Homework 7

Due Thursday, October 29

Implementation Part

For several of the implementation parts of the problem you will need to choose large secret primes, so make sure you have access to `findPrime` and all the functions it depends on from the takehome project.

1. Let's begin by writing functions that efficiently compute Legendre and Jacobi symbols.

   (a) Write a function `legendreSymbol(a,p)` which takes as input an integer a and an *odd* prime p, and returns the Legendre symbol $\left(\frac{a}{p}\right)$ in $\mathcal{O}(\log(p))$ time.

   (b) Write a function `jacobiSymbol(a,b)` which takes as input integers a and b where b is odd and positive and returns the Jacobi symbol $\left(\frac{a}{b}\right)$ *without factoring* b. We remind you of the following properties of Jacobi symbols which should help with your computation.

   - This only depends on the residue of a modulo b.
   - If $a \equiv -1, 0, 1, 2 \mod b$ this is easy to compute directly (using quadratic reciprocity for -1 and 2).
   - If b is prime then this is a Legendre symbol! (`probablyPrime` will help determine this quickly).
   - You can use quadratic reciprocity to relate $\left(\frac{a}{b}\right)$ and $\left(\frac{b}{a}\right)$. Since we can reduce b modulo a this gives us a strictly smaller problem! (**Warning:**, if a is even the $\left(\frac{b}{a}\right)$ doesn't make sense! You will have to factor out the 2's from a use the multiplicativity of the Jacobi function to deal with this case!)

   (c) Compute the following Jacobi symbols. For the first 3 you can check your work by hand.

$$\left(\frac{8}{15}\right), \left(\frac{11}{15}\right), \left(\frac{12}{15}\right), \left(\frac{171337608}{536134436237}\right).$$

2. Implement the Goldwasser-Micali probabilistic encryption scheme to securely send 1 bit of data.

   (a) Create a function `GenerateGMKey(b)` which creates a Goldwasser-Micali key from b-bit primes. In particular, it should output a *public key* $[N, a]$ where $N = pq$ is a product of (secret) b bit primes and $a \in \mathbb{Z}/N\mathbb{Z}$ is a quadratic *non*residue modulo p and q, as well as a *private key* which should just consist of one of the secret primes. (You can use part 1 to compute Legendre symbols.)

   (b) Write functions `GMEncrypt(publicKey,m)` and `GMDecrypt(privateKey,c)`. The first takes a Goldwasser-Micali public key, and a bit $m = \{0, 1\}$, and returns a ciphertext $c \in \mathbb{Z}/N\mathbb{Z}$ which is a square modulo N if and only if $m = 0$. The second recovers m from c and the private key and Legendre symbols. (*Note*: When you compute the random integer in the encryption function make sure it is larger than the square root of N. For extremely large N, you should make use of the sage integer square root function `isqrt()` since floats may round up to infinity and have trouble reconverting to ints).

   (c) Generate and print a Goldwasser-Micali key from 16 bit primes. Use it to encrypt and then decrypt both both 0 and 1. Confirm that you recover the bit correctly.

(d) Implement a Goldwasser-Micali key from primes $p = 151$ and $q = 233$. Recover the bit from the cipher $c = 33482$.

3. Implement the RSA Digital Signature algorithm. It will look very similar to the first project, and you are welcome to reuse code from that assignment (especilly for key generation).

   (a) Write a function `generateRSAKey(b)` which generates an RSA private signing key and public verification key from primes `b` bits long. The verification key will be a pair `[N,e]` where $N = pq$ is a product of secret primes b bits long and e is an integer prime to $(p-1)(q-1)$, and the signing key will be a pair `[N,d]` for the same N and d the inverse of e modulo $(p-1)(q-1)$ (sound familiar?).

   (b) Write functions

   `RSASign(signingKey,document)` and
   `RSAVerify(verificationKey,document,signedDocument)`.

