Quasigroups and Related Systems **19** (2011), 301 – 316

On random error correcting codes based on quasigroups

Aleksandra Popovska-Mitrovikj, Verica Bakeva and Smile Markovski

Abstract. Random error-correcting codes based on quasigroups transformations are proposed elsewhere. They are similar to convolution codes and the dependence of the properties of the codes from the used quasigroups are investigated in earlier paper of ours. In this paper we compare the Random error correcting codes based on quasigroups with the well know Reed-Muller and Reed-Solomon codes. The obtained experimental results show that in the case when the bit-error probability of binary symmetric channel is $p > 0.05$ ($p > 0.06$) then the random codes based on quasigroups over perform the Reed-Muller and Reed-Solomon codes for the packet-error probability (for the bit-error probability).

1. Introduction

A new class of codes, Random codes based on quasigroups (RCBQ), are proposed by Gligoroski et al [4]. In RCBQ, similar to recursive convolution codes, the correlation exists between any two bits of a codeword, and they can have infinite length, theoretically. However, in contrast to convolution codes, RCBQ are nonlinear and almost random.

RCBQ have several parameters, and we have investigated the influence of the code parameters to the code performances [8]. Since RCBQ are designed using quasigroup string transformations on messages extended by introduced redundancy, we have investigated how the following parameters affect the codes: the pattern of redundancy, the chosen quasigroups, the number of application of quasigroup transformations. The main goal of this paper is to compare the performances of RCBQ regarding the performances of the Reed-Muller codes (RMC) and Reed-Solomon codes (RSC). For that aim we have chosen an RCBQ with best performances.

2010 Mathematics Subject Classification: 20N05, 68P30, 94B60
Keywords: Quasigroup, quasigroup transformation, error-correcting code, random code, Reed-Muller code, Reed-Solomon code, packet-error and bit-error probability

The paper is organized as follows. The Section 2 contains the definition of quasigroup transformations and the definition of TASC (Totally Asynchronous Stream Ciphers) that is used for code definition. A description of the code, i.e., the algorithms for coding and for decoding, are given in Section 3. In Section 4 the definitions of codes RMC and RSC are given. In Section 5 we show how optimal parameters for RCBQ can be chosen. The comparison results for the performances of RCBQ regarding RMC and RSC are presented in Section 6, which is the section with the main results in this paper. Section 7 contains some conclusions.

2. Quasigroup transformation and TASC

A *quasigroup* $(Q, *)$ is a groupoid, i.e., a set Q with a binary operation $* : Q^2 \longrightarrow Q$, such that for all $u, v \in Q$, there exist unique $x, y \in Q$, satisfying the equalities $u * x = v$ and $y * u = v$. Further on, we assume that the set Q is a finite set.

Given a quasigroup $(Q, *)$, a new operation "$\backslash$", called a *parastrophe*, can be derived from the operation $*$ as follows:

$$x * y = z \iff y = x \setminus z.$$

Then the algebra $(Q, *, \backslash)$ satisfies the identities: $x \setminus (x * y) = y$ and $x * (x \setminus y) = y$, and $(Q, \backslash)$ is also a quasigroup.

Quasigroup string transformations are defined on a finite set Q (i.e., an alphabet Q) endowed with a quasigroup operation $*$, and they are mappings from Q^+ to Q^+, where Q^+ is the set of all nonempty words on Q. Note that $Q^+ = Q \cup Q^2 \cup Q^3 \cup \dots$. Here, we use two types of quasigroup transformations as explained below.

Let $l \in Q$ be a fixed element, called a leader. For every $a_i, b_i \in Q$, e- and d-transformations are defined as follows.

$$e_l(a_1 a_2 \dots a_n) = b_1 b_2 \dots b_n \Leftrightarrow b_{i+1} = b_i * a_{i+1},$$
$$d_l(a_1 a_2 \dots a_n) = b_1 b_2 \dots b_n \Leftrightarrow b_{i+1} = a_i \setminus a_{i+1},$$

for each $i = 0, 1, \dots, n-1$, where $b_0 = a_0 = l$. By using the identities $x \setminus (x * y) = y$ and $x * (x \setminus y) = y$, we have that $d_l(e_l(a_1 a_2 \dots a_n)) = a_1 a_2 \dots a_n$ and $e_l(d_l(a_1 a_2 \dots a_n)) = a_1 a_2 \dots a_n$. This means that e_l and d_l are permutations on Q^n, mutually inverse. Compositions of e- and d-transformations are used in the design of RCBQ.

