Chapter 6. Results, Conclusion, And Future Work

6.1. The Key Idea

All the time during this research, we were always seeking an evaluation technique that satisfies all the following needs:

1. The need to have a standard technique
2. The need to have a straight forward technique
3. The need to have a reliable

Due to the lack of analytical techniques for evaluating encryption algorithm, the evaluation process becomes more difficult and terribly confusing. Analytical evaluation technique is algorithm dependent and hard to find.

We got the key idea to our evaluation method from that fact that One-Time Pad is the only encryption algorithm that is proved to be secure. We found that the extensive need for encryption key was the only existing problem of One-Time Pad, and was the motivation of inventing new encryption algorithms. Therefore, we can evaluate any encryption algorithm by comparing it to the One-Time Pad algorithm. The more similar they are, the more powerful this algorithm is.

The main difference between any encryption algorithm and One-Time Pad is that the One-Time Pad uses a random key of the same length as the message. While other encryption algorithms use a key-stream of the same length as the encrypted message. The stream is generated from a pretty shorter random key rather than being random itself. That is the point. The more random is that stream, the more similar to the One-Time Pad that algorithm is, and consequently, the more secure it is. This way, we transformed the evaluation of the security of any encryption algorithm into a process of randomness test. We do not need to deeply study and think about the encryption algorithm any more. We just need to construct a key-stream generator from that algorithm and use a random key that is generated from a

good RNG like BBS. Then feed this key to the key-stream generator and generate as long stream as we need. Then test the randomness of that stream. The more random the stream appears, the more secure is that algorithm. We need not to know how many rounds is the algorithm, how can it resist Linear of Differential attacks. All these issues can be ignored.

Various statistical tests can be applied to a sequence to attempt to compare and evaluate the sequence to truly random sequence. Where randomness is probabilistic property, that is, the properties of random sequence can be characterized and described in terms of probability. The likely outcome of statistical tests, when applied to a truly random sequence and can be described in probabilistic terms. There are an infinite number of possible statistical tests, each assessing the presence or absence of '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 test 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.

In this chapter, we will apply a collection of the evaluation criteria introduced earlier on both RC4 as a mode of stream cipher and RC6 as a mode of block cipher using NIST statistical test suite. A statistical test is formulated to test a specific null hypothesis (H_0). For the purpose of this document, the null hypothesis under test is that the sequence being tested is random. Associated with this null hypothesis is the alternative hypothesis (H_a) which, for this document, is that the sequence is not random. For each applied test, a decision or conclusion is derived that accepts or rejects the null hypothesis. i.e., whether the generator is (or is not) producing random values, based on the sequence that was produced.

The test statistic is used to calculate a P-value that summarizes the strength of the evidence against the null hypothesis. For these tests, each P-value is the probability that a perfect random number generator would have produced a sequence less random than the sequence that was tested, given the kind of non-randomness assessed by the test. If a P-value for a test is determined to

be equal to 1, then the sequence appears to have perfect randomness. A P-value of zero indicates that the sequence appears to be completely nonrandom. A significance level (α) can be chosen for the tests. If P-value >= α, then the null hypothesis is accepted; i.e., the sequence appears to be random. If P-value < α, then the null hypothesis is rejected; i.e., the sequence appears to be non-random. The parameter α denotes the probability of the Type I error. Typically, α is chosen in the range [0.001, 0.01].

6.2. NIST Tests

The NIST (National Institute of Standards and Technology, Gaithersburg, MD, U.S.A.) statistical test suite (NIST-STS in the following) is a comprehensive battery of statistical test for the evaluation of the randomness of pseudo random number generators (PRNGs). The NIST-STS consists of 16 core statistical tests. Each of them yields a P-value which is the probability of obtaining a test statistic as large as or larger than the observed if the sequence is random. In particular, the NIST-STS is fixed to a significance value, α, such that for each test a P-value less than 0.01 is interpreted as evidence that a sequence is unlikely to be random. Table 13 lists the core tests. The majority of tests in the test suite either (1) examine the distribution of zeroes and ones in some fashion, (2) study the harmonics of the bit stream utilizing spectral methods, or (3) attempt to detect patterns via some generalized pattern matching technique on the basis of probability theory or information theory. In what follows, we will describe the most important NIST tests:

6.2.1. Frequency 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. The test assesses the closeness of the fraction of ones to 1/2 that is, the number of ones and zeroes in a sequence should be about the same. All subsequent tests depend on the passing of this test; there is no evidence to indicate that the tested sequence is non-random.

