Selection of sampling keys for cryptographic tests

George Marinakis [1]

Abstract

The keys of modern cryptographic algorithms have an enormous size, so the testing of the algorithm performance for all key combinations, will take practically an infinite time. To avoid this, the sampling method is used, where a much smaller number of keys is tested and then the estimation of the algorithm performance for all the keys is calculated with a predetermined sampling error. For each sampling key, an output sample of the algorithm must be generated and tested. Therefore, in order to have sampling results as close as possible to the real performance of the algorithm, the key question is whether the selection of the keys should be random or it must follow some rules. If the selection of the keys is completely random, there is a high probability that the tests will not find some "weak" or "equivalent" keys, which give non-random or similar outputs and therefore reduce the total number of active keys. But if the sampling keys are selected with some specific criteria, there is a much greater probability of detecting any weak or equivalent key. In this study an optimal key selection methodology is proposed, which combines the random and the non-random key selection.

Keywords: Cryptography, Data encryption, Communication security, Computer security, Data security, Information security.

1. Introduction

When there is a need to evaluate the cryptanalytic strength of a cryptographic algorithm, some specific tests must be performed on its output sequences of bits (keystreams) as well as on selected encrypted texts. These tests investigate the complexity and the non-linearity of the algorithms and are based on specific statistical and cryptanalytic methods, which test the randomness and the uniform distribution in the bits of the keystreams, as well as unwanted similarities between the different keystreams. The tests also check the independence between the clear text and the encrypted text and they generally search for security weaknesses which can be used for cryptanalytic attacks.

It is obvious that due to the very large size of cryptographic keys (usually between 128 to 256 bits), it is practically impossible to test the algorithm for all the possible combinations of its keys (which is 2^{128} to 2^{256} respectively). Therefore, it is necessary to use the sampling method, in which from the total number of the N key values (combinations), a much smaller number of n key values are selected for the testing. Using a software simulation of the algorithm, for each of the n sampling keys, a sample output of the algorithm is generated and subsequently these n samples are submitted to the relevant tests. Finally, the results of the tests are processed for the calculation of the total performance of the algorithm for all the keys, using a predetermined sampling error.

---

[1] George Marinakis holds a MS in Electrical Engineering from University of Patras (Greece) and a PhD in Cryptography from National Technical University of Athens (NTUA). He is a former professor at Telecommunications and Electronics School of Signal Officers (Athens, Greece). He is currently instructor and scientific collaborator at Hellenic Army Academy. He can be reached at gmari@tee.gr.

The final decision on the cryptographic power of the algorithm (randomness, independence, unpredictability of outputs, etc.) is made based on the overall success rate of the tests in the samples. These tests are extremely time consuming, therefore if we want to have a reliable sampling (small sampling error) but also a practically feasible time to perform the checks, the main problems that arise are the following:

a. How many output samples should we check?
b. What should be the size of each sample?
c. How can we reduce the time of the tests?
d. What criteria should we use in order to select the sampling keys?
e. How do we rate the strength of the algorithm based on the test results of its samples?

The problems (a), (b), (c) were addressed in our previous study (Marinakis, 2021) [1] , in which we proposed documented and practical solutions. In the present study we will propose solutions to problem (d). The problem (e) will be addressed in a future study.

The methods which will be proposed are focused on symmetric cryptographic algorithms (block ciphers and stream ciphers), but similar methods can be applied for asymmetric cryptographic algorithms.

1.1. Weak and Equivalent keys

Before we proceed in the description of the study, it is necessary to define the terms weak key and equivalent key , which will be used in the following paragraphs.

In general, weak key is a key, which, used with a specific cipher, makes the cipher behave in an undesirable way. Weak keys usually represent a very small fraction of the overall keyspace, but it is desirable for a cipher to have no weak keys. A cipher with no weak keys is said to have a *flat* or *linear* key space , which means that all the keys are equally strong. In the case of the DES block cipher, according to (Menezes et al., 1997) [2], (Schneier, 1996) [3], weak keys are those for which the encryption of the encrypted text X with the same key K gives the open text:

Weak key $K \rightarrow E_K [E_K (X)] = X$

Also, a semi-weak key pair is a pair of keys for which, if we encrypt with the first key K1 the encrypted text which was encrypted with the second key K2 , we will have the open text:

Semi-weak key pair K1 , K2 $\rightarrow E_{K1} [E_{K2} (X)] = X$

We note that the DES cipher has 4 weak keys and 6 semi-weak key pairs, as described in (NIST S.P. 800-67 Rev.2, 2017) [4].

