Journal of Theoretical and Applied Information Technology
20th February 2014. Vol. 60 No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

A HYBRID INDEXED TABLE AND QUASIGROUP ENCRYPTION APPROACH FOR CODE SECURITY AGAINST VARIOUS SOFTWARE THREATS

[1]Dr.N. SASIREKHA, [2]Dr. M.HEMALATHA

[1]Assistant Professor and Head, Department of Computer Science, Rathinam College of Arts and Science, Coimbatore, Tamilnadu, India
[2]Department of Computer Science, Karpagam University,Coimbatore,Tamilnadu, Inida
E-mail: [1]nsasirekhaudt@gmail.com , [2]hema.bioinf@gmail.com

ABSTRACT

Security is of fundamental importance in digital communication. The system should be secure against brute force attacks and impersonation by the eavesdropper. Code and the sensitive data should only be accessed and understood by legitimate user/authority. Software in recent scenario has been highly susceptible to various attacks and threats. Reverse engineering is one of the key technique by which an intruder can understand the inner working of the software. Most of the existing software protection techniques do not provide reliable security against various attacks. Cryptographic approaches are observed to be very efficient in providing security and authentication to the software. Encryption of the code in the software has received much attention in the software engineering domain and various researches are being done in that area. This paper proposes a novel hybrid software protection code encryption scheme based on the index table. This approach uses a novel and efficient encryption technique for encryption the indexed table. The encryption technique used is the quasigroup approach which provides least resemblance of the original data when encrypted. The performance of the proposed approach is evaluated based on the time cost and space cost and it is observed that the proposed approach provides significant results and performance.

Keywords: *Cryptography, Decryption,Encryption, Quasigroup, Index Table, Reverse Engineering, Software Protection.*

1. INTRODUCTION

Software is a form of data and it as very much susceptible to theft and misuse. Most of the organizations have great concern about their software security [1, 2]. A secret algorithm that is mined and reused by an intruder can have major consequences for software companies. Moreover secret keys, confidential data or security related code are very much susceptible to the attacks and threats [3]. Even if legal actions like patenting and cyber crime laws are available, software threats and attacks still remains a substantial threat to software developers and security experts. There have been billions of dollars spent each year by the industries especially for software piracy and digital media piracy [4].

Protecting the reliability of software platforms, particularly in unmanaged customer computing systems is a tough task [5]. Attackers may try to carry out buffer overflow attacks to look for the right of entry to systems, steal secrets and patch on the available binaries to hide detection. Software protection has become one of the attractive domains with high commercial interest [6].

Reverse engineering by obfuscation, modification by software tamper resistance, program-based attacks by software diversity and BORE – break-once run everywhere – attacks by architectural design [7] are the major attacks on the software. Clearly, there is a considerable need for developing more efficient approaches to protect software. However, most of the existing approaches utilized by software developers do not offer significant protection, especially on recent computing platforms [8]. For protecting and securing data in networked systems, several protection approaches such as cryptographic controls, access controls, information flow controls, inference controls were used by the researchers. Among these techniques, cryptographic approaches have received the greatest academic attention, because of its classic mathematical data-manipulation algorithms involving secret keys, encryption algorithms for confidentiality and Message Authentication Codes (MACs) and digital

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

signature algorithms for real-time authentication, data origin authentication, integrity or non-repudiation [9]. Therefore, Cryptography is observed to be the technique that can be incorporated in the software protection technique for improved protection [10].

Software protection comprises of a wide range of principles, approaches and techniques focused to enhance software security, providing increased protection against threats ranging from buffer overflow attacks [11] to reverse engineering and tampering [12]. For decades encryption has provided the means to hide information. In this research, the self-encrypting code is used as a means of software protection [13]. In this research work, the concept of efficient code encryption techniques, which offers confidentiality and a method to create code dependencies that implicitly protect integrity need to be established.

This paper proposes an efficient code encryption technique based on an index table. The encryption technique used in this approach is the quasigroup approach for encrypting the indexed table data to make it tough for the intruder to hack the data.

