Institut for Matematik og Datalogi
Syddansk Universitet

November 16, 2011
JFB

Introduction to Computer Science
E11 – Lecture 15

Lecture, November 15, 8:15–10, U26

There was an introduction to the symbolic computation programming language Maple. We also continued with cryptography, concentrating on RSA.

Lecture, November 22, 8:15–10, U26

We will finish with cryptography, concentrating on the implementation of RSA (including modulo exponentiation), and then finish with security.

Lecture, November 24, 12:15–14, U26

We will cover sections 6.6 and 6.7 of chapter 6.

Laboratory: November 30 and December 2 - Terminal Room

Discuss the following problems in groups of two (or possibly three). Note that two labs (four hourse) are allocated to this. Read about all parts of this lab before beginning. Discuss in your study groups which hash functions would be interesting to implement and what results you expect.

In your study groups, try some examples of modulo exponentiation and make sure you all understand the running time analysis.

1. Design three hash functions and program them in Python or Java (or Maple). (You could consider letting SHA-1 be one of them. There is pseudocode for SHA-1 on Wikipedia. What does the "0x" in front of the various constants mean?)

2. Test your hash functions. Create an array of length $m + 1$ which will count how many random large integers are hashed to each of the distinct values between zero and m. Look at the results for some relatively small values of m (for example, $m = 4, 8, 12, 20, 50, 100$). (Note that SHA-1 has a fixed length output, so to get the values in the range zero to m, you have to take the output modulo $m + 1$.)

3. With at least four different values of m and some different length large strings (integers) which are hashed, check how many values you need to hash before you get a collision. Does it behave as one would predict with the Birthday Paradox?

4. With at least four different values of m and some different length large strings, check how even your distribution is? What is the difference between the largest number of strings hashing to any particular value and the smallest number? How does this seem to depend on the number of strings you hash?

5. With your hash functions, how would you find collisions (two strings that hash to the same value) when m is very large (say 2^{80})? Are your hash functions cryptographically secure? Recall that a hash function is cryptographically insecure if one can find even one collision efficiently.