A Proposed Approach for Unique Random Key Generation

Dalal N. Hamood *dalal.naeem@nahrainuniv.edu.iq*
Computer Department/College of Science, *Dal_scin81@yahoo.com*
University AL-Nahrain,
Baghdad, 10001, Iraq

Abdulrahman Q. Hammod *abdulrahman.qutaiba@proton.me*
Computer Department/College of Science, *abdulrahman.business@proton.me*
University AL-Nahrain,
Baghdad, 10001, Iraq

Abstract

Regarding network security, many cryptographic techniques use random numbers. To boost its robustness, a precise quantity of randomness must be used to make encryption and decryption unexpected. This paper offered a mechanism that chooses a distinctive number based on data instead of a temporal seed, which is what the bulk of applications now do. The suggested method is the basis for Unique Random Generation and uses the parallel technique. This approach has been tried, and the findings demonstrate that it is easy to use and yields the best results because it chooses a distinct number based purely on data. The random key is better than regular keys since it is dynamic whereas regular keys are static. After testing the proposed work, concluded when using seeds with the appropriate levels of entropy, this method can generate sequences whose randomness cannot be distinguished from that of an ideal random generator, with a confidence level of 99%. Additionally, the proposed method used base=2 to produce the highest entropy, the lowest space complexity, and the highest time complexity than methods based time seed.the proposed method success in all NIST tests (high randomness) and has a short time in generation (faster method). This method appropriates for high security applications.

Keywords: Random Number Generator, Randomness, Binary Tree List, Ranges, Time Generation.

1. INTRODUCTION

The study of utilizing mathematics to encrypt and decrypt data is known as cryptography. With the help of cryptography, you may store and send private data in a way that only the intended receiver can read it [1] over insecure networks like the Internet. While cryptanalysis is the discipline of deciphering secure communication, cryptography is the science of protecting data. Traditional cryptanalysis is an intriguing blend of mathematical application, pattern recognition, analytical thinking, perseverance, and good fortune [2]. Cryptanalysts are also called attackers. Cryptology embraces both cryptography and cryptanalysis. Cryptography can be strong or weak [3][4]. Cryptographic strength is measured in the time and resources it would require to recover the plaintext. The result of strong cryptography is ciphertext that is very difficult to decipher without possession of the appropriate decoding tool. With given all of today's computing power and available time—even a billion computers doing a billion checks a second—it is not possible to decipher the result of strong cryptography before the end of the universe [5][6]. A key is a value that works with a cryptographic algorithm to produce a specific ciphertext. When we have a strong key we have a strong ciphertext and more difficult to suggest the plaintext [7][8].Larger and random keys will be cryptographically secure for a longer period of time. If what you want to encrypt needs to be hidden for many years, you might want to use a very large and random key [9]. Due to keys importance in cryptography, Random Number Generators (RNGs) are an essential component of key generation in the cryptography. An RNG, which generates unpredictable and fast random numbers is essential for better data confidentiality and integrity protection. The next applications of cryptography used random number generators [10] [11]:

- PIN code and different password generation
- Private keys for digital signature algorithms
- Values that are utilized in protocols like key establishment

It's crucial in cryptography to ensure that secret keys are random and unpredictable, or, to put it another way, that they might follow the randomness' rules [12]. In this paper, suggested a method to generate unique random numbers (small, large) without depending on time seed because the random number generated depending on time is vulnerable when an attacker can analyze and watch victim actions. A full article usually follows a standard structure: 1. Introduction, 2. Types of Random Number Generators, 3. Proposed Method, 4. Results and Discussion, and 5. Conclusion.

2. LITERATURE REVIEW

Literature review that has been various advancements in RNG with encryption were made in recent years; a few of these are briefly detailed below. The authors of [13] developed a Deterministic Random Bit Generator (DRBG) which satisfies the security standards for cryptographic applications as a Cryptographically Secure Pseudo-Random Number Generator (CSPRNG), along with an entropy source that demonstrated a high level of entropy and high portability. The suggested design has undergone extensive testing to determine its randomness and entropy using the BSI and NIST suites, and it is now prepared to be incorporated into the European Processor Initiative (EPI) chip. The authors at [14] presented work discovered the drawbacks of linear-feedback shifter register (LFSR), i.e., predictable and periodic random sequences, and have suggested a polynomial modulator to prevent the predictability. According to simulation results, the dynamic polynomial modifications in the suggested design allow for the generation of random numbers which are over 4000 times larger before they recur and become unpredictable. The authors in [15] suggested the approach of producing real random numbers using a circuit primarily intended as PUF depending on ring oscillators. The objective is to demonstrate that it is possible to create a universal crypto system which could be utilized for a variety of purposes. For example, the PUF might be utilized for asymmetric cryptography and producing asymmetric keys, while the TRNG could be utilized for symmetric cryptography and producing session and ephemeral keys, salts, and nonces. The evaluation regarding a circuit used for TRNG purposes is evaluated, and the results are reported in the study. The author of [16] uses a quantum RNG that is based on a balanced homodyne measurement of vacuum electromagnetic field fluctuations. With a quick randomness extraction method depending on a linear feedback shift register, the digitized signal is instantly processed. The random bit stream passes an extensive test suite for random numbers and is continually read in a computer at a rate of roughly 480 Mbit/s. Through observing the vacuum variations regarding the electromagnetic field, we developed a RNG method. This study produced uniformly distributed random numbers quickly from a fundamentally unpredictable quantum measurement through calculating the amount of useable entropy of quantum noise and employing an effective randomness extractor depending on linear feedback shift registers. In [17] according to the space of all UUIDs, a UUID can be defined as an identifier that is distinct over both time and space. Values may rollover (around A.D. 3400, based on the specific algorithm used) in UUID since it has a time field and is fixed in size. A UUID could be utilized for a variety of tasks, including correctly detecting extremely persistent objects over a network and labeling objects with an extremely limited lifetime. The internal representation regarding a UUID is a particular pattern of bits stored in memory. It is important to convert the bit representation into a string form in order to appropriately represent a UUID as a URN.In [18] proposed the SDSM system is based on a set of guidelines for a very secure system, free of implementation dependencies. The elimination of single instance of encryption key by adopting a dynamic approach to determine encryption key, prevents the attacker from getting an access to the encryption key. The algorithm presented in the paper demonstrates the concept of dynamic generation of encryption key, as well as key rotation across files. This combination makes the system extremely secure. In [19] introduces a brand-new discrete-time chaotic map-based Bernoulli random number generator. The discrete-time chaotic map and the dual oscillator architecture are used in the construction of the presented RNG to increase throughput and improve the statistical quality of the output sequence. A mathematical model of the proposed design had been developed, and it had been algebraically proven that the resulting bit stream had passed the FIPS-140-2 test suite's four main tests. The bit stream, which was produced in a similar manner from the hardware realization of the circuit, has likewise been verified to have passed all NIST-800-22 tests without requiring any post-processing. The experiment and simulation findings, which support the circuit's effectiveness, are provided. We can use the integrated circuit to create the recommended RNG. In [20] suggest Rando, a multipurpose genuine random number generator for use with normal computing hardware. Rando does not consume additional spaces and has an O(m) time complexity, where m is the bit size of a random number. Our suggested algorithm is a straightforward true random number generator that is both easy to use and effective. It is based on hashing algorithms and system clocks. Our test results demonstrate that Rando performs better in randomness than cutting-edge methods. NIST SP 800-22 validates Rando's real randomness and finds that it passes all 15 statistical tests.