2. LITERATURE SURVEY

Collberg et al. [14] provided a compact outline of the approaches to protect against these threats. Software watermarking for instance focuses on protecting software reactively against piracy. It usually implants hidden, distinctive data into an application in such a way that it can be guaranteed that a particular software instance belongs to a particular individual or company. When this data is distinctive for each example, one can mark out copied software to the source unless the watermark is smashed. The second group, code obfuscation, protects the software from reverse engineering attacks. This approach comprises of one or more program alterations that alter a program in such a way that its functionality remains identical but analyzing the internals of the program becomes very tough. A third group of approaches focuses to make software "tamper-proof", also called tamper-resistant.

Cappaert et al. [15] [22][23] presented a partial encryption approach depending on a code encryption approach. In order to utilize the partial encryption approach, binary codes are partitioned into small segments and encrypted. The encrypted binary codes are decrypted at runtime by users. Thus, the partial encryption overcomes the faults of illuminating all of the binary code at once as only the essential segments of the code are decrypted at runtime.

Jung et al. [16][22][23] presented a code block encryption approach to protect software using a key chain. Jung's approach uses a unit block, that is, a fixed-size block, rather than a basic block, which is a variable-size block. Basic blocks refer to the segments of codes that are partitioned by control transformation operations, such as "jump" and "branch" commands, in assembly code. Jung's approach is very similar to Cappaert's scheme. Jung's approach tries to solve the issue of Cappaert's approach. If a block is invoked by more than two preceding blocks, the invoked block is duplicated.

However, the above discussed schemes did not meet the security requirements and moreover had an efficiency problem. Moreover, time cost and space cost should also be taken into consideration. Thus, a novel cryptographic technique is proposed in this approach.

3. METHODOLOGY

A code encryption scheme is proposed based on an indexed table to protect software. The indexed table can solve the problem of multiple paths. Moreover, it solves such problems as loops, recursions, and multiple calls.

Step 1: Compilation of source code. After this step, the source code is compiled and outputs a binary image.

Step 2: Construction of the indexed table. It is the most important procedure of our scheme. We describe the details of step 2 in Figure 1.

A. Construction of Index Table

The correct key chain is obtained by means of the indexed table. The construction of the index table follows the set of procedure [18].

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

```
Procedure ConstructTable()
1. entrypoint ← Find_EntryPoint(); // store an address of entry point
2. currentPointer ← entrypoint;
3. nextPointer ← currentPointer++;
4. index ← 0;
5.
6. while(File pointer is not end of file) {
7. if(Current_opcode == jump or Current_opcode == branch){
8. // branch or jump command is an unit of block
9. nextAddress ← operand;
10. // store an address of current address
11. if(currentAddress == nextAddress) {
12. // loop, or recursion
13. Tuple[index].Address ← currentAddress;
14. Tuple[index].Size ← sizeofBlock;
15. Tuple[index].Cnt ← prev_operand_2;
16. Tuple[index].flag ← 0;
17. //index, entry point address, size, number of calling, and no protected key
18. StoreAttribute(Tuple[index]);
19. }
20. else{
21. if(FindAddress(Tuple[index].currentAddress) {
22. // repeated calling
23. GenerateProtectedKey();
24. StoretoDatasection();
25. Tuple[index].flag ← 1
26. }
27. }
28. }
29. nextBlock ← Get_NextBlock(currentAddress);
30. // Get next block's address
31. currentAddress ← nextAddress;
32. Sort(Tuple);
```

Figure 1: Pseudo-code to Construct Indexed Table

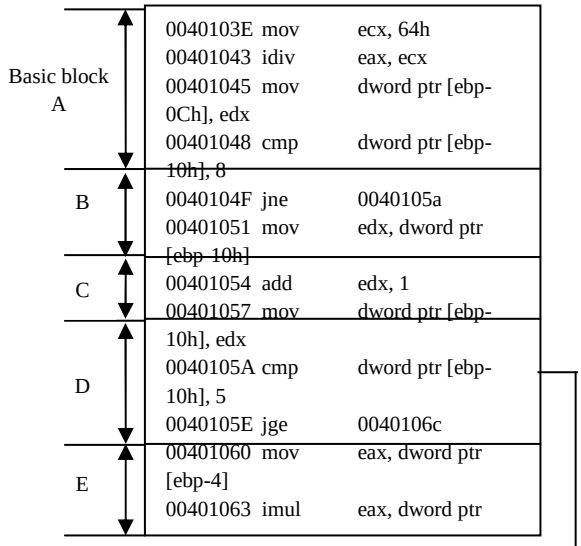
Initially, the present address of the basic block is stored, and the "*jump*" or "*branch*" command in the basic block is examined by moving the pointer. The commands consist of a block's address, which will be executed in the subsequent step. If the next address refers to the present address of the basic block, this shows a loop or recursion. When a loop or a recursion takes place because of the "*cmp*" command with the number of calls, the number of calls is marked in the table. Similarly, if a current address of a block is already stored in the table, this shows multiple calls. The *PK* is created at this time and stored in the binary image in a data section [18].

