International Journal of Electrical and Computer Engineering (IJECE)
Vol. 11, No. 1, February 2021, pp. 589~595
ISSN: 2088-8708, DOI: 10.11591/ijece.v11i1.pp589-595

Adaptive key generation algorithm based on software engineering methodology

Muayad Sadik Croock, Zahraa Abbas Hassan, Saja Dhyaa Khuder
Department of Computer Engineering, University of Technology, Iraq

Article Info

Article history:

Received Mar 23, 2020
Revised Jun 14, 2020
Accepted Jun 26, 2020

Keywords:

Key generation
Security
Self-checking process
SIGABA
Software engineering

ABSTRACT

Recently, the generation of security keys has been considered for guaranteeing the strongest of them in terms of randomness. In addition, the software engineering methodologies are adopted to ensure the mentioned goal is reached. In this paper, an adaptive key generation algorithm is proposed based on software engineering techniques. The adopted software engineering technique is self-checking process, used for detecting the fault in the underlying systems. This technique checks the generated security keys in terms of validity based on randomness factors. These factors include the results of National Institute of Standard Test (NIST) tests. In case the randomness factors are less than the accepted values, the key is regenerated until obtaining the valid one. It is important to note that the security keys are generated using shift register and SIGABA technique. The proposed algorithm is tested over different case studies and the results show the effective performance of it to produce well random generated keys.

Corresponding Author:

Muayad Sadik Croock,
Department of Compute Engineering,
University of Technology,
Al-sinaa Street, Baghdad, Iraq.
Email: Muayad.S.Croock@uotechnology.edu.iq, 120052@uotechnology.edu.iq,
120099@uotechnology.edu.iq

1. INTRODUCTION

In the last years, the searching for strong security keys that stand against the hacker key-breaking methods has been increased sharply. Thus, the introduced studies, methods and algorithms aimed to obtain these types of keys for using in security algorithms. This is to produce a security level sufficiently acts against the attacks and hackers. Different approaches have been adopted to generate the security keys with high variation in long, sub-sequence and correlations [1, 2]. At the other hand, software engineering methods have been combined with these approaches to increase the reliability of the generated security keys to resist the dominated attacks over numerous applications. One of these methods is the fault tolerance technique that is based on self-checking process to detect the happened faults [3-8]. The solution for the detected fault can be addressed throughout different ways depending on the utilized application.

Due to the importance of the security key management in the security research, many researchers have focused on proposing efficient algorithms. In [9], the authors provided a study on using software engineering for evaluating the security risks including identification system in humans using some image features that need software engineering for producing the reliable codes. In [10], a complete study for hardware security was introduced. This study included key points of using the techniques of software engineering in the key generation and management. Researchers of [11] considered the security of cloud computing in terms of authentication levels. The introduced method uses the software engineering in

managing the keys used in authentication, in which the security was guaranteed. In [12], secure keys were produced for cryptography methods to be used in network communication. This field had an important data to be transferred and this was the reason behind providing it. In [13], the cyber-physical systems had been secured using the investigated method based on software engineering technology. The presented system worked at the physical layer to adapt the hardware components within the system. In [14], different practical software engineering developments for security systems were introduced. The study provided a deep-thinking way to tackle the problems of security in developing the used software.

The authors of [15] introduced a dynamic mechanism for generating the required security keys. This mechanism was based on genetic algorithm in addition to the use of software engineering technology. Moreover, multiple key generation method was proposed in [16]. These keys were employed in combination with the actions of sharing and backup for data. In [17], ciphering method of DES was combined with generated keys for securing the storage of data in cloud. The keys were generated based on a proposed algorithm that depended on software engineering. The authors of [18] presented design and implementation of key generation used for image security. The image security utilized the generated keys in coding of included components. In [19], memory efficient based multi key generation method was proposed. This method has been employed for securing the sensitive data transfer over IoT technology in the cloud environment.

At the other side, the authors of [20] utilized the OFDM subcarrier response systems for transferring important data in secure way. The used security keys were generated in high randomness to ensure the strong of them. In [21], the key generation method for multilevel quantization was provided. In addition, the pre-processing was also secured using the same method. These keys were generated using software engineering techniques. The researchers of [22] used the genetic algorithm in designing the key generation method that could guarantee high randomness. The enhanced random keys were produced for symmetric ones.

