

Multilevel Indexed Quasigroup Encryption for Data and Speech

Maruti Satti and Subhash Kak

Abstract—This paper presents a practical implementation of a quasigroup based multilevel, indexed scrambling transformation for use in signal authentication, encryption, and broadcasting applications in secret-key cryptography. Results of experiments with text and speech scrambling are presented. It is shown that the quasigroup transformation is very effective in destroying the structure of the input signal and, therefore, it can be an excellent encryption technique.

Index Terms—Encryption, quasigroups, scramblers.

I. INTRODUCTION

PERMUTATION transformations are a basic operation in many scrambling and encryption systems and key distribution. Since an authentication authority is required in practical public key systems also, the use of permutations remains competitive. Many early ciphers for data and speech security [2], [3], [6], [8], [12], [13] were based on permutation transformations, and message authentication systems also use such transformations. Number theoretic transformations, such as those by decimal sequences [14], exponentiation or elliptic curves [15], or NTRU [16], which are used in encryption, may also be viewed as performing permutations. Such transformations are also of use in watermarking of signals [17]–[19] and in key broadcasting where several other techniques are also used [20]–[22].

It was shown in an earlier study by N.S. Jayant and one of the authors of the present paper [6] that a simple permutation of samples does very well in destroying intelligibility of speech, but for further protection against brute force attack it is essential to increase the number of possible keys. Quasigroups (or Latin squares) provide a powerful method for generating a larger set of permutation transformations by permuting not only the samples but also transforming the amplitudes themselves across their range [11]. By doing this, they provide an immensely large number of keys, even for small alphabets. Therefore, quasigroup based ciphers can have a variety of applications, and in some cases can be competitive to number theory based systems in terms of the difficulty they offer to brute force attacks.

Manuscript received August 22, 2007; revised December 19, 2008. First published April 14, 2009; current version published May 22, 2009.

M. Satti is with the Department of Electrical and Computer Engineering, Louisiana State University, Baton Rouge, LA 70803 USA.

S. Kak is with the Department of Computer Science, Oklahoma State University, Stillwater, OK 74078 USA (e-mail: subhash.kak@okstate.edu).

Color versions of one or more of the figures in this paper are available online at <http://ieeexplore.ieee.org>.

Digital Object Identifier 10.1109/TBC.2009.2014993

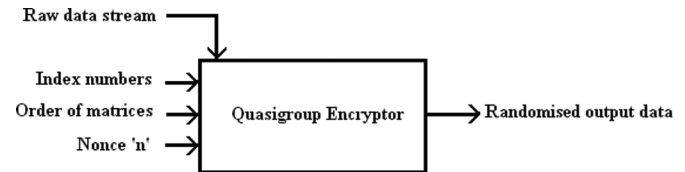


Fig. 1. Quasigroup encryptor.

This paper presents an implementation of quasigroup-based permutations that has very good encryption properties and, therefore, it has potential uses in symmetric cryptography as well as in design of message authentication and hash functions as well as in speech scrambling. If the purpose of the scrambler is to maximize the entropy at the output, then we see that the quasigroup based system accomplishes this successfully, even for input data that is constant. The immense complexity associated with the task of finding the scrambling transformation ensures the effectiveness of the encryption process.

Quasigroup based scrambling has been proposed before, but the earlier techniques such as those of Markovski and collaborators [4], [5], [9], [10] are rather insecure because their security is predicated only on the initial multiplier element that is like the seed of a random number generator. Here we propose a new multilevel indexed implementation that allows several layers of permutations to be carried out in a manner that is not only flexible but which enhances security. It distributes raw data stream based upon the indices and the order of the quasigroup under consideration. This unique key which consists of the index numbers and the matrix orders r and s (associated with the permutations that are performed) is kept secret and is transmitted by the trusted authority in a way that the keys do not repeat till a pre-specified network time expires.

The output of the proposed encryptor is dependent upon the index numbers and the orders of the matrix transformations that are sent by the trusted authority. The encryption is also dependent on six multiplier elements that are generated by a secret algorithm based on the index numbers, the order of the matrices under consideration and *nonce* (random number generated by trusted authority). This *key* is updated by the network on a regular basis (once in every time interval that is far less than T , the time needed to use brute force to decrypt the key sent by the trusted authority). Fig. 1 illustrates the quasigroup encryptor. The module here takes the raw data stream and randomizes it based on the encryption key (the encryption key). We show later that the output data has desirable autocorrelation properties. We have applied this multilevel scrambling approach to text and speech problems with good effect.

TABLE I
MULTIPLICATION TABLE FOR A QUASIGROUP, $n = 6$

*	1	2	3	4	5
1	4	3	2	1	5
2	2	1	5	4	3
3	5	4	3	2	1
4	1	5	4	3	2
5	3	2	1	5	4

This paper is organized as follows. Section II presents an overview of quasigroups. Section III introduces quasigroup signal processing. Sections IV–VI present the multilevel indexed quasigroup encryption (MIQE) system. Section VII describes experiments showing that the randomization performed by the quasigroup system is excellent. Section VIII presents a quasigroup voice scrambling system, and Section IX presents the conclusions of this study.

II. QUASIGROUP DEFINITION

A quasigroup Q may be defined [1] as a group of elements $(1, 2, 3, \dots, n)$ along with a multiplication operator such that for its elements x and y there exists a unique solution z , also belonging to Q , such that the following two conditions are obeyed:

$$\begin{aligned} x * a &= z \\ y * b &= z \end{aligned} \quad (1)$$

A quasigroup may be defined alternatively as a binary system $(Q, *)$ satisfying the two conditions: (see (2), shown at the bottom of the page). The multiplication table of a finite quasigroup is a Latin square.

Example 1: An example of a quasigroup Q of order 5 with elements $(1, 2, 3, 4, 5)$ is given by the element multiplications of Table I.

A multiplication operator in a quasigroup behaves as a mapping between the row and the column indices. Suppose $x = 2$ and $a = 3$; the resulting z can be determined by looking up the element having the row index as 2 and the column index as 3 so the obtained value of z is 5 ($z = 2 * 3$).

III. QUASIGROUP TRANSFORMATIONS

This section introduces the notation in the description of our multilevel indexed quasigroup encryption (MIQE) system.

Input data	$d_1, d_2, d_3, \dots, d_n$
Output data	$e_1, e_2, e_3, \dots, e_n$
Transformation matrices	R, S

Multiplier elements $q_1, q_2, q_3, \dots, q_n$

Indices $I_1, I_2, I_3, \dots, I_n$

The encryptor is represented by QE (stands for Quasi-Encryptor) and the decryptor is represented as QD (stands for Quasi-Decryptor).

A. Encryption

The mathematical equation used for encryption is given by:

$$E_a(a_1, a_2, a_3, \dots, a_n) = b_1, b_2, b_3, \dots, b_n \quad (3)$$

where the output sequence is defined by:

$$\begin{aligned} b_1 &= a * a_1 \\ b_i &= b_{i-1} * a_i \end{aligned}$$

where i increments from 2 to the number of elements that have to be encrypted, and a is the *leader* or the *hidden key*. Equation (6) describes a typical single level quasigroup encryptor.

We now illustrate the working of (3) with the help of the illustration of Fig. 2, for which the value of $a=2$. This equation maps the initial input data vector $(a_1, a_2, a_3, a_4, a_5, a_6) = (2, 4, 1, 2, 3, 3)$ into the vector $(b_1, b_2, b_3, b_4, b_5, b_6)$ for the case of the quasigroup of Example 1 with the multiplication relationship of Table I. The following steps are used during the process of encryption:

$$\begin{aligned} b_1 &= a * a_1 = 2 * 2 = 1 \\ b_2 &= b_1 * a_2 = 1 * 4 = 1 \\ b_3 &= b_2 * a_3 = 4 * 1 = 4 \\ b_4 &= b_3 * a_4 = 4 * 2 = 5 \\ b_5 &= b_4 * a_5 = 5 * 3 = 1 \\ b_6 &= b_5 * a_6 = 1 * 3 = 2 \end{aligned}$$

The sequence thus obtained $(1, 1, 4, 5, 1, 2)$ is then given as an input to another level of the encryptor. This process is repeated of times. This multiple levels of mapping ensure that the resemblance of the output data to that of the input data is minimized, making it harder for the eavesdropper to decrypt the data until he has sufficient information regarding the number of times the mapping was done by the sender.

