IMAGE BASED MULTI-LENGTH RANDOM KEY GENERATOR

Dr. Firas Ali Sabir
University of Baghdad
College of Engineering
Computer Engineering Department

Sadiq Habeeb Abdulhussain
University of Baghdad
College of Engineering
Computer Engineering Department

ABSTRACT

Random Number Generators (RNGs) are an important building block for algorithms and protocols in cryptography. They are dominant in the construction of encryption keys and other cryptographic algorithm parameters. In practice, statistical testing is employed to gather evidence that a generator indeed produces numbers that appear to be random. In this paper a new algorithm is proposed to generate variable length random binary sequence. The random sequence is generated by selecting different point from hashed digital images; the selecting process is organized in such a way to ensure randomness and to avoid regeneration of same sequence within a year. The generated sequences are tested to meet the National Institute of Standard and Technology (NIST) criteria. In proposed algorithm the traditional key exchange is not needed which gives advantage to the system from the security point of view. This proposed algorithm is capable of generating random binary sequences that can meet security requirements of cryptographic algorithms.

الخلاصة:

تعتبر مولّدات العددِ العشوائيةِ (RNG) كتلَ بناء مهمةَ للخوارزمياتِ والإتفاقياتِ في الكتابة المشفرةِ. فهي المسيطرة في بناءِ مفاتيح التشفير ومتغيرات الخوارزميةِ المشفّرةِ الأخرى. من الناحية العملية، يوجد إختبار إحصائي يستَخدم لجمع الادلة لاثبات عشوائية مولّد الاعداد الحقيقة. تم اقتراح خوارزمية جديدة لتَوليد سلسلةِ ثنائيةِ العشوائية متِغيرة الطول. إن السلسلةَ العشوائيةَ مولدةُ بإخْتيار نقاط مختلفة مِن الصور الرقميةِ المثْرومةِ؛ إن عمليةَ الإخْتيار منظمةُ بطريقة بحيث تَضمن العشوائية ولتَفادي تجديدِ السلسلةِ نفسها خلال سنَة. تم اختبار السلاسلَ المولدةَ لمطابقة معايير (NIST). هذه الخوارزميةِ المقتَرحةِ قادرة على تَوليد السلاسل الثنائيةِ العشوائيةِ التي يمكِن أَن تَلبي متطلباتَ أمن الخوارزمياتِ المشفّرةِ.

Keywords: Key Generator, Randomness, NIST, Hypothesis, Hash Function.

1. INTRODUCTION

Randomness and random numbers have traditionally been used for a variety of purposes in many applications such as statistical sampling, experimental simulation, cryptography and etcetera. Computer Algorithm introduces randomness in the form of pseudo-random number generators. As the name suggests, pseudorandom numbers are not truly random. Rather, they are computed from a mathematical formula Pseudorandom numbers have the characteristic that they are predictable, meaning they can be regenerated if you know where in the sequence the first number is starting from [Nur and Sharin, 2010]. Traditionally, the concern in the generation of a sequence of allegedly random numbers has been that the sequence of numbers be random in some well-defined statistical sense [Nur and Sharin, 2010][Nur et al, 2010]. The following two criteria are used to validate that a sequence of numbers is random [William, 2006]:

1. **Uniform distribution:** the distribution of numbers in the sequence should be uniform; that is; the frequency of occurrence of each of

the numbers should be approximately the same.

2. **Independence:** no one value in the sequence can be inferred from the others.

This paper will go through design criteria random key generation in next sections.

2. KEY GENERATION TECHNIQUES

There are several ways to generate keys which are widely used in cryptography. Keys are considered the secret part of any cryptographic system. Its robustness and characteristic specifies the security of any cryptographic system. Some of these techniques are [Bruce, 2006][Menezes, 1997]:

2.1. Pseudorandom Number Generator (PRNGs):

Cryptographic applications typically make use of algorithmic techniques for random number generation. These algorithms are deterministic and therefore produce sequences of numbers that are not statistically random. However, if the algorithm is good, the resulting sequence will pass many reasonable tests of randomness. Such numbers are referred to as ***Pseudo-random numbers***.

A **Feedback shift register** is made up of two parts: a **shift register** and a **feedback function.** The simplest kind of feedback shift register is a linear feedback shift register, or LFSR (see **Fig. 1**). The feedback function is simply the XOR of certain bits in the register; the list of these bits is called a **tap sequence**.

An n-bit LFSR can be in one of 2^n-1 internal states. This means that it can, in theory, generates a 2^n-1 bit long pseudo-random sequence before repeating. Only LFSRs with certain tap sequences will cycle through all 2^n-1 internal states, these are the maximal-period LFSRs. The resulting output sequence is called an **m-sequence.**

The maximal length sequence has the following properties:

- The number of ones in a sequence approximately equals the number of zeros.
- The statistical distribution of ones and zeros is well defined and always the same.

The number of ones and zeros in any linear maximal code is [Bruce, 2006][Menezes, 1997]:

$$\frac{2^n}{2} = number\ of\ ones$$

$$\frac{2^n}{2} - 1 = number\ of\ zeros$$

In order for a particular LFSR to be a maximal-period LFSR, the polynomial formed from a tap sequence plus the constant 1 must be a primitive polynomial mod 2. The *degree* of the polynomial is the length of the shift register. The easiest way is to choose a random polynomial and test whether it is primitive.

2.2. Linear Congruential Generator:

By far, the most widely used technique for pseudo-random number generation is the linear congruential method. The algorithm is parameterized with four numbers, as follows:

m	*the* mod*ulus*	$m > 0$
a	*the multiplier*	$0 < a < m$
c	*the increment*	$0 \le c < m$
X_o	*the starting value, or seed*	$0 \le X_o < m$

The sequence of random numbers $\{X_n\}$ is obtained via the following iterative equation:

$$X_{n+1} = (aX_n + c) \bmod m$$

If m,a,c, and Xo are integers, then this technique will produce a sequence of integers with each integer in the range $0 \le X_n < m$.

The selection of values for *a*,*c*, and *m* is critical in developing a good random number generator. For example, consider *a=c=1*. The sequence produced is obviously not satisfactory. Now consider the values *a=7*, *c=0*, *m=32*, and X_o*=1.* This generates the sequence {7,17,23,1,7, etc.}, which is also clearly unsatisfactory. Of the 32 possible values, only 4 are used; thus, the sequence is said to have a period of 4. if, instead, we change the value of *a* to 5, then the sequence is {5,25,29,17,21,9,13,1,5, etc.}, which increases the period to 8.

We would like *m* to be very large, so that there is the potential for producing a long series of distinct random numbers.