The concept of TASC was introduced in [3]. That cryptographic concept is the corner stone for the new algorithm for error correction. Here we use a way of implementation of TASC by quasigroup string transformations. We take the alphabet $Q = \{0, 1, \ldots, 9, a, b, c, d, e, f\}$, whose elements are 4-bit words, and we choose a quasigroup $(Q, *)$ (given in Table 1) with good properties according to the investigation in [8]. In fact, by using TASC, we can encrypt and decrypt messages. The TASC algorithm for encryption and decryption that we use for designing of RCBQ is given in Figure 1. TASC uses a key k for the encryption and decryption purposes and the length of the key has influence on the performances of RCBQ (smaller key length produces faster code with worser decoding results).

Encryption	**Decryption**
Input: Key $k = k_1k_2 \ldots k_n$ and message $L = L_1L_2 \ldots L_m$ **Output**: message (codeword) $C = C_1C_2...C_m$	**Input**: The pair $(a_1a_2 \ldots a_s, k_1k_2 \ldots k_n)$ **Output**: The pair $(c_1c_2 \ldots c_s, K_1K_2 \ldots K_n)$
For $j = 1$ to m $X \leftarrow L_j$; $T \leftarrow 0$; For $i = 1$ to n $X \leftarrow k_i * X$; $T \leftarrow T \oplus X$; $k_i \leftarrow X$; $k_n \leftarrow T$ **Output**: $C_j \leftarrow X$	For $i = 1$ to n $K_i \leftarrow k_i$; For $j = 0$ to $s - 1$ $X, T \leftarrow a_{j+1}$; $temp \leftarrow K_n$; For $i = n$ to 2 $X \leftarrow temp \setminus X$; $T \leftarrow T \oplus X$; $temp \leftarrow K_{i-1}$; $K_{i-1} \leftarrow X$; $X \leftarrow temp \setminus X$; $K_n \leftarrow T$; $c_{j+1} \leftarrow X$; **Output**: $(c_1c_2 \ldots c_s, K_1K_2 \ldots K_n)$

Figure 1: TASC algorithm for encryption and decryption

The main characteristic of TASC is that the error propagation is unbounded and it propagates until the end of the stream. However, by adding some redundant information in the stream, the correction of some errors can be done. That is in fact the main idea behind TASC Error Correction. We emphasis here that the pseudo random properties of RCBQ are obtained according to the following theorem.

Theorem 1. [6] *Consider an arbitrary string $\alpha = a_1a_2\dots a_n$ where $a_i \in Q$, and let β be obtained after k applications of an e-transformation. If n is enough large integer then, for each $1 \leqslant t \leqslant k$, the distribution of substrings of β of length t is uniform.* □

Note that for $t > k$ the distribution of substrings of β of length t is not uniform (see [1]).

3. Description of RCBQ

The code design uses the alphabet $Q = \{0, 1, \dots, 9, a, b, c, d, e, f\}$ of nibbles and a quasigroup operation $*$ on Q, together with its parastrophe $\backslash$ (as example, see Table 1).

3.1. Description of coding

Let $M = m_1m_2\dots m_r$ be a block of N_{block} bits, where m_i is a nibble (4-bit letter); hence, $N_{block} = 4r$. We first add redundancy as zero bits and produce block $L = L^{(1)}L^{(2)}\dots L^{(s)} = L_1L_2\dots L_m$ of N bits, where $L^{(i)}$ are 4-nibble words, L_i are nibbles, so $m = 4s, N = 16s$. After erasing the redundant zeros from each $L^{(i)}$ the message L will produce the original message M. On this way we obtain an (N_{block}, N) code with rate $R = N_{block}/N$. The codeword is produced from L after applying the encryption algorithm in TASC given in Figure 1. For that aim, previously, a key $k = k_1k_2\dots k_n$ of length n nibbles should be chosen. The obtained codeword of M is $C = C_1C_2\dots C_m$, where C_i are nibbles.

3.2. Description of decoding

After transmitting through a noise channel (for our experiments we use binary symmetric channel), the codeword C will be transformed to a received message $D = D^{(1)}D^{(2)}\dots D^{(s)} = D_1D_2\dots D_m$, where $D^{(i)}$ are blocks of 4 nibbles and D_j are nibbles. The decoding process consists of four steps: (i) a procedure for generating the sets with predefined Hamming distance, (ii) an inverse coding algorithm, (iii) a procedure for generating decoding candidate sets and (iv) a decoding rule.