   The former will sign a document with the signing key (by taking an appropriate root), and the second will verify that a document is correctly signed (by exponentiating).

   (c) Generate an RSA digital signature key from 16 bit primes. Use it to sign the document $D = 314159$. Run your verification algorithm for twice, once with the signed document and once with the unsigned document, and confirm you get the expected results.

   (d) My RSA key has not changed since the project. In this exercise we will use it as a public verification key.
   $N =$
   1052213111141408392014271339505076961744243365769525514653731189643431114436152105697372008556663566615508174430418471897265040190399740387723795071615822787835192969996871948722241847235749521621664409450915052922155849207003940344222636566164753741150860865113711385670400779741438758156704996824685344664 39.
   $e =$
   21016836287029986747723759800774101835281275854873180166245543058525887395193963077668602549901653960543032592283482973078890937198353136566265224452055576540700523282705867083716006709316664895297672507188173908369139892947290295938964443139784969754551470562127704086246141892044152375206351855610254115515.
   You recieve two documents:
   $D = 445915856905197344451931056052999335315688923420907486019700081 37$
   $D' =$
   33737713092926002711997965938686781423985198340985423980678740668540837920165714308759593604261668849045652971875876 2313
   Run `intToText` to read this documents (note: they are not encrypted). Each claims to be signed by me, and comes with a digital signature:
   $D^{sig} =$
   8426665688163375964593143441458964637347414029969631856884655887135468399266491115775348447213721512083388365998130055074496366371336684453834515335491064699408986735313535595863367588020756514250266686972301325507655516539455483216842477664155990369846421116298283435969524302141991267497103600153062626 9241

$D'^{sig} =$
90592809313509991477767898543561252818730285233946708276918316631997361944737
64728663367105923665686312882726901254765910591149065525748838442609274074694
58299675381621160449150556922441295777712973489019131240798265816551541474157
14324848399155672059930064485203220452985824881282662030076919344957027591627
Which message is truthful?

4. Implement Elgamal digital signatures.

   (a) Write a function `generateElgamalKey(p,g)` which takes as input a prime p and a primitive root $g \in \mathbb{F}_p^*$, chooses a secret exponent a (at random) and returns a private signing key $[a, p, g]$ and a public verification key $[A, p, g]$.

   (b) Write functions

   `elgamalSign(signingKey,document)` and
   `elgamalVerify(verificationKey,document,signedDocument)`.

   The former will sign a document with the given signing key, returning the signed document as a pair $[S_1, S_2]$ where $0 \leq S_1 < p$ and $0 \leq S_2 < p-1$. The second will verify that the document was correctly signed. Both should follow the elgamal digital signature protocol defined in the October 22 lecture and described in Table 4.2 of [HPS].

   (c) If you sign 2 different documents with the same random element a, your Elgamal signing key becomes insecure. In problem 8 you will describe an algorithm to steal that signing key. Implement that algorithm here. In particular, implement an algorithm

   `stealElgamalSignature(verificationKey,D,Dsig,D',D'sig)`.

   which takes as input verification key associated to an Elgamal signature (which is public information), as well as 2 distinct documents signed with that key. It will first check if those 2 documents were signed with the same random value k, and if they were it will return the signers secret exponent a. (This attack was used to steal Sony's digital signature in 2013!).

   (d) Let $p = 3700273081$, and $g = 7$. Create an Elgamal key, and use it to sign the document $D = 314159$. Then verify that the signature was valid.

   (e) Suppose Samantha has a public verification key $[A, p, g] = [185149, 348149, 113459]$. Suppose Samantha signed the following 2 documents:

$$\begin{aligned} D &= 153405 \qquad D^{sig} = (S_1, S_2) = (208913, 209176) \\ D' &= 127561 \qquad D'^{sig} = (S_1', S_2') = (208913, 217800). \end{aligned}$$

   Use `stealElgamalSignature` to steal Samantha's signing exponent.

