Int. J. Appl. Math. Comput. Sci., 2002, Vol.12, No.4, 559–569

GENERATING QUASIGROUPS FOR CRYPTOGRAPHIC APPLICATIONS

CZESŁAW KOŚCIELNY*

* University of Zielona Góra, Institute of Control and Computation Engineering
ul. Podgórna 50, 65–246 Zielona Góra, Poland
e-mail: C.Koscielny@issi.uz.zgora.pl

A method of generating a practically unlimited number of quasigroups of a (theoretically) arbitrary order using the computer algebra system Maple 7 is presented. This problem is crucial to cryptography and its solution permits to implement practical quasigroup-based endomorphic cryptosystems. The order of a quasigroup usually equals the number of characters of the alphabet used for recording both the plaintext and the ciphertext. From the practical viewpoint, the most important quasigroups are of order 256, suitable for a fast software encryption of messages written down in the universal ASCII code. That is exactly what this paper provides: fast and easy ways of generating quasigroups of order up to 256 and a little more.

Keywords: quasigroups, latin squares, stream-ciphers, cryptography

1. Introduction

According to eminent specialists (Dénes and Keedwell, 1999) the earliest use of quasigroups in cryptography is attributed to the German mathematician R. Shauffler, who reduced in his Ph.D. dissertation of 1948 the problem of breaking the Vigènere cipher to determining a particular latin square. In the above source, publications (Kościelny, 1995; 1996), concerning explicitly the application of quasigroups for construction stream and block ciphers, are also quoted as rather rare serious contemporary efforts to utilize the simplest non-associative algebraic systems in cryptography. In this context the work (Ritter, 1998) is by all means also worth noticing.

The paper presents a method of generating a practically unlimited number of quasigroups of an arbitrary order (practically ≤ 256) by means of the computer algebra system Maple 7, for applications to cryptography. The mathematical background knowledge, mandatory for understanding this article, is rather extensive. It is assumed that the reader is familiar with combinatorial structures and knows the basic properties of groups and Galois fields, and that he or she is acquainted with the basics of number theory. The only essential topics concerning latin squares and quasigroups are given in Section 2, since these mathematical constructions are not generally known. Section 3 briefly comments the obtained results, while the routines, implemented in Maple 7, for producing latin squares and quasigroups, are presented in Appendix.

2. Latin Squares and Quasigroups

A latin square of order n is an $n \times n$ array in which n^2 symbols, taken from a set A, are arranged so that each symbol occurs only once in each row and exactly once in each column. For any positive integer n there exists a latin square of order n. A latin square of order n for which

$$A = \{0, 1, \ldots, n-1\} \tag{1}$$

is said to be normalized or reduced if the elements of both its first row and its first column are in a natural order. For each $n \geq 2$ the total number $LS(n)$ of latin squares of order n (Laywine and Mullen, 1998) is given by

$$LS(n) = n!\,(n-1)!\,T(n), \tag{2}$$

where $T(n)$ denotes the number of reduced latin squares of order n.

The number of latin squares $LS(n)$ of order n increases very quickly with n and is indeed great, even for rather small n. It should be noted that the number of reduced latin squares is exactly known for $n \leq 10$ (McKay and Rogoyski, 1995) (see Table 1). For $n = 11, 12, \ldots, 15$, $T(n)$ is estimated in Table 2.

For $n > 15$ the bounds of $LS(n)$ can be calculated (Jacobson and Matthews, 1996) using the formula

$$\prod_{k=1}^{n} (k!)^{\frac{n}{k}} \geq LS(n) \geq \frac{(n!)^{2n}}{n^{n^2}}. \tag{3}$$

In Table 3 some upper and lower bounds of $LS(n)$ for several values of n, most frequently used in practice and obtained by means of (3), are given.

Table 1. Number of reduced latin squares $T(n)$ vs. n.

n	$T(n)$
2	1
3	1
4	4
5	56
6	9 048
7	16 942 080
8	535 281 401 585
9	377 597 570 964 258 816
10	7 580 721 483 160 132 811 489 280

Table 2. Estimates of $T(n)$ for $n = 11, 12, 13, 14, 15$.

n	$T(n)$
11	$5.36 \cdot 10^{33}$
12	$1.62 \cdot 10^{44}$
13	$2.51 \cdot 10^{56}$
14	$2.33 \cdot 10^{70}$
15	$1.50 \cdot 10^{86}$

Table 3. Estimates of $LS(n)$ for $n = 2^k$, $k = 4, 5, 6, 7, 8$.

$$0.689 \cdot 10^{138} \geq LS(16) \geq 0.101 \cdot 10^{119},$$
$$0.985 \cdot 10^{784} \geq LS(32) \geq 0.414 \cdot 10^{726},$$
$$0.176 \cdot 10^{4169} \geq LS(64) \geq 0.133 \cdot 10^{4008},$$
$$0.164 \cdot 10^{21091} \geq LS(128) \geq 0.337 \cdot 10^{20666},$$
$$0.753 \cdot 10^{102805} \geq LS(256) \geq 0.304 \cdot 10^{101724}.$$

As can be seen (Kościelny, 1995; 1997; Kościelny and Mullen, 1999), in the case of quasigroup-based stream ciphers, a latin square is an essential element for the implementation of both a key generator and the enciphering/deciphering procedure. Since the number of latin squares of order q (usually $32 \leq q \leq 256$), equal to the number of elements of an alphabet used in practice to represent the plain-text and the cryptogram, is immense, the key space is practically unlimited. Therefore, quasigroup-based ciphers are much more efficient and more secure than those based on regular algebraic systems such as fields or groups, or on deterministic algorithms (elliptic curve cryptosystems, conventional public-key cryptosystems).

