Institut for Matematik og Datalogi
Syddansk Universitet

April 4, 2013
JFB

Cryptology – F13 – Week 7

Lecture, March 8

We covered the McEliece Cryptosystem. Then we introduced digital signatures from chapter 7, and started on chapter 4, covering through section 4.2. A meet-in-the middle attack was demonstrated.

Lecture, March 14

We finished chapter 4, skipping the subsection on the Merkle–Damgård construction. We also covered SHA-3, Keccak. See the course homepage for Keccak's specification. We covered up through section 7.2 of chapter 7 and introduced the ElGamal signature scheme.

Lecture, April 8 – note at 12:15, not 10:15

We will finish through section 7.4.2 of chapter 7. We will discuss subliminal channels and then begin on chapter 8.

Lecture, April 12

We will finish chapter 8, and possibly return to section 7.6, which we will do from some notes.

Problem session April 11

We will continue first with those we didn't finish on March 15, listed below. We will also look at the problems from the last assignment.

1. Do problems 6.20 (work in the multiplicative group modulo 1103), and 6.22a in the textbook.

2. In class we have discussed the discrete logarithm problem modulo a prime, which means that we have discussed them over fields of prime order. There are also finite fields of prime power order, so for any prime p and any exponent $e \geq 1$, there is a field with $q = p^e$ elements, $GF(q)$. The elements of such a field can be represented by polynomials over $GF(p)$ of degree no more than $e-1$. The operations can be performed by working modulo an irreducible polynomial of degree e. For example, $y = x + x^5 + x^7$ is an element of the field $GF(2^{10})$, represented by $GF(2)[x]/(x^{10} + x^3 + 1)$. One can calculate a representation for y^2, by squaring y and then computing the result modulo $x^{10} + x^3 + 1$, so one gets $x^2 + 2x^6 + 2x^8 + x^{10} + 2x^{12} + x^{14} \pmod{x^{10} + x^3 + 1} = 1 + x^2 + x^3 + x^4 + x^7$. In Maple, you can use the `powmod` function to do these calculations.

   Try raising y to the powers $e \in \{33, 93, 341, 1023\}$ to see what result you get. What do you get? What does this prove about y?

3. On my computer using Mathematica (last time I tried), raising to the power 1023 directly failed due to lack of memory. What does this say about how Mathematica did the calculations? What can you do to get around this problem when you try these calculations? (Maple has no problems with these calculations.)

4. Why would there be a preference for working in $GF(2^k)$ for some large k, rather than modulo a prime for some very large prime? Hint: think about how arithmetic is performed.

5. Do problem 4.1 in the textbook. For part (d), use the fact that the left-hand side in (c) is at least zero.

6. Do problem 4.6.

7. Do problem 4.12. For part (b), you can find a (1,1)-forger. Skip the difficult case mentioned.

8. Let p be an odd prime and g_0 and g_1 be generators of Z_p^*. Consider the following two functions: $f_0(x) = g_0^x \pmod{p}$ and $f_1(x) = g_1^x \pmod{p}$. Use these two functions to create a hash function which will hash an arbitrary length message down to a value in Z_p^*. Can you make it secure under the assumption that the discrete log problem is infeasible?

9. Do problem 6.21 in the textbook.

10. Do problem 7.1 in the textbook. (You might want to look at the notes on the course home page on number theory to recall how to solve linear congruences.)

11. In the discussion of the Schnorr signature scheme on page 293, it says tha t to find a qth root of 1 modulo p, one should begin with a primitive element α_0 of Z_p^* and compute $\alpha_0^{(p-1)/q}$. (Recall that p and q are both primes.)

    a. Why is this correct? What subgroup does the result generate?

    b. How long does it take to do this computation?

    c. Is it necessary that α_0 be a primitive element?

Assignment due Monday, April 29, 12:15

Note that this is part of your exam project, so it must be approved in order for you to take the exam in June, and you may not work with others not in your group. If it is late, it will not be accepted (though it could become the assignment you redo). You may work in groups of two or three.

Write a program which implements the algorithm called Dixon's Random Squares Algorithm, in the textbook. (I called it Morrison-Brillhart/Dixon in class.)

Test your program on numbers of different lengths, and check how your experimental running time compares with the theoretical prediction. Write efficient code.

To deal with the long numbers necessary, you may use Java. There is a class in java.math called BigInteger which should be efficient and easy to use. There is documentation available at

http://java.sun.com/javase/6/docs/api/.

You may use the standard methods provided there.

Please turn in your program, some output and a report. You should send me your program and any extra files via e-mail (they can just be attachments).

Please tell me what language you will be using by Thursday, April 11.