3. TYPES OF RANDOM NUMBER GENERATORS

Pseudorandom number generators (PRNGs) and True random number generators (TRNGs) are the two categories into which RNGs fall [21].

A. True Random Numbers Generators: A TRNG doesn't create its own seed values; instead, it took entropy sources already present in the environment. The physical surroundings of the computer could provide entropy sources like timing samples of keystrokes, mouse movement, electrical activity on disks, current values of the system clock, and many more [22]. The method that generates random binary output is given either a single source or a combination of several sources as input. Examples include the timing of keystrokes and mouse movements [23], as seen in Fig. 1.

B.

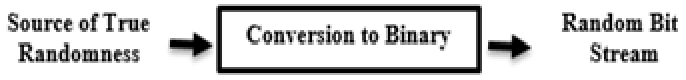


FIGURE 1: True Random Number Generator [28].

C. Pseudorandom Numbers Generators: PRNGs are RNGs that don't base their sequence of events in the real world. The features regarding the number sequences produced through PRNGs are comparable to those of random numbers [24]. A random starting state is used when utilizing a seed state PRNG. If the starting point within the given set is known, several numbers could be created later on in a short amount of time. As a result, the numbers produced are accurate and predictable [25]. A PRNG must locate the entropy in order to maintain its unpredictable nature; TRNGs convert entropy sources directly into sequences [26]. We can obtain the entropy needed for the PRNGs by using the time of day, the mouse's position or location, or the activity on the keyboard. As there is a chance that an attacker may intentionally change the system to bias it. As a result, we exposed this approach in a secure setting [27], as illustrated in Fig. 2.

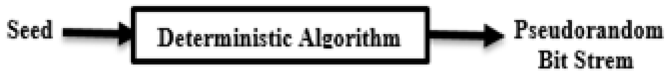


FIGURE 2: Pseudorandom Number Generator [28].

4. THE PROPOSED METHOD

In this paper, suggested a new method to generate unique random numbers (small, large) without depending on time base because the random number generated depending on time is vulnerable when an attacker can analyze.

4.1 Data Structure

The abstract data structure used in the proposed method is [Node List] with sequential numbers start {0...max} and later another class can derive it to reach nodes in a better way (such as BinaryTreeList) where each node is fixed array of with size = 2 cells. Each node has (start, size, values), where the start is the position of the first number (zero by default) represents the beginning of the node. And size is the number of allocated items with values where (size <= 2), as shown in Fig. 3.

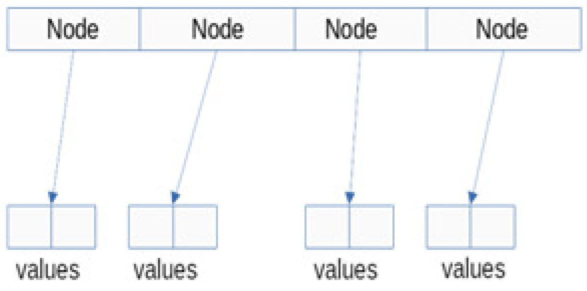


FIGURE 3: Data Structure Used An Array Of Nodes.

4.2 Delete Operation

In the proposed method, Deletion operation causes a change of the index values of upcoming nodes in case we avoid calculating unallocated items. Pop operation retrieves the relatively positioned item and delete (unallocated) it; This is the first step and the base idea behind unique choosing such that choosing value is not repeated and we achieve it by deallocating the specific item.

4.3 Node Sequential (Derives Node List)

To get better access to nodes, we have to group them in the node list in addition to get a bigger index range instead of 0 and 1 in case of a single node. The node list receives cardinality as part of its definition to make the number of items are 2^cardinality. The reason for using cardinality is to represent the raw positive integer numbers as an array; This makes them easy to access, modify, delete, insert.

For example: Let the cardinality = 4 then 2^4 = 16 then array [4, 4] is equivalent to number 0100 0100 which is 0x44 in hexadecimal or 68 in decimal.

4.3.1 Remove items based on next index

Let be in the interface a method called next () that retrieve an integer number each time we invoke it.

If the retrieved index of the key is >=NNodeList.Size () then the index becomes [index mod NodeList. Size()] and This achieves abstract approach of Unique Choosing.

For proposed method, the key type that implements the proposed interface uses secret sequence and this achieves approach of Unique Random Generation. And we call it Random because it is unknown how the key will produce the next it.

Let’s consider a key returns the following sequences [0, 0, 0, 0, 1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3] and another method uses the output of next () method as remove index for proposed node sequence and print removed number.