In a variation to this approach, the multiplier element is varied, and generated by a special algorithm called MEG1 that generates the multiplier elements based on the index numbers, nonce, and r and s . This variant implementation may be given by the following equations:

$$E_{h_1, h_2, h_3, \dots, h_n}(a_1, a_2, a_3, a_4 \dots a_n) = e_1, e_2, e_3, \dots, e_n \quad (4)$$

For any a, b belonging to Q there exists a unique x belonging to Q such that $a * x = b$

For any a, b belonging to Q there exists a unique y belonging to Q such that $y * a = b$ (2)

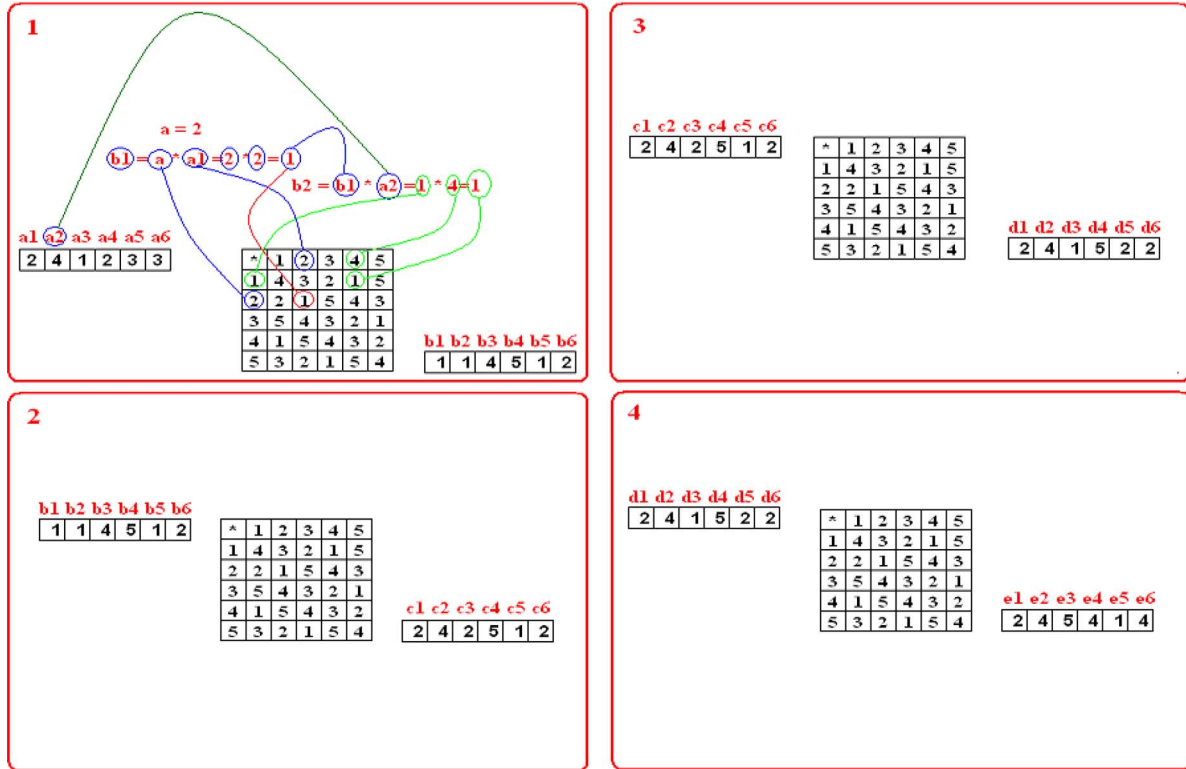


Fig. 2. Quasigroup mapping using an order 5 quasigroup.

where

$$e_1 = a * a_1 \text{ and } e_i = e_{i-1} * a_i$$

In the above equation the incoming stream of data is first mapped using the first multiplier element h_1 then the resultant steam is mapped considering the second multiplier element h_2 , and this process continues till all the multiplier elements are exhausted.

$$\begin{aligned} b_1 &= h_1 * a_1; b_2 = b_1 * a_2; \dots b_n = b_{n-1} * a_n \\ c_1 &= h_2 * b_1; c_2 = c_1 * b_2; \dots c_n = c_{n-1} * b_n \\ &\vdots \\ e_1 &= h_n * s_1; e_2 = e_1 * s_2; \dots e_n = e_{n-1} * s_n \end{aligned} \quad (5)$$

where the vector $(h_1, h_2, h_3, \dots, h_n)$ consists of all the multiplier elements. In this approach this encryption key is transmitted along with the quasigroup (this key is itself encapsulated by another layer of encryption). It is understood that in the above two approaches another reliable encryption algorithm is required to preserve the secrecy of the encryption. Moreover, one must transmit information regarding the quasigroups being used for encryption, and this constitutes one of the main drawbacks of the above approach.

The third approach is the index based approach where the given data is encrypted through several levels by the encryption. Let us consider that this changed order encryptor is given an input of all 1s (as illustrated in Fig. 3). We see that after the second level encryption the input vector is mapped to the

sequence that has symbols ranging from 1 to the order of the second matrix. Therefore, if we have an index key which references the matrices stored in the memory of the reception device, the eavesdropper would not know which matrix is stored at a given index. To further improve the efficiency of this encryptor we can include another function that arranges the quasigroups based upon a nonce that makes the output more independent of the input. At any given time the output of the encryptor is different even if the same set of indices is supplied to the algorithm.

Thus encryption in MIQE is represented by

$$QE_{h_1, h_2, \dots, h_n}^{I_r, I_s}(a_1, a_2, a_3 \dots a_n) = e_1, e_2, e_3, \dots e_n \quad (6)$$

where $(a_1, a_2, a_3, \dots, a_n)$ is the input vector and $(e_1, e_2, e_3, \dots, e_n)$ is the output vector, I_r and I_s are indices corresponding to the order of the quasigroups. The vector $(h_1, h_2, h_3 \dots, h_n)$ is called the *hidden key* or the *secret key*. It is the output of the MEG-1 algorithm.

B. Decryption

This process is similar to encryption. First, the generation of the inverse matrix will be described. For this one needs to understand the left inverse ‘\’ used for the quasigroup decryption (Fig. 4 illustrates the encryption and decryption of data).