In the context of the present study, we introduce a different definition for the weak keys and also we introduce the new definition of “equivalent keys” , as follows:

Weak key is any key for which the cryptographic algorithm output is not random.

Equivalent key is any key that gives the same (or similar) cryptographic algorithm output, with the output which is given by another key.

It is obvious that the presence of weak and equivalent keys reduces the nominal number N of the keys in a cryptographic algorithm, which is $N = 2^L$, (where L = the key length in bits). In this case, the number N_A of the active keys is $N_A = N - N_W - N_E$, where N_W is the number of weak keys and N_E is the number of equivalent keys.

From what has been stated in this paragraph, it is essential that the randomness tests on the algorithmic outputs are intended to reveal the weak keys, while the similarity tests between the different algorithmic outputs are intended to reveal the equivalent keys.

2. Selection of the sampling keys

When the output samples of a cryptographic algorithm are tested, an important problem is how to select the sampling keys, in order our results will be as close as possible to the total performance of the algorithm. The key question is whether the selection of the keys should be random or it must follow some rules. If the selection is completely random, then we may not detect some "weak" or "equivalent" keys, which give non-random or similar results and therefore they will reduce the total number of active keys and they will give starting points for cryptanalytic attacks. But if we choose the keys with some specific criteria, there is a much greater possibility of detecting any weak or equivalent key. In the following paragraphs we will propose an optimal key selection methodology, which combines the random and the non-random key selection.

3. Random Key sampling

When the production method of the n sampling keys is completely random, there is a significant possibility that inside the total space of the key values from 1 to N (where $N = 2^L$ the total number of keys and L = key length), some large ranges of key values will not be considered. Figure 1 shows one such example, in which with the use of a software random function or with a hardware Random Number Generator we produced 40 random keys (showed with vertical lines). As is it shown, there are two significant ranges of the key values which are not examined (from n_1 to n_2 and from n_3 to n_4).

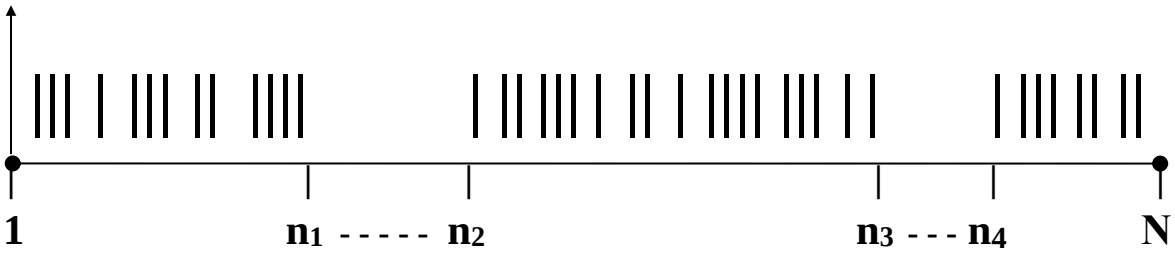

Figure 1. Random key sampling which leaves out two ranges of key values (from n_1 to n_2 and from n_3 to n_4)

4. Stratified random key sampling

In order to avoid the previous problem, we can perform a different method for the production of the random sampling keys. That is, we divide the whole area of the possible key values in smaller equal and non-overlapping areas and within each of them we perform the random sampling. These equal and non-overlapping areas are called *strata*, that's why in the sampling bibliography (Wadsworth, 1990) [5], (Thompson, 2012) [6], this method is called stratified random sampling. Stratified random sampling is used when the population that we want to test is heterogeneous (which is common for large-scale data sets like in our case).

Figure 2 shows such an example, where the 40 random values of the key are stratified (divided) into four equal regions (from 1 to N / 4, from N / 4 to N / 2, from N / 2 to 3N / 4 and from 3N / 4 to N). Each of the four equal regions contains 10 random keys.

