Homework 4

CSC 152 – Cryptography

If anything in this assignment does not make sense, please ask for help.

Quiz: We will have a closed-note 20-30 minute quiz on this homework in class Fri Mar 29.

Due: You should consider the due date to be when you take the quiz because the quiz may cover anything related to this homework, including programming topics. However, anything submitted before grading occurs will be considered on-time.

Non-submitted work:

Read: Chapters 6 and 7 from *Serious Cryptography*. Notes on universal hashing: `http://krovetz.net/152/universal.pdf`.

Written Problems:

There are three steps to follow for the written problems. Step 1: Do them as if they were homework assigned to be graded (ie, take them seriously and try to do a good job). Step 2: Self-assess your work by comparing your solutions to the solutions provided by me. Step 3: Revise your original attempts as little as possible to make them correct. Submit both your original attempt and your revision as separate files.

Submit two files to DBInbox following the procedure documented on Piazza. The first file should be named exactly **hw4.pdf** and should be your homework solutions before looking at my solutions. The second file should be named exactly **hw4revised.pdf** and should be your homework solutions after looking at my solutions and revising your answers.

NOTE: If you wish to handwrite your solutions you may, but only if your handwriting is easy to read and the file you submit is less than 1MB in size.

1) Recall that you can compute a value that is equivalent to $x \bmod (2^a - b)$ as $(x \operatorname{div} 2^a) \cdot b + (x \bmod 2^a)$. Use this fact to reduce (base-10) 123456789 mod $(2^{12} - 2)$ to a smaller, equivalent number. If after doing this reduction once, the result is more than 12 bits, do it a second time to reduce it further.

2) Recall that H is ε-almost-universal if the probability $h(a) = h(b)$ is no more than ε when $a \neq b$ and $h \in H$ is chosen randomly. The following H is a family of functions all with domain $\mathbb{Z}_6$ and co-domain $\mathbb{Z}_4$. For what value of ε is H ε-almost-universal? Show your work. H is defined as follows:

	h1	h2	h3	h4	h5
0	2	3	0	1	3
1	3	2	1	0	0
2	0	1	3	2	1
3	0	0	2	2	3
4	2	1	1	3	2
5	0	3	3	2	0

Programming:

Read: C program requirements at `http://krovetz.net/152/programs.html`.

A) Implement a sponge-based cryptographic hash using P152 that we'll call P152-Hash. It should be constructed as described in and around Figure 6-7 of *Serious Cryptography*. H_0 should be all zero bits, 10^* padding should be used, capacity c should be 384 bits, rate r should be 128 bits, and the output should be 192 bits. Note that in class I showed M_i being xor'd with the end of the intermediate values and Figure 6-7 shows it xor'd with the beginning. Both are equally secure, but let's follow the book. Also note that although the book shows two arrows going into P, it is really a single input but drawn separately to emphasize the rate and capacity parts of the input.

Endianness is not an issue in this program. P152 has a memory-based interface, so whenever we want to do an xor with the beginning of an intermediate value, we do it in memory. For example the following xor's the first `num_bytes` of the `src` memory buffer into the first `num_bytes` of the `dst` memory buffer.

```
void xor(unsigned char *dst, unsigned char *src, int num_bytes) {
    for (int i=0; i<num_bytes; i++)
        dst[i] ^= src[i];
}
```

The code you submit should have the following function defined.

```
void P152_hash(unsigned char m[], unsigned mbytes, unsigned char res[24])
```

It reads `mbytes` bytes from `m` and writes the 24 byte result to `res`. Do not include your P152 implementation in your file. Instead put the following external declaration in your file and compile P152 in a separate file.

```
void P152(unsigned char dst[64], unsigned char src[64]);
```

Submit your function via DBInbox in a file named exactly `hw4_P152_hash.c`. You will be supplied with a working P152 function via Piazza if your P152 does not work.

B) Implement polynomial hashing using Horner's method and divisionless mod. Data you read should be 10^* padded to a multiple of two bytes, then read little-endian two bytes at a time, and then evaluated using a given key and reduced modulo $2^{31}-1$. If after padding and reading the data you have integers $x_0, x_1, \ldots, x_{n-1}$, then the polynomial you should evaluate is $(k^{n+1} + x_0 k^n + x_1 k^{n-1} + \cdots + x_{n-1} k^1) \bmod (2^{31}-1)$. Note that this version is slightly different from the version presented in class. This version begins with k^{n+1} which allows the hash to be secure when hashing data with differing lengths. Part of your assignment is to figure out how to modify Horner's method to make this change.

The code you submit should have the following function defined.

```
uint32_t poly31(unsigned char m[], unsigned mbytes, uint32_t k)
```

Note that the key and return types should be considered numbers rather than byte-strings and so you do not need to worry about endian issues with them. Both the key and return values will be less than $2^{31}-1$.

Submit your function via DBInbox in a file named exactly `hw4_poly31.c`.