The basic equation for encryption is as below:

$$D(a_1, a_2, a_3, \dots, a_n) = e_1, e_2, e_3, \dots e_n \quad (7)$$

where

$$e_1 = a \setminus a_1 \text{ and } e_i = a_i - 1 \setminus a_i$$

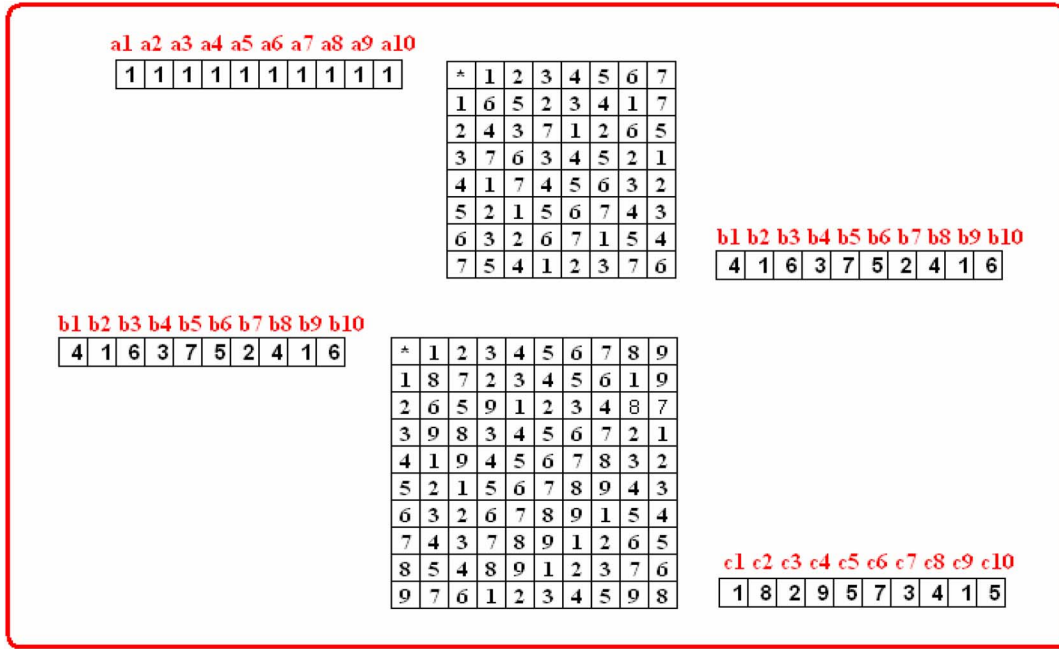


Fig. 3. Encryption using 2 different groups of different order.

To perform the process of decryption we need to first generate the inverse matrix of a given quasigroup and execute mapping procedure as described in the previous section (Fig. 2). This must be done using (7) rather than (6).

The decryptor for a multilevel indexed based algorithm can be defined as follows:

$$QD_{h_n, h_{n-1}, \dots, h_1}^{I_r, I_s}(e_1, e_2, e_3, \dots, e_n) = a_1, a_2, a_3, \dots, a_n \quad (8)$$

The method of the MIQE decryptor is similar to the MIQE encryptor as shown in the next section.

Note the following regarding the above Fig. 4:

- 1) The elements of the quasigroup (marked as 1) are labeled as w , the indices along the horizontal are labeled v and the indices along the vertical are labeled u .
- 2) The elements of the inverse (left-inverse) of quasigroup (labeled as 2) are labeled as v , the indices along the horizontal are labeled w and the indices along the vertical are labeled u^{-1} .

C. How to Determine the Left Inverse of a Given Quasigroup

By executing the following algorithm, one can generate the left inverse of a given quasigroup:

- 1) Scan the row of the quasigroup
- 2) Locate an element i (start the value of i from 1) and note the value of v in the corresponding location of the inverse matrix.
- 3) Increment i .
- 4) Go to step 1.
- 5) This process continues till all the elements in the row are exhausted.
- 6) Then go to the next row and repeat steps 1 through 3.

IV. MIQE RECEIVER

MIQE is a randomized algorithm that works on the encrypted key sent by a trusted authority. As illustrated in the above figure (Fig. 5), the input to this algorithm is a packet of data that consists of the following fields:

- 1) The first field consists of the orders of the two quasigroups that have to be generated,
- 2) The second field consists of the index numbers I_1, \dots, I_r and $I_1 \dots I_s$,
- 3) The third field consists of the nonce.

The algorithm first generates a quasigroup of order r with all the required properties, followed by the generation of all the possible isotopes of that particular base quasigroup. These isotopes are stored in the data base in a specified manner. Later it generates another quasigroup of order s and stores the isotopes in a similar fashion.

These quasigroups can be accessed by the encryptor module based upon the index numbers supplied to it. The encryptor module (described in detail in the later sections) codes the incoming data based upon the indices supplied to it; this module is also dependent on the inputs from another module that generates the multiplier seeds $q_1 \dots q_n$. Experimental results show that this algorithm has the capability of scrambling data over a wide range depending upon the index numbers, r, s and the multiplier elements.

V. THE PARAMETERS

In MIQE the final output data is dependent on the following factors:

- 1) Index numbers
- 2) Order of the index numbers
- 3) Order of the matrices r and s
- 4) The multiplier elements $(q_1, q_2, q_3, \dots, q_n)$

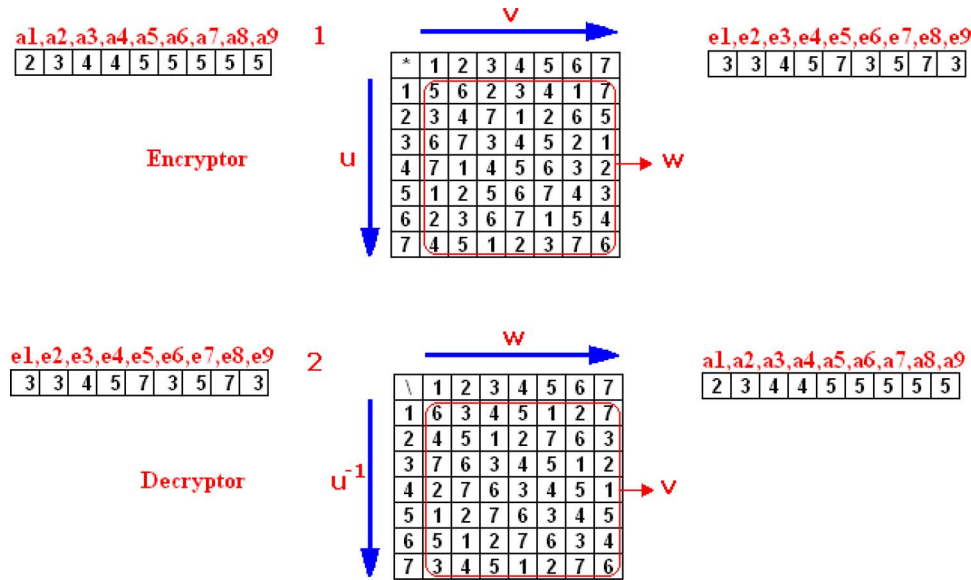


Fig. 4. Determination of left division and the complete process of encryption and decryption.

A. Index Numbers and Their Order

Perhaps the most important parameters underlying the effectiveness of MIQE, the index numbers may vary depending on the nonce and the order of the matrices. The nonce has two important functions:

- 1) It is used by the receiver to reset the database after the nonce expires
- 2) It is used by the Trusted Authority to generate an index and the order of the matrices. This way even the trusted authority does not know the next key that is to be transmitted at any given point of time. The algorithm that generates the index numbers and the orders of the matrices is named as FG-1 (the frame generator.)

Example 2: Let us assume that the FG1 algorithm generates the values of r , s and the index numbers as shown in the example below. Let us assume the following values:

$$r = 150 \text{ and } s = 270$$

$$I_1, I_2, I_3, I_4, I_5, I_6 : 1, 2, 3, 4, 5, 6$$

$$\text{Hidden key } 2, 3, 4, 5, 6, 1, 4$$

Let $d_1, d_2, d_3, d_4, d_5, d_6, d_7, d_8, d_9, d_{10} = 1, 1, 1, 1, 1, 2, 2, 2, 2, 2$. Given this amount of structure, this example should show the true scrambling capability of the encryptor.

The output data is represented by the vector: $e_1, e_2, e_3, \dots, e_{10}$

It must be noted that in the example we encode the data using six different matrices and the final output data would have any number in between 1 and 270 (which the order of the second matrix). One might argue that the largest number in the encrypted sequence might be considered as the order of the second matrix, but this is not possible because the encrypted data is produced by mapping the input data through several levels and the probability of the occurrence of the largest number becomes a function of the index vector which is random.