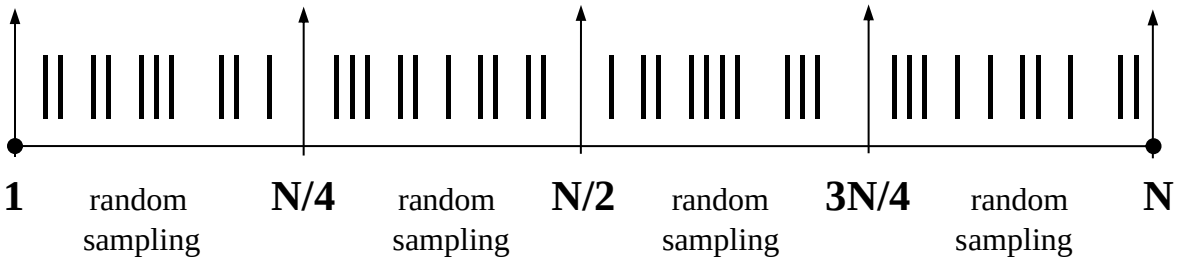

Figure 2. Mixed sampling (stratified) into four equal areas of key values

Comparing the previous two figures, we see that in Figure 2 the distribution of the keys is more homogeneous than the distribution of the keys in Figure 1. This will lead us in a greater probability of detecting any weak or equivalent key. Of course, we can divide the total population of the keys into more than four equal areas. Also, these key value areas may not be equal in bandwidth. But the basic point is to cover as much as possible of the whole key diversity without leaving large empty ranges of keys like these in Figure 1.

4.1 Mixed sampling with random keys and continuous keys

A significant case which combines the stratified random sampling and the non-random sampling key selection, is when we want to examine the behavior of the algorithm output when the keys have continuous values. This is because the outputs of cryptographic algorithms must be very unpredictable, and this can be achieved only if the algorithms are very "sensitive" to small key changes (avalanche effect). This means that for small changes in the value of the key, the algorithm must not only maintain the randomness of its output, but also must have no similarities between its different outputs.

Figure 3 shows a mixed selection of keys, where in each of the four equally stratified areas, in addition to the random keys (dotted vertical lines) we select a group of keys that have continuous values (continuous vertical lines). In this case, except the randomness tests to each algorithm output, we must always compare the algorithm outputs which are produced from the continuous keys, in order to find any similarities between them (search for equivalent keys).

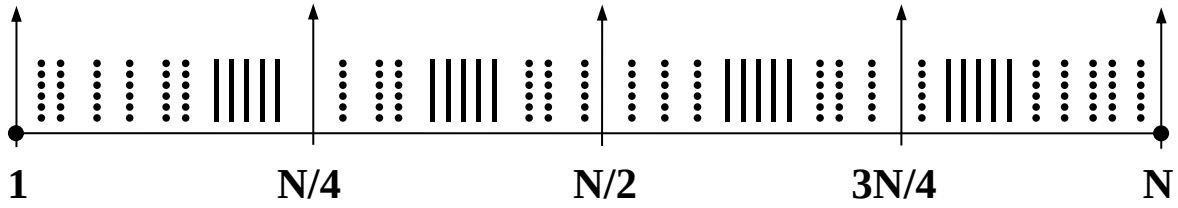

Figure 3. Mixed sampling keys, where **n = x + y** (x random keys and y continuous keys)

Figure 4 shows an example of selecting four keys which have continuous outputs: From an initial key with a value of K_1 (which may be random or specified), we construct three more keys with values $K_2 = K_1 + 1$, $K_3 = K_2 + 1$, and $K_4 = K_3 + 1$. The changes are done in the three least significant bits (showed with grey color).