It is well known that the researchers of [9-22] worked hard for producing high quality security methods based on random generated keys. In this paper, an adaptive random key generation algorithm is proposed. The proposed algorithm adopts the self-checking process in fault detection as a part of fault tolerance technique to produce high randomness keys. The NIST tests are considered as a success threshold for self-checking process. The generated keys should pass the allocated thresholds to be adopted as valid keys. The proposed algorithm is evaluated in terms of performance and the obtained results prove the claim of this paper.

2. PROPOSD SOFTWARE ENGINEERING BASED METHOD

As mentioned above, the proposed method is based on software engineering process to produce high quality random security keys. In order to ease the reading flow, this section is divided into the followings.

2.1. Structure of the proposed algorithm

Figure 1 shows the structure of the key generation algorithm. It is clearly explained that the data block firstly enters to the SIGABA approach. This approach is built using five main polynomial equations to increase the randomness in choosing the seeds for shift register. SIGABA was produced in World War 2 to encrypt the transmitted messages as shown in Figure 2 [23-24]. The selected polynomial by the SIGABA approach with 32-bit is passed to the least forwarded shift register (LFSR). The LFSR is responsible on generating the key with 128-bit length. Figure 3 shows the structure of the adopted LFSR.

Figure 1. Key generation algorithm structure

Figure 2. SIGABA device

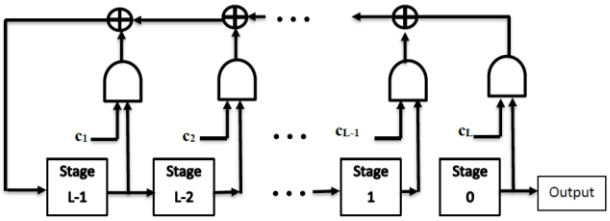

Figure 3. LFSR structure

2.2. Proposed algorithm

Figure 4 (shown in Appendix) explains the adaptive key generation algorithm of that adopts SIGARA method [25] and the software engineering technique in verifying the validity of the generated keys. The adopted steps of this algorithm can be summarized as follows:

- For each loop of the proposed algorithm, it chooses five polynomial equations that are sorted.
- According to SIGABA approach, the proposed algorithm chooses equation 5.
- Its exams the least significant bit (LSB) if 0 or 1.
- In case the LSB is 1, the algorithm selects path 2 of Figure 4(b), else it selects path 1 of Figure 4(b).
- Path 1 chooses equation 3, while path 2 chooses equation 4.
- Path 1 exams the LSB of equation 3, and path 2 tests LS of equation 4.
- In both paths, if the LSB of equation (3 and 4) is 1, it selects equation 2 and if LSB is 1 the algorithm selects equation 1. The same procedure is performed for path 2.
- The results of path 1 is XORing with equation 4, while the results of path 2 is XORing with equation 3.
- The results of XOR is considered as a seed equation for LFSR, as shown in Figure 4(c).
- The LFSR receives the seed equation and applies the adopted procedure to produce 128-bit key.
- The obtained key is tested using the self-checking procedure. This is done by checking the validity of the key in terms of randomness using NIST (frequency and serial) tests. The results are compared with the acceptable thresholds. In case the results are out of the expected values, the key is rejected and the loop returns to the first step of the proposed algorithm to generate new key. Otherwise, the resulting key is valid.

3. EXPERMENTAL RESULTS

In order to test the proposed algorithm of generating reliable security keys, different training generations for keys are performed. Figure 5 shows the run screen of the proposed algorithm that explain the procedure mentioned in section 2. It is noted that there are five polynomial equations that create five sequences of binary 32-bit. After applying the proposed algorithm, the testing results of NIST frequency and serial are appeared to accept the generated 128-bit keys as they pass the acceptable threshold ranges. Therefore, the generated key is valid.

