Quasigroups and Related Systems **13** (2005), 237 – 244

Maple implementation of the ElGamal public key encryption scheme working in SMG($\mathbf{p^m}$)

Czesław Kościelny

Abstract

It has been shown [2, 3, 4] that in ElGamal public key encryption scheme [6, 7, 8] working over $SMG(p^m)$, the need of finding primitive elements of $GF(p^m)$, necessary if the system works traditionally but unfeasible in the case of huge fields, is eliminated. Thus, the discussed system is user-friendly, giving the possibility of very strong encryption with the key of order 10.000 bits and more. The construction of such cryptosystem is of great practical importance, therefore, the paper informs the reader in detail how to resolve this problem using Maple.

1. Introduction

Recall that algorithms describing ElGamal public-key encryption scheme [3, 4, 5] adapted for working in $SMG(p^m)$ are as follows:

Key generation: Each entity creates its public key and the corresponding private key. So each entity $\mathcal{N}$ ought to do the following:

- Choose an arbitrary polynomial $f(x)$ of the degree m over $GF(p)$ and construct a spurious multiplicative group of $GF(p^m)$ that is $SMG(p^m)$, consisting of the set $G = \{1, \ldots, p^m - 1\}$ and of the operation of multiplication of elements from this set, which is performed by means of a function $\mathbf{M}\_(x, y)$, $x, y \in G$. The function $\mathbf{P}\_(x, k)$, carrying

2000 Mathematics Subject Classification: 94A60, 20N
Keywords: cryptography, public-key ciphers, spurious multiplicative group of Galois field, Maple.

out the operation of rising any element x from G to a k^{th} power, $p^m - 1 \leqslant k \leqslant -p^m + 1$, is also defined.

- Select a random invertible element $\alpha \in SMG(p^m),\ \alpha \neq 1$.
- Choose a random integer $a \in G,\ \ 2 \leqslant a \leqslant p^m - 2$, and compute the element $\beta = \mathbf{P\_}(\alpha, a)$.
- $\mathcal{N}$'s public key is α and β, together with $f(N)$ and the functions $\mathbf{M\_}$ and $\mathbf{P\_}$, if these last three parameters are not common to all the entities.
- $\mathcal{X}$'s private key is a.

Encryption: Entity $\mathcal{B}$ encrypts a message m for $\mathcal{A}$, which $\mathcal{A}$ decrypts. Thus $\mathcal{B}$ should make the following steps:

- Obtain $\mathcal{A}$'s authentic public key α, β, and $f(x)$ together with the functions $\mathbf{M\_}$ and $\mathbf{P\_}$ if these parameters are not common.
- Represent the message m as a number from the set G.
- Choose a random integer $k \in G$.
- Determine numbers $c_1 = \mathbf{P\_}(\alpha, k)$ and $c_2 = \mathbf{M\_}(m, \mathbf{P\_}(\beta, k))$.
- Send the ciphertext $c = (c_1,\ c_2)$ to $\mathcal{A}$.

Decryption: To find plaintext m from the ciphertext $c = (c_1,\ c_2)$, $\mathcal{A}$ should perform the following operations:

- Use the private key a to compute $g = \mathbf{P\_}(c_1,\ a)$ and then retrieve the plaintext by computing $m = \mathbf{M\_}(\mathbf{P\_}(g, -1), c_2)$.

2. Maple routines as elements of an application for very secure encrypting of electronic mail using ElGamal algorithm working in $SMG(p^m)$

The application discussed will realize a practical task, so it is assumed that the public, key, secret key, plain text and cryptogram will be files. Therefore, the routines for converting file into number and number into file, denoted as `f2n` and `n2f`, correspondingly, are needed first. Here they are:

```
> f2n := proc(fn::string)
 local l2n, f2l;
     l2n := proc(l::list)
         local t, m;
             t := modp1(ConvertIn(l, x), nn^2);
             subs(x = 256, t)
         end proc;
     f2l := proc(fn::string)
         local l, f, fs;
             f := fopen(fn, READ, BINARY);
             fs := filepos(f, infinity);
             filepos(f, 0);
             l := readbytes(f, fs);
             fclose(f);
             l
         end proc;
     l2n(f2l(fn))
 end proc:


>   n2f := proc(n::nonnegint, fn::string)
 local l2f, n2l;
     l2f := proc(l::list, fn::string)
         local f;
             f := fopen(fn, WRITE, BINARY);
             writebytes(f, l);
             fclose(f)
         end proc;
     n2l := proc(nn::nonnegint)
             if nn = 0 then [0]
             else convert(nn, base, 256)
             end if
         end proc;
     l2f(n2l(n), fn)
 end proc:
```

