CSc 466/566 Computer Security

Assignment 5

Due Tuesday, Nov 6, 2017
Worth 18% (ugrads), 18% (grads)

Christian Collberg
Department of Computer Science, University of Arizona

1. Introduction

In this assignment you will implement DES, in ECB and CTR modes.

You can do this assignment alone, or in teams of two. If you work with someone, please indicate in your writeup who did what.

Efficiency counts for this assignment. Therefore, you must do all your DES manipulations using C's bitwise operations: `&, |, ~, <<, >>`.

NOTE: I'm well aware of the fact that there are many DES implementations available on the web, but you can't look at or copy code from any of them. Of course, you are free to look at any descriptions of the algorithm in the slides, in books, on wikipedia, etc, as long as they don't contain actual code.

You will be graded on correctness as well as on style. it's a good idea to add your own fuctions or macros that makes your implementation as clean as possible. For example, a `swap()` macro that swaps the two halves of a word would probably be useful, as would a function that permutes the bits of a word.

You will be given a file `des.c` to modify. I'm describing pieces of it below.

2. Command Line Iterface

```
/*
 * des takes two arguments on the command line:
 *    des -enc -ecb      -- encrypt in ECB mode
 *    des -enc -ctr      -- encrypt in CTR mode
 *    des -dec -ecb      -- decrypt in ECB mode
 *    des -dec -ctr      -- decrypt in CTR mode
 * des also reads some hardcoded files:
 *    message.txt            -- the ASCII text message to be encrypted,
 *                              read by "des -enc"
 *    encrypted_msg.bin      -- the encrypted message, a binary file,
 *                              written by "des -enc"
 *    decrypted_message.txt  -- the decrypted ASCII text message
 *    key.txt                -- just contains the key, on a line by itself, as an ASCII
 *                              hex number, such as: 0x34FA879B
*/
```

3. DES Tables

All the DES tables have been provided for you:

```
uint64_t init_perm[] = {
        58,50,42,34,26,18,10,2,
        60,52,44,36,28,20,12,4,
        62,54,46,38,30,22,14,6,
        64,56,48,40,32,24,16,8,
        57,49,41,33,25,17,9,1,
        59,51,43,35,27,19,11,3,
        61,53,45,37,29,21,13,5,
        63,55,47,39,31,23,15,7
};
....
```

4. Subkey Generation

To make this assignment manageable, I've ommitted the subkey generation part. Instead, I've given you some hardcoded keys:

```
uint64_t hardcoded_subkeys[] =
{
        0x1b02effc7072,
        ....
};
```

For extra credit, implement the key generation part, and you will have a complete DES implementation:

```
// Each subkey is 48 bits. To simplify the assignment we're hardcoding keys here.
// Note that this means that the key argument to this assignment doesn't matter at
// all, since the subkeys are generated from the input key!
uint64_t getSubKey(int i) {
   return hardcoded_subkeys[i];
}

// For extra credit, write the key expansion routine.
void generateSubKeys(KEYTYPE key) {
   // TODO for extra credit
}
```

5. I/O

The input to the encryption is an ASCII text file, the output is a binary file. The input to the decryption is a binary file (the same that was written by the encryptor), and the output is an ASCII text file (it should be the same as the encryptor read!).

The following functions need to be implemented:

```
BLOCKLIST pad_last_block(BLOCKLIST blocks)
BLOCKLIST read_cleartext_message(FILE *msg_fp)
BLOCKLIST read_encrypted_file(FILE *msg_fp)
KEYTYPE read_key(FILE *key_fp)
void write_encrypted_message(FILE *msg_fp, BLOCKLIST msg)
void write_decrypted_message(FILE *msg_fp, BLOCKLIST msg)
```

6. Encryption

The encryptor and decryptor operate on linked lists of blocks. I.e., we read in the entire input file, split it into 8-byte blocks, and create a linked list from it. This is clearly not a good idea for large files, but it simplifies our implementation.

The function `des_enc` encrypts one block. This is where most of your implementation will be done. The remaining functions work on a list of blocks, either in ECB or CTR mode:

```
BLOCKTYPE des_enc(BLOCKTYPE v)
BLOCKLIST des_enc_ECB(BLOCKLIST msg)
BLOCKLIST des_enc_CTR(BLOCKLIST msg)
```

7. Decryption

The decryptor interface is identical to the encryptor. In fact, feel free to merge the two, only passing in an argument tha indicates the order in which the subkeys will be used.