Cardinality = 4 so the node sequence is set {0.....15}

Output:
[0, 1, 2, 3, 5, 6, 7, 8, 10, 11, 12, 13, 15, 4, 9, 14]

NODE LIST	NOTES
[Node<0,1>, Node<2,3>, Node<4,5>, Node<6,7>, Node<8,9>, Node<10,11>, Node<12,13>, Node<14,15>,]	While creating an object of type (NodeSequence), the node list will be
[Node<0,1>, Node<2,3>, Node<4,5>, Node<6,7>, Node<8,9>, Node<10,11>, Node<12,13>, Node<14,15>,]	Because remove index is 0 at the first 4 next key indices the first 4 items are removed and retrieved.
[Node<0,1>, Node<2,3>, Node<4,5>, Node<6,7>, Node<8,9>, Node<10,11>, Node<12,13>, Node<14,15>,]	The next 4 key indices are 1 and since index=0 points to 4 so it retrieves the next 4 numbers after 4 which are [5, 6, 7, 8].
[Node<0,1>, Node<2,3>, Node<4,5>, Node<6,7>, Node<8,9>, Node<10,11>, Node<12,13>, Node<14,15>,]	The next 4 key indices are 2 and since index=0 points to 4 and index=1 points to 9 so it retrieves the next 4 numbers after 9 which are [10, 11, 12, 13]. And at last, the next 4 indices are 3 and since index=0 points to 4 and index=1 points to 9 and index=2 points to 14 then it retrieves 15 then next 3 are List [3 mod 3], List [3 mod 2], List [3 mod 1] which are the same as [List[0], List[1], List[0]] (where [3, 2, 1] at the right side of the mod are sizes) so it returns [4, 14, 9]

TABLE 1: An explanation for what is going on.

Note: reset() function within NodeList will reallocate all previous items.

4.4 Binary Tree List (Derives Node List)

For more optimal approach, split array to sub-ranges and each range index represent the division result of the numbers that within the range over a specific value called range-division. Each two ranges are then grouped with parent range and each two parents are grouped and so on till reaching root range represent the allocated area from beginning to the end of the Node List, as shown in Fig. 4.

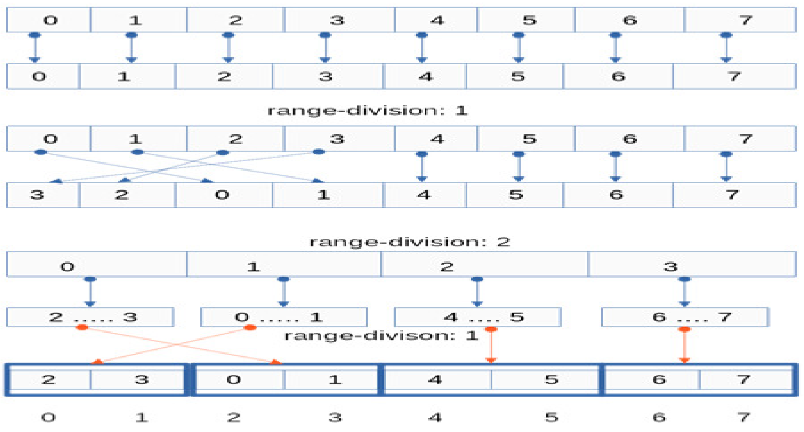


FIGURE 4: Splitting Array to Sub-Ranges.

Let range-division be count of values that can be sorted ascending/descending for easy implementation such as adding or removing value, for example:

Node List	Range-Division	Sub-Ranges
[1, 2, 3, 6, 7, 8, 23, 24, 25]	3	[1, 2, 3], [6, 7, 8], [23, 24, 25]

As an abstract proposed approach a parent sub-range has two children and each child has (start, end) where the end might be included or discarded according to the type of application, as shown by the following algorithm, also see Fig. 5.

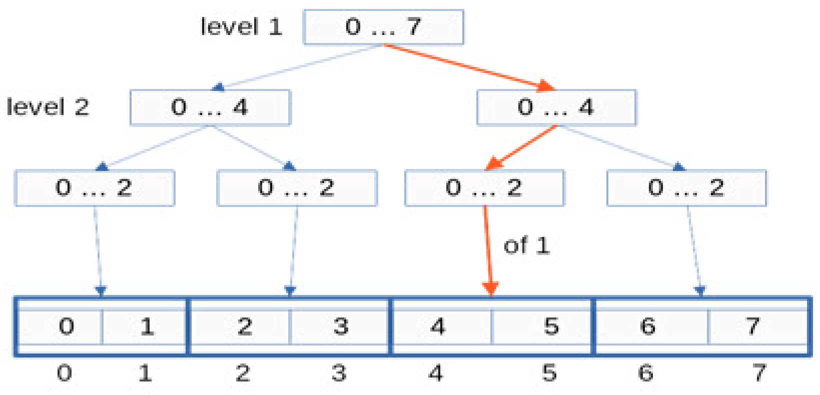
Algorithm (1): Sub-Ranges Sorted Ascending/Descending

Input:a1, b1 (sub-range)

Output: a2, b2 (sibling) then for second sibling

Processes:

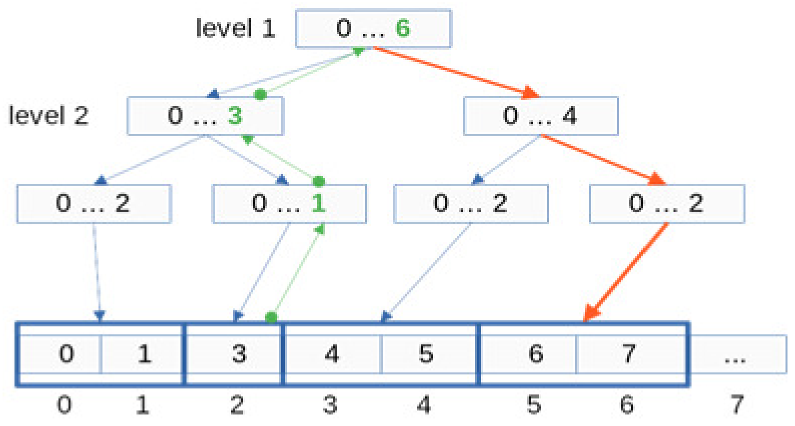
```
        (either a2 = b1 or a2 = b1 + 1) and pos = pos – a2
        state a2 = b1 → if pos >= b1 then
                goto left
        else if pos >= $b_2$ then
                goto right
        else index out of range
        state $a_2$ = $b_1$ + 1 → if pos > $b_1$ then
                goto left
        else if pos > $b_2$ then
                goto right
else index out of range
```



