IET Communications

Research Article

IET Journals
The Institution of Engineering and Technology

Gaussian channel transmission of images and audio files using cryptcoding

ISSN 1751-8628
Received on 3rd July 2018
Revised 11th February 2019
Accepted on 28th March 2019
doi: 10.1049/iet-com.2018.5636
www.ietdl.org

Verica Bakeva[1], Aleksandra Popovska-Mitrovikj[1] ✉, Daniela Mechkaroska[1], Vesna Dimitrova[1], Boro Jakimovski[1], Vladimir Ilievski[2]

[1]*Faculty of Computer Science and Engineering, Ss. Cyril and Methodius University in Skopje, P.O. Box 393, Republic of North Macedonia*
[2]*Ecole Polytechnique Fédérale de Lausanne, Switzerland*
✉ *E-mail: aleksandra.popovska.mitrovikj@finki.ukim.mk*

Abstract: Random codes based on quasigroups (RCBQ) are cryptcodes, i.e. they are error-correcting codes, which provide information security. Cut-Decoding and 4-Sets-Cut-Decoding algorithms for these codes are defined elsewhere. Also, the performance of these codes for the transmission of text messages is investigated elsewhere. In this study, the authors investigate the RCBQ's performance with Cut-Decoding and 4-Sets-Cut-Decoding algorithms for transmission of images and audio files through a Gaussian channel. They compare experimental results for both coding/decoding algorithms and for different values of signal-to-noise ratio. In all experiments, the differences between the transmitted and decoded image or audio file are considered. Experimentally obtained values for bit-error rate and packet error rate and the decoding speed of both algorithms are compared. Also, two filters for enhancing the quality of the images decoded using RCBQ are proposed.

1 Introduction

In this study, we investigate the performance and application of random codes based on quasigroups (RCBQs) for transmission of images and audio files through a Gaussian channel. RCBQs correct a certain amount of errors appeared during transmission through a noisy channel and at the same time encrypting the input messages, i.e. they are cryptcodes. This property is an advantage of RCBQs versus other error-correcting codes. There are few publications for the design of algorithms that combine error-correcting codes and ciphers [1–3], but almost all of them are for cryptographic purposes.

RCBQs were initially proposed in [4]. Furthermore, we denote the coding/decoding algorithms given in [4] as the standard coding/decoding algorithm for RCBQs. The previous investigations of RCBQ [5, 6] showed that the speed of the decoding process is one of the biggest problems for standard RCBQs. The main reason for this is the length of the lists (called decoding-candidate sets) since the decoding of RCBQ is actually a list decoding. The length of the lists depends on the parameter $B_{\max}$ – the predicted maximal number of bit errors appeared during transmission of a block. The larger value of $B_{\max}$ gives larger lists and slower decoding process. In order to improve the decoding speed, in [7, 8] the authors proposed two new coding/decoding algorithms: Cut-Decoding and 4-Sets-Cut-Decoding algorithms. In these algorithms, the authors proposed coding/decoding in two (in Cut-Decoding) or four (in 4-Sets-Cut-Decoding) parallel processes. The decoding-candidate sets are reduced using the intersection of the sets obtained in the parallel decoding processes. In this way, the authors obtain faster decoding and also better results for packet-error rate (PER) and bit-error rate (BER) probabilities.

The rest of the paper is organised in the following way. In Section 2, we give some preliminary definitions, description of coding/decoding algorithms of RCBQ and analysis of the cryptographic properties of these codes. In Section 3, we describe how our experiments are made. Several experimental results for coding/decoding images are given and analysed in Section 4. Also, in this section, we propose two filters for enhancing the quality of decoded images. The experimental results obtained for audio files are presented in Section 5. In Section 6, we make some conclusions.

2 Description and properties of RCBQ

Random codes based on quasigroups are error-correcting codes designed using the encryption algorithm of the totally asynchronous cipher (TASC) implemented by quasigroup string transformations [9]. This cipher uses a quasigroup $(Q, *)$, i.e. a groupoid consisting of a set (called an alphabet) Q and operation $*: Q^2 \to Q$ with the following property: for all $a, b \in Q$ there exist unique $x, y \in Q$ satisfying the equations $a*x = b$ and $y*a = b$. The main body of the multiplication table of a quasigroup is a Latin square over the set Q. For a quasigroup $(Q, *)$ a new operation $\backslash$, called a parastrophe, can be derived from the operation $*$ as follows: $x*y = z \Leftrightarrow y = x\backslash z$. Then the algebra $(Q, *, \backslash)$ satisfies the identities

$$x\backslash(x*y) = y \quad \text{and} \quad x*(x\backslash y) = y \tag{1}$$

and $(Q, \backslash)$ is also a quasigroup.

Quasigroup string transformations are defined on a finite quasigroup $(Q, *)$ and they are mappings from Q^+ to Q^+ (Q^+ is the set of all non-empty words on Q). In RCBQs, we use two types of quasigroup transformations defined as follows. 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 by

$$\begin{aligned} e_l(a_1a_2\ldots a_n) = b_1b_2\ldots b_n &\Leftrightarrow b_{i+1} = b_i*a_{i+1}, \\ d_l(a_1a_2\ldots a_n) = b_1b_2\ldots b_n &\Leftrightarrow b_{i+1} = a_i\backslash a_{i+1}, \end{aligned} \tag{2}$$

for each $i = 0, 1, \ldots, n-1$, where $b_0 = a_0 = l$. By using the identities (1), we have that $d_l(e_l(a_1a_2\ldots a_n)) = a_1a_2\ldots a_n$ and $e_l(d_l(a_1a_2\ldots a_n)) = a_1a_2\ldots a_n$. This means that e_l and d_l are permutations on Q^n, mutually inverse. We use compositions of e_l and d_l in the code design.

