Quasigroups and Related Systems **13** (2005), 245 – 264

Computing in GF($\mathbf{p^m}$) and in gff($\mathbf{n^m}$) using Maple

Czesław Kościelny

Abstract

It is mentioned in [1] that the author intends to show how to construct strong ciphers using $SMG(p^m)^{\diamond}$. But in order to implement such cryptosystems, an effective tool for computing in $GF(p^m)$ and $SMG(p^m)$, in the form of an appropriate hardware or software, is needed. The operation of this hardware or software ought to be defined by means of the detailed algorithms. Thus, to get ready the execution of his intention, the author describes in the paper these algorithms, which are represented as routines, written in the comprehensive Maple interpreter, intelligible both for mathematicians and programmers as well. The routines may be used either immediately as elements of encrypting/decrypting procedures in the Maple programming environment or can be easily translated into any compiled programming language (in this case encryption/decryption can be performed at least 100 times faster than in the Maple environment). Aside from that on the basis of the mentioned routines any VLSI chip as the encrypting/decrypting hardware for $SMG(p^m)-$ and $GF(p^m)-$based cryptosystems can be produced.

It has also been shown that $SMG(p^m)$ can be considered as a multiplicative system of an algebraic structure with addition and multiplication operations, containing a large class of systems, including $GF(p^m)$. The system is denoted as $gff(n^m)$, and multiplication in it is performed modulo an arbitrary polynomial od degree m over the ring Z_n. That way $gff(n^m)$ is a generalization of Galois field, very well suited for applications in cryptography. This system is named a generalized finite field.

$^{\diamond}$For all prime p, for any positive integer $m \geq 2$ and for any polynomial $f(x)$ of degree m over $GF(p)$ there exists an algebraic system $SMG(p^m) = \langle Gx, \bullet \rangle$, consisting of the set Gx of all $p^m - 1$ non-zero polynomials of degree d over $GF(p)$, $0 \leqslant d \leqslant m-1$, and of an operation of multiplication of these polynomials modulo polynomial $f(x)$. Such an algebraic system is a generalization of the multiplicative group of $GF(p^m)$, therefore, it is called the spurious multiplicative group of $GF(p^m)$.

2000 Mathematics Subject Classification: 05B15, 20N05, 94B05

Keywords: Galois fields, generalized finite fields, cryptography, Maple.

1. Introduction

It is not possible to do serious application research in the area of cryptology without complete knowledge concerning several algebraic systems. Prominent position on the list of such systems takes the Galois field. Thus, now it will be reminded to the reader about the main properties of this system, which are the most important for cryptographic practice.

Recall that for all prime p, for any positive integer $m \geq 1$ and for any irreducible polynomial

$$f(x) = x^m + \sum_{i=1}^{m} f_{m-i}\, x^{m-i} \tag{1}$$

of degree m over $GF(p)$ there exists an algebraic system called Galois field and denoted as $GF(p^m)$

$$GF(p^m) = \langle \mathcal{F},\ +,\ \cdot \rangle, \tag{2}$$

consisting of the set $\mathcal{F}$ of all p^m polynomials of degree d over $GF(p)$, $0 \leqslant d \leqslant m-1$, and of operations of addition and multiplication of these polynomials. Since $GF(p^m)$ is a field, it must satisfy the following set of axioms, concerning any field:

F1 The system $\langle \mathcal{F},\ + \rangle$, is an abelian group.

F2 The system $\langle \mathcal{F}^*,\ \cdot \rangle$, is an abelian group, $\mathcal{F}^* = \mathcal{F} \setminus \{0\}$, 0 is an additive identity element.

F3 $\forall a,\ b,\ c\ \in \mathcal{F}\ (a \cdot (b+c) = a \cdot b + a \cdot c) \wedge ((a+b) \cdot c = a \cdot c + b \cdot c)$.

In the case of $GF(p^m)$ the above axioms are fulfilled if addition and multiplication are performed according to the way shown beneath.

Let

$$a(x) = \sum_{i=1}^{m} a_{m-i}\, x^{m-i}, \quad b(x) = \sum_{i=1}^{m} b_{m-i}\, x^{m-i} \tag{3}$$

be two elements of $\mathcal{F}$. Then their sum will be

$$a(x) + b(x) = c(x) = \sum_{i=1}^{m} c_{m-i}\, x^{m-i}, \tag{4}$$

where

$$c_i \equiv a_i + b_i \pmod{p}, \quad i = 0, \ldots, m-1.$$

Similarly

$$a(x) - b(x) = d(x) = \sum_{i=1}^{m} d_{m-i}\, x^{m-i}, \tag{5}$$

where

$$d_i \equiv a_i - b_i \pmod{p}, \quad i = 0, \ldots, m-1.$$

The multiplication is more complicated. To calculate the product of two elements belonging to $GF(p^m)$ one must first compute