At this moment, random number *r* is created for the secret key of *D*, and then it is encrypted with *IK*. The result of the encryption is *PK*. The *PK* is stored in executable images. Generic operating systems, such as Windows or Linux, store variables in the data section of an executable image. Thus, the *PK* is stored in the data section of an executable image. The indexed table consists of the number of iterations and recursions. If this is not taken into account, a basic block which has loops or/and recursions will be decrypted several times. Thus, if the number of loops and recursions in the table is marked, this problem can be prevented. When a basic block has been called, the number of calls is minimized by one, and then if the number of calls became zero, the block should be re-encrypted from memory to prevent against a memory dump.

The second operand of the "*cmp*" command is 0Ah. It shows the block "loc_401006" will be executed 10 (=0x0A) times, and that is the number of loops or recursions. Moreover, an example of constructing the indexed table is shown in Figure 2. The example code consists of five basic blocks. The basic blocks are partitioned by "*jump*" or "*branch*" commands. In the beginning, initialization is carried out to construct the indexed table (Sungkyu Cho et al. 2011). 0x0040103E is set as the starting point of the program. Then, the commands are examined to discover the "*jump*" or "*branch*." If the command is "*jump*" or "*branch*," store the operand of the command in the table as it becomes the first address of another block. In this example, 0x0040105A is stored in the table due to the command "*jne* 0x0040105A," which is at0x0040104F. The next address of the command becomes the first address of another block. So, 0x00401051 is stored in the indexed table. Thus, 0x0040106C and 0x00401060 are stored in order. At 0x0040106A, the command "*jmp* 0x00401051" is discovered. 0x00401051 has been stored already, which shows that there are multiple paths taking into account the address 0x00401051. Thus, the block's data should be updated, and the random number should also be created. Thus, all the blocks can be identified (Sungkyu Cho et al. 2011).

Journal of Theoretical and Applied Information Technology
20th February 2014. Vol. 60 No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195



Address (offset)	Block size	Number of Calls	Flag
0x0040103E	19	1	0
0x00401051	9	2	1
0x0040105A	6	2	1
0X00401060	12	1	0
0X0040106C	8	2	1

Figure 2: Example of Constructing Indexed Table

Step 3: The above constructed index table is given as input to the Quasi group encryption technique.

B. Quasigroup Encryption for the Index Table

The encryption technique used in this approach is the quasi group encryption technique (Maruti Venkat Kartik Satti, 2007) shown in figure 3. The quasigroup encryptor has very good data-scrambling properties and thus, it has effectively used in symmetric cryptography. The purpose of the scrambler is to maximize the entropy at the output, even in cases where the input is constant. The great complexity connected with the task of identifying the scrambling transformation assures the effectiveness of the encryption process. Quasigroup encryption is a development that has permutation based scrambling (Kościenly, 2002) at its basis.

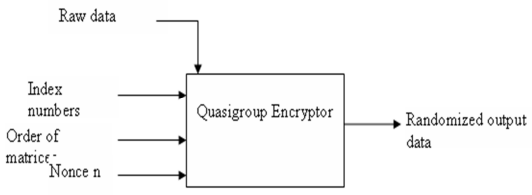


Figure 3: Quasi Group Encryptor

Input data*:* $d_1, d_2, d_3, \ldots d_n$
Output data: $e_1, e_2, e_3, \ldots e_n$

The two matrices*: R, S*
Multiplier Elements: $q_1, q_2, q_3, \ldots q_n$
The indices: $I_1, I_2, I_3, \ldots I_n$
The encryptor is defined by QE (stands for Quasi-Encryptor), and the decryptor is defined as QD (stands for Quasi-Decryptor).