5. Implement DSA.

   (a) Write a function `generateDSA(p,q,g)` which takes as input primes p, q with $p \equiv 1 \mod q$, and an element $g \in \mathbb{F}_p^*$ of order q. It then chooses chooses a secret exponent a (at random) and returns a private signing key $[a, p, q, g]$ and a public verification key $[A, p, q, g]$.

   (b) Write functions

`DSASign(signingKey,document)` and
`DSAVerify(verificationKey,document,signedDocument)`.

The former will sign a document with the given signing key, returning the signed document as a pair $[S_1, S_2]$ where $0 \leq S_i < q$ for $i = 1, 2$. The second will verify that the document was correctly signed. Both should follow the DSA protocol defined in the October 22 lecture and described in Table 4.3 of [HPS].

(c) Let $p = 48731$ and $q = 443$. Assume that 7 is a primitive root for $\mathbb{F}_p^*$. Use this to find an element of order 443 in $\mathbb{F}_p^*$. Call this element g.

(d) Generate a DSA key for p, q, g in part (c). Use it to sign the document $D = 314$ and verify that this signature is valid.

Written Part

6. Let's do a few checks from the implementation part.

   (a) Compute the Jacobi symbols $\left(\frac{8}{15}\right), \left(\frac{11}{15}\right), \left(\frac{12}{15}\right)$ by hand and confirm your solutions from 1(c) are correct.

   (b) In the Goldwasser-Micali algorithm it was suggested that the random number be chose as greater than $\sqrt{N}$. Why?

7. Let p be an odd prime and $g \in \mathbb{F}_p^*$ a primitive root. Fix any $h \in \mathbb{F}_p^*$. In this problem we study how to get information about $\log_g(h)$.

   (a) Describe how to easily tell $\log_g(h)$ is even or odd.

   (b) We can write $\log_g a$ in binary:

   $$\log_g a = \varepsilon_0 + \varepsilon_1 \cdot 2 + \varepsilon_2 \cdot 2^2 + \varepsilon_3 \cdot 2^3 + \cdots \qquad \varepsilon_i \in \{0, 1\}.$$

   Explain why (a) means that we know ε_0. This property is summarized as saying that the *first bit* of the discrete log problem over $\mathbb{F}_p$ is insecure.

   (c) If $p - 1$ is divisible by higher powers of 2, we can recover more bits! Factor $p - 1 = 2^k m$. Describe an algorithm to compute the first k bits of $\log_g h$, that is, to recover $\varepsilon_0, \varepsilon_1, \cdots, \varepsilon_{k-1}$. You may assume that there is a fast algorithm to compute square roots modulo p (if $p \equiv 3 \mod 4$ we described such an algorithm is class, but there is a general fast algorithm which we may encounter in the coming weeks).

8. Let p be a prime number and $g \in \mathbb{F}_p^*$ a primitive root. Let i and j be integers such that $\gcd(j, p - 1) = 1$. Let A be arbitrary. Set:

   $$\begin{aligned} S_1 &\equiv g^i A^j \mod p \\ S_2 &\equiv -S_1 j^{-1} \mod p-1 \\ D &\equiv -S_1 i j^{-1} \mod p-1 \end{aligned}$$

   (a) Show that the pair (S_1, S_2) is a valid Elgamal signature for the document D. In particular, this means Eve can produce valid Elgamal signatures.

   (b) Explain why this doesn't mean that Eve can forge Sam's signature on a given document. What extra information would allow Eve to do this?

9. In this exercise we describe a potential security flaw in the Elgamal digital signature algorithm. Suppose that Samantha made the mistake of signing two documents D and D' using the same random value k.

   (a) Explain how Eve can immediately recognize that Samantha has made this blunder.

   (b) Let the signature for D be $D^{sig} = (S_1, S_2)$ and the signature for D' be $D'^{sig} = (S'_1, S'_2)$. Explain how Eve can recover Samantha's secret exponent k.