Access to position 5?
level 1 check $\rightarrow$ 5 < 4 $\rightarrow$ false
goto right
right of level1 $\rightarrow$ $a_2 = b_1 = 4$ $\rightarrow$ pos = pos-a_2 = 5-4 = 1
level 2 check $\rightarrow$ 1 < 2 $\rightarrow$ true
left of level2 $\rightarrow$ do nothing
get at sub-range[pos=1] $\rightarrow$ value = 5

FIGURE 5: Sub-Ranges Sorted Ascending/Descending.

If we try to remove valuables (2) and shift all next values, then, as shown in Fig. 6:



Access to position 5?
level 1 check $\rightarrow$ 5 < 3 $\rightarrow$ false
goto right
right of level1 $\rightarrow$ $a_2 = b_1 = 3$ $\rightarrow$ pos = pos-a_2 = 5-3 = 2
level 2 check $\rightarrow$ 2 < 2 $\rightarrow$ false
left of level2 $\rightarrow$ $a_2 = b_1 = 2$ $\rightarrow$ pos = pos-a_2 = 2-2 = 0
get at sub-range[pos=0] $\rightarrow$ value = 6

FIGURE 6: Remove Value (2) and Shift All Next Values in Sub-Ranges.

From the above case, notice that only b's parents have been updated and automatically shift next sub-ranges because the next sub-range depends on previous sub-ranges. Another optimal approach is done by replacing (start, end) by the end because start is always zero, as shown in Fig. 7.

if right child → pos = pos - end1 or pos = pos - end1 – 1

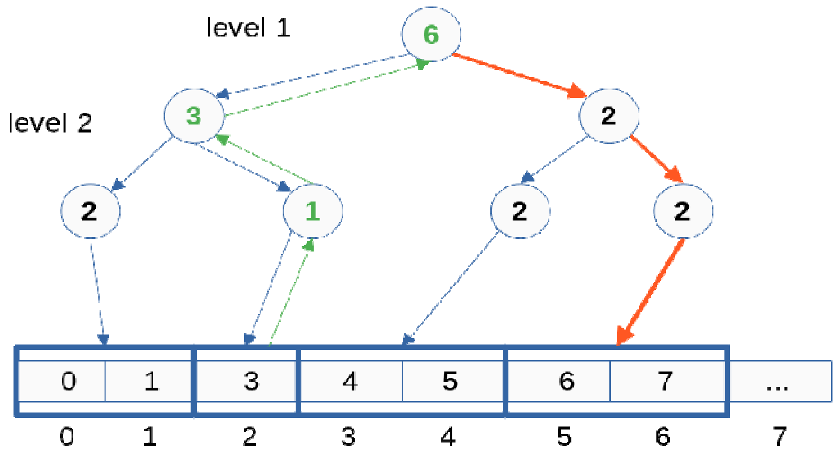


FIGURE 7: Replacing (Start, End) by the End Because Start has been Always Zero.

4.5 Unique Random Generation

Unique random generation is required to generate a large number, If we use unique random choosing it will suffer from space complexity. The unique random choosing is a special case of unique random generation if we use one unique random choosing object within a unique random generation as it will be explained.

4.5.1 Natural Counting System

The natural counting system starts from 0 to the (cardinality-1) like from 0 to 1 and from 0 to 9 and from 0 to F; It looks like as if the chosen number is going to be removed from list until all numbers are deleted from 0 to (cardinality-1) then reset the list and choose one from next rank, as shown in Fig. 8 where the number 5 is now first element after removing previous numbers so the number as position 0 is chosen, then remove again, then choose 6, 7, 8,… till 9.

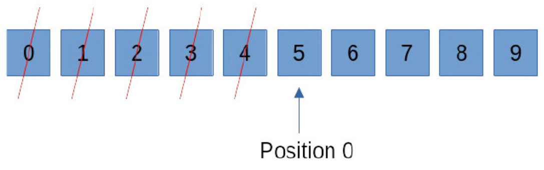


FIGURE 8: The Number 5 is Now First Element.

To explain this case, let's consider a Counting Key just like as before and Counting Key always returns 0 as the next random relative position because it always chooses next number.

4.5.2 Generate with Concept of Ranks

If we look at the Natural Counting System, it can be considered the natural counting as a special case of set of unique choosing objects where we always choose next number. In this case, if we use unique random choosing object for each rank of the specified counting system. E.g. decimal system; then we are going into generating a unique number because counting system is always increasing and can never return to start or previous point.

Generating numbers is done by the sum of each chosen value for each rank * (rank cardinality ^ rank position) or let's call it rank * rank_weight, just like how decimal system has ranked 1, 10. 100, 1000, …. etc. The proposed method, supposed to change the position of rank value by changing rank weight so instead of making a lower order at the beginning, it can be in a different position within the number to increase randomization instead of making the weights fixed.

Theory Example: Let's consider the following case and cardinality=4 and the combination of key1 with rank1 node list produces a sequence [2, 1, 0, 3] and combination of key2 with rank2 node list produces a sequence [1, 3, 2, 0] and rank1 weight = 4^0 or 1 and rank2 weight = 4^1 or 4.