6.2.2. Frequency test within a block

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. For block size M=1, this test degenerates to test 1, the frequency test.

6.2.3. 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 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.

6.2.4. Rank test

The focus of the test is the rank of disjoint sub- matrices of the entire sequence. The purpose of this test is to check for the linear dependence among fixed length substrings for the original sequence. Note that this test also appears in the DIEHARD battery of tests.

6.2.5. Discrete Fourier transform 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 from 5%.

6.2.6. Linear complexity test

The focus of this test is the length of a linear feedback shift register (LFSR). The purpose of this test is to determine whether the sequence is complex

enough to be considered random. Random sequence is characterized by longer LFSRs. An LFSR that is too short implies non-randomness.

6.2.7. Serial test

The purpose of this test is to determine whether the number of occurrences of m-bit overlapping patterns is approximately the same as would be expected for a random sequence.

6.2.8. Approximate entropy test

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 normally distributed sequence.

6.2.9. Cumulative sums (CuSum) test

This test is based on the maximum absolute value of the partial sums of the sequence represented in the ones fashion. Large value of this statistic indicates that there are either too many ones or too many zeros at the early stages of the sequence. Small values indicate that ones and zeros are intermixed too evenly.

6.2.10. Universal statistical test

The test is claimed to measure the actual cryptographic significance of a defect because it is " related to the running time (an) enemy's optimal key-search strategy,' or the effective key size of a cipher system. The test is not designed to detect a very specific pattern or type of statistical defect. A generator should pass the test if and only if its output sequence cannot be compressed significantly.