The simulation of the algorithm gives the results in Table II: From the above table the following deductions may be made:

- 1) The output sequence does tend to be similar for some sequences of the index numbers, however it does not exhibit any periodicity and is rather unpredictable when we consider the practical case where the output at any instance is dependent on other constraints such as the order of the matrix involved and the Hidden key.
- 2) One might argue that the eavesdropper can crack this code by simply doing a known plaintext attack. But that is impossible because the algorithm that generates the keys has a random input (owing to the nonce), and this nonce assures that even the Trusted Authority does not know the key that it would be generating in the next instant. Moreover the nonce changes.

B. Order of the Matrices r and s

In general there is no specified restriction on the order of the matrices supplied. However it is mandatory that the order of r be smaller than that of s . One must also note that the final output sequence also depends on the relative difference in the orders of the two matrices. It can be observed that the range of permissible sequence only depends upon the relative difference in between the order of the first and the order of the second matrix.

C. The Multiplier-Elements

The multiplier elements act like a seed to trigger the encryption process. In order to ensure the reliability of the decrypted data, the whole network must have this key (Hidden key). This problem is resolved by embedding a special algorithm called MEG1 into all the valid network devices including the trusted Authority and the user hand sets.

The multiplier elements depend upon the order of the matrix under consideration, the index number, and the nonce. This algorithm ensures the following:

- 1) It is virtually impossible to break this cipher based on known plaintext attack

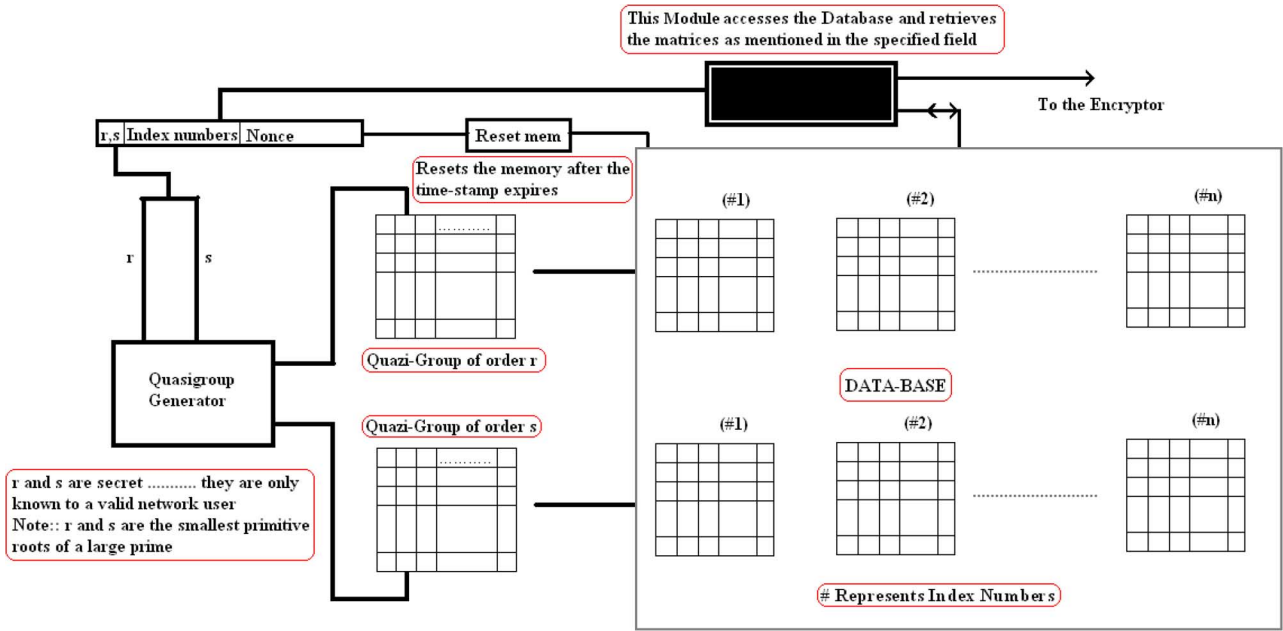


Fig. 5. The MIQE encryptor.

TABLE II
THE DECRYPTION PROCESS

Serial Num.	Index Numbers	Output Sequence ($e_1, e_2, e_3, \dots, e_{10}$)
1	1,2,3,5,4,6	126,95,162,16,123,93,163,216,176,249 **
2	1,6,5,4,2,3	125,92,154,267,83,17,30,268,107,14
3	1,4,2,5,6,3	126,95,162,16,123,93,163,216,176,249 **
4	1,5,6,4,2,3	126,95,162,16,123,93,163,216,176,249 **
5	1,4,5,2,6,3	127,99,172,36,158,149,247,66,71,199
6	1,4,6,2,5,3	121,82,137,244,58,268,30,36,209,214
7	2,3,5,6,1,4	270,255,190,9,140,21,91,27,67,169
8	5,6,2,3,1,4	123,84,136,240,62,45,48,22,113,252
9	2,5,3,1,6,4	1,255,189,7,137,17,86,21,60,161 ##
10	2,3,5,1,6,4	1,255,189,7,137,17,86,21,60,161 ##
11	2,3,5,4,6,1	4,261,200,24,161,49,127,72,122,235
12	1,1,1,1,1,1	124,87,139,232,13,161,90,208,152,109
13	2,2,2,2,2,2	2,255,193,18,159,55,146,110,186,63
14	3,3,3,3,3,3	5,259,180,220,231,134,164,31,237,149
15	4,4,4,4,4,4	269,240,131,104,126,62,65,3,113,2

2) It reduces the effort to encrypt and decrypt the data for a legitimate user while the computational requirement required to break this cipher is made tremendously high for an illegitimate user.

VI. THE FG-1 AND MEG-1 ALGORITHMS

A. The FG 1 Algorithm

From our study of the MIQE, it is evident that the security of the entire system depends on the unpredictability of the key (r, s, index elements and nonce). The FG 1 module ensures that even the Trusted authority doesn't know the key that it would be transmitting in the next instant (next time slot).

This algorithm generates the order of the matrices (r,s), index numbers and the nonce, based on a random input from a pseudo-random number generator. This module is essential due to the following reason:

- 1) This module ensures that the next sequence of indices generated by the system is unpredictable.
- 2) The nonce is it self dependent on the random input which makes the system more dynamic.
- 3) The security of the system is dependent on the period of the random number generator and the network time (network time is denoted by T. This time judges the value of nonce t that is to be sent in the next frame and at any given instant the value of t should not exceed the value of T).
- 4) The security also depends on the other factors like length of the random symbol, number of indices and the order of the quasigroups.

