Efficient Quasigroup Block Cipher for Sensor Networks

Matthew Battey  
Principle Solutions Architect  
Aspect Software, Inc.  
Omaha, NE 68164  
Email: mattbattey@cox.net

Abhishek Parakh  
Nebraska University Center for Information Assurance  
University of Nebraska at Omaha  
Omaha, NE 68182  
Email: aparakh@unomaha.edu

Abstract—We present a new quasigroup based block encryption system with and without cipher-block-chaining. We compare its performance against Advanced Encryption Standard-256 (AES256) bit algorithm using the NIST statistical test suite (NIST-STS) that tests for randomness of a sequence. Since it is well known that a good encryption algorithm must destroy any statistical properties of the input sequence and produce an output close to a true random sequence, the NIST-STS suite results provide a good test bench. In almost all tests from the suite the proposed algorithm performs better than AES256.

I. INTRODUCTION

Sensor networks provide a challenging area of research, because of constraints such as low computational power, low memory capacity and limited communication ranges. As their popularity for various applications such as border surveillance, patient health monitoring, surveillance and environment data collection increases, the demand for security and privacy also increases. Moreover, now a days smart-phones have several sensors within them that may be used to monitor health conditions and used for emergency purposes. In this case, security, integrity and privacy of data that is being transmitted are of utmost importance.

The most popular method for encryption in sensor networks is the use of secret key encryption systems such as Triple DES or AES [14], [13]. This is because secret key algorithms have much lower computational requirements compared with public-key systems, which is crucial especially in resource constrained environments such as sensor networks. In this paper we develop a new secret key encryption scheme, that is ideally suited for encryption in low computational and memory constrained environments. We run statistical tests on both the input and output streams, testing them for randomness using the NIST-STS package. The test results are compared with the popularly used Advanced Encryption Standard 256 (AES-256) bit encryption. The results show equal or better performance under all tests and that the encryption method is very good in destroying the structure of the input sequence.

Quasigroups are similar to Sudoku and Latin squares. They have been previously investigated for their application to encryption. Gligoroski et. al. [6], [8], [7] looked at stream cipher and public key implementations of quasigroups. A multi-level quasigroup implementation was proposed by Satti and Kak [16] where they used different sizes of quasigroups to encrypt data. They combined it with indices and nonces to improve on the strength of the encryption. However, their system also focuses on a stream cipher implementation. Marnas et. al. [11] implement a quasigroup all-or-nothing system. However, they only use quasigroup encryption to replace the XOR operation used within other all-or-nothing system, hence in the end the actual encryption is done using other cryptosystems. Quasigroups have also been applied to error correction [9] and in construction of message authentication codes (MAC) [11].

One may view quasigroup transformation as a substitution and permutation operation. These transforms form the basis of numerous encryption systems specially in speech encryption [12], [3]. Further, public key systems such as NTRU [10] and elliptic curve cryptosystems [2] have lower power consumptions compared to RSA however compared to secret key systems they are much more computationally expensive. Moreover, the algorithms proposed in this paper do not require any computations to be performed but only table look up operations for encryption and decryption.

To our knowledge, this is the first quasigroup block encryption algorithm similar in strength to AES with advantages in both computational and memory requirements over the latter.

II. BACKGROUND ON QUASIGROUPS

Quasigroups used in cryptography consist of an \( n \times n \) matrix consisting of permutations of elements of a finite field \( Z_n \) such that no element repeats in any row or column and all elements appear in every row and column. Here \( n \) is called the order of the quasigroup. Commonly chosen value of \( n \) is 256 such that it allows for us to work with the input stream at byte level. Quasigroups support an operation, denoted by \( \cdot \), for any two elements in the matrix such that a corresponding inverse operation, denoted by \( \backslash \), exists. For example, for any two elements \( x \) and \( y \), the following holds true: \( x \cdot y = x \cdot z \Rightarrow y = z \) and \( y \cdot x = z \cdot x \Rightarrow y = z \). Further, \( x \cdot y = z \) implies \( y = x \backslash z \).