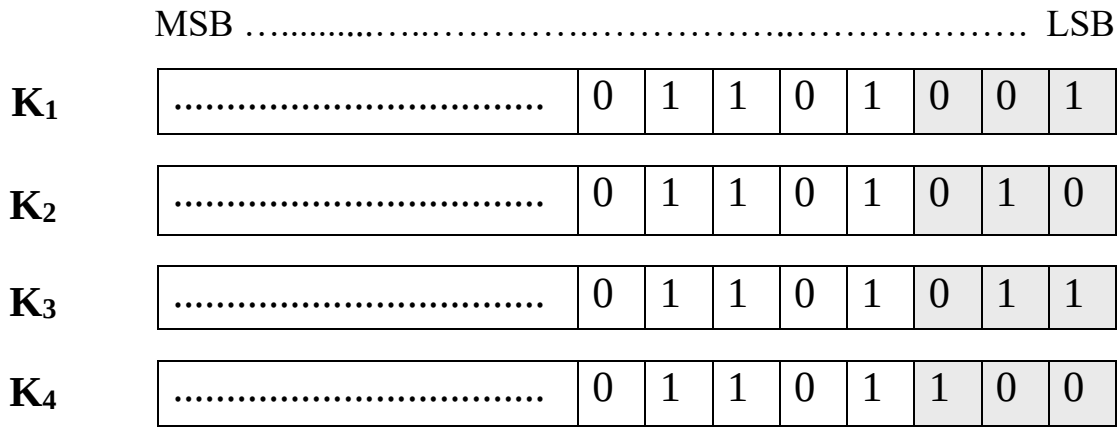

Figure 4. Selection of four continuous keys K_1, K_2, K_3, K_4 (only their eight least significant bits are shown)

4.2 Mixed sampling with random keys and keys with one bit difference

Another significant case of stratified random sampling and non-random sampling key selection, is to include cryptographic keys that differ by only one bit from each other. This option can reveal the existence of equivalent keys in the algorithm and is based on three theoretical properties that all cryptographic algorithms must meet, which according to (Menezes et al., 1997) [2], are :

a. Completeness property:

Each bit of the encrypted text must be depended from all the bits of the key. This property was first formulated by Claude Shannon in 1949 as confusion property.

b. Avalanche effect:

Changing one bit of the algorithm input should affect as many as possible bits of its output. This property was formulated by Claude Shannon as diffusion property, in the sense that the statistical peculiarities of the open text should be "multiplied" within the encrypted text.

c. Strict Avalanche Criterion (SAC):

When one bit of the plain text changes, each bit of the encrypted text must change with probability of 1/2 (i.e. half of its bits must be changed).

We must note that property (a) is valid when we consider the key as the input of the cryptographic algorithm, while the (b) and (c) properties are valid when we consider the plain text as the input of the algorithm. However, the properties (b) and (c) must also apply when we consider the key as input, because the key must equally (if not more) affect the output of the cryptographic algorithm (and therefore the encrypted text).

Figure 5 shows an example of selecting three keys which have one bit difference from an initial key. That is, from the key K_1 (which may be selected randomly or specifically), we construct the three keys K_2, K_3, K_4, changing each time the first, the second and the third bit of the initial K_1 key (changes are shown with gray color). The changes are done to the least significant bits, so that the new key values are within the same stratified area of the original K_1 key. The same process can be repeated many times, in order to produce more keys which will have one bit difference between them (starting every time from an initially selected key).

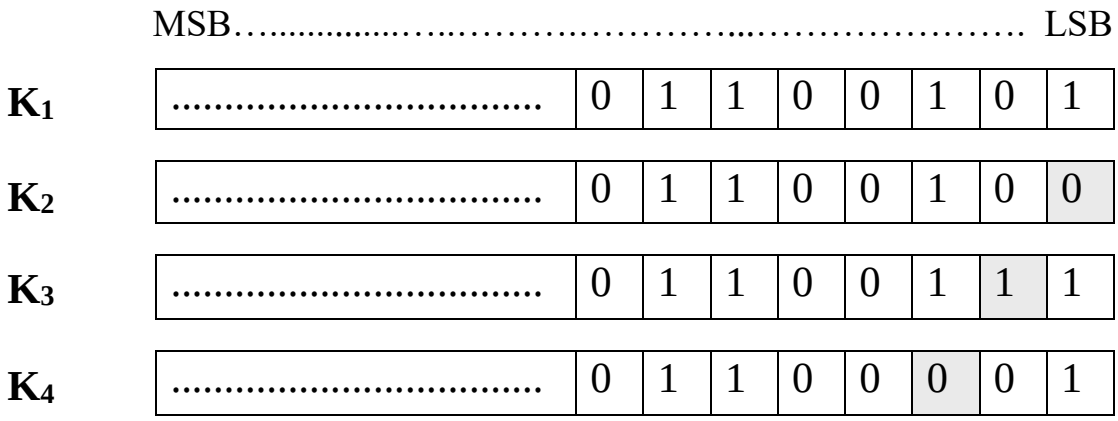

Figure 5. Selection of three keys K_2, K_3, K_4 which have one bit difference from K_1 (only their eight least significant bits are shown)