$$g(x) = a(x) \cdot b(x) = g_{2\,m-2}\, x^{2\,m-2} + g_{2\,m-3}\, x^{2\,m-3} + \cdots + g_2\, x^2 + g_1\, x + g_0$$

where

$$\begin{aligned}
&g_0 \equiv a_0\, b_0 \pmod{p},\\
&g_1 \equiv a_1\, b_0 + a_0\, b_1 \pmod{p},\\
&g_2 \equiv a_2\, b_0 + a_0\, b_2 + a_1\, b_1 \pmod{p},\\
&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\\
&g_{2\,m-3} \equiv a_{m-1}\, b_{m-2} + a_{m-2}\, b_{m-1} \pmod{p},\\
&g_{2\,m-2} \equiv a_{m-1}\, b_{m-1} \pmod{p}.
\end{aligned}$$

Next, to obtain finally the product $h(x)$ of two $GF(p^m)$ elements (3), we must represent $g(x)$ as

$$g(x) = u(x) \cdot f(x) + h(x) \tag{6}$$

using addition and multiplication modulo p, wherefrom

$$a(x) \cdot b(x) = h(x).$$

The operation of multiplication in $GF(p^m)$ may also be shortly written as

$$h(x) \equiv a(x) \cdot b(x) \pmod{f(x)}.$$

The multiplicative inverse $a^{-1}(x)$ of the element $a(x)$ can be determined by means of extended Euclidean algorithm for polynomials, which yields:

$$a(x) \cdot a^{-1}(x) + w(x) \cdot f(x) = 1,$$

that is

$$a(x) \cdot a^{-1}(x) \equiv 1 \pmod{f(x)}.$$

So

$$a(x)/b(x) = a(x) \cdot b^{-1}(x).$$

We see that we can compute in $GF(p^m)$ as in any field, performing addition, subtraction, multiplication, division and the operation of rising to a power (by repeating the multiplication operation). The presented principles of computing in Galois field may be suitably optimized or improved to be well adapted for hardware or software implementation. It's worth mentioning here that elements of $GF(p^m)$ can be represented not only as polynomials or vectors over $GF(p)$, but also as numbers. The latter case is the most interesting for cryptography, therefore, we will continue the problem of computing in Galois field, considering mainly the system

$$GF(p^m) = \langle \mathrm{F},\ +,\ \bullet \rangle, \tag{7}$$

where $\mathrm{F} = \{0,\ 1,\ \ldots,\ p^m - 1\}$. The system (7) is obtained from the system (2) using the isomorphic mapping

$$\sigma : \mathcal{F} \to \mathrm{F}, \tag{8}$$

defined by the function

$$\sigma(a(x)) = a(p) = \mathrm{A}\ \in \mathrm{F}, \tag{9}$$

converting a polynomial $a(x) \in \mathcal{F}$ to a number from the set F.

The mapping σ is an isomorphism, so the inverse mapping σ^{-1} exists and is described by means of the following two-step algorithm:

Step 1:

convert a base 10 number $\mathrm{A} \in \mathrm{F}$ to base p, namely,

$$\mathrm{A} = a_{m-1}\ \cdots\ a_1\ a_0,\ a_i\ \in \{0,\ 1,\ \ldots,\ p-1\},$$

Step 2:

$$\sigma^{-1}(\mathrm{A}) = a_0 + a_1\, x + \cdots + a_{m-1}\, x^{m-1} \ \in \mathcal{F}.$$

Thus,

$$\begin{aligned} \forall\ \mathrm{A},\, \mathrm{B} \in \mathtt{F}\ (\mathrm{A} \bullet \mathrm{B} = \sigma(\sigma^{-1}(\mathrm{A}) \cdot \sigma^{-1}(\mathrm{B}))) \wedge \\ (\mathrm{A} + \mathrm{B} = \sigma(\sigma^{-1}(\mathrm{A}) \ + \ \sigma^{-1}(\mathrm{B}))). \end{aligned} \tag{10}$$

It is also said that the field $GF(p^m)$ is the field extension $GF(p)[x]/(f(x))$ where $f(x)$ is an irreducible polynomial of degree m over the integers modulo p.

According to the above description of operations in a Galois field we may note that to efficiently implement arithmetic in $GF(p^m)$ we need fast routines doing addition of polynomials over $GF(p)$ and their multiplication over $GF(p)$ modulo irreducible polynomial with coefficients from $GF(p)$. Besides we also need a function which realizes the mappings σ and σ^{-1}, the function determining the extended Euclidean algorithm for polynomial, etc. The system Maple provides such set of the routines which use a special data representation. Knowledge of this representation is not required by the user who wants to compute in Galois fields only. In this case a user-friendly module **GF** suffices.

