Problem Set 5

This problem set is due *via email,* to `6.857-submit@theory.csail.mit.edu` on *Monday, October 30* by the beginning of class.

You are to work on this problem set in groups of three or four people. Problems turned in by individuals, pairs, pentuples, etc. will not be accepted. Be sure that all group members can explain the solutions. See Handout 1 (*Course Information*) for our policy on collaboration. If you do not have a group, let us know.

Homework must be submitted electronically! Each problem answer must appear on a separate page. Mark the top of each page with your group member names, the course number (6.857), the problem set number and question, and the date. We have provided templates for LaTeX and Microsoft Word on the course website (see the *Resources* page).

Grading and Late Policy: Each problem is worth 10 points. Late homework will not be accepted without prior approval. Homework should not be submitted by email except with prior approval. (*Somebody* from your group should be in class on the day that the homework is due.)

With the authors' permission, we will distribute our favorite solution to each problem as the "official" solution—this is your chance to become famous! If you do not wish for your homework to be used as an official solution, or if you wish that it only be used anonymously, please note this on your homework.

Problem 5-1. Elliptic Curve Digital Signature Algorithm

For this problem you will implement an Elliptic Curve (EC) analogue of the Digital Signature Algorithm (DSA), so you will be working in an elliptic curve group $E(Z_p)$. Note that for historical reasons the group operation for an elliptic curve has been called addition, so for instance the discrete log problem is: given $P \in E(Z_p)$ and $Q = aP$, find a.

The algorithms for ECDSA are given below (taken from Johnson and Menezes, "Elliptic Curve DSA: An Enhanced DSA").

ECDSA key generation.

1. Select an elliptic curve E defined over Z_p. The number of points in $E(Z_p)$ should be divisible by a large prime n.
2. Select a point $P \in E(Z_p)$ of order n.
3. Select a statistically unique and unpredictable integer d in the interval $[1, n-1]$.
4. Compute $Q = dP$.
5. output the public key (E, P, n, Q), and secret key d.

ECDSA Signature generation. To sign a message m:

1. Select a statistically unique and unpredictable integer k in the interval $[1, n-1]$.
2. Compute $kP = (x_1, y_1)$ and $r = x_1 \bmod n$. If $r = 0$, then go to step 1.
3. Compute $k^{-1} \bmod n$.
4. Compute $s = k^{-1}\{h(m) + dr\} \bmod n$, where h is the Secure Hash Algorithm (SHA-1).
5. If $s = 0$, then go to step one.
6. Output the signature (r, s) for the message m.

ECDSA Signature verification. To verify a signature (r, s) on a message m:

1. Obtain the public key (E, P, n, Q).

2.Verify that r and s are integers in the interval $[1, n-1]$.

3.Compute $w = s^{-1} \bmod n$ and $h(m)$.

4.Compute $u_1 = h(m)w \bmod n$ and $u_2 = rw \bmod n$.

5.Compute $u_1 P + u_2 Q = (x_0, y_0)$ and $v = x_0 \bmod n$.

6.Accept the signature if and only if $v = r$.

In the above key generation algorithm, we need to calculate the number of points on an elliptic curve. Although this can be computed in polynomial time, it is a bit messy, so we will follow the guideline of NIST (FIPS PUB 186-2, "Digital Signature Standard (DSS)"), and use one of its suggested curves, curve P-192 (page 29). Note that the group size (called r by NIST) is prime for P-192. Also note that NIST calls the base point P by the name G (with components G_x and G_y).

(a) Read Appendix 6.4: Generation of pseudo-random curves(prime case), page 63 of the NIST document. Argue that the choice of a and b satisfies $4a^3 + 27b^2 \not\equiv 0 \bmod p$.

(b) Implement the key generation (with E and P as suggested by NIST), signature generation and verification algorithms using Python. Turn in your code.

(c) Using your implementation, generate a signature on some message m and verify that the signature is accepted. Turn in your message and the signature generated.

Problem 5-2. Evil Key Generators

Alice has been hired by BitDiddle Software to write the RSA key generation routines for their new software. Alice's routine will take as input a number k (e.g. $k = 1024$) and produce a k-bit integer n that is the product of two primes p, q of approximately $k/2$ bits each. Alice's routine will also produce values e and d that are multiplicative inverses of each other modulo $\phi(n)$. Alice's software will be used by BitDiddle's customers to produces their RSA keypairs. (Of course, Alice's software also has available a good source of random bits.)