2.1 Description of coding/decoding algorithms

At first, we describe the standard coding algorithm for RCBQs proposed in [4]. Let $M = m_1m_2\ldots m_l$ be a block of $N_{\text{block}} = 4l$ bits where $m_i \in Q$ and Q is an alphabet of 4-bit symbols (nibbles). First, we add a redundancy as zero symbols and produce a message

$$L = L^{(1)}L^{(2)}\ldots L^{(s)} = L_1L_2\ldots L_m,$$

of $N = 4m$ bits ($m = rs$), where $L_i \in Q$, $L^{(i)}$ are sub-blocks of r symbols from Q. After erasing the redundant zeros from each $L^{(i)}$, the message L will produce the original message M. In this way, we obtain (N_{block}, N) code with rate $R = N_{\text{block}}/N$. The codeword is produced after applying the encryption algorithm of TASC (given in Fig. 1) on the message L. For this purpose, a key $k = k_1k_2\ldots k_n \in Q^n$ should be chosen. The obtained codeword of M is

$$C = C_1C_2\ldots C_m,$$

where $C_i \in Q$. Notice that the encryption algorithm of the TASC uses compositions of e_l transformations.

In the Cut-Decoding algorithm, instead of using (N_{block}, N) code with rate R, we use two $(N_{\text{block}}, N/2)$ codes with rate $2R$ for coding/decoding the same message of N_{block} bits. Namely, for coding, we apply the same encryption algorithm (given in Fig. 1) two times, on the same redundant message L using different parameters (different keys or quasigroups). In this way, we obtain the codeword of the message as a concatenation of two codewords of $N/2$ bits. The code rate is again $R = N_{\text{block}}/N$.

In the 4-Sets-Cut-Decoding algorithm, we use four $(N_{\text{block}}, N/4)$ codes with rate $4R$, on the same way as in coding with the Cut-Decoding algorithm and the codeword of the message is a concatenation of four codewords of $N/4$ bits (code rate $R = N_{\text{block}}/N$).

The decoding in all three cases is actually a list decoding and it is described below.

In the standard decoding algorithm for RCBQs, after transmission through a noisy channel (for our experiments we use Gaussian channel), the codeword C will be received as message $D = D^{(1)}\,D^{(2)}\ldots D^{(s)} = D_1D_2\ldots D_m$ where $D^{(i)}$ are blocks of r symbols from Q and $D_i \in Q$. The decoding process consists of four steps: (i) procedure for generating the sets with a predefined Hamming distance, (ii) inverse coding algorithm, (iii) procedure for generating decoding candidate sets and (iv) decoding rule.

Let $B_{\max}$ be a given integer, which denotes the expected maximal number of errors occur in a block during transmission. The probability that at most t bits in D^i are not correctly transmitted is

$$P(P_{\text{b}}; t) = \sum_{k=0}^{t} \binom{4r}{k} P_{\text{b}}^k (1 - P_{\text{b}})^{4r-k},$$

where P_{b} is the probability of bit-error in a Gaussian channel. Then $P(P_{\text{b}}; B_{\max})$ is the probability that at most $B_{\max}$ errors occur in a block during transmission. We generate the sets $H_i = \{\alpha \,|\, \alpha \in Q^r,\ H(D^{(i)}, \alpha) \le B_{\max}\}$, for $i = 1, 2, \ldots, s$, where $H(D^{(i)}, \alpha)$ is the Hamming distance between $D^{(i)}$ and α.

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 $i \ge 1$. Then S_i is the set of all pairs $(\delta, w_1w_2\ldots w_{4ri})$ obtained by using the sets S_{i-1} and H_i as follows (w_j are bits). For each element $\alpha \in H_i$ and each $(\beta, w_1w_2\ldots w_{4r(i-1)}) \in S_{i-1}$, we apply the inverse coding algorithm (i.e. decryption algorithm given in Fig. 2) with input (α, β) (the decoding algorithm uses compositions of d_l transformations). If the output is the pair (γ, δ) and if both sequences γ and $L^{(i)}$ have the redundant zeros in the same positions, then the pair $(\delta, w_1w_2\ldots w_{4r(i-1)}c_1c_2\ldots c_r) \equiv (\delta, w_1w_2\ldots w_{4ri})$ ($c_i \in Q$) is an element of S_i.

In the Cut-Decoding algorithm, after transmission through a noisy channel, we divide the outgoing message $D = D^{(1)}D^{(2)}\ldots D^{(s)}$ into two messages $D_1 = D^{(1)}D^{(2)}\ldots\ D^{(s/2)}$ and $D_2 = D^{(s/2+1)}\ D^{(s/2+2)}\ldots D^{(s)}$ with equal lengths and we decode them parallelly

Input: Key $k = k_1k_2\ldots k_n$ and $L = L_1L_2\ldots L_m$
Output: codeword $C = C_1C_2...C_m$

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: $C_j \leftarrow X$

Fig. 1 *Encryption algorithm*

Input: The pair $(a_1a_2\ldots a_r, k_1k_2\ldots k_n)$
Output: The pair $(c_1c_2\ldots c_r, K_1K_2\ldots K_n)$

For $i = 1$ to n
  $K_i \leftarrow k_i$;
For $j = 0$ to $r - 1$
  $X, T \leftarrow a_{j+1}$;
  $temp \leftarrow K_n$;
  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: $(c_1c_2\ldots c_r, K_1K_2\ldots K_n)$

Fig. 2 *Decryption algorithm*

with the corresponding parameters. In this decoding algorithm, we make modification in the procedure for generating decoding candidate sets. Let $S_i^{(1)}$ and $S_i^{(2)}$ be the decoding candidate sets obtained in the i^{th} iteration of both parallel decoding processes, $i = 1, \ldots, s$. Then, before the next iteration, we eliminate from $S_i^{(1)}$ all elements whose second part does not match with the second part of an element in $S_i^{(2)}$, and vice versa. In the $(i+1)^{th}$ iteration both processes use the corresponding reduced sets $S_i^{(1)}$ and $S_i^{(2)}$.