Unfortunately, linear congruential generators cannot be used for cryptography; they are predictable. Linear congruential generators were broken. Quadratic generators denoted by the following equation were also broken [Bruce, 2006][Menezes, 1997].

$$X_{n+1} = (aX_n^2 + bX_n + c) \bmod m \qquad \ldots(1)$$

And cubic generator:

$$X_{n+1} = (aX_n^3 + bX_n^2 + cX_n + d) \bmod m \ldots(2)$$

Linear congruential generators remain useful for non-cryptographic applications, however, such as simulations. They are efficient and show good statistical behavior with respect to most reasonable empirical tests.

2.3. Blum Blum Shub Generator:

A popular approach to generate secure pseudorandom is known as Blum, Blum, Shub (BBS) generator, named for its developers. It has perhaps the strongest public proof of its cryptographic strength. The procedure is as follows. First, choose two large prime numbers, p and q, that both have a remainder of 3 when divided by 4. That is, $p \equiv q \equiv 3 (\bmod 4)$.

For example, the prime numbers 7 and 11 satisfy $7 \equiv 11 \equiv 3 (\bmod 4)$. Let $n = p \times q$. Next, choose a random number s, such that s is relatively prime to n; this is equivalent to say that neither p nor q is a factor of s. then the BBS generator produces a sequence of bits according to the following algorithm. Thus, the least significant bit is taken at each iteration.

```
start
$X_o = s^2 \bmod n$
for $i = 1$ to $\infty$
$X_i = (X_{i-1})^2 \bmod n$
$Generated\,Key_i = X_i \bmod 2$
end
```

3. STATISTICAL TEST SUITE FOR RNG AND PRNG

The focus of this paragraph is on the way to examine whether the generated sequence has randomness for cryptographic purposes. A set of statistical tests for randomness is described in this paragraph. The National Institute of Standards and Technology (NIST) believe that these procedures are useful in detecting deviations of a binary sequence from randomness. Various statistical tests can be applied to a sequence in attempt to compare and evaluate the sequence to a truly random sequence. Randomness is a probabilistic property; that is, the properties of a random sequence can be characterized and described in terms of probability. The likely outcome of statistical tests, when applied to a truly random sequence, is known a prior and can be described in probabilistic terms. There are an infinite number of possible statistical tests, each assessing the presence or absence of a "pattern" which, if detected, would indicate that the sequence is nonrandom. Because there are so many tests for judging whether a sequence is random or not, no specific finite set of tests is deemed "complete." In addition, the results of statistical testing must be interpreted with some care and caution to avoid incorrect conclusions about a specific generator and the purpose of each test is given [Andrew *et al*, 2008][Andrew and Walter, 2003]:

3.1. Frequency (Monobit) Test

The focus of the test is the proportion of zeroes and ones for the entire sequence. The purpose of this test is to determine whether the number of ones and zeros in a sequence are approximately the same as would be expected for a truly random sequence [Andrew *et al*, 2008].

3.2. Frequency Test within a Block

The focus of the test is the proportion of one's within M-bit blocks. The purpose of this test is to determine whether the frequency of ones in an M-bit block is approximately M/2, as would be expected under an assumption of randomness [Andrew *et al*, 2008].

3.3. Runs Test

The focus of this test is the total number of runs in the sequence, where a run is an uninterrupted sequence of identical bits. A run of length *k* consists of exactly *k* identical bits and is bounded before and after with a bit of the opposite value. The purpose of the runs test is to determine whether the number of runs of ones and zeros of various lengths is as expected for a random sequence. In particular, this test determines whether the oscillation between such zeros and ones is too fast or too slow [Andrew *et al*, 2008].

3.4. Test for the Longest Run of Ones in a Block

The focus of the test is the longest run of ones within M-bit blocks. The purpose of this test is to determine whether the length of the longest run of ones within the tested sequence is consistent with the length of the longest run of ones that would be expected in a random sequence [Andrew *et al*, 2008].

3.5. Discrete Fourier Transform (Spectral) Test

The focus of this test is the peak heights in the Discrete Fourier Transform of the sequence. The purpose of this test is to detect periodic features (i.e., repetitive patterns that are near each other) in the tested sequence that would indicate a deviation from the assumption of randomness. The intention is to detect whether the number of peaks exceeding the 95 % threshold is significantly different than 5 % [Andrew *et al*, 2008].

3.6. Serial Test

The focus of this test is the frequency of all possible overlapping *m*-bit patterns across the entire sequence. The purpose of this test is to determine whether the number of occurrences of the 2*m* *m*-bit overlapping patterns is approximately the same as would be expected for a random sequence. Random sequences have uniformity; that is, every *m*-bit pattern has the same chance of appearing as every other *m*-bit pattern [Andrew *et al*, 2008].

3.7. Approximate Entropy Test

The focus of this test is the frequency of all possible overlapping m-bit patterns across the entire sequence. The purpose of the test is to compare the frequency of overlapping blocks of two consecutive/adjacent lengths (*m* and *m+1*) against the expected result for a random sequence [Andrew *et al*, 2008].

3.8. Cumulative Sums (Cusum) Test

The focus of this test is the maximal excursion (from zero) of the random walk defined by the cumulative sum of adjusted (-1, +1) digits in the sequence. The purpose of the test is to determine whether the cumulative sum of the partial sequences occurring in the tested sequence is too large or too small relative to the expected behavior of that cumulative sum for random sequences. This cumulative sum may be considered as a random walk. For a random sequence, the excursions of the random walk should be near zero. For certain types of non-random sequences, the excursions of this random walk from zero will be large [Andrew *et al*, 2008].

4. RANDOM HYPOTHESIS TESTING

A statistical test is formulated to test a specific null hypothesis (H0). For the purpose of this study, the null hypothesis under test is that the sequence being tested is random against the alternative hypothesis (H1) for which the sequence is not random [Andrew et al, 2008][Nur and Sharin, 2010][Nur et al, 2010].

For each statistical test, a set of p-values (corresponding to the set of sequences) is produced. For a fixed significant level, a certain percentage of p-values are expected to indicate failure. For example, if the significant level is chosen to be 0.01 (i.e. α=0.01), then about 1% of the sequences are expected to fail. A sequence passes a statistical test whenever the p-values $\geq \alpha$ and fails otherwise [Wong et al, 2009][Nur and Sharin, 2010][Nur et al, 2010].

The parameter denotes the significance level that determines the critical region of acceptance and rejection. NIST recommended that α be in the range (0.001, 0.01) [Andrew et al, 2008][Nur and Sharin, 2010][Nur et al, 2010][Wong et al, 2009].

Only 8 tests are particularly suitable for practical cryptographic keys size here. The selected NIST random tests for short keys are listed in the **Table (1)**. This set of random test shall be called upon to check on the validity of short random for proposed method. The 8 selected tests are basically relies heavily on the randomness of the binary sequence. **Fig. (2)** shows the hierarchy of the tests. Once a particular block key set fails one test it is considered non-random and will certainly fail the next test in lower hierarchy [Wong et al, 2009][Nur and Sharin, 2010].

Additional numerical experiments should be conducted on different samples of the generator to determine whether the phenomenon was a statistical anomaly or a clear evidence of non-randomness.