It is mentioned in the Maple manual that if the modulus p is sufficiently small, operations in $GF(p^m)$ are performed directly by the hardware. The largest prime for which computations are done in this way is the number 46327 (on a 32 bit machine).

2. Computing in $GF(p^m)$ using Maple GF package

The Maple library package **GF**, having the structure of a module, returns routines and constants performing arithmetic in $GF(p^m)$. To begin computing we must first create an instance `F` of a Galois field $GF(p^m)$ using, for example, the statements

```
#arithmetic in GF(125) has been defined
>  p := 5: m := 3:
   f := 1 + 2*x  + x^3:
   F := GF(p, m, f):
```

The actual parameters `p`, `m` and `f` exactly correspond to the variables p, m, and the polynomial $f(x)$, used in Section 1. The parameter `f` is optional - if it is absent in the invocation statement of the module `GF`, then the system Maple selects itself the irreducible polynomial `f(T)`. This case will not be considered here, because we must control the behavior of the field using the polynomial `f`.

Addition, subtraction, multiplication, raising to the $k-$th power, computation of the multiplicative inverse and division in the Galois field are performed by means of the following routines, respectively:

`F:-‘+‘(x1, x2, ..., xn::zppoly)`	: $n-$ary addition
`F:-‘-‘(x1, x2::zppoly)`	: unary or binary subtraction
`F:-‘*‘(x1, x2, ..., xn::zppoly)`	: $n-$ary multiplication
`F:-‘^‘(x::zppoly, k::integer)`	: raising x to the $k-$th power
`F:-inverse(x::zppoly)`	: unary inversion
`F:-‘/‘(x1, x2::zppoly)`	: unary or binary division

The operands `x1, x2, ..., xn` and `x` of the routines performing operations in $GF(p^m)$ must be of a special type, `zppoly`, relating to the Maple `modp1` function. The results returned by these routines are of the same type. But we may need to operate using operands of type `polynom`, `nonnegint` and `zppoly` and obtain these three type of results. To achieve the aim we ought to use the following unary conversion routines:

routine name	type of result
`F:-input(x::integer)`	`zppoly`
`F:-output(x::zppoly)`	`integer`
`F:-ConvertIn(x::symbol, +, * or ^)`	`zppoly`
`F:-ConvertOut(x::zppoly)`	`symbol, +, * or ^`

In practice, we usually use operands of type `polynom` or `nonnegint` and we want to have the type of results of computations of the same type as that of operands.

Example 1. Suppose that the statements in the beginning of the section and the statements beneath have been executed. We will now compute in $GF(125)$. After defining three elements of $GF(125)$ in the form of polynomials `ax`, `bx` and `cx`, we can observe how to compute the multiplicative inverse of `ax`, the additive inverse of `bx`, the sum of `ax`, `bx` and `cx` and the

product of these three elements:

```
> ax := 4*x + 3: bx := 2*x + 1: cx := x^2 + 2:
> F:-ConvertOut(F:-inverse(
                F:-ConvertIn(ax)));
```

$$4x^2 + 2x + 4$$

```
> F:-ConvertOut(F:-`-`
                F:-ConvertIn(bx)));
```

$$3x + 4$$

```
>   F:-ConvertOut(F:-`+`(
                  F:-ConvertIn(ax),
                  F:-ConvertIn(bx),
                  F:-ConvertIn(cx)));
```

$$x^2 + x + 1$$

```
>  F:-ConvertOut(F:-`*`(
                 F:-ConvertIn(ax),
                 F:-ConvertIn(bx),
                 F:-ConvertIn(cx)));
```

$$3x^2 + 2x + 1$$

Further let us define three numbers `A`, `B` and `C` which will play the role of elements form $GF(125)$ by means of the appropriate statement and let's execute the same operations as previously:

```
> A := 23: B := 11: C := 27:
> F:-output(F:-inverse(
            F:-input(A)));
```

$$114$$

```
> F:-output(F:-`-`(
            F:-input(B)));
```

$$19$$

```
>  F:-output(F:-‘+‘(
             F:-input(A),
             F:-input(B),
             F:-input(C)));
```

31

```
>  F:-output(F:-‘*‘(
             F:-input(A),
             F:-input(B),
             F:-input(C)));
```