The \( \cdot \) and \( \backslash \) operations are table lookup operations as illustrated by the following example of the working of a conventional quasigroup cipher implementation.

Example 1: Table 1 presents a quasigroup of order 6. The left most column and the top most row are index numbers. An
The proposed algorithm is very effective in randomizing the input data stream. However, given an input data stream and its corresponding output data stream a known plain text attack can be launched because $qGroup[c_{i-1}][m_i] = c_i$. If a long enough mapping is available it may be possible to fill in a significant number of elements in the quasigroup matrix, thereby decreasing the number of possibilities for the group. This is a weakness as the security of the above encryption depends on keeping the quasigroup secret.

### III. PROPOSED ALGORITHM 1: QUASIGROUP BLOCK CIPHER

In order to make quasigroup similar in functionality to the popular AES system, we use 32 different seeds for each round of encryption. Multiple rounds of encryption with different seeds in different rounds finesse the known-plaintext attack and provide a higher level of security, as in the case of Triple DES and AES. We choose 32 seeds, because we assume that each seed is one byte in size and 32 bytes is equal to 256 bits, which is the commonly used key length for AES systems.

In order to introduce dependencies between bytes of input data, we divide the data into 128 bit (16 byte) blocks and encrypt each block separately using Algorithm 1.

**Algorithm 1**

1. Construct a 256x256 size quasigroup.
2. Generate a random 256 bit encryption key and divide it into 8 bit (1 byte) blocks which will be used as seed elements at every round of encryption. This results in 32, 1 byte, seeds.
3. Divide the source data into 128 bit (16 byte) blocks.
4. For each block do the following:
   a) For each 8-bit block in the cipher key do the following:
      i) Using the current block as a stream of 16, 8-bit integers, apply the current 8-bit key as the quasigroup cipher seed and encrypt the block.
      ii) Left shift the currently encrypted block by 1, 3, 5 or 7 bits depending on the index of the current 8-bit key block modulo 4.
   b) For each seed block:
      i) For each block do the following:
         i) For each 8-bit block in the cipher key do the following:
            i) Using the current block as a stream of 16, 8-bit integers, apply the current 8-bit key as the quasigroup cipher seed and encrypt the block.
            ii) Left shift the currently encrypted block by 1, 3, 5 or 7 bits depending on the index of the current 8-bit key block modulo 4.
   c) Rotate the resulting 64 bit block by the appropriate distance:
         i) Rotate the 64 bit block by the appropriate distance.
   d) For each 8-bit block in the source data:
      i) Using the current block as a stream of 16, 8-bit integers, apply the current 8-bit key as the quasigroup cipher seed and encrypt the block.
      ii) Left shift the currently encrypted block by 1, 3, 5 or 7 bits depending on the index of the current 8-bit key block modulo 4.
   e) For each block do the following:
      i) Using the current block as a stream of 16, 8-bit integers, apply the current 8-bit key as the quasigroup cipher seed and encrypt the block.
      ii) Left shift the currently encrypted block by 1, 3, 5 or 7 bits depending on the index of the current 8-bit key block modulo 4.
   f) For each block do the following:
      i) Using the current block as a stream of 16, 8-bit integers, apply the current 8-bit key as the quasigroup cipher seed and encrypt the block.
      ii) Left shift the currently encrypted block by 1, 3, 5 or 7 bits depending on the index of the current 8-bit key block modulo 4.
   g) For each block do the following:
      i) Using the current block as a stream of 16, 8-bit integers, apply the current 8-bit key as the quasigroup cipher seed and encrypt the block.
      ii) Left shift the currently encrypted block by 1, 3, 5 or 7 bits depending on the index of the current 8-bit key block modulo 4.
   h) For each block do the following:
      i) Using the current block as a stream of 16, 8-bit integers, apply the current 8-bit key as the quasigroup cipher seed and encrypt the block.
      ii) Left shift the currently encrypted block by 1, 3, 5 or 7 bits depending on the index of the current 8-bit key block modulo 4.
   i) Rotate the resulting 64 bit block by the appropriate distance:
      i) Rotate the 64 bit block by the appropriate distance.
