Creating a Framework to Strengthen Cryptographic Security for Group Communication by Distributing Processed Keys

[1] Bilas Haldar, [2] Rohit Sinha, [3] Pranam Paul * (Corresponding Author)
[1] CSE, The Neotia University, [2] AI Engineer, Ailabs, [3] CSE, The Neotia University
[1] bilasphd2020@gmail.com, [2] rohit123sinha456@gmail.com, [3] pranam.paul@gmail.com

Article Info

Page Number: 4673 - 4679

Publication Issue:

Vol 71 No. 4 (2022)

Article History

Article Received*: 25 March 2022*

Revised*: 30 April 2022*

Accepted*: 15 June 2022*

Publication*: 19 August 2022*

Abstract

In recent years the amount of secure transactions taking place over the internet has increased exponentially. The security of these transactions depends on the strength of the encryption and decryption techniques, and what lies at the heart of these techniques is sharing keys in a secure fashion. The primary goal of key exchange methods is to provide inherit security to the techniques and not depends on the security of the communication channels. In this work, we propose novel methods for generating keys for multi-party communication using the nth order of Quasi Group of numbers with arrangement and rearrangement. After extensive experimentation it was found that the proposed method have improved performance in comparison to benchmark tests in terms of time complexity and better security against standard attacks.

Index Terms— Cryptography, Quasi Group, Multiparty Authentication, Key Generation, Key Regeneration.

I. INTRODUCTION

A Key Exchange Protocol (KEP) deals with generation, sharing and authentication of keys between two parties. However, with the ever growing landscape of communication and the morphing of the usage of the internet multi-party key exchange and authentication are becoming more and more critical.

In a physical setting, when multiple parties are involved in a transaction, authentication and authorization of each party is required for the transaction to succeed. For example, clearing of fund from a joint account, Issuance of order across a chain of command and many more. However, in case of electronic representation of such transactions only one person is entrusted to carry out the transaction. This create a inherent lack of transparency in electronic mode of transaction. To the extent of addressing this issue, the problem of multi-party authentication and authorization problem is formed by us. Over the last decades, key exchange between two parties have been the major focus of research. Most of the work done in the domain of key exchange inherently base their security on the Hardness of Arithmetic and Geometric Problems. To the best of our knowledge our work is among the first few to adapt the Hardness of combinatorics problems into a key exchange protocol. In this work we propose a method that exploits the nth order of Quasi Group of numbers with arrangement and rearrangement and same non-overlapping block of cells. From Generalized Quasi-group, a quasi-group with unique solution is generated and the values of the empty places are transformed in a row priority fashion so as to generate a master key. Form this master key we generate multiple child keys for multi-party authentication. From the child keys the generating transformation are applied in

reverse for authentication. To summarize our contribution, we formalize the problem of nth order of Quasi Group of numbers as a bijective function so as can be used for key generation and validation. We propose a row priority algorithm for generation of a master key from an unordered hole values of a balanced nth order Quasi Group of numbers. We formalize and propose the algorithm for generation and regeneration multiple child keys from a master key and vice-versa respectively.

II. Literature Review

B. Indrani et al. provided an efficient and fast key pool generation algorithm using the QG solving mechanisms for generating the cipher keys [1]. The work done by T. Sivakumar presented a novel method to generate random keys using a word grid puzzle which is a novel idea for generating random key streams [2]. A new key generation approach is presented by C. Manikandan et al. using values produced from QG matrices and synthetic colour images [3]. Evolutionary computation can be successfully applied in cryptography, as demonstrated by K. Knezevi et al. using asymmetric key cryptography, symmetric key cryptography, and pseudo-random number generator [4]. The key chains constructed in the work of K. G. Srinivasa et al. were based on multiple key spaces instead of a single key-space. [5]. An analysis of deterministic, probabilistic, and hybrid key distribution 1 algorithms for signaling pair-wise, group-wise, and network-wise keys was conducted by S. Khalid et al. that improves network resiliency and provides sufficient security [6]. The work done by C. Ansotegui et al. addresses QG problems using two techniques, namely Constraint Satisfaction Problems (CSPs) and Satisfiability Problems (SATs) [7].

III. Methodology

Key exchange forms an integral part of secure communication. The existing literature in the key generation and key exchange are mostly based on using mathematical Hard problems. In this proposed work we put forward a novel method that uses Hard problems in conjunction with exponential number based combinatorial problems for key generation.

