Elementary Encraption: Implementing Cipher Block Chaining in C

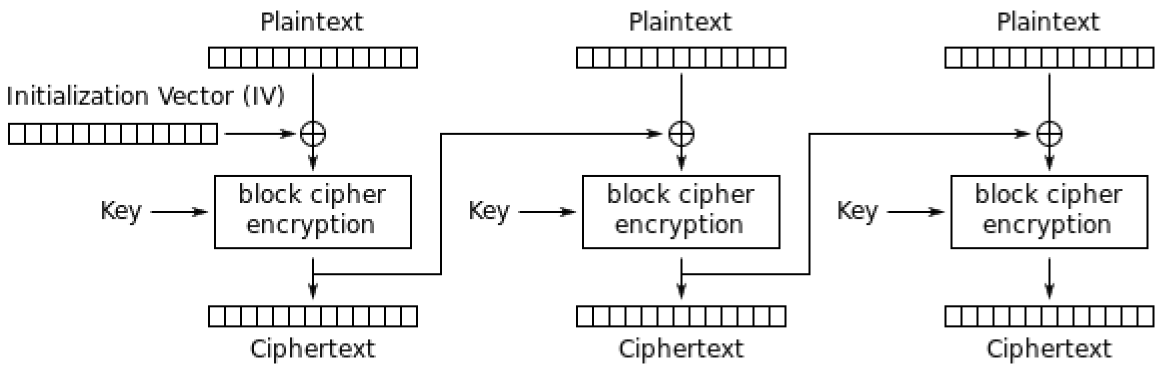
Peter A. H. Peterson <pahp@d.umn.edu>
University of Minnesota Duluth

Overview

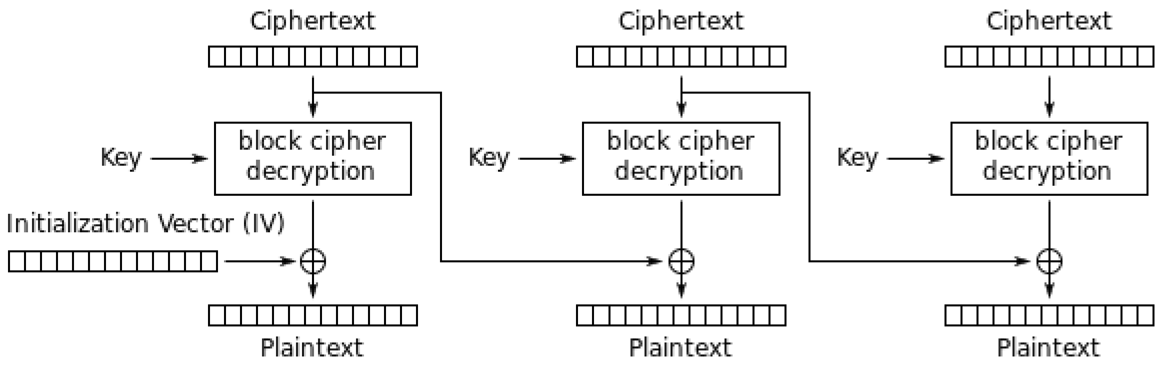
In this exercise, you will implement a CBC mode in your Product Cipher (PC) and answer questions about the output. This will help you understand different cipher modes and the relative quality of our encraption algorithm.

Background

Your primary task is to add Cipher Block Chaining (CBC) mode encryption to your existing product cipher code. In CBC encryption mode, the plaintext is XORed with a block of data prior to the encryption step. For the first plaintext block p_0, the XOR data is called an *initialization vector* or IV and should be drawn from a source of strong randomness. For subsequent blocks p_n, the XOR data used is the ciphertext of the previous block $c_{(n-1)}$. This requires mechanisms for creating the IV and caching the previous ciphertext. For decryption, the first ciphertext block c_0 must be first decrypted and then XORed with the IV. Subsequent blocks c_n are, following decryption, XORed with the previous ciphertext block $c_{(n-1)}$. This can be diagrammed as:



Cipher Block Chaining (CBC) mode encryption



Cipher Block Chaining (CBC) mode decryption

From: https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation#Cipher_Block_Chaining_.28CBC.29 (Accessed 2015/10/13)

In general, the IV is a nonce (like a salt) does not need to be secret but it does need to be unique and random. In practice, the IV can be stored with the data. However, for this lab, you will read the IV from a file (and will not store it with the output).

