CS 558 Introduction to Computer Security, Spring 2024
Programming Assignment 1

Out: 2/7/2024 Wed.
Due: 2/24/2024 Sat. 23:59:59

Read the Salsa20 specification, which can be downloaded from: https://cr.yp.to/snuffle/spec.pdf. Your task is to implement the Salsa20/12 encryption algorithm, a variant of the Salsa20 cipher as specified by Daniel J. Bernstein. Be aware, the given specification describes a Salsa20/20 cipher. This implementation must support key sizes of 128-bit, 256-bit, and in-addition, non-standard 64-bit.
The constant string "expand 08-byte k" is used for 64-bit key, and key is repeated 4 times to fill the initial state in expansion function. Here is a formal definition of the non-standard 64-bit key Salsa20 expansion function: Define $\alpha_0 = (101, 120, 112, 97), \alpha_1 = (110, 100, 32, 48), \alpha_2 = (56, 45, 98, 121), \alpha_3 = (16, 101, 32, 107)$. If k is 8-byte sequences and n is 16-byte sequences then $Salsa20_k(n) = Salsa20(\alpha_0, k, k, \alpha_1, n, \alpha_2, k, k, \alpha_3)$
Your program must accept the following inputs from the command-line: the key length(in bits), the key represented as a hexadecimal string, nonce(IV) represented as a hexadecimal string, and the text string to be encrypted or decrypted also in hexadecimal format. For encryption, this will be plaintext. For decryption, it will be ciphertext. The program's output should be the resulting ciphertext when performing encryption or the original plaintext when performing decryption. They are in the hexadecimal format. A command-line example:

```
#Input:
./your_prog 128 "deadbeefdeadbeefdeadbeefdeadbeef" \
            "1234567890abcdef" "546869736973706c61696e74657874"
#Outpt:
"a1c7720e1abadb96e5a2600d0ce028"
```

In this example, 128 represents the key size in bits, "deadbeefdeadbeefdeadbeefdeadbeef" is the hexadecimal string key, "1234567890abcdef" is the nonce (IV) in hexadecimal string format, and the final argument is the input text, "546869736973706c61696e74657874" hexadecimal format of ASCII string "Thisisplaintext", which can be either plaintext (for encryption) or ciphertext (for decryption). We assume the block number in the encryption and decryption always starts from 0 in this assignment. Eventually, your program should output the resulting(ciphertext/plaintext) string to the command-line.
You can use any programming languages that you are comfortable with for this assignment. Make sure the department machines(Lab G7) or remote machines support your language choices. The assignment will be graded on these machines. Contact the TA if your language is not supported by the department machines.

Recommended implementation steps:

- Read the specification thoroughly. Pay special attention to the algorithm's core functions.
- Implement the `quarterround` function first. This function is the basic building block of the Salsa20 cipher.
- Then, implement the `rowround` and `columnround` function on top of that `quarterround` function.

- Implement the `doubleround` function based on `rowround` and `columnround`.
- Implement the Salsa20 hash function based on `doubleround` function.
- Implement the `expansion` function for different key sizes. Pay attention to the different values used for the different keys.
- Implement the encryption/decryption that can handle the command-line inputs correctly.
- Test your implementation correctness of each function using the input/output inside the specification, make sure its correctness and move to implement the next function.
- Implement the standard Salsa20/20 in the specification, make sure it works as expected, then you can change it to Salsa20/12 without much effort.
- Then make the necessary modification to support the non-standard 64-bit key.

Log and submit your work

Log your work: besides the source files of your assignment, you must also include a README file which minimally contains your name, B-number, programming language you used and how to compile/execute your program. Additionally, it can contain the following:

- The status of your program (especially, if not fully complete).
- A description of how your code works, if that is not completely clear by reading the code (note that this should not be necessary, ideally your code should be self-documenting).
- Possibly a log of test cases which work and which don't work.
- Any other material you believe is relevant to the grading of your project.

Provide a seperate Makefile if you can, so we can just run a make command instead of manually compile your programming using the instructions in your README which could lead to problems.

Compress the files: compress your README file, all the source files in the same folder, and any additional files you add into a ZIP file. Name the ZIP file based on your BU email ID. For example, if your BU email is "abc@binghamton.edu", then the zip file should be "proj1_abc.zip".

Submission: submit the ZIP file to Brightspace before the deadline.

Grading guidelines

(1) Prepare the ZIP file on a Linux machine. If your zip file cannot be uncompressed, 5 points off.

(2) If the submitted ZIP file/source code files included in the ZIP file are not named as specified above (so that it causes problems for TA's grading scripts), 10 points off.

(3) If the submitted code does not compile or execute:

```
1   TA will try to fix the problem (for no more than 3 minutes);
2   if (problem solved)
3     1%-10% points off (based on how complex the fix is, TA's discretion);
4   else
5     TA may contact the student by email or schedule a demo to fix the problem;
6     if (problem solved)
7       11%-20% points off (based on how complex the fix is, TA's discretion);
8     else
9       All points off;
```

So in the case that TA contacts you to fix a problem, please respond to TA's email promptly or show up at the demo appointment on time; otherwise the line 9 above will be effective.

(4) If the code is not working as required in this spec, the TA should take points based on the assigned full points of the task and the actual problem.

(5) Lastly but not the least, stick to the collaboration policy stated in the syllabus: you may discuss with your fellow students, but code should absolutely be kept private.