follow the guidelines and steps for **enum_key** and **crack**.
*Step 1: Valid Key Extraction and Enumeration (**enum_key**)*
You are assigned a key range to enumerate, i.e. $[K_1, K_{2^{28}}]$. They are 64-bit keys in hex format. To use them, you need to do

1. Extract the valid part. A valid 56-bit key is obtained by taking the lowest 7 bits of each of the first eight bytes of the key. The highest bit in each byte is an odd "parity bit". For example, row "Key" in Table 1 shows a DES key with the parity bit, and row "Valid Part" shows the corresponding valid bits.
2. Enumerate all the 56-bit valid keys within $[K_1, K_{2^{28}}]$ and convert them to the corresponding 64-bit keys by setting parity key bits.
3. Implement **enum_key**, so that, it computes the next key from current key.
4. Run this command to verify your **enum_key**
    a. *cd task2*
    b. *python verify.py $LANGUAGE*

Table 1 Comparison of original key (with parity bit) and valid key

Bits	63…56	55…48	47…40	39…32	31…24	23…16	15…8	7…0
Key	00000001	10010001	00000001	10010001	00000001	10010001	00000001	10010001
Valid Part	0000001	0010001	0000001	0010001	0000001	0010001	0000001	0010001

*Step 2: Brute-force and Time Logging (**crack**)*
For each 64-bit key obtained in Step 1, you need to use the provided DES block cipher (**des.c/java/py, or des_wrapper.py**) to encrypt the plaintext and compare it with the ciphertext, or vice versa. You are required to log total number of keys tried and total CPU time it takes. Examples of how to log time in each language are provided in **task1.cpp/java/py** at granularity of microseconds.

Deliverable: task2.cpp/java/py and **task2.txt** with four lines of content. In **task2.txt**, **the first line** is the language you used [C++/Python/Java], **the second line** is the number of tried keys and **the third line** is the total time it takes (in **seconds**). **The fourth line** is either the found key in hex format or text 'NOT FOUND'.
Note: 1. We recommend Python users to use **des_wrapper.py as block cipher** (a wrapper of **des.c**, run **make** to compile the necessary library) because it is faster than the pure Python implementation. 2. The key enumeration can be time consuming and we recommend you to implement pause/resume functionality.

Task 3: Fill-in-the-Blank Question (20 points)

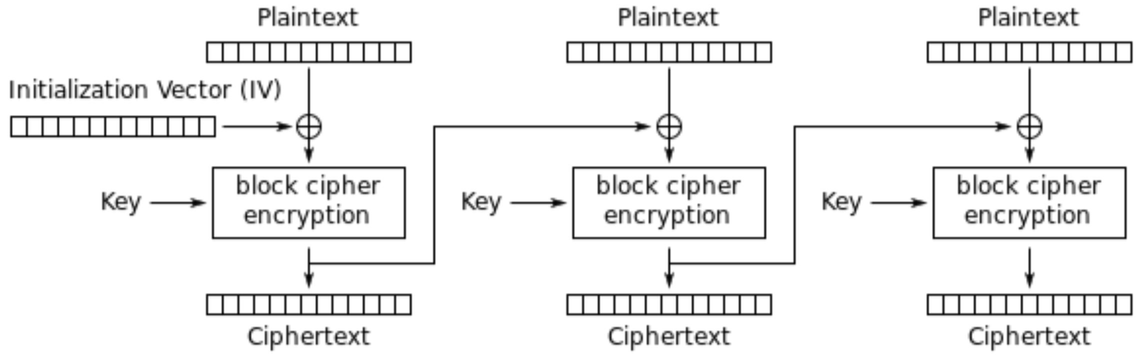
1. In Task 2, you should have an idea of how long it takes to crack a 28-bit DES key. Your time is ________ seconds. How many keys do you need to try to brute-force a 56-bit DES key? The number is ________ keys. How about triple-DES with three different keys? The number is _________ keys. How about AES-256? The number is __________ keys.
2. In block ciphers, plaintexts and ciphertexts are one-to-one mappings, otherwise, the same ciphertext can decrypt to different plaintexts, or vice versa. How about keys, can we find more than one valid keys for a given plaintext and ciphertext pair (as in *Task 2*)? Suppose **K1** maps a given plaintext block **p** to a ciphertext block **c**. Is it possible to find another key **K2** that encrypt **p** to the same ciphertext block **c** in DES? __________ (Yes/No). If Yes, the probability is ___________. Similarly, is it possible to find such keys in AES-256 which takes a 256-bit key and a 128-bit message block? _________(Yes/No). If Yes, the probability is _________. (Put 0 for the probabilities if you select No. Put floating point numbers if you select Yes.)

Deliverable: task3.txt, one line per blank answering the above questions. You submission should have 8 lines.

Note: A formula that might be useful $\lim_{n\to\infty}\left(1+\frac{1}{n}\right)^n = e$

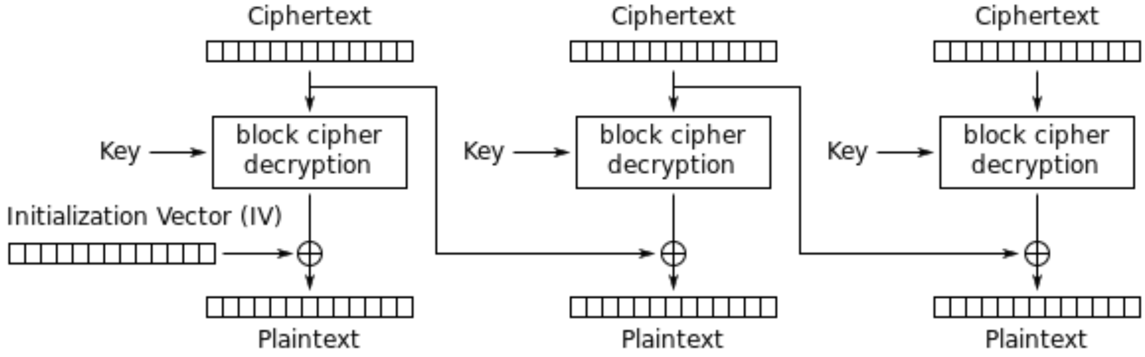
Appendix

CBC Mode Encryption



Cipher Block Chaining (CBC) mode encryption

CBC Mode Decryption



Cipher Block Chaining (CBC) mode decryption