In [8], the authors proposed four different versions of decoding with the 4-Sets-Cut-Decoding algorithm. The best results are obtained using 4-Sets-Cut-Decoding algorithm #3. Here, we use only this version and further on we briefly describe it. After transmission through a Gaussian channel, we divide the outgoing message $D = D^{(1)}D^{(2)}\ldots D^{(s)}$ into four messages $D^1 = D^{(1)}D^{(2)}\ldots D^{(s/4)}$, $D^2 = D^{(s/4+1)}D^{(s/4+2)}\ldots D^{(s/2)}$, $D^3 = D^{(s/2+1)}D^{(s/2+2)}\ldots\ D^{(3s/4)}$ and $D^4 = D^{(3s/4+1)}D^{(3s/4+2)}\ldots D^{(s)}$ with equal lengths and we decode them parallelly with the corresponding parameters. Similarly, as in the Cut-Decoding algorithm, in each iteration of the decoding process, we reduce the decoding candidate sets obtained in four parallel decoding processes. In the 4-Sets-Cut-Decoding algorithm #3, we generate decoding candidate sets in the following way.

Step 1. Let $S_0^{(1)} = (k_1^{(1)}\ldots k_n^{(1)}; \lambda)$, …, $S_0^{(4)} = (k_1^{(4)}\ldots k_n^{(4)}; \lambda)$, where λ is the empty sequence and $k_1 = k_1^{(1)}\ldots k_n^{(1)}$, …, $k_4 = k_1^{(4)}\ldots k_n^{(4)}$ are the initial keys used for obtaining four codewords, respectively.

Step 2. Let $S_{i-1}^{(1)}$, …, $S_{i-1}^{(4)}$ be defined for $i \ge 1$.

Step 3. Let four decoding candidate sets $S_i^{(1)}$,…, $S_i^{(4)}$ be obtained in four parallel decoding processes, in the same way as in the standard RCBQ.

Step 4. Let $V_1 = \{w_1w_2\ldots w_{r\cdot a\cdot i} \,|\, (\delta, w_1w_2\ldots w_{r\cdot a\cdot i}) \in S_i^{(1)}\}$,…, $V_4 = \{w_1w_2\ldots w_{r\cdot a\cdot i} \,|\, (\delta, w_1w_2\ldots w_{r\cdot a\cdot i}) \in S_i^{(4)}\}$ and $V = V_1 \cap V_2 \cap V_3 \cap V_4$.

If $V = \varnothing$ then $V = (V_1 \cap V_2 \cap V_3) \cup (V_1 \cap V_2 \cap V_4) \cup (V_1 \cap V_3 \cap V_4) \cup (V_2 \cap V_3 \cap V_4)$.

Step 5. For each $j = 1, 2, 3, 4$ and for each $(\delta, w_1 w_2 \ldots w_{r \cdot a \cdot i}) \in S_i^{(j)}$, if $w_1 w_2 \ldots w_{r \cdot a \cdot i} \notin V$ then $S_i^{(j)} \leftarrow S_i^{(j)} \backslash \{(\delta, w_1 w_2 \ldots w_{r \cdot a \cdot i})\}$.

Step 6. If $i < s/4$ then increase i and go back to Step 3.

After the last iteration, if all reduced sets $S_{s/2}^{(1)}$, $S_{s/2}^{(2)}$ in the Cut-Decoding algorithm (or $S_{s/4}^{(1)}$, $S_{s/4}^{(2)}$, $S_{s/4}^{(3)}$, $S_{s/4}^{(4)}$ in 4-Sets-Cut-Decoding) have only one element with the same second component $w_1 \ldots w_{r \cdot a \cdot s/4}$, then $L = w_1 \ldots w_{r \cdot a \cdot s/2}$ (or $L = w_1 \ldots w_{r \cdot a \cdot s/4}$). In this case, we say that we have a *successful decoding*. If the decoded message is not the correct one then we have an *undetected-error*. If the reduced sets obtained in the last iteration have more than one element then we have a *more-candidate-error*. In this case, we apply the following heuristic: we randomly select a message from the reduced sets in the last iteration and we take this message as the decoded message. If we obtain $S_i^{(1)} = S_i^{(2)} = \varnothing$ in some iteration of Cut-Decoding or $S_i^{(1)} = S_i^{(2)} = S_i^{(3)} = S_i^{(4)} = \varnothing$ in some iteration of the 4-Sets-Cut-Decoding algorithm, then the process will stop (a *null-error* appears). However, if we obtain at least one non-empty decoding candidate set (in step 3) then the decoding continues with the non-empty sets (the reduced sets are obtained by the intersection of the non-empty sets only). If we obtain only one non-empty set, in some iteration, then the decoding continues with the non-empty set using the standard RCBQ decoding algorithm.

