Mersenne Primes-Based Symmetric-Key Masquerade Block Cipher

© Czeslaw Koscielny 2005

Academy of Management in Legnica, Poland
Faculty of Computer Science
e mail: c.koscielny@wsm.edu.pl

Introduction

A strong cryptographic system, consisting of very simple encrypting/decrypting algorithms, has been described. The system presented is a sort of a flexible symetric-key block cipher in which plaintext blocks, ciphertext blocks and keys have a form of files. Flexibility denotes here that the size and the format of a plaintext file and a cryptogram file may be arbitrary. The size of the plaintext and cryptogram file may vary from 2 to 3245619 bytes. The system can have a very huge keyspace (key size from 16 to 25964952 bits, while AES offers maximum 256 bits) and works in a masquerade mode. As regards the masquerade mode of operation, it should be emphasized that in this mode the cipher provides cryptograms entirely statistically independent on plaintexts, and the only method of breaking the cipher is the exhaustive search of the key space. In the masquerade mode the sender arbitrarily selects a cryptogram file, and encrypting routine uses it and a plaintext file to compute a suitable key file. The cryptogram file, as usual, is sent to the addressee by means of a channel unprotected against eavesdropping while to deliver to him the secret key file, the secure channel is used. It is also possible to negociate with the recipient which file will play the role of a cryptogram: it should be known both to him and to the sender. In this case the only secret key is transmitted. One can also send to the recipient the cryptogram file once, and then use it several times or for several weeks or months.

Traditionally cryptograms look like sequences of random characters and are easily distinguishable form plaintexts. So, working in the masquerade mode, it is recommended to select as a cryptogram any reasonable message to make a task of cryptanalyst more difficult. This justifies the name of the operation mode: the cryptogram is a plausible but completely different message than the plaintext contained in this cryptogram. For example, if we want to encrypt a message being a secret market data, as its cryptogram we may take a music file with a fragment of a Beethoven symphony.

At last, it is worth noticing that the cipher allows, in practice, to encrypt the whole message in one cryptogram due to permissible large size of a plaintext file.

Encryption/Decryption Algorithms

The encryption operation is very simple and consists in computing a secret key

$$K = \mathbf{i2f}(\mathbf{f2i}(M) - \mathbf{f2i}(C))$$

where *M* denotes a plaintext file, **f2i** and **i2f** - conversion procedures from file to integer and *vice versa*, *C* denotes a name of an arbitrarily selected cryptogram file and K - a name of a key file, respectively. Thus, in the masquerade mode of operation, *C* may be entirely independent on *M* and *K*. Cryptograms, as in the conventional mode of operation, are transmitted through unsecured channel, whereas secret keys through the secure channel.

During decryption the recovered plaintext file is determined using the equation

$$M = \mathbf{i2f}(\mathbf{f2i}(\mathrm{K}) + \mathbf{f2i}(C))$$

The operations of addition and subtraction in encryption/decryption algorithms are operations in the addititive group of *GF(p)* (it is possible, of course, to operate in the multiplicative group of this field, but multiplicative operations are much more slower than addition and subtraction).These algorithms can yield strong encryption if the key size is equal to at least 4000 bits, i.e. if p = nextprime(2^{4000})). But everybody knows that it is difficult to find such huge primes, even for Maple, therefore, the Mersenne primes are used.

Maple Implementation of a Mersenne Primes-Based Symmetric-Key Masquerade Block Cipher

In practice, for encrypting/decrypting, the reader can use only two following procedures:

- **`ENCmers := proc(ptfn, cfn::string) ... end proc: -`** the encrypting procedure. The actual parameters which are substituted for formal parameters **`ptfn`** and **`cfn`** are names of a plaintext file and a cryptogram file, respectively. The procedure creates the key file, assigns the name to it, and reveals this name to the user.
- **`DECmers := proc(cfn, kfn::string) ... end proc: -`** the decrypting procedure, in which the formal parameters **`cfn`** and **`kfn`** are replaced by the actual names of a cryptogram file and a key file. The routine forms the recovered plaintext file and similarly as previous routine selects the name for it, communicating this name to the user.

These routines work correctly even if a plaintext file or selected for cryptogram file have leading and trailing zero bytes (as, e.g. some **`*.dll`** and **`*.exe`** files).