It is justified to recall here that an algebraic system

$$\mathbf{Q} = \langle A, \bullet \rangle \tag{4}$$

consisting of a finite set of elements A in which a binary operation $\bullet$ is defined, is called a quasigroup if for any two elements $a, b \in A$ each of the equations

$$a \bullet x = b, \quad y \bullet a = b \tag{5}$$

has exactly one solution (Dénes and Keedwell, 1974). A quasigroup is then a simple algebraic system with one operation, which must be neither commutative nor associative, but, according to (5), the operation table in a quasigroup ought to be a latin square. A quasigroup

$$\mathbf{L} = \langle A, \bullet \rangle \tag{6}$$

in which there exists an identity element $e \in A$ with the property that

$$\forall\, x \in A \ [\, e \bullet x = x \bullet e = x \,] \tag{7}$$

is called a loop. If the operation $\bullet$ is associative and if

$$\forall a \in A \ \exists\, \tilde{a} \in A \ [\, a \bullet \tilde{a} = \tilde{a} \bullet a = e \,], \tag{8}$$

then the loop $\mathbf{L}$ becomes a group (the element $\tilde{a}$ is called the opposite of a or the inverse of a). Although the set A can have elements of any kind, it is assumed in this paper that its elements are non-negative integers, as in (1).

To obtain a multiplication table of an n-element quasigroup, one must have a suitably bordered latin square of order n. The bordering means that the latin square must be supplemented with the first row, which enumerates columns, and with the first column, which enumerates rows. If, e.g., the following latin square of order 8 is given:

$$LS = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 1 & 0 & 3 & 2 & 5 & 4 & 7 & 6 \\ 2 & 3 & 0 & 1 & 6 & 7 & 4 & 5 \\ 3 & 2 & 1 & 0 & 7 & 6 & 5 & 4 \\ 4 & 5 & 6 & 7 & 0 & 1 & 2 & 3 \\ 5 & 4 & 7 & 6 & 1 & 0 & 3 & 2 \\ 6 & 7 & 4 & 5 & 2 & 3 & 0 & 1 \\ 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \end{bmatrix},$$

then the multiplication table of the corresponding quasigroup $\langle \{0, 1, 2, 3, 4, 5, 6, 7\}, \bullet \rangle$ will be

$\bullet$	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	0	3	2	5	4	7	6
2	2	3	0	1	6	7	4	5
3	3	2	1	0	7	6	5	4
4	4	5	6	7	0	1	2	3
5	5	4	7	6	1	0	3	2
6	6	7	4	5	2	3	0	1
7	7	6	5	4	3	2	1	0

. (9)

In the case considered the operation table in the group isomorphic to an additive group of $GF(8)$ has been obtained: the system $\langle\{0,\,1,\,2,\,3,\,4,\,5,\,6,\,7\},\,\bullet\rangle$ is simultaneously a quasigroup, a loop and a group. Therefore any finite group provides a latin square to a cryptographer, who may easily transform it into many quasigroups, using the notion of isotopy explained below.

Let

$$\mathbf{Q}_p = \langle G,\,\circ\rangle \tag{10}$$

and

$$\mathbf{Q}_i = \langle H,\,\star\rangle \tag{11}$$

be two quasigroups with $|G| = |H|$. An ordered triple

$$(\pi_x,\ \pi_y,\ \pi_t) \tag{12}$$

of one-to-one mappings $\pi_x,\ \pi_y,\ \pi_t$ of the set G onto the set H is called an $isotopism$ of $\mathbf{Q}_p$ upon $\mathbf{Q}_i$ if

$$\pi_x(x) \star \pi_y(y) = \pi_t(x \circ y) \tag{13}$$

for all $x,\ y \in G$. It should be noted that the mapping π_t permutes the elements in the table of operations in a quasigroup $\mathbf{Q}_p$, while π_x and π_y operate on the elements of the row and column borders of this table, respectively. It is also said that $\mathbf{Q}_i$ is an $isotope$ of a primary quasigroup $\mathbf{Q}_p$. One can prove that the set of all isotopisms of a quasigroup of order n forms a group of order $(n!)^3$.