Encryption: It should be that if Q is a quasigroup such that $a_1, a_2, a_3, \ldots a_n$ belong to it then the encryption operation QE, which is defined over the defined elements, maps those elements to another vector $b_1, b_2, b_3, \ldots b_n$ such that the elements of the resultant vector also belong to the same quasigroup. The mathematical equation used for encryption (basic level) is defined by:

$$E_a(a_1, a_2, a_3, \ldots, a_n) = b_1, b_2, b_3, \ldots b_n \quad (1)$$

where the output sequence is defined by:
$b_1 = a * a_1$
$b_i = b_{i-1} * a_i$
where i increments from 2 to the number of elements that have to be encrypted, and a is the *hidden key* (*leader* in Markovski and Dimitrova terminology (Dimitrova and Markovski, 2004). Equation (1) describes a typical single level quasigroup encryptor.

The workings of equation (1) are illustrated with the help of Figure 4. It is assumed that the initial input data given by the vector $a_1, a_2, a_3, a_4, a_5, a_6$. It is mapped to the vector $b_1, b_2, b_3, b_4, b_5, b_6$ by equation (4). The following steps are used during the process of encryption:

$$b_1 = a * a_1 = 2 * 2 = 1$$
$$b_2 = b_1 * a_2 = 1 * 4 = 1$$
$$b_3 = b_2 * a_3 = 4 * 1 = 4$$
$$b_4 = b_3 * a_4 = 4 * 2 = 5$$
$$b_5 = b_4 * a_5 = 5 * 3 = 1$$
$$b_6 = b_5 * a_6 = 1 * 3 = 2$$

Journal of Theoretical and Applied Information Technology
20th February 2014. Vol. 60 No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

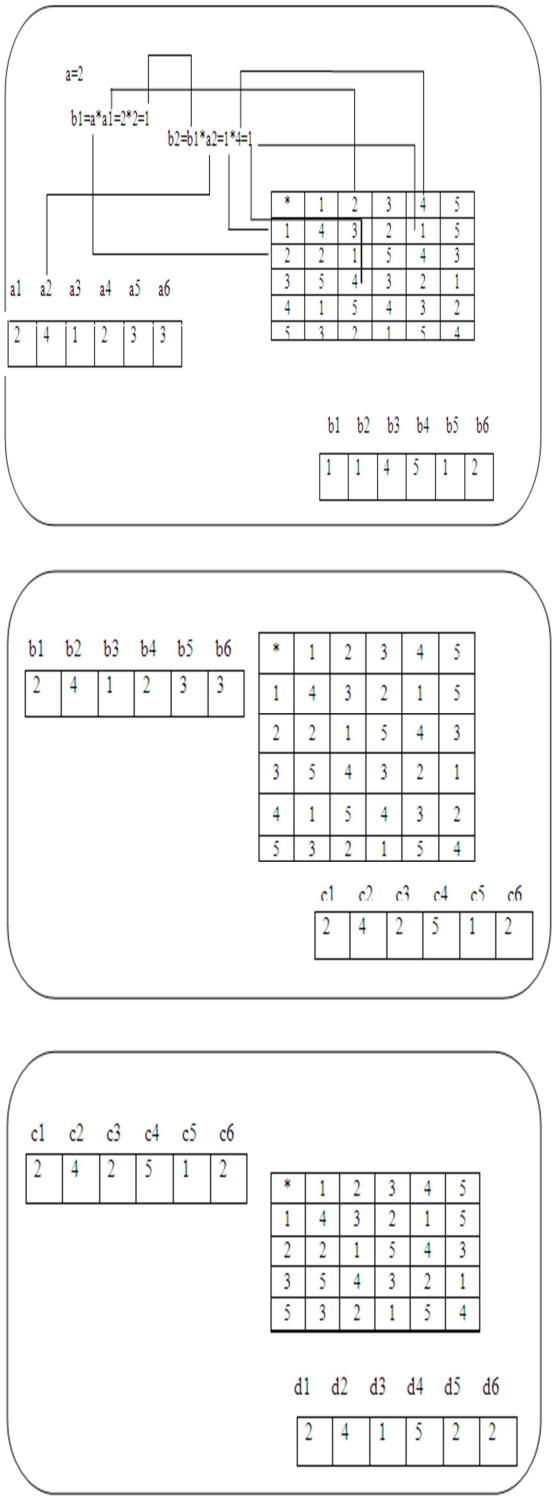


Figure 4: Quasigroup Mapping Using An Order 3 Quasigroup

The sequence attained is given as an input to another level of the encryptor. This process is repeated several times. Multiple levels of mapping assure lesser resemblance of the output data to that of the input data. This makes the decryption of the original data very tough.