86

It is also possible to calculate more complicated expressions over $GF(125)$ using directly the package **GF**. E.g. the expression

$$w = \frac{A\,B + A\,C + B\,C}{A + B + C}$$

may be calculated as follows:

```
> w := F:-output(F:-‘/‘(
          F:-‘+‘(
          F:-‘*‘(F:-input(A), F:-input(B)),
          F:-‘*‘(F:-input(A), F:-input(C)),
          F:-‘*‘(F:-input(B), F:-input(C))),
          F:-‘+‘(F:-input(A), F:-input(B), F:-input(C))));
```

$$w := 6$$

Programming of similar expressions can be considerably simplified by means of auxiliary procedures having short names. For example, if we use routines named `a_`, `m_` and `d_` for performing addition, multiplication and division in $GF(p^m)$, respectively, then the above expression will have the form

```
> w := d_(a_(m_(A, B), m_(A, C), m_(B, C)), a_(A, B, C));
```

$$w := 6$$

which gives the same result but is much more simple. In Appendices A and B it is shown how to construct such routines.

The module **GF** also exports the following functions:

```
F:-trace(x::zppoly),
F:-norm(x::zppoly),
F:-order(x::zppoly),
F:-random(),
F:-isPrimitiveElement(x::zppoly),
F:-PrimitiveElement(),
F:-zero,
F:-one,
F:-variable,
F:-size,
F:-factors(),
F:-extension,
```

which allow to do an advanced research on applications of Galois fields, but cogitation about them is not within the scope of this paper.

3. A system $gff(n^m)$ - a generalized finite field

It is possible to view Galois field from another angle. Now let n be an arbitrary integer ≥ 2, $m-$ an arbitrary integer ≥ 1, $f(x)-$ an arbitrary polynomial of degree m over the ring Z_n. Next let

$$gff(n^m) = \langle \mathsf{F}[x],\ +,\ \cdot \rangle, \tag{11}$$

be an algebraic system consisting of the set $\mathsf{F}[x]$ of all n^m polynomials of degree d , $0 \leqslant d \leqslant m-1$, 0 included, over the ring $\mathrm{Z_n}$ and of operations of addition and multiplication of these polynomials. Operations on elements of $gff(n^m)$ are performed nearly in the same manner as in $GF(p^m)$: addition over the ring $\mathrm{Z_n}$, multiplication over the same ring modulo polynomial $f(x)$.

It is easy to observe that $gff(n^m)$ fulfills the following set of axioms:

f1 The system $\langle \mathsf{F}[x],\ + \rangle$, is an abelian group.

f2 The system $\langle \mathsf{F}[x]^*,\ \cdot \rangle$ is an abelian quasigroupoid[✠],
$\mathsf{F}[x]^* = \mathsf{F}[x] \ \setminus \ \{0\}$, where 0 is an additive identity element.

[✠] The groupoid is an algebraic structure on a set with a binary operator. The only restriction on the operator is closure. It is assumed here that for the quasigroupoid a closure is not required.

f3 $\forall a,\ b,\ c\ \in \mathsf{F}[x]\ (a \cdot (b + c) = a \cdot b + a \cdot c) \wedge ((a + b) \cdot c = a \cdot c + b \cdot c)$.

The multiplicative system of $gff(n^m)$ is an abelian quasigroupoid $\langle \mathsf{F}[x]^*,\ \cdot \rangle$, which is not closed under multiplication, since if n is not a prime, then for some $a,\ b\ \in \mathsf{F}[x]$ the case $a \cdot b = 0$ may occur. Several properties of this quasigropupoid in [1] are described. For example, the elements of this quasigroupoid belong to two disjoint sets - a set of invertible elements and a set of non invertible elements. Any invertible element is a generator of cyclic group, being a subgroup of the groupoid. Furthermore one should know that if n is not a prime of if $f(x)$ is not irreducible then the system $gff(n^m)$ is not an integral domain. In this case the extended Euclidean algorithm for polynomials fails and cannot be able to determine all invertible elements in $gff(n^m)$.

After applying the mapping (8) to the system (11), taking into account that now $p = n$, we obtain the system

$$gff(n^m) = \langle \mathsf{F},\ +,\ \bullet \rangle, \tag{12}$$

the elements of which are numbers from the set $\{0,\ 1,\ \ldots,\ n^m - 1\}$. Such system is the most useful for cryptography.

Example 2. To familiarize the reader with some properties of $gff(n^m)$ having elements in the form of numbers the tables of operations in $gff(4^2)$ and in $gff(16)$ have been calculated and shown in Table 1 and Table 2. We may notice that multiplication on invertible elements is commutative and associative, so, an appropriate fragment of the multiplication table is a Latin square.