Generating sets with predefined Hamming distance: The probability that $\leqslant t$ bits in $D^{(i)}$ are not correct is

$$P(p;t) = \sum_{k=0}^{t} \binom{16}{k} p^k (1-p)^{16-k}.$$

where p is probability of bit-error in a binary symmetric channel. Let B_{max} be an integer such that $1 - P(p; B_{max}) \leqslant q_B$, where q_B $(0 < q_B \leqslant 1)$ is given. Consider the set

$$H_i = \{\alpha | \alpha \in Q^4, \;\; H(D^{(i)}, \alpha) \leqslant B_{max}\},$$

for $i = 1, 2, \ldots, s$, where $H(D^{(i)}, \alpha)$ is the Hamming distance between $D^{(i)}$ and α. Then, with probability at least $1 - q_B$ the block $C^{(i)}$ is an element of the set H_i, for $i = 1, 2, \ldots, s$. The cardinality of the sets H_i is

$$B_{checks} = 1 + \binom{16}{1} + \binom{16}{2} + ... + \binom{16}{B_{max}}$$

and the number B_{checks} determines the complexity of the decoding procedure: for finding the element $C^{(i)}$ in the set H_i, less than or equal to B_{checks} checks have to be made. Clearly, for efficient decoding the number of checks B_{checks} has to be reduced as much as possible.

Inverse coding algorithm: The inverse coding algorithm is the decrypting algorithm of TASC given in Figure 1.

Generating decoding candidate sets: The decoding candidate sets S_0, S_1, S_2,..., S_s are defined iteratively. Let $S_0 = (k_1 \ldots k_n; \lambda)$, where λ is the empty sequence. Let S_{i-1} be defined for $i \geqslant 1$. Then S_i is the set of all pairs $(\delta, w_1 w_2 \ldots w_{16i})$ obtained by using the sets S_{i-1} and H_i as follows (Here, w_j are bits). For each $(\beta, w_1 w_2 \ldots w_{16(i-1)}) \in S_{i-1}$ and each element $\alpha \in H$, we apply the inverse coding algorithm with input (α, β). If the output is the pair (γ, δ) and if both sequences $\gamma = c_1 c_2 \ldots c_{16}$ and $L^{(i)}$ have the redundant nibbles in the same positions, then the pair $(\delta, w_1 w_2 \ldots w_{16(i-1)} c_1 c_2 \ldots c_{16}) \equiv (\delta, w_1 w_2 \ldots w_{16i})$ is an element of S_i.

Decoding rule: The decoding of the received codeword D is given by the following rule: If the set S_s contains only one element $(d_1 \ldots d_n, w_1 \ldots w_{16s})$ then $L = w_1 \ldots w_{16s}$. In this case, we say that we have a *successful decoding*.

In the case when the set S_s contains more than one element, we say that the decoding of D is unsuccessful (and then we say that error of type *more-candidate-error* appears).

In the case when $S_j = \emptyset$ for some $j \in \{1, \ldots, s\}$, the process will be stopped (and then we say that error of type *null-error* appears); we conclude that for some $m \leqslant j$, $D^{(m)}$ contains more than B_{max} errors, resulting with $C_m \notin H$. In this case, whenever it is possible, we may increase the value of B_{max} by 1 and repeat the decoding procedure for the block $D^{(m)}$ again.

Theorem 2. [4] *The packet-error probability of RCBQ is* $q = 1 - (1 - q_B)^s$. □

4. Description of RMC and RSC

RMC are amongst the oldest and most known codes. They were discovered and proposed by D. E. Muller and I. S. Reed in 1954 ([2], [5]). The r^{th} order Reed-Muller code, denoted as $RM(r,m)$, is defined as the set of all polynomials of degree at most r in the ring $F_2[x_0, x_1, \dots x_{m-1}]$. There is a recursive definition of $RM(r,m)$ given as follows.

1. $\mathcal{RM}(0,m) = \{\underbrace{00...0}_{2^m}\ \underbrace{11...1}_{2^m}\}$;
2. $\mathcal{RM}(m,m) = \mathbb{F}_2^{2^m}$;
3. $\mathcal{RM}(r,m) = \{x \| (x \oplus y) \mid x \in \mathcal{RM}(r, m-1), y \in \mathcal{RM}(r-1, m-1)\}$, for $0 < r < m$.

Here, $a \| b$ denotes the concatenation of the words a and b.

For decoding, majority logic decoding is applied.

RMC have many interesting properties that are important for examination. They form an infinite family of codes and larger RMC can be constructed from smaller ones. Unfortunately, RMC become weaker as their length increases. However, they are often used as building blocks in other codes.

The distance of Reed-Muller $RM(r,m)$ code is 2^{m-r} and this code can correct $2^{m-r-1} - 1$ bit errors in the message transmitted through the noise channel.

The RSC were invented in 1960 by I. S. Reed and G. Solomon ([7]). The first application of RSC in mass-production was for the compact discs (1982), where two interleaved RSC are used. Today RSC are used in hard disk drive, DVD, telecommunication, and digital broadcast protocols. These codes are defined over the Galois fields $GF(q)$. The Reed-Solomon code $C_{RS}(n,k)$ of length $n = q - 1$ is defined by the set of polynomials $A(x)$ of degree less than k with coefficients from $GF(q)$. The set of code words for this code is

$$C = \{(c_0, c_1, ..., c_{n-1}) | c_i = A(\alpha^i),\ i = 0, 1, ..., n-1,\ \deg(A(x)) < k\}$$

where α is a primitive element of $GF(q)$. The input message consists of k symbols from $GF(q)$ and they are the coefficients of the polynomials $A(x)$. The decoding is usually realized by using Berlekamp-Massey algorithm.