In certain implementation, the multiplier element is varied. The multipliers are constructed by a special algorithm called "MEG1" that constructs the multiplier elements depending on the index numbers, Nonce, r and s given by the following equations (Dimitrova and Markovski, 2004):

$$E_{h1,h2,h3....hn}\ (a_1, a_2, a_3 \ldots a_n) = e_1, e_2, e_2, \ldots e_n \quad (2)$$

where

$$e_1 = a * a_1 \ and \ e_i = e_i - 1 * a_i$$

In the above equation, the incoming data is first mapped through the first multiplier element h_1 then the resultant data is mapped taking into account the second multiplier element h_2. This process continues till all the multiplier elements are exhausted.

$$b_1 = h_1 * a_1;\ b_2 = b_1 * a_2; \ldots b_n = b_n - 1 * a_n \quad (3)$$

$$c_1 = h_2 * b_1;\ c_2 = c_1 * b_2; \ldots c_n = c_n - 1 * b_n$$

$$\vdots$$

$$e1 = hn * s1;\ e2 = e1 * s_2; \ldots e_n = e_{n-1} * s_n$$

where the vector $(h_1, h_2, h_3, \ldots h_n)$comprises of all the multiplier elements. In this approach, this encryption key is transmitted along with the quasigroup (this key is itself summarized by another layer of encryption). It is to be observed that in the above two techniques another reliable encryption approach is necessary to preserve the secrecy of the encryption. Moreover, it is necessary to transmit the quasigroup that is being used for encryption, which is one of the main limitations of the above technique. If the eavesdropper breaks the encapsulating cipher, it is possible to get access to the quasigroup used for the encryption and all the other needed data to get the data.

This paper uses the index based approach where the given data is encrypted through a number of levels of encryption. The second level encryption, the input vector is mapped to the sequence which has symbols ranging from 1 to the order of the second matrix. Thus, if an index key is present which references the matrices stored in the memory of the reception device, the intruder would not know which matrix is stored at a given index. In order to further enhance the efficiency of this quasigroup encryptor, another function can be included that arranges the quasigroups according to the Nonce and this makes the encryption more time dependent and it can be observed that at any given point of time the output of the encryptor is different even if the same set of indices are given to the

Journal of Theoretical and Applied Information Technology
20th February 2014. Vol. 60 No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

technique. The Multi Level Indexed encryptor is denoted as

$$QE^{I_r,I_s}_{h_1,h_2,\ldots,h_n}(a_1,a_2,a_3\ldots,a_n) = e_1,e_2,e_3,\ldots,e_n \quad (4)$$

where $a_1,a_2,a_3\ldots,a_n$is the input data and $e_1,e_2,e_2,\ldots e_n$ is the output vector I_r and I_s are called indices that are arrays which have the indices of quasigroups having corresponding order. The vector $(h_1,h_2,\ldots.,h_n)$ is the Hidden key or the Secret key. It is the output of the MEG-1 algorithm.

Decryption : This process is highly alike the process of encryption which has just been discussed. The key point to be considered is the construction of the inverse matrix. The left inverse '\' is used for the quasigroup decryption as described in the Figure 5. The fundamental equation for encryption is:

$$D(a_1,a_2,a_3\ldots,a_n) = e_1,e_2,e_3,\ldots,e_n \quad (5)$$

where

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

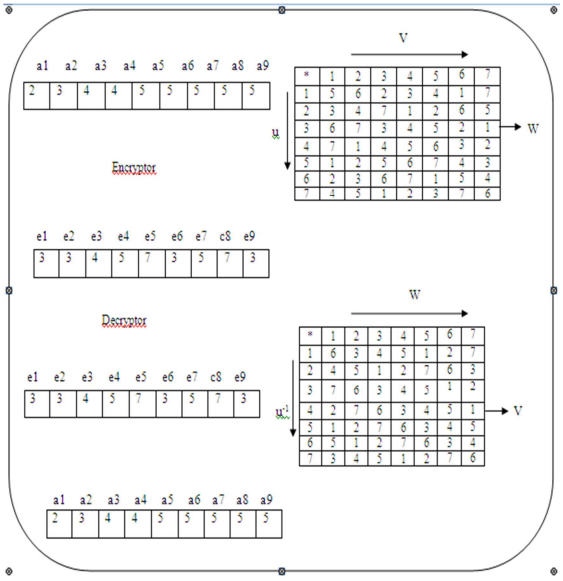


Figure 5: Determination of Left Division and the Complete Process of Encryption and Decryption