A. Generating a nth order Quasi Group

The brute force approach to generate a nth order of Quasi Group of numbers is to create n×n grid following the constraints of a solved QG, followed by randomly removing values from the grid and leaving the space blank. However, one critical problem that such method induces is the existence of a non-unique solution of the puzzle. Even if a nth order of Quasi Group of numbers with unique solution is generated by multiple repetition of the brute-force method, it does not guarantee that the generated puzzle is balances and is Hard.

The work done by Ansotegui et al. [7] defines a Latin Square (LS), or Quasigroup, of order n, is an matrix, with each of its n 2 cells filled with one of n symbols, such that no symbol is repeated in a row or column with the additional restriction that each symbol occurs exactly once in each contiguous set of n non-overlapping pre-defined cells henceforth referred as a block region. There are exactly s block regions in a QG of order s. With respect to the worst-case complexity, they prove that QG with block regions of m rows and n columns with m = n is NP-Hard. In our proposed method we follow the work done by Ans´otegui et al. [7] with minor modifications to generate. So to generate a generalized nth order of Quasi Group of numbers with arrangement and Rearrangement with holes (empty spaces). We start by generating a valid and complete Quasi Group (QG) [8]. To generate a QG as described by [8] an initial n × n Latin Square (LS) is taken. By following a uniform distribution with some random moves defined between the visited LS, all the n^2 spaces of LS is visited. These moves define a chain of states, and transition from one state to another is independent, thus making a Markov Chain (MC).

Once a valid QG is generated, a nth order Quasi Group with Holes can be generated by removing values from random cells of the QG [9]. However, this creates an imbalance and makes it easier to

solve. To resolve this a balanced QG needs to be created (i.e. number of empty spaces in rows and columns and block region).

To generate balanced hole in QG that has a unique solution [7], the set total number of holes H is taken in the range H = {1…81}. Iterating over the set H, the total number of holes can be distributed over every row, every column, every block region using a regular bipartite graph generation algorithm based on Markov Chain Algorithm [10].

The algorithm 1 is used to generate a QG with h holes S for which there exists a unique solution. The functions used in Algorithm 1 are deterministic in nature.

Algorithm 1 Generating QG with Unique Solution

```
H ← {1 · · · 81}
S ← generateQG()
for H as h do
    X ← createQGwithEmpties(S, h)
    if existsUniqueSolution(X) is True then
       return(X,h)
    end if
end for
```

B. Generating the Master Key

Once the QG S is generated and the number of holes h for which an unique solution exists is determined, now the master key is created.

To create the master key, holes h are included in the QG S and the values of the QG that were replaced are appended together to form a list referred to as the masker keys.

The method createQGwithEmpties() in Algorithm 1 returns two things. First, the original QG S itself. Secondly, a list of tuples of the indices of the holes LH = $\{(i_1, j_1),(i_2, j_2), \cdots (i_k, j_k)\}$. Let's say we have sequence M known as master keys, where the ith element is denoted as m_i . The Equation 1 can be used to compute the master key for our proposed key generation method.

$$m_i = S_{a,b} \forall 1 \leq i \leq |LH| \quad (1)$$

Where, a is the 0th element and b is the 1st element of the ith tuple of LH respectively.

Algorithm 2 Generating the Master Keys

```
Require: M[]
    H ← {1 · · · 81}
    Se, LH ← createQGwithEmpties(S, h)
    for Each LH do
        a = LH[0]
        b = LH[1]
        M[i] = S[a][b]
    end for
```

In Algorithm 2, the sequence of master keys are generated. It's an implementation of the Equation 1. The elements of the QG are taken in a row priority fashion for generating the Master Key M.

C. Generating the Keys

After the master keys are generated, now we generate the keys for the preferred number of users. Since, the proposed method is multi party key generation and authentication technique, we select a minimum number of users required to pass the authorization. Let's say that the minimum number of user is u.

Algorithm 3 Generating the Keys

u ← Minimum Number of Users
indices[]
for 1 · · · u as i do
$r_{indices}$ = RandomNumberofLength(u)
if $r_{indices}$ not in indices[] then
OTP $User_i$ = M − {M[$r_{indices}$]}
indices[] = $r_{indices}$
end if
end for

To expand Algorithm 3, for every number of minimum users |u| number of random numbers are chosen. If the chosen random number are distinct and are unique compared to previously chosen random numbers, then these indices are removed from the Master key list M and the remaining of the Master list is appended together to form one single string OTP User. The constraint of the random numbers rn has to follow is $1 \leq rn \leq |M|$.

