Massachusetts Institute of Technology
6.857: Network and Computer Security
Professor Ronald L. Rivest

Handout 16
October 3, 2002

Problem Set 4

Note the special due date because of the upcoming quiz. This problem set is due on *Thursday, October 17, 2002* at the beginning of class. Late homeworks will *not* be accepted. Problem 1 is a group problem. Problem 2 is an individual assignment.

At the end of each problem, tell us how each person contributed (writing, designing, coding, proof reading, moral support, etc).

Problem 4-1. Uniformly selected primes and factorization of $p-1$

In lecture, you learned that it's often necessary to produce a prime p and a generator g of Z_p^* in public key cryptosystems. We do not, however, know how to find a generator of Z_p^* without also knowing the factorization of $p-1$. In problem set 3, you generated co-Sophie-Germain primes of the form $p = 2q+1$ where $p-1$ has the prime factors 2 and q. While it's not proven whether there are an infinite number of Sophie-Germain primes, it's widely believed that the probability of p and q being prime in $p = 2q+1$ is independent. There are other ways to generate a prime p and the prime factors of $p-1$. This problem will explore this exciting realm!

Ben Bitdiddle is the director of the factory floor at PRIMES Computers, Inc. where they produce primes p and the factorization of $p-1$. They currently produce co-Sophie-Germain primes since the algorithm is straightforward. The CEO asked Ben for assistance on computing the the next quarter's financial estimates. Because PRIMES Computers, Inc. makes money by selling special primes, it's important to have many primes to sell. That night, Ben had a vision. He saw a future where the number of Sophie-Germain primes is finite! Yikes. This is contrary to what most smart cryptographers believe. Nevertheless, Ben takes drastic measures to improve the quarterly outlook. He decides to take market factors into consideration by diversifying his prime portfolio. Ben looks for a different algorithm to produce a prime p and the factorization of $p-1$. He turns to the following algorithm and experts at the Michigan Institute for Trucking (MIT).

Adam Kalai explains a relatively new and simple method for efficiently generating uniformly a random number r (not necessarily prime) along with its prime factorization.[1]

Input: Integer $n > 0$.
Output: A random number r and its prime factorization where r is selected uniformly between 1 and n inclusively ($1 \leq r \leq n$):

1. Generate a sequence $n \geq s_1 \geq s_2 \geq \ldots \geq s_l = 1$ by choosing $s_1 \in \{1, 2, \ldots, n\}$ and $s_{i+1} \in \{1, 2, \ldots, s_i\}$, until reaching 1.
2. Let r be the product of the prime s_i's.
3. If $r \leq n$, output r and the set of s_i's with probability r/n.
4. Otherwise, restart by going to step 1.

[1] The complete analysis of the algorithm appears on `http://www.mit.edu/~akalai/factor/factor.html`.

(a) Implement the random number generator

Being a manager, Ben turns to you to implement the scheme. Can you save your company from a disappointing financial quarter?!

Implement the uniform random number generator described above. Now layer a prime number generator on top of this algorithm. That is, write a new prime number generator that uses the above algorithm as a subroutine. Your new generator takes as input a bit size b and outputs a prime p and the prime factors of $p-1$ such that $2^{b-1} \leq p \leq 2^b - 1$ and $p-1$ is a uniformly random number between $2^{b-1} - 1$ and $2^b - 2$ inclusive. You can reuse code from problem set 3. Submit your pretty-printed code.

(b) Compare to your co-Sophie-Germain prime generator

Ben is also concerned about the running time of the new prime number generator. Calm his nerves by presenting the performance analysis described below.

You have now implemented a more general prime number generator than the co-Sophie-Germain generator. There are possibly several factors of $p-1$ instead of just two factors.

If a call to your primality testing code constitutes a single unit of work, (1) how many units of work do you expect to exert to find a 512-bit prime with your new prime generator? (2) How many units of work do you expect to exert to find a 512-bit prime with your old co-Sophie-Germain prime generator?

(c) Generate 'em

Generate a 512-bit prime using both your new algorithm and your old algorithm. (1) What are the two primes p_1 and p_2 and the prime factors of $p_1 - 1$ and $p_2 - 1$? (2) Give the actual number of calls you made to the primality testing function for both algorithms.

(d) The probability of generating a weak key

We do not know of an efficient algorithm to take discrete logs in the general case, but the DLP becomes easy when there are no large factors of $p-1$. Therefore, it is important to have at least one large factor in $p-1$ for strong security against known DLP attacks.

Let $|p-1| = |n| = 1{,}024$ bits. (1) What is the probability that all the prime factors q_i of $p-1$ are $\leq 2^{160}$? In other words, what's the probability that an adversary's work factor to take a discrete log is less than 2^{80}?

Here are some helpful facts that you can assume in your argument:

We use Dixon's theorem to justify the following approximation. If a number n is generated at "random," the probability that all the prime factors q_i satisfy $\frac{\lg n}{\lg q_i} \geq u$ is approximately u^{-u}.

The DLP is solvable in time $\approx \sqrt{q_k}$ where q_k is the largest prime factor of $p-1$.

Problem 4-2. How secure is your operating system?

This is an individual assignment.

Review the recently released FBI/SANS "top 20" vulnerabilities on `http://www.sans.org/top20/#W9` and determine whether or not your personal machine is vulnerable to any of them. Give us a 1–2 page summary of your findings. Do as many as you can, but don't spend more than 4 hours on this problem. Be kind to the network in any testing you do. Please only test your own machines.