Number Generated	Note
2 + 1*4 = 6	The first generated number
1+1*4 = 5	then choose next number in rank1 and the next number rank2
then next 2 numbers are 4, 7	
The next number is 2 + 3*4 = 14….. etc.	Then reset rank1 and choose next number for rank2. i.e 3 so
2 + 1*4 = 6 1 + 1*4 = 5 0 + 1*4 = 4 3 + 1*4 = 7 2 + 3*4 = 14 1 + 3*4 = 13 0 + 3*4 = 12 3 + 3*4 = 15 2 + 2*4 = 10 1 + 2*4 = 9 0 + 2*4 = 8 3 + 2*4 = 11 2 + 0*4 = 2 1 + 0*4 = 1 0 + 0*4 = 0 3 + 0*4 = 3	At the end, we should have the following numbers

TABLE 2: illustrated the Example.

The second state that generating numbering with change weight is more random than the first state that generation numbering with fixed weights.

5. RESULTS AND DISCUSSION

In this section, consists of several test suites, which are done on random binary numbers. Table 3 illustrated the tests that perform testing of the proposed method, [25].

Test ID	Description
T1	Frequency
T2	Block Frequency
T3	Cumulative Sums
T4	Runs
T5	Longest Run
T6	Rank
T7	FFT
T8	NonOverlaping Template
T9	Overlaping Template
T10	Universal
T11	Approximate Entropy
T12	Serial
T13	Linear Complexity
T14	Random Exclusions
T15	Random Exclusions Variants

TABLE 3: Test ID with Description.

The previous Randomness tests used in this paper to evaluate the seven samples of random numbers that generated by the proposed method with the length of 64 bits, as shown in table 4.

Also evaluate the seven samples of random numbers that caused by NIST tests, as shown in table 4.

Table 4 illustrated the 7 samples of random number generating by the proposed method with binary coding and cardinality = 4.

Sample of Random Number	Length of Sample (Bit)	Binary coding	Decimal number
S1	64	0010-1001-1010-0110-0001-0000-0111-0101-1101-1000-1011-0100-0011-1110-1111-1100	2, 9, 10, 6, 1, 0, 7, 5, 13, 8, 11, 4, 3, 14 , 15, 12
S2	64	1111-0000-0111-1000-0010-1011-1100-1010-1101-0100-0101-1110-0110-0011-1001-0001	15, 0,7, 8, 2, 11, 12, 10,13, 4, 5, 14, 6, 3, 9, 1
S3	64	1010-0000-0110-0010-0100-1111-0011-1001-0111-0001-1011-1101-1110-10000101-1100	10, 0,6, 2, 4, 15, 3,9, 7, 1, 11, 13,14, 8, 5, 12
S4	64	0111-1010-1101-0101-0011-0100-0110-0010-1000-0001-1001-1011-1110-1100-0000-1111	7, 10,13,5,3, 4, 6, 2, 8, 1, 9, 11, 14, 12, 0, 15
S5	64	1110-1100-1010-1000-1111-0101-0001-0000-0111-0011-1001-1011-0100-0110-0010-1101	14, 12, 10, 8, 15, 5, 1,0, 7, 3, 9, 11, 4, 6, 2, 13
S6	64	1100-0001-1111-0110-1101-0011-0101-1110-0111-0010-1001-1000-0100-1010-0000-1011	12, 1, 15, 6, 13, 3, 5, 14, 7, 2, 9, 8, 4, 10, 0, 11
S7	64	1110-1101-0001-1011-0011-0101-0000-1111-0010-0110-0100-1001-0111-1010-1000-1100	14, 13, 1, 11, 3, 5, 0, 15, 2, 6,4, 9, 7, 10, 8, 12

TABLE 4: The 7 Samples of Random Number Generating.

Table 5 illustrated the result of the NIST random tests of the 7 samples of the random number generation that generated by the proposed method and Fig. 9 illustrated the results of NIST randomnesstests.

Test ID	S1	S2	S3	S4	S5	S6	S7
T1	0.9411	0.9901	0.9465	0.9510	0.9876	0.9890	0.9889
T2	0.7865	0.7980	0.7799	0.7855	0.7798	0.7810	0.7870
T3	0.8761	0.8821	0.8890	0.8790	0.8901	0.8711	0.8796
T4	0.9912	0.9789	0.9965	0.9865	0.9901	0.9890	0.9901
T5	0.9091	0.9102	0.9120	0.9110	0.9098	0.9096	0.9104
T6	0.8099	0.8297	0.8123	0.8201	0.8079	0.8120	0.8096
T7	0.9001	0.9231	0.9108	0.9220	0.9190	0.9199	0.9090
T8	0.9798	0.9876	0.9819	0.9769	0.9814	0.9822	0.9790
T9	0.9435	0.9433	0.9432	0.9436	0.9437	0.9434	0.9436
T10	0.9213	0.9234	0.9218	0.9220	0.9219	0.9224	0.9230
T11	0.9980	0.9891	0.9897	0.9910	0.9925	0.9892	0.9962
T12	0.9123	0.9144	0.9125	0.9135	0.9139	0.9127	0.9140
T13	0.9245	0.9267	0.9241	0.9248	0.9246	0.9253	0.9259
T14	0.9432	0.9398	0.9391	0.9401	0.9419	0.9420	0.9427
T15	0.9451	0.9450	0.9456	0.9453	0.9452	0.9451	0.9454

TABLE 5: The Results of NIST Randomness Tests of 7 Samples.

