Randomness Analysis of 128 bits Blowfish Block Cipher on ECB and CBC Modes

Ashwak ALabaichi, Ramlan Mahmod, Faudziah Ahmad

Abstract

Randomness of output is a significant factor in measuring the security of any cryptographic algorithm. A non-random block cipher is vulnerable to any type of attack. This paper presents the National Institute of Standard and Technology (NIST) statistical tests of the 128-bit BA to investigate its randomness. The structure of this algorithm resembles that of the 64-bit Blowfish algorithm, except for the block size is 128 bits in the extended version. Moreover, all operations in the 64-bit Blowfish are based on 64 bits instead of 32 bits. The 128-bit BA is a symmetric block cipher with variable key lengths from 64 bits up to a maximum of 896 bits, which enhances the security of the algorithm. The 128-bit BA with the electronic codebook (ECB) and cipher block chaining (CBC) modes is employed for these tests. In addition, comparisons are conducted. The analysis shows that the 128-bit BA with ECB mode is inappropriate for text and image files that contain large strings of identical bytes. This inconsistency in the results of the text and image files is attributed to the fact that the majority of the 188 NIST statistical tests failed in all rounds.

Keywords— Block Cipher, Blowfish 128 bits, ECB mode, CBC mode, Randomness Test.

1. Introduction

The Blowfish Algorithm (BA) was designed by Schneier at the Cambridge Security Workshop in December 1994 to replace the Data Encryption Standard (DES). This algorithm has been widely analyzed and gradually accepted as a powerful encryption algorithm offering several advantages, including its suitability for and efficiency in implementing hardware. The BA is also unpatented and therefore does not require any license. The elementary operators of the BA comprise table lookup, addition, and the exclusive-or-operation (XOR), with the table being made up of four S-boxes and a P-array. Based on Feistel rounds, the BA is a cipher with an F-function design. This design is a simplified version of the principles employed in DES to provide similar security, faster speed, and higher efficiency in software.

Effective cryptanalysis is not presented because of its good encryption rate in software [1–4]. However, the efficiency of this algorithm has resulted in its growing popularity in the open-source community [5].

Alabaichi, Mahmood, Ahmad in [6] as well as Alabaichi, Mahmood, Ahmad, and Mechee in [7], uncovered an issue with Blowfish. The issue lies in the compatibility of the algorithm with image and text files that involve large strings of identical bytes, in particular, problems related to randomness of the output with encrypted text and image files.

Nechvatal et al. stated in [8] that 128-bit input is the minimum requirement for block size. For information that needs to be secure for only minutes, hours, or perhaps weeks, a 64-bit symmetric key will suffice. For data that need to be secure for years or decades, a 128-bit key should be used. For data that need to remain secure for the foreseeable future, a 160-bit key may be preferable [3]. The strength
of symmetric key encryption depends on the size of key used. For the same algorithm, encryption using a longer key is harder to break than one using a shorter key [9].

In this paper, we attempted to strengthen the security of the 64-bit BA by increasing both block size and key length. The 128-bit BA increased the security of the original 64-bit BA by increasing the key space by up to 112 bytes instead of 56 bytes, thus increasing the complexity of brute force attack as well as the block size to 128 bits.

The block cipher requires the generated cipher text to be distributed uniformly when dissimilar plaintext blocks are used during encryption. By statistically analyzing the block cipher, we can determine whether the tested algorithm meets this requirement. A non-random block cipher can be susceptible to many types of attacks [10].

The test suite in [11] from NIST was selected to test the 128-bit Blowfish-generated sequences. These statistical tests are consistent in estimating the generators of random and pseudo-random numbers utilized in cryptographic applications. This attempt is considered an initial analysis of 128-bit Blowfish, considering that no researcher has performed statistical tests on the 128-bit BA with electronic codebook (ECB) and cipher block chaining (CBC) modes.

The remainder of this paper is organized as follows: Section 2 describes the 64-bit and 12-bit Blowfish algorithms, as well as the ECB and CBC modes. Section 3 categorizes and explains 128-bit Blowfish data types for statistical tests. Section 4 presents the results of the experiment and empirical analysis on the randomness testing using 128-bit Blowfish with ECB and CBC modes. Finally, Section 5 presents the conclusion and recommendations for future work.