D. Regenerating the Keys

For Authentication of the OTPs that the users input and authorization, the master key is regenerated from the OTPs entered by the user. Let's say u users enters the OTP. For representation let's consider i^{th} user enters OTP_i

Algorithm 4 Generating the Keys

Require: M
for all OTP_i do
for 1 · · · |OTP_i | as k do
if OTP_i [k] == M[k] then
Regen[k] ← OT P_i [k]
end if
end for
end for

For Regenerating the Master Key the Algorithm 4 is defined. In 4 we are iterating over all the OTPs entered by the user and performing a index wise check with the master key. If the value matches we are entering the value. On completion of this Algorithm we have a list Regen that would be exactly similar to the Master Key M if all of the user enter their correct OTP.

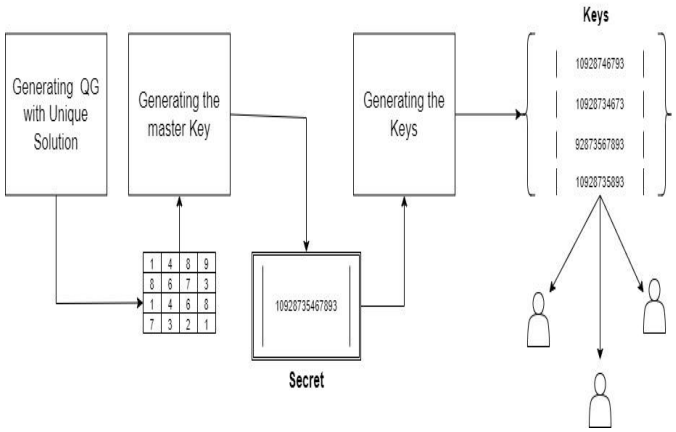

Figure 1: An Example of Key Generation

E. Authentication

For Authentication we fill up the Generalised QG Se as per Algorithm 2 with the Regnerated keys Regen in a row priority order, as we did while generating the Master Key M in Section III.B. If the original Generalised QG S and the regenerated QG are element-wise similar then only the Authentication succeeds.

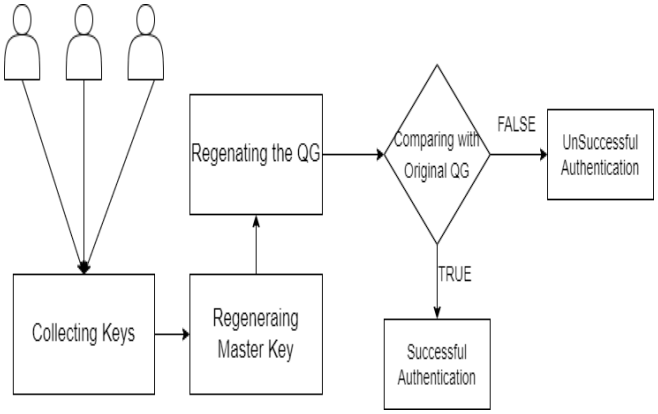

Figure 2: An Example of Key Authentication

IV. RESULTS

In this section, we present the results of our proposed algorithm. In addition to demonstrating the feasibility of the proposed algorithm, that has been implemented using Python to generate keys, distribute keys, and also validate keys. With the fewest possible users, we evaluate the performance of our method in terms of key generation and distribution timeframes. We also check how long it takes to regenerate the original key. Table 1 shows the important key generation time; distribution time, and original key regeneration time based on the minimum number of users. The first column from the left in this table shows the minimum number of users in the range of two to ten. In the same table, the key generation and distribution time is listed in the second column from the left. It has varying between 0.0015534 and 0.0107373. The original key regeneration time is indicated in the third column from the left in Table 1. It has been varying from 0.0006566 to 0.0051109.

Minimum Number of Users	Key Generation and Distribution Time	Original Key regeneration Time
2	0.00198	0.00086
3	0.00155	0.00066
4	0.00242	0.00109
5	0.00766	0.00487
6	0.00354	0.00155
7	0.00416	0.00091
8	0.01074	0.00511
9	0.00592	0.00412
10	0.00462	0.00477

Table 1: Relation between the minimum number of users along with key generation, distribution time, and original key regeneration time.

V. ANALYSIS

The experimental analysis of our proposed approach is presented in this section. We have analyzed all of the results of section IV with the minimum number of users. We use a graph to analyze the result of key generation time and key regeneration time.