To the family of systems $gff(n^m)$ belongs a big class of algebraic structures. E.g. if n is a prime and $f(x)$ is not irreducible then the multiplicative structure of $gff(p^m)$ forms $SMG(p^m)$, if n is prime and $f(x)$ irreducible, $gff(n^m)$ becomes $GF(p^m)$. Thus, $gff(n^m)$, as a generalization of finite fields, may be called a generalized finite field. Although all properties of $gff(n^m)$ are not yet known, this algebraic structure will certainly be broadly applied, mainly in cryptography and coding.

4. A method of computing in $gff(n^m)$

While defining Galois field using Maple package one invokes the **GF** module with or without the third actual parameter, namely, without the irreducible polynomial. If we use this parameter, the polynomial must be absolutely

Table 1: Addition and multiplication tables in $gff(4^2)$ with $f(x) = x^2+x+3$ over Z[4]. The set of invertible elements: $\{1, 3, 4, 5, 6, 7, 9, 11, 12, 13, 14, 15\}$

+	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	2	3	0	5	6	7	4	9	10	11	8	13	14	15	12
2	2	3	0	1	6	7	4	5	10	11	8	9	14	15	12	13
3	3	0	1	2	7	4	5	6	11	8	9	10	15	12	13	14
4	4	5	6	7	8	9	10	11	12	13	14	15	0	1	2	3
5	5	6	7	4	9	10	11	8	13	14	15	12	1	2	3	0
6	6	7	4	5	10	11	8	9	14	15	12	13	2	3	0	1
7	7	4	5	6	11	8	9	10	15	12	13	14	3	0	1	2
8	8	9	10	11	12	13	14	15	0	1	2	3	4	5	6	7
9	9	10	11	8	13	14	15	12	1	2	3	0	5	6	7	4
10	10	11	8	9	14	15	12	13	2	3	0	1	6	7	4	5
11	11	8	9	10	15	12	13	14	3	0	1	2	7	4	5	6
12	12	13	14	15	0	1	2	3	4	5	6	7	8	9	10	11
13	13	14	15	12	1	2	3	0	5	6	7	4	9	10	11	8
14	14	15	12	13	2	3	0	1	6	7	4	5	10	11	8	9
15	15	12	13	14	3	0	1	2	7	4	5	6	11	8	9	10

•	0	1	3	4	5	6	7	9	11	12	13	14	15	2	8	10
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	3	4	5	6	7	9	11	12	13	14	15	2	8	10
3	0	3	1	12	15	14	13	11	9	4	7	6	5	2	8	10
4	0	4	12	13	1	5	9	14	6	7	11	15	3	8	10	2
5	0	5	15	1	6	11	12	7	13	3	4	9	14	10	2	8
6	0	6	14	5	11	13	3	12	4	15	1	7	9	8	10	2
7	0	7	13	9	12	3	6	5	15	11	14	1	4	10	2	8
9	0	9	11	14	7	12	5	1	3	6	15	4	13	2	8	10
11	0	11	9	6	13	4	15	3	1	14	5	12	7	2	8	10
12	0	12	4	7	3	15	11	6	14	13	9	5	1	8	10	2
13	0	13	7	11	4	1	14	15	5	9	6	3	12	10	2	8
14	0	14	6	15	9	7	1	4	12	5	3	13	11	8	10	2
15	0	15	5	3	14	9	4	13	7	1	12	11	6	10	2	8
2	0	2	2	8	10	8	10	2	2	8	10	8	10	0	0	0
8	0	8	8	10	2	10	2	8	8	10	2	10	2	0	0	0
10	0	10	10	2	8	2	8	10	10	2	8	2	8	0	0	0

Table 2: Addition and multiplication tables in $gff(16)$ with $f(x) = x$ over Z[16]. The set of invertible elements: $\{1, 3, 5, 7, 9, 11, 13, 15\}$

+	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0
2	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	1
3	3	4	5	6	7	8	9	10	11	12	13	14	15	0	1	2
4	4	5	6	7	8	9	10	11	12	13	14	15	0	1	2	3
5	5	6	7	8	9	10	11	12	13	14	15	0	1	2	3	4
6	6	7	8	9	10	11	12	13	14	15	0	1	2	3	4	5
7	7	8	9	10	11	12	13	14	15	0	1	2	3	4	5	6
8	8	9	10	11	12	13	14	15	0	1	2	3	4	5	6	7
9	9	10	11	12	13	14	15	0	1	2	3	4	5	6	7	8
10	10	11	12	13	14	15	0	1	2	3	4	5	6	7	8	9
11	11	12	13	14	15	0	1	2	3	4	5	6	7	8	9	10
12	12	13	14	15	0	1	2	3	4	5	6	7	8	9	10	11
13	13	14	15	0	1	2	3	4	5	6	7	8	9	10	11	12
14	14	15	0	1	2	3	4	5	6	7	8	9	10	11	12	13
15	15	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