6.3. Sample of our implementation

In this section, a sample of the Visual Basic implementation of the NIST statistical tests is presented. In this implementation, we used a well-known mathematical library called Cephes Math Library Release 2.8.

```
'*************************************
'        C U M U L A T I V E  S U M S  T E S T
'*************************************
' mode : A switch for applying the test either forward through the input sequence (mode = 0) or backward through the sequence (mode = 1)
' n    : The length of the bit string.
Public Function CumulativeSums(BitStream() As Byte, mode As Long, n As Long, Optional ByRef oMaxPartialSum As Double) As Double
   Dim i As Long
   Dim k As Long
   Dim start As Long
   Dim finish As Long
   Dim p_value As Double
   Dim CuSum As Double
   Dim z As Double
   Dim sum As Double
   Dim Sum1 As Double
   Dim Sum2 As Double
   sum = 0#
   CuSum = 1
   If (mode = 0) Then
      For i = 0 To n - 1
         sum = sum + 2 * BitStream(i) - 1
         CuSum = MaxReal(CuSum, Abs(sum))
      Next
   ElseIf (mode = 1) Then
      For i = n - 1 To 0 Step -1
         sum = sum + 2 * BitStream(i) - 1
         CuSum = MaxReal(CuSum, Abs(sum))
      Next
   End If
   z = CuSum
   oMaxPartialSum = CuSum
   Sum1 = 0#
   start = (-n / z + 1) / 4
   finish = (n / z - 1) / 4
   For k = start To finish - 1
      Sum1 = Sum1 + (NormalDistribution((4 * k + 1) * z / Sqr(n)) - NormalDistribution((4 * k - 1) * z / Sqr(n)))
   Next k
   Sum2 = 0#
   start = (-n / z - 3) / 4
   finish = (n / z - 1) / 4
   For k = start To finish - 1
      Sum2 = Sum2 + (NormalDistribution((4 * k + 3) * z / Sqr(n)) - NormalDistribution((4 * k + 1) * z / Sqr(n)))
   Next k
   p_value = 1# - Sum1 + Sum2
   CumulativeSums = p_value
End Function
```

```
'* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
'                        R U N S  T E S T
'* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
' n    : The length of the bit string.
Public Function Runs(BitStream() As Byte, n As Long, Optional ByRef oTotalNumberOfRuns As Long, Optional ByRef oPreTestProportionOfOnes As Double, Optional oArgument As Double) As Double
    Dim i As Long
    'Dim state As Long
    Dim r() As Byte
    Dim Argument As Double
    Dim Pi As Double
    Dim V_n_obs As Double
    Dim tau As Double
    Dim p_value As Double
    Dim product As Double
    Dim sum As Double
    ReDim r(n) As Byte
    sum = 0#
    For i = 0 To n - 1
        sum = sum + BitStream(i)
    Next i
    Pi = sum / n
    tau = 2# / Sqr(n)
    If (Abs(Pi - 0.5) < tau) Then
        For i = 0 To n - 2
            If (BitStream(i) = BitStream(i + 1)) Then
                r(i) = 0
            Else
                r(i) = 1
            End If
        Next i
        V_n_obs = 0
        For i = 0 To n - 2
            V_n_obs = V_n_obs + r(i)
        Next i
        V_n_obs = V_n_obs + 1
        product = Pi * (1# - Pi)
        Argument = Abs(V_n_obs - 2# * n * product) / (2# * Sqr(2# * n) * product)
        p_value = ErfC(Argument)
    Else
        'PI ESTIMATOR CRITERIA NOT MET!
        'REJECTION
        p_value = 0#
    End If
    oPreTestProportionOfOnes = Pi
    oTotalNumberOfRuns = V_n_obs
    oArgument = Argument
    Runs = p_value
End Function
```

```
'* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
'                B L O C K  F R E Q U E N C Y  T E S T
'* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
' m    : The length of each block.
' n    : The length of the bit string.
Public Function BlockFrequency(BitStream() As Byte, m As Long, n As Long,
Optional ByRef oChi2 As Double) As Double
    Dim i As Long
    Dim j As Long
    Dim NN As Long
    'Dim State As Long

    Dim BlockSum As Double
    Dim arg1 As Double
    Dim Arg2 As Double
    Dim p_value As Double

    Dim sum As Double
    Dim Pi As Double
    Dim v As Double
    Dim chi_squared As Double

    NN = Fix(n / m)                          ' # OF SUBSTRING BLOCKS
    sum = 0#
    For i = 0 To NN - 1                                  ' NN=10000 FOR EACH
SUBSTRING BLOCK   */
        Pi = 0#                              ' This step seems useless
        BlockSum = 0#
        For j = 0 To m - 1                           ' m=100 COMPUTE The "i"th Pi
Value */
            BlockSum = BlockSum + BitStream(j + i * m)
        Next j
        Pi = BlockSum / m
        v = Pi - 0.5
        sum = sum + v * v
    Next i
    chi_squared = 4# * m * sum
    oChi2 = chi_squared
    arg1 = NN / 2#
    Arg2 = chi_squared / 2#

    p_value = igamc(arg1, Arg2)
    BlockFrequency = p_value

End Function
```

```
'**************************************
'                 F R E Q U E N C Y  T E S T
'**************************************
' n    : The length of the bit string.
Public Function Frequency(BitStream() As Byte, n As Long, Optional ByRef oNthPartialSum As Double) As Double
   Dim i As Long
   'Dim State As Long
   Dim F As Double
   Dim s_obs As Double
   Dim p_value As Double
   Dim sum As Double
   Const Sqrt2 As Double = 1.4142135623731
   sum = 0#
   For i = 0 To n - 1
      sum = sum + 2 * BitStream(i) - 1
   Next i
   oNthPartialSum = sum
   s_obs = Abs(sum) / Sqr(n)
   F = s_obs / Sqrt2
   p_value = ErfC(F)
   Frequency = p_value
End Function
```

6.4. RC4 stream cipher

To evaluate RC4, we generated a data set D using the RC4 key-stream generation algorithm. Then D is subjected to a collection of the NIST statistical tests. In order to be accurate and fair, we used a 128-bit key K generated by Blum-Blum-Shub pseudorandom number generator, which is proved cryptographically secure. The data set D is so a bit stream of length 1000000 bit using the key K as a seed. The code that is used for generating the key-stream is shown below:

```
Public Function GetKeyStream(Key() As Byte, n As Integer) As Byte()
   Dim I As Byte : Dim J As Byte : Dim K As Byte : Dim nn As Integer
   Dim KeyStream() As Byte:  ReDim KeyStream(n) As Byte
   Dim S(0 To 255) As Byte : Dim key_length As Byte
   key_length = UBound(K)
   'Key Setup Phase:
   For I = 0 To 255
      S [I] = I
   Next I
```

```
    J = 0
    For I = 0 To 255
        J = (J + S (I) + Key(I Mod key_length)) Mod 256
        Dim temp As Byte
        'Swap S [I] with S [J]
        temp = S(I): S(I) = S(J): S(J) = temp
    Next I
    'Key-stream Generation Phase:
    I = 0 : J = 0
    For nn = 1 To n
        I = (I + 1) Mod 256
        J = (J + S(I)) Mod 256
        'Swap S [I] with S [J]
        temp = S(I): S(I) = S(J): S(J) = temp
        K = (S(I) + S(J)) Mod 256
        KeyStream(nn) = S(K)
    Next K
    GetKey-stream = KeyStream
End Function
```

