Problem Set 5

This problem set is due on *Thursday, October 21, 1999* at the end of class. Late homeworks will *not* be accepted. You are to work on this problem set in groups of three to four people. (Be sure that you all understand the solutions turned in to each problem.)

Mark the top of each sheet with your names, 6.857, the problem set number and question, and the date. **Type up your solutions**[1], and be clear. Points may be deducted if your TA has problems understanding your solution.

Problem 5-1. Unconditional commitments

A *commitment scheme* consists of two protocols: a *commit phase* and a *reveal phase*. Alice begins with a value x that she wishes to commit to Bob. During the commit phase, she sends information to Bob that represents her commitment. This information should tell Bob nothing about the value of x. During the reveal phase, Alice sends additional information to Bob that allows Bob to know the value of x. (Typically, but not necessarily, Alice just sends Bob the value of x itself during the reveal phase.) These protocols have the property that once the commit phase is over, Alice is unable to change the value of x committed to. That is, there is only one value of x that Alice can cause to be revealed during the reveal phase. If she tries to reveal some other value, Bob will notice and will accuse Alice of cheating.

A typical application would be in an sealed-bid auction, where Alice first submits her sealed bid (the commit phase), and later allows Bob the auctioneer to open her bid (the reveal phase). Alice can't cheat by changing her bid after the commit phase.

When we are allowed to make assumptions about computational difficulty, Alice can commit to x by sending Bob a value $y = f(x)$, where f is a suitable public "one-way collision-resistant" function such that Bob can't invert f on y to obtain x, and Alice can't find two values x, x' that map to the same y. The computational assumption is that such functions exist. During the reveal phase Alice merely reveals x.

However, in this problem we are not allowing you to make such convenient computational assumptions. We want an "unconditionally secure" commitment scheme.

Design an unconditionally secure commitment scheme that utilizes a "trusted third party"–Ted. Ted is honest, and both Alice and Bob trust that he will execute his role correctly. Alice and Bob can each communicate privately with Ted, and with each other. (Assume that there are secure phone lines between each two of the three parties.) Your scheme should explain carefully how the commitment phase works, and how the reveal phase works.

You should, if possible, keep Ted from ever finding out the value of x. (Alice and Bob both trust him to be honest, but the value of x is none of Ted's business, and Alice and Bob prefer that he never learns x.) So the "obvious" solution wherein Alice gives x to Ted during the commit phase, and Ted gives x to Bob during the reveal phase, is not a workable solution.

[1] If you latex your solutions, include a URL or AFS path where I can find the latex file(s); I will distribute my favorite solution for each problem to the class as the 'official' solution – this is your chance to become famous! Note that if I choose your solution, then you get an extra point on that problem.

As stated, this problem is not so hard. But if this is far as you can get, please turn in your solution to the problem as stated so far.

See if you can produce a high-quality solution in which Ted is only active during the commit phase, and has no role to play during the reveal phase. Best of all would be a scheme where Ted only has to do anything during an initial "setup phase" that precedes the commit phase, and has nothing to do during either the commit or reveal phases. In such a scheme, Ted is done before Alice has even decided which x to commit to. Can you find such a scheme?

Your solution(s) should be unconditionally secure: Bob should not be able to guess x better than chance after the commit phase, even if he has unlimited computing power. And the reveal phase should not cause Bob to obtain a value of x that is different than Alice's original x with more than a negligible probability, even if Alice has unlimited computing power.

Problem 5-2. Finding t-bags

A *"t-bag"* is a sequence of t consecutive integers of the form, for some $t \geq 1$:

$$p_1, 2p_2, 3p_3, 4p_4, \ldots, tp_t \ ,$$

where each p_i is a prime number. For example,

$$3, 2 * 2$$

is the first 2-bag,

$$13, 2 * 7, 3 * 5$$

is the first 3-bag, and

$$12721, 2 * 6361, 3 * 4241, 4 * 3181$$

is the first 4-bag.

The notion of a t-bag was invented for this problem—as far as we know there is no literature on this (yet–go to it!). In particular, we do not know if there are t-bags for arbitrarily large values of t, nor do we know if there are arbitarily large t-bags for small fixed values of t.

Do *one* of the following:

- Find a t-bag for the largest value of t that you can.
- Find the largest 4-bag that you can.

Turn in your code with your solution. No other writeup is necessary. However, you must code up yourself any necessary primality-testing routines, rather than using one from an available library.

(The best solvers will be encouraged to submit a note to *American Math Monthly* for publication!)

Problem 5-3. Inetd DoS vulnerabilities

In Unix, the Internet Daemon (inetd) provides the "handshaking" that goes on when a TCP/IP connection is made in support of a service such as telnet, ssh, or sendmail.

Inetd is typically susceptible to a denial of service (DoS) attack involving many consecutive connects to the same service; this is sometimes called a "flooding" attack. When too many connections are made within a specified short period of time, inetd will terminate that service for a short period of time (typically five minutes), and print an error message on the system error log of the form:

```
inetd[345]: telnet/tcp server failing (looping), service terminated
```

There are attack programs, such as octopus, which are capable of opening and maintaining hundreds or even thousands of connections at once. Octopus will reliably terminate services that inetd controls, and for standalone daemons it can be used to overflow the target's process table or file descriptors. Indeed, it's particularly effective against sshd and sendmail, because they fail to exit cleanly when the descriptor table is full or nearly full.

Consider the tradeoffs that the designers of inetd weighed when they decided to implement inetd the way they did. What are some alternative designs that solve this DoS vulnerability? Do they introduce new vulnerabilities?

Describe a solution which addresses services offered from inetd as well as standalone daemons. This solution should provide a more robust system, making sure to support easy installation onto current machines. Describe and analyze the tradeoffs involved with upgrading to your design.

Some interesting reading:

- octopus `http://www.hoobie.net/security/exploits/hacking/octopus.c`
- xinetd `http://synack.net/xinetd/`
- SYN cookie summary `http://www.8lgm.org/Exploits/TCP/SYN_flood/default.htm`
- eyenetdee `ftp://ftp.technotronic.com/denial/eyenetdee.c`
- Abacus Project `http://www.psionic.com/abacus/`