For the interpretation of test results, NIST adopts following two approaches, the examination of the proportion of success-sequences (Success Rate). The range of acceptable proportions is determined using the confidence interval defined as [Charmaine, 2005][Andrew et al, 2008][Juan, 1999][R. B. P. Dept., 2003]:

$$P^{'} - Value = P^{'} \mp 3\sqrt{\frac{P^{'}(1-P^{'})}{n}} \qquad \ldots(3)$$

Where $p' = 1-\alpha$, and n is the sample size.

If the proportion falls outside of this interval, then there is evidence that the data is nonrandom.

5. PROPOSED ALGORITHM

The idea of proposed model which is shown in **Fig. (3)** is to generate multi length random binary sequences suitable for use in cryptography and other applications. The seeds of the proposed model are selective images and the date. In this model four lengths of key are possible to be generated which are 128, 256, 512, and 1024 bits. Visual Basic 6.0 is used for model implementation. The function of blocks is illustrated in the coming paragraphs.

5.1 Input : In this block the end user enters date or could use the auto dating option which leads to current date. User also should enter number of bits to be taken from the selected pixel (1, 2, 4 or 8) in order to specify the key length.

5.2 Generating X,Y,S: In this phase, three carefully selected equations are used to obtain one-to-one system that there are no duplicate in(X,Y,S) values within a year. These values are then used as initial value (X_0, Y_0, S_0) of the seven bit PN- generator (7 Xor 6 for maximum-length LFSR [Peter, 1996]) resulting in 127 different states. The PN generator denoted by (S) is used to select one of the 43 generated images for each state according to the equation [Integer(S/3) +1] and also it has been used to determine which byte to be deal with (Red, Green, or Blue). This is done through the use of the following equation [Round(S/3 – Image Number)*3+3] which must give [0, 1 or 2] indicating red, green or blue respectively.

The other PN generators marked (X and Y) are used to coordinate the pixel position by intersecting the x-axis and y-axis on the selected image. This process goes on for all states of PN generator resulting in 127 pixels randomly generated.

5.3 Main Module: The inputs to this phase are 127 bytes which are randomly chosen and number of bits (LSBs) to be taken from these bytes which is determined by user to specify how long the key is. The keys are then padded according to certain rules to reach predefined key lengths. The resulting keys are then examined by NIST test to evaluate its randomness and to check whether it is valid for cryptographic application or not.

6. RESULTS AND DISCUSSION

As previously mentioned, the proposed algorithm generates multi-length keys. These keys are examined for randomness issue under the NIST tests. The obtained results for 10 key sequences of different lengths are shown in **Tables (2, 3, 4, and 5)** as all p-values of the test are larger than required value (α=0.01) in order to reject the null hypothesis as random sequence.

Fig. (4, 5, 6, and 7) show the average value for 128, 256, 512 and 1024 bits key length respectively the tests with 10 generated sequences pass the threshold value (α)[Nur and Sharin, 2010].

For long sequences more than 10 sequences another tests will be taken into account which gives an indication for sequence randomness. It has been shown that as long as the sequence length increases, the possibility of fail mark may appear, so success rate test should be done to validate sequences for use in cryptographic application with respect to their randomness. **Table (6)** clearly illustrates the success rate test for multi-length keys according to equation (3)

$$\begin{aligned} P^{'} - Value &= 0.99 \mp 3\sqrt{\frac{0.99 * 0.01}{365}} \\ &= 0.99 \mp 0.015624 = \begin{cases} 1.005625 \\ 0.974376 \end{cases} \end{aligned} \qquad \ldots(4)$$

Tables (7,8 and 9) shows the proportion test for Linear Feedback Shift Register (LFSR), Blum Blum Shub (BBS) and Linear Congruential Generator (LCG) respectively, and show that these techniques fail in some of the tests. From **Table (10)** its obvious that the proposed algorithm show better results when compared with other techniques.

7. SOFTWARE IMPLEMENTATION

The proposed method is implemented using Visual Basic 6.0 as shown in **Fig. (8)** and **Fig. (9)**. It can be seen from these two figures the flexibility and the reliability taken in software design. It is very easy to the interested people in cryptographic field use this application and generate the random multi-length keys by specifying the input parameters by pressing *generate* button and *Select No. of Bits* button and then follow the procedure.

8. CONCLUSION

In this paper, a new method of key generation is proposed and modeled depending on hashed images and date-dependent algorithm. The proposed method of key generation is examined and tested for different cases and different key lengths and compared with classical techniques and it is well proven that this method is very suitable for applications used in cryptography as it has random nature and has all properties of randomness.

9. REFERENCES

- Andrew Rukhin, *et al* "A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications", National Institute of Standard and Technology, 2008.
- Andrew W, Walter A., "Hardware comparison of seven random number generators for smart cards", In: ITG-GI-GMM Workshop of Test Methods and Reliability of Circuits and Systems, Timmendorfer Beach, pp. 55–58, 2003.
- Bruce Schneier, "Applied Cryptography", John Wiley and Sons, 2nd Edition, 1996.
- Charmaine Kenny, "Random Number Generators: An Evaluation and Comparison of Random.org and Some Commonly Used Generators" the distributed systems group, Computer Science Department, TCD, 2005.
- Juan Soto, "Statistical Testing of Random Number Generators", Proceedings of the 22nd National Information Systems Security Conference, 1999.
- Menezes A., van Oorschot P., and Vanstone S., "Handbook of Applied Cryptography", CRC Press, 1997.
- Nur A. Abu, and Sharin Sahib, "Random ambience key generation live_on demand", Signal Processing Systems (ICSPS), 2010 2nd International Conference on 2010, Vol. 1, pp. 110-114.
- Nur A. Abu, Nanna S. Herman, and Sharin Sahib, "An Enhancement of the statistical test for randomness", Networking and Information Technology (ICNIT), 2010 International Conference 2010, pp. 521-525.
- Peter Alfke, "Efficient Shift Registers, LFSR Counters, and Long Pseudo-Random Sequence Generators", Xilinx application note, 1996.
- R. B. P. Dept. "The Evaluation of Randomness of RPG100 by Using NIST and DIEHARD Tests". Technical report, FDK Corporation, 2003.
- William Stallings, "Cryptography and Network Security", Perntice Hall, Fourth Edition, 2006.
- Wong Siaw Lang, Nur Azman Abu, Shahrin Sahib, "Cryptographic Key From Webcam Image", International Journal of Cryptology Research 2009, vol. 1, pp. 115-127.

