ABSTRACT
A block cipher or a message authentication code (MAC) should behave as random mapping which can be evaluated during statistical tests. Statistical test suites typically used for evaluation of random mapping are NIST, Diehard, Dieharder and TestU01 which are not well-suited for block ciphers and MACs. These test suites employ frequentist approach; thus, making it difficult to overall evaluate the randomness of mapping. This paper describes the strong avalanche test, which overcomes the above mentioned deficiencies. The strong avalanche test is well suited for block ciphers and MACs and uses the Bayesian approach which yields the overall evaluation of the mapping’s randomness. Results of applying the strong avalanche test to KASUMI block cipher along with other statistical tests, which uses the same Bayesian approach, are reported and the results are analyzed to determine the randomness margin of the test.

Keywords
Statistical tests, Bayes Factor, KASUMI block cipher, Strong Avalanche test, MAC

1. INTRODUCTION
A block cipher or a message authentication code (MAC) is supposed to produce random outputs. The output’s randomness can be evaluated using a pseudo random number generator (PRNG) statistical test suite which performs various statistical tests on the produced outputs. Some of the widely used tests are NIST[1], Diehard[2], Dieharder[3] and TestU01[4].

NIST test suite is used for cryptographic applications, but its actually a general-purpose statistical test suite for evaluating the randomness of binary sequences. However, NIST does not care about the source of these binary sequences - whether from a cryptographic function or any other source. The paper [5] puts forward two problems which arise when the NIST test suite (or any other popular test suites mentioned in the paper) is used to evaluate a block cipher or MAC.

The first problem arises from the fact that block cipher or MAC is not itself a PRNG. Block ciphers or MACs do not generate arbitrarily long binary sequences as they take a fixed-length plaintext along with a fixed length key and maps it to a fixed length ciphertext. The mapping here is random (or supposed to be random). To apply NIST test suite on block cipher or MAC, the latter is made to behave as a PRNG and generate binary sequences. When a block cipher or MAC is turned into a PRNG and the NIST test is applied, it is the PRNG’s randomness that is being tested and not the randomness of the block cipher or MAC mapping itself. Thus, if non-randomness is detected in the output, it is not clear if the biasness is because of the block cipher or MAC, or because of the mode in which the block cipher or MAC is employed.

In short, the NIST test suite is not well-suited for evaluating block ciphers or MACs. It would be better to use statistical tests which directly evaluates the mappings of the block cipher, without needing to turn the block cipher or MAC into a PRNG.

The second problem arises from the fact that NIST test suite, like most other popular statistical test suites, takes a frequentist approach to statistical testing i.e. while analyzing the generated binary sequences, each test calculates a statistic and a p-value. The p-value is the probability that the observed value of the statistic would occur by chance, even if the generated binary sequence is actually random. If the p-value falls below a significance threshold, the test fails and indicates non-randomness, otherwise the test passes. The NIST statistical test suite consists of 15 statistical tests out of which some have multiple variations. Each test and variation is applied to multiple binary sequences that are produced, yielding several hundreds of p-values. The frequentist approach does not specify a method for combining these several hundreds of p-values into a single number that provides an overall random/non-random decision. On the other hand, Bayesian approach do specify a method to combine multiple test results into a single number, the posterior odds ratio, which determines whether a particular mapping is random or not.

The rest of the paper is organized as follows. Section 2 describes the previous work done on statistical tests using
a Bayesian approach. It also provides results of the test on some of the popular block ciphers, MACs and hash functions. Section 3 describes the KASUMI block cipher for which statistical tests of Bayesian approach are used to evaluate the randomness margin of the block cipher. Section 4 talks about the various components of KASUMI block cipher. Section 5 describes a new statistical test called the strong avalanche test which evaluates the randomness mapping of block cipher’s (plaintext, key)-to-ciphertext directly. The strong avalanche test uses a Bayesian approach yielding a clear random/non-random decision. Section 6 describes the various results produced by doing a statistical test of strong avalanche on the KASUMI block cipher. The results also show other statistical tests which uses Bayesian approach and are present in the cryptostat library. Section 7 discusses future work.

2. BACKGROUND

The paper [5] described a new statistical test called the coincidence test which used a Bayesian approach. The coincidence test evaluated the randomness mapping of block cipher’s (plaintext, key)-to-ciphertext directly. The coincidence test provided a clear random/non-random decision because of its Bayesian approach. The paper [5] also described the usefulness of the coincidence test by applying the test to reduced-round and full-round versions of various block ciphers like PRESENT and IDEA, hash function SipHash and a SQUASH MAC. The results of the test were reported in paper [5] which revealed each algorithm’s(block cipher or MAC or Hash function) randomness margin [5] - the number of rounds for which the algorithm produces random outputs, compared to full number of rounds.