The Reed-Solomon code $C_{RS}(n,k)$ has minimum distance $n-k+1$ and it can correct $t = \lfloor (n-k)/2 \rfloor$ symbol errors in a code word.

5. Choosing parameters for optimal RCBQ

RCBQ have several parameters, and we have investigated the influence of the code parameters to the code performances [8]. Since RCBQ are designed using quasigroup string transformations on messages extended by introduced redundancy, we have pointed out how 1) the pattern of the redundancy, 2) the length of the key of TASC and 3) the chosen quasigroups, affect the codes.

We have made experiments in the following way. First, we extend input message using different patterns for redundant zero nibbles, and after that we encode the extended message and transmit it through a binary symmetric channel with probability p of bit error. For coding and decoding we use the codes described in Section 3. The outgoing message is decoded and if the decoding process completed successfully (the last set S_s of candidates for decoding has only one element), the decoded message is compared with the input message. If they differ at least one bit, then we say that an uncorrected-error appears. Then we compute the number of incorrectly decoded bits as Hamming distance between the input and the decoded message. Experiments showed that this type of package error occurs rarely.

In our experiments we also calculate the number of incorrectly decoded bits when the decoding process finish with more-candidate-error or null-error. Then, that number is calculated as follows.

When null-error appears, i.e., $S_i = \emptyset$, we take all the elements from the set S_{i-1} and we find their maximal common prefix substring. If this substring has k bits and the length of the sent message is m bits ($k \leqslant m$), then we compare this substring with the first k bits of the sent message. If they differ in s bits, then the number of incorrectly decoded bits is $m-k+s$.

If a more-candidates-error appears we take all the elements from the set S_s and we find their maximal common prefix substring. The number of incorrectly decoded bits is computed as previous.

The total number of incorrectly decoded bits is the sum of all of the previously mentioned numbers of incorrectly decoded bits.

We compute the probability of packet-error as

$$\text{PER} = \#(\textit{incorrectly decoded packets})/\#(\textit{all packets})$$

and the probability of bit-error as

$$\text{BER} = \#(\textit{incorrectly decoded bits in all packets}) \ / \ \#(\textit{bits in all packets}).$$

Experiments are made for different values of bit-error probability p of binary symmetric channel and $B_{max} = 3$ and $B_{max} = 4$. For $B_{max} > 4$, the experiments do not terminate in real time.

Redundancy pattern. We made experiments for different 6 patterns for redundant zero nibbles for (72,288) code with rate R=1/4. In these experiments we have used the quasigroup given in Fig. 1, the initial key $k = 01234$ and the following 6 patterns:

patt.1	**patt.2**	**patt.3**	**patt.4**	**patt.5**	**patt.6**
1000 1000	1100 1100	1100 1100	1100 1100	1100 1000	1100 1100
1000 1000	0000 1100	1000 0000	1100 0000	0000 1100	1000 0000
1000 1000	1100 0000	1100 1000	0000 1100	1000 0000	1100 1100
1000 1000	1100 1100	1000 0000	1100 1100	1100 1000	1000 0000
1000 1000	0000 1100	1100 1100	0000 0000	0000 1100	1100 1100
1000 1000	1100 0000	1000 0000	1100 1100	1000 0000	1000 0000
1000 1000	1100 0000	1100 1000	1100 0000	1100 1000	1000 1000
1000 1000	0000 0000	1000 0000	0000 0000	0000 1100	1000 0000
1000 1000	0000 0000	0000 0000	0000 0000	1000 0000	0000 0000

From the experimental results obtained for all six proposed patterns we conclude that the best results for PER and BER are obtained for the third pattern **patt.3**.

Key length. Theoretical probability of packet-error given in Theorem 2 is determined under the assumption that the code is perfectly random (i.e., the r-tuple are uniformly distributed in each codeword with length N, $r \leqslant N$). Therefore, in that theorem the more-candidates-errors are not provided. In Theorem 1 it is proved that if we apply t quasigroup transformations on a string, we obtain string where n-tuples of letters are uniformly distributed for $n \leqslant t$. In the design of these codes, the length of the key k determines how many times quasigroup transformations will be applied in forming of codeword. Therefore, longer key of the code gives "more random" code. This means that the results of experimental PER will be closer to the theoretical values for PER, i.e., the number of more-candidates-errors will be reduced. So, we made experiments with the third pattern (which give the best results) with key length 10. From the obtained results we saw that in some experiments more-candidates-error are not appeared, and if they appear, their number is very small. We can conclude that when we use a longer key, we can obtain better results for PER with almost the same