In order to carry out the process of decryption, the inverse matrix of a given quasigroup has to be constructed and execute mapping procedure as described in the previous section, equation (4) has to be used instead of (1). The decryptor for a multilevel indexed based algorithm may be defined as follows:

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

In Figure 4, the elements in 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*.

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} .

4. EXPERIMENTAL RESULTS AND DISCUSSION

This experimental result section mainly focuses on the security analysis and performance analysis of the proposed approach. The performance of the proposed approach is compared with standard software schemes.

A. *Security Analysis*

In order to enhance the security, the indexed table approach is adopted based on a quasi group encryption scheme. The performance of the proposed approach is compared with Cappaert's scheme and indexed table based self encryption code by (Sungkyu Cho et al. 2011).

The main focus of the software protection is to secure the original binary code from various attacks by remaining confidential. The proposed approach uses the quasi group encryption technique that has very significant data-scrambling properties and thus it has significant uses in symmetric cryptography. The main aim of the scrambler is to increase the entropy at the output, even if the input is constant. The enormous complexity connected with the task of identifying the scrambling transformation assures the effectiveness of the encryption process. Cappaert's scheme did not satisfy the correct key chain requirement.

B. *Experimental Set up and Result*

The implementation of the proposed approach is based on certain set up. The operating system used for the proposed approach is Windows XP and is implemented using Microsoft Visual Basic.Net. The cryptographic library and CPU used is Win32 OpenSSL version 0.9.8 and Intel Core2Duo CPU E7200 respectively. PEDasm version 0.33 is referred for this experiment which is an open source disassembler. In order to evaluate the performance, three small default executable files in Windows XP are chosen. A stream cipher, quasi group is used as a cryptographic algorithm to encrypt and decrypt the code. Initially, the executable file is entered, disassembled, and partitioned into basic blocks. Then, the program executes table indexing and code encryption through the partitioned basic block. The

Journal of Theoretical and Applied Information Technology
20th February 2014. Vol. 60 No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

performance of the proposed approach is compared with the (Sungkyu Cho et al. 2011) code encryption scheme. For instance, if a program *P* and its modified version *P'* is available. Then, the time cost C_t and the space cost C_s is defined as

$$C_t(P, P') = \frac{T(P')}{T(P)} \quad (7)$$

$$C_s(P, P') = \frac{S(P')}{S(P)} \quad (8)$$

where $T(X)$ is the execution time of program X, and $S(X)$ is its size. The Encryption process, indexed table generation process is implemented and shown in figure 6 whereas figure 7 displays the decryption process.