Then we applied the NIST test battery to the generated stream. The table below gives the p-values of different NIST tests:

Table 12. RC4 Test Results

Test	P-Value	Result
Frequency	0.145393	Success
Block Frequency	0.873736	Success
Cumulative Sums - forward	0.238935	Success
Cumulative Sums - Reverse	0.261565	Success
Runs	0.293662	Success
Longest Runs	0.383199	Success
Rank	0.193190	Success
Discrete Fourier Transform	0.114478	Success
Universal Statistical	0.162533	Success
Approximate Entropy	0.227031	Success
Serial1	0.265247	Success
Serial2	0.287037	Success
Linear Complexity	0.140225	Success

From the tabulated results, it is clear that P-Value is very far away from the 0.01 value. In addition, it can be clearly seen that the values of the parameters -frequency, block frequency and linear complexity- is far away from the critical values, and so satisfies the requirement for randomness. Where there are no strong indications of statistical weakness in the RC4 key-stream, which mean that RC4 -which is a mode of a stream cipher family- is secure and could be used safely in a small applications with restricted memory, where the most significant attacks on RC4 have been based on exploiting the simplicity of initialization algorithms.

Indeed, it is found that relying on the results obtained from single sample can be misleading. Therefore, 100 samples were generated in the same way and the NIST battery was applied on the 100 generated streams. The next table shows the average, best, and worst cases:

Table 13. RC4 Detailed Test Results

Test	Best Case	Ave P-Value	Worst Case
Frequency	0.748359	0.145393	0.077556
Block Frequency	0.946908	0.873736	0.253001
Cumulative Sums – forward	0.468737	0.238935	0.185113
Cumulative Sums – Reverse	0.271916	0.261565	0.198979
Runs	0.868967	0.293662	0.208217
Longest Runs	0.411173	0.383199	0.158657
Rank	0.889160	0.19319	0.152713
Discrete Fourier Transform	0.445252	0.114478	0.110122
Universal Statistical	0.892340	0.162533	0.009140
Approximate Entropy	0.961009	0.227031	0.082644
Serial1	0.650896	0.265247	0.203474
Serial2	0.325184	0.287037	0.170057
Linear Complexity	0.543202	0.140225	0.041810

6.5. RC6 block cipher

To evaluate RC6, we generated a data set D using the RC6 encryption algorithm. Then D is subjected to a collection of the NIST statistical tests. In order to be accurate and fair, we used 64 keys 128-bit each generated by Blum-Blum-Shub pseudorandom number generator. The data set is constructed as follows. For each key, the all-zero plaintext is encrypted with

one of the 128 perturbed keys where a perturbed key is the base key with one of the 128 bits flipped. The resultant ciphertext for the perturbed key is then XORed with the ciphertext that results from encryption using the base key. Thus, each of 64 keys yields 128 perturbed keys with which 128 bits. This gives a stream of 64*128*128 = 1048576 bits, which we truncated to 1000000 bits. The code that is used for encrypting all-zero plaintext is shown below:

```
Public Function GetKeyStream(S() As Long, r As Integer, w As Integer, N As Integer) As Byte()
    Dim I As Integer
    Dim A As Long =0
    Dim B As Long =0
    Dim C As Long =0
    Dim D As Long =0
    B = B + S (0)
    D = D + S (1)
    For I = 1 To r
        T = Rotate ((B - (2 * B + 1)), Log (w))
        u = Rotate ((D - (2 * D + 1)), Log (w))
        A = Rotate ((A - T), u) + S (2 * I)
        C = Rotate((C - u), T) + S(2 * I + 1)
        A = B
        B = C
        C = D
        D = A
    Next I
    A = A + S(2 * r + 2)
    C = C + S(2 * r + 3)
    GetKey-stream = A & B & C & D
End Function
Private Function Rotate(N As Long, B As Integer) As Long
    Dim T As Long
    T = N Mod (2 ^ B) 'T=1110 mod 100=10
    N = N \ (2 ^ B) 'N=1110\100=11
    T = T * 2 ^ (32 - B) 'T=10*100=1000
    N = N + T 'N=11+1000=1011
    Rotate = N '1011
End Function
```