$*$	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	3	c	2	5	f	7	6	1	0	b	d	e	8	4	9	a
1	0	3	9	d	8	1	7	b	6	5	2	a	c	f	e	4
2	1	0	e	c	4	5	f	9	d	3	6	7	a	8	b	2
3	6	b	f	1	9	4	e	a	3	7	8	0	2	c	d	5
4	4	5	0	7	6	b	9	3	f	2	a	8	d	e	c	1
5	f	a	1	0	e	2	4	c	7	d	3	b	5	9	8	6
6	2	f	a	3	c	8	d	0	b	e	9	4	6	1	5	7
7	e	9	c	a	1	d	8	6	5	f	b	2	4	0	7	3
8	c	7	6	2	a	f	b	5	1	0	4	9	e	d	3	8
9	b	e	4	9	d	3	1	f	8	c	5	6	7	a	2	0
a	9	4	d	8	0	6	5	7	e	1	f	3	b	2	a	c
b	7	8	5	e	2	a	3	4	c	6	0	d	f	b	1	9
c	5	2	b	6	7	9	0	e	a	8	c	f	1	3	4	d
d	a	6	8	4	3	e	c	d	2	9	1	5	0	7	f	b
e	d	1	3	f	b	0	2	8	4	a	7	c	9	5	6	e
f	8	d	7	b	5	c	a	2	9	4	e	1	3	6	0	f

$\backslash$	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	8	7	2	0	d	3	6	5	c	e	f	9	1	a	b	4
1	0	5	a	1	f	9	8	6	4	2	b	7	c	3	e	d
2	1	0	f	9	4	5	a	b	d	7	c	e	3	8	2	6
3	b	3	c	8	5	f	0	9	a	4	7	1	d	e	6	2
4	2	f	9	7	0	1	4	3	b	6	a	5	e	c	d	8
5	3	2	5	a	6	c	f	8	e	d	1	b	7	9	4	0
6	7	d	0	3	b	e	c	f	5	a	2	8	4	6	9	1
7	d	4	b	f	c	8	7	e	6	1	3	a	2	5	0	9
8	9	8	3	e	a	7	2	1	f	b	4	6	0	d	c	5
9	f	6	e	5	2	a	b	c	8	3	d	0	9	4	1	7
a	4	9	d	b	1	6	5	7	3	0	e	c	f	2	8	a
b	a	e	4	6	7	2	9	0	1	f	5	d	8	b	3	c
c	6	c	1	d	e	0	3	4	9	5	8	2	a	f	7	b
d	c	a	8	4	3	b	1	d	2	9	0	f	6	7	5	e
e	5	1	6	2	8	d	e	a	7	c	9	4	b	0	f	3
f	e	b	7	c	9	4	d	2	0	8	6	3	5	1	a	f

Table 1: Quasigroup of order 16 and its parastrophe used in the experiments

duration of the decoding process.

Choosing of a quasigroup. Since we work with finite sequences, the randomness of a sequence obtained by quasigroup transformations depends on the used quasigroup. So, we did experiments with several quasigroups, which showed that the choice of the quasigroup does not affect only the values of PER and BER, but they have an enormous influence on the speed of decoding.

First we did experiments with the cyclic group of order 16 and the length of the key 10. Decoding for the third pattern was too slow. So, we did experiment with the first pattern for binary symmetric channel with $p = 0.02$ and $B_{max} = 3$, and we received PER=0.734087 and BER=0.460359 (that is much worse then PER=0.1186, BER=0.0089 obtained by the quasigroup in Table 1). Hence, it is clear that the choice of quasigroup has enormous influence over the performances of the code.

After that we made experiments with quasigroup of order 16 obtained as a direct product of a quasigroup of order 2. Experimental results obtained

with this quasigroup are worse than the results for cyclic group. For the first pattern, $p = 0.02$ and $B_{max} = 3$ we got PER=0.99424 and BER=0.80869.

The cyclic group and the direct product of quasigroups of order 2 are examples of so called fractal quasigroups, they produce biased sequences. The quasigroup in Table 1 is an example of so called non-fractal quasigroups. The results obtained by this quasigroup were quite satisfactory.

From the experiments, we can conclude that the best results for a RCBQ(72,288) were obtained for the third pattern, key length 10, quasigroup given in the Figure 1 (together with its parastrophe) and $B_{max} = 4$. We compare that code with RMC and RSC.

6. Experimental results for comparison

We have made several experiments in order to compare the performances of RCBQ(72,288) of rate 1/4 with Reed-Muller and Reed-Solomon codes of the same rate. (The experiments were made on ordinary PC, 2.6 GHz and 2 Gb RAM.) Because of the construction of RMC, its rate was chosen to be $130/512 \approx 1/4$. We considered transmissions through binary symmetrical channel for several values of probability p of bit-error. In order to obtain relevant statistics, we have made experiments with 13888 packets for RSC, 7692 packets for RMC and 3200 packets for RCBQ. (The experiments for RCBQ are time consuming.) In the experiments we have analyzed the packet-errors and the bit-errors.