The formal parameter `fn` of the procedure `f2n` represents the name of the file, which will be converted into a number, and formal parameters of the routine `n2f` denote the number `n` which will be converted into the file named

`fn`. The variable `nn` appearing in the routine `f2n` is global, and it is computed by the routine

```
> INIT_ := proc(pn::posint, fx::polynom)
 global ext, n, nn;
     nn := pn^degree(fx);
     ext := modp1(ConvertIn(modp(fx, pn), x), pn);
     n := pn
 end proc:
```

which initializes calculations in $SMG(p^m)$ and returns, in addition, global variables `n` and `ext`, necessary for routines `M_`, `P_` and `MI_`. These three routines (contained in [5], Appendix C) are also indispensable. They perform multiplication, raising to a power and finding the multiplicative inverse in $SMG(p^m)$, respectively.

The fundamental task in the algorithm of key generations fulfills the routine

```
> frel := proc()
  local alpha, beta, a, l, res, si, i;
    randomize;
    res := "*";
    while res = "*" do
      alpha := rand(rand((nn-1)/2)() .. nn - 1)();
      try res := MI_(alpha) catch: res := "*" end try
    end do;
    a := rand(2 .. nn - 2)();
    beta := P_(alpha, a);
    alpha, beta, a
 end proc:
```

which returns a random invertible element α and an appropriate element β from $SMG(p^m)$ being the components of a public key, and an integer a, playing the role of a secret key. It is evident that there exist many procedures able to do this task.

At last the encryption routine

```
>  ElGEnc := proc(ptfn, c1fn, c2fn::string, alpha, beta)
local c1, c2, k, m;
    m := f2n(ptfn);
```

```
    k := rand(2 .. nn - 2)();
    c1 := P_(alpha, k);
    c2 := M_(m, P_(beta, k));
    n2f(c1, c1fn);
    n2f(c2, c2fn)
end proc:
```

and the decryption routine

```
>  ElGDec := proc(c1fn, c2fn, rptfn::string, a::posint)
local c1, c2, g, ig, m;
    c1 := f2n(c1fn);
    c2 := f2n(c2fn);
    g := P_(c1, a);
    m := M_(P_(g, -1), c2);
    n2f(m, rptfn)
end proc:
```

acting according to the description given in Section 1, will be necessary. The `ElGEnc` procedure enciphers the plaintext file having the filename `ptfn` taking into account the public key components `alpha` and `beta` and creates two cryptogram files named as `c1fn` and `c2fn`. The `ElGDec` procedure deciphers cryptogram files `c1fn` and `c2fn` and creates the retrieved plaintext file `rptfn` taking into account the secret key `a`.

From the above 9 blocks a practiced programmer will easily assemble a user-friendly application for encrypting the electronic mail in the Maple environment. As an example, the author, using the above bricks, has built an application consisting of three procedures:

- `KeyGen(plaintextf_file_size, degree_of_fx)`,
- `Encrypting(plaintextf_file, smg_data_file)`,
- `Decrypting(c1_file, c2_file, smg_data_file)`.

The `KeyGen` procedure automatically chooses a Mersenne prime p in such a way that the cryptosystem could process files of the desired size. Doing this it must take into account the degree of the polynomial $f(x)$ over $GF(p)$. Next the routine randomly generates this polynomial, computes the integer $n = p^{deg(f(x))}$ and generates a random cryptographic key. The public and private keys in the files `pkf` and `skf` are stored, correspondingly, while n and $f(x)$ to the file `nfx.smg` are written.

The `Encrypting` procedure takes $f(x)$ and n from the file `nfx.smg` and the public key of the desired correspondent form the file `pkf`. These data are sufficient to encrypt a plaintext file. It is assumed that the name of a plaintext file consists of one character and exactly of three characters of extension. Assuming that the name of a plaintext file is `"n.eee"`, the generated cryptogram is written to the files `c1neee.cry` and `c2neee.cry`. The contents of the plaintext file may be arbitrary (text, voice, picture, etc.), but, evidently, the file could not contain leading 0 bytes.