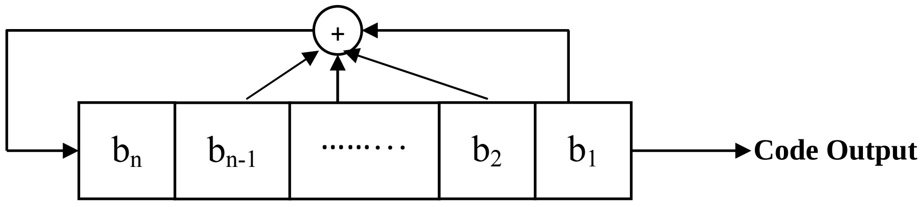


Fig. (1): Typical linear sequence generator using LFSR.

Table(1). List of suitable tests for short keys.

Test Code	*Statistical Test*	*Test Parameters*
1	Frequency Test	No Parameter
2	Block Frequency	M=8
3a	Cumulative Sum (Forward)	No Parameter
3b	Cumulative Sum (Backward)	No Parameter
4	Runs	No Parameter
5	Longest Run of Ones	M=8
6	Spectral DFT	No Parameter
7	Approximate Entropy	m=7
8a	Serial P-Value1	m=7
8b	Serial P-Value2	m=7

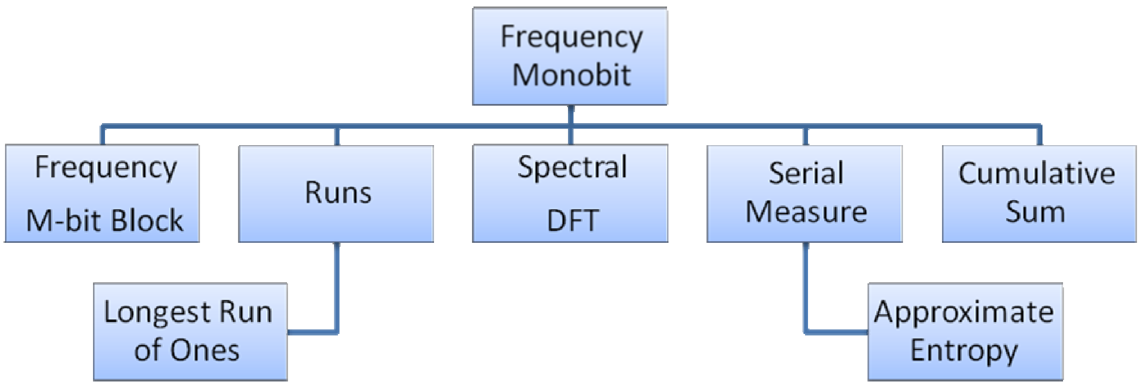


Fig. (2). The hierarchy of Random tests

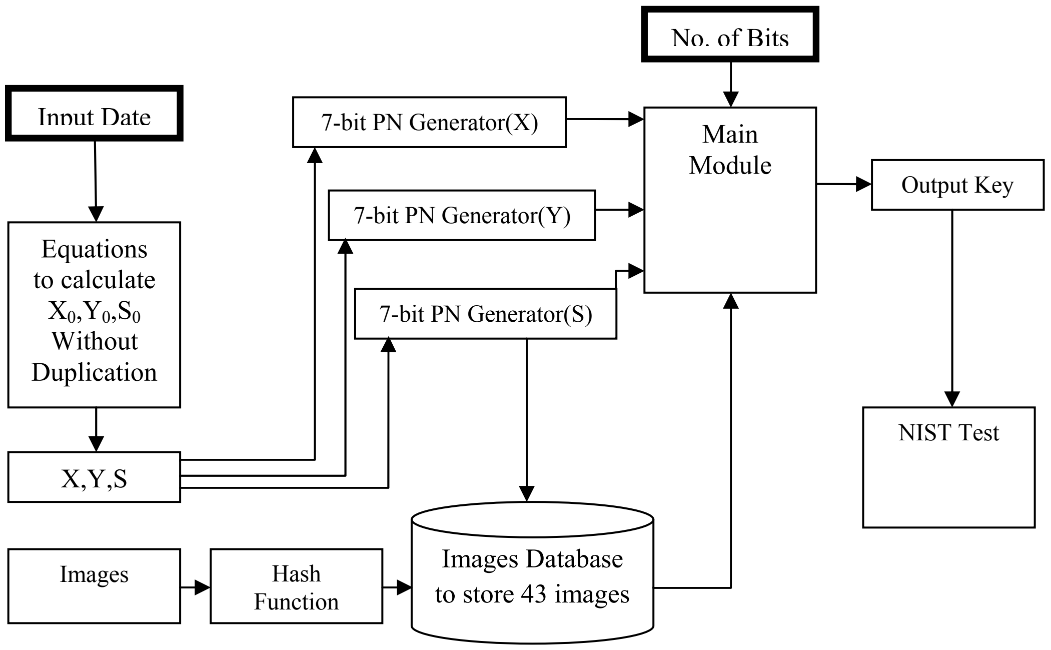


Fig. (3) The proposed Model

Table (2) 128 bit Key p-values for 10 sequence test result

Test / Sequence	1	2	3	4	5	6	7	8	9	10
Frequency	0.3768	1.0000	0.8597	0.4795	0.7237	0.0339	0.2159	0.2159	0.0771	0.5959
Block Frequency	0.3540	0.8095	0.2436	0.7440	0.1432	0.0110	0.5615	0.8392	0.1432	0.1010
Cusum (Forward)	0.0841	0.9842	0.8188	0.5748	0.7375	0.0543	0.3697	0.3697	0.1037	0.8188
Cusum (Backward)	0.4999	0.9842	0.8920	0.6548	0.8920	0.0207	0.3146	0.4314	0.1542	0.4999
Runs	0.9137	0.3768	0.7257	0.4507	0.6032	0.8189	0.3939	0.8252	0.9368	0.7418
Longest Run	0.5837	0.0788	0.7632	0.4360	0.4749	0.0371	0.8569	0.1068	0.3374	0.9936
Spectral DFT	0.5164	0.5164	0.5164	0.5164	0.5164	0.5164	0.3304	0.1443	0.0231	0.5164
Approx. Entropy	0.9456	0.9445	0.9623	0.8460	0.9025	0.7680	0.6176	0.7601	0.9155	0.6912
Serial 1	0.1121	0.7202	0.3427	0.2063	0.1636	0.0149	0.5119	0.1841	0.0743	0.3745
Serial 2	0.0382	0.9513	0.3944	0.7411	0.6192	0.2490	0.6694	0.2867	0.5937	0.0278
Minimum	0.0382	0.0788	0.2436	0.2063	0.1432	0.0110	0.2159	0.1068	0.0231	0.0278
Maximum	0.9456	1.0000	0.9623	0.8460	0.9025	0.8189	0.8569	0.8392	0.9368	0.9936