In RCBQ, unsuccessful decoding with *null-error* occurs when more than predicted $B_{\max}$ bit errors appear during transmission of some blocks. Some of these errors can be eliminated if we cancel a few of the iterations of the decoding process and reprocess all of them or part of them with a larger value of $B_{\max}$. Therefore, in [5], the authors have proposed a method for decreasing the number of *null-errors* by backtracking. In this method, if an empty set is obtained in some iteration (e.g. ith), then k previous iterations ($(i-1)$th, $(i-2)$th, …, $(i-k)$th) are cancelled. After that the first of cancelled iterations ($(i-k)$th) is reprocessed with $B_{\max} = B_{\max} + 1$ or $B_{\max} = B_{\max} + 2$, and the next iterations continue with the old value of $B_{\max}$. With this procedure, only part of *null-errors* will be eliminated since we cannot know exactly in which iteration the correct sub-block does not enter in the decoding candidate set and exactly how many transmission errors ($B_{\max} + 1$, $B_{\max} + 2$ or more) occur in this sub-block. Also, we must note that using backtracking in some cases, we can obtain *more-candidate-error* instead of *null-error*. A similar method with backtracking in the case of *more-candidate error* is proposed in [7]. In this method, if the decoding process ends with more elements in the reduced decoding-candidate sets after the last iteration then a few of the iterations are cancelled and the first cancelled iteration is reprocessed using a smaller value of $B_{\max}$ (the next iterations use the old value of $B_{\max}$).

In the experiments, we use the following combination of these two methods with backtracking. If the decoding ends with *null-error*, then the last two iterations are cancelled and the first of them is reprocessed with $B_{\max} + 2$ (the next iterations use the previous value of $B_{\max}$). If the decoding ends with *more-candidate-error*, then the last two iterations of the decoding process are cancelled and the penultimate iteration is reprocessed with $B_{\max} - 1$. In the decoding of a message, we use at most one backtracking for *null-error* and at most one backtracking for *more-candidate-error*.

2.2 Cryptographic properties of RCBQ

The encryption and decryption algorithms are based on quasi-group string transformations ($e-$ and $d-$transformation). Therefore, here, we consider some cryptographic properties of quasigroups and quasigroups string transformations.

One of the most important cryptographic properties is the uniform distribution of the symbols in the encrypted string, which provides resistance against statistical attack. This property of quasi-group $e-$transformation is addressed in [10], where the authors proved the following property.

Let $(Q, *)$ be a given finite quasigroup and $\mathbf{p} = (p_1, p_2, \ldots, p_{|Q|})$ be the probability distribution of the symbols in Q, such that $p_i > 0$ for each i and $\sum_i p_i = 1$. Then the following theorem (given in [10]) holds.

Theorem 1: Consider a random string $\alpha = a_1 a_2 \ldots a_n$ ($a_i \in Q$) drawn i.i.d. according to the probability distribution $\mathbf{p}$. Let β be obtained after k applications of an $e-$transformation on α. If n is a large enough integer then the distribution of substrings of β of length t is uniform for each $1 \le t \le k$. (We note that for $t > k$ the distribution of substrings of β of length t is not uniform.)

Another important cryptographic property of the encrypted string is its period. In [11], the authors proved that the period of quasigroup processed strings grows at least linearly and the increasing of the period depends on the chosen quasigroup. More about the period of the quasigroup processed string is given in [12].

Most cryptographic properties of quasigroup processed strings depend on the chosen quasigroup. Therefore, we give several cryptographic properties that are important for the choice of the quasigroup used in the encryption algorithm.

A quasigroup $(Q, *)$ of order n is *shapeless* if and only if it is non-idempotent, non-commutative, non-associative, it has neither left nor right unit, it does not contain proper sub-quasigroups, and there is no $k < 2n$ for which identities of these kinds are satisfied

$$\underbrace{x(\ldots*(x}_{k}*y)) = y,\ y = ((y*\underbrace{x)*\ldots)*x}_{k}. \tag{3}$$

The condition $k < 2n$ for identities (3) means that any left and right translation of quasigroup $(Q, *)$ should have the order $k \ge 2n + 1$. For cryptographic purposes, it is preferable to choose a shapeless quasigroup [9].

The classifications of finite quasigroups are very important for the successful application of quasigroups in cryptography and coding theory. It is a difficult problem since the number of quasigroups (even of small order) is very large.

A classification of quasigroups by graphical presentation of quasigroup processed strings is given in [13]. In that paper, the authors classified the set of all quasigroups of given finite order n into two disjoint classes, the class of so-called *fractal* quasigroups (if the graphical presentation of quasigroup processed strings has a structure), and the class of so-called *non-fractal* quasigroups (if the graphical presentation of quasigroup processed strings has no structure). The class of fractal groups is not recommended to be used for designing cryptographic primitives.

In order to obtain RCBQs with good cryptographic properties, the quasigroup used in our experiments is chosen to satisfy the above properties. This means that the chosen quasigroup is shapeless and non-fractal quasi-group.

3 Experiments

In this section, we explain how the experiments for the transmission of images and audio files through a Gaussian channel are made. All experiments are for code $(72, 576)$ with rate $R = 1/8$, $B_{\max} = 5$, different values of signal-to-noise ratio (SNR) and the following parameters:

- In the Cut-Decoding algorithm – redundancy pattern: 1100 1100 1000 0000 1100 1000 1000 0000 1100 1100 1000 0000 1100 1000 1000 0000 0000 0000, for rate 1/4 and two different keys of 10 nibbles.
- In 4-Sets-Cut-Decoding algorithms – redundancy pattern: 1100 1110 1100 1100 1110 1100 1100 1100 0000 for rate 1/2 and four different keys of ten nibbles.
- In all experiments, we used the same quasigroup of order 16 on Q given in [8].

In all experiments (for different values of SNR in the channel), we consider the differences between the transmitted and decoded image or audio file. Also, we compare experimentally obtained

a *b*

Fig. 3 *Original and encrypted image*
(a) Original image, ***(b)*** Encrypted image

Fig. 4 SNR $= -3$

values for bit-error probability (BER) and packet-error probability (PER) and the duration of the decoding processes with both algorithms.