   j) For each 8-bit block in the source data:
      i) Using the current block as a stream of 16, 8-bit integers, apply the current 8-bit key as the quasigroup cipher seed and encrypt the block.
      ii) Left shift the currently encrypted block by 1, 3, 5 or 7 bits depending on the index of the current 8-bit key block modulo 4.
   k) For each block do the following:
      i) Using the current block as a stream of 16, 8-bit integers, apply the current 8-bit key as the quasigroup cipher seed and encrypt the block.
      ii) Left shift the currently encrypted block by 1, 3, 5 or 7 bits depending on the index of the current 8-bit key block modulo 4.
   l) Rotate the resulting 64 bit block by the appropriate distance:
      i) Rotate the 64 bit block by the appropriate distance.
   m) For each 8-bit block in the source data:
      i) Using the current block as a stream of 16, 8-bit integers, apply the current 8-bit key as the quasigroup cipher seed and encrypt the block.
      ii) Left shift the currently encrypted block by 1, 3, 5 or 7 bits depending on the index of the current 8-bit key block modulo 4.
   n) For each block do the following:
      i) Using the current block as a stream of 16, 8-bit integers, apply the current 8-bit key as the quasigroup cipher seed and encrypt the block.
      ii) Left shift the currently encrypted block by 1, 3, 5 or 7 bits depending on the index of the current 8-bit key block modulo 4.
   o) Rotate the resulting 64 bit block by the appropriate distance:
      i) Rotate the 64 bit block by the appropriate distance.
   p) For each 8-bit block in the source data:
      i) Using the current block as a stream of 16, 8-bit integers, apply the current 8-bit key as the quasigroup cipher seed and encrypt the block.
      ii) Left shift the currently encrypted block by 1, 3, 5 or 7 bits depending on the index of the current 8-bit key block modulo 4.
   q) For each block do the following:
      i) Using the current block as a stream of 16, 8-bit integers, apply the current 8-bit key as the quasigroup cipher seed and encrypt the block.
      ii) Left shift the currently encrypted block by 1, 3, 5 or 7 bits depending on the index of the current 8-bit key block modulo 4.
   r) Rotate the resulting 64 bit block by the appropriate distance:
      i) Rotate the 64 bit block by the appropriate distance.
   s) For each 8-bit block in the source data:
      i) Using the current block as a stream of 16, 8-bit integers, apply the current 8-bit key as the quasigroup cipher seed and encrypt the block.
      ii) Left shift the currently encrypted block by 1, 3, 5 or 7 bits depending on the index of the current 8-bit key block modulo 4.
   t) For each block do the following:
      i) Using the current block as a stream of 16, 8-bit integers, apply the current 8-bit key as the quasigroup cipher seed and encrypt the block.
      ii) Left shift the currently encrypted block by 1, 3, 5 or 7 bits depending on the index of the current 8-bit key block modulo 4.
   u) Rotate the resulting 64 bit block by the appropriate distance:
      i) Rotate the 64 bit block by the appropriate distance.
   v) For each 8-bit block in the source data:
      i) Using the current block as a stream of 16, 8-bit integers, apply the current 8-bit key as the quasigroup cipher seed and encrypt the block.
      ii) Left shift the currently encrypted block by 1, 3, 5 or 7 bits depending on the index of the current 8-bit key block modulo 4.
   w) For each block do the following:
      i) Using the current block as a stream of 16, 8-bit integers, apply the current 8-bit key as the quasigroup cipher seed and encrypt the block.
      ii) Left shift the currently encrypted block by 1, 3, 5 or 7 bits depending on the index of the current 8-bit key block modulo 4.
   x) Rotate the resulting 64 bit block by the appropriate distance:
      i) Rotate the 64 bit block by the appropriate distance.
   y) For each 8-bit block in the source data:
      i) Using the current block as a stream of 16, 8-bit integers, apply the current 8-bit key as the quasigroup cipher seed and encrypt the block.
      ii) Left shift the currently encrypted block by 1, 3, 5 or 7 bits depending on the index of the current 8-bit key block modulo 4.
   z) For each block do the following:
      i) Using the current block as a stream of 16, 8-bit integers, apply the current 8-bit key as the quasigroup cipher seed and encrypt the block.
      ii) Left shift the currently encrypted block by 1, 3, 5 or 7 bits depending on the index of the current 8-bit key block modulo 4.