4.3 Mixed sampling with random keys and complementary keys

Another significant case of stratified random sampling and non-random sampling key selection, is to include cryptographic keys that have complementary bits. This is because the bitwise complementary keys have some special cryptographic properties. For example, the DES block cipher has an undesirable property with complementary keys, because according to (Menezes et al., 1997) [2] :

$$c = DES(p, k) \Rightarrow \overline{c} = DES(\overline{p}, \overline{k})$$

where c is the encrypted data , p is the plain data , k is the key and $\overline{c}$, $\overline{p}$, $\overline{k}$, are their bitwise complementation.

The complementation property of DES normally does not help a cryptanalyst in known plain-text exhaustive key search, because it rules out only two keys (the key variations are decreased from 2^{55} to 2^{54}). However, in our study the undesirable property of complementary keys is relevant with the term of equivalent keys which we introduced in paragraph 1.1 (that is, we are searching for complementary keys which give the same or similar cipher outputs).

In Figure 6 we give an example with two couples of complementary keys K_1 , $\overline{K_1}$ and K_2 , $\overline{K_2}$.

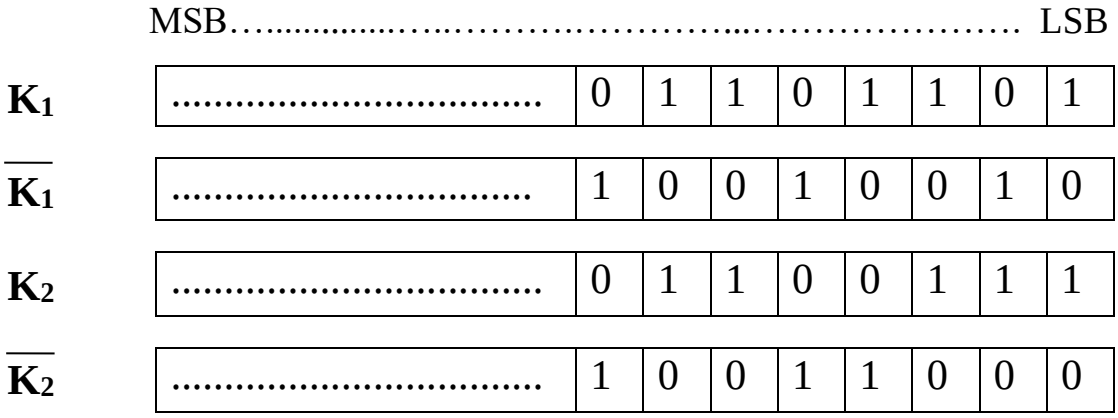

Figure 6. Selection of two couples of complementary keys K_1 , $\overline{K_1}$ and K_2 , $\overline{K_2}$ (only their eight least significant bits are shown)

In practice, when we want to produce a number *y* of complementary keys, the best way is to take as starting point an equal number *y* of random keys (which were generated in a previous step) and calculate their complementary keys. It is obvious that the values of these complementary keys may not belong to the same stratified area with the values of their complementary random keys, but this will not be a problem. We can use them independently and include them to the total number of the non-random sampling keys.