In all decoding algorithms for RCBQ, when a *null-error* appears, the decoding process ends earlier and only a part of the message is decoded. Therefore, in the experiments with images and audio files, we use the following solution. In the cases of a *null-error* (all reduced sets are empty in some iteration), we take the strings without redundant symbols from all elements in the sets from the previous iteration and find their maximal common prefix substring. If this substring has k symbols then in order to obtain a decoded message of l symbols, we take these k symbols and add $l - k$ zero symbols at the end of the message.

In the next sections, we present and compare the results obtained using both algorithms for RCBQ with modifications for reducing the number of unsuccessful decoding.

4 Experimental results for images

In the experiments for image transmission, we use the image of 'Lenna', given in Fig. 3*a*. In the same figure, the second image is an encrypted image of Lenna (using the encryption algorithm given in Section 2.1) before transmitting through a Gaussian channel. It is apparent that the algorithm crypts the images. After that, the image is transmitted through the channel (with different values of SNR) and the corresponding decoding algorithm is applied. In Figs. 4–8, we present images obtained for SNR $= -2, -1, 0$ and 1, respectively. In each figure, the first image is obtained after transmission through the channel without using any error-correcting code, the second image is obtained using the Cut-Decoding algorithm and the third one – using the 4-Sets-Cut-Decoding algorithm.

Fig. 5 SNR $= -2$

Fig. 6 SNR $= -1$

From the figures, we can see that the Cut-Decoding and 4-Sets-Cut-Decoding algorithms correct many errors that appeared during transmission. Also, we obtain less damages (lines) of the images with the 4-Sets-Cut-Decoding algorithm than with the Cut-Decoding algorithm.

The values of BER and PER obtained with both algorithms (presented in Tables 1 and 2) confirm our conclusions from the analyses of images. In these tables, $\mathrm{BER_{cut}}$ and $\mathrm{PER_{cut}}$ denote the probabilities for the Cut-Decoding algorithm and $\mathrm{BER_{4\text{-}sets}}$ and $\mathrm{PER_{4\text{-}sets}}$ – corresponding probabilities obtained by using the 4-Sets-Cut-Decoding algorithm. From the results in the tables, we can see that for all values of SNR, $\mathrm{BER_{4\text{-}sets}}$ is more than three times smaller than $\mathrm{BER_{cut}}$. Also, the packet-error probabilities obtained with the 4-Sets-Cut-Decoding algorithm are more than two times smaller than the probabilities obtained with the Cut-Decoding algorithm.

As we explain before when *null-error* appears we add $l - k$ zero symbols at the end of the message and this makes horizontal black lines on the image. The horizontal white and grey lines are obtained in the case of *more-candidate-error* when the randomly selected message from the reduced sets in the last iteration differs from the original message. On the other hand, the images obtained

Fig. 7 SNR = 0

Fig. 8 SNR = 1

without using error-correcting codes do not have these lines, but the entire images have damaged pixels from incorrectly transmitted symbols.

In Tables 3 and 4, probabilities of *null-error* (PER_{null}) and *more-candidate-error* ($PER_{more-candidate}$) are given. From these tables, we can conclude that with the Cut-Decoding algorithm, we obtain a much greater number of unsuccessful decodings with *null-error* than with the 4-Sets-Cut-Decoding algorithm, but the number of *more-candidate-errors* is smaller. Therefore, the images decoded with the Cut-Decoding algorithm have more black lines than the images decoded with the 4-Sets-Cut-Decoding algorithm. Also, most of the lines in the third images (obtained with the 4-Sets-Cut-Decoding algorithm) are white or grey.

Also, we analysed the speed of two algorithms and concluded that the 4-Sets-Cut-Decoding algorithm is faster than the Cut-Decoding algorithm. The speed of the algorithms depends on the value of SNR and it increases as SNR increases. For example, for SNR = 1, the 4-Sets-Cut-Decoding algorithm is two times faster than the Cut-Decoding algorithm, and for SNR = −2, it is 16 times faster.

Table 1 Experimental results for BER

SNR	BER_{cut}	$BER_{4\text{-sets}}$
−3	0.31351	0.10411
−2	0.13257	0.03487
−1	0.04029	0.01233
0	0.00990	0.00306
1	0.00161	0.00040

Table 2 Experimental results for PER

SNR	PER_{cut}	$PER_{4\text{-sets}}$
−3	0.52898	0.24045
−2	0.24265	0.08019
−1	0.07429	0.02622
0	0.01675	0.00645
1	0.00316	0.00082

Table 3 Experimental results with Cut-Decoding

SNR	PER_{null}	$PER_{more-candidate}$
−3	0.52719	0.00096
−2	0.24155	0.00041
−1	0.07402	0.00027
0	0.01675	0
1	0.00316	0

Table 4 Experimental results with 4-Sets-Cut-Decoding

SNR	PER_{null}	$PER_{more\text{-}candidate}$
−3	0.07278	0.10588
−2	0.02691	0.03131
−1	0.01181	0.00796
0	0.00247	0.00288
1	0.00041	0.00027

In order to clear some of the damages (horizontal lines) on the images, in the next subsection, we propose two filters that visually enhance pixels damaged by *null-errors* and *more-candidate-errors*.

