

## MCS 320 Project One : a prototype for a password manager

The goal is to use Maple to prototype a password manager with RSA public key cryptography. The data stored on file by the password manager is in comma separated values (csv) format. Each line contains three items: (1) URL to a login form, (2) login name, (3) password. For example: one line for your webmail could be

```
https://webmail.uic.edu/,mynetid,mypassword
```

where the strings mynetid and mypassword would be actual netid and password.

Storing this file unencrypted is very unsecure. Each line will be encrypted as one large natural number with the RSA public key cryptography system.

### 1. strings as numbers

To compute with strings, we have to convert them into large numbers. This is done by the given procedures `to_number` and `to_string`.

```
[> mydir := "/Users/jan/Courses/MCS320/Spring11/Project_One/": # on Mac OS X
```

Most likely you will have to adjust the definition of `mydir` to the location where you saved the `.mpl` files.

```
[> read '|| mydir || "to_number.mpl";
]> read '|| mydir || "to_string.mpl";
[> message := "symbolic computation is cool";
      "symbolic computation is cool"
[> encoded_message := to_number(message);
      19251302151209030003151316212001200915140009190003151512
[> to_string(encoded_message);
      "symbolic computation is cool"
```

**Assignment One.** The given procedures `to_string` and `to_number` cannot handle an URL like `http://www.uic.edu`. Extend the procedures so they can handle URLs, special punctuation symbols (we will need a comma for example) and also upper case letters.

### 2. public key cryptography

We use the RSA public key cryptosystem to encrypt our data. The security of this system is based on the hardness of factoring large numbers into primes. We need a procedure to do modular exponentiation.

```
[> read '|| mydir || "modexp.mpl":
[> encryption_key := 27:
[> modulus_factor := 55:
[> encrypted_message := modexp(encoded_message, encryption_key, modulus_factor);
      19413782546001127469327486538614807955779936390750716873
```

We see that the decoding of the encrypted message is meaningless.

```
[> to_string(encrypted_message);
  .. output omitted ..
[> decryption_key := 3:
[> decrypted_message := modexp(encrypted_message, decryption_key, modulus_factor);
  19251302151209030003151316212001200915140009190003151512
[> to_string(decrypted_message);
  "symbolic computation is cool"
```

The encryption key and the modulus factor are public information, but the decryption key is private. The modulus factor (the number we use in the modular computations) is the product of two prime numbers:

```
[> fs := ifactor(modulus_factor);
  (5) (11)
```

While the modulus factor is public, the two factors are private.

```
[> p := op(op(fs)[1]); q := op(op(fs)[2]);
  5
  11
```

To make our own encryption scheme, we first choose two large enough primes  $p$  and  $q$ . Then we choose the encryption key so that it is relative prime with respect to  $(p-1)*(q-1)$ :

```
[> igcd(encryption_key, (p-1)*(q-1));
  1
```

We find the decryption key via the extended gcd algorithm, as one of the cofactors of the gcd:

```
[> igcdex(encryption_key, (p-1)*(q-1), 'a', 'b');
  1
[> a; b;
  3
  -2
[> a*encryption_key + b*(p-1)*(q-1);
  1
```

So we find  $a = 3$  as the decryption key.

```
[> decryption_key*encryption_key mod (p-1)*(q-1);
  1
```

The problem with a small modulus factor is that the system is easy to crack, as show above.

**Assignment Two.** Experiment with increasing sizes of  $p$  and  $q$  to make the modulus factor that is very hard for Maple to factor. In your experiment, take primes of increasing number of digits (you can use successive applications of the `nextprime` command) and measure how long you must wait for the result of `ifactor`. Extrapolate from these timings a save size for the values of  $p$  and  $q$ . Justify your choice of size, referring to the algorithm to factor a natural number  $n$ : try all divisors from 2 to the square root of  $n$ .

### 3. a password manager

The messages we will encrypt will be strings like

```
[> s1 := "urla logina passa"; s2 := "urlb loginb passb";
      "urla logina passa"
      "urlb loginb passb"
```

The procedure `encrypt` takes on input a string, the encryption key, modulus factor, and the name of a file. It returns the number (the encryption of the string) that was appended to the file.

```
[> read '|| mydir || "encrypt.mpl";
[> datafile := convert(' || mydir || "passfile.txt",string);
"/Users/jan/Courses/MCS320/Spring11/Project_One/passfile.txt"
[> n1 := encrypt(s1,encryption_key,modulus_factor,datafile);
      2120439406886682339534459806959526
[> n2 := encrypt(s2,encryption_key,modulus_factor,datafile);
      2120440940334648723754614052248523
```

The procedure `decrypt` takes on input a number of a line on file, the decryption key, modulus factor, and the name of a file. It returns the decryption of the given line on file.

```
[> read '|| mydir || "decrypt.mpl";
[> decrypt(1,decryption_key,modulus_factor,datafile);
      "urla logina passa"
[> decrypt(2,decryption_key,modulus_factor,datafile);
      "urlb loginb passb"
```

**Assignment Three.** Write code for the `encrypt` and `decrypt` procedures. The example above is with strings that do not contain commas or special symbols and with small keys to show that you could complete this assignment independently from the other two assignments. In your answer you have to illustrate that your prototype works on plausible data (I do not need to know your actual account information, just invent some account urls, login names and passwords) and with sufficiently large numbers to guarantee the security of the encrypted data on file.

### 4. the deadline is Monday 14 February 2011, at 11AM

Bring to class the printout of a Maple worksheet that contains

1. listings of the Maple procedures you have written; and
2. output of the experiments.

Structure your document along the answers to the assignments. Write appropriate comments.

*This project must be solved individually, collaborations are not allowed.*

Your worksheet should run like a program, via the menu `Edit` → `Execute` → `Worksheet`.

In addition, also email your Maple worksheet and the code for the procedures to [jan@math.uic.edu](mailto:jan@math.uic.edu).

If you have questions, concerns, or technical difficulties, feel free to come to my office for help.