From (13) it follows that

$$X \star Y = \pi_t\Big(\pi_x^{-1}(X) \circ \pi_y^{-1}(Y)\Big) \tag{14}$$

for all $X,\ Y \in H$. Equation (14) allows us to operate on elements of $\mathbf{Q}_i$ if the table of operations in $\mathbf{Q}_p$ is known. It is evident that if $\pi_x = \pi_y = \pi_t$, then the algebraic systems (10) and (11) are isomorphic.

The isotopy is then a practically inexhaustible source of quasigroups, since without any problems we can construct many groups of an arbitrary finite order n, and thereby many latin squares, and, using (14), generate $(n!)^3$ quasigroups. For example, assume that $\mathbf{Q}_p = \langle\{0,\,1,\,2,\,3,\,4,\,5,\,6,\,7\},\,\bullet\rangle$ with the operation table given by (9). If

$$\pi_x^{-1} = \begin{pmatrix} 01234567 \\ 46275130 \end{pmatrix}, \quad \pi_y^{-1} = \begin{pmatrix} 01234567 \\ 47106532 \end{pmatrix},$$

$$\pi_t = \begin{pmatrix} 01234567 \\ 17246053 \end{pmatrix},$$

then, using (14), we obtain the quasigroup $\mathbf{Q}_p = \langle\{0,\,1,\,2,\,3,\,4,\,5,\,6,\,7\},\,\circ\rangle$ with the operation table as in (15):

$\circ$	0	1	2	3	4	5	6	7
0	1	4	0	6	2	7	3	5
1	2	7	3	5	1	4	0	6
2	5	0	4	2	6	3	7	1
3	4	1	5	3	7	2	6	0
4	7	2	6	0	4	1	5	3
5	0	5	1	7	3	6	2	4
6	3	6	2	4	0	5	1	7
7	6	3	7	1	5	0	4	2

. (15)

The reader can easily verify (assuming that the first operand from the left is the number of a row and the second operand from the left is the number of a column) that $(0 \circ 2) \circ 3 = 6,\ 0 \circ (2 \circ 3) = 0$, which means that the operation $\circ$ is not associative. Then Q_i is a real noncommutative quasigroup, since the operation table (15) is not symmetrical with respect to the main diagonal.

The notion of the *simple product of quasigroups* is also very important for application cryptographers since, if two quasigroups are given, it permits to construct a quasigroup of the order equal to the product of the orders of these quasigroups.

Let

$$\begin{aligned} &\mathbf{Q}_1 = \langle R,\,\bullet\rangle, \quad \mathbf{Q}_2 = \langle S\,\circ\rangle, \\ &R = \{0,\,1,\,2,\,\ldots,\,n_1 - 1\}, \\ &S = \{0,\,1,\,2,\,\ldots,\,n_2 - 1\} \end{aligned} \tag{16}$$

be two arbitrary quasigroups. Then the simple product $\mathbf{Q}$ of these quasigroups is the algebraic system

$$\mathbf{Q} = \mathbf{Q}_1 \times \mathbf{Q}_2 = \langle T,\,\star\rangle, \tag{17}$$

where $T = \{0,\,1,\,\ldots,\,n_1 n_2 - 1\}$. To define the operation $\star$ in the set T, let us first assume that

$$T_{cp} = R \times S = \{t_0,\,t_1,\,\ldots,\,t_{n_1 n_2 - 1}\} \tag{18}$$

is the Cartesian product of the sets R and S. Further, let

$$\begin{aligned} &t_i = (r_i, s_i), \quad t_k = (r_k,\,s_k), \\ &i,\,k \in T, \quad r_i,\,r_k \in R, \quad s_i,\,s_k \in S. \end{aligned} \tag{19}$$

Noting that the quasigroup with elements belonging to the set T_{cp}

$$\mathbf{Q}_{cp} = \langle T_{cp},\,*\,\rangle \tag{20}$$

is the simple product of quasigroups (16), we can define the operation $\star$ as follows:

$$t_i * t_k = \big((r_i \bullet r_k),\,(s_i \circ s_k)\big). \tag{21}$$

Table 4. Operation table for the quasigroup $\mathbf{Q}_{cp} = \langle T_{cp}, * \rangle = \mathbf{Q}_1 \times \mathbf{Q}_2$.