4.4 Keys with special bit patterns

Besides the non-random keys which were described in previous paragraphs, there are some keys with specific bit patterns, which may give undesirable behavior to cryptographic algorithms. For example, as we noted in paragraph 1.1, the DES cipher has 4 weak keys and 6 semi-weak key pairs. These keys have repetitive bit patterns and according to (NIST S.P. 800-67 Rev.2, 2017) [4], are (in hexadecimal format):

• 01010101 01010101 • FEFEFEFE FEFEFEFE • E0E0E0E0 F1F1F1F1 • 1F1F1F1F 0E0E0E0E Weak keys of DES	• 011F011F010E010E and 1F011F010E010E01 • 01E001E001F101F1 and E001E001F101F101 • 01FE01FE01FE01FE and FE01FE01FE01FE01 • 1FE01FE00EF10EF1 and E01FE01FF10EF10E • 1FFE1FFE0EFE0EFE and FE1FFE1FFE0EFE0E • E0FEE0FEF1FEF1FE and FEE0FEE0FEF1FEF1 Semi- weak keys of DES

Obviously, the above keys affect only the behavior of the DES cipher and they do not affect the function of other ciphers. But it has been found that there are many ciphers which have their own weak keys. The fact is that the weak keys are very few compared to the huge number of the active keys, but in general we can say that, if there is any indication that there are some keys with specific bit patterns which may give undesirable behavior to the under-test algorithm, we must include these keys in our tests.

4.5 General remarks for the mixed sampling keys

From the previous paragraphs, it is concluded that when we want to use stratified sampling, we must divide the total number N of the cryptographic keys into S equal and non-overlapping stratified areas. This means that each stratified sampling area will have the same number of keys, which will be N/S. In each of these areas we apply a mixed stratified sampling with x random keys and y non-random sampling keys. Therefore, the total number of the mixed sampling keys in all stratified areas will be:

$$n = Sx + Sy$$

Furthermore, the number of the y non-random keys can be divided in the special categories which were described in the previous paragraphs:

y_1 = continuous keys y_3 = complementary keys
y_2 = one bit difference keys y_4 = special bit pattern keys ($y = y_1 + y_2 + y_3 + y_4$)

5. Production of the stratified mixed sampling keys

Figure 7 shows a flow chart for the production of x random keys and y non-random keys in each of four equally stratified areas. From the previous paragraph we have that, if N is the total number of the cryptographic keys, then the number of keys of each stratified sampling area will be N/4. Also, the total mixed sampling keys will be n = 4x + 4y.

The flow chart has one main loop which contains two internal loops. The main loop concerns the production of the total number of the mixed sampling keys which are produced in the four stratified areas. The two internal loops concern the production of the x random and y non-random keys respectively in each stratified area.

The main loop starts with the setup of the first stratified area of key values, where A is the beginning and B is the end of each area and S is the counter for the stratified areas (from 1 to 4). At the end of the main loop (after the completion of the two internal loops), the value of S is increased by 1, the values of A and B are increased by N/4 and the same process runs for the second stratified area. This process is repeated until S > 4 , where the whole process ends.

First internal loop (random keys): The first internal loop concerns the production of the x random keys in each stratified area (between A and B). In the beginning of the loop we initialize the counter c for the random keys (c=1). For the production of the random keys we must use a Random Number Generator (RNG), which can be implemented in hardware (True Random Number Generator -TRNG) or in software (Pseudo Random Number Generator -PRNG).

There are many hardware TRNG in the market in the form of integrated circuits or in the form of flexible USB modules. They take their random seed from an internal electronic noise source and some of them have integrated randomness tests. More information about True and Pseudo Random Number Generators can be found in (Marinakis, 2015) [7].

Instead of a hardware TRNG, a software PRNG can be used, which can be implemented with the random function which is included in every programming language. In Figure 8 we give an example of a program in C++ language which uses the "random" function in order to create ten random numbers between 1 and 100000 and prints them to the screen and to the file "random.txt". These ten random numbers are different in every run of the program, with the additional use of the "srand" function which sets a different starting point (seed) before the execution of the "rand". The "srand" takes the seed from the computer clock, which gives a different value every time the program runs. After the production of each random key, it is always checked that the generated random key is not the same with an already produced random or non-random key. At the end of the first internal loop, the random keys counter c is increased and if it is greater than x, the program goes to the second internal loop.

Second internal loop (non-random keys): The second internal loop concerns the production of the y non-random keys in each stratified area (between A and B). In the beginning of the loop we initialize the counter c for the non-random keys (c=1). The detailed process of this generator depends from the kind of the y non-random keys that we want to produce (continuous keys, complementary keys, one bit difference keys etc.). In order to have a more exhaustive key sampling, the best is to include in the tests all the above kinds of non-random keys. A good practice is to select some random keys which were produced in each stratified area during the first internal loop, and construct their continuous, complementary and one bit difference keys. After the production of each non-random key, it is always checked that the generated non-random key is not the same with an already produced random or non-random key. At the end of the second internal loop, the non-random keys counter c is increased and if it is greater than y, the program goes to the next stratified keys area.