4.1 Filter for images decoded by cryptcodes based on quasi-groups

For repairing damage, a filter has to locate where it appears. Locating of the *null-errors* is easy since we add zero symbols in the place of the undecoded part of the message. In order to locate *more-candidate-errors*, we change the decoding rule for these kinds of errors. Instead of a random selection of a message from the reduced sets in the last iteration, we take a message of all zero symbols as a decoded message. Now, one pixel is considered as damaged if it belongs to a zero sub-block with at least four consecutive zero nibbles. The basic idea in the definition of the filters is to replace the damaged pixel intensity value with a new value taken over a neighbourhood of fixed size. In the first one, we use the median of the non-zero grey values of the surrounding pixels, so our filter is a median filter.

For each damaged pixel in the position (i,j), the filter uses the following algorithm:

1. Take a 3×3 region centred around the pixel (i,j).
2. Sort the non-zero intensity values of the pixels in the region in the ascending order.
3. Select the middle value (the median) as the new value of the pixel (i,j).

In Figs. 9–12, we present images obtained with the Cut-Decoding and 4-Sets-Cut-Decoding algorithm before and after the application of the proposed filter for SNR = −2, −1, 0, and 1,

Fig. 9 SNR $= -2$

Fig. 11 SNR $= 0$

Fig. 10 SNR $= -1$

Fig. 12 SNR $= 1$

respectively. In each figure, first two images are for Cut-Decoding (without and with the filter, respectively) and last two images for the 4-Sets-Cut-Decoding algorithm (without and with the filter).

From the presented images, we can notice that the proposed filter provides a great improvement of the images for all considered values of SNR. Also, this filter gives better results for the images obtained with the Cut-Decoding algorithm than with the 4-Sets-Cut-Decoding algorithm. The reason for this is the larger number of *undetected-errors* produced with the 4-Sets-Cut-Decoding algorithm.

Notice that instead of the median filter, some other filters more suitable for hardware implementation can be used. For example, we can use the average filter given by the following algorithm:

1. take a 3×3 region centred around the pixel (i,j);
2. calculate the average of the selected pixels and take this value as the new value of the pixel (i,j).

For smaller SNR, we recommend the median filter since it gives better results for images with larger number of damaged pixels. In Fig. 13, we present image transmitted through the Gaussian channel for SNR $= -2$ filtered by the median (the first image) and average filter (the second image).

Fig. 13 *Median and average filter*

It is apparent that the median filter gives better results. For a larger value of *SNR*, both filters give similar results, so the average filter can be used, too.

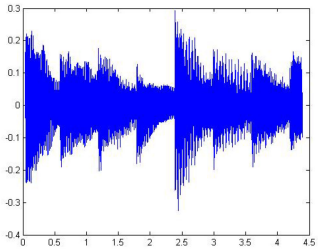


Fig. 14 *Original audio samples*

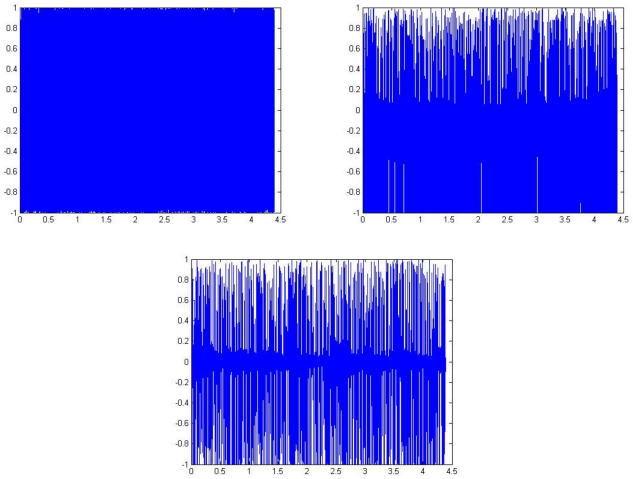
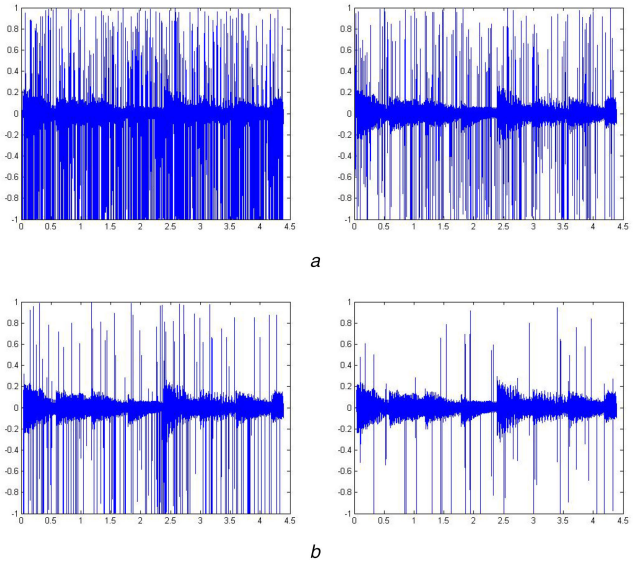


Fig. 15 SNR $= -1$



a

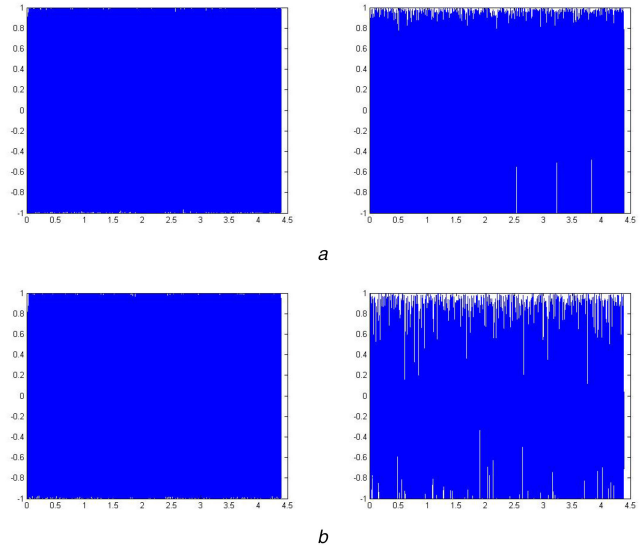
b

Fig. 16 *Results for* SNR $= 0$ *and* SNR $= 1$
(a) SNR $= 0$, ***(b)*** SNR $= 1$

5 Experimental results for audio files

In these experiments, we use an audio, which consists of one 16-bit channel with a sampling rate of 44,100 Hz and is a part of Beethoven's 'Ode to joy' with a total length of ~4.3 s.

