CSC 152 programming assignment: Build a random permutation

Programming Assignment

One of the videos on Canvas is called "A cryptographic permutation". This assignment has you implement the permutation described there in C. Watch that video, copy the drawing of the "p152 mixing function" into your notes, and make sure you understand it (ask questions if you don't).

Do each of the following. Each will be tested separately.

Step 1) Mixing function

Watch the video "A cryptographic permutation" and implement the drawing called "p152 mixing function". Put the code in a file called `mix.c`. Here is starter code. Implement the mixing function drawing where is says YOUR CODE HERE in the starter code. There is a simple test in the supplied main to give you a signal when you have done it correctly.

```c
#include <stdint.h>

// Static means that it is not exported for linking and so
// can only be called from this file.
static uint32_t rotl32(uint32_t x, int n) {
    return (x << n) | (x >> (32-n));
}

// ap - dp are references to four variables to be mixed.
// A good compiler will keep everything in registers, so even
// though it looks like we're reading memory, we probably aren't.
void mix(uint32_t *ap, uint32_t *bp, uint32_t *cp, uint32_t *dp) {
    // Receive the four words to be mixed
    uint32_t a = *ap;
    uint32_t b = *bp;
    uint32_t c = *cp;
    uint32_t d = *dp;
    // Mix the four words

    YOUR CODE HERE

    // Update the caller's values
    *ap = a;
    *bp = b;
    *cp = c;
    *dp = d;
}

#if 0  // Set to 1 while testing and 0 for submission

#include <stdio.h>

int main() {
    uint32_t a = 0x00010203;
    uint32_t b = 0x04050607;
    uint32_t c = 0x08090A0B;
    uint32_t d = 0x0C0D0E0F;
    mix(&a, &b, &c, &d);
    printf("Is       : %x %x %x %x\n", a, b, c, d);
    printf("Should be: b54718aa afd1b4f0 501eb3c9 4210a1b3\n");
```

```
}

#endif
```

Step 2) p152: Read, mix multiple times, write

In Step 2 we complete the permutation. As described in the video we call `mix` on the four columns and then `mix` on the four diagonals, and we do this ten times. Complete the following starter code and put it into a file named `p152.c`. I suggest you follow the following pseudocode for p152. Again, there's a small main that indicates when your code is correct.

```
p152(in, out):
    copy 64 bytes from in into an auto-allocated array of 16 uint32_t
    10 times do:
        call mix four times, once per column
        call mix four times, once per diagonal
    copy 64 bytes from your array to out
```

Note: Your code will be run on a little-endian machine, and let's agree that p152 is defined to use little-endian reads from in and writes to out. This means that you can use natural C reads and writes throughout your program and do not need to concern yourself about endianness in this assignment. If you wanted your code to work on both big and little endian machines you would need to copy from in and write to out using little-endian reads in both cases.

```c
#include <stdint.h>
#include <stdlib.h>
#include <string.h>

PASTE YOUR CODE FROM PART 1 HERE

void p152(void *in, void *out) {

    YOUR P152 CODE HERE

}

#if 0  // Set to 1 while testing and 0 for submission

#include <stdio.h>

int main() {
    unsigned char buf[64] = {1};  // Puts 1 in frist byte, rest get auto zeroed
    p152(buf, buf);
    // As a test, output the first 16 bytes of the output
    printf("Is       : ");
    for (int i=0; i<16; i++)
        printf("%02x", buf[i]);
    printf("\n");
    printf("Should be: 14627e9771052d97a8a025cc5531572f\n");
}

#endif
```

Step 3) A stream cipher based on p152

We saw in class that a stream cipher can be made out of a public random permutation like p152. The pseudocode looked something like this:

```
// let k be 32 random bytes, n be 16 bytes and previously unused with k
p152stream(k,n):

    copy k || n || 16 bytes of 0 64 bytes from in into an auto-allocated array of 16 uint32_t
    while more output desired:
        p152(in, out)
        output (out xor in)  // The xor of in secures the inverse function
        in += 1
```

The idea here is that every input to p152 is different because "in" is different each time and also secret because the adversary does not know k. Thus the public nature of p152 is of no advantage to the adversary, but the fact that it is designed to act random ensures a good output distribution.

We can turn this into more programmer-friendly pseudocode as follows.

```
p152stream(k,n,infbuf,outbuf,nbytes):
    create two auto-allocated arrays, in and out, of 64 bytes each
    copy k, n, and 16 bytes of 0 into in
    while nbytes >= 64:
        p152(in, out)
        out = out xor in   // This is the next 64 bytes of stream cipher output
        out = out xor (next 64 bytes of inbuf)
        write out to the next 64 bytes of outbuf
        in += 1
        nbytes -= 64
    if nbytes > 0:
        p152(in, out)
        out = out xor in   // This is the next 64 bytes of stream cipher output
        out = (first nbytes of out) xor (next nbytes bytes of inbuf)
        write out to the next nbytes bytes of outbuf
```

Note that because xor is its own inverse, p152stream can be used for turning a plaintext into a ciphertext or for turning a ciphertext into a plaintext. In other words p152stream(k, n, pt, ct, nbytes) followed by p152stream(k, n, ct, pt, nbytes) encrypts pt and then decrypts ct back to its original value.