2. Blowfish Block Cipher

Blowfish 64 is a symmetric block cipher that uses the Feistel network, iterating simple encryption and decryption functions for 16 times. Each Feistel structure offers various advantages, particularly in terms of hardware. In the decryption process of the cipher text, the only requirement is to reverse the key schedule. The BA can be divided into key expansion and data encryption [1],[5],[12].

The key expansion of BA begins with the P-array and S-boxes, with the utilization of many sub-keys. This process requires pre-computation before data encryption or decryption. The P-array comprises eighteen 32-bit sub-keys: P1, P2… P18.

In this section, a maximum key of 448 bits is converted into several sub-key arrays of up to a total of 4168 bytes.

Each of the four 32-bit S-boxes has 256 entries as follows:

\[
S_{1,0}, S_{1,1},..., S_{1,255} \\
S_{2,0}, S_{2,1},..., S_{2,255} \\
S_{3,0}, S_{3,1},..., S_{3,255} \\
S_{4,0}, S_{4,1},..., S_{4,255}
\]

These subkeys are calculated using the following procedure:

1. First, the P-array, followed by the four S-boxes, is initialized with a fixed string, which has the hexadecimal digits of pi.
2. XOR \( P_1 \) with the first 32 bits of the key, XOR \( P_2 \) with the second 32 bits, and so on, until the key bits reach \( P_{14} \). The cycle is iterated through the key bits until the entire P-array has been XOR’ed with key bits.
3. The BA is then used to encrypt the all-zero string, employing the described subkeys in steps 1 and 2.
4. \( P_1 \) and \( P_2 \) are replaced with the step 3 output.
5. The step 3 output is encrypted with the BA using the modified subkeys.
6. \( P_1 \) and \( P_2 \) are replaced with the output of step 5.
7. The process is continued, and all elements of the P-array are replaced, followed by all four S-Boxes, with continuously changing output.

Data encryption commences with a 64-bit block element of plaintext morphing into a 64-bit ciphertext. First, the 64-bit segment is split into two equal segments that form the base of the BA. The next step is the implementation of the XOR that is performed between the first segment of the 32-bit block (L) and the first P-array. The 32-bit data obtained from step 2 are moved to the F function, which permutes the data into a 32-bit block segment, which is XOR’ed with the second segment of the 32-bit
block (R) of the 64-bit plaintext split. Upon completion of the XOR operation, the 32-bit segments, L, and R are exchanged for future iterations of BA. Figure 1 illustrates the architecture of the BA with 16 rounds. The input is an element of 64-bit data, X, which is divided into two 32-bit halves: XL and XR. Data decryption is similar to encryption data, but P1, P2... P18 are used in reverse order.

![Blowfish Architecture](image)

**Figure 1. Blowfish architecture**

The F function of BA is probably the most complex part of the algorithm because it is the only part that utilizes the S-boxes. This function accepts a 32-bit stream of data and splits the data into four equal parts. Each 8-bit subdivision is changed into a 32-bit data stream using the corresponding subdivision S-box. The 32-bit data obtained are XOR’ed or combined to obtain a final 32-bit value for permutations of the BA (notably, all the additions are modulo $2^{32}$). Figure 2 describes the architecture of the F function [5],[13–15].

![F-function Architecture](image)

**Figure 2. F-function architecture**

The 128-bit Blowfish has the same outer structure as that of the 64-bit Blowfish, that is, a Feistel network. This network iterates simple encryption and decryption functions for 16 times. Each operation in 128-bit Blowfish is converted into 64 bit instead of 32 bit. In 128-bit Blowfish, the block size is 128 bits with a variable key of up to 112 bytes instead of 56 bytes. In the expansion part, the maximum key is 896 bits converted to the several subkey arrays up to a total of 8336 bytes instead of 4168 bytes.

Each of the four 64-bit S-boxes has 65536 entries as follows:

- $S_{1,0}$, $S_{1,1}$, ..., $S_{1,65535}$
- $S_{2,0}$, $S_{2,1}$, ..., $S_{2,65535}$
- $S_{3,0}$, $S_{3,1}$, ..., $S_{3,65535}$
- $S_{4,0}$, $S_{4,1}$, ..., $S_{4,65535}$