Table (3) 256 bit Key p-values for 10 sequence test result

Test / Sequence	1	2	3	4	5	6	7	8	9	10
Frequency	0.5320	0.3816	0.7077	0.4533	0.7077	0.8026	0.2606	0.0801	0.8026	0.4533
Block Frequency	0.6939	0.7853	0.5425	0.2867	0.4917	0.9862	0.2674	0.5170	0.9074	0.1565
Cusum (Forward)	0.5731	0.5197	0.9459	0.5731	0.6872	0.9980	0.2363	0.0415	0.8580	0.6292
Cusum (Backward)	0.9459	0.4693	0.8035	0.8035	0.7458	0.9064	0.2083	0.1216	0.9908	0.2672
Runs	0.1268	0.2800	0.6107	0.3336	0.3213	0.1678	0.0113	0.2804	0.0794	0.9719
Longest Run	0.7165	0.5793	0.1040	0.5855	0.8809	0.0260	0.8077	0.3157	0.9560	0.8894
Spectral DFT	0.3588	0.0512	0.0512	0.8185	0.3588	0.3588	0.8185	0.1359	0.4220	0.4220
Approx. Entropy	0.1871	0.2364	0.0447	0.0330	0.0280	0.0921	0.1294	0.0255	0.1405	0.4462
Serial 1	0.5731	0.5197	0.9459	0.5731	0.6872	0.9980	0.2363	0.0415	0.8580	0.6292
Serial 2	0.9459	0.4693	0.8035	0.8035	0.7458	0.9064	0.2083	0.1216	0.9908	0.2672
Minimum	0.1268	0.0512	0.0447	0.0330	0.0280	0.0260	0.0113	0.0255	0.0794	0.1565
Maximum	0.9459	0.7853	0.9459	0.8185	0.8809	0.9980	0.8185	0.5170	0.9908	0.9719

Table (4) 512 bit Key p-values for 10 sequence test result

Test / Sequence	1	2	3	4	5	6	7	8	9	10
Frequency	0.3768	0.9296	0.7909	0.1116	0.2888	0.2505	0.8597	0.1573	0.0421	0.1849
Block Frequency	0.3273	0.0797	0.7666	0.3427	0.7360	0.6361	0.2063	0.0407	0.0916	0.2063
Cusum (Forward)	0.0196	0.6960	0.8188	0.8188	0.2040	0.2040	0.5748	0.8920	0.8188	0.4999
Cusum (Backward)	0.2348	0.9231	0.7375	0.8188	0.4649	0.1542	0.2438	0.8920	0.6548	0.4999
Runs	0.5756	0.3750	0.4797	0.9296	0.2848	0.3595	0.9724	0.2159	0.1582	0.1329
Longest Run	0.2471	0.7019	0.2690	0.7971	0.7815	0.8187	0.7650	0.4566	0.7111	0.1910
Spectral DFT	0.7456	0.3723	0.0885	0.7456	0.4654	0.1233	0.1233	0.1233	0.4654	0.4654
Approx. Entropy	0.5159	0.6769	0.1723	0.2719	0.4572	0.4396	0.4846	0.7793	0.7802	0.3599
Serial 1	0.0261	0.4245	0.5565	0.7740	0.8953	0.2559	0.8092	0.4416	0.3197	0.3427
Serial 2	0.0927	0.2068	0.3170	0.8935	0.8258	0.0435	0.7691	0.4792	0.4792	0.1111
Minimum	0.0196	0.0797	0.0885	0.1116	0.2040	0.0435	0.1233	0.0407	0.0421	0.1111
Maximum	0.7456	0.9296	0.8188	0.9296	0.8953	0.8187	0.9724	0.8920	0.8188	0.4999

Table (5) 1024 bit Key p-values for 10 sequence test result

Test / Sequence	1	2	3	4	5	6	7	8	9	10
Frequency	0.0244	0.2880	0.4918	0.3485	0.5320	0.4165	0.2606	0.9005	1.0000	0.5320
Block Frequency	0.1319	0.2298	0.7367	0.2298	0.9625	0.1259	0.7881	0.0807	0.0585	0.0726
Cusum (Forward)	0.0108	0.2220	0.3193	0.3785	0.6580	0.3382	0.3382	0.9742	0.7458	0.4693
Cusum (Backward)	0.0351	0.3579	0.7458	0.2220	0.4941	0.3382	0.1713	0.9064	0.7458	0.3785
Runs	0.0964	0.7752	0.6274	0.9720	0.2301	0.5877	0.1567	0.2115	0.0699	0.1539
Longest Run	0.0618	0.6673	0.4714	0.8769	0.2426	0.9741	0.1676	0.9077	0.4156	0.0299
Spectral DFT	0.3296	0.6881	0.3296	0.3019	0.6881	0.3296	0.4559	0.4559	0.1871	0.1871
Approx. Entropy	0.0977	0.1775	0.0125	0.3482	0.1919	0.4139	0.2546	0.8374	0.0098	0.0448
Serial 1	0.2461	0.4034	0.4546	0.2151	0.4076	0.8753	0.2798	0.8024	0.1277	0.0586
Serial 2	0.5329	0.2722	0.1990	0.2148	0.6632	0.9631	0.5075	0.7853	0.0413	0.0501
Minimum	0.0108	0.1775	0.0125	0.2148	0.1919	0.1259	0.1567	0.0807	0.0098	0.0299
Maximum	0.5329	0.7752	0.7458	0.9720	0.9625	0.9741	0.7881	0.9742	1.0000	0.5320