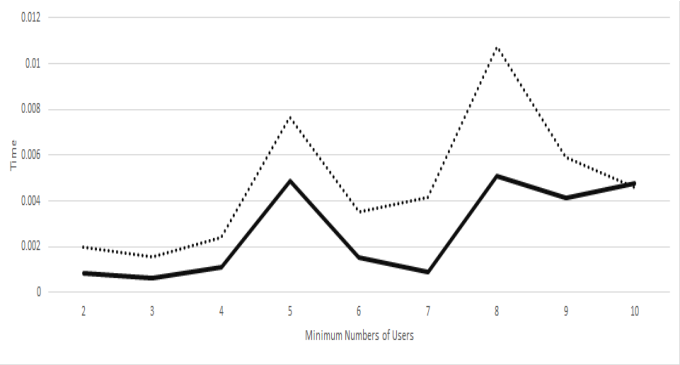

Figure 3: Minimum Number of Users vs Key Generation and Regeneration time

A graphical representation for Table 1 is shown in Figure 3 with a continuous solid line and dotted line. In this figure dotted line indicates the key generation and distribution time based on the number of users and the solid line indicates the key regeneration time. It has been observed that the two lines in the graph are almost the same in nature. According to the graph, there is a tendency that key generation and distribution time to fluctuate between 0.0015534 and 0.0107373 whereas key regeneration time varies between 0.0006566 to 0.0051109. Here, it generates master keys randomly that are also affected in the graph. It is analyzed that if the number of users increases then key generation time sometimes increases and also sometimes decreases that impact also effected in key regeneration time. because key generation time depends on the value of prime numbers but not the number of the prime numbers. It has been observed from the graph that both the curves are almost the same in nature based on the number of users. So, it takes the almost same time for key generation and also key regeneration time based on number of users.

CONCLUSION

In this work we have formalized the problem of multi-party authentication and authorization problem and proposed a method using the nth order of Quasi Group of numbers with arrangement and rearrangement to address the problem. Even though the proposed method performed quite well under performance testing, the domain of multi-party authentication and authorization have a lot of room for significant progress. The future direction of the work would be to include improved version of the proposed algorithms but optimizing performance and efficiency.

REFERENCES

[1] B. Indrani and M. K. Veni, “An efficient algorithm for key generation in advance encryption standard using sudoku solving method,” in 2017 International Conference on Inventive Systems and Control (ICISC), pp. 1–8, IEEE, 2017.
[2] T. Sivakumar, S. Veeramani, and T. Anusha, “Generation of random key stream using word grid puzzle for the applications of cryptography,” WSEAS Transactions on Computers, vol. 20, pp. 1–9, 2021.
[3] C. Manikandan, K. S. S. Satwik, T. Smarani, and P. Umamaheshwari, “A combined sudoku and synthetic colour image techniques for cryptographic key generation,” in Journal of Physics: Conference Series, vol. 1767, p. 012050, IOP Publishing, 2021.

[4] K. Kneˇzevi´c, “Combinatorial optimization in cryptography,” in 2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), pp. 1324–1330, IEEE, 2017.
[5] K. Srinivasa, V. Poornima, V. Archana, C. Reshma, K. Venugopal, and L. Patnaik, “Combinatorial ap proach to key generation using multiple key spaces for wireless sensor networks,” in 2008 16th International Conference on Advanced Computing and Communications, pp. 279–284, IEEE, 2008.
[6] F. A. Saba Khalid and M. R. Beg, “Secure key pre-distribution in wireless sensor networks using combina torial design and traversal design based key distribution,” 2012.
[7] C. Ans´otegui, R. B´ejar, C. Fern´andez, C. Gomes, and C. Mateu, “Generating highly balanced sudoku problems as hard problems,” Journal of Heuristics, vol. 17, no. 5, pp. 589–614, 2011.
[8] M. T. Jacobson and P. Matthews, “Generating uniformly distributed random latin squares,” Journal of Combinatorial Designs, vol. 4, no. 6, pp. 405–437, 1996.
[9] R. Lewis, “Metaheuristics can solve sudoku puzzles,” Journal of heuristics, vol. 13, no. 4, pp. 387–401, 2007.
[10] R. Kannan, P. Tetali, and S. Vempala, “Simple markov-chain algorithms for generating bipartite graphs and tournaments,” Random Structures & Algorithms, vol. 14, no. 4, pp. 293–308, 1999.