Here, we also present experimental results for different values of SNR obtained using the Cut-Decoding and 4-Sets-Cut-Decoding algorithms. In all experiments, we consider the differences between the sample values of the original and transmitted signals. We present these analyses on graphs where the sample number in the sequence of samples consisting the audio signal is on the x-axes and the value of the sample is on the y-axis. In Fig. 14, the original audio samples are presented. The samples for SNR $= -1$ are given in Fig. 15. The first graph in this figure is obtained when the audio signal is transmitted through the channel without using any error-correcting code, the second graph is for Cut-Decoding and the third one is for the 4-Sets-Cut-Decoding algorithm. In Fig. 16, we present only two graphs per SNR $= 0$ (a) and SNR $= 1$ (b): the first one is for Cut-Decoding and the second is for the 4-Sets-Cut-Decoding algorithm. The graphs obtained when the audio signal is transmitted through the channel without using any error-correcting code are very similar to the first graph in Fig. 15 and we do not present them.



a

b

Fig. 17 *Results for SNR* $= -3$ *and* -2
(a) SNR $= -3$, ***(b)*** SNR $= -2$

Table 5 Experimental results for BER

SNR	BER_{cut}	$BER_{4\text{-sets}}$
−3	/	0.10386
−2	/	0.03658
−1	0.04741	0.01122
0	0.01114	0.00281
1	0.00175	0.00052

Table 6 Experimental results for PER

SNR	PER_{cut}	$PER_{4\text{-sets}}$
−3	/	0.24074
−2	/	0.08451
−1	0.08623	0.02499
0	0.02208	0.00632
1	0.00383	0.00113

From the figures, it is evident that for all values of SNR, the results obtained using the 4-Sets-Cut-Decoding algorithm are better than the results obtained with the Cut-Decoding algorithm.

For SNR $= -3$ and -2, experiments with the Cut-Decoding algorithm did not finish in real-time since during decoding of some messages we obtain a large number of elements in the decoding-candidate sets. Therefore, we conclude that for SNR < -1 there is no sense to make experiments with this algorithm and in Fig. 17, we present the results only for the 4-Sets-Cut-Decoding algorithm. The first graph on these figures is obtained when the audio signal is transmitted through the channel without any error-correcting code and the second one when the 4-Sets-Cut-Decoding algorithm is used.

In Tables 5 and 6, experimental results for BER and PER are given. These tables confirm our conclusions obtained from the figures. We can see that for the decoding audio files transmitted through a Gaussian channel, the 4-Sets-Cut-Decoding algorithm is better than the Cut-Decoding algorithm.

All audio files obtained in our experiments for transmission through a Gaussian channel with different SNR can be found in the following link:

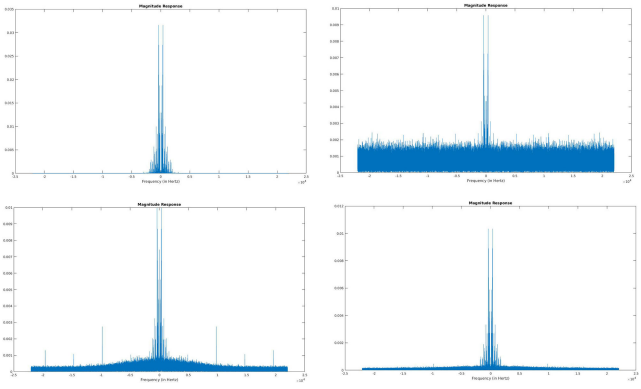


Fig. 18 *Energy spectral density for* SNR = 0

https://www.dropbox.com/sh/hwhk63ylgukvfq9/AACymAdyUBE1K_QJ-1SM39Jea?dl=0.

If one listen to these audio files, he/she would notice that as SNR decreases, the noise increases, but the original melody can be listened completely in the background.

In order to explain the reason behind this phenomena, we calculate the energy spectral density of the audio signal. The energy spectral density shows how the energy of the signal is distributed with frequency and it is defined as

$$E = \int_{-\infty}^{+\infty} \left| s(t) \right|^2 \mathrm{d}t,$$

where s is the signal. Since we consider finite discrete signals, in this case 4.3 s long audio signal, using Parseval's theorem, we can express the energy spectral density in the frequency domain using Fourier analysis, i.e.

$$\int_{-\infty}^{+\infty} \left| s(t) \right|^2 \mathrm{d}t = \int_{-\infty}^{+\infty} \left| \hat{s}(f) \right|^2 \mathrm{d}f,$$

where $\hat{s}(f)$ is the Fourier transformation of the signal s.

Therefore, using the fast Fourier transform algorithm from the MatLab's signal processing toolbox, in Fig. 18, we plot the energy densities for the sounds. The first graph is for the original sound, the second is for transmitted sound in the channel with $V_1 \cap V_2 \cap V_3 \cap V_4$, without using the error-correcting code and the last two graphs are for the same sound coded/decoded using the Cut-Decoding and 4-Sets-Cut-Decoding algorithms, respectively.