To reduce the risks of clever attack on the cipher, consecutive blocks of message can be chained together so that identical blocks of plaintext are not seen as identical blocks in ciphertext. The attacker would then be unable to identify the file type. A random non-zero initialization vector (IV) of the same length as a common block is used to begin the chain. The CBC mode is demonstrated by the following steps [2], [16]:
CBC Encryption
\[
C_1 = \text{CIPH}_K (P_1 \oplus IV)
\]
\[
C_j = \text{CIPH}_K (P_j \oplus C_{j-1})
\]

CBC Decryption
\[
P_1 = \text{CIPH}^{-1}_K (C_1) \oplus IV
\]
\[
P_j = \text{CIPH}^{-1}_K (C_j) \oplus C_{j-1}
\]
for \( j = 2 \ldots n \), where \( P_j \) is the \( j \)th plaintext block, \( C_j \) is the \( j \)th ciphertext block, \( \text{CIPH}_K \) is the forward cipher function of the block cipher algorithm under the key \( K \) that is applied to the data block \( X \), and \( \text{CIPH}^{-1}_K \) is the inverse cipher function of the block cipher algorithm under the key \( K \) that is applied to the data block \( X \).

The ECB mode is a confidentiality mode. In this mode, data are divided into blocks, and blocks are encrypted one at a time. Separate encryptions with different blocks are totally independent of one another, which indicates means that if data are transmitted over a network or phone line, transmission errors will only affect the block containing the error. ECB is the weakest of the various modes because no additional security measures are implemented aside from the basic algorithm. However, ECB is the fastest and the easiest to implement [4].

The definition of the ECB mode is as follows:

ECB Encryption
\[
C_j = \text{CIPH}_K (P_j)
\]

ECB Decryption
\[
P_j = \text{CIPH}^{-1}_K (C_j), \text{ for } j = 1 \ldots n.
\]

In ECB encryption, the forward cipher function is applied directly and independently to each block of the plaintext. The resulting sequence of output blocks is the ciphertext. In ECB decryption, the inverse cipher function is applied directly and independently to each block of the ciphertext. The resulting sequence of output blocks is the plaintext. The ECB mode is illustrated in Figure 3 [16].

![Figure 3. ECB mode](image)

**3. Blowfish 128-bit Data Types**

The randomness of the 128-bit Blowfish was tested by applying the NIST Statistical Suite [11]. The main test consisting of 15 core statistical tests that can be viewed as 188 statistical tests under different parameter inputs. Table 1 shows the breakdown of the 188 statistics tests used in the experiments. In this section, we provide six categories of data, including Random Plaintext/Random 128-Bit Keys, Low Density, and High Density, which are used to test the Advanced Encryption Standard (AES) candidate algorithms [17], Image, Text, and Video files.
3.1 Random Plaintext/Random 128-Bit Keys

The basis of this experiment is the data produced by the Blum–Blum–Shub (BBS) pseudo-random bit generator, which has been demonstrated to be a secure cryptographic pseudo-random bit generator and similar to the data type employed in testing AES finalist candidates [17]. One hundred and twenty-eight sequences were constructed for the examination of the randomness of ciphertext (based on random plaintext and random 128-bit keys). Each sequence was attributed to the concatenation of 8128 ciphertext blocks of 128 bits (1040384 bits) using 8128 random plaintext blocks of 128 bits and a random 128-bit key in ECB mode at one instance and another instance in CBC. BBS is implemented by using Java language (NetBeans IDE 7.2).

3.2 High-Density Plaintext

This experiment was based on data sets of 128 high-density sequences. Each sequence consisted of 8257 ciphertext blocks, using different random 256-bit keys per sequence in the ECB mode at one instance and another instance in CBC. The first ciphertext block was calculated using an all-ones plaintext block. Ciphertext blocks 2 to 129 were calculated using plaintext blocks consisting of a single zero and 127 ones, the zero appearing in each of the 128-bit positions of the plaintext block. Ciphertext blocks 130 to 8257 were calculated using plaintext blocks consisting of two zeros and 126 ones, the zeros appearing in each combination of two bit positions of the plaintext block [18].

3.3 Low-Density Plaintext

This experiment was based on data sets of 128 low-density sequences. Each sequence consisted of 8257 ciphertext blocks, using a distinct random 256-bit key per sequence in the ECB mode at one instance and another instance in CBC. The first ciphertext block was calculated using an all-zero plaintext block. Ciphertext blocks 2 to 129 were calculated using plaintext blocks consisting of a single one and 127 zeros, with the one appearing in each of the 128-bit positions of the plaintext block. Ciphertext blocks 130 to 8257 were calculated using plaintext blocks consisting of two ones and 126 zeros, with the ones appearing in each combination of two bit positions of the plaintext block [18].