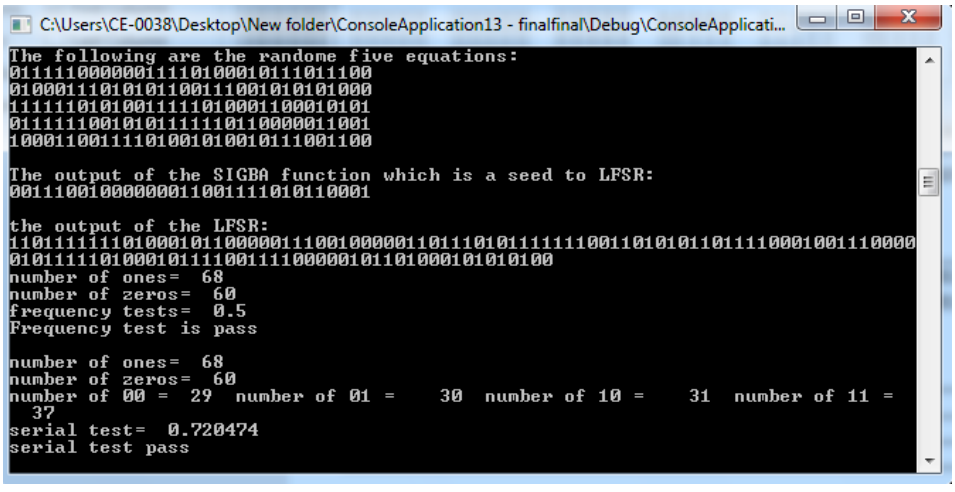

Figure 5. Valid generated key case study

At the other hand, Figure 6 illustrates the second case study, where the NIST frequency test is passed, while the serial test is failed. In this case, the generated key is considered as invalid and rejected. According to the proposed algorithm, the key is regenerated from again following the same steps. The regenerated key is passed both NIST tests as shown in Figure 7. In this case, the generated key is valid and ready to be used in the security methods.