For coding with the Reed-Muller code $RM(r, m)$, an input message is divided into blocks of $k = \sum_{i=0}^{r} \binom{m}{i}$ bits, and these blocks are encoded with code words with length $n = 2^m$. In this case, the code rate is $R = k/n$. We need to choose appropriate values for the parameters r and m, such that the code rate will be the closest to $R = 1/4$. Therefore, we made experiments with the code RMC(3,9), the length of the messages is $k = 130$ bits, and the length of the corresponding code words is $n = 512$ bits, i.e., the code rate is $R = 130/512 = 0.2539$.

In our experiments, we have used a shortened version of the RSC(63,27). It is the code RSC(48,12) defined over the Galois field $GF(2^6)$ with a primitive polynomial $p(x) = 1 + X + X^6$ (and it has the same good properties as general RSC). The shortened RSC has the same length of the code words (288 bits) and the same rate (1/4) as the considered RCBQ.

The probability of packet error. The results of the experiments for the PER are given in Table 2 and presented in Figure 2. We can derive the following conclusions.

p	RMC(3,9)	RSC(48,12)	RCBQ(72,288)
0.01	0	0	0.001250
0.02	0	0	0.001250
0.03	0	0.000216	0.003125
0.04	0.000650	0.003312	0.005938
0.05	0.010400	0.028874	0.015938
0.06	0.083073	0.107503	0.035938
0.07	0.260530	0.290251	0.066563
0.08	0.533021	0.505112	0.113125
0.09	0.759750	0.713062	0.188750
0.10	0.914587	0.845694	0.257813
0.11	0.971659	0.933899	0.350000

Table 2: Experimental results obtained for PER.

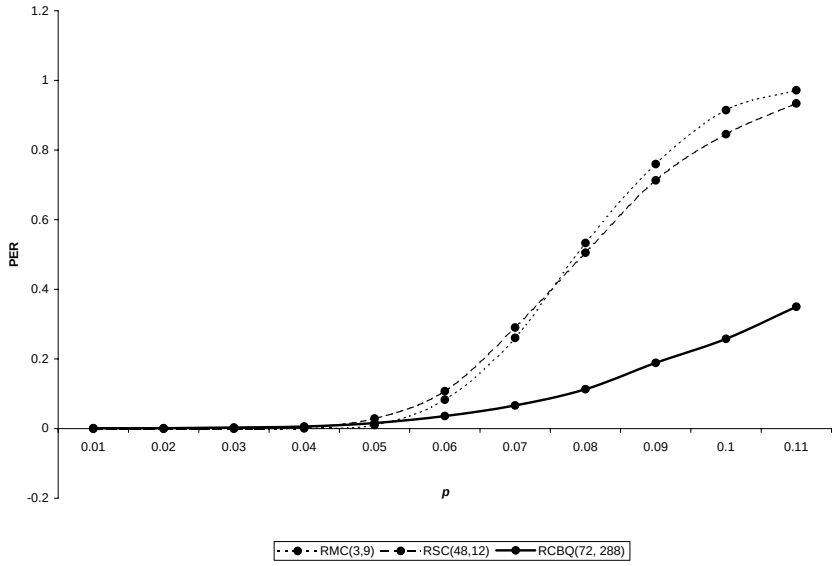

Figure 2: Comparison of PER for all three codes.

The RMC is the best code (with smallest values of PER) for $p \leqslant 0.05$. For $0.05 < p \leqslant 0.07$, RMC is better than RSC, but it is worse than RCBQ. For $p \geqslant 0.08$, RMC has the worst performances. The RSC has better

performances than RMC only for $p \geqslant 0.08$. RCBQ is better than RMC and RSC for $p > 0.05$. It is noticeable that RMC and RSC have similar performances for $p \geqslant 0.06$, while the performances of RCBQ become much better. Maybe the best characteristic for RCBQ appears when $p \geqslant 0.08$, since then RMC and RSC are useless (see Table 2), while RCBQ gives still reliable values.

The rate of growing of these codes is given in Table 3. So, RMC has the highest rate of growing, but it starts with very small PER for small values of p. The similar conclusion holds for RSC, too. The results for RCBQ are much better since the rates of growing are smaller than the suitable rates of RMC and RSC for all considered values of p. Therefore, we conclude that RCBQ is capable to decode for higher values of p.