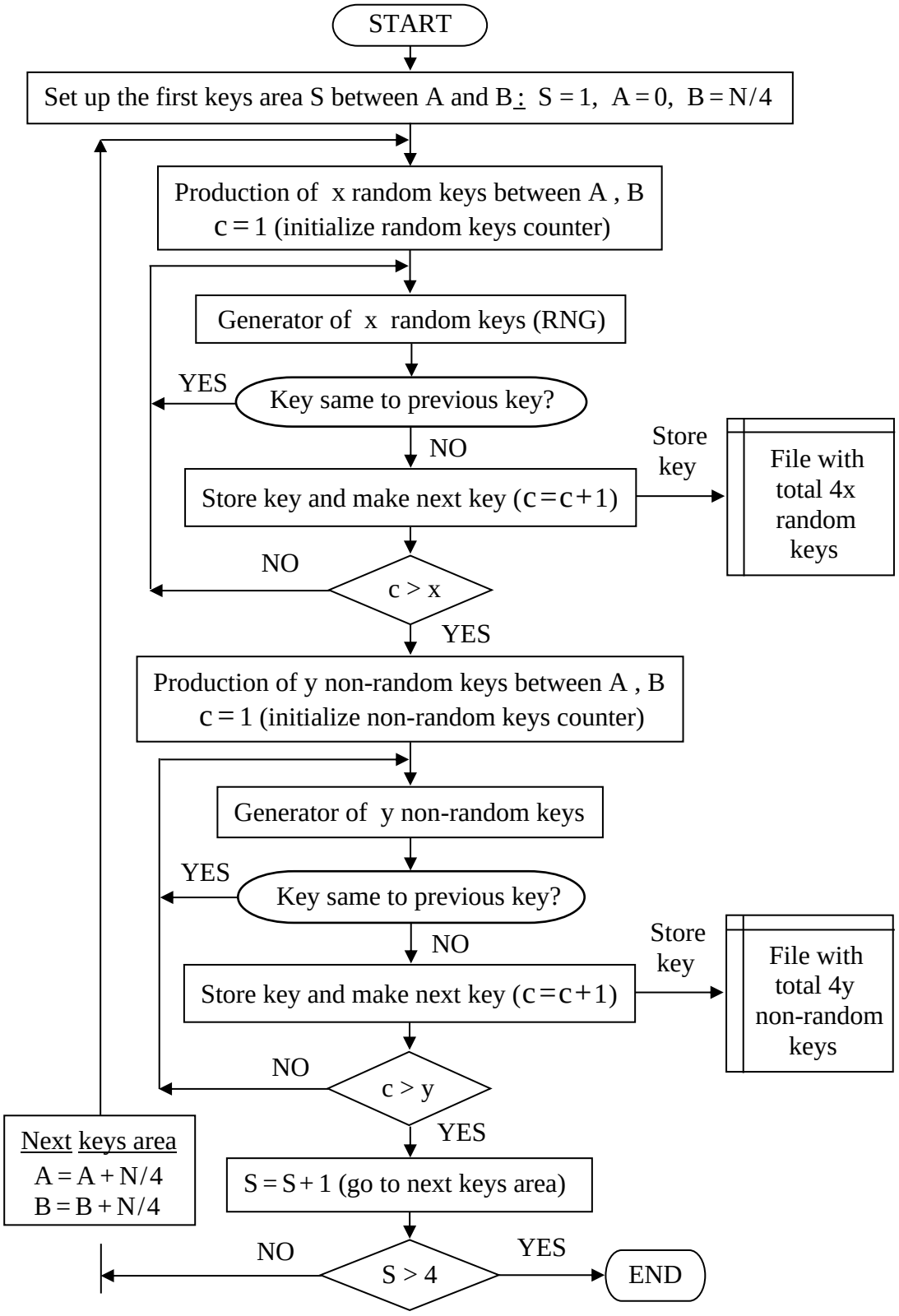

Figure 7. Production of n mixed sampling keys (x = random keys and y = non-random keys in each of four equally stratified areas, N = total number of the keys and n = 4x + 4y).

```
 1  #include <stdio.h>
 2  #include <stdlib.h>
 3  #include <fstream>
 4  #include <iostream>
 5  #include <time.h>
 6  using namespace std;
 7  int main( )
 8  { int c, n;
 9  // open the file in write mode
10   ofstream outfile ("random.txt" , ios::out );
11 // write the header to the file
12   outfile <<"Generate ten random numbers in [1-100000]:\n";
13   outfile << endl; // new line
14 // print the header to the screen
15    printf ("Generate ten random numbers in [1-100000]:\n\n");
16 // create ten different random numbers in every run
17    srand (time(0)); // change the seed using current time
18    for (c = 1; c <= 10; c++)
19    { n = rand( ) % 100000 + 1;
20 // write the random numbers to the screen
21     printf ("%d\n", n);
22 // write the random numbers to the file
23    ofstream outfile ("random.txt" , ios::app );
24    outfile << n << endl; }
25 // Close the file
26    outfile.close( );
27  return 0; }
```

Figure 8. C++ code which creates ten random numbers between 1 and 100000 (which are different in every program run) and prints them in the screen and to file “random.txt”.

We must note that after the production of the n sampling keys, it may be needed to convert their values to the appropriate format in order to feed the software simulation for the production of the algorithm output samples, as we described in paragraph 1 (e.g. conversion from decimal format to binary format or to hexadecimal format).

We close this paragraph with a practical implication. According to (Marinakis, 2021) [1] , if we want to achieve a sampling error of 3% we must test 1067 keys, for sampling error of 2% we will need 2401 keys, while for sampling error of 1% we will need 9604 keys. Therefore, if in the example of Figure 7 we will choose a sampling error of 1% and an equal number of random and non-random sampling keys in each stratified area, we will have n = 9604 and x = y. And since n = 4x + 4y , this means that we will need x = 1200 random sampling keys and y = 1200 non-random sampling keys in each of the four equally stratified areas. Furthermore, the 1200 non-random keys can be divided in 400 continuous keys, 400 one-bit difference keys and 400 complementary keys.