•	0	1	3	5	7	9	11	13	15	2	4	6	8	10	12	14
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	3	5	7	9	11	13	15	2	4	6	8	10	12	14
3	0	3	9	15	5	11	1	7	13	6	12	2	8	14	4	10
5	0	5	15	9	3	13	7	1	11	10	4	14	8	2	12	6
7	0	7	5	3	1	15	13	11	9	14	12	10	8	6	4	2
9	0	9	11	13	15	1	3	5	7	2	4	6	8	10	12	14
11	0	11	1	7	13	3	9	15	5	6	12	2	8	14	4	10
13	0	13	7	1	11	5	15	9	3	10	4	14	8	2	12	6
15	0	15	13	11	9	7	5	3	1	14	12	10	8	6	4	2
2	0	2	6	10	14	2	6	10	14	4	8	12	0	4	8	12
4	0	4	12	4	12	4	12	4	12	8	0	8	0	8	0	8
6	0	6	2	14	10	6	2	14	10	12	8	4	0	12	8	4
8	0	8	8	8	8	8	8	8	8	0	0	0	0	0	0	0
10	0	10	14	2	6	10	14	2	6	4	8	12	0	4	8	12
12	0	12	4	12	4	12	4	12	4	8	0	8	0	8	0	8
14	0	14	10	6	2	14	10	6	2	12	8	4	0	12	8	4

irreducible, otherwise the module does not work. This means that the **GF** module is not suitable for computing in $gff(n^m)$, in which multiplication of its elements is taken over Z_n modulo an arbitrary polynomial $f(x)$ with coefficients from Z_n. In order to overcome this obstacle one should resort to the source on the basis of which the **GF** module has been built: the `modp1` function.

It may be said, without going into details, that operations in $gff(n^m)$ are performed by means of the function `modp1` according to the description given in Section 1. Using the function `modp1`, the author worked out the procedures for computing in $gff(n^m)$ and listed them in the Appendix C. These are the routines: `A_`, `S_`, `M_`, `D_`, `AI_`, `MI_` and `P_`, for performing addition, subtraction, multiplication, division, calculation of additive and multiplicative inverses and raising to a power in $gff(n^m)$, respectively. The routine `MI_` determines multiplicative inverses by means of extended Euclidean algorithm for polynomials (and usually cannot find all invertible elements). The routine `MIp_`, for computing multiplicative inverses using the multiplication operation and raising to a power in rather small $gff(n^m)$, is also listed.

To begin calculations one ought to invoke the procedure `INIT_` with determined actual parameters corresponding to formal parameters `nn` and `fx`, representing the modulus n and the polynomial $f(x)$, which defines multiplication. The procedure turns the actual parameter corresponding to the formal parameter `nn` into the global variable `n` and the actual parameter which replaces the formal parameter `fx` into the global variable `ext`. These two global variables are indispensable for assuring the operation of the remaining routines.

Here is an example of usage of these procedures:

Example 3. We will compute the expression

$$t = \frac{A\,B + A\,C + B\,C}{\frac{1}{A} - \frac{1}{B} + \frac{1}{C}}$$

in $gff(16^2)$ with $f(x) = x^2+2x+11$ for $A = 13,\ B = 254,\ C = 50,$. Then we calculate the same expression in $GF(2^8)$ when $f(x) = x^8+x^4+x^3+x+1$.

```
> INIT_(16, x^2 + 2*x + 11): A := 13: B := 24: C := 1:
>  t := D_(A_(M_(A, B), M_(A, C), M_(B, C)),
          A_(A_(MI_(A), AI_(MI_(B))), MI_(C)));
```

$$t := 60$$

```
> INIT_(2, x^8 + x^4 + x^3 + x + 1):
> t := D_(A_(M_(A, B), M_(A, C), M_(B, C)),
              A_(A_(MI_(A), AI_(MI_(B))), MI_(C)));
```

$$t := 184$$

The result of computations in $gff(16^2)$ has been achieved since there exist there multiplicative inverses for A, B and C. The element $27 \in gff(16^2)$ is not invertible, then if $A = 27$ the expression t will not be determined.