To combine the IV with the plain- or ciphertext blocks, simply perform a bitwise XOR on each character in the block. This is similar to performing a shift based on the symbols in a Vigenère key, but with the XOR operation instead. For example, the following code will XOR two buffers together:

```
for (i = 0; i < bytes; i++) {

    buf[i] = buf[i] ^ xorbuf[i];

}
```

Requirements

Code and Compilation

Your submission will be electronically graded for correctness and uniqueness. Therefore, your code must successfully compile on the MWAH 187 machines running Ubuntu and must follow the following requirements.

1. It must be submitted as a C source file (.c plain text), not as a PDF, .doc, .docx, etc.
2. Your code **must not** use any of the Banned Functions listed in the Vigenère Encraption lab manual or it will be scored a **zero (0)**.
3. You need to create a single C source file capable of generating object files for both an encoder and decoder. Your code will be compiled using the commands
   `gcc cbc-ctr.c -o encoder -D MODE=ENCODE` and
   `gcc cbc-ctr.c -o decoder -D MODE=DECODE`
   See the Appendix of the Vigenère lab manual for instructions to do this.
4. Your code should not use any libraries other than `stdio`, `stdlib`, `string`, `errno` and `limits` (in particular, it should not require any additional flags to the compiler or modifications to the test script to compile).
5. Your code **should not create any temporary files** during encryption or decryption.
6. You should **NOT** use other online examples of Columnar Transposition, padding, or the Vigenère Cipher as a coding template; if you do not understand how they work, talk to your instructor or TA.
7. **The number of rounds will be read from the command line** (like dimension was for Lab 4).
8. **Keys will be read from a file** as in Lab 3.
9. **An initialization vector (IV) will be read from a file.**
10. **All keys and IVs will be 16 bytes in length.**
11. **Your code must accept an option to disable CBC mode (see below).**
12. The dimension of the transposition box should be hard-coded to 4 (for a block size of 16 bytes).
13. Input may **not** be limited to text only (i.e., test using binary data like applications or images).
14. Input may **not** be limited to any particular length (i.e., test using large files).
15. Your `main()` function should return `0` in the case of a successful exit.

Encoder and Decoder

Your encoder and decoder must take the following parameters as input:

1. The number of rounds to use, e.g., 1 or 5 or 10, etc. (the number of rounds may be arbitrary)
2. A path to a key file, e.g., “keys/key_01”
3. A path to an initialization vector, e.g., “ivs/00_random_01”
4. A path to an input file, e.g., “input/file_01”
5. A path to an output file, e.g., “output/file_01_encoded”
6. An optional parameter “1” that disables CBC mode and encrypts in ECB mode

Upon running:

./encoder 5 keys/key_01 ivs/00_random_01 /tmp/input /tmp/output

...the file 'output' should contain the contents of input as encoded using a 5-round VC-TC product cipher in CBC mode as described above. Upon running:

./decoder 5 keys/key_01 ivs/00_random_01 /tmp/encoded /tmp/decoded

... decoded should contain the contents of encoded as decoded using a a 5-round VC-TC product cipher in CBC mode as described above.

Running:

./encoder 5 keys/key_01 ivs/00_random_01 /tmp/input /tmp/output 1 or
./decoder 5 keys/key_01 ivs/00_random_01 /tmp/encoded /tmp/decoded 1

… should encrypt and decrypt in ECB mode – **skipping the CBC XOR steps**.

Padding, Block Ciphers and More

With the exceptions of adding the CBC step and reading in the IV, everything about this cipher, including padding, fixed dimension of 4x4, order of Vigenère and Columnar, etc., should be the same as your Product Cipher. In other words, your programming task is to add the CBC code to your existing Product Cipher.