Complete the following starter code and put it into a file called `p152stream.c`. Again, the small main indicates to you a hint that your code may be working correctly.

```c
#include <stdint.h>
#include <stdlib.h>
#include <string.h>

PASTE YOUR CODE FROM PARTS 1 AND 2 HERE

// Increment buf as if it's a 64 byte big-endian integer
static void increment_block(void *buf) {
    unsigned char *blk = buf;
    int i = 63;
    blk[i] += 1;
    while (blk[i] == 0) {
        i -= 1;
```

```
        blk[i] += 1;
    }
}

// k is 32 bytes, n is 16 bytes, 0* is 16 bytes
// Use blk = k || n || 0* to produce output as blk xor p152(blk), then increment blk
// This output is xor'd with the next 64 bytes of buf to encrypt/decrypt
void p152stream(void *k, void *n, void *inbuf, void *outbuf, int nbytes) {
    unsigned char *pinbuf = inbuf;
    unsigned char *poutbuf = outbuf;
    unsigned char in[64];
    unsigned char out[64];
    // Init in as k || n || 0*
    memcpy(in, k, 32);
    memcpy(in+32, n, 16);
    memset(in+48, 0, 16);

    YOUR CODE HERE

}

#if 1  // Set to 1 while testing and 0 for submission

#include <stdio.h>

int main() {
    unsigned char inbuf[200] = {1};
    unsigned char outbuf[200];
    unsigned char k[32] = {1,2,3,4};
    unsigned char n[16] = {1,2,3,4};
    p152stream(k, n, inbuf, outbuf, 200);
    printf("Is       : ");
    for (int i=0; i<200; i++)
        printf("%02x", outbuf[i]);
    printf("\n");
    printf("Should be: d6c0e5ef8745f9fc4657510ce896e96b9f27c18ba5a8cadc7e2a"
           "6872c51c704983726c6633a9e924a5e9a75b8b9980cfad91501f74315fea6da"
           "0936286e5866ac66e8c3d766b6248f88ee99b468dd9fdcf2c4e65e6df35637b"
           "b245246e0cb97ce689c0b91dbd7212257f98744fae42484ea3afbd419db90ca"
           "38a96d4c6e68cd6c003af8b842733ad4162099b2b2d10bfd48a3fb6e8c5e5ea"
           "59dde8bae3206ce3e80f0acad1540e83e2858f39bccec0a4ece5172194f6d15"
           "e7fd5a26a05cb3b8b8fea979965daf5c1\n");
}

#endif
```

Testing and Grading

Programs in this class are tested against test cases. If your program behaves as the test case expects (ie, according to spec) then credit for the test case is given. Otherwise no credit is given for the test case. This credit/no-credit grading means that it is very important that you test your code well before submission.

You may assume that the inputs are error free.

Testing is a little like a game. You should try to think of all sorts of weird inputs that are legal because they don't violate the specification and verify that your code does the right thing for them. Any ambiguities in the specification should be asked about well before the due date. Throughout this class you will be expected to put some thought into test cases, ask questions, and test your code thoroughly.

The easiest test setup for an assignment that asks you to write a particular function is something like this.

```
#include <stdint.h>
#include <string.h>
#include <stdlib.h>

int foo(void *a, int alen, void *b, int blen) {

}

#if 1     // Set to 1 while testing and 0 for submission
#include <stdio.h>
int main() {
  // Write test from your function here
  return 0;
}
#endif
```

Your main can then contain your test code. Once you're satisfied and ready to submit, set the conditional compilation to 0 to stop it.

Submission

1. The files you turn in (`mix.c`, `p152.c`, and `p152stream.c`) should have a small comment block at the top containing your name, the date, anyone you worked with, and the URL of any online source that provided significant help to you. It is an important habit to give credit to important sources of help. For example:

   ```
   // contains.c by Ted Krovetz. Submitted for CSC 152 July 4, 1776.
   // Pair-programmed with Ellen Watermellon
   // Used the "reverse" code from https://www.techiedelight.com/reverse-a-c-string
   ```

2. I will compile your code with gcc flags `-Wall -Werror -fsanitize=address`. Wall "enables all the warnings about constructions that some users consider questionable, and that are easy to avoid". Werror turns all warnings into errors which prevent successful compilation. Fsanitize=address causes runtime checks to detect some (but maybe not all) illegal memory accesses. You should run some tests with these settings as well to ensure maximum credit.
3. Follow the directions at https://krovetz.net/152/fileinbox.html very carefully to submit your file. Remember your file should not have a main because I will use my main to test.
4. Your submission should not print anything (like debugging statements) unless the assignment says it should.

Collaboration

You may collaborate with *one or two* other students on this homework if you wish, or work alone. Collaboration must be true collaboration however, which means that the work put into each problem should be roughly equal and all parties should come away understanding the solution. Here are some suggested ways of collaborating on the programming part.

- Pair programming. Two or three of you look at the same screen and only one of you operate the keyboard. The one at the keyboard is the "driver" and the other is the "navigator". The driver explains everything they are doing as they do it and the navigator asks questions and makes suggestions. If the navigator knows how to solve the problem and the driver does not, the navigator should not dictate solutions to the driver but instead should tutor the driver to help them understand. The driver and navigator should switch roles every 10-15 minutes. Problems solved this way can then be individually submitted.

- Code review. The members of the collaborative each try to solve the problem independently. They then take turns analyzing each other's code, asking questions trying to understand each other's algorithms and suggesting improvements. After the code reviews, each of you can then fix your code using what you learned from the reviews. Do not copy code. If the result of code review is that your code needs changes to be more like your partner's, do not copy it. Instead recreate your own variant without looking at your partner's.

The goal is to learn enough from one another so that you each can do similar problems independently in an exam situation.

If you want a collaborator but don't know people in the class, you can ask on Discord and/or use the group-finding post on Piazza.