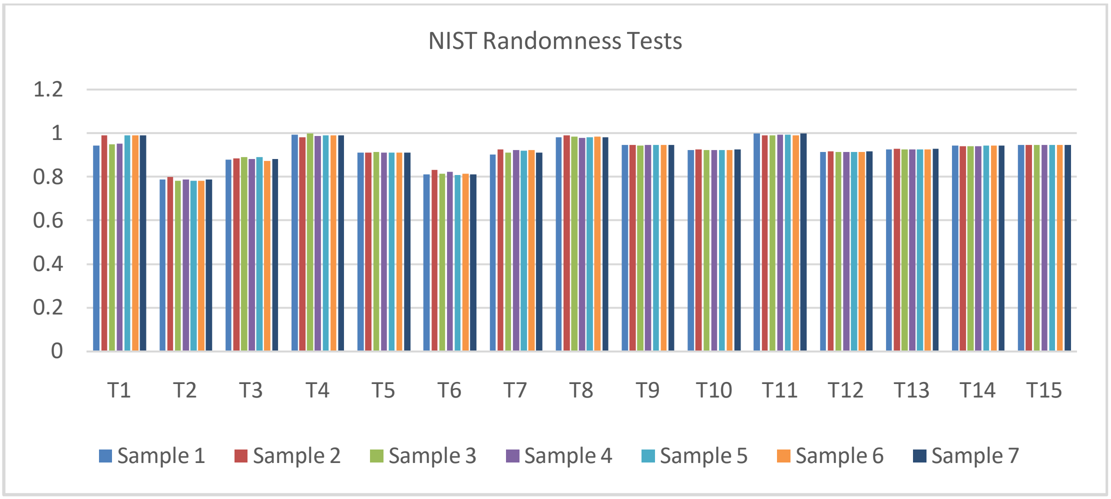


FIGURE 9: Results of the NIST Randomness Tests.

From the above figure, tested 16 samples of random numbers that are generated by this approach, each sample passed all randomness tests, the threshold value is (0.1) that represented in Fig. 9 by the blue dotted line.

In additional, calculated the time generation for each example, table 6 illustrated the time generation for each sample of random number generation that measured by microsecond (us) and Fig. 10 illustrated generation time curve for seven samples.

Sample of Random Number	Generation Time (us)
S1	2.0
S2	2.2
S3	2.1
S4	1.9
S5	2.3
S6	1.8
S7	2.2

TABLE 6: The Time Generation for seven Sample of Random Number Generation.

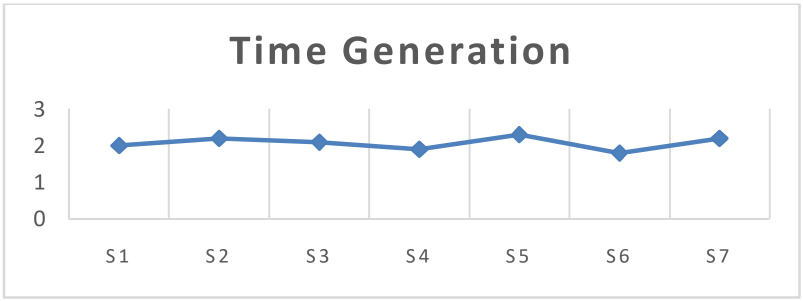


FIGURE 10: Time Generation Curve.

The proposed method does not make use of additional spaces and has a time complexity of O(m), where m is the bit size of a random number.

Also, The Randomness tests used in this paper to evaluate the seven samples of unique choosing numbers that generated by the proposed method with the length of 64 bits, as shown in table 7. Also evaluate the seven samples of random numbers that caused by NIST tests, as shown in table 7.

Sample Of Unique Choosing Number	Length of Sample (Bit)	Binary coding	Decimal number
SU1	64	0001-0101-0011-0111-0110-1010-1011-1101-1110-0000-1001-0010-1111-0100-1100-1000	1, 5, 3, 7, 6, 10, 11, 13, 14, 0, 9, 2, 15, 4 , 12, 8
SU2	64	0011-0110-1001-1100-1010-0010-0101-0111-0000-0001-1110-0100-1111-1000-1011-1101	3, 6,9, 12,10, 2, 5, 7,0, 1, 14, 4, 15, 8,11, 13
SU3	64	0100-0011-0101-1111-0111-1100-1011-1010-0110-0000-0001-0010-1001-1000-1110-1101	4, 3, 5, 15, 7, 12, 11,10, 6, 0, 1, 2,9, 8, 14, 13
SU4	64	1011-1101-0110-1111-0011-1001-1010-0010-0100-0001-1000-1110-0101-0111-1100-0000	11, 13,6,15,3, 9, 10, 2, 4, 1,8, 14, 5, 7,12, 0
SU5	64	0011-1000-1111-0110-0100-1001-0101-1010-0010-0001-1011-1101-1110-0000-0111-1100	3, 8, 15, 6, 4, 9, 5,10, 2, 1, 11, 13,14,0, 7, 12
SU6	64	0000-1100-0100-0001-1110-0010-0101-1010-1000-1001-1101-0110-0111-1011-1111-0011	0, 12, 4, 1, 14, 2, 5, 10, 8,9, 13, 6, 7, 11,15, 3
SU7	64	0101-0100-1100-0110-0011-1110-1111-1001-1011-0000-1000-0111-1101-0001-1010-0010	5, 4, 12, 6, 3, 14, 15, 9, 11, 0,8, 7,13, 1, 10, 2

TABLE 7: The Seven Samples of Unique Choosing Numbers.

Table 8 illustrated the result of the NIST random tests of the 7 samples of the random number generation that generated by the proposed method and Fig. 11 illustrated the results of NIST randomness tests.

Test ID	SU1	SU2	SU3	SU4	SU5	SU6	SU7
T1	0.8311	0.8901	0.8565	0.861	0.8876	0.899	0.8989
T2	0.6765	0.698	0.6899	0.6955	0.6798	0.691	0.697
T3	0.7661	0.7821	0.799	0.789	0.7901	0.7811	0.7896
T4	0.8812	0.8789	0.9065	0.8965	0.8901	0.899	0.9001
T5	0.7991	0.8102	0.822	0.821	0.8098	0.8196	0.8204
T6	0.6999	0.7297	0.7223	0.7301	0.7079	0.722	0.7196
T7	0.7901	0.8231	0.8208	0.832	0.819	0.8299	0.819
T8	0.8698	0.8876	0.8919	0.8869	0.8814	0.8922	0.889
T9	0.8335	0.8433	0.8532	0.8536	0.8437	0.8534	0.8536
T10	0.8113	0.8234	0.8318	0.832	0.8219	0.8324	0.833
T11	0.888	0.8891	0.8997	0.901	0.8925	0.8992	0.9062
T12	0.8023	0.8144	0.8225	0.8235	0.8139	0.8227	0.824
T13	0.8145	0.8267	0.8341	0.8348	0.8246	0.8353	0.8359
T14	0.8332	0.8398	0.8491	0.8501	0.8419	0.852	0.8527
T15	0.8311	0.8901	0.8565	0.861	0.8876	0.899	0.8989

TABLE 8: The Results of NIST Randomness Tests of 7 Samples.