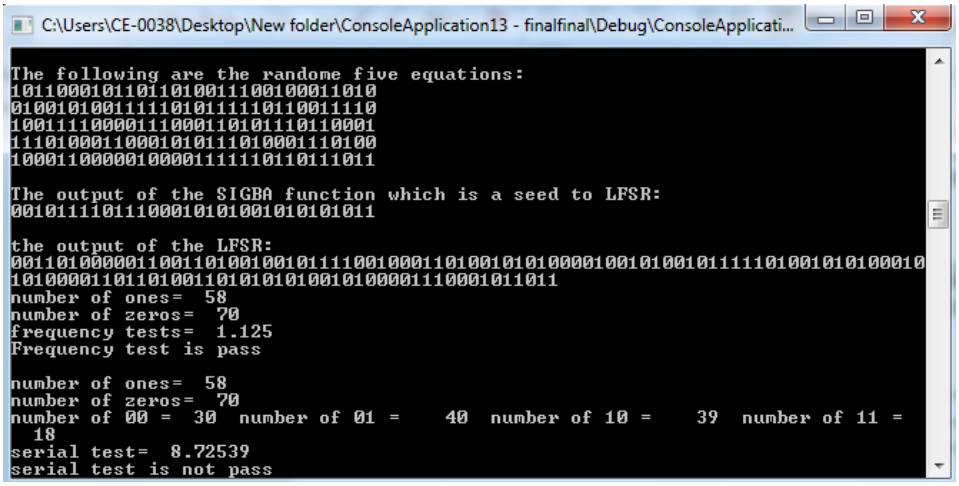

Figure 6. Invalid generated key case study

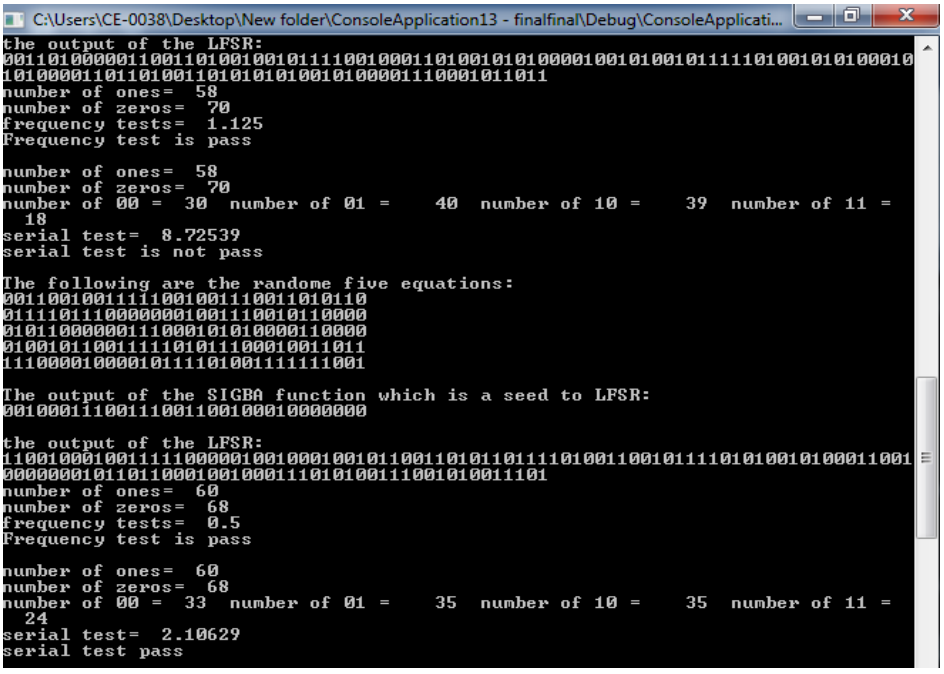

Figure 7. Valid regenerated key case study

For more testing results, five examinations are performed for the proposed algorithm. The results of these examinations are shown Table 1. Table 2 explains the result analysis of all case studies including both NIST tests and the common couples of bits. All case studies are passed and produce valid keys, yet the case study number 5 is failed in the serial test. Thus, the proposed algorithm uses the software engineering based fault tolerance to regenerate the failed key. The regenerated key is also tested over the same NIST tests. The obtained regenerated key is shown in Table 3, while Table 4 lists the analysis of the new key and it well shown that the key is passed both tests.

Table 1. Generated keys for different case studies

No	5 Polynomials as input to SIGBA (8 Bits)	32-bit output of SIGBA (8 Bits)	Generated keys (128 Bits)
1	01100100100101011111110000010011 00100111111001011101101101110101 01101001000000110110001010000000 01110100011110001101010001111100 10100100000101111111111100010001	00001101100101101001111010010011	00011001011011100110001001001111 01111110110100110111011110010111 11100000101100011011011000100111 01111100001000011010011100011111 1001
2	10110000110100101000010011101010 10111111011000100110110101101110 00100010011010100000110110000000 10111101110001100111100100011100 11011110010110110010110100110010	00001101000101001111110111110110	01011000110101000001001000110110 00010001110110001011000000100111 11110101100110111111011011010000 10101101011111110000100011010011011
3	01011000111110101001001001111011 10110000101100011010010111000100 01111011001001011011011000000110 00011001110001010000001111011101 10011010100010010001100100001101	11001011100101000001001111000010	10011100010101111110101100010111 11111100111000010101110010100010 00000000010111000010110010111010 00001110110010110001111000000000 0010
4	01111100000011110100010111011100 01000111010101100111001010101000 11111101010011111010001100010101 01111110010101111110110000011001 10001100111101001010010111001100	00111001000000011001111010110000	11011111110100010110000011100100 00001101110101111111001101010110 01111000100111000001011111010000 10111100111100000101101000101010 0100
5	**10110001011011010011100100011010 01001010011111010111110110011110 10011110000111000110101110110001 11101000110001010111010001110100 10001100000100001111110110111011**	**00101111011100010101001010101011**	**00110100000110011010010010111100 01000110100101010000100101001010 11110100101010001010100001101100 10011010101010010100001110001011011**

Table 2. Results analysis for 5 case studies

No.	No. of zeros	No. of ones	No. of 00	No. of 01	No. of 10	No. of 11	Frequency Test	Serial Test
1	59	69	29	30	29	39	0.781=pass	1.447=pass
2	63	65	30	33	32	32	0.0312=pass	0.118=Pass
3	68	60	39	28	29	31	0.5=pass	1.854=pass
4	60	68	29	30	31	37	0.5=pass	0.720=Pass
5	**70**	**58**	**30**	**40**	**39**	**18**	**1.125=pass**	**8.725=Not pass**

Table 3. Regenerated key

No	5 Polynomials as input to SIGBA (8 Bits)	32-bit output of SIGBA (8 Bits)	Generated keys (128 Bits)
5	00110010011111001001110011010110 01111011100000001001110010110000 01011000000111000101010000110000 01001011001111101011100010011011 11100001000010111101001111111001	00100011100111001100100010000000	11001000100111110000010010001001 01100111010111110100110010111010 10010100011001000000001011011000 10 01000111010100111001010011101

Table 4. Regenerated key analysis

No.	No. of zeros	No. of ones	No. of 00	No. of 01	No. of 10	No. of 11	Frequency Test	Serial Test
5	68	60	33	35	35	24	0.5 pass	2.10629 pass

4. CONCLUSION

An adaptive key generation algorithm based on software engineering process was proposed. This algorithm aimed to generate security keys of 128-bit with high randomness to ensure the accepted resilience against the attacks. The software engineering fault tolerance technique was used for checking the generated keys in terms of randomness based on NIST tests. These tests were employed as accepted threshold to be decided if the generated keys were valid or not. The proposed algorithm was tested over different case studies and the obtained results proved the validity of this algorithm in terms of generating reliable and random keys.