5. Conclusions

Apart from the discussion about using the **GF** Maple library package in application research, in the paper a new algebraic structure denoted as $gff(n^m)$ and named generalized finite field, has been defined. For the defined structure a complete set of routines for performing all possible operations on elements of $gff(n^m)$ has been presented. Since $gff(n^m)$ is a generalization of Galois field, the routines can be also used for doing arithmetic in finite fields and may stand in for the Maple **GF** module in the case of computing in huge fields, when this module is useless (i.e. when it is not able to factorize $p^m - 1$). The time of execution of any operation depends on the number of elements of $gff(n^m)$ and on the size of operands. If, for example, $n^m \approx 10^{30}$, 10^{300}, 10^{3000} and 10^{30000}, $x = \lceil n^m/2 \rceil$, $y = x$, $C(x, y)$ denotes an arbitrary binary or n-ary operation on elements $x,\ y \in gff(n^m)$, then the time of execution of one such operation equals approximately to 0.1 milliseconds, 0.3 milliseconds, 5 milliseconds and 250 milliseconds, respectively (Maple 9.5 on PC with the processor Pentium 4). The **GF** module gives similar results, but it has problems with computing in many fields of order higher than 10^{100}.

The generalized finite field, in comparison with Galois field, seems to be messy and defective. This feature ensures that $gff(n^m)$ will be used mainly for implementing transformations creating diffusion and confusion during the encryption process, and in random number and cryptographic key generators.

Apology and acknowledgment

The author feels obliged to state that not all the errors which were observed in his work [1] by Prof. A. D. Keedwell, have been corrected. It refers to "reversible" instead of a proper term "invertible". This mistake was unintentional and the author apologizes to Prof. A. D. Keedwell and to the readers.

The author also very much appreciates kind advice of Prof. Keedwell, concerning errors in the draft of this paper.

Appendix A

In this Appendix the routines `ax_`, `sx_`, `mx_`, `dx_`, `px_`, `mix_` and `aix_`, for doing addition, subtraction, multiplication, division, rising to a power, computing additive and multiplicative inverses in $GF(p^m)$, respectively, are listed. The procedures will work properly if we create an instance of p^m- element Galois field by means of the statement

```
> F := GF(p, m);
```

or

```
> F := GF(p, m, fx);
```

after previously defining actual parameters `p`, `m`, and, in the second statement, `fx`, which denote a prime, a positive integer and an irreducible polynomial of degree m over $GF(p)$. The routine `ax_` is $n-$ary, the routines `aix_` and `mix_` are unary and the remaining ones binary. The parameters of these routines are elements of $GF(p^m)$ in the form of polynomials and the routines return also the results as polynomials.

```
>  ax_ := proc()
 local i, s, ss;
     s := proc(a, b::polynom)
             F:-ConvertOut(
             F:-`+`(F:-ConvertIn(b), F:-ConvertIn(a)))
         end proc;
     ss := 0;
     for i to nargs do ss := s(ss, args[i]) end do;
     ss
```

```
 end proc:

> sx_ := proc(a, b::polynom)
     F:-ConvertOut(F:-`-`(F:-ConvertIn(b), F:-ConvertIn(a)))
 end proc:

>  mx_ := proc(a, b::polynom)
     F:-ConvertOut(F:-`*`(F:-ConvertIn(b), F:-ConvertIn(a)))
 end proc:

>  dx_ := proc(a, b::polynom)
     F:-ConvertOut(F:-`/`(F:-ConvertIn(b), F:-ConvertIn(a)))
 end proc:

> mix_ := proc(a::polynom)
      F:-ConvertOut(F:-inverse(F:-ConvertIn(a)))
 end proc:

> aix_ := proc(a::polynom)
      F:-ConvertOut(F:-`-`(F:-ConvertIn(a)))
 end proc:

> px_ := proc(a::polynom, k::integer)
      F:-ConvertOut(F:-`^`(F:-ConvertIn(a), k))
 end proc:
```

Appendix B

Similarly as in Appendix A, the routines `a_`, `s_`, `m_`, `p_`, `d_`, `ai_` and `mi_`, for doing addition, subtraction, multiplication, division, rising to a power, computing additive and multiplicative inverses in $GF(p^m)$, respectively, are listed here. The procedures will work properly if we create an instance of p^m- element Galois field by means of the statement

```
> F := GF(p, m);
```