3.4 Image Files

This experiment was based on a data set of 128 sequences of image files in different formats. Each sequence was a result of the concatenation of 12290 (1573120 bits) 128-bit ciphertext blocks using

<table>
<thead>
<tr>
<th>Statistical Test</th>
<th>No. of P-values</th>
<th>Test ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Block Frequency</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Cumulative Sum</td>
<td>2</td>
<td>3–4</td>
</tr>
<tr>
<td>Runs</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>Longest Run</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>Rank</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>FFT</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>Non Overlapping Template</td>
<td>148</td>
<td>9–156</td>
</tr>
<tr>
<td>Overlapping Template</td>
<td>1</td>
<td>157</td>
</tr>
<tr>
<td>Universal</td>
<td>1</td>
<td>158</td>
</tr>
<tr>
<td>Approximate Entropy</td>
<td>1</td>
<td>159</td>
</tr>
<tr>
<td>Random Excursions</td>
<td>8</td>
<td>160–167</td>
</tr>
<tr>
<td>Random Excursions Variant</td>
<td>18</td>
<td>168–185</td>
</tr>
<tr>
<td>Serial</td>
<td>2</td>
<td>186–187</td>
</tr>
<tr>
<td>Linear Complexity</td>
<td>1</td>
<td>188</td>
</tr>
</tbody>
</table>
12290 128-bit plaintexts blocks and a random 256-bit key in ECB mode at one instance and another instance in CBC mode.

3.5 Text Files

This experiment was based on a data set of 128 text files. Each file consisted of a sequence resulting from the concatenation of 8128 (1040384 bits) 128-bit ciphertext blocks using 16256 128-bit plaintext blocks and a random 256-bit key in ECB mode at one instance and another instance in CBC mode.

3.6 Video files

This experiment was based on a data set of 128 video files. Each file consisted of a sequence resulting from the concatenation of 8128 (1040384 bits) 128-bit ciphertext blocks using 16256 64-bit plaintext blocks and a random 256-bit key in ECB mode at one instance and another instance in CBC mode.

4. Experimental results

The randomness on the 128-bit Blowfish was tested on the six types of data, as mentioned in the previous section, in both partial and full-round considerations.

4.1 Full Round Testing (FRT)

In FRT for 128-bit Blowfish, all six data types were generated, which indicates that 16 types of data had to be derived.

4.2 Partial Round Testing (PRT)

Soto and Bassham in [18] tested Twofish rounds in pairs. Twofish, being a Feistel network, caused some of the data bits to be left unaltered after each round. Twofish does not seem to be random under the test conditions. Nevertheless, after two rounds, all data bits were affected. Twofish rounds were measured in pairs, that is, in rounds with even numbers from 2 to 14. Thus, in PRT on 128-bit Blowfish, all six data types were generated using pairs from 2 to 14.

We then discuss the output of the random test implementation for the six types of data on the extension of 128-bit Blowfish with ECB and CBC modes in PRT and FRT, respectively.

1-Random Plaintext/Random 128-Bit Keys

We illustrated the results of PRT and FRT of 128-bit Blowfish for two pairs of rounds on this type of data with the ECB and CBC modes in Figures 4 and 5, respectively. The dashed line at 96.09% indicates the smallest proportion satisfying the 0.01 criterion of acceptance, whereas the solid line at 99% indicates the proportion expected.
Randomness Analysis of 128 bits Blowfish Block Cipher on ECB and CBC Modes
Ashwak ALabaichi, Ramlan Mahmod, Faudziah Ahmad

Figure 4 (a–b). Results of random plaintext/random 128-bit keys for the (a) second round with EBC mode and (b) fourth round with EBC mode

Figure 5 (a–b). Results of random plaintext/random 128-bit keys for the (a) second round with CBC mode and (b) fourth round with CBC mode

The results evidently show that the output from the 128-bit Blowfish algorithms with both modes is random by the end of the second round (the first round pair) because the majority of the 188 statistical tests scored greater than 96%. Subsequent rounds produced similar statistics.

2-High-density plaintext

We illustrate the results of PRT and FRT of 128-bit Blowfish for two pairs of rounds on this type of data with ECB and CBC modes in Figures 6 and 7, respectively.