The `Decrypting` procedure takes $f(x)$ and n from the file `nfx.smg`, an appropriate secret key from the file `skf` and decrypts the crytogram files `c1neee.cry` and `c2neee.cry` (assuming that the file `n.eee` has been enciphered). The retrieved plaintext in the file `nr.eee`, having the proper extension, is stored.

If, for example, we want to process plaintext files of size 1.300 Bytes using $f(x)$ of degree 5, we ought to execute three statements. Under the above assumption a typical use and output of this application is the following:

```
> KeyGen(1300, 5);

KEY GENERATION:
Maximal plaintext file size = 1376 bytes.
Keys computed in 7 s and saved.
Public key file name:  "pkf".
Private key file name:  "skf".
Required for encryption/decryption data,
i.e.  n and f(x) saved in the file "nfx.smg".

> Encrypting("m.txt", "nfx.smg");

ENCRYPTING:
Plaintext file size = 1301 bytes.
Plaintex file name:  "m.txt".
Public key file name:  "pkf".
Cryptogram files named "c1mtxt.cry"
and "c2mtxt.cry" computed in 14 s and saved.

> Decrypting("c1mtxt.cry", "c2mtxt.cry", "nfx.smg");

DECRYPTING:
Secret key taken from the file "skf".
SMG data taken from the file:  "nfx.smg".
```

```
Cryptogram files named:  "c1mtxt.cry" and "c2mtxt.cry".
Recovered plaintext file named "mr.txt"
computed in 7 s and saved.
```

We see that the program informs the user about the stages of processing. In the case considered the computations are performed on integers having 3316 decimal digits. The secret key is an integer belonging to the set of order $0.7004904817 \cdot 10^{3316}$, therefore, the cryptographic key of the system equals exactly to 10.007 bits.

The three discussed procedures are not listed here, because they occupy place without making a contribution to the main problem of the paper. But if the reader wants to see them, the author willingly realizes his wish by email (`joczeko@poczta.onet.pl`).

3. Conclusions

In the paper it has been proved that ElGamal public key encryption scheme working in $SMG(p^m)$ may be easily implemented in Maple. It is obvious that each elementary routine mentioned in Section 2 and needed for this implementation can be, without difficulty, translated into any compiled language. Such translation allows to considerably accelerate encryption/decryption rate. Thus, the discussed cryptosystem is suitable not only for encrypting keys for symmetric cipher but also for direct encryption of messages.

It is worth noticing that ElGamal public key encryption may also work in the multiplicative system of $gff(n^m)$ [5].

References

[1] **N. Ferguson, B. Schneier**: *Practical Cryptography*, John Wiley & Sons, 2003.

[2] **C. Kościelny**: *A New Approach to the ElGamal Encryption Scheme*, Int. J. Appl. Math. Comput. Sci. **14** (2004), 101 – 103.

[3] **C. Kościelny**: *User-Friendly ElGamal Public-Key Encryption Scheme*, `http://www.mapleapps.com`, 2003.

[4] **C. Kościelny**: *Spurious multiplicative group of $GF(p^m)$: a new tool for cryptography*, Quasigroups and Related Systems **12** (2004), 61 – 73.

[5] **C. Kościelny**: *Computing in $GF(p^m)$ and in $gff(n^m)$ using Maple*, Quasigroups and Related Systems **13** (2005), 245 – 264.

[6] **A. J. Menezes, P. C. van Oorschot, S. A. Vanstone**: *Handbook of Applied Cryptography*, CRC Press, 1998.

[7] **B. Schneier**: *Applied Cryptography*, (Second Edition): Protocols, Algorithms, and Cource Code in C, John Wiley & Sons, 1996.

[8] **D. R. Stinson**: *Cryptography – Theory and Practice*, CRC Press, 1995.

Received June 6, 2005

Academy of Management in Legnica
Faculty of Computer Science
ul. Reymonta 21
59-220 Legnica
Poland
e-mail: c.koscielny@wsm.edu.pl