B. The MEG 1 Algorithm

While the FG 1 algorithm is only required at the trusted authority the MEG 1 algorithm plays a major role in the encryption and the decryption process (Fig. 6). During the process of encryption It generates the *hidden key* (explained before) and

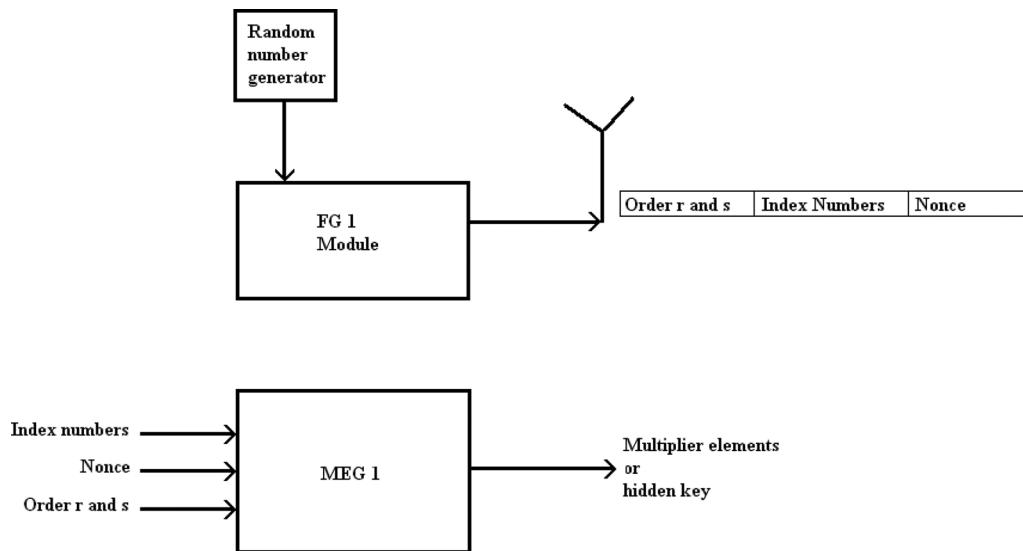


Fig. 6. The MEG 1 and FG 1 algorithms.

during the process of decryption the same key has to be generated at the receiver in order to ensure proper decryption. The MEG 1 algorithm takes the entire incoming frame and generates the multiplier elements.

1) *Nonce t*: The nonce acts like a random input to this algorithm. Since the hidden key generated is dependent on the value of the nonce supplied to it, this algorithm would generate different *hidden keys* even if the same orders (r,s) and index numbers are supplied to it. The nonce might be given by the formula

$$\text{Nonce } t = f(\text{prng})$$

where “prng” stands for the “Pseudo Random Number Generator”. We can even use the same random number that is given as the input to the FG-1 module. Since the nonce is generated by FG-1, it is evident that it remains constant for the entire network till it expires and a new transmission is made.

There is a tradeoff of security and overhead involved here. It is required that this nonce changes as frequently as possible, but that increases the computational costs of the entire network. In order to avoid any unnecessary overhead the nonce must be above a certain specified value. Assume that this arbitrary limit is given by T_1 :

$$T_1 < \text{Nonce 't'} < T$$

The above equation can be very significant in determining the network overload. If T_1 is very small then the devices in the network would have to recalculate the groups and indices several times per session and on the other hand if T_1 is made too large ($T_1 \rightarrow T$) then there is a high chance of the network being compromised.

2) *Orders r, s*: Since in quasigroup encryption the multiplier element for a certain level must belong to the quasigroup under consideration, the set of multipliers ($q_1, q_2, q_3 \dots q_n$) belongs

to the set of quasigroup (corresponding indices) that are being used at that particular instant for the process of encryption.

Let us suppose that I_1, I_2, I_3 are the indices corresponding to the quasigroup of order r and let I_4, I_5, I_6 be the indices corresponding to the quasigroup of order s . Let us suppose that the MEG-1 generates $q_1, q_2, q_3, q_4, q_5, q_6$. This means that q_1, q_2, q_3 are used as the multiplier elements during the encryption using the quasigroup of order r . In other words, for the data to be properly mapped q_1, q_2, q_3 must belong to the group of order r . Likewise, the multipliers q_4, q_5, q_6 must belong to the second group.

VII. RANDOMIZATION OF DATA

In this section we will study the characteristics of the output data by supplying some test data to MIQE. In general the text information sent can be very random, repetitive or have certain characteristics (for example, in English text the probability of occurrence of e is maximum) that would make an encoded file vulnerable to the *known plain text attack*. Let us first consider the case of normal English text.

Consider the text document shown below is supplied to the MIQE, the characters are then permuted based on the key provided. The scrambling transform used is time dependent and this time dependency makes it cryptographically strong. The corresponding output data is also shown in the same table. The key used was (35,41,5,4,2,1,6,3).

Example 3: Input text together with the transformed sequence (Fig. 7)

Input data:

ONE DAY A COUNTRYMAN GOING TO THE NEST OF HIS GOOSE FOUND THERE AN EGG ALL YELLOW AND GLITTERING WHEN HE TOOK IT UP IT WAS AS HEAVY AS LEAD AND HE WAS GOING TO THROW IT AWAY BECAUSE HE THOUGHT A TRICK HAD BEEN PLAYED UPON HIM. BUT HE TOOK IT HOME ON SECOND THOUGHTS AND SOON FOUND TO

Encoded data:

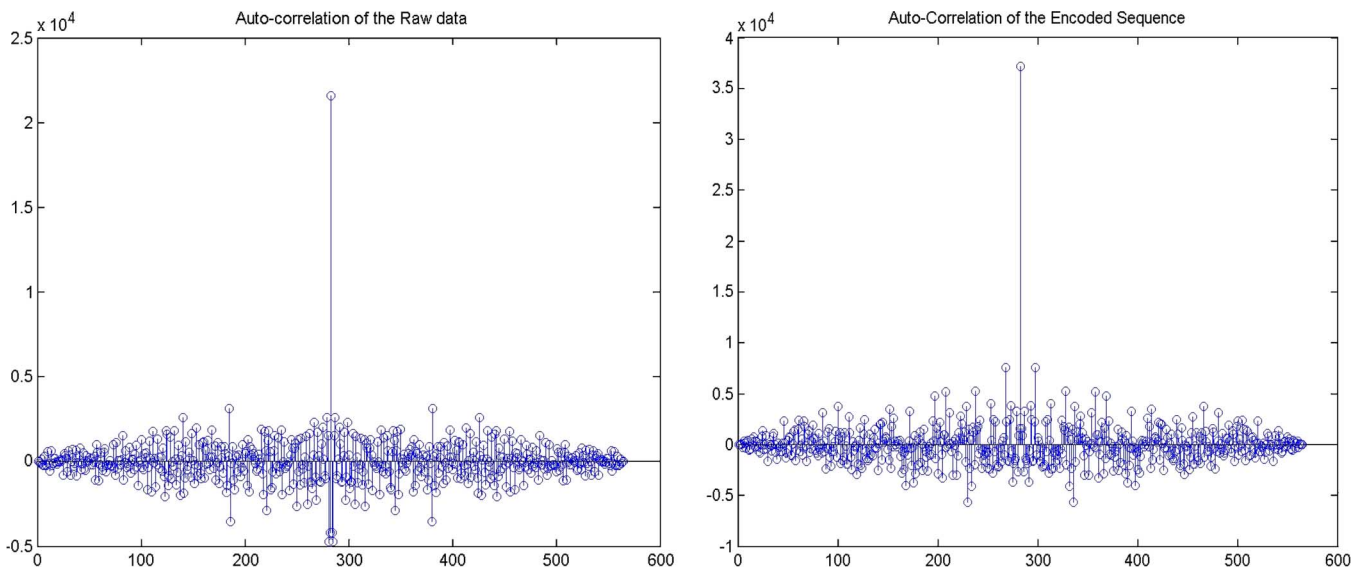


Fig. 7. Autocorrelation of the input data and the encoded data for Case 1.