Figure 6 (a–b). Results of high-density plaintext for the (a) second round with EBC mode and (b) fourth round with EBC mode
The output from 128-bit Blowfish on this type of data with both modes was random by the end of the second round (the first round pair) in both modes because most of the 188 statistical tests scored greater than 96%. Subsequent rounds produced similar statistics.

3-Low-Density Plaintext

We illustrate the results of PRT and FRT of 128-bit Blowfish on this type of data with the ECB and CBC modes in Figures 8 and 9, respectively.

Figure 8 (a–b). Results of low-density plaintext for the (a) second round with EBC mode and (b) fourth round with EBC mode
The output from 128-bit Blowfish in ECB mode was evidently non-random at the end of the second round (the first round pair) because the majority of the 188 statistical tests scored less than 96%. However, the output was random by the end of the fourth round (second pair round) because the majority of the 188 statistical tests scored greater than 96%. Subsequent rounds produced similar statistics. In CBC modes, the output from 128-bit Blowfish was random at the end of the second round (the first round pair) because the majority of the 188 statistical tests scored greater than 96%. Subsequent rounds produced similar statistics.

4-Image files

We illustrated the results of PRT and FRT of 128-bit Blowfish on this type of data with the ECB and CBC modes in Figures 10 and 11, respectively.

Figure 10 (a–b). Results of image files for the (a) second round with EBC mode and (b) fourth round with EBC mode

Figure 11 (a–b). Results of image files for the (a) second round with CBC mode and (b) fourth round with CBC mode
The output from 128-bit Blowfish in ECB was evidently non-random by the end of the second round (the first round pair) because the majority of the 188 statistical tests scored less than 96%. At the fourth round (second pair round), the output remained non-random because the majority of the 188 statistical tests still scored less than 96%. Subsequent rounds produced similar statistics. In CBC modes, the output from 128-bit Blowfish was evidently random at the end of the second round (the first round pair) because the majority of the 188 statistical tests scored greater than 96%. Subsequent rounds produced similar statistics.

5-Text Files

We illustrated the results of PRT and FRT of 128-bit Blowfish on this type of data with the ECB and CBC modes in Figures 12 and 13, respectively.

(a)                (b)

Figure 12 (a–b). Results of text files for the (a) second round with EBC mode and (b) fourth round with EBC mode

(a)                (b)

Figure 13 (a–b). Results of text files for the (a) second round with CBC mode and (b) fourth round with CBC mode

The results evidently show that the outputs from 128-bit Blowfish in ECB mode remained non-random by the end of the second round (the first round pair) because the majority of the 188 statistical tests scored less than 96%. At the fourth round (second pair round), the output remained non-random because the majority of the 188 statistical tests still scored less than 96%. Subsequent rounds produced similar statistics. In the CBC mode, the output from 128-bit Blowfish was evidently random at the end of the second round (the first round pair) because the majority of the 188 statistical tests scored less than 96%. Subsequent rounds produced similar statistics.
6-Video Files

We illustrated the results of PRT and FRT of 128-bit Blowfish on this type of data with the ECB and CBC modes in Figures 14 and 15, respectively.

Figure 14 (a–b). Results of video files for the (a) second round with EBC mode and (b) fourth round with CBC mode

Figure 15 (a–b). Results of video files for the (a) second round with CBC mode and (b) fourth round with CBC mode

The output from 128-bit Blowfish in the ECB mode was evidently random by the end of the second round (the first round pair) because the majority of the 188 statistical tests scored greater than 96%. Subsequent rounds produced similar statistics. In the CBC mode, the output of 128-bit Blowfish was evidently random at the end of the second round (the first round pair) because the majority of the 188 statistical tests scored less than 96%. Subsequent rounds produced similar statistics.
5. Conclusion and Future Work

The results indicate that the randomness of the 128-bit BA with the ECB mode was unsuitable for image and text files with large strings of identical bytes. However, the 128-bit BA was more secure against attacks than 64-bit Blowfish because key space and block size were doubled in the former, thus increasing the complexity of brute attacks. However, this advantage is at the expense of the need for large memory. This finding can be considered a benchmark for the investigation of the effectiveness of the security of 128-bit Blowfish by enhancing its randomness as well as its performance by reducing memory requirements. Thus, in the future work, we will focus on reducing the memory requirements of 128-bit Blowfish without compromising security.

References