$*$	$(0,0)$	$(0,1)$	$(0,2)$	$(1,0)$	$(1,1)$	$(1,2)$	$(2,0)$	$(2,1)$	$(2,2)$	$(3,0)$	$(3,1)$	$(3,2)$
$(0,0)$	$(3,1)$	$(3,2)$	$(3,0)$	$(0,1)$	$(0,2)$	$(0,0)$	$(1,1)$	$(1,2)$	$(1,0)$	$(2,1)$	$(2,2)$	$(2,0)$
$(0,1)$	$(3,0)$	$(3,1)$	$(3,2)$	$(0,0)$	$(0,1)$	$(0,2)$	$(1,0)$	$(1,1)$	$(1,2)$	$(2,0)$	$(2,1)$	$(2,2)$
$(0,2)$	$(3,2)$	$(3,0)$	$(3,1)$	$(0,2)$	$(0,0)$	$(0,1)$	$(1,2)$	$(1,0)$	$(1,1)$	$(2,2)$	$(2,0)$	$(2,1)$
$(1,0)$	$(2,1)$	$(2,2)$	$(2,0)$	$(1,1)$	$(1,2)$	$(1,0)$	$(0,1)$	$(0,2)$	$(0,0)$	$(3,1)$	$(3,2)$	$(3,0)$
$(1,1)$	$(2,0)$	$(2,1)$	$(2,2)$	$(1,0)$	$(1,1)$	$(1,2)$	$(0,0)$	$(0,1)$	$(0,2)$	$(3,0)$	$(3,1)$	$(3,2)$
$(1,2)$	$(2,2)$	$(2,0)$	$(2,1)$	$(1,2)$	$(1,0)$	$(1,1)$	$(0,2)$	$(0,0)$	$(0,1)$	$(3,2)$	$(3,0)$	$(3,1)$
$(2,0)$	$(1,1)$	$(1,2)$	$(1,0)$	$(3,1)$	$(3,2)$	$(3,0)$	$(2,1)$	$(2,2)$	$(2,0)$	$(0,1)$	$(0,2)$	$(0,0)$
$(2,1)$	$(1,0)$	$(1,1)$	$(1,2)$	$(3,0)$	$(3,1)$	$(3,2)$	$(2,0)$	$(2,1)$	$(2,2)$	$(0,0)$	$(0,1)$	$(0,2)$
$(2,2)$	$(1,2)$	$(1,0)$	$(1,1)$	$(3,2)$	$(3,0)$	$(3,1)$	$(2,2)$	$(2,0)$	$(2,1)$	$(0,2)$	$(0,0)$	$(0,1)$
$(3,0)$	$(0,1)$	$(0,2)$	$(0,0)$	$(2,1)$	$(2,2)$	$(2,0)$	$(3,1)$	$(3,2)$	$(3,0)$	$(1,1)$	$(1,2)$	$(1,0)$
$(3,1)$	$(0,0)$	$(0,1)$	$(0,2)$	$(2,0)$	$(2,1)$	$(2,2)$	$(3,0)$	$(3,1)$	$(3,2)$	$(1,0)$	$(1,1)$	$(1,2)$
$(3,2)$	$(0,2)$	$(0,0)$	$(0,1)$	$(2,2)$	$(2,0)$	$(2,1)$	$(3,2)$	$(3,0)$	$(3,1)$	$(1,2)$	$(1,0)$	$(1,1)$

But we want to represent the simple product of quasigroups (16) as a quasigroup (17). To this end, we must convert the set T_{cp} into the set T. There are many ways of doing such a conversion. The mapping

$$\lambda : t_x \to X, \quad t_x \in T_{cp}, \quad X,\ x \in T, \tag{22}$$

defined by the function

$$\lambda(t_x) = \lambda(r_x, s_x) = n_2 r_x + s_x, \tag{23}$$

$$\lambda^{-1}(X) = t_x = \big(\text{iquo}(X,\, n_2),\, X \bmod n_2\big), \tag{24}$$

gives one of the simplest solutions, since taking account of (21)–(24), we have

$$X \star Y = \lambda\big(\lambda^{-1}(X) * \lambda^{-1}(Y)\big). \tag{25}$$

Finally,

$$X \star Y = n_2\big(\text{iquo}(X,\, n_2) \bullet \text{iquo}(Y,\, n_2)\big) + (X \bmod n_2) \circ (Y \bmod n_2). \tag{26}$$

In (24) and (26) $X, Y \in T$, $\text{iquo}(m, n)$ computes the integer quotient of m divided by n, operators +, mod and multiplication by n_2 denote the usual arithmetic operations, while $\bullet$ and $\circ$ are operations in quasigroups Q_1 and Q_2, respectively.

To explain in detail the calculation of the simple product of quasigroups, assume that

$$\mathbf{Q}_1 = \langle R, \bullet \rangle, \quad \mathbf{Q}_2 = \langle S \circ \rangle,$$
$$R = \{0, 1, 2, 3\}, \quad S = \{0, 1, 2\}$$

and

$\bullet$	0	1	2	3
0	3	0	1	2
1	2	1	0	3
2	1	3	2	0
3	0	2	3	1

$\circ$	0	1	2
0	1	2	0
1	0	1	2
2	2	0	1

.

In this case

$$T_{cp} = \big\{(0,0), (0,1), (0,2), (1,0), (1,1), (1,2), (2,0), (2,1), (2,2), (3,0), (3,1), (3,2)\big\}$$

and the quasigroup $\mathbf{Q}_{cp} = \langle T_{cp}, * \rangle = \mathbf{Q}_1 \times \mathbf{Q}_2$ has the operation table given in Tab. 4.

Thus, after the application of the mapping (22), the operation table in the quasigroup $\mathbf{Q} = \mathbf{Q}_1 \times \mathbf{Q}_2 = \langle T, \star \rangle$ is given by Tab. 5.

Table 5. Operation table for the quasigroup $\mathbf{Q} = \langle T, \star \rangle = \mathbf{Q}_1 \times \mathbf{Q}_2$.