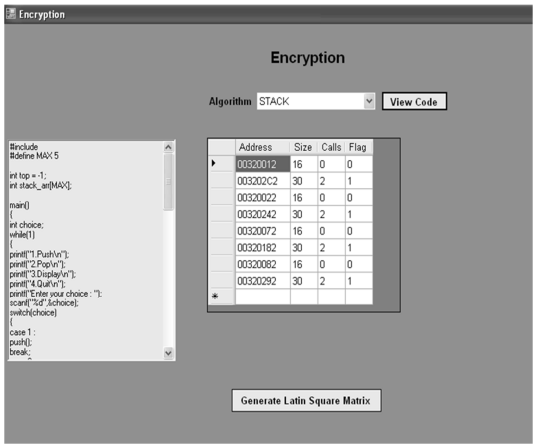


Figure 6: Encryption Process-Index table generation

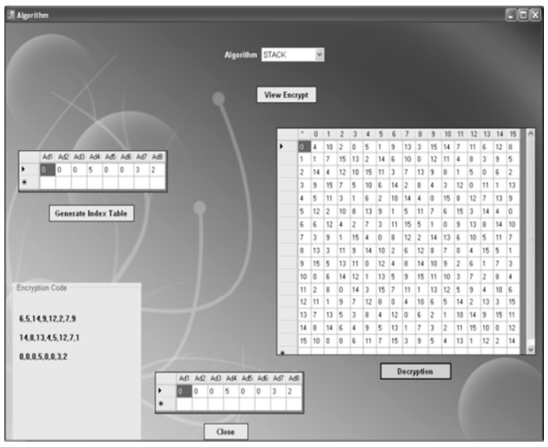


Figure 7: Decryption Process

The encryption time and decryption time of two programs are evaluated. At the moment, external libraries such as ".dll" files are eliminated as they are implemented externally to the executable file. The results are shown in Table 1 and figures 8, 9 and 10.

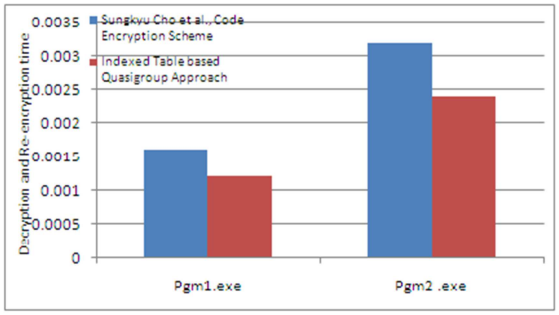


Figure 8: Comparison of Decryption and Re-Encryption Time

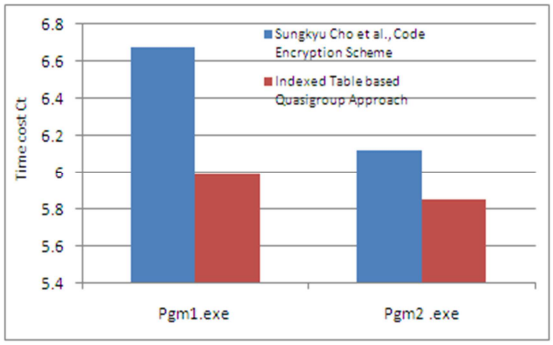


Figure 9: Comparison of Time Cost Ct

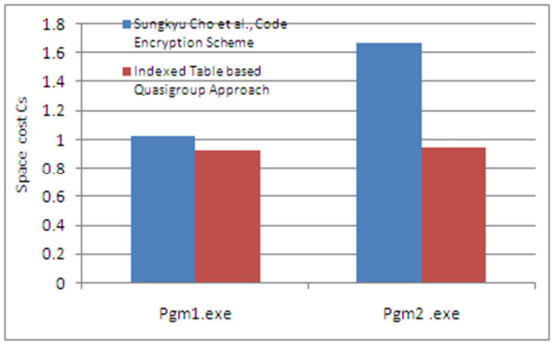


Figure 10: Comparison of Space Cost Cs

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Table 1:Results Comparison

Features	Sungkyu Cho et al., Code Encryption Scheme		Indexed Table based Quasigroup Approach	
	Pgm1.exe	Pgm2.exe	Pgm1.exe	Pgm2.exe
Original file size (B)	3584	4096	3584	3584
Number of blocks	12	26	12	26
Decryption and re-encryption time (s)	0.0016	0.0032	0.0012	0.0024
C_t	6.680	6.119	5.985	5.845
C_s	1.027	1.674	0.925	0.942

5. CONCLUSION AND FUTURE WORK

Confidentiality and data authenticity are mainly focused in assuring efficient security. Several techniques are available in the literature for providing security to the software. However, most of the schemes do not meet the security requirements for code encryption schemes, and also had efficiency problems. Recently, encryption has provided the means to hide information. This paper presented and discussed code encryption schemes for protecting software against various attacks like reverse engineering, tampering etc. A new code encryption approach based on an indexed table to guarantee secure key management and efficiency is proposed in this paper. Efficient Quasi group encryption technique is used in this paper. The performance of the proposed approach is evaluated based on the time cost and space cost. It is observed that the proposed approach shows good performance when compared with the Sungkyu Cho et al., Code Encryption Scheme. In future, result comparison can be extended for complex programs. Further the obtained results can be extended to several open source programs and can be compared with the existing scheme. Statistical based data scrambling techniques can be used to meet the security requirements for code encryption.