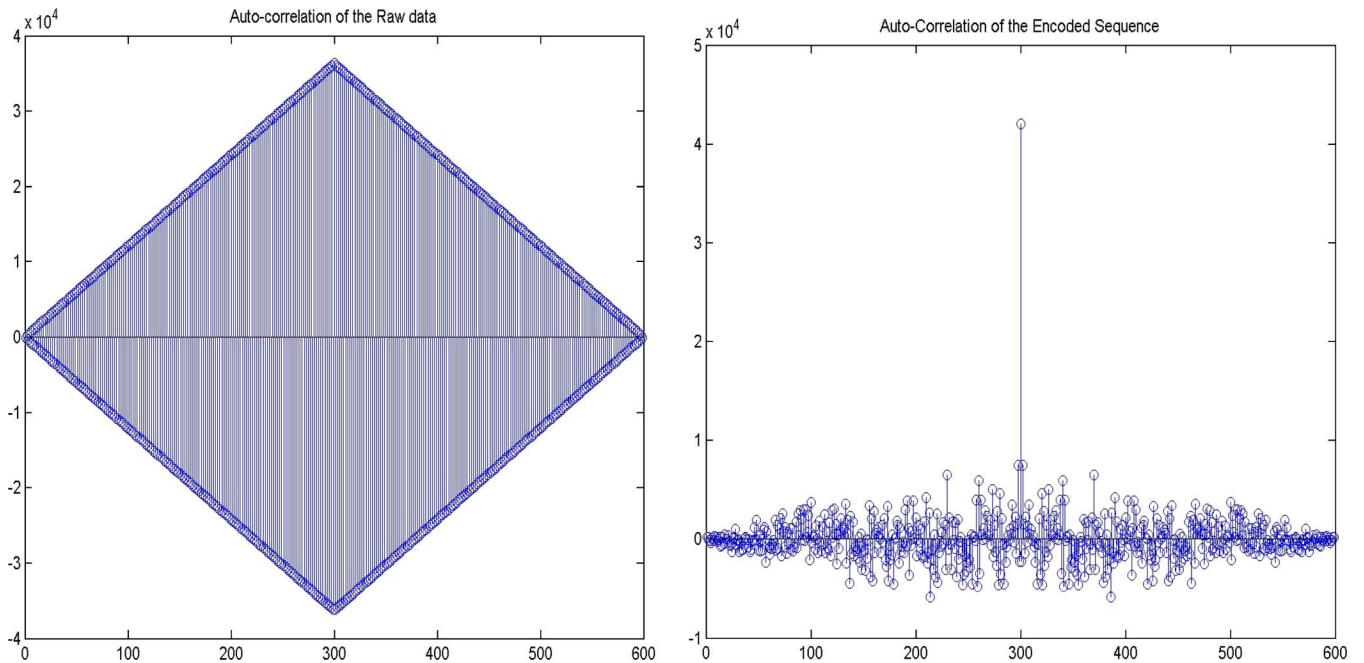


Fig. 8. Autocorrelation of the raw data and the encoded data for a constant input sequence.

23 32 17 38 35 8 12 21 24 39 1 26 6 39 15 1 3 6 39 34 4 11
 40 29 4 19 20 2 9 34 20 5 6 29 37 6 5 41 23 17 29 37 16 30 28
 33 13 41 30 18 11 40 40 19 40 13 23 21 30 34 27 40 8 37 8 14
 5 15 23 20 30 32 35 24 26 34 21 39 41 14 4 5 16 26 37 33 11 3
 37 34 8 12 33 16 18 36 2 26 12 29 22 36 27 16 18 25 17 19 8 8
 33 6 28 26 17 39 19 33 22 38 34 32 27 12 4 19 5 29 16 33 33
 39 34 37 38 36 24 25 5 14 39 3 38 13 20 30 30 26 37 31 17 8 3
 19 8 33 17 34 19 27 25 6 19 30 19 19 7 10 1 30 20 3 29 6 40 33
 19 13 36 39 35 25 30 22 34 16 18 13 31 19 28 4 28 9 39 14 24
 4 6 22 10 39 32 21 15 37 19 19 4 29 13 26 4 16 6 29 18 29 7 21
 19 19 16 38 19 9 37 6 16 14 25 22 26 14 19 34 11 13 3 13 27
 27 17 9 17 34 30 3 9 15 34 15 3 11 22 18 39 26 19 41 32 29 10
 20 11 11 13 35 40 39 30 24 10 12 17 3 28 41 26 5 32 19 19

Key used: 35,41,5,4,2,1,6,3

Example 4: Suppose that the data supplied to the MIQE is highly redundant and it consists of a string of Es (Fig. 8). In spite of this the output is mapped to a range of values from 1 to 41. It must be noted that the first two integers in the key are the order of the matrices and the remaining are the encryption indices.

Input data:

EE
 EEE
 EEE
 EEE
 EEE

Encoded data:

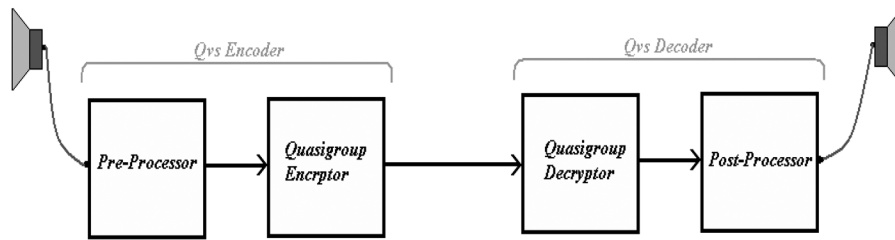


Fig. 9. QVS block diagram.

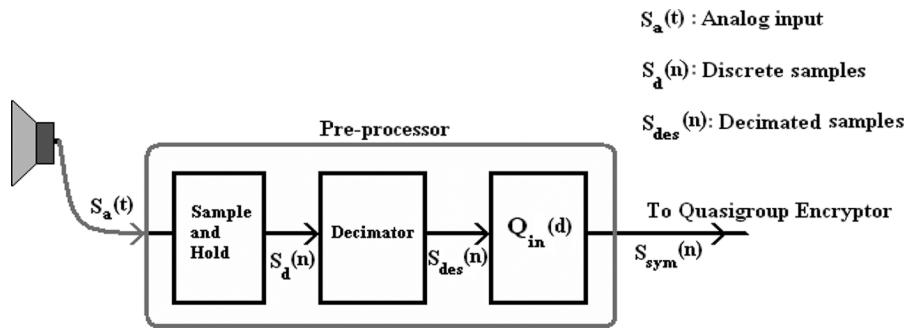


Fig. 10. Block diagram of the pre-processor.

11 10 24 28 33 11 23 8 28 10 17 20 2 40 29 15 22 2 30 6 30
 7 29 37 22 10 7 9 18 4 29 38 12 30 17 33 7 10 27 28 38 38 33
 12 11 19 4 12 29 6 7 12 9 37 16 12 34 26 18 5 40 17 39 36 41
 38 36 16 31 34 8 32 35 9 33 1 34 29 20 1 33 7 33 34 33 13 9 5
 38 3 39 38 6 32 18 34 16 21 34 9 41 24 12 34 30 35 19 7 32 7
 32 1 24 21 20 5 27 11 23 7 12 27 35 12 9 18 2 33 41 9 33 32 23
 35 5 29 15 16 23 5 21 40 3 6 16 4 27 18 23 12 30 1 27 15 19 17
 22 18 35 23 38 27 36 18 35 24 17 27 6 9 23 37 32 36 23 41 6 5
 28 20 26 2 6 6 33 32 8 21 3 12 1 32 29 22 6 26 21 4 33 7 38 33
 16 23 5 8 12 21 19 40 6 24 1 28 9 4 5 7 37 2 35 8 23 4 16 7 3
 12 22 18 36 27 31 37 19 34 35 7 17 8 36 9 34 21 30 28 7 27 40
 35 25 16 19 10 15 20 6 8 18 1 22 8 34 22 25 36 19 32 40 29 9
 35 34 10 35 41 34 35 11 17 23 11 36 35 22 25 34 38 14 25 39
 12 34 27 26 35 26 20 27 1

Key used: 35,41,5,4,2,1,6,3

We see from the autocorrelation graphs that the encrypted sequence is essentially two-valued and, therefore, the output sequence is very random.

VIII. QVS—QUASIGROUP VOICE SCRAMBLER

This section describes applications of MIQE to speech scrambling. Since the sampled speech is not of integer type, some sort of pre-processing must be done because MIQE only scrambles integer data (or binary). The quasigroup voice scrambler (QVS) constitutes of an encoder and a decoder (Fig. 9 is the encoder-decoder pair).