Table 14. RC6 Test Results

Test	P-Value	Result
Frequency	0.913996	Success
Block Frequency	0.925328	Success
Cumulative Sums – forward	0.464274	Success
Cumulative Sums – Reverse	0.552714	Success
Runs	0.583684	Success
Longest Runs	0.738422	Success
Rank	0.742435	Success
Discrete Fourier Transform	0.132333	Success
Universal Statistical	0.134172	Success
Approximate Entropy	0.611241	Success
Serial1	0.982470	Success
Serial2	0.907410	Success
Linear Complexity	0.702873	Success

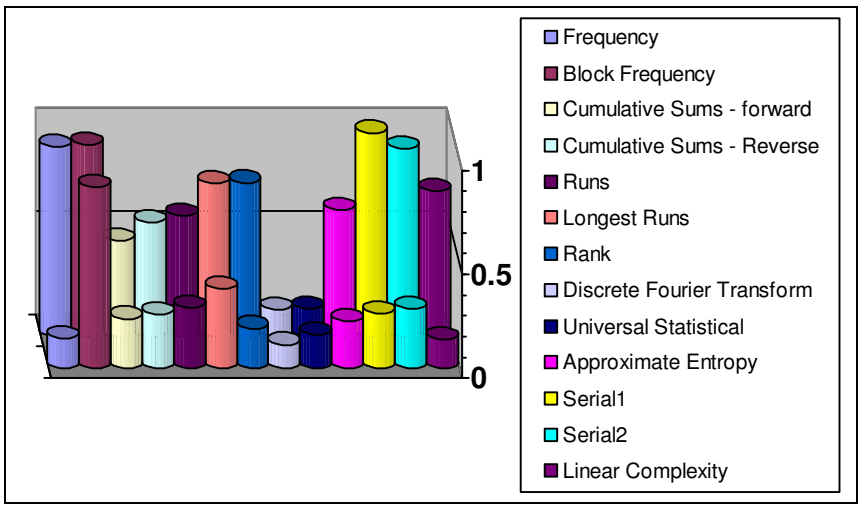
From the tabulated results, it is clear that P-Value is very far away from the 0.01 value and better than RC4. Where the values recorded for frequency, block frequency and linear complexity are much better than in case of rc4 stream cipher, which mean that rc6 block cipher are statistically strong enough, where testing has been performed on rc6 by NIST has gone on to confirm that the results for certain statistical tests on rc6 were reasonable. The results presented in this note confirm that view. Again, we found that relying on the results obtained from single sample could be misleading. Therefore, we generated 100 samples the same way and applied the NIST battery on the 100 generated streams. See the next table.

Table 15. RC6 Detailed Test Results

Test	Best Case	Ave P-Value	Worst Case
Frequency	0.967550	0.913996	0.592106
Block Frequency	0.945026	0.925328	0.258483
Cumulative Sums – forward	0.908820	0.464274	0.382841
Cumulative Sums – Reverse	0.816238	0.552714	0.545027
Runs	0.962933	0.583684	0.132418
Longest Runs	0.920249	0.738422	0.723656
Rank	0.805263	0.742435	0.396366
Discrete Fourier Transform	0.224626	0.132333	0.132256
Universal Statistical	0.719624	0.134172	0.002107
Approximate Entropy	0.834849	0.611241	0.061156
Serial1	0.984276	0.98247	0.784880
Serial2	0.933750	0.90741	0.041423
Linear Complexity	0.790755	0.702873	0.268505

For that results stream ciphers are not generally considered as secure as block ciphers. They are attacked through analyzing the random bit generator. On the plus side, stream ciphers do tend to be the fastest ciphers. For that fact, a large number of applications, which are initially based on stream ciphers, have shifted to block ciphers.

Rc6 is a secure, compact and simple block cipher. It offers good performance and considerable flexibility. Furthermore, its simplicity will allow analysts to quickly refine and improve our estimates of its security. Additional (and extensive) statistical testing has been performed on rc6. NIST has gone on to confirm that the results for certain statistical tests on rc6 were reasonable. The results presented in this thesis confirm that view.



Figuer 28. RC6 versus RC4

6.6. Conclusion and Future Work

Throughout this thesis, we established a coherent collection of criteria that is very adequate for the process of evaluating or comparing various symmetric key encryption algorithms. The NIST statistical test suit proved to be a powerful practical tool for evaluating random sequences to ensure the security of corresponding encryption algorithms. In the future, the same evaluation techniques can be applied to other known stream and block cipher algorithms. Moreover, other test suits like Diehard battery and ENT can be considered besides the NIST. Or rather, a more generalized test suit can be created.