6. Conclusions

When the sampling method is used during the cryptographic algorithms tests, an important problem is how to select the cryptographic keys, in order to have reliable sampling results. The key question is whether the choice of the sampling keys should be random or it must follow some rules. If the choice is completely random, then we may not detect some "weak" or "equivalent" keys, which will give non-random or similar algorithm outputs and therefore they will reduce the total number of active keys and they will give starting points for cryptanalytic attacks. But if the keys are chosen with some specific criteria, there is a much greater chance of detecting any weak or equivalent key.

In this study, we proposed a mixed sampling method, which uses the combination of random and non-random stratified sampling for the production of the cryptographic keys. Initially, in each stratified area we generate an equal number of random keys. In addition to the random keys, in each stratified area we select a number of continuous keys, complementary keys and keys with one bit difference. With this method, which combines the stratified random sampling and the stratified non-random sampling for the production of the test keys, there is a greater probability to find any anomalies and weaknesses of the cryptographic algorithm which is under test.

References

[1]. George Marinakis, "Sampling methods for cryptographic tests" , May 14, 2021. https://www.scienpress.com/journal_focus.asp?main_id=57&Sub_id=IV&Issue=2143151

[2]. Alfred Menezes, Paul C. van Oorschot, Scott A.Vanstone, "Handbook of Applied Cryptography", CRC Press 1997.

[3]. Bruce Schneier, "Applied Cryptography", John Wiley, New York, 1996.

[4]. NIST Special Publication 800-67 Revision 2, "Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher" https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-67r2.pdf

[5]. Harrison M. Wadsworth, "Handbook of Statistical Methods for Engineers and Scientists", Mc Graw-Hill, 1990.

[6]. Steven K. Thompson, "Sampling" , Wiley, 3rd edition, 2012.

[7]. George Marinakis, "Design and evaluation of random number generators" Sept.15, 2015. http://www.scienpress.com/journal_focus.asp?main_id=57&Sub_id=IV&Issue=1608