Homework 4

CSC 152/252 – Cryptography
Due: 11:59pm, Saturday, October 27

Read the document "Homework procedures" posted in Piazza resources for how to turn in your homework and policies on collaboration. A small sampling of your written homework and most of your programs will be graded over the semester. To minimize the chance that bad luck causes you a bad grade, do every problem to the best of your ability. Show your work. Ask questions if you need help.

What to turn in: Program files named exactly `hw4_P52_hash.c`, `hw4_collision.c`, and `hw4_poly61.c` and a single file with your written solutions named `hw4.pdf`. Submit all three to `http://dbinbox.com/cscx52/xxxx` where `xxxx` should be replaced by your four-digit code. Verify submission using the DBInbox link sent to you earlier in the semester. Mis-submitted or mis-named files will receive no credit.

References: Chapters 6 and 7 from *Serious Cryptography*. Notes on universal hashing: `http://krovetz.net/x52/universal.pdf`.

Written Problems:

1) Recall that you can compute a value that is equivalent to $x \bmod (2^a - b)$ as $(x \operatorname{div} 2^a)b + (x \bmod 2^a)$. Use this fact to reduce $123456789 \bmod (2^{16} - 2)$ to a 16-bit value. If after doing this reduction once, the result is more than 16 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

3) Write one of the following reductions: from PreimageFinder(H, y) to 2ndPreimageFinder(H, x), or from 2ndPreimageFinder(H, x) to PreimageFinder(H, y). What security implication does your reduction establish?

Programming:

P1) Implement a sponge-based cryptographic hash using P52. We'll call it P52-Hash. Our sponge will have capacity 32 bytes, rate 16 bytes, and a specifiable number of output bytes. Hash inputs should use 10^*1 padding. The code you submit should have the following function defined.

```
void P52_hash(unsigned char m[], unsigned mbytes, unsigned char res[], unsigned rbytes)
```

It reads `mbytes` bytes from `m` and writes the `rbytes` bytes result to `res`. Do not include your P52 implementation in your file. Instead put the following external declaration in your file and compile P52 in a separate file.

```
void P52(unsigned s[12]);
```

P2) Update and submit the following program by changing the values of `SZ`, `buf1`, and `buf2` so that `SZ` is as big as you can make it and the program outputs that a collision occurs. Do not include your P52-Hash

implementation in your file. Instead have an external declaration in your file and compile P52-Hash in a separate file.

```
#include <stdio.h>
#include <string.h>

#define SZ 5

void P52_hash(unsigned char m[], unsigned mbytes, unsigned char res[], unsigned rbytes);

int main() {
    unsigned char buf1[] = {0x12, 0x17, 0xf3};
    unsigned char buf2[] = {0x0a, 0xc3, 0x19, 0x34};
    unsigned char res1[SZ], res2[SZ];
    P52_hash(buf1, sizeof(buf1), res1, sizeof(res1));
    P52_hash(buf2, sizeof(buf2), res2, sizeof(res2));
    if (memcmp(res1,res2,SZ)==0)
        prtinf("%d-byte Collision\n", SZ);
}
```

You will probably need to write another program to find the values, but you do not need to submit this other program. Some of your grade on this program will be associated with how big a `SZ` value you can find.

P3) Implement polynomial hashing using Horner's method and divisionless mod. Data you read should be 10^* padded to a four-byte boundary, then read little-endian four bytes at a time, and then evaluated using a given key and reduced modulo $2^{61}-1$. If after reading the data you have integers $x_0, x_1, \ldots, x_{n-1}$, then the polynomial you should evaluate is $(k^n + x_0 k^{n-1} + x_1 k^{n-2} + \cdots + x_{n-1}k^0)$ mod $(2^{61}-1)$. To avoid timing-attacks, your code should have no data-dependent conditional code. The code you submit should have the following function defined.

```
uint64_t poly61(unsigned char *m, unsigned mbytes, uint64_t k)
```

The type `uint64_t` is defined in `stdint.h` and is a 64-bit unsigned integer. 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 $2^{61}-1$.

Compiling) Put your code into the files specified above and submit them via DBInbox. Your files should include the required function and should not include `main` unless specified to do so, should not print anything unless specified to do so, should be appropriately documented, and should compile without warning or error when compiled using `gcc` or `clang` with compiler options `-std=c99 -Wall -Wextra -pedantic -c`. The only headers your file is allowed to include are the standard ANSI C headers.

As an extra check, I may compile and run your code using Clang's `-fsanitize=address` command line option which looks for your program writing to memory it should not write to. Athena and Macs have Clang, so you may want to test your code on one of those machines. The best way to test your program is on Athena or a Mac with the following command line:

```
clang -std=c99 -Werror -Wall -Wextra -pedantic -fsanitize=address
```

If your code compiles without problem, doesn't produce any error reports when running, and produces correct results, then your code is in good shape.