	RMC(3,9)	RSC(48, 12)	RCBQ(72, 288)
p_1-p_2	PER(p_2)/PER(p_1)	PER(p_2)/PER(p_1)	PER(p_2)/PER(p_1)
0.01 - 0.02	/	/	1.00
0.02 - 0.03	/	/	2.50
0.03 - 0.04	/	15.33	1.90
0.04 - 0.05	16.00	8.72	2.68
0.05 - 0.06	8.00	3.72	2.25
0.06 - 0.07	3.14	2.70	1.85
0.07 - 0.08	2.05	1.74	1.70
0.08 - 0.09	1.43	1.41	1.67
0.09 - 0.10	1.20	1.19	1.37
0.10 - 0.11	1.06	1.10	1.36

Table 3: The rate of growing of PER.

The probability of bit error. The results of the experiments for the BER are given in Table 4 and presented in Figure 3.

It can be seen that we have similar results for BER as for PER, but the differences between the results for RCBQ and those for RMC and RSC are not so significant. The reason for that lies in the constructions of the codes. Namely, for RCBQ, when a bit is incorrectly decoded, then almost all consecutive bits are incorrectly decoded. On the other side, the number of bit-errors in a packet decoded by RMC or RSC are smaller, but they appear in almost all packets when $p \geqslant 0.08$.

Although the code of Reed-Solomon did not give the best results for PER and BER for any value of p, compared with the other two reviewed codes, this code has the best performance in terms of speed of the decoding

process. This is a very important feature in the coding theory. Namely, as a rule, a coding procedure is easily to be done fast and simple, but the problem in designing of codes is to make the decoding process to be time effective (in terms of capabilities for fast detecting and correcting the errors). The speed of decoding is a very important factor since in many real applications the codes work with huge amounts of data.

p	RMC(3,9)	RSC(48, 12)	RCBQ(72, 288)
0.01	0	0	0.000577
0.02	0	0	0.000759
0.03	0	0.000041	0.001359
0.04	0.000181	0.000671	0.003429
0.05	0.002488	0.006082	0.009279
0.06	0.020369	0.023117	0.022396
0.07	0.064838	0.063642	0.040647
0.08	0.135881	0.113436	0.064852
0.09	0.206767	0.166291	0.113572
0.10	0.268919	0.206621	0.156029
0.11	0.320262	0.239463	0.218902

Table 4: Experimental results obtained for BER.

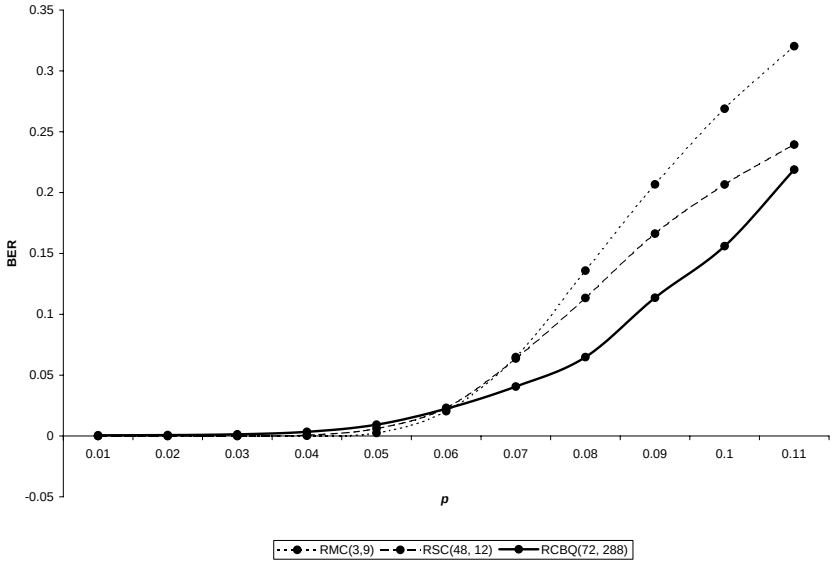

Figure 3: Comparison of BER for all three codes.

In the experiments with these three codes it can be noted that decoding with RCBQ is very complex and it is not time effective, especially for larger values of B_{max}. Thus, decoding of a packet is in average 25 times slower by using $B_{max} = 4$ instead of $B_{max} = 3$. Decoding a packet by using $B_{max} = 5$ could not be finished for a day. Nevertheless, RCBQ can be very useful for decoding messages transmitted through a very noisy channels (with up to 10% noise), especially when the decoding speed is not an important factor (for example, decoding a pictures transmitted from the deep space). The difference of the obtained decoding by using $B_{max} = 3$ and $B_{max} = 4$ can be noticed from Table 5. There, we can see that we have the same values for PER and BER when $B_{max} = 3$ with bit-error $p = 0.05$, and $B_{max} = 4$ with bit-error $p = 0.08$. The values PER_t in Table 5 are the theoretical probabilities according to Theorem 2. The paper [4] contains results for $B_{max} = 5$ and $B_{max} = 6$, obtained by using some auxiliary heuristic algorithms. The same results for PER and BER as above are obtained when $B_{max} = 5$ with bit-error $p \approx 0.115$, and $B_{max} = 6$ with bit-error $p \approx 0.155$.

