Homework 5

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 Monday April 22.

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 8 and 10 from *Serious Cryptography*.

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 **hw5.pdf** and should be your homework solutions before looking at my solutions. The second file should be named exactly **hw5revised.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 the extended GCD algorithm goes like this.

```
When calculating the GCD of a and b, repeatedly do the following:
   Rewrite egcd(x, y) as egcd(y, r) where x = qy + r for some 0 <= r < y
   Solve for r:  r = x + (-q)y
   Substitute combinations of a and b for x and y, and simplify
```

At each iteration of the algorithm, you get a new r as a combination of a and b, which can be used in later iterations for substitution. The algorithm terminates when $r = 0$, meaning that y is the GCD.

Follow this process to find egcd(59,55) and format your intermediate results as seen in class.

2) Compute $55^{-1} \bmod 59$ using the result of Problem 1. Explain.

3) In a prior homework you needed to compute $(x^3 + x^2 + x + 1)^{-1} \bmod x^4 + x + 1$. Use the egcd algorithm to compute it.

4) Let $p = 367$ and $q = 373$ be randomly chosen primes. Use them to produce a public and private RSA key. When it comes time to pick e, choose the smallest value greater than 1 that qualifies. When it comes time to find an inverse, use the extended GCD algorithm to find it. Use your public key to encrypt 5, and show that your private key returns 5 when decrypting the result.

5) In class we saw an exponentiation algorithm that runs in time proportional to the log of the exponent. Follow that algorithm to compute $12^{13} \bmod 13$. Mod each of your intermediate values to keep them from getting too big. Format your computation similar to the following example which computes $3^5 \bmod 10$: Exponent 5 in binary is 101 indicating (from left-to-right) SQ/MULT - SQ - SQ/MULT.

$$\begin{aligned} 1 &= 3^0 \\ 1^2 \cdot 3 = 3 &= 3^1 \\ 3^2 = 9 &= 3^{10} \\ 9^2 \cdot 3 = 3 &= 3^{101} \end{aligned}$$

The first line shows the identity equal to the base raised to 0. Each subsequent line shows on the left the prior value squared

or squared and multiplied by the base, as appropriate, and then the result modulo the modulus. This is followed by the base to the current exponent in binary.

Programming:

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

A) Recently, there has been a push to combine the operations of encryption and authentication. When done separately the number of keys is increased and there is a greater chance of making a programming mistake (such as using the same key for both in an unsafe way or ordering the operations suboptimally). The most popular authenticated-encryption algorithm is GCM (`https://en.wikipedia.org/wiki/Galois/Counter_Mode`). It basically combines a Wegman-Carter MAC (using polynomial hashing for the ε-AU hash and AES for the random function) and AES-CTR for encryption. It uses the same AES key for several purposes, but is careful to ensure that the inputs to AES are all distinct (as long as all the nonces used are distinct).

To acquaint yourself with the inner workings of GCM, I've designed an authenticated-encryption algorithm that uses the same basic design as GCM, but uses Poly31 for hashing and P152-BC as the block cipher. This is not as secure as GCM because Poly31 has such a short modulus. I compensate some for this by running Poly31 multiple times with different keys, but it's still not as secure.

In the following description `num2strBE` and `num2strLE` describe whether to write a number big-endian or little-endian (respectively) and how many bits to use when writing. `num2strBE` is used to make sure CTR encryption is done with a big-endian counter. Other than that, everything else is done with natural little-endian reads and writes. In this description `s[i..j]` means bits i through j of s, inclusive, with indices beginning at 0; and `||` means concatenation. The 0* should insert minimal zeros to make `A||C||0*` a multiple of 4 bytes in length (if `A||C` already is, none are inserted).

```
// K can be any length up to 63 bytes, X is 64 bytes
P152-BC(K, X)
   paddedK = K || 10*   // pad to make K 512 bits
   return P152(X xor paddedK) xor paddedK

// K can be any length up to 63 bytes, N is 12 bytes
PCM-Encrypt(K, N, A, P)
   Kpoly = P152-BC(K, num2strBE(0, 512))
   WCfinal = P152-BC(K, N || num2strBE(1, 416))
   C = P152-BC-CTR(K, N, P)                          // Begin ctr with N || num2strBE(2, 416)
   X = A || C || 0* || num2strLE(bitlen(A), 32) || num2strLE(bitlen(P), 32)
   for i in {0,1,2,3}
      Ki = str2numLE(Kpoly[i*32...i*32+31]) >> 2
      Yi = num2strLE(Poly31(X, Ki), 32)
   T = (Y0 || Y1 || Y2 || Y3) xor WCfinal[0..127]
   return (C, T)
```

NOTE: The above pseudocode was modified 4/12 12pm.

The code you submit should have the following function defined.

```
void pcm_encrypt(int kbytes, unsigned char k[kbytes],  // key, up to 63 bytes allowed, 16 or 32 recommended
                 unsigned char n[12],                  // nonce, must not repeat with same key
                 int abytes, unsigned char a[abytes],  // associated data - is authenticated too
                 int pbytes, unsigned char p[pbytes],  // plaintext to be encrypted
                 unsigned char c[pbytes],              // destination for ciphertext
                 unsigned char t[16])                  // destination for authentication tag
```

Submit your function via DBInbox in a file named exactly `hw5.c`. Your file should include the following headers and should not include definitions of these functions. When I test your submission, I will compile it with my version of these functions.

```
uint32_t poly31(unsigned char m[], unsigned mbytes, uint32_t k);
void P152(unsigned char dst[64], unsigned char src[64]);
```

Crowdsourcing is your best test for correctness. Keep an eye on Piazza for (possibly) more hints an/or information.