Sample Vectors

Sample vectors (i.e., various round settings, keys, and input along with correctly encrypted and decrypted data) will be provided at http://www.d.umn.edu/~pahp/CS4821/labs/cbc-ctr-vectors.tar.gz for correctness testing. See the included README.txt for more information. Grading may use both the sample vectors and new untested vectors. Therefore, you should test your code against the sample vectors and various other types of input and keys that you choose.

Compressibility as an Estimate of Entropy

While encryption attempts to reversibly encode data with a maximum of entropy, compression attempts to reversibly encode data using a minimum of entropy. In other words, compression attempts to encode data using the least number of bits possible. Therefore, you can *estimate* the relative entropy of data by attempting to compress it with an good algorithm like ZIP or gzip and comparing it to the original input. If entropy is high, the data will not compress very much (or might even expand during compression). If entropy is low, the data will compress significantly owing to the large amount of redundancy.

If *F* is the size of a file when not compressed, and *C* is the size of the file when compressed, we say that the *compression ratio* (CR) achieved by the algorithm is *C/F*. For example, if the file was 4 megabytes before compression and 3 megabytes afterwards, we say that the CR is 0.75 because it is ¾ its original size. If the file expands (i.e., *C* is larger than *F*), then the CR will be greater than 1. Normally, when compressing files, we are trying to save space, so we desire the lowest possible CR. However, when compressing encrypted data as a means of estimating entropy, we would like to see a large CR, which would indicate that the ciphertext was difficult to compress and thus had little redundancy.

We are going to use gzip to estimate the effectiveness of our encryption algorithms. First, make sure that your code passes all of the `test.sh` tests. Then, compress all of your output in `ENCODED`. (If you are not able to complete your code, you can answer the questions by compressing the sample data in the `correct` directory.) To compress all the files in the current directory, execute the following command:

```
gzip *
```

This will compress *filename*, creating *filename.gz*.

Next, list the files (and their sizes) using `ls -al > /tmp/encodezip.txt`. Import this file into a spreadsheet (using space and/or tab as the delimiter and having multiple delimiters combine). You need only import columns 5 (the size in bytes) and 9 (the filename). Sort the spreadsheet by the file size. You can calculate the CR for each file by finding the size of the original input and creating a formula using the original and compressed size. For example if the original input size is 4080 bytes, and the compressed size is in cell `A1`, the formula `=A1/4080` will compute the CR. You can “stretch” the formula in one cell into the other cells in a column and it will compute the CR for each cell in the row.

If you are interested in searching the data in other ways, you can also import smaller selections of this data by using wildcard matching. For example, to only list the file sizes of files using CBC mode with the key `00_16_zeros`, you can execute `ls -l *00_16_zeros*CBC`.

Short Answer Questions

Answer the following questions based on the information in your spreadsheet. (All file sizes should reflect their size after compression.)

1. What is the file size and compression ratio of the largest ECB-encoded file(s)?
2. What is the file size and compression ratio of the largest CBC-encoded file(s)?
3. We would expect all CBC encoded and compressed files to be larger than the equivalent ECB files. Why?

4. Unfortunately, some CBC files are actually *smaller* than the equivalent ECB files. Which CBC files are smaller than the ECB files?
5. Files are named according to this scheme: *rounds.key.iv.filename.mode* where *rounds* is the number of rounds, *key* is the key used, *iv* is the initialization vector, *filename* is the input file used and *mode* is CBC or ECB. Can you find any patterns in the smallest CBC files? In particular, are any of our test keys “weak” (i.e., do they lead to poorly encrypted output)? Are any numbers of rounds especially weak?
6. Generally, multiple rounds do not seem to substantially improve entropy for either ECB or CBC mode, and (with some exceptions) key choice for ECB mode has little effect. What properties of the Vigenère and Columnar Transformations result in multiple rounds being largely ineffective?
7. What simple transformation could we add to our product cipher that might improve the value of multiple rounds of Columnar Transformation? Why might it improve entropy?
8. One key in particular is stronger than the others. What is it? Why is this surprising? Does this imply anything about the optimality of the design of our cipher?

Submission

Submit your cbc-ctr.c source and your short answer file according to the directions you have received from your instructor.