$\star$	0	1	2	3	4	5	6	7	8	9	10	11
0	10	11	9	1	2	0	4	5	3	7	8	6
1	9	10	11	0	1	2	3	4	5	6	7	8
2	11	9	10	2	0	1	5	3	4	8	6	7
3	7	8	6	4	5	3	1	2	0	10	11	9
4	6	7	8	3	4	5	0	1	2	9	10	11
5	8	6	7	5	3	4	2	0	1	11	9	10
6	4	5	3	10	11	9	7	8	6	1	2	0
7	3	4	5	9	10	11	6	7	8	0	1	2
8	5	3	4	11	9	10	8	6	7	2	0	1
9	1	2	0	7	8	6	10	11	9	4	5	3
10	0	1	2	6	7	8	9	10	11	3	4	5
11	2	0	1	8	6	7	11	9	10	5	3	4

where

$$T = \{0, 1, \ldots, 11\}.$$

It should be noted that the simple product of two quasigroups is not, in general, commutative, and that it can be easily generalized to an arbitrary number of quasigroups.

To generate quasigroups of an arbitrary order, a powerful software tool for doing the mathematics is needed. For that purpose the computer algebra system Maple 7 can be used, since this software offers, among other things, powerful and effective integer arithmetic and operations in the domain of univariate polynomials over the integers modulo n, allowing computations in the finite field $GF(p^m)$. There are also other useful Maple packages, defining all number theory functions and routines, returning not only random permutations, but also all the elements of the combinatorial class of required size. Furthermore, each algorithm can be intelligibly and concisely written down by means the Maple interpreter, easily comprehensible by any mathematician and programmer. But as regards quasigroups, Maple is very poor: the only package MOLS returns a list of up to $p^m - 1$ mutually orthogonal latin squares of order p^m, which are, in fact, interiors of tables of addition in $GF(p^m)$ with permuted rows. Therefore, in Appendix several useful algorithms for generating quasigroups, written in the Maple 7 interpreter, are presented.

3. Conclusions

The problem concerning the construction of quasigroups, important for the practice of quasigroup-based ciphers, has been resolved. The Maple 7 routines presented in Appendix can be used directly for generating quasigroups, while two routines are auxiliary: the first computes Zech's logarithms needed for constructing addition and subtraction tables in $SGF(q)$, the other can write an arbitrary latin square into a text file. All the routines can be used as tools for designing both the encrypting and decrypting procedures of quasigroup-based stream-ciphers, and the key generators for such ciphers as well.

If, in some applications, the presented routines seemed to be slow, one can implement them in other programming languages (e.g. in C, C++, Fortran 95, etc.), equipped with the appropriate libraries for doing advanced mathematics. The execution time can then be shortened even up to 100 and more times.

It should also be noted that the routine `fsgfz` can be considerably optimized and, for applications, it should be written in a high level programming language rather than in the Maple interpreter. To do it, one must be familiar with the properties of Zech's logarithms in $SGF(q)$ (Kościelny, 1995).

References

Dénes J. and Keedwell A.D. (1974): *Latin Squares and Their Applications*. — Budapest: Akadémiai Kiadó.

Dénes J. and Keedwell A.D. (1999): *Some applications of non-associative algebraic systems in cryptology*. — Techn. Rep. 99/03, Dept. Math. Stat., University of Surrey.

Jacobson M.T. and Matthews P. (1996): *Generating uniformly distributed random latin squares*. — J. Combinat. Desig., Vol. 4, No. 6, pp. 405–437.

Kościelny C. (1995): *Spurious Galois fields*. — Appl. Math. Comp. Sci., Vol. 5, No. 1, pp. 169–188.

Kościelny C. (1996): *A method of constructing quasigroup-based stream-ciphers*. — Appl. Math. Comp. Sci., Vol. 6, No. 1, pp. 109–121.

Kościelny C. (1997): NLPN *sequences over* $GF(q)$. — Quasigr. Related Syst., No. 4, pp. 89–102.

Kościelny C. and Mullen G.L. (1999): *A quasigroup-based public-key cryptosystem*. — Int. J. Appl. Math. Comp. Sci., Vol. 9, No. 4, pp. 955–963.

Laywine C.F. and Mullen G.L. (1998): *Discrete Mathematics Using Latin Squares*. — New York: Wiley.

McKay B. and Rogoyski E. (1995): *Latin Squares od Order 10*. — Electr. J. Combinat., Vol. 2, No. 3.

Ritter T. (1998): *Latin squares: A literature survey—Research comments from ciphers by Ritter*. — Available at `http://www.io.com/~ritter/RES/LATSQR.HTM`

Appendix

Maple 7 Routines for Generating Latin Squares

All the names of the routines presented here, generating directly latin squares, begin with the letters `ls`, which means *latin square*.