2.1 Bayesian Model Selection


As mentioned in paper [5], let \( \mathcal{H} \) denote a hypothesis or a model which describes a particular process. Let \( D \) denote a data sample generated from running a process or performing some kind of experiment. Let \( \text{pr}(\mathcal{H}) \) be the probability of the hypothesis. Let \( \text{pr}(\mathcal{H} | D) \) be the conditional probability of the data sample, given a hypothesis. Let \( \text{pr}(D) \) be the probability of the data sample. According to Bayes’s Theorem,

\[
\text{pr}(\mathcal{H} | D) = \frac{\text{pr}(D | \mathcal{H}) \text{pr}(\mathcal{H})}{\text{pr}(D)}
\]

(1)

For two alternative hypothesis \( H_1 \) and \( H_2 \) and for data samples \( D \), the posterior odds ratio of the two hypothesis, \( \text{pr}(H_1 | D)/\text{pr}(H_2 | D) \), is calculated from Equation 1 as,

\[
\frac{\text{pr}(H_1 | D)}{\text{pr}(H_2 | D)} = \frac{\text{pr}(D | H_1) \text{pr}(H_1)}{\text{pr}(D | H_2) \text{pr}(H_2)}
\]

(2)

where, \( \text{pr}(H_1)/\text{pr}(H_2) \) is the prior odds ratio of the two hypothesis and \( \text{pr}(D | H_1)/\text{pr}(D | H_2) \) is the Bayes factor. The odds ratio represents the relative probabilities of the two models. The prior odds ratio is one’s assumption before observing any samples and posterior odds ratio is one’s assumption after observing a sample. It can be said that, posterior odds ratio = Bayes Factor x prior odds ratio.

For two observed samples, \( D_1 \) and \( D_2 \) and assuming that the samples are independent, the calculation of the posterior odds ratio based on the observed samples is given by,

\[
\begin{align*}
\text{pr}(H_1 | D_1, D_2) &= \frac{\text{pr}(D_2 | H_1) \text{pr}(H_1 | D_1)}{\text{pr}(D_2 | H_2) \text{pr}(H_2 | D_1)} \\
\text{pr}(H_2 | D_1, D_2) &= \frac{\text{pr}(D_2 | H_2) \text{pr}(H_2 | D_1)}{\text{pr}(D_2 | H_1) \text{pr}(H_1 | D_1)}
\end{align*}
\]

(3)

In short, the posterior odds ratio for the first experiment becomes the prior odds ratio for the second experiment. Equation 3 can be extended to any number of independent samples \( D_i \). The final posterior odds ratio will be the product of the initial prior odds ratio and the Bayes Factor for all observed samples.

The previous model or hypothesis had no parameters. If model \( H_1 \) has a parameter \( \theta_1 \) and model \( H_2 \) has a parameter \( \theta_2 \), then the conditional probabilities of the given samples is given by

\[
\begin{align*}
\text{pr}(D_1 | H_1) &= \int \text{pr}(D_1 | \theta_1, H_1) \text{pr}(\theta_1 | H_1) d\theta_1 \\
\text{pr}(D_2 | H_2) &= \int \text{pr}(D_2 | \theta_2, H_2) \text{pr}(\theta_2 | H_2) d\theta_2
\end{align*}
\]

where, \( \text{pr}(D_1 | \theta_1, H_1) \) is the probability of the observed sample for model \( H_1 \) with \( \theta_1 \) as its parameter. \( \text{pr}(\theta_1 | H_1) \) is the prior probability density of \( \theta_1 \) for \( H_1 \) model. \( \text{pr}(D_2 | \theta_2, H_2) \) is the probability of the observed sample for model \( H_2 \) with \( \theta_2 \) as its parameter. \( \text{pr}(\theta_2 | H_2) \) is the prior probability density of \( \theta_2 \) for \( H_2 \) model. Bayes Factor is then calculated by taking the ratios of these two integrals.