From the graphs, it is evident that after the transmission of the signal, the frequencies carrying the most of the signal's energy are still present, although a bit altered, all other energies are amplified. Consequently, this is the reason why we can still hear the original melody in the background intermixed with the noisy sounds.

The graphs confirm that the noise is reduced when Cut-Decoding, especially the 4-Sets-Cut-Decoding algorithm is used. The same results are obtained for other values of SNR.

The previous link also contains the audio file crypt_audio.wav obtained by encrypting the original audio. In this encrypted file, we cannot hear anything from the original melody, which confirms that the algorithm also crypts audio files.

6 Conclusion

From all experiments made for investigation of the performances of Cut-Decoding and 4-Sets-Cut-Decoding algorithms for RCBQ for transmission of images and audio files through a Gaussian channel, we can conclude that these two algorithms show good performances in all experiments. We can see that for all values of SNR, a great part of errors appeared during the transmission are corrected. Also, from the comparison, we can conclude that the 4-Sets-Cut-Decoding algorithm is better than the Cut-Decoding algorithm. Namely, the values of PER and BER obtained by using the 4-Sets-Cut-Decoding algorithm are more than three times smaller than values obtained by using the Cut-Decoding algorithm. Moreover, the 4-Sets-Cut-Decoding algorithm in all experiments is more than two times faster than the Cut-Decoding algorithm. Also, we define filters for improving the quality of decoded images. We must note that the encryption algorithm in the design of these codes provides information security of the transmitted data. This can be concluded from the considered cryptographic properties of the algorithm and the encrypted image and audio file.

Due to the good properties of the described codes, they can be successfully applied to different usage scenarios. One practical usage scenario might be using the codes in highly noisy wireless channels, where the code naturally fits in the data link layer, covering both coding and encryption purposes in one scheme. The proposed codes can also be used in satellite digital video broadcasting (DVB-S) coding and encryption schemes. Currently, DVB-S and DVB-S2 use two layers of encoding schemes, followed by an encryption scheme. Our coding approach replaces these layers of coding and encryption into a single scheme implementing all required features.

As future work, we will consider the performances of RCBQ for correction of burst errors, especially when a Rayleigh fading channel is used.

7 Acknowledgments

This research was partially supported by the Faculty of Computer Science and Engineering at the University 'Ss Cyril and Methodius' in Skopje.

8 References

[1] Hwang, T., Rao, T.R.N.: 'Secret error-correcting codes', in Goldwasser, S (Ed.): '*Advance in cryptology - CRYPTO'88*', LNCS 403 (Springer, 1990), pp. 540–563

[2] Zivic, N., Ruland, C.: 'Parallel joint channel coding and cryptography', *Int. J. Electr. Electron. Eng.*, 2010, **4**, (2), pp. 140–144

[3] Mathur, C.N., Narayan, K., Subbalakshmi, K.P.: 'High diffusion cipher: encryption and error correction in a single cryptographic primitive', in Yung, J., Bao, M., Zhou, F. (Eds.): '*Applied cryptography and network security*', LNCS 3989 (Springer, 2006), pp. 309–324

[4] Gligoroski, D., Markovski, S., Kocarev, L.: 'Error-correcting codes based on quasigroups'. Proc. 16th Int. Conf. on Computer Communications and Networks, Honolulu, USA, August 2007, pp. 165–172

[5] Popovska-Mitrovikj, A., Markovski, S., Bakeva, V.: 'Performances of error-correcting codes based on quasigroups', in Gómez, D., Davcev, J.M (Ed.): '*ICT innovations 2009*' (Springer, Berlin, Heidelberg, 2010), pp. 377–389

[6] Popovska-Mitrovikj, A., Markovski, S., Bakeva, V.: 'On random error correcting codes based on quasigroups', *Quasigroups Related Syst.*, 2011, **19**, (2), pp. 301–316

[7] Popovska-Mitrovikj, A., Markovski, S., Bakeva, V.: 'Increasing the decoding speed of random codes based on quasigroups', ICT innovations 2012, pp. 93–102. Available at: http://proceedings.ictinnovations.org/2012/paper/41/increasing-the-decoding-speed-of-random-codes-based-on-quasigroups

[8] Popovska-Mitrovikj, A., Markovski, S., Bakeva, V.: '4-Sets-Cut-Decoding algorithms for random codes based on quasigroups.', *AEU - Int. J. Electron. Commun.*, 2015, **69**, (10), pp. 1417–1428

[9] Gligoroski, D., Markovski, S., Kocarev, L.: 'Totally asynchronous stream ciphers+ redundancy= Cryptcoding'. Proc. Int. Conf. on Security and Management, Las Vegas, USA, 2007, pp. 446–451

[10] Markovski, S., Gligoroski, D., Bakeva, V.: 'Quasigroup string processing: part 1', *Contrib. Section Natural Math. Biotechnical Sci.*, 1999, **20**, (1–2), pp. 13–28

[11] Markovski, S., Gligoroski, D., Kocarev, L.: 'Unbiased random sequences from quasigroup string transformations', in Gilbert, H., Handschuh, H (Eds.): '*Fast software encryption*' (Springer, Berlin, Heidelberg, 2005), pp. 163–180

[12] Dimitrova, V., Markovski, J.: 'On quasigroup pseudo random sequence generators'. Proc. 1st Balkan Conf. in Informatics, Thessaloniki, Greece, 2003, pp. 393–401

[13] Dimitrova, V., Markovski, S.: 'Classification of quasigroups by image patterns'. Proc. 5th Int. Conf. for Informatics and Information Tehnology, Bitola, Macedonia, 2007, pp. 152–160