1. LSs Isomorphic with the Interior of the Operation Table of a Cyclic Group, of a Multiplicative Group of $GF(p^m)$, and of an Additive Group of $GF(p^m)$

The first one-argument routine `lscg` (`cg` denoting *cyclic group*) computes the interior of the table of addition modulo n and returns a latin square in the form of an $n \times n$ matrix:

```
>  lscg := proc(q::posint)
    local i, j, ls;
    ls := array(1 .. q, 1 .. q);
    for i to q do
        for j to q do ls[i, j] := (i + j - 2) mod q end do
    end do;
    evalm(ls)
end proc:
```

The procedure `lscg` can be called with an arbitrary actual argument q.

As is known, the order of a multiplicative group of $GF(p^m)$ equals $p^m - 1$. Therefore, using a routine which generates a latin square of order q, isomorphic with the interior of the multiplicative group of $GF(p^m)$, we must remember that the condition $q + 1 = p^m$ must be satisfied.

The one-argument routine `lsmgff` (`mgff` stands for *multiplicative group of a finite field*) uses the GF package. This routine generates also a latin square of the order $q = 256$, which is important for practice since, in this case, $q + 1$ is a prime.

```
> lsmgff := proc(q::posint)
   local i, j, ls, m, p, w, g;
   w := ifactor(q + 1);
   p := op(convert(w, list)[1]);
   if nops(w) = 1 then m := 1 else m := op(op(w)[2]) end if;
   if p^m <> q + 1 then error "q+1 must be a power of prime"
   end if;
   g := GF(p, m);
   ls := array(1 .. q, 1 .. q);
   for i to q do for j to q do ls[i, j] :=
           g:-output(g:-‘*‘(g:-input(i), g:-input(j))) - 1
       end do
   end do;
   evalm(ls)
end proc:
```

Similarly to the previous procedure, the two-argument routine `lsagff` (`agff` stands for *additive group of a finite field*) makes use of the GF package. Thus, the order of the returned latin square, isomorphic to the operation table in the additive group of $GF(p^m)$, is equal to the power of the prime p^m.

```
>  lsagff := proc(p::prime, m::posint)
    local ls, g, i, j, n;
    g := GF(p, m);
    n := p^m;
    ls := array(1 .. n, 1 .. n);
    for i from 0 to n - 1 do for j from 0 to n - 1 do
            ls[i + 1, j + 1] :=
            g:-output(g:-‘+‘(g:-input(i), g:-input(j)))
        end do
    end do;
    evalm(ls)
end proc:
```

A2. Two LSs Isomorphic with the Interior of Addition and Subtraction Tables of a Spurious Galois Field

Each $SGF(q)$ can provide two latin squares being the interiors of addition and subtraction tables (it is clear, however, that the table of addition in $GF(q)$ of characteristic 2 does not differ from the subtraction table in this field).

To determine addition and subtraction tables in $SGF(q)$, the appropriate Zech logarithm is needed. It can be computed using the two-argument routine `fsgfz` (`f` for file, `sgf` for spurious Galois field, `z` for Zech's logarithm). The routine implements an algorithm similar to that in (Kościelny, 1995, p. 179) and returns all values of Zech's logarithms for $SGF(q)$. The result of calculation is written in the text file named `fn`. Each row of the file contains $Z(x),\ x = 1, 2, \ldots, q-2$ if q is even, or $Z(x),\ x = 0, 1, 2, \ldots, (q-3)/2, (q+1)/2, \ldots, q-2$ if q is odd.

```
>  fsgfz := proc(q::posint, fn::string)
   local i, k, b, f, nd, nz, p, z, za, zs, q2, n, nc, q1, c, ip, z2f;
    q1 := q - 1;
    q2 := q - 2;
    f := fopen(fn, WRITE);
    nd := convert(nops(convert(q1, base, 10)), string);
    if q mod 2 = 0 then n := 1/2*q2 else n := 1/2*q1 end if;
    z := array(1 .. q2);
    with(combstruct);
    c := iterstructs(Combination(q2), size = n);
    nz := 0;
    for i to count(Combination(q2), size = n) do
        p := convert(nextstruct(c), list);
        ip := iterstructs(Permutation(p));
        while not finished(ip) do
            za := nextstruct(ip);
            b := false;
            if q mod 2 = 0 then
                for k to n do b := b or za[k] = k end do;
                if not b then
                    for k to n do
                        z[k] := za[k];
                        z[q1 - k] := (z[k] - k + q1) mod q1
                    end do;
                    zs := convert(z, set);
                    if nops(zs) = q2 and not member(0, zs)
                    then
                        nz := nz + 1;
                        for k to q2 do fprintf(f, cat("%", nd, "d "), z[k])
                        end do;
                        fprintf(f, "\n")
                    end if
                end if
            else
                for k to n - 1 do b := b or za[k] = k - 1
                end do;
                if not b then
                    for k to n do z[k] := za[k] end do;
                    for k to n - 1 do z[q2 - k + 1] :=
                        (za[k + 1] - k + q1) mod q1
                    end do;
                    zs := convert(z, set);
                    if nops(zs) = q2 and not member(0, zs)
                    then
```

```
                        nz := nz + 1;
                        for k to q2 do fprintf(f, cat("%", nd, "d "), z[k])
                        end do;
                        fprintf(f, "\n")
                    end if
                end if
            end if
        end do
    end do;
    fprintf(f, "%s%d%s %d", "Number of sgf(", q, ") equals to", nz);
    fclose(f)
end proc:
```

Calling the routine as in the following statements:

```
> fsgfz(9,"d:\\latsqr\\z09"); fsgfz(10,"d:\\latsqr\\z10");
```

will create two text files, named `z09` and `z10`, situated on the disc **d:** in the directory **latsqr** (this directory must exist on the disc **d:**).