The encoder as indicated in the illustration has a pre-processor and MIQE while the decoder has MIQE and post-processor.

1) *The Pre-Processor*: This subsystem (Fig. 10) consists of a *sample and hold circuit*, a *decimator of a factor r* and a pre-transforming function $Q_{in}(d)$, where d represents the raw data. After the speech signal is sampled, it is passed through a decimator. This step is essential to reduce the redundancy that is

inherent in speech data. The third step is perhaps the most important one. The transform $Q_{in}(d)$ takes the sample values of the speech signal and converts them to integer values without sacrificing the voice clarity.

It must be noted that for telephone quality speech we need to be concerned with frequencies between 300 Hz to 3.6 kHz. Thus we also need a band pass filter to pass only the frequencies of interest. However we assume that $S_a(t)$ is an NB analog-signal.

The output of the pre-processor are integer symbols representing the discrete samples $S_{des}(n)$. There is yet another restriction on the largest symbol that is used by the pre-processor to represent an input sample. If the incoming sample is 857 then we need a quasigroup of the size 857 to effectively encode the symbol. Creating a quasigroup of that size could take considerable time and resources. By keeping the symbol range to a controllably small value, one can prevent heavy costs and maximize the speed and efficiency of the encoder.

2) *Experimental Results*: In Fig. 11 the first graph (a) is the output of the pre-processor. We can see that it is periodic from its auto-correlation properties. However after scrambling the output becomes highly randomized (the auto-correlation of the encoded data is same as that of a random sequence). Hearing tests show that the encoded sequence sounds similar to the sequence generated by the 'rand' function (in Matlab). The decoded sequence on the other hand sounds exactly the same as that of the original signal.

3) *The Post-Processor*: The post processor does exactly the reverse operation of the pre-processor. The input to the post-processor is the decoded symbol vector $S_{sym}(n)$ from the MIQE. This sequence of symbols is transformed to the original form by the inverse transform $Q^{-1}(e)$ (where e is the decoded symbol vector). The discrete set of samples are interpolated (the discrete set of samples is represented as d in Fig. 12) to obtain the analog signal that was initially transmitted. (This step is only meant to

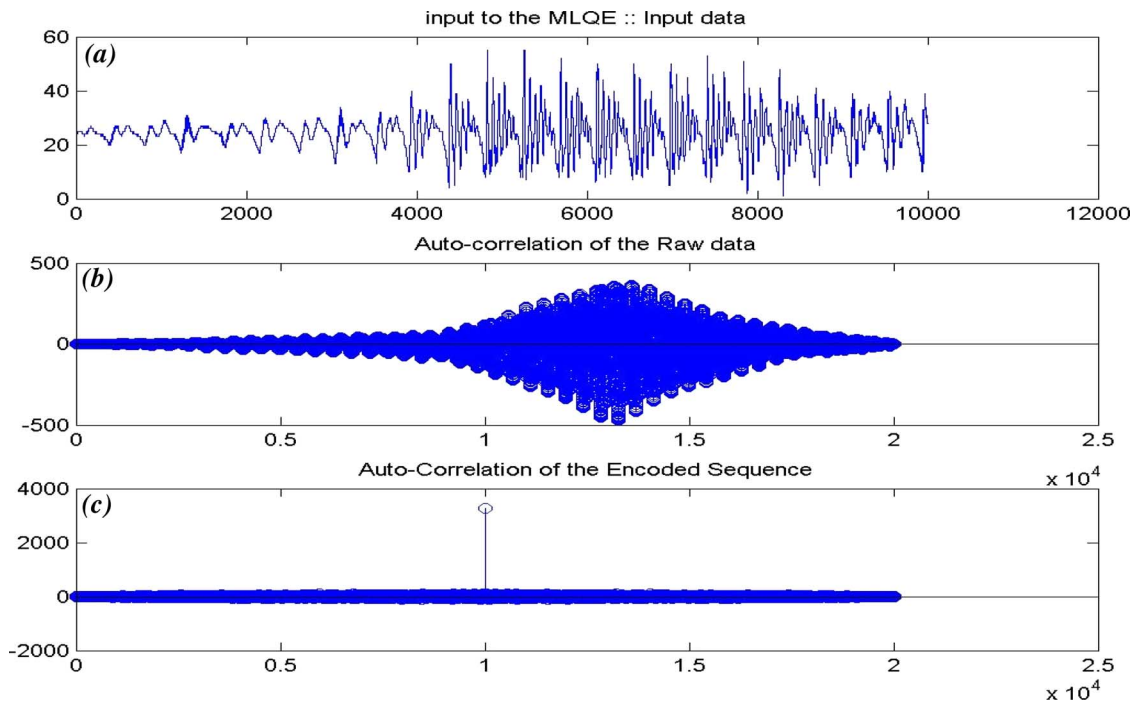


Fig. 11. Speech input and autocorrelation of input and output.

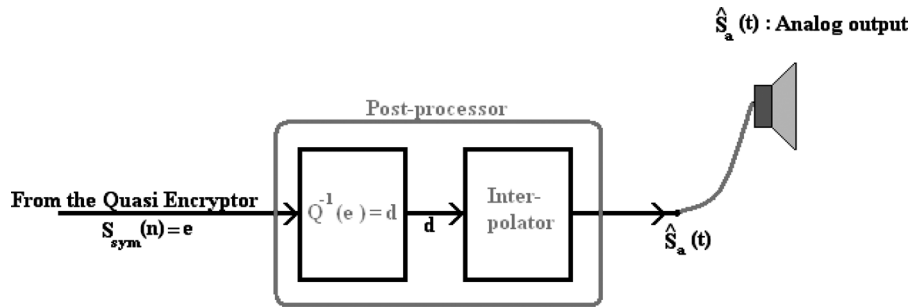


Fig. 12. Block diagram of a post-processor.

improve the speech clarity). If the pre-processor decimates the original samples by a factor r then the post-processor interpolates the sequence by the same factor r . This is done to counter the loss of speech information caused by the decimator. It would be difficult to interpret the decoded data without the knowledge of the inverse transform.

4) *Experimental Results:* It may be observed from Fig. 13 that the encoded sequence is highly randomized. However the decoded sequence is similar to the transmitted sequence. The autocorrelation property of the decoded sequence (Fig. 13 (c)) is slightly different than the autocorrelation of the input data; this is because of the mismatch in the interpolation factor r_1 and the decimation factor r_2 . It must be noted that the autocorrelation graphs cannot be identical because of the inherent loss of information in the transmitted data.

If one could find a loop in a quasigroup transformation that would facilitate the task of breaking it. But determining this is not easy and this task will equal the number of transformations associated with the quasigroup. Since a quasigroup is defined by its multiplication table which is a Latin square, its security would depend on the number of such Latin squares. The eavesdropper will have to guess the specific Latin square used out of the total number that could have been chosen.