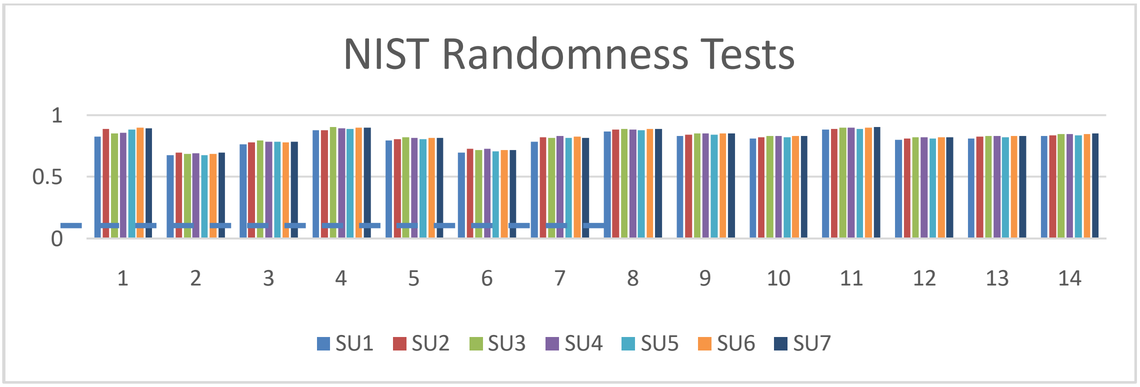


FIGURE 11: Results of the NIST Randomness Tests.

From the above figure, tested seven samples of unique choosing numbers, each sample passed all randomness tests, the threshold value is (0.1) that represented in Fig. 12 by the blue dotted line.

In additional, calculated the time generation for each example table 9 illustrated the time generation for each sample of unique number choosing that measured by microsecond (us) and Fig. 12 illustrated generation time curve for seven samples.

Sample of Unique Random Choosing	Generation Time (us)
SU1	2.7
SU2	2.9
SU3	2.8
SU4	2.6
SU5	3
SU6	2.5
SU7	2.9

TABLE 9: The Time Generation for 7 Sample of Unique Number Choosing.

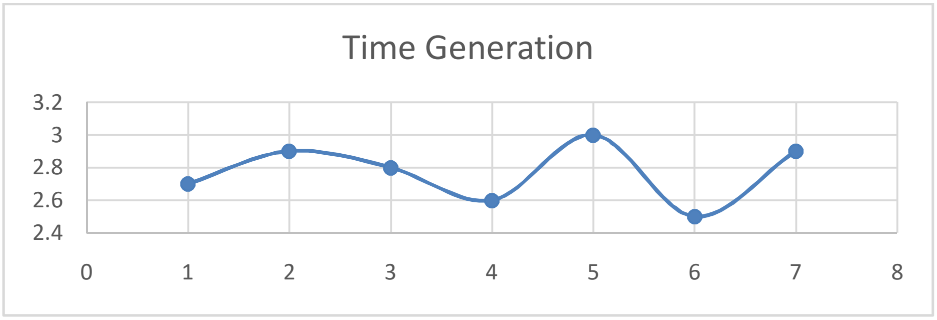


FIGURE 12: Time Generation Curve of UniqeNumbers Choosing.

The proposed method does not make use of additional spaces and has a time complexity of O(m), where m is the bit size of a random number.

Space Complexity for each unique choosing object is O(2^Cardinality) with SequentialNodeList so consider using Unique Random Generation to reduce the space complexity, so instead of using 256 slots for Cardinality = 8 it can be 32 slots by splitting them to 2 unique choosing objects Within a unique random generation objects where Cardinality = 4 bits are used for each rank. Also, Using Cardinality=2 will get maximum entropy and minimum space complexity and maximum Time Complexity.

When comparison between the proposed method and some related work, the result of the comparisonexplains the proposed method success for generating a random number safety for using in the security applications, as shown in table 10.

No.	Method	Seed	Tests	result
[13]	DRBG	input ports for external seed and personalization string for internal seed	Entropy Probability	High Entropy High probability
[19]	Bernoulli discrete-time chaotic map	discrete-time	FIPS-140-2 test NIST-800-22 tests	passed the FIPS-140-2 test suite's four main tests passed all NIST-800-22 tests
[20]	Rando	light, image, voltage, currents, etc., to achieve true randomness	NIST SP 800-22 Time Complexity	Rando is validated its true randomness in NIST SP 800-22 and passes all 15 statistical tests of randomness. The time complexity of Rando is O(m)
Proposed method	Random number Generation based on data instead of a temporal seed	Data(Hex. Decimal)only	NIST-800-22 tests Space Complexity Time Consuming time complexity	passed all NIST-800-22 tests Space Complexity of O(2^Cardinality) Short time time complexity of O(m),

TABLE 10: Comparison between proposed method and related works.

From the previous results, the proposed method success in all NIST tests (high randomness), has a short time in generation (faster method), has the time complexity O(m), and has the space complexity of O(2^Cardinality).