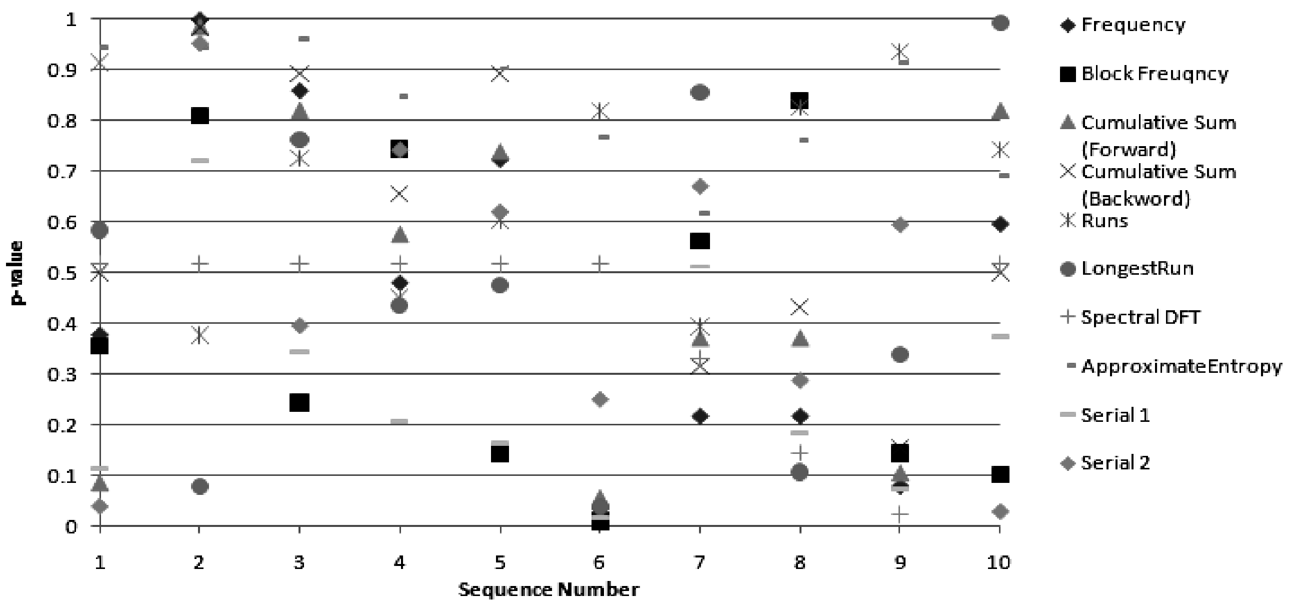


Fig. (4) 128 bit Key p-values and average of p-values

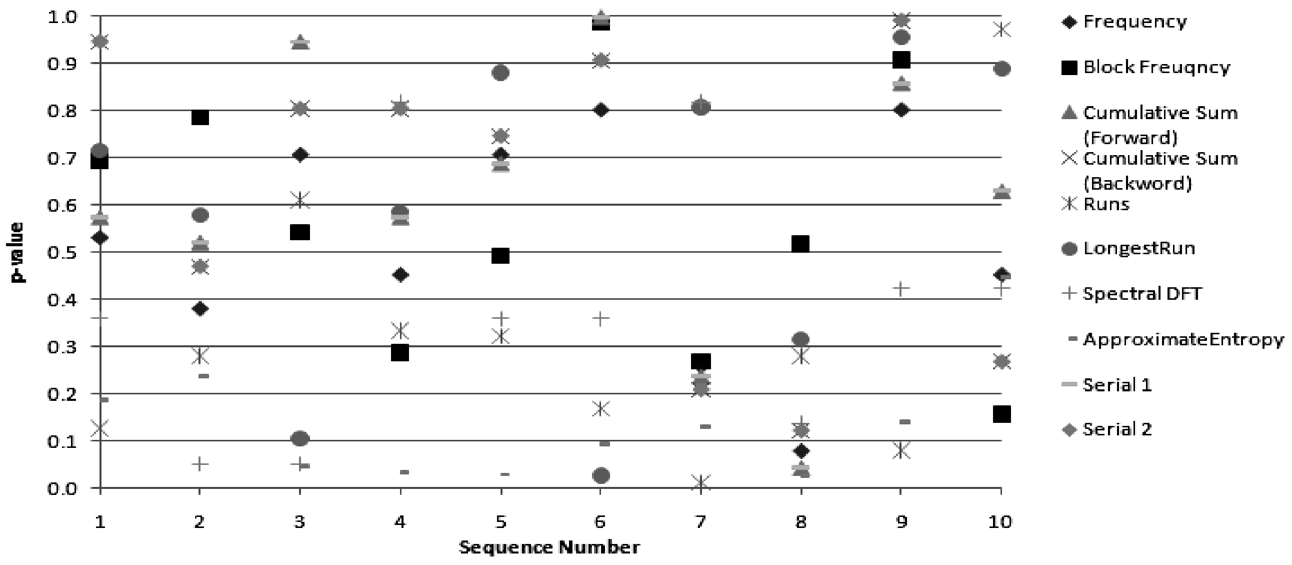


Fig. (5) 256 bit Key p-values and average of p-values

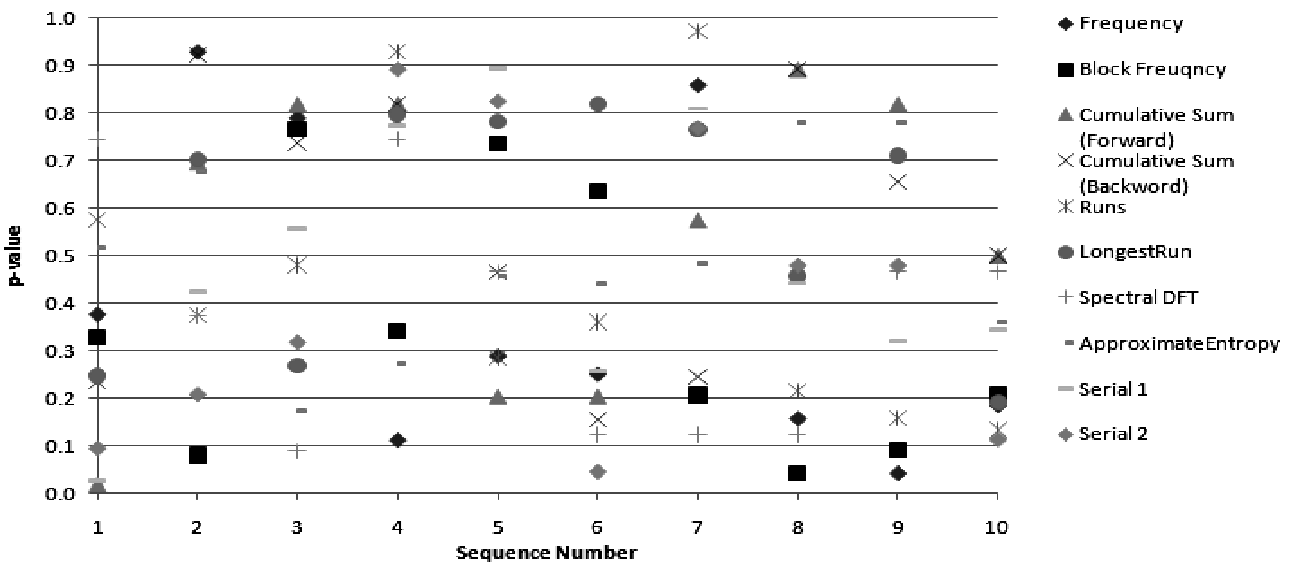


Fig. (6) 512 bit Key p-values and average of p-values

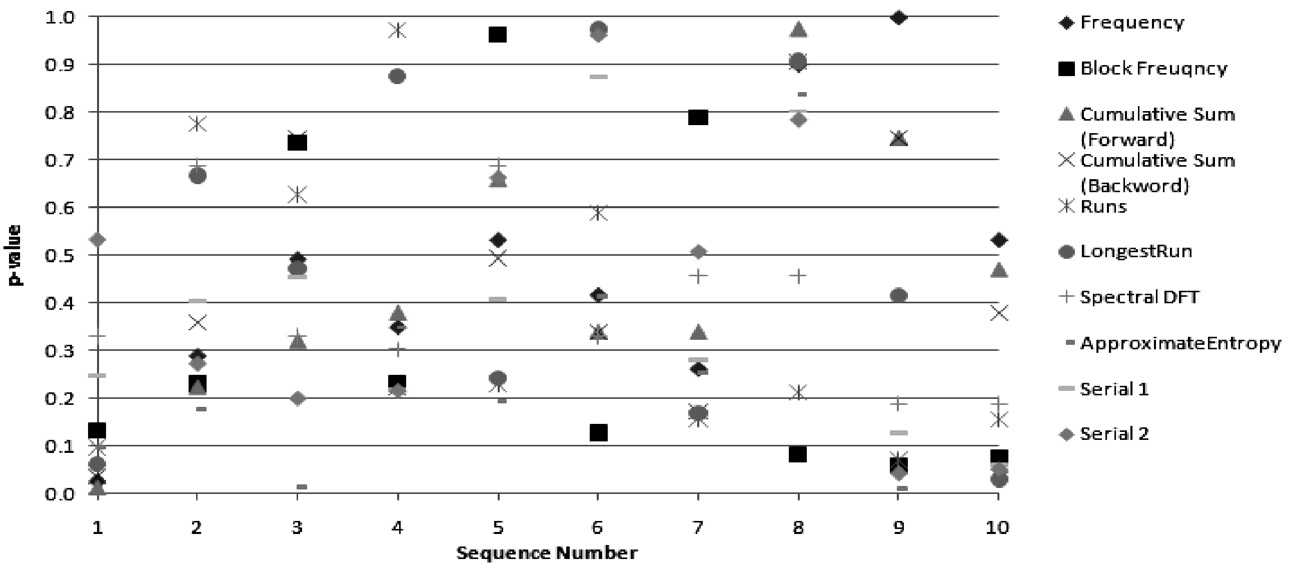


Fig. (7) 1024 bit Key p-values and average of p-values

Table (6) p-value proportion test for 365 (one year) sequences

Test Code	p-value proportion of 128 bit	p-value proportion of 256 bit	p-value proportion of 512 bit	p-value proportion of 1024 bit
1	0.98904	0.99178	0.99178	1.00000
2	0.98904	0.99726	0.99178	0.98630
3a	0.98630	0.98904	0.98356	0.99726
3b	0.99178	0.99726	0.99452	0.99452
4	0.99452	0.98630	0.99726	0.99178
5	0.99452	0.98904	0.99178	0.98904
6	0.97808	0.98904	0.98904	0.98630
7	1.00000	0.97534	0.97808	0.99452
8a	0.98904	0.98356	0.99452	0.99726
8b	0.99178	0.97808	0.99726	1.00000

Table (7) p-value proportion test for 365 (one year) sequences for LFSR