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. A Latin square 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 $n \geq 2$, the total number $LS(n)$ of Latin squares of order n is given by

IX. SECURITY OF MLQE

$$LS(n) = n!(n - 1)!T(n)$$

A loop is a quasigroup with an identity element e :

$$x * e = x = e * x.$$

where $T(n)$ denotes the number of reduced Latin squares of order n . The numbers $T(n)$ and $LS(n)$ increase very quickly with n . Tables III–V give the number of reduced Latin squares (see

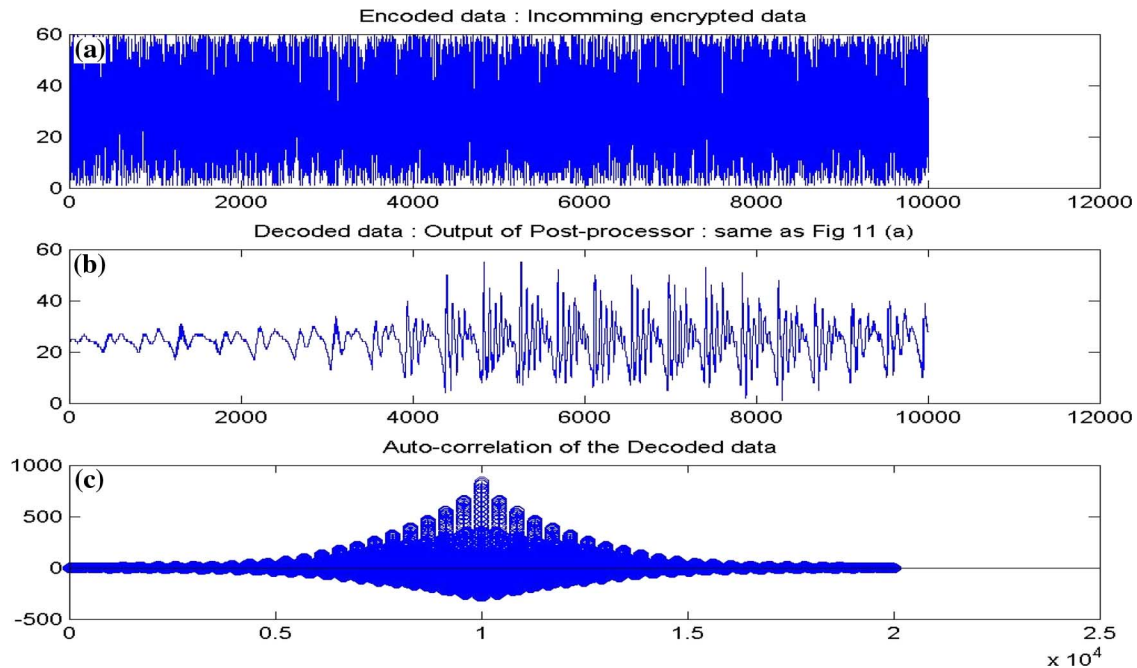


Fig. 13. Graphs of the encoded data, decoded: post-processor output and autocorrelation of decoded data.

TABLE III
REDUCED LATIN SQUARES FOR $2 \leq n \leq 10$

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 IV
REDUCED LATIN SQUARES 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	$1.50 \cdot 10^{86}$

[8] for details), where for large n , the number of Latin squares is estimated using the double bound:

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

It is clear from these tables that the task of breaking the quasigroup cipher is of astronomical complexity. If certain parameters of the design were to become available to the eavesdropper,

TABLE V
BOUNDS ON THE NUMBER OF LATIN SQUARES FOR $n = 16, 32, 64, 128, 256$

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

then the task of breaking it will become correspondingly less hard.

X. CONCLUSIONS

We have shown that quasigroup scrambling constitutes an excellent method of encryption and generation of pseudo-random sequences. Quasigroups (or Latin squares) provide a powerful method for generating a larger set of permutation transformations by permuting not only the samples but also transforming the amplitudes themselves across their range. By doing this, they provide an immensely large number of keys, even for small alphabets. The randomization obtained is very good, and, therefore, one can foresee practical applications of this method.

REFERENCES

- [1] R. A. Bailey and P. J. Cameron, "Latin Squares: Equivalents and Equivalence," Aug. 1, 2003, Technical Report.
- [2] S. Bakhtiari, R. Safavi-Naini, and J. Pieprzyk, "A message authentication code based on latin squares," in *Proc. Australasian Conference on Information Security and Privacy*, 1997, pp. 194–203.
- [3] H. Feistel, "Cryptography and computer security," *Scientific American*, May 1973.
- [4] D. Gligoroski, "Stream Cipher Based on Quasigroup String Transformations in Z_p^* ," arXiv: cs.CR/0403043, 2004.

- [5] D. Gligoroski, "Candidate One-Way Functions and One-Way Permutations Based on Quasigroup String Transformations," arXiv:cs.CR/0510018, 2005.
- [6] S. Kak and N. S. Jayant, "Speech encryption using waveform scrambling," *Bell System Technical Journal*, vol. 56, pp. 781–808, 1977.
- [7] S. Kak, "Overview of analogue signal encryption," in *IEE Proceedings F. Communications, Radar and Signal Processing*, 1983, vol. 130, pp. 399–404.
- [8] C. Kościelny, "Generating quasigroups for cryptographic applications," *Int. J. Appl. Math. Comput. Sci.*, vol. 12, no. 4, pp. 559–569, 2002.
- [9] S. Markovski, D. Gligoroski, and V. Bakeva, "Quasigroup string processing: Part 1," *Proc. of Maced. Acad. of Sci. and Arts for Math. and Tech. Sci.*, vol. XX 1-2, pp. 13–28, 1999.
- [10] S. Markovski and V. Kusakatov, "Quasigroup string processing: Part 2," *Proc. of Maced. Acad. of Sci. and Arts for Math. and Tech. Sci.*, vol. XXI, 2000, 1-2, 15–32.
- [11] T. Ritter, "Orthogonal latin squares, nonlinear balanced block mixers," Ritter Software Engineering Report, 1998.
- [12] C. P. Schnorr and S. Vaudenay, "Black box cryptanalysis of hash networks based on multipermutations," *Lecture Notes in Computer Science*, vol. 950, pp. 47–57, 1995.
- [13] S. Vaudenay, "On the need for multipermutations: Cryptanalysis of MD4 and SAFER," *Proc. Fast Software Encryption*, pp. 286–297, 1994.
- [14] S. Kak, "Encryption and error-correction using d-sequences," *IEEE Trans. Computers*, vol. C-34, pp. 803–809, 1985.
- [15] W. Stallings, *Cryptography and Network Security*. Upper Saddle River: Prentice Hall, 2006.
- [16] J. Hoffstein, J. Pipher, and J. H. Silverman, "NTRU: A ring based public key cryptosystem," in *Algorithmic Number Theory (ANTS III)*, Portland, OR, June 1998, J. P. Buhler, Ed. Berlin: Springer-Verlag, Jun. 1998, vol. Lecture Notes in Computer Science 1423, pp. 267–288.
- [17] N. Mandhani and S. Kak, "Watermarking using decimal sequences," *Cryptologia*, vol. 29, pp. 50–58, 2005.
- [18] K. Penumarthi and S. Kak, "Augmented watermarking," *Cryptologia*, vol. 30, pp. 173–180, 2006.
- [19] A. Parthasarathy and S. Kak, "An improved method of content based image watermarking," *IEEE Trans. Broadcasting*, vol. 53, pp. 468–479, 2007.
- [20] J. H. Park, H. J. Kim, M. H. Sung, and D. H. Lee, "Public key broadcast encryption schemes with shorter transmissions," *IEEE Trans. Broadcasting*, vol. 54, pp. 401–411, 2008.
- [21] X. Du, Y. Wang, J. Ge, and Y. Wang, "An ID-based broadcast encryption scheme for key distribution," *IEEE Trans. Broadcasting*, vol. 51, pp. 264–266, 2005.
- [22] Y. Nishimoto, A. Baba, T. Kurioka, and S. Namba, "A digital rights management system for digital broadcasting based on home servers," *IEEE Trans. Broadcasting*, vol. 52, pp. 167–172, 2006.

Maruti Satti did Bachelor of Technology in Electronics and Communications from Jawaharlal Nehru Technological University, Hyderabad. He completed his Masters in Electronic Engineering at Louisiana State University, Baton Rouge in 2007. His areas of interest include data security and communications.



Subhash Kak is professor and head of computer science department at Oklahoma State University in Stillwater. He has worked in the fields of cryptography, quantum information science, and neural networks.