**TABLE I**

| 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|
| 1 | 1 | 3 | 2 | 6 | 4 |
| 2 | 2 | 6 | 4 | 5 | 1 |
| 3 | 3 | 2 | 6 | 4 | 5 |
| 4 | 4 | 5 | 1 | 3 | 2 |
| 5 | 5 | 1 | 3 | 2 | 6 |
| 6 | 6 | 4 | 5 | 1 | 3 |

**TABLE II**

| 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|
| 1 | 1 | 3 | 2 | 6 | 4 |
| 2 | 5 | 1 | 6 | 3 | 4 |
| 3 | 6 | 2 | 1 | 4 | 5 |
| 4 | 4 | 3 | 5 | 6 | 1 |
| 5 | 5 | 1 | 3 | 2 | 6 |
| 6 | 6 | 4 | 5 | 1 | 3 |

The quasigroup ciphertext is constructed (table II). To construct the invQGroup[] matrix, do the following: in the $j$th column of the $i$th row in invQGroup[] matrix write the column number of element $j$ from the $i$th row in gGroup[].

To decrypt do the following:

1. $m_1 = \text{invQGroup}[s][c_1]$
2. For $i > 1$, do until all $c_i$s are decrypted
   a) $m_i = \text{invQGroup}[c_{i-1}][c_i]$

In general, the direct application of the above encryption algorithm is very effective in randomizing the input data stream.
For Each K in Key
  CipherText = QuasiGroupCipher(QGMS, K, CipherText)
  CipherText = LeftShift(CipherText, ShiftDistance[IndexOf(K,Key) Modulo 4])
Next K
Output[IndexOf(Block,Source)] = CipherText

The shift distances of 1, 3, 5, and 7 are each relatively prime to 2 and thus to 8 (size of a byte). Their sum is 16 (size of 2 bytes) and if each shift is applied 8 times, their sum becomes 128, which is equal to the block size of 128 bits (16 bytes) into which the input data was divided. Therefore, one full rotation of block occurs with shifts of 1, 3, 5 and 7 when all the 32 seeds are used. This ensures that all the bytes in the encrypted block become interdependent.

A. Test Implementation

A test implementation was written in C#.net, because of the popular adoption of the pre-existing AES cipher suite inbuilt in C#. Additionally, Microsoft Visual Studio 2010 has built in unit-testing facilities, which combined with Test-Driven-Development, produced well-tested code in reduced increments of time. The test implementation has the ability to overwrite the plaintext buffer, in place, limiting the memory footprint required to encode a buffer. The quasigroup matrix is generated using the Knuth/Fisher-Yates Shuffle [5]. Keys were generated using random-number generator, System.Random, allocating 16 random bytes per request. Both the encryption and decryption routines were constructed and tested.

B. Analysis

We used the National Institute of Technology - Statistical Test Suite (NIST-STS) suite to evaluate the randomness introduced by the system in the cipher. The NIST-STS package gives a P-value for various standardized tests. The P-value is the probability that a perfect random number generator would have produced a less random sequence than the one being tested [15]. Control tests were performed against the plain text source. The NIST-STS test suite is available freely in C source code, and downloadable from http://csrc.nist.gov/groups/ST/toolkit/rng/index.html. The tool can be configured to read a source file as a stream of bits, and evaluate the randomness of that stream. We report the results for the following tests - approximate entropy, block frequency, cumulative sums forward (CS-F) & cumulative sums reverse (CS-R), fast fourier transform, frequency, longest run, runs, rank and serial 1 and serial 2; where the parameters used for the tests are given in table [II-B].