Test Code	p-value proportion of 128 bit	p-value proportion of 256 bit	p-value proportion of 512 bit	p-value proportion of 1024 bit
1	1.00000	1.00000	1.00000	1.00000
2	1.00000	1.00000	1.00000	1.00000
3a	1.00000	1.00000	1.00000	1.00000
3b	1.00000	1.00000	1.00000	1.00000
4	1.00000	1.00000	1.00000	1.00000
5	1.00000	1.00000	1.00000	1.00000
6	0.30136	0.20273	0	0
7	0.21917	1.00000	1.00000	1.00000
8a	1.00000	1.00000	1.00000	1.00000
8b	1.00000	1.00000	1.00000	1.00000

Table (8) p-value proportion test for 365 (one year) sequences for BBS

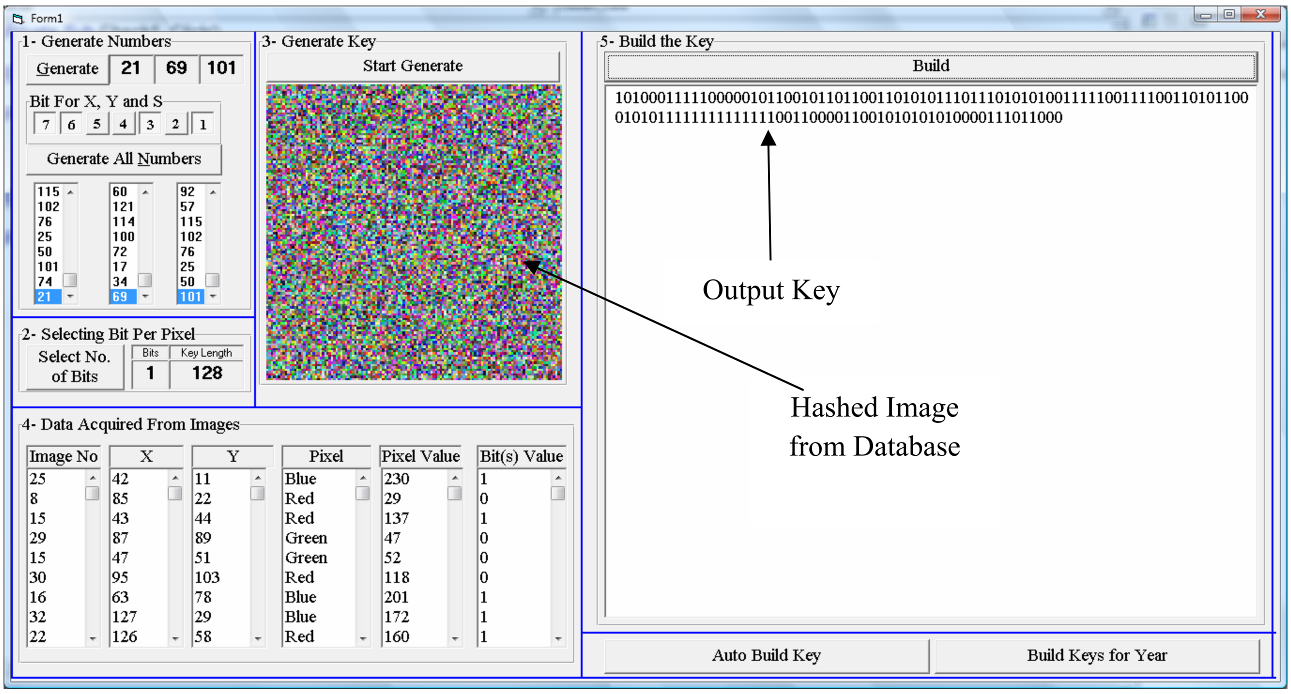
Test Code	p-value proportion of 128 bit	p-value proportion of 256 bit	p-value proportion of 512 bit	p-value proportion of 1024 bit
1	0.99452	1.00000	0.99452	0.98904
2	0.99178	0.98630	1.00000	0.99726
3a	0.99452	1.00000	0.99178	0.99178
3b	0.98630	1.00000	0.99452	0.98904
4	0.98630	0.98630	0.98630	0.98904
5	0.99452	0.99452	0.99726	0.98356
6	0.98630	0.99178	0.99452	0.98904
7	1.00000	0.93698	0.81369	0.92602
8a	0.98356	0.98356	0.98904	0.98082
8b	0.98904	0.98904	0.99178	0.98904

