The University of North Carolina at Greensboro
CSC 589: Trusted Computing and Security Models
Prof. Stephen R. Tate

Handout 2
January 24, 2008

Assignment 1: Due Thursday, February 7

Objectives: There are two objectives with this assignment: To start you thinking about models of security and to give you some practice using LaTeX in writing mathematical material. Therefore, solutions should be prepared using LaTeX!

1. Write up the coin-flipping protocol from Bellare/Rogaway Section 1.2.3 in your own words, using a step-by-step presentation (use an `enumerate` environment in LaTeX). Next, make a table of some values of p, q, and N that could be used by Alice to commit to a 0 and to a 1 (at least two examples for each value). Obviously, to be able to write this out you'll want to use values that are much smaller than the 500-bit primes mentioned in the notes, but this is simply to illustrate the process, not to give values that would provide any security. Your table should look something like this:

Bit Committed	p	q	N
0	...	...	...
0	...	...	...
1	...	...	...
1	...	...	...

2. The most fundamental notion of computational complexity used in discussing the security of cryptographic protocols is the notion of polynomial-time algorithms. Specifically, we would like for our protocols to be such that no polynomial-time algorithm can break them. A key assumption for many public-key algorithms is that there is no polynomial-time algorithm for factoring — in other words, an algorithm that can take the product of two large prime numbers ($N = pq$) as input and produce the factors p and q.

   Consider the basic algorithm of trying all integers from $2, \ldots, N-1$ as possible divisors and testing them all using a trial division. This obviously is a correct factoring algorithm, but what is the complexity of the algorithm? Carefully justify your analysis, and clearly justify your answer to the basic question: Is this a polynomial time algorithm?

3. (*Graduate Students Only*) Answer this question with as much detail and justification as you can: a pseudorandom number generator (a "PRNG" — described in Section 1.2.1) should be one-way in the sense that if you are given b bits of output you cannot (in polynomial time) compute the seed used by the PRNG. What does this mean as far the size of the seed? Can it be a constant number of bits (unrelated to b)? Could it be $\Theta(\log b)$ bits? What is the right restriction on the size?