Each test, upon successful completion, produced a P-value result which is to be interpreted as above. If a P-value for a test is determined to be equal to 1, then the sequence appears to have perfect randomness. A P-value of zero indicates that the sequence appears to be completely non-random [15]. However, both P-values of 1 and 0 are failure conditions in the tests.

Table [IV] shows the P-values for the various tests. In the table the first three columns show the average P-values for all zero (0x00) input, all 0xFF input and a text taken from Aesop fables (“From the Goose and the Golden Eggs”). The first column lists the various tests done, second column is the average P-values for encryption of all three inputs using quasigroups, third column is the average P-value for all three inputs using AES and the third column is the ratio of the P-value of encryption using quasigroups to that using AES multiplied by 100. The last four columns are P-values for all zero (0x00) and 0xFF inputs.

IV. PROPOSED ALGORITHM 2: QUASIGROUP BLOCK ENCRYPTION WITH CIPHER BLOCK CHAINING

To improve the performance of quasigroup block ciphers in the Approximate Entropy, Serial 1 and Serial 2 tests, we extended algorithm 1 to include cipher block chaining (CBC). Mathematically, CBC is written as:

$$C_0 := e(k, M_0 \oplus iv)$$
$$C_{n+1} := e(k, M_{n+1} \oplus C_n)$$

Where, $C_n$: an indexed cipher text block, $M_n$: an indexed plain text block, $k$: the cipher key (here seed), $iv$: A random initialization vector, where $|iv| = |C_n| = |M_n|$, $e(k,m)$: the encryption function, QGBC in this case.

A. Test Implementation

After implementing quasigroup block cipher with cipher block chaining, tests were repeated 20 times using a 256 bit random key (32, 1 byte seeds) each time. The resulting encrypted data was tested for randomness using the NIST-STS test suite, using the same parameters as before.

Table [V] compares a average P-value results from the NIST-STS test suite. The quasigroup block cipher with CBC outperformed AES256 with CBC in almost all cases.

It is to be noted that the variance of P-values between different test results may be misleading, as each test has different acceptance tolerance for P-values.

B. Test on Audio Input

Since sensors may be used to collect audio signals we perform the encryption operation using quasigroups on an audio input file. The source (taken from
### Table IV

| Test               | P-value for QG | P-value for AES | P-value QG as % of P-value of AES | All 0x00 input AES | All 0x00 input QG | All 0xFF input AES | All 0xFF input QG |
|--------------------|----------------|----------------|----------------------------------|--------------------|--------------------|--------------------|--------------------|
| Block Frequency    | 0.57189        | 0.53593        | 106.71                           | 0.59109            | 0.57530            | 0.48253            | 0.64041            |
| CS-F               | 0.47759        | 0.45340        | 105.33                           | 0.47739            | 0.42955            | 0.36766            | 0.50679            |
| CS-R               | 0.47995        | 0.46111        | 104.08                           | 0.48052            | 0.43870            | 0.36949            | 0.49906            |
| FFT                | 0.15798        | 0.15622        | 106.71                           | 0.03377            | 0.043198           | 0.05215            | 0.05501            |
| Frequency          | 0.40314        | 0.40006        | 100.77                           | 0.38935            | 0.34988            | 0.29779            | 0.39156            |
| Longest Run        | 0.30803        | 0.29188        | 105.53                           | 0.24881            | 0.21313            | 0.17118            | 0.27998            |
| Runs               | 0.40384        | 0.40136        | 100.62                           | 0.37347            | 0.37045            | 0.38143            | 0.35849            |

Table IV shows average P-values (over 20 runs) for Quasigroup encryption as compared to AES256 encryption system when the same encryption key is used for both cryptosystems without Cipher-Block-Chaining (CBC).