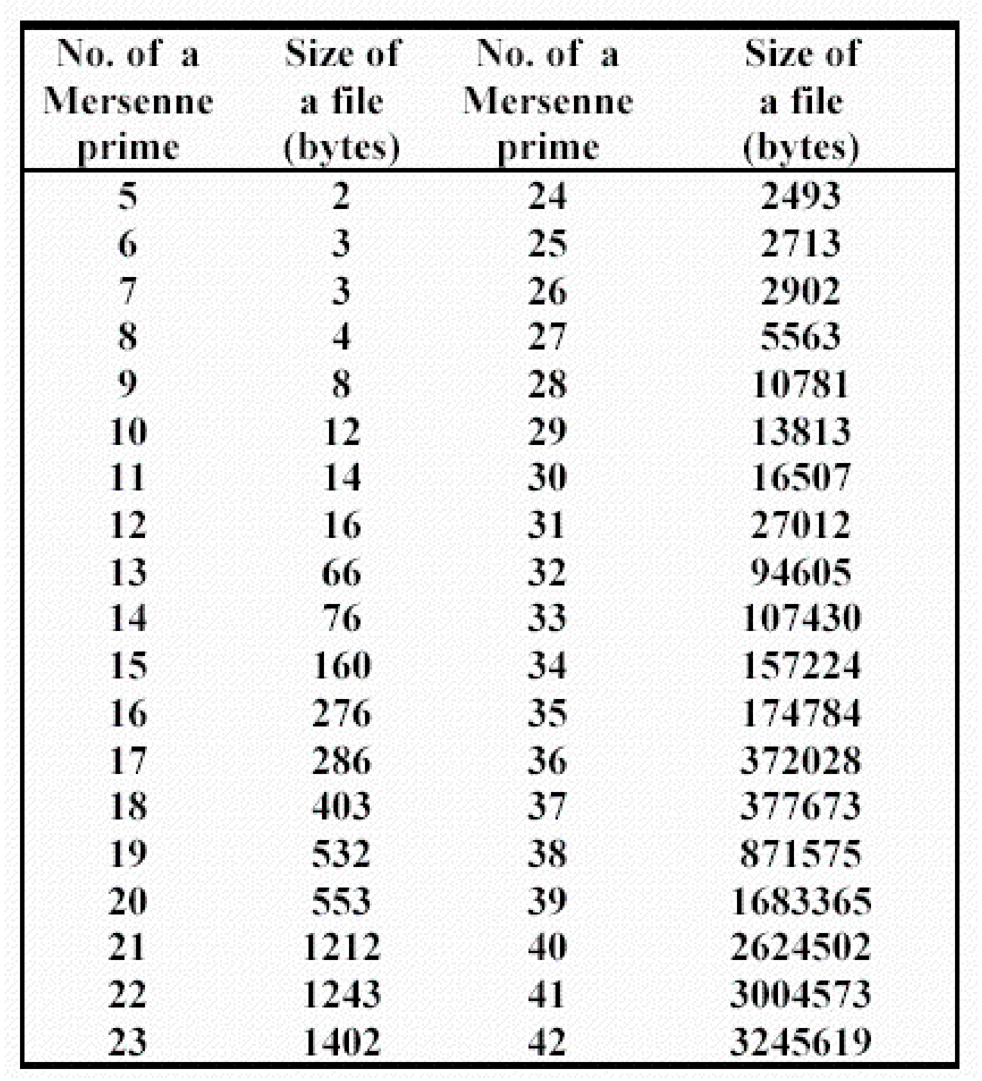
To verify whether, after deciphering, the original and recovered plaintext files are the same, we may put to use the procedure

- **`filcomp(f1, f2::string) ... end proc:`**

which compares two files with the names **`f1`** and **`f2`**.

The above procedures, together with 9 necessary auxiliary routines (**`setp`**, **`Emrs`**, **`Dmrs`**, **`l2f`**, **`f2l`**, **`i2f`**, **`f2i`**, **`i2l`**, **`l2i`**), written in Maple interpreter, are contained in the file **`mbc.m`** and my be listed by means of the statement **`showstat`**. One should also know that the cipher decides itself which number of a Mersenne prime, depending on the plaintext file and cryptogram file sizes, ought to be used (see Table 1).

Table 1. Approximate value of a maximal size of a plaintext and cryptogram file versus a number of a Mersenne prime *p* used for computing in *GF(p)*



No. of a Mersenne prime	Size of a file (bytes)	No. of a Mersenne prime	Size of a file (bytes)
5	2	24	2493
6	3	25	2713
7	3	26	2902
8	4	27	5563
9	8	28	10781
10	12	29	13813
11	14	30	16507
12	16	31	27012
13	66	32	94605
14	76	33	107430
15	160	34	157224
16	276	35	174784
17	286	36	372028
18	403	37	377673
19	532	38	871575
20	553	39	1683365
21	1212	40	2624502
22	1243	41	3004573
23	1402	42	3245619

Encrypting/Decrypting Experiments

To begin with, we must first execute the statements mentioned below. The actual parameter **`ap`** in the statement **`currentdir(ap)`** ought to be the name of the directory, where the files: **`m1.jpg`**, **`m2.gif`**, **`m3.mid`**, **`m4.txt`**, **`mersmbc.mw`** and Maple internal format file **`mbc.m`** are stored.

```
> restart;
currentdir("d:/mersmbc"); #in the case if ap := "d:/mersmbc"
read "mbc.m":
```

In the first example as the the cryptogram of the plaintext file **`m1.jpg`** the file **`m2.gif`** is selected.

```
> ENCmers("m1.jpg", "m2.gif");
```

The rate of encrypting is about 36 kbyte/s using PC with a clock 3.192 GHz.

