NOVA SCHOOL OF SCIENCE & TECHNOLOGY

Computer Architecture 2023/24

TPC1

This homework consists of a programming exercise **to be carried out in a group of no more than two students**. You can clarify general doubts with colleagues, but the solution and writing of the code should be strictly carried out by the members of the group. All resolutions will be automatically compared and cases of plagiarism will be punished in accordance with the regulations in force. **If you use tools such as CoPilot or ChatGPT, you must include a comment in the source code reporting this use.**

Deadline for delivery: 26/3 (Tuesday) at 9:00PM.
Mode of delivery: create a zip file which name is TPC1-XXXXX-YYYYY.zip where XXXXX is the student number of the 1st author and YYYYY is the student number of the 2nd author. If the group has only one element, the name is TPC1-XXXXX-00000.zip. The zip file should contain only two files: the Shell script and the source code of program C. The zip file should be **emailed** only by the student with number XXXXX to the email of Prof. **Kevin Gallagher (k.gallagher@fct.unl.pt)**

Exercise

Encryption is currently the fundamental basis of most techniques used for data protection. There are many different cryptographic schemes, from the simplest to the most complex, with the most complex being the schemes used in real scenarios. One of the simplest data encryption schemes is the so-called *one time pad* (OTP). In this scheme, a *plaintext message* (name given to the original, unencrypted data) is encrypted to a *ciphered message* (name given to the data after encryption) using a *secret key* that has exactly the same length as the *plaintext*, through an XOR or *exclusive or* logical operation. Suppose the clear text has S bytes and is stored in the vector *P[0..S-1]*, the ciphered message will be in the vector *C[0..S-1]*, and the key is in a vector *K[0..S-1]*. For each byte we will have the following

$$C[i] = P[i] \text{ xor } K[i]$$
$$P[i] = C[i] \text{ xor } K[i]$$

One way to generate the appropriate key for this work is to use the pseudo-random number generator that exists in the C library. The major functions to use are:

```
void srand( unsigned int seed)
int rand( )
```

The *srand( )* function takes as an argument a number X (>= 0) that works as the seed of the sequence of random numbers generated by rand( ). When the *rand( )* function is invoked, the next element in the sequence is generated. For the same value of X, the same sequence of numbers is generated.

The practical work consists of creating two programs:

1. A C program that accepts as arguments an input file, an output file and a number (which functions as a seed), and encrypts the contents of the input file using the OTP scheme, through the XOR operation and a pseudo-random key generated from the seed.
2. A bash script that finds all files in a directory (and its subdirectories) and uses the program described in 1. to encrypt them.

Program in C

Specifically, the C program will receive the following arguments:

- The *pathname* of the input file.
- The *pathname* of the output file.
- An integer to be used in the *srand function*.

Using the arguments described above, the C program should, for each byte of the input file,

1. Get the next element of the pseudo-random sequence.
2. XOR this element with the byte.
3. Write the byte that results from 2. into the output file.

If the encryption process is successful, print "Encryption of <input file name> succeeded." otherwise, print "Encryption of <input file name> failed.

Don't forget that arguments in C are stored in the *argv vector* that is passed to the main function

Script bash

The bash script will receive the following arguments:

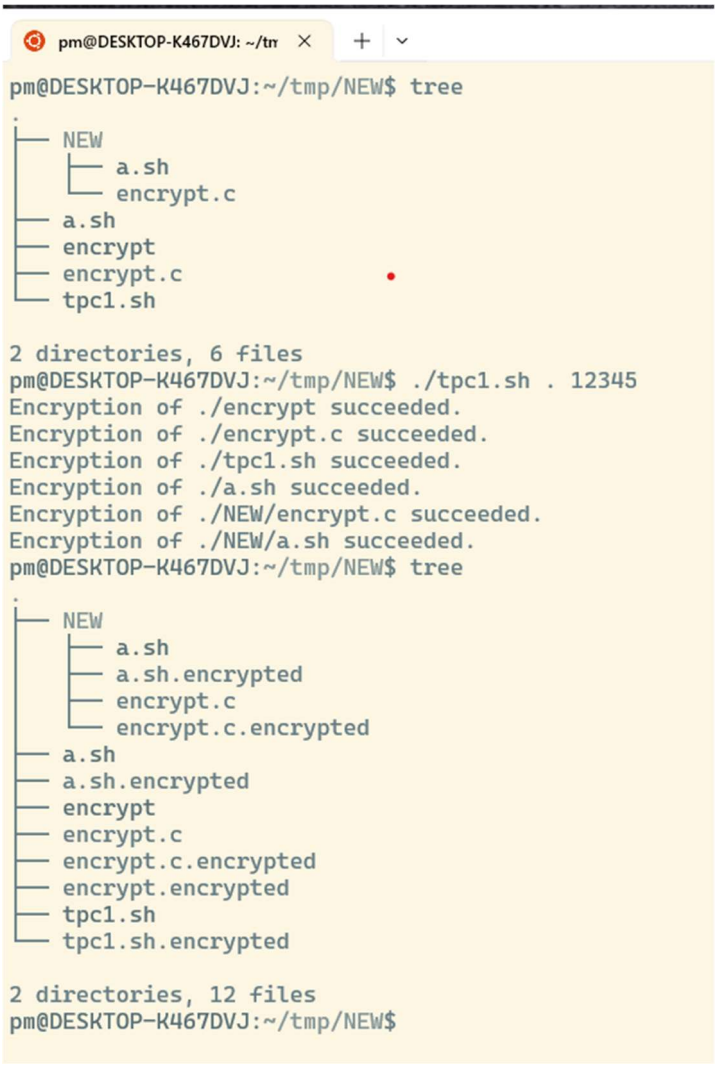
- A directory pathname.
- A *seed*.

Using the arguments described above, the *bash script* should:

- Find all files in the input directory and its subdirectories
- For each file:
    - Ignore files with ".encrypted" extension.
    - Call program C described earlier with the following arguments:

        - The name of the original file
        - the filename of the output that is the result of concatenating the original filename with the string ".encrypted"
        - the seed that is the 2nd argument of the script

The figure below shows an example where the *tpc1.sh* script invokes is invoked with the current directory and the number 12345 as arguments.


pm@DESKTOP-K467DVJ: ~/tm
pm@DESKTOP-K467DVJ:~/tmp/NEW$ tree
.
├── NEW
│   ├── a.sh
│   └── encrypt.c
├── a.sh
├── encrypt
├── encrypt.c
└── tpc1.sh
2 directories, 6 files
pm@DESKTOP-K467DVJ:~/tmp/NEW$ ./tpc1.sh . 12345
Encryption of ./encrypt succeeded.
Encryption of ./encrypt.c succeeded.
Encryption of ./tpc1.sh succeeded.
Encryption of ./a.sh succeeded.
Encryption of ./NEW/encrypt.c succeeded.
Encryption of ./NEW/a.sh succeeded.
pm@DESKTOP-K467DVJ:~/tmp/NEW$ tree
.
├── NEW
│   ├── a.sh
│   ├── a.sh.encrypted
│   ├── encrypt.c
│   └── encrypt.c.encrypted
├── a.sh
├── a.sh.encrypted
├── encrypt
├── encrypt.c
├── encrypt.c.encrypted
├── encrypt.encrypted
├── tpc1.sh
└── tpc1.sh.encrypted
2 directories, 12 files
pm@DESKTOP-K467DVJ:~/tmp/NEW$