Unknown to the higher-ups at BitDiddle, however, Alice has been recently corrupted by BitDiddle's competitor, BitTwiddle (Bob, who runs BitTwiddle, has offered Alice some nicely developed beach property in Second Life if Alice cooperates...)

Bob asks Alice to "fix things" so that BitDiddle's key generation routine produces "weak" RSA keys that Bob can easily "break".

Alice thinks of three approaches.

1. Use a "crippled" random number generator that can only produce 10^9 different primes p or q. (Thus, there are about 10^{18} different moduli n that could be produced; p and q are produced independently with this crippled generator.) Bob can build a table of the possible primes, and easily figure out which ones divide the public n of each BitDiddle customer.
2. Alice's software picks a value of e randomly, and then uses that as a seed for a pseudorandom number generator to generate the starting values in the search for the primes p and q. (If e turns out not to be relatively prime to $\phi(n)$, the process is restarted.) Bob knows the code for the pseudorandom generator, which is embedded in Alice's software, and so can regenerate the primes himself once he sees a customer's e.
3. Alice's software includes Bob's public RSA key PK_B. When a BitDiddle customer requests a public key, Alice's software generates primes p and q randomly as usual, and then computes $e = E(PK_B, p)$. That is, the customer's e is just the encryption of the customer's p, under Bob's public RSA key. (A minor point: if this number e is not relatively prime to $(p-1)(q-1)$, the process is restarted.)

Answer each of the following problem parts.

(a) Discuss the possible ways for Alice's treachery to become discovered with these three methods.

(b) Discuss the difference in consequences for the these methods, should Alice's treachery become publicly known.

(c) Discuss methods that BitDiddle should use to prevent and/or detect this sort of treachery on the part of its employees or contractors. (Specifically, we are asking here about detecting deliberately weak cryptographic key generators.)

(d) Much to her consternation, once Alice starts work, she is told that her RSA generation routine must always produce $e = 2^{16} + 1$ as the RSA encryption exponent. Can you handle this requirement in such a way that your result has properties similar to those of approaches (2) or (3) above. In other words, can you come up with a weak key generation scheme that works even for a fixed e and that does not, like scheme (1), work simply by drastically reducing the space of possible n values? Sketch your approach briefly, and describe its properties.

Problem 5-3. El Gamal Variants

Alice and Bob are trying to find a simple way to improve El Gamal so that it is $\mathsf{IND-CCA2}$ secure. They come up with the following seven proposals. For each proposal, try to evaluate the security of the proposal. Is it semantically secure? Is it $\mathsf{IND-CCA2}$ secure? Is it neither? Give your reasoning or arguments when you can. Be brief. Don't give formal proofs; this isn't a theory class. Feel free to say "my best guess is that..." if you get stuck...

All of these schemes work, like El Gamal, in a group G of prime order q with generator g. Assume that DDH holds in G.

In all of these schemes, Alice has a secret key x drawn randomly from Z_q, and a public key defined by $y = g^x$.

For the last three proposals, Alice also has a second secret key w drawn randomly from Z_q, and a second public key z defined by $z = g^w$.

A hash function H is also available that maps elements of G into fixed-length strings. (For part (e), assume as well that elements in the range of H can be interpreted in some natural manner as elements of G.) For the purposes of this problem, H can be modeled as a "random oracle".

In all seven proposals, the encryption c of a message m consists of three parts.

The first two parts are the usual El Gamal encryption components: g^r and my^r , where r is randomly chosen from Z_q.

The third part varies from proposal to proposal. But the mode of operation is always the same: during decryption, the message m is obtained as usual for El Gamal from the first two components, and then the correctness of the third part is checked. If it is not correct, decryption fails, and decryption produces no output. Otherwise the message m is output.

(a) $c = (g^r, my^r, H(m))$

(b) $c = (g^r, my^r, H(y^r))$

(c) $c = (g^r, my^r, H(g^r))$

(d) $c = (g^r, my^r, mz^r))$

(e) $c = (g^r, my^r, H(m)z^r)$

(f) $c = (g^r, my^r, H(mz^r))$

(g) $c = (g^r, my^r, H(m||z^r))$