REFERENCES:

[1] Howard, M., LeBlanc, D.C., 2002. "Writing Secure Code", 2nd ed., Microsoft Press

[2] Viega, J, McGraw, G., 2001. "Building Secure Software", Addison Wesley

[3] Eilam, E., 2005. "Reversing: Secrets of Reverse Engineering", Wiley Publishing, Inc.,

[4] Hongxia Jin, Lotspiech, J., 2003. "Forensic analysis for tamper resistant software", 14th International Symposium on Software Reliability Engineering.

[5] Jan Memon, Asma Khan, Amber Baig and Asadullah Shah, 2007. "A Study of Software Protection Techniques," Innovations Adv. Techniques Computer Inf. Sci. Engin., pp. 249-253

[6] Stytz, M.R, Whittaker, J.A, 2003. "Software Protection: Security's Last Stand," IEEE Security Privacy, vol. 1, no. 1, pp. 95-98.

[7] Ogiso, T., Sakabe, U., Soshi, M., Miyaji, A., 2002. "Software Tamper Resistance Based on the Difficulty of Interprocedural Analysis", 3rd Workshop on Information Security Applications (WISA 2002), Korea.

[8] Kent, S., 1980. "Protecting Externally Supplied Software in Small Computers", Ph.D. thesis, M.I.T.,

[9] Denning, D., 1982. "Cryptography and Data Security", Addison Wesley.

[10] Nicol, D.M., Okhravi, H., 2005. "Performance analysis of binary code protection", Proceedings of the Winter Simulation Conference

[11] Wilander, J., Kamkar, M., 2003. "A Comparison of Publicly Available Tools for Dynamic Buffer Overflow Prevention", pp.149-162, Proc. of NDSS'03 (Internet Society): Network and Distributed System Security Symp.,

[12] Chang, H., Atallah M., 2002. "Protecting Software Code by Guards", pp.160–175, Proc. 1st ACM Workshop on DRM (DRM 2001), Springer-Verlag LNCS 2320

[13] Jan Cappaert, Nessim Kisserli, Dries Schellekens, and Bart Preneel, 2006.l "Self-encrypting Code to Protect Against Analysis and Tampering", 1st Benelux Workshop Inf. Syst. Security.

[14] Collberg, C.S., Thomborson, C., 2002. "Watermarking, tamper-proofing, and obfuscation - tools for software protection", IEEE Transactions on Software Engineering, Volume: 28 , Issue: 8, Page(s): 735 – 746,

[15] Jan Cappaert, Nessim Kisserli, Dries Schellekens, and Bart Preneel, 2008. "Toward Tamper Resistant Code Encryption: Practice and Experience," LNCS, vol. 4991, pp. 86-100.

[16] Jung, D.W., Kim, H.S., Park, J.G., 2008. "A Code Block Cipher Method to Protect Application Programs From Reverse Engineering,"J. Korea Inst. Inf. Security

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Cryptology, vol. 18, no. 2, pp. 85-96 (in Korean)

[17] Gutmann, P., 2000. "An Open-source Cryptographic Co-processor", Proc. 2000 USENIX Security Symposium.

[18] Sungkyu Cho, Donghwi Shin, Heasuk Jo, Donghyun Choi, Dongho Won, and Seungjoo Kim, 2011. "Secure and Efficient Code Encryption Scheme based on Indexed Table", ETRI Journal, Volume 33, Number 1.

[19] Maruti Venkat Kartik Satti, 2007. "Quasi Group based Crypto-System", A Thesis.

[20] Kościenly, C. 2002. Generating quasi groups for cryptographic applications. Int. J. Appl. Math. Comput. Sci., vol.12, No.4, 559–569.

[21] Dimitrova, V., Markovski J., 2004, On Quasigroup Sequence Random Generator. Proceedings of the 1st Balkan Conference in Informatics, Y. Manolopoulos and E. Spirakis, Eds., 21-23, Thessaloniki, Greece, pp. 393 – 401.

[22] Sasirekha, N., & Hemalatha, M. 2012. A Thorough Investigation on Software Protection Techniques against Various Attacks. Bonfring Int. J of Software Engineering and Soft Computing, 2(3), pp. 10-15.

[23] Hemalatha, M., & Sasirekha, N. 2012. A Survey on Software Protection Techniques against Various Attacks. Global Journal of Computer Science and Technology, 12(1).