Table (9) p-value proportion test for 365 (one year) sequences for LCG

Test Code	p-value proportion of 128 bit	p-value proportion of 256 bit	p-value proportion of 512 bit	p-value proportion of 1024 bit
1	0.98082	0.99178	0.99726	0.98630
2	0.99178	1.00000	1.00000	0.99452
3a	0.98904	0.98630	1.00000	0.98356
3b	0.98082	0.99178	0.99452	0.98356
4	0.98904	0.99178	0.98356	0.98630
5	0.99452	0.99726	0.99726	0.98082
6	0.98904	0.99452	0.99726	0.98082
7	1.00000	0.95616	0.75068	0.92602
8a	0.97260	0.97808	0.99452	0.98630
8b	0.98904	0.99452	0.99726	0.99452

Table (10) the Pass / Fail for the proposed algorithm (TPA), LFSR, BBS, and LCG

Test Code	p-value Pass/Fail for 128 bit				p-value Pass/Fail for 256 bit				p-value Pass/Fail for 512 bit				p-value Pass/Fail for 1024 bit			
	TPA	LFSR	BBS	LCG	TPA	LFSR	BBS	LCG	TPA	LFSR	BBS	LCG	TPA	LFSR	BBS	LCG
1	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P
2	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P
3a	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P
3b	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P
4	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P
5	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P
6	P	F	P	P	P	F	P	P	P	F	P	P	P	F	P	P
7	P	F	P	P	P	P	F	F	P	P	F	F	P	P	F	F
8a	P	P	P	F	P	P	P	P	P	P	P	P	P	P	P	P
8b	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P
	(P = Pass) (F = Fail)															



Generate	21	69	101

Bit For X, Y and S: 7 6 5 4 3 2 1

115	60	92
102	121	57
76	114	115
25	100	102
50	72	76
101	17	25
74	34	50
21	69	101

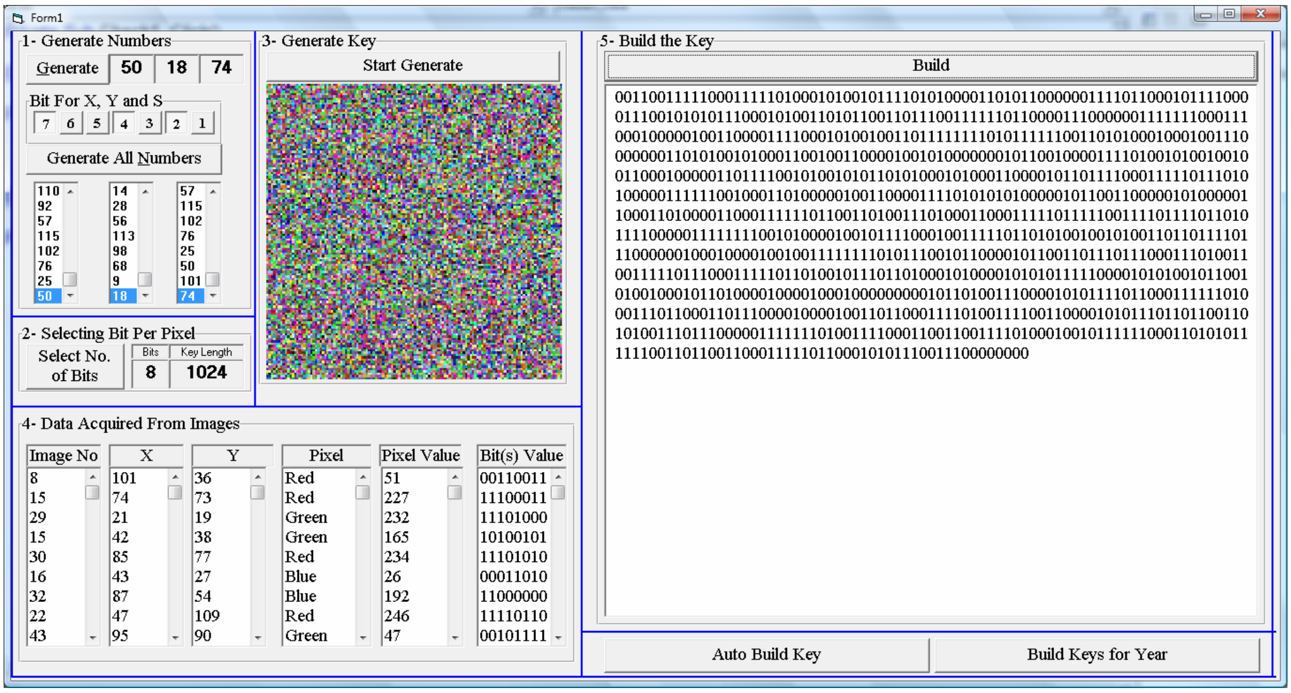
Bits	Key Length
1	128

4- Data Acquired From Images

Image No	X	Y	Pixel	Pixel Value	Bit(s) Value
25	42	11	Blue	230	1
8	85	22	Red	29	0
15	43	44	Red	137	1
29	87	89	Green	47	0
15	47	51	Green	52	0
30	95	103	Red	118	0
16	63	78	Blue	201	1
32	127	29	Blue	172	1
22	126	58	Red	160	1

```
10100011111000001011001011011001101010111011101010100111110011110011010110
0010101111111111111111001100001100101010101000011101100
```

Fig. (8) 128 Bit Generation



110	14	57
92	28	115
57	56	102
115	113	76
102	98	25
76	68	50
25	9	101
50	18	74

4- Data Acquired From Images

Image No	X	Y	Pixel	Pixel Value	Bit(s) Value
8	101	36	Red	51	00110011
15	74	73	Red	227	11100011
29	21	19	Green	232	11101000
15	42	38	Green	165	10100101
30	85	77	Red	234	11101010
16	43	27	Blue	26	00011010
32	87	54	Blue	192	11000000
22	47	109	Red	246	11110110
43	95	90	Green	47	00101111

```
001100111110001111101000101001011110101000011010110000001111011000101111000
011100101010111000101001101011001101110011111101100001110000001111111000111
0001000000100110000111100010100100110111111110101111110011010100010001001110
000000110101001010001100100110000100101000000010110010000111101001010010010
0110001000001101111001010010101101010001010001100001011011110001111101111010
1000001111110010001101000001001100001111010101010000010110011000001010000001
100011010000110001111110110011010011101000110001111011111001111011111011010
1111000001111111100101000010010111100010011111011010100100101001101101111101
1100000010001000010010011111111010111001011000010110011011101110001110100 11
0011111011100011111011010010111011010001010000101010111110000101010010110 01
0100100010110100001000010001000000000010110100111000010101111011000111111010
0011101100011011100001000010011011000111101001111001100001010111011011001 10
1010011101110000011111110100111100011001100111101000100101111111000110101011
1111001101100110001111101100010101110011100000000
```

Fig. (9) 1024 Bit Generation