Comp26120
Academic Session: 2016-17

Laboratory Exercise 7: Public-key cryptosystems

Ian Pratt-Hartmann

Preamble

In the lectures, you learned about modular arithmetic and fast modular exponentiation. You also learned about the El Gamal public-key cryptosystem. In this lab, you will implement a proof-of-concept system. The numbers involved are not large enough for this to be anything like secure; nor do we tackle the issue of deployment in practical situations. But these restrictions are, from a conceptual point of view, inessential. If you wish, you can always build a usable system of your own for fun.

The tasks

1. Let a and b be positive integers. Implement a function

```
unsigned long hcf(unsigned long a,unsigned long b)
```

to compute the highest common factor of a and b. Your program must run in time linear in the total size of the inputs, a and b. (Remember: the size of a positive integer a is the number of bits required to write it in standard notation, namely, $\lfloor \log a \rfloor + 1$.)

4 Marks

2. Let p be a prime, and g a primitive root modulo p. Implement a function

```
unsigned long fme(unsigned long g,unsigned long x,
                  unsigned long p)
```

to compute the function $x \mapsto g^x \mod p$. Of course, `fme` stands for 'fast modular exponentiation'. What is the running time of your program as a function of the sizes of g, x and p? Justify your answer.

4 Marks

3. Let p be a prime, and g a primitive root modulo p. Implement a function

```
unsigned long dl(unsigned long y,unsigned long g,unsigned long p)
```

to compute, for a given y ($1 \leq y < p$) the unique integer x ($1 \leq x < p$) such that $y = g^x \mod p$. Of course, `dl` stands for 'discrete logarithm'. What is the running time of your program as a function of the sizes of y, g and p? Justify your answer. Explain why modular exponentiation can be regarded as a *one-way* (or *trapdoor*) function.

4 Marks

4. Let p be a prime. Implement a function

```
unsigned long imp(unsigned long y, unsigned long p)
```

which returns, for a given y ($1 \leq y < p$) the unique integer x ($1 \leq x < p$) such that $x \cdot y \equiv 1 \mod p$. In case you were wondering, `imp` stands for 'inverse modulo prime', a name which I just made up. There are actually two sensible ways to do this: either using exercise 1 (if you implemented it in a certain way) or using exercise 2. Either way, you have to think a little bit about what you are doing, but I recommend using exercise 2. What is the running time of your program as a function of the sizes of y and p? Justify your answer.

2 Marks

5. Now for the business end. Write a program that prints out the following message plus list of user-options:

```
Prime modulus is 65537

Primitive root wrt 65537 is 3

Choose: e (encrypt) | d (decrypt) |k (get public key) | x (exit)?
```

You will see that p and g have been set at fixed values. A simple check will confirm that $g = 3$ is indeed a primitive root of $p = 65537$. If the user selects `k`, he is prompted for a private key x in the range $1 \leq x < p$. The program then computes the El Gamal public key $y = g^x \mod p$. The program goes back to the prompt. Here is the an example of the expected behaviour.

```
Choose: e (encrypt) | d (decrypt) |k (get public key) | x (exit)? k

Type private key: 40000

Public key is: 64675
```

If the user selects e, he is prompted for a message M in the range $1 \leq M < p$ and the public key y (as generated by the `k`-option). The program then selects a random number k in the range $1 \leq x < p$ and computes the El Gamal encryption of M as the integer pair (a, b), where

$$\begin{aligned} a &= g^k \mod p \\ b &= M \cdot y^k \mod p, \end{aligned}$$

where y is the public key. The program goes back to the prompt. Here is the an example of the expected behaviour.

```
Choose: e (encrypt) | d (decrypt) |k (get public key) | x (exit)? e

Type secret number to send: 11121
Type recipient's public key: 64675

The encrypted secret is: (53509, 46390)
```

If the user selects d, he is prompted for a cypher-text C of the form (a, b), with $1 \leq a, b < p$, and the *private* key x entered above. (Remember that x was used to generate the public key by means of which C was computed from some plain-text.) The program then decodes C so as to recover the plain-text. The program goes back to the prompt. Here is the an example of the expected behaviour.

```
Choose: e (encrypt) | d (decrypt) |k (get public key) | x (exit)? d

Type in received message in form (a,b): (53509,46390)
Type in your private key: 40000

The decrypted secret is:  11121
```

6 marks

Hint

You may have problems with number overflows. The data type `unsigned long long` may (or may not) solve them. Your program need not work for primes larger than 4093082899. For testing, I recommend using very small primes. Here is a table of a few primes p with primitive roots modulo p.

p	primitive root modulo p
17	3
257	3
443	2
65537	3
4093082899	2

Evaluation

There are 20 marks in total for this lab exercise, distributed as indicated. The demonstrator will ask you about your program and its design. You will get 20 marks for a fully working program written in readable and properly commented C code, which you can explain to the Demonstrator. Solutions falling short of the ideal will receive marks proportionately.

Resources

There are no auxiliary files required for this lab.