or

```
> F := GF(p, m, fx);
```

after previously defining actual parameters `p`, `m`, and, in the second statement, `fx`, which denote a prime, a positive integer and an irreducible polynomial of degree m over $GF(p)$. The routine `a_` is $n-$ary, the routines `ai_` and `mi_` unary and the remaining ones binary. The parameters of these routines are elements of $GF(p^m)$ in the form of numbers from the set $\{0,\ 1,\ \ldots,\ p^m - 1\}$ and they return also the results as numbers from this set.

```
> a_ := proc()
  local i, s, ss;
     s := proc(a, b::nonnegint)
             F:-output(
             F:-`+`(F:-input(a), F:-input(b)))
         end proc;
     ss := 0;
     for i to nargs do ss := s(ss, args[i]) end do;
     ss
  end proc:

> s_ := proc(a, b::nonnegint)
     F:-output(F:-`-`(F:-input(a), F:-input(b)))
  end proc:

> m_ := proc(a, b::nonnegint)
     F:-output(F:-`*`(F:-input(a), F:-input(b)))
  end proc:

> d_ := proc(a, b::nonnegint)
     F:-output(F:-`/`(F:-input(a), F:-input(b)))
  end proc:

> ai_ := proc(a::nonnegint)
      F:-output(F:-`-`(F:-input(a)))
  end proc:

> mi_ := proc(a::nonnegint)
      F:-output(F:-inverse(F:-input(a)))
  end proc:
```

```
> p_ := proc(a::nonnegint, k::integer)
          F:-output(F:-‘^‘(F:-input(a), k))
  end proc:
```

Appendix C

In this Appendix the routines `INIT_`, `A_`, `S_`, `M_`, `P_`, `D_`, `AI_`, `MI_` and `MIp_`, for initializing computations and for doing addition, subtraction, multiplication, division, rising to a power, computing additive and multiplicative inverses in the generalized finite field $gff(n^m)$, respectively, are listed. The procedures doing computations in $gff(n^m)$ will work properly if we first execute the statement

```
> INIT_(nn, fx);
```

after previously defining actual parameters corresponding to the formal parameters `pn`, and `fx`, which denote an arbitrary positive integer and an arbitrary polynomial of degree m over the ring Z_n, respectively. This routine calculates the global variables $n = nn$ and ext, which represent the modulus n and the polynomial `fx` as the polynomial of type `zppoly`, respectively. These global variables are necessary for all routines doing arithmetic in $gff(n^m)$. The routine `A_` is $n-$ary, the routines `AI_`, `MI_` and `MIp_` are unary and the remaining ones binary. The parameters of these routines are elements of $gff(n^m)$ in the form of numbers from the set $\{0,\ 1,\ \ldots,\ n^m-1\}$ and they also return the results as numbers from this set.

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

> A_ := proc()
 local a, i, t;
     a := [args];
     for i to nargs do a[i] :=
         modp1(ConvertIn(convert(a[i], base, n), x), n)
     end do;
     t := modp1(ConvertOut(modp1('Add'(op(a)), n), u), n);
     subs(u = n, t)
```

```
 end proc:

> S_ := proc(a, b::nonnegint)
  local t, u;
      t := modp1(ConvertOut(modp1('Subtract'(
          modp1(ConvertIn(convert(a, base, n), x), n),
          modp1(ConvertIn(convert(b, base, n), x), n)), n)
          , u), n);
      subs(u = n, t)
  end proc:

> M_ := proc(a, b::nonnegint)
 local t, u;
     t := modp1(ConvertOut(modp1(Rem('Multiply'(
         modp1(ConvertIn(convert(a, base, n), x), n),
         modp1(ConvertIn(convert(b, base, n), x), n)), ext), n)
         , u), n);
     subs(u = n, t)
 end proc:

> P_ := proc(a::nonnegint, k::integer)
 local t, u;
     t := modp1(ConvertOut(modp1('Powmod'(
         modp1(ConvertIn(convert(a, base, n), x), n), k, ext),
         n), u), n);
     subs(u = n, t)
 end proc:

> D_ := proc(a::nonnegint, b::posint)
       M_(a, MI_(b))
 end proc:

> AI_ := proc(a::nonnegint)
  local t, u;
      t := modp1(ConvertOut(modp1(
          'Subtract'(modp1(ConvertIn(convert(a, base, n), x),
                     n)), n), u), n);
      subs(u = n, t)
```

```
  end proc:

> MI_ := proc(a::posint)
 local s, t;
     modp1('Gcdex'(modp1(ConvertIn(convert(a, base, n), x), n),
         ext, 's'), n);
     t := modp1(ConvertOut(s, x), n);
     subs(x = n, t)
 end proc:

> MIp_ := proc(a)
 local mi, k, mk, nn;
     mi := a;
     k := 0;
     nn := n^degree(modp1(ConvertOut(ext, x), n));
     if a = 1 then return 1 end if;
     while mi > 1 do
         k := k + 1;
         mi := M_(mi, a);
         if mi = 0 or k > nn - 1 then
             error "inverse does not exist"
         end if
     end do;
     P_(a, k)
 end proc:
```

References

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

[2] **R. Lidl, H. Niederreiter**: *Introduction to finite fields and their applications*, Cambridge University Press (1986).

[3] **A. J. Menezes, Editor**: *Applications of finite fields*, Kluwer Academic Publishers, (1993).

Received June 16, 2005

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