## Table V

| Test               | P-value for QG | P-value for AES-CBC | P-value QG as % of P-value of AES-CBC | All 0x00 input AES-CBC | All 0x00 input QG-CBC | All 0xFF input AES-CBC | All 0xFF input QG-CBC |
|--------------------|----------------|---------------------|---------------------------------------|------------------------|------------------------|------------------------|------------------------|
| Block Frequency    | 0.48822        | 0.51274             | 95.22                                 | 0.52155                | 0.47478                | 0.50250                | 0.48499                |
| CS-F               | 0.51939        | 0.50588             | 102.67                                | 0.50527                | 0.49851                | 0.48968                | 0.48843                |
| CS-R               | 0.52502        | 0.48904             | 107.36                                | 0.49205                | 0.51126                | 0.47860                | 0.49353                |
| FFT                | 0.50188        | 0.48532             | 103.41                                | 0.46172                | 0.48304                | 0.49187                | 0.49118                |
| Frequency          | 0.50190        | 0.47353             | 105.99                                | 0.48847                | 0.47584                | 0.46486                | 0.48745                |
| Longest Run        | 0.50468        | 0.47228             | 106.86                                | 0.47476                | 0.46822                | 0.46320                | 0.53736                |
| Runs               | 0.54392        | 0.51232             | 106.17                                | 0.53926                | 0.55004                | 0.51784                | 0.54467                |
| Serial 1           | 0.53517        | 0.53584             | 99.98                                 | 0.53300                | 0.51054                | 0.54146                | 0.56983                |
| Serial 2           | 0.51635        | 0.49246             | 104.85                                | 0.49903                | 0.52310                | 0.47274                | 0.51659                |

Table V shows average P-values (over 20 runs) for Quasigroup encryption as compared to AES256 encryption system when the same encryption key is used for both cryptosystems with Cipher-Block-Chaining (CBC).

We further perform a comparison of the randomness of the signal using the NIST-STS and tabulate the results for the various tests in Table [VI]. We see that in most cases the quasigroup block cipher with CBC randomizes the input waveform much more than AES256 does, especially in the case of Fast Fourier Transform (FFT) tests.

![Plot of original input audio waveform](http://www.nch.com.au/acm/11k16bitpcm.wav) and the encrypted audio waveforms are plotted in Figures [1](#) and [2](#) respectively. As we can see the quasigroup encryption system is very good at distributing the amplitude of the audio signal over the entire range.

### Table VI

| Tests               | P-value for AES-CBC | P-value for QG-CBC | P-value of QG-CBC as % of P-value of AES-CBC |
|---------------------|---------------------|---------------------|-----------------------------------------------|
| Approximate Entropy | 0.00240             | 0.00221             | 92.13                                         |
| Block Frequency     | 0.47894             | 0.46862             | 97.84                                         |
| CS-F                | 0.45362             | 0.50949             | 112.32                                        |
| CS-R                | 0.45870             | 0.49385             | 107.66                                        |
| FFT                 | 0.44155             | 0.49124             | 111.26                                        |
| Frequency           | 0.45255             | 0.50741             | 112.12                                        |
| Longest Run         | 0.47043             | 0.48993             | 104.15                                        |
| Rank                | 0.48931             | 0.46916             | 95.88                                         |
| Runs                | 0.48137             | 0.49307             | 102.43                                        |
| Serial 1            | 0.52025             | 0.50224             | 96.54                                         |
| Serial 2            | 0.50510             | 0.50571             | 100.12                                        |

Table VI shows P-values for the audio encryption using Quasigroup encryption and AES256.
V. On Theoretical Security of Quasigroup Ciphers

The total number of Latin squares of order \( n, n > 2 \), is given by \( LS(n) = n!(n - 1)!T(n) \), where \( T(n) \) denotes the number of reduced Latin squares of order \( n \). The numbers \( T(n) \) and \( LS(n) \) increase very quickly with \( n \) [16]. Table VIII gives the number of reduced Latin squares.

From table VIII we see that the number of possibilities for the Latin square is astronomical. Therefore, if the quasigroup is kept secret along with the 256 bit key (32 random seeds) the system provides very good security.

VI. Conclusion and Future Work

In this paper we have proposed algorithms for implementation of quasigroup block cipher. The strength of the algorithms was assessed by assessing the randomizing property of the system and the use of statistical test suite by NIST (NIST-STS). Results of the simulations are tabulated and it is observed that in almost all the cases the output generated by the quasigroup encryption system is as or more random than that produced by AES256 for the same encryption key used. The results presented were for average P-values over 20 runs for all zero (0x00) input, all 0xFF input, and an Aesop fable. We also performed tests on audio input and results have been presented.