After computing the set of all Zech's logarithms for a desired q, we can generate two latin squares of order q for any element of this set, using the procedure `lssgf`. The second formal parameter of this routine, `z`, is a list containing the values of an arbitrary row of the file, created by the routine `fsgfz`.

```
>  lssgf := proc(q::posint, z::list)
    local q1, q2, zech, i, k, m, ni, S, lsa, lsb;
    S := proc(x, y::nonnegint)
        local d, mn;
            if x = 0 or y = 0 then return x + y
            elif x <= y then mn := x
            else mn := y
            end if;
            d := abs(x - y);
            if d = ni then return 0
            else return 1 + ((mn - 1 + zech[d]) mod q1)
            end if
        end proc;
    q1 := q - 1;
    q2 := q1 - 1;
    lsa := array(1 .. q, 1 .. q);
    lsb := array(1 .. q, 1 .. q);
    zech := array(0 .. q2);
    zech[0] := 1;
    if q mod 2 = 0 then
        ni := 0; for i to q2 do zech[i] := z[i] end do
    else
        ni := 1/2*q1;
        for i to ni do zech[i - 1] := z[i] end do;
        for i from ni + 1 to q2 do zech[i] := z[i] end do
    end if;
    zech[ni] := 0;
    for i to q do
        for k to q do lsa[i, k] := S(i - 1, k - 1) end do
    end do;
    for i to q do for k to q do for m to q do
                if lsa[i, k] = m - 1 then lsb[m, k] := i - 1
```

```
                end if
            end do
        end do
    end do;
    evalm(lsa), evalm(lsb)
end proc:
```

The algorithm according to which the procedure `fsgfz` is implemented is not the fastest one. Therefore, it practically operates on the value of the actual parameter $q < 20$. But it has been observed (Kościelny, 1996) that the discrete function $Z(2x-1) = q/2 - x + 1,\ Z(2x) = x,\ x = 1, 2, \ldots, (q-2)/2$ satisfies the conditions for Zech's logarithm of $SGF(q)$ for an arbitrary even $q \geq 6$.

The above is taken into account by the routine `lspsgf` (`psgf` stands for *particular sgf*), having one argument and operating exactly in the same manner as the procedure `lssgf`.

```
>  lspsgf := proc(q::posint)
   local S, i, k, m, a, b, q1;
    if q mod 2 = 1 or q < 6 then
        error "q must be an even integer > 5"
    end if;
    S := proc(x, y)
        local d, mn, z;
            if x = 0 or y = 0 then return x + y end if;
            if x <= y then mn := x else mn := y end if;
            d := abs(x - y);
            if d mod 2 = 0 then z := 1/2*d
            else z := 1/2*q + 1/2*d - 1/2
            end if;
            mn := mn - 1;
            if d = 0 then return 0
            else return 1 + ((mn + z) mod q1)
            end if
        end proc;
    a := array(1 .. q, 1 .. q);
    b := array(1 .. q, 1 .. q);
    q1 := q - 1;
    for i to q do
        for k to q do a[i, k] := S(i - 1, k - 1) end do
    end do;
    for i to q do for k to q do for m to q do
                if a[i, k] = m - 1 then b[m, k] := i - 1
                end if
            end do
        end do
    end do;
    evalm(a), evalm(b)
end proc:
```

A3. Routine for Computing the Simple Product of an Arbitrary Number of Quasigroups

The routine `lsspr` (`spr` corresponds to *simple product*) with an arbitrary number of arguments can be applied to generating a simple product of an arbitrary number of quasigroups, equal to the number of arguments passed in the procedure call. It contains a local procedure `sp2`, which is based exactly on the formulae (16)–(26). The procedure `sp2` computes the simple product of two arbitrary quasigroups and its use by the main routine depends on the number of actual parameters in the call to the procedure `lsspr`.

```
>  lsspr := proc()
    local sp2, i, ltsq;
    sp2 := proc(a::matrix, b::matrix)
        local i, k, cp, n1, n2, n12, ls;
            n1 := linalg[rowdim](a);
            n2 := linalg[rowdim](b);
            n12 := n1*n2;
            cp := array(1 .. n12);
            ls := array(1 .. n12, 1 .. n12);
            for i from 0 to n12 - 1 do
                cp[i + 1] := [iquo(i, n2), i mod n2]
            end do;
            for i to n12 do for k to n12 do ls[i, k] :=
                    n2*a[cp[i][1] + 1, cp[k][1] + 1]
                     + b[cp[i][2] + 1, cp[k][2] + 1]
                end do
            end do;
            evalm(ls)
        end proc;
    ltsq := args[1];
    if nargs = 1 then return args[1] end if;
    for i from 2 to nargs do ltsq := sp2(ltsq, args[i])
    end do
end proc:
```

For example, the call

```
> lsspr(lsmgff(4),lscg(2), lsagff(2,1));
```

will compute the product of three quasigroups of orders 4, 2 and 2.