The paper [5] describes that the coincidence test performs \( n \) Bernoulli trials, where \( \theta \) is the success probability and counts the number of successes \( k \), which follows binomial distribution. Based on the model \( H \), the probability of \( D \) given \( H \) with its parameter as \( \theta \) is given by

\[
\text{pr}(D | \theta, H) = \binom{n}{k} \theta^k (1 - \theta)^{n-k}
\]

(4)

As mentioned in paper [5], the prior probability density of \( \theta_1 \) is a delta function, \( \text{pr}(\theta_1 | H_1) = \delta(\theta_1 - p) \), and the Bayes factor numerator is now given by,

\[
\text{pr}(D | H_1) = \frac{n!}{k!(n-k)!} p^k (1 - p)^{n-k}
\]

(5)

Similarly, the Bayes factor denominator is given by,
\[ pr(D|H_2) = \int_0^1 \frac{n!}{k!(n-k)!} \theta_2^k (1 - \theta_2)^{n-k} d\theta_2 = \frac{1}{n+1} \]

Putting all the equations together, Bayes Factor is given by,

\[
\frac{pr(D|H_1)}{pr(D|H_2)} = \frac{n!}{k!(n-k)!} p^k (1-p)^{n-k} (n+1) \quad (6)
\]

The paper [5] mentions the use of the log of Equation 6 to deciding randomness mapping of statistical tests on block ciphers or MACs.

3. KASUMI BLOCK CIPHER

There are two standard algorithms used within the security architecture of the 3GPP [3GPP][7]: a confidentiality algorithm f8, and an integrity algorithm f9. These algorithms are based on the KASUMI block cipher algorithm. KASUMI is a feistel structured block cipher with eight rounds that produces a 64-bit output from a 64-bit input under the control of a 128-bit key. Table 1 shows the symbols used to represent different operations. Table 2 refers to the functions and variables used in various components of KASUMI.

<table>
<thead>
<tr>
<th>#</th>
<th>Assignment operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Bitwise exclusive-OR operator</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>≪</td>
<td>Left circular rotation of the operand by ( n ) bits</td>
</tr>
<tr>
<td>ROL()</td>
<td>Left circular rotation of the operand by one bit</td>
</tr>
<tr>
<td>∩</td>
<td>Bitwise AND</td>
</tr>
<tr>
<td>∪</td>
<td>Bitwise OR</td>
</tr>
</tbody>
</table>

Table 1: Symbol Representation[7]

4. COMPONENTS OF KASUMI

4.1 Function \( f_i \)

As shown in figure 1, the function \( f_i() \) takes a 32-bit input \( I \) and returns a 32-bit output \( O \) under the control of a round key \( RK_i \), where the round key comprises of three subkeys: \( KL_i, KO_i, KI_i \). Function \( f_i() \) is constructed from two subfunctions: \( FL() \) where subkey \( KL_i \) is used and \( FO() \) where subkeys \( KO_i \) and \( KI_i \) are used. The \( f_i() \) function is defined differently for even and odd rounds. For odd rounds, the round data is first passed through \( FO() \) and then through \( FL() \), while for even rounds, the round data is first passed through \( FL() \) and then through \( FO() \).

For odd rounds (1, 3, 5, 7), \( f_i(I, RK_i) = FO(FL(I, KL_i), KO_i, KI_i) \)

For even rounds (2, 4, 6, 8), \( f_i(I, RK_i) = FL(FO(I, KO_i, KI_i), KL_i) \)

<table>
<thead>
<tr>
<th>( f_i )</th>
<th>Round function for the ( i^{th} ) round of KASUMI</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL()</td>
<td>Subfunction within KASUMI that translates a 16-bit input to a 16-bit output using 16-bit subkey</td>
</tr>
<tr>
<td>FO()</td>
<td>Subfunction within KASUMI that translates a 32-bit input to a 32-bit output using 32-bit subkey</td>
</tr>
<tr>
<td>KO_i, KI_i, KL_i</td>
<td>Subkeys used within ( f_i() )</td>
</tr>
<tr>
<td>K</td>
<td>A 128-bit key</td>
</tr>
<tr>
<td>S7[]</td>
<td>An S-Box translating 7-bit input to a 7-bit output</td>
</tr>
<tr>
<td>S9[]</td>
<td>An S-Box translating 9-bit input to a 9-bit output</td>
</tr>
</tbody>
</table>

Table 2: Functions/Variables[7]

4.2 Function \( FL() \)

As shown in figure 2, the function \( FL() \) takes a 32-bit input \( I \) and a 32-bit subkey \( KL_i \). The subkey is split into two 16-bit subkeys, \( KL_{i,1} \) and \( KL_{i,2} \), where, \( KL_i = KL_{i,1} || KL_{i,2} \)

The input data is split into two 16-bit halves, \( L \) and \( R \), where,

\( I = L || R \)

Function \( FL() \) defines the following operations:

\[
\begin{align*}
R' & