Problem Set 2

This problem set is due on *Thursday, September 27, 2001* at the beginning of class. Late homeworks will *not* be accepted. You are to work on this problem set in groups of three or four people. If any of your group members leave the class, contact the TAs immediately to form a new group.

Mark the top of each sheet with your names, 6.857, the problem set number and question, and the date. **Type up your solutions** and be clear. Each problem should begin on a new sheet of paper. That is, you will turn in each problem in a separate pile of paper.

Problem 2-1. Primes

(a)

Write an efficient program (i.e. probabilistic polynomial time) that takes as input a string s and produces a number likely to be a prime and whose high order bytes are s. The input s is a byte string, and the output number should be printed in decimal.

For example, the string "rivest+abhi+fubob" is

```
0x72 69 76 65 73 74 2B 61 62 68 69 2B 66 75 62 6F 62 00 00
```

when each ASCII character is converted to its hexadecimal representation[1] and two zero bytes are appended. The two extra zero bytes provide the flexibility needed to modify this number into a prime without having to change the original message. In decimal form, our starting number is 2551472030826120444243689867576938224490250240. By flipping some of the bits in the last two bytes of this number, we can generate the following number which is likely prime: 2551472030826120444243689867576938224490250261.

You may use any programming language to implement this program. However, you CANNOT use any built-in methods for primality testing (e.g., `java.math.BigInteger.isProbablePrime()`). You must implement your own primality testing code. We recommend that you either use Java or C. Java comes with a package for arbitrarily large integers (`java.math.BigInteger`). You might find the GNU Multi-Precision (GMP) library useful for programming in C[2]. Submit the source code to your program. We have provided a minimal framework in Java to get you started. Check the 6.857 Web page for this code.

See `http://www.theregister.co.uk/content/6/17681.html` for a prime number that represented the source code to break DeCSS[3].

(b) Use your program to encode the email addresses of your group in a number that is likely prime. Email the output of your program to `6.857-staff@mit.edu` with the subject "My Special Message In a Bottle."

If you want to have some fun on your own time, write a program that outputs a prime that when decoded is the source code of the program to create the prime. This is completely optional and just for fun, not to be graded.

[1]Type "man ascii" for help on Athena
[2]`http://www.swox.com/gmp/`
[3]This prime number might be illegal to distribute because of the Digital Millennium Copyright Act.

Problem 2-2. Reuse, reduce, collide

Let $g : \{0,1\}^* \to \{0,1\}^n$ be a hash function mapping arbitrarily long binary strings to n-bit strings. Define the hash function $h(x) = g(g(x))$.

(a) Argue that if g is one-way (pre-image resistant) then h is one-way.

(b) Argue that if g is collision-resistant, then h is collision-resistant.

(c) Argue that if g is weakly collision-resistant (i.e. second preimage-resistant), then h need not be weakly collision-resistant. Although a counter-example would be splendid, it is sufficient to sketch how it might be possible to have a counter-example.

Problem 2-3. Voting

Evaluate the following voting proposal. Note advantages and deficiencies and suggest improvements or remedies as appropriate. Or, if you think the scheme can not be improved sufficiently to be usable, say why. Limit your discussion to one page.

The election officials for a county with n voters generate n random RSA keys and place each public/private key pair on a separate floppy disk. The officials also keep a record of $h(PK)$ for each public key PK, for some suitable hash function h. Each floppy has a random two-digit number R stuck on it with a post-it note; this number is also recorded with the corresponding $h(PK)$. During the month before the election, each voter goes to the county courthouse, identifies and authenticates herself, and grabs a random floppy from the box of floppies. (The election official should not be able to correlate the voter with the floppy he took.) The voter memorizes the two-digit number and discards the post-it note before leaving with the floppy.

The voter creates her ballot B on her home computer (or, if the voter does not have a PC, on one at the local high school). He uses the private key SK from the floppy to sign the pair (B, R), and sends B, R, PK, and the signature through an anonymizing email channel over the Internet to the county courthouse.

The county computer receives everything, computes $h(PK)$, confirms it is on the list, checks that R is correct, verifies the digital signature, and if everything checks, adds (B, R, PK) to a list of "validated ballots." When the election is over, the validated ballots are tallied.