A4. Routine for Determining the Isotope of an Arbitrary Quasigroup

The one-argument procedure `lsis` (`is` stands for *isotope*) transforms any latin square into the interior of an operation table in a quasigroup, according to (14). The required three permutations are generated at random by means of the `combinat` package.

```
>   lsis := proc(ls::matrix)
     local i, k, n, pix_, piy_, pit, q;
     n := linalg[rowdim](ls);
     q := array(1 .. n, 1 .. n);
     pix_ := combstruct[draw](Permutation(n));
     piy_ := combstruct[draw](Permutation(n));
     pit :=  combstruct[draw](Permutation(n));
     for i to n do for k to n do
             q[i, k] := pit[ls[pix_[i], piy_[k]] + 1] - 1
         end do
     end do;
     evalm(q)
end proc;
```

Since the permutations `pix_`, `piy_` and `pit` are generated at random, any call, repeated several times, may give different quasigroups. In order to obtain the same result in any call, a seed for the random number generator must be set. For example, by repeating the instructions

```
>  _seed := 142857: lsis(lsspr(lsmgff(4), lscg(4)));
```

the reader will obtain every time the same result.

A5. Routine for Writing LS to a File

The implementation of quasigroup-based cryptographic systems is generally done in languages other than the Maple interpreter. In this case a quasigroup, determined by Maple, ought to be written to a file and then imported into other encrypting/decrypting software. To achieve this, we must modify the procedure `lsqg`. Of course, there are many ways of doing it. For example, the procedure `flsqg` computes the same quasigroup as the routine `lsqg`, but, instead of returning a computed quasigroup, it writes it to a text file named `lsq` and created in the current directory.

```
  fls := proc(ls::matrix, fn::string)
local f, i, k, n, nc;
    n := linalg[rowdim](ls);
    nc := convert(nops(convert(n, base, 10)), string);
    f := fopen(fn, WRITE);
    for i to n do
        for k to n do
            fprintf(f, cat("%", nc, "d "), ls[i, k])
        end do;
        fprintf(f, "\n")
    end do;
    fclose(f)
end proc:

> _seed := 13: fls(lsis(lspsgf(16)[2]),"d:\\latsqr\\q16");
```

Finally, from among plenty of statements using the routine `fls`, cooperating with the others procedures of this section in order to create a file containing the interior of the operation table in a quasigroup of order 256, several examples are given:

```
> fls(lsis(lsspr(lspsgf(16)[1], lspsgf(16)[2])), "d:\\latsqr\\q256a");
> fls(lsspr(lsis(lsmgff(16)), lsis(lsagff(2,4))),"d:\\latsqr\\q256b");
> fls(lsis(lsspr(lsis(lspsgf(8)[2]), lsis(lsagff(2,5)))),"d:\\latsqr\\q256c");
> fls(lsis(lsspr(lsagff(2,5), lsagff(2,3))),"d:\\latsqr\\q256d");
> fls(lsis(lsspr(lspsgf(16)[2], lscg(4), lsis(lsmgff(4)))),"d:\\latsqr\\q256e");
> fls(lsis(lsspr(lspsgf(64)[1], lsis(lsagff(2,2)))),"d:\\latsqr\\q256f");
> fls(lsis(lsspr(lsis(lsagff(2,3)), lspsgf(32)[2])),"d:\\latsqr\\q256g");
> fls(lsis(lsspr(lspsgf(8)[1], lspsgf(8)[2],lscg(4))),"d:\\latsqr\\q256h");
> fls(lsis(lsagff(2,8)),"d:\\latsqr\\q256i");
> fls(lsis(lspsgf(256)[2]),"d:\\latsqr\\q256j");
> fls(lsis(lsmgff(256)),"d:\\latsqr\\q256k");
> fls(lsis(lsspr(lspsgf(16)[1],lspsgf(16)[1])),"d:\\latsqr\\q256l");
> fls(lsis(lsspr(lsmgff(16),lsmgff(16))),"d:\\latsqr\\q256m");
> fls(lsis(lscg(256)),"d:\\latsqr\\q256n");
> fls(lsis(lsspr(lsagff(2,7),lsmgff(2))),"d:\\latsqr\\q256o");
> fls(lsis(lsspr(lspsgf(32)[1],lscg(8))),"d:\\latsqr\\q256p");
```

As can be seen, all the procedures presented here provide poweful means for generating quasigroups.

Received: 26 April 2001
Revised: 21 January 2002
Re-revised: 1 March 2002