6. CONCLUSIONS

In the security purposes and the gaming industry, random numbers are frequently utilized. In this paper, a new technique that selects a unique number depending on data rather than a time seed (as what the majority of applications doing currently) was conducted. Because it is the foundation for Unique Random Generation and is applied in parallel form, this technique solves the problem at hand. This method has a high-throughput RNG and high-entropy, which indicated that it satisfies the securityrequirements with regard to cryptographic applications. The results show that as soon as the seeds with the right levels of entropy are utilized, itcould generate sequences with a confidence level of 99 % that can't be distinguished fromthe randomness of an ideal random generator, also of highly qualified and validated cryptographic keys. This method of using number generators is simple to use and produces the best results because it selects a unique number based solely on data. The random key is superior to regular keys since it changes over time while regular keys remain fixed where, chose value is not repeated so achieved that by deallocating the specific value also, used a different position within the number to increase randomization instead of the weights are fixed. Where, the generated number with change weight is more random than the generated number with fixed weights. When Using one unique choosing object inside a unique random generator object is similar to using unique choosing as alone. But unique random choosing has a higher entropy than Unique Random Generation. When used base=2 will get maximum entropy and minimum space complexity and maximum Time Complexity. The scope of this paper could be applied to a wide range of RNG features.Also, the proposed method success in all NIST tests (high randomness), has a short time in generation (faster method). This method appropriates for high security applications for generating the best encryption keys.

7. REFERENCES

Adi N. R. K and Vishnuvardhan B., (2014), Secure Linear Transformation Based Cryptosystem using Dynamic Byte Substitution, International Journal of Security (IJS), Vol (8), No (3), PP:24-32.

Challita K., andFarhat H., (2011), Combining steganography and cryptography: new directions, International Journal of New Computer Architectures and their Applications (IJNCAA), Volume 1, Issue 1, pp.199-208.

Chatterjee D.,Nath J.,Dasgupta S., andNath A., (2011), A new Symmetric Key Cryptography Algorithm using extended MSA method: DJSA symmetric key algorithm. Paper presented at the Communication Systems and Network Technologies (CSNT), 2011 International Conference on.

Chavan P. V.,Atiquedan M., andMalik L.,(2014), Design and Implementation of Hierarchical Visual Cryptography with Expansionless Shares, International Journal of Network Security, vol. 6, no. 1, pp. 91-102.

Crocetti L., Matteo S. D., Nannipieri P., Fanucci L. and Saponara S., (2022)," Design and Test of an Integrated Random Number Generator with All-Digital Entropy Source.

Das K. and Bandyopadhyay S. K., (2016), A REVIEW PAPER ON VARIOUS VISUAL CRYPTOGRAPHY SCHEMES, International Journal of Current Research Vol. 8, Issue, 06, pp.32445-32449.

Gamil R.S.Q. and Sanjay N. T., (2012), Encryption and Decryption of Digital Image Using Color signal, IJCSI International Journal of Computer Science Issues, Vol. 9, Issue 2, No 2.

Jyoti T., Anu S., Kishan, Nikhil, Shazad, (2020),Enhanced Visual Cryptography: An Augmented Model for Image Security, International Conference on Computational Intelligence and Data Science (ICCIDS 2019), Elsevier B.V.

Leach P., (2005), A Universally Unique IDentifier (UUID) URN Namespac, LLC R. SalzDataPower Technology.

MangiH.. and YoungminK. , (2017), Unpredictable 16 bits LFSR-based true random number generator.

Maxim integrated, (2019), https://www.maximintegrated.com/en/appnotes/index.mvp/id/4400, last accessed 2019/02/11.

Panda S. and Kavana B. R., (2018), Electronic Document Verification using Visual Cryptography, International Journal of Innovative Science and Research Technology, Volume 3, Issue 11.

Raphael A. J., and Sundaram V., (2011), "Cryptography and Steganography- A Survey", International Journal of Computer Technology and Applications, Volume 2, Issue 3.

Ripon Patgiri, (2021), Rando: A General-purpose True Random Number Generator for Conventional Computers, 2021 IEEE 20th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), Shenyang, China, DOI:10.1109/TrustCom53373.2021.00032.

Ross A. and Othmen A., (2011), Visual Cryptography for Biometric Privacy", IEEE Transactions on Information forensics and security, vol. 6, no. 1, pp. 70- 81.

Rukhin A., Soto J., Smid M., E. Barker, S. Leigh, M. Levenson., M. Vangel, D. Banks, A. Heckert, J.DrayAnd S. Vo,(2010), A statistical test suite for random and pseudorandom number generators for cryptographic applications, NIST Speical Publication 800-22.

Salih E. and Sercan T., (2018), Random Number Generation Using Dual Oscillator Architecture and Discrete-Time Chaos, Conference: 2018 international Symposium on Electronics and Smart Devices (ISESD), DOI: 10.1109/ISESD.2018.8605452.

Selman Y., Taner T., Ahmet B. O., (2019), Secure and Efficient Hybrid Random Number Generator Based on Sponge Constructions for Cryptographic Applications.

Shyu S. J., (2013), Visual Cryptography of Random Grids for General Access Strctures, IEEE, vol. 23. no. 3.

Simona B.,Róbert L., FilipK., and JiríB. , (2016), True Random Number Generator Based on ROPUF Circuit, Published in: 2017 International SoC Design Conference (ISOCC).

Srivatsan I. and Tejas A., (2014), Multi-part Dynamic Key Generation For Secure Data Encryption, International Journal of Security (IJS), Vol (8), No (4), PP:37-46.

Stalling W., (2013), Cryptography And Network Security, 5th Edition.

Wang D., Yi F., Li X., (2009), On General Construction For Extended Visual Cryptography Schemes, Pattern Recognition 42(2009), pp 3071– 3082.

Yadagiri Rao R. and Swetha R., (2013),SECURE VISUAL CRYPTOGRAPHY, International Journal of Scientific & Engineering Research Vol(4), No (3).

Yaprak G. D. and Muhammet K., (2019),A computational method for large-scale differential symmetric Stein equation, Special Issue: ICOMATH2018 - International Conference on Mathematics: An Istanbul Meeting for World Mathematicians 2018, Vol(42), No (16).

Yicheng S., Brenda C. and Christian K., (2016), Random numbers from vacuum fluctuations, Published in: 2016 Euromicro Conference on Digital System Design (DSD).

Yutaka S., (2020), Unpredictable random number generator,Cite as: AIP Conference Proceedings 2286, 040004 (2020); https://doi.org/10.1063/5.0029701 Published Online: 03 December 2020.