Prof. Friedrich Eisenbrand
Nicolai Hähnle

Discussions from: March 22, 2011

Computer Algebra

Spring 2011

Assignment Sheet 3

Exercises marked with a $\star$ can be handed in for bonus points. Due date is April 5.

Exercise 1

Determine the remainder that one gets when dividing $2^{37\,500\,120\,314\,007\,842\,499}$ by 101.

Exercise 2 ($\star$)

Let $N = pq$, where $p \neq q$ are primes. Show that given only N and $\varphi(N)$, one can compute the prime factors p and q efficiently.

Hint: You may assume that roots of a polynomial can be computed efficiently.

Exercise 3

We say that $x \in \mathbb{Z}_N^\star$ is a *Fermat liar* if N is composite and $x^{N-1} \equiv 1 \pmod{N}$. Show that if p and $2p-1$ are both prime and $N = p(2p-1)$, then exactly half of the elements of $\mathbb{Z}_N^\star$ are Fermat liars.

Hint: Show that $x \in \mathbb{Z}_N^\star$ is a Fermat liar if and only if it is a square modulo $2p-1$.

Exercise 4

Let $N = p^k$ where p is prime and $k \geq 2$. Show that N is not a Carmichael number.

Exercise 5

Suppose you are given $N = pq$, where $p \neq q$ are primes, and $x \in \mathbb{Z}_N^\star$ such that $x \equiv 1 \pmod{p}$ and $x \not\equiv 1 \pmod{q}$. Show how to compute p and q efficiently.

Exercise 6

Let $N = pq$, where $p \neq q$ are primes, and let $e \neq d$ be natural numbers such that $ed \equiv 1 \pmod{\varphi(N)}$. Show that given only N, e, and d, one can efficiently compute the prime factorization of N.

Note: In terms of the security of the RSA cryptosystem, this implies that computing the private key from the public key is computationally as hard as factoring N.

Hint: This is a possible approach to the problem:

1. Analyze $ed-1$. Show that $x^{ed-1} \equiv 1 \pmod{N}$ for all $x \in \mathbb{Z}_N^\star$.

2. Pick a random $x \in \mathbb{Z}_N^\star$ and take successive powers as in the strong primality test.

3. Show that you can apply Exercise 5 with significant probability, by analyzing the group structure of $\mathbb{Z}_N^\star$ and how the sequence of successive powers (or the probability distribution of the successive powers) evolves in that structure in detail.

Exercise 7 ($\star$)
Download the accompanying source code `assignment03.py`. Implement

1. the Extended Euclidean algorithm (function `exgcd`),

2. fast modular exponentiation (function `modexp`),

3. the strong primality test (function `primalitytest_single`), and

4. basic RSA key generation (function `rsa_generate_key`).

Optionally implement the "attack" described in Exercise 6 (this is roughly the same amount of code as the strong primality test).