In future work, we intend to perform cryptanalytic attacks on the proposed quasigroup algorithms. We would also like to make the quasigroup matrix public with only 32 seeds kept secret. A research question would be to see what is a good candidate for a quasigroup (out of numerous possibilities) when it is public [4]. We also intend to look into FPGA implementations of the proposed system.

REFERENCES

[1] S. Bakhtiari, R. Safavi-Naini, and J. Pieprzyk. A message authentication code based on latin squares. In Proceedings of the Second Australasian Conference on Information Security and Privacy, ACISP ’97, pages 194–203, London, UK, UK, 1997. Springer-Verlag.
[2] I. Blake, G. Seroussi, N. Smart, and J. W. S. Cassels. Advances in Elliptic Curve Cryptography (London Mathematical Society Lecture Note Series). Cambridge University Press, New York, NY, USA, 2005.
[3] S. Borujeni. Speech encryption based on fast fourier transform permutation. In Electronics, Circuits and Systems, 2000. ICECS 2000. The 7th IEEE International Conference on, volume 1, pages 290 –293 vol.1, 2000.
[4] J. Dvorsky, E. Ochodkova and, and V. Sna ands andel. Quasigroups with good statistical properties. In Computer Information Systems and Industrial Management Applications (CISIM), 2010 International Conference on, pages 244 –249, oct. 2010.
[5] R. Fisher and F. Yates. Statistical Tables for Biological, Agricultural, and Medical Research. Oliver and Boyd, sixth edition, 1963.
[6] D. Gligoroski. Stream cipher based on quasigroup string transformations in \( zp^m \). CoRR, cs.CR/0403043, 2004.
[7] D. Gligoroski. Candidate one-way functions and one-way permutations based on quasigroup string transformations. IACR Cryptology ePrint Archive, 2005:352, 2005.
[8] D. Gligoroski, S. Markovski, and S. J. Knapskog. Public key block cipher based on multivariate quadratic quasigroups. IACR Cryptology ePrint Archive, 2008:320, 2008.
[9] D. Gligoroski, S. Markovski, and L. Kocarev. Error-correcting codes based on quasigroups. In Computer Communications and Networks, 2007. ICCCN 2007. Proceedings of 16th International Conference on, pages 165 –172, aug. 2007.
[10] J. Hoffstein, J. Pipher, and J. H. Silverman. Ntru: A ring-based public key cryptosystem. Algorithmic number theory, 1423:267–288, 1998.
[11] S. I. Marnas, L. Angelis, and G. L. Blairis. An application of quasigroups in all-or-nothing transform. Cryptologia, 31(2):133–142, Apr. 2007.
[12] E. Mosa, N. Messihia, and O. Zahran. Chaotic encryption of speech signals in transform domains. In Computer Engineering Systems, 2009. ICCESS 2009. International Conference on, pages 300 –305, dec. 2009.
[13] A. Parakh and S. Kak. Efficient key management in sensor networks. In GLOBECOM Workshops (GC Wkshps), 2010 IEEE, pages 1539 –1544, dec. 2010.
[14] A. Parakh and S. Kak. Matrix based key agreement algorithms for sensor networks. In Advanced Networks and Telecommunication Systems (ANTS), 2011 IEEE 5th International Conference on, pages 1 –3, dec. 2011.
[15] A. Rukhin, J. Soto, J. Nechvatal, E. Barker, S. Leigh, M. Levenson, D. Banks, A. Heckert, J. Dray, S. Vo, A. Rukhin, J. Soto, M. Smid, S. Leigh, M. Vangel, A. Heckert, J. Dray, and L. E. B. Lii. A statistical test suite for random and pseudorandom number generators for cryptographic applications, 2001.
[16] M. Satti and S. Kak. Multilevel indexed quasigroup encryption for data and speech. Broadcasting. IEEE Transactions on, 55(2):270 –281, June 2009.