Problem Set 7

Due by 5:30 pm on Monday, April 25, 2005.

Note: An automatic penalty-free 2-day extension will be granted to all who request it. However, no work will be accepted after the end of reading period, with or without late penalty, except when authorized by the dean.

Problem 21: (Oblivious Transfer)

Oblivious Transfer is a two party protocol $(A(b), B)$ such that at the end of this protocol one of the following two events occurs, each with probability $1/2$:

(a) B learns the value b.

(b) B gains no information about b beyond what, if anything, B knew about b before the protocol.

At the end of the protocol, B knows which of the two events occurred, and A has no idea which event occurred.

One-out-of-two Oblivious Transfer is a two party protocol $(A(b_0, b_1), B(s))$, such that at the end of this protocol, all of the following three conditions hold:

(a) B learns the value b_s.

(b) B gains no information about b_t beyond what, if anything, B knew about b_t before the protocol, where $t = 1 - s$.

(c) A learns nothing about s.

Here is an implementation of one-out-of-two oblivious transfer using a basic oblivious transfer primitive as a black box. (This protocol is adapted from one in lecture notes by Rafail Ostrovsky, `http://www.cs.ucla.edu/~rafail/TEACHING/WINTER-2005/L10/L10.pdf`.)

1. Let n be a security parameter, and set $M = 3n$. A chooses a random bit string $r = r_1 r_2 \ldots r_M$ of length M. A uses the basic oblivious transfer protocol M times to transfer r to B, one bit at a time. B learns approximately 1/2 of the bits r_i. Let $I \subseteq \{1, \ldots, M\}$ be the set of indices i for which B does learn r_i.

2. B's input bit is s. B wants to learn A's secret b_s. B chooses a random subset I_s of I of size n and a random subset I_{1-s} of $\{1, \ldots, M\} - I$, also of size n, and sends the sets I_0 and I_1 to A.

3. A checks that I_0, I_1 are disjoint subsets of the right form. A then computes $c_i = b_i \oplus \left(\bigoplus_{j \in I_i} r_j\right)$, for $i = 0, 1$, and sends c_0, c_1 to B.

4. B computes $b_s = c_s \oplus \left(\bigoplus_{j \in I_s} r_j\right)$.

Questions:

(a) This protocol can sometimes fail. Explain how.

(b) The above definition of one-out-of-two oblivious transfer does not allow for failure. Make a minor change to the definition so that it matches what this protocol is actually able to achieve.

(c) Describe why B learns the desired value b_s. Is this always true or only true with high probability?

(d) Describe why B gains no information about b_{1-s}. Is this always true or only true with high probability?

(e) Describe why A learns nothing about s. Is this always true or only true with high probability?

(f) Describe why a cheating B cannot learn both b_0 and b_1. Is this always true or only true with high probability?

(g) Why does Alice need to check I_0 and I_1 in step (3)? Explain how B could cheat if she failed to do so.

(h) Does the protocol still work if M is defined to be $2n$ instead of $3n$? Defined to be $5n$ instead of $3n$? Explain.

The next two problems concern the Blum-Blum-Shub pseudorandom sequence generator. See Handout 18 for the exact definitions assumed by these problems.

Problem 22: (BBS Pseudorandom Sequence Generator)

Write a C function to implement the Blum-Blum-Shub pseudorandom sequence generator. You can assume the inputs to your programs are numbers at most 15 bits long (so they are short enough to fit into a variable of type `short int`).

Your function should have the prototype

```
short int bbs_random( short int len,
                      short int buf[],
                      short int seed,
                      short int n );
```

`buf` is assumed to be a buffer of length `len`, `seed` is the seed (starting value) for the BBS generator, and `n` is the modulus for the BBS generator. You may assume that `seed` is in $\mathbf{Z}^*_{\mathtt{n}}$ and that `n` is a Blum integer. A call to `bbs_random()` generates `len` pseudorandom bits and places them in `buf[0]`, ..., `buf[len-1]`, one bit per array element. The new seed is returned.

To test your function, write a command `bbs` that calls `bbs_random()`. The command line "`bbs len seed n`" generates `len` bits starting from seed `seed` and modulus `n` and prints three lines of output. The first line echos the command line arguments. The second contains the pseudorandom bit sequence, printed as a sequence of 0's and 1's with no intervening spaces. The third contains the new seed, printed in decimal.

Run your command on the arguments `80 3 13589`. (Note that $13589 = 107 \times 127$ is a Blum integer.) Write your answers to a file called `bbsout.txt` and submit both the program and the answers file.

Problem 23: (Cycle Lengths)

The purpose of this problem is to explore the cycle lengths of the various possible seeds in the BBS generator of problem 22. For any seed $s_0 \in \mathbf{Z}_n^*$, define the *cycle length* of s_0 to be $k-1$, where k is the least integer > 1 such that $s_k = s_1$ in the BBS-generated sequence $s_1, s_2, s_3, \ldots$, where $s_i = s_{i-1}^2 \bmod n$, for $i = 1, 2, 3, \ldots$.

Questions:

(a) Why is the cycle length well defined for every $s_0 \in \mathbf{Z}_n^*$? That is, why does s_1 occur in the sequence $s_2, s_3, s_4, \ldots$?

(b) What is the expected cycle length when s_0 is chosen uniformly at random from $\mathbf{Z}_n^*$, where $n = 13589 = 107 \times 127$.

For part (b), you should write a program to build a table of quadradic residues and the cycles they lie in. Then compute a table of cycles and their lengths. Finally, compute the expected cycle length. For example, for $n = 33 = 3 \times 11$, there are 5 quadratic residues, so the table of quadratic residues and the table of cycles might look as follows:

x	$(x^2 \bmod 33)$	cycle #
1	1	1
4	16	2
16	25	2
25	31	2
31	4	2

cycle #	length
1	1
2	4

From this table, we see that there are only two cycles: (1) and $(4, 16, 25, 31)$, Of the 20 possible seeds in $\mathbf{Z}_{33}^*$, 4 lead to the first cycle and 16 lead to the second cycle. Hence, the expected cycle length is

$$\frac{4}{20} \times 1 + \frac{16}{20} \times 4 = \frac{68}{20} = 3.4$$