Operating Systems - Winter 2018

Sambuddho Chakravarty

April 8, 2018

Assignment 4 (Total points: 75)

Due date: April 26, 2018. Time: 23:59 Hrs.

1 Encryption Device

You are already aware that devices also appear as files in the VFS file system. I this assignment you need to create a character device using the `mknod` command. Let us call this device `encdev`. The device computes a simple block cipher (encryption) on a file (the input) and outputs the encrypted file. You need to write a *kernel module* based device driver for the `encdev` device.

As mentioned in the class, kernel modules are binaries which can be "attached" to a running kernel. One does not need to re-compile the kernel, only the module needs to be compiled and loaded/attached to the running kernel.

You would require to write a module so as to use VFS data structures and define `open()`, `read()`, `write()` and `close()` functions for this device. Once you have these functions, you need to be able to access the device via a call to `open()` system call, passing the device file path as an argument. The first call to `write()` internally initializes the shared random key (a random 128-bit number). You may read off random bytes from the `/dev/urandom` device and use that as the random shared key. Subsequent `write()`s would result in the bytes being encrypted and stored internally in the kernel in some data structure, e.g. some dynamically allocated location. The last block to be written would be signaled through an EOF byte being written to the file, which MUST not be encrypted.

Subsequent `read()`s would result in reading off the encrypted blocks, until an EOF is encountered.

Finally, upon calling `close()`, the initialized data structures would be deallocated.

2 Decryption Device

You would also need to create a similar `decdev` device file which would be used to decrement an encrypted file. Similar to `encdev`, you would need top `open()` the file and `write()` blocks of encrypted file to the device. The first block of 128-bits (16 bytes) happens to be the key (the same is one used for encryption and decryption). The subsequent blocks written should be the cipher blocks (encrypted previously via the `encdev`). Here again the last block is signaled via

EOF character. Each of these blocks should be decrypted and stored in some internal data structure.

Thereafter, subsequent `read()` calls should result in reading blocks of decrypted blocks. The final block should be signaled through reading off the previously written EOF character.

The description of the encryption and decryption set-ups are shown schematically in figure 1 and 2 respectively.

You would need to demonstrate the use of these character devices through two programs. The first program would use the `encdev`. You would call the `open()` system call to open the device, write the first block, the randomly generated key to it, and thereafter write out the subsequent blocks, marking the end of file via EOF. Finally, you would need to `read()` the encrypted bytes from the file and store them in a separate file. You would also need to save the key for decryption (say in another file).

To decrypt, you would require another program which would `open()` the `decdev` device. Thereafter you would need to `write()` the encryption key. Subsequent `write()`s would result in decryption of individual blocks of the file (terminated by the EOF). Finally, like earlier, subsequent `read()`s would give you the decrypted plaintext blocks of the file. You may write these decrypted blocks to a separate file.

For this exercise, you could use as input a file containing ASCII text.

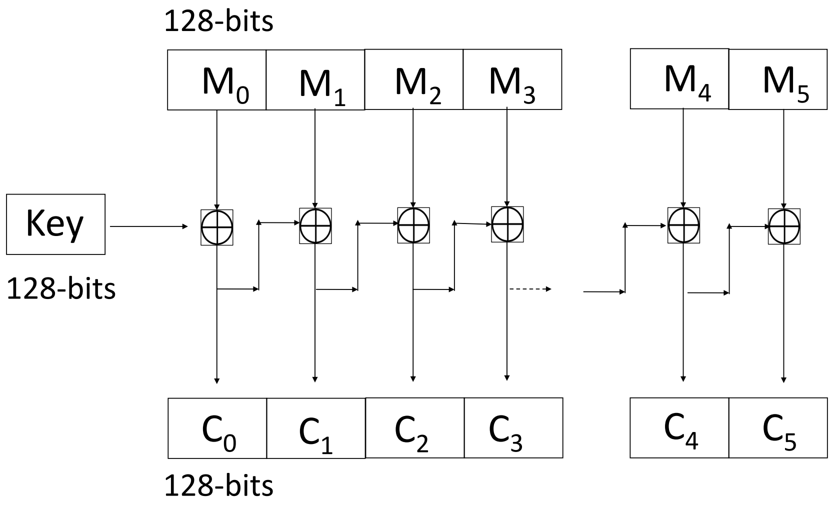

Figure 1: Encryption Set-up

What To Submit

- Program source code, Makefile and script to create the device files and the two test programs to encrypt and decrypt the sample ASCII text file.
- Write-up describing the following:
  - Description of your code and how you implemented the function – the logical and implementation details.

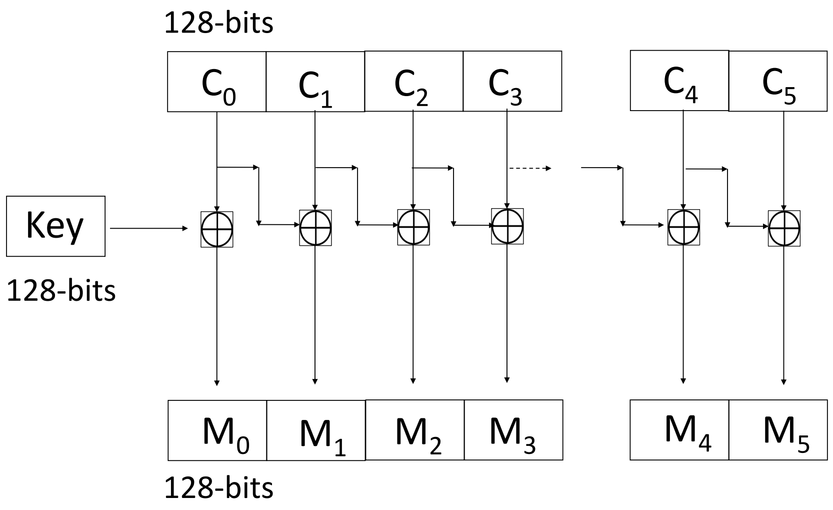

Figure 2: Decryption Set-up

  - Description of how to compile and test the program
  - The inputs the user should give
  - Expected output (and how to interpret it).
  - Error values and how to interpret them.

Grading Rubric

- Successful compilation your the module which can be loaded and unloaded – 20 points.
- Correctly functioning encryption device, demonstrated by the encryption program (though cannot be checked if decryption works correctly or not) – 20 points.
- Correctly functioning decryption device, demonstrated by the decryption program (the final output of the decryption program should be the original plaintext program) – 30 points.
- Description of the systems, test cases *etc.* – 5 points.