$B_{max} = 3$			
p	PER_t	PER	BER
0.02	0.004314	0.004752	0.002093
0.03	0.019674	0.018433	0.008493
0.04	0.055435	0.055588	0.026289
0.05	0.118838	0.117584	0.054876
$B_{max} = 4$			
0.03	0.001447	0.003125	0.001359
0.04	0.005541	0.005938	0.003429
0.05	0.015319	0.015938	0.009279
0.06	0.034361	0.035938	0.022396
0.07	0.066467	0.066563	0.040647
0.08	0.114889	0.113125	0.064852

Table 5: Experimental results for $B_{max} = 3$ and $B_{max} = 4$

7. Conclusion

The RMC and the RSC are well known codes that are applied for many practical purposes. The RCBQ are new kind of codes defined by using quasigroups and quasigroup transformations, so RCBQ are based on quite

different principals than those of RMC and RSC. Here we have compared the decoding capacities of these three types of codes in terms on time effectiveness and capabilities for detecting and correcting the errors. For that aims several experiments were produced and relevant statistics were inferred from them. Generally, the RMC and the RSC have better decoding performances in a binary symmetrical channel with bit-error probability $p < 0.05$. In the opposite case, the RCBQ outperforms them significantly. Nevertheless, the time efficiency of the RMC and the RSC is much higher than that of RCBQ. So, the speed of decoding of RCBQ is its disadvantage and it is a challenge for further improvements.

We note that there are interesting results considering RMC and RCBQ when performed in a bounded binary symmetrical channel. A bounded binary symmetrical channel where the maximal number of erroneous bits in every 16 transmitted bits was 5 was considered in [4]. There was shown that RCBQ of rate 3/16 could decode a 32-bit message with 8-10 erroneous bits, while RMC (with the same rate) was not capable to do the decoding. These results suggest further investigations on the performances of RCBQ in different kind of channels.

Another feature of the RCBQ is that they have cryptographic properties. Namely, if the data are encoded with these random codes, then the recipient can decode the original data only if s/he knows exactly which parameters are used in the process of encoding, even if the communication channel is without noise. By the definition and the design of RCBQ, it is clear that if we want to implement the algorithms for decoding we must know the quasigroup and the primary key that is used for encoding the messages and, of course, the pattern for introducing the redundant nibbles into original message. (We could use not only zeros as a redundancy information, it can be any string, even with semantic meaning.) Therefore, the usage of RCBQ for cryptographic purposes is its another advantage over RMC and RSC.

References

[1] **V. Bakeva, V. Dimitrova**, *Some probabilistic properties of quasigroup processed strings useful in cryptanalysis*, M.Gushev, P.Mitreski (Eds.): ICT-Innovations 2010, Springer (2010), 61 – 70.

[2] **M. Bossert**, *Channel coding for telecommunications*, John Wiley and Sons, Ltd (1999).

[3] **D. Gligoroski, S. Markovski, Lj. Kocarev**, *Totally asynchronous stream ciphers + redundancy = cryptcoding*, S. Aissi, H.R. Arabnia (Eds.): Proc.

Internat. Confer. Security and Management, SAM 2007, Las Vegas, CSREA Press (2007), 446 – 451.

[4] **D. Gligoroski, S. Markovski, Lj. Kocarev**, *Error-correcting codes based on quasigroups*, Proc. 16th Intern. Confer. Computer Communications and Networks (2007), 165 – 172.

[5] **D. G. Hoffman, D. A. Leonard, C. C. Lindner, K. T. Phelps, C. A. Rodger, J. R. Wall**, *Coding Theory, The Essentials*, Auburn University, Auburn, Alabama (1992).

[6] **S. Markovski, D. Gligoroski, V. Bakeva**, *Quasigroup string processing: Part 1*, Maced. Acad. of Sci. and Arts, Sec. Math. Tech. Scien. **XX** 1-2 (1999), 13 – 28.

[7] **J. C. Moreira, P. G. Farrell**, *Essentials of error-control coding*, John Wiley and Sons, Ltd (2006).

[8] **A. Popovska-Mitrovikj, S. Markovski, V. Bakeva**, *Performances of error-correcting codes based on quasigroups*, ICT-Innovations 2009, Springer (2009), 377 – 389.

Received May 31, 2011

Faculty of Computer Science and Engineering, Ss Cyril and Methodius University,
Rudzer Boshkovikj, 16, P.O.Box 393, Skopje, Republic of Macedonia
E-mails: aleksandra.popovska.mitrovikj@finki.ukim.mk,
verica.bakeva@finki.ukim.mk, smile.markovski@finki.ukim.mk