APPENDIX

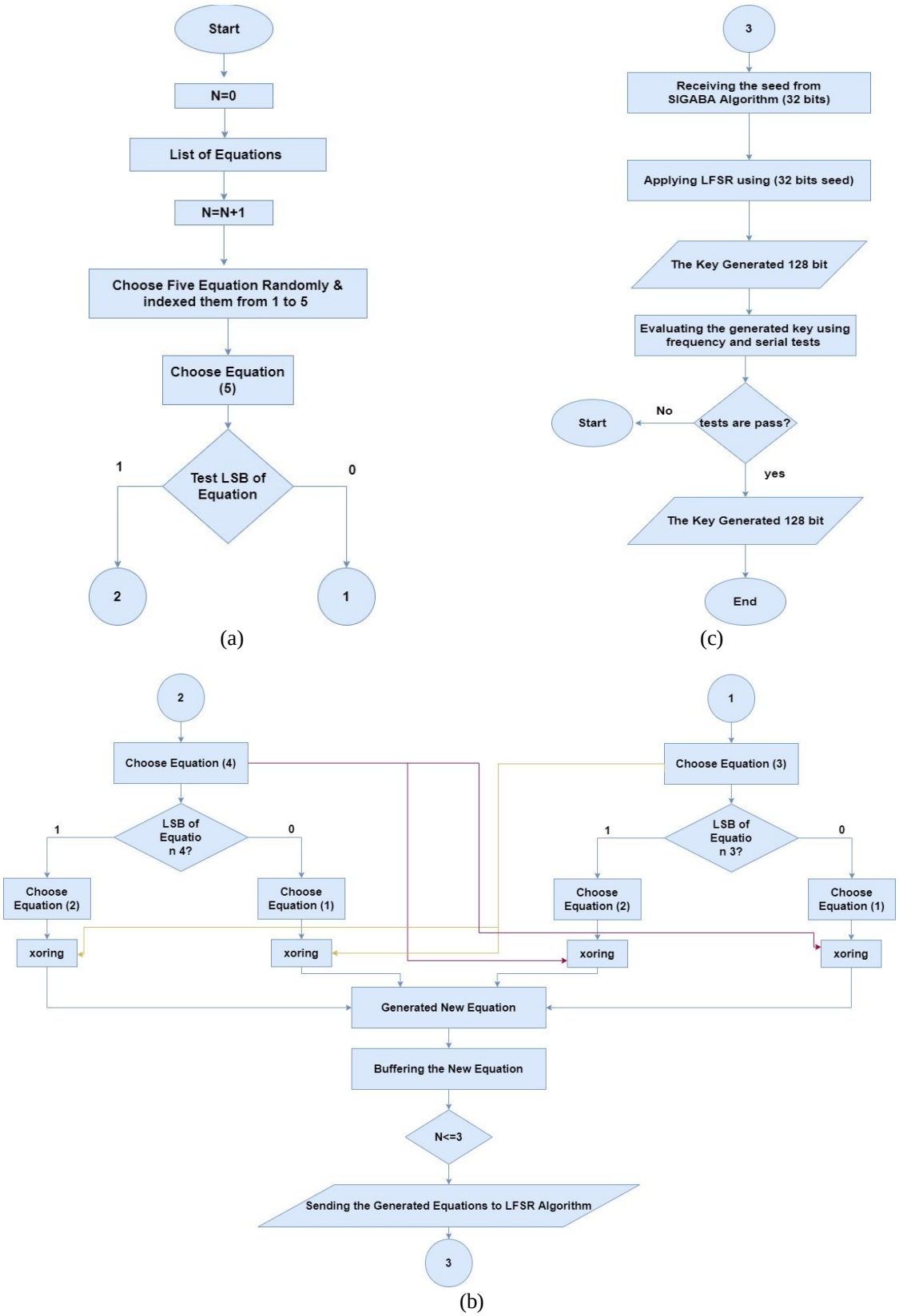

Figure 4. Proposed algorithm

REFERENCES

[1] E. Barker and A. Roginsky, "Recommendation for Cryptographic Key Generation," *NIST Publication*, 2019.
[2] B. Buchanan, "The Hacker and the State: Cyber Attacks and the New Normal of Geopolitics," *Politics of Privacy & Surveillance*, 2020.
[3] I. Sommerville, "Software engineering," *10th Edition, Pearson Education, Inc*, 2017.
[4] R. J. Ellison, et al., "Evaluating and Mitigating Software Supply Chain Security Risks," *Research, Technology, and System Solutions (RTSS) and CERT Programs*, 2010.
[5] M. Kuutilaa, et al., "Time Pressure in Software Engineering: A Systematic Review," *Elsiver*, 2020.
[6] L. Mottola, et al., "Simplifying the Integration of Wireless Sensor Networks into Business Processes," *IEEE Transaction on Software Engineering*, vol. 45, no. 6, pp. 576-596, 2019.
[7] Ayuba J., et al., "Software Development of Integrated Wireless Sensor Networks for RealTime Monitoring of Oil and Gas Flow Rate Metering Infrastructure," *Journal of Information Technology & Software Engineering*, vol. 8, no. 2, pp. 1-10, 2018.
[8] I. Almomani and A. Alromi, "Integrating Software Engineering Processes in the Development of Efficient Intrusion Detection Systems in Wireless Sensor Networks," *Journal of Semsors*, vol. 20, no. 5, pp. 1-28, 2020.
[9] S. D. Khudhur and M. S. Croock, "Dental X-Ray Based Human Identification System for Forensic," *Engineering and Technology Journal*, vol. 35, no. 1, pp. 49-60, 2017.
[10] Y. Jin, "Introduction to Hardware Security," *Journal of Electronics*, vol. 4, pp. 763-784, 2015.
[11] S. Dey and S. D. Joshi, "ECC Based Authentication System for Performance Improvement in Security of Cloud," *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 6, no. 3, pp. 820-823, 2016.
[12] Suresh H. and R. S. Hegadi, "DCA-SNC: Dual Cryptosystem Architecture for Secure Network Communication," *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 7, no. 1, pp. 198-205, 2017.