As a result of encrypting the key file **m1m2jpg.key** has been created. It is worth mentioning that the name of the key file, produced during encryption, contains the extension of a plaintext file name (it is useful in decrypting). The extension of any file name must have three characters.

Now, knowing the names of cryptogram and key files, we may again recover from the cryptogram file the plaintext file by calling the decrypting procedure:

```
> DECmers("m2.gif","m1m2jpg.key"):
```

We see that the name of the recovered plaintext file has a proper extension. It is possible now to compare the original and recovered plaintexts:

```
> filcomp("m1.jpg", "m1m2.jpg");
```

The reader will easily follow the remainig examples and, without trouble, certainly will experiment with his own files.

```
> ENCmers("m2.gif", "m1.jpg"):
> DECmers("m1.jpg", "m2m1gif.key"):
> filcomp("m2.gif", "m2m1.gif");
> ENCmers("m4.txt", "m3.mid"):
> DECmers("m3.mid", "m4m3txt.key"):
> filcomp("m4.txt", "m4m3.txt");
> ENCmers("m3.mid", "m4.txt"):
> DECmers("m4.txt", "m3m4mid.key"):
> filcomp("m3.mid", "m3m4.mid");
```

Conclusions

In the contribution an example of a method of construction of *GF(p)*-based strong block cipher has been presented. The security of cipher discussed can be tuned since the keysize can vary from several to 13466920 bits. The cipher may work either as a masquerade cipher with the key file size equal to the maximal size of a plaintext file or as a conventional symmetric-key block cipher. The worksheet enables the reader to experimentally encrypt/decrypt arbitrary files of a size up to 3.25 Mbyte. After executing the worksheet **mersmbc.mw** 8 files have been created and stored in the current directory. Any file used in the experiment and obtained as a result of encryption/decryption procedures may be preliminarily cryptanalysed using [1]. Due to this action we may be sure that the cipher is really strong.

The cipher works quite fast in the Maple environment. By implementing encryption/decryption procedures in C or in other high level programming language we may attain the encryption/decryption rate of an order from hundreds of kbyte/s to several Mbyte/s..

However, the procedures presented have several limitations and imperfections - they are not entirely resistant to input data errors and the plaintext, key and cryptogram file names ought to consist of rather two characters and must have exactly three characters in the extension part of a file name.

The worksheet **mersmbc.mw** handles intensively binary files. Therefore, if the reader, executing the worksheet, causes an input data or another error, some files remain open. In this case, to continue the experiment, the internal memory of Maple kernel should be cleared using the statement **restart** and after it the file **mbc.m** ought to be read again.

References

[1] Koscielny, C.: Maple Tools for Preliminary Cryptanalysis,
http://www.maplesoft.com/applications