[13] S. Ur Rehman and V. Gruhn, "An Effective Security Requirements Engineering Framework for Cyber-Physical Systems," *Technologies*, vol. 6, no. 65, pp. 1-20, 2018.
[14] Safe Code, "Fundamental Practices for Secure Software," *Third Edition, SAFECode*, 2018.
[15] C. Chunka, et al., "An efficient mechanism to generate dynamic keys based on genetic algorithm," *Special Issue Article, Wiley*, pp. 1-10, 2018.
[16] R. Hariharan, et al., "Multiple key Generation for Securing Data Sharing and Backup," *International Journal of Engineering and Advanced Technology*, vol. 8, no. 6, pp. 166-169, 2019.
[17] M. Tajammul and R. Parveen, "Key Generation Algorithm Coupled with DES for Securing Cloud Storage," *International Journal of Engineering and Advanced Technology*, vol. 8, no. 5, pp. 1452-1458, 2019.
[18] A. Sharma, et al., "Design and Implementation of key Generation Algorithm for Secure Image," *International Journal of Engineering and Advanced Technology*, vol. 8, no. 2, pp. 5139-5146, 2019.
[19] C. Thirumalai and H. Kar, "Memory Efficient Multi Key (MEMK) Generation Scheme for Secure Transportation of Sensitive Data over Cloud and IoT Devices," *International Conference on Innovations in Power and Advanced Computing Technologies*, pp. 1-6, 2017.
[20] J. Zhang, et al., "Efficient Key Generation by Exploiting Randomness from Channel Responses of Individual OFDM Subcarriers," *IEEE Transactions on Communications*, vol. 64, pp. 2578-2588, 2016.
[21] M. Yuliana, et al., "A Simple Secret Key Generation by Using a Combination of Pre-Processing Method with a Multilevel Quantization," *Journal of Entropy*, vol. 21, no. 192, pp. 1-25, 2019.
[22] A. Z. Zakaria, et al., "Enhancing the Randomness of Symmetric Key using Genetic Algorithm," *International Journal of Innovative Technology and Exploring Engineering*, vol. 8, no. 8S, pp. 327-330, 2019.
[23] Level 1 Validation, "Sigaba Gateway FIPS 140-1 Non-Proprietary Security Policy," *Multi-chip standalone*, 2001.
[24] H. E. Kwong, "Cryptanalysis of the Sigaba Cipher," San José State University, 2008.
[25] M. Stamp and W. O. Chan, "SIGABA: Cryptanalysis of the Full Keyspace," San Jose State University, 2007.