The content of the file `mbc.m`

```
l2f := proc(l::list, fn::string)
local f;
   1   f := fopen(fn,WRITE,BINARY);
   2   writebytes(f,l);
   3   fclose(f)
end proc:


f2l := proc(fn::string)
local l, f, fs;
   1   f := fopen(fn,READ,BINARY);
   2   fs := filepos(f,infinity);
   3   filepos(f,0);
   4   l := readbytes(f,fs);
   5   fclose(f);
   6   l
end proc:


i2f := proc(n::nonnegint, fn::string)
   1   l2f(i2l(n),fn)
end proc:


f2i := proc(fn::string)
   1   l2i(f2l(fn))
end proc:


i2l := proc(nn::nonnegint)
   1   if nn = 0 then
   2     [0]
       else
   3     convert(nn,base,256)
       end if
end proc:


l2i := proc(l::list)
```

```
local t, m;
    t := modp1(ConvertIn(l,x),p);
    subs(x = 256,t)
end proc:


ENCmers := proc(ptfn, cfn::string)
local ptfs, cfns, kfn, mx, t, ptfl, cfl, f;
global p;
    printf("ENCRYPTING:");
    ptfs := filepos(ptfn,infinity);
    fclose(ptfn);
    cfns := filepos(cfn,infinity);
    fclose(cfn);
    if cfns < ptfs then
      mx := ptfn
    else
      mx := cfn
    end if;
    p := setp(mx);
    kfn := cat(substring(ptfn,1 .. length(ptfn)-4),substring(cfn,1
.. length(ptfn)-4),substring(ptfn,length(ptfn)-2 ..
length(ptfn)),".key");
    printf("%s%s%s","\nPlaintex file name: <<",ptfn,">>");
    printf("%s%s%s",".\nCryptogram file name <<",cfn,">>.");
    t := time();
    f := fopen(ptfn,READ,BINARY);
    ptfl := readbytes(f,ptfs);
    fclose(f);
    ptfl := [1, op(ptfl), 1];
    f := fopen(cfn,READ,BINARY);
    cfl := readbytes(f,cfns);
    fclose(f);
    cfl := [1, op(cfl), 1];
    Emrs(ptfl,cfl,kfn);
    t := time()-t;
    printf("%s%s%s%6.2f%s","\nKeyfile named <<",kfn,">> computed in
",t," s.\n");
    printf("%s","Now you can send safely cryptogram");
    printf(" and key file to your correspondent.\nThe key file ");
    printf(" should be sent to him using a secure channel. ");
    printf("%s%6.2f%s","\nEnryption rate: ",1/1000*ptfs/t,"
kbyte/s.")
end proc:


DECmers := proc(cfn, kfn::string)
local rptfn, t, f, fs, rptfl, cfl, l, cfns;
global p;
    printf("DECRYPTING:\n");
    rptfn := cat(substring(kfn,1 .. length(kfn)-
7),".",substring(kfn,length(kfn)-6 .. length(kfn)-4));
```

```
   printf("%s%s%s","Key file: <<",kfn,">>.\n");
   printf("%s%s%s","Cryptogram file: <<",cfn,">>.\n");
   printf("%s%s%s","Recovered plaintext file: <<",rptfn,">>.");
   t := time();
   cfns := filepos(cfn,infinity);
   fclose(cfn);
   p := setp(kfn);
   f := fopen(cfn,READ,BINARY);
   cfl := readbytes(f,cfns);
   fclose(f);
   cfl := [1, op(cfl), 1];
   Dmrs(cfl,kfn,rptfn);
   f := fopen(rptfn,READ,BINARY);
   fs := filepos(f,infinity);
   filepos(f,0);
   l := readbytes(f,fs);
   fclose(f);
   f := fopen(rptfn,WRITE,BINARY);
   l := [op(2 .. nops(l)-1,l)];
   writebytes(f,l);
   fclose(f);
   t := time()-t;
   printf("%s%6.2f%s","\nDeciphering done in ",t," s.");
   printf("%s%6.2f%s","\nDecryption rate: ",1/1000*cfns/t,"
kbyte/s.")
end proc:


Emrs := proc(ptfl, cfl::list, kfn::string)
   i2f(`mod`(l2i(ptfl)-l2i(cfl),p),kfn)
end proc:


Dmrs := proc(cfn::list, kfn, rptfn::string)
   i2f(`mod`(l2i(cfn)+f2i(kfn),p),rptfn)
end proc:


setp := proc(pcfn::string)
local pcfs, i, n, p;
   pcfs := filepos(pcfn,infinity);
   fclose(pcfn);
   i := 1;
   n := nops(convert(numtheory[mersenne]([i]),base,256));
   while n < pcfs do
     i := i+1;
     p := numtheory[mersenne]([i]);
     n := nops(convert(p,base,256))
   end do;
   printf("%s%d%s","\nKey size = ",8*n," bits."), p
end proc:
```

```
filcomp := proc(f1, f2::string)
local v1, v2, fs1, fs2, i, b1, b2, nd;
       if f1 = f2 then
         printf("compared files must have different names\n");
         return
       end if;
       v1 := fopen(f1,READ,BINARY);
       v2 := fopen(f2,READ,BINARY);
       fs1 := filepos(f1,infinity);
       filepos(f1,0);
       fs2 := filepos(f2,infinity);
       filepos(f2,0);
       if fs1 <> fs2 then
         printf("Files of different size:\n");
         printf("%s%s%s%d%s%s%s%s%d%s","<<",f1,">>: ",fs1,"
bytes,\n","<<",f2,">>: ",fs2," bytes.")
       end if;
       nd := 0;
       for i to min(fs1,fs2) do
         b1 := readbytes(f1);
         b2 := readbytes(f2);
         if b1 <> b2 then
           nd := nd+1;
           printf("%8d%s%s%4d%s%4d%s",i,"-th position: ",cat("
",f1),b1[1],cat(", ",f2),b2[1],"\n")
         end if
       end do;
       fclose(f1);
       fclose(f2);
       if nd = 0 and fs1 = fs2 then
         printf("OK! files identical!")
       elif fs1 = fs2 then
         printf("%s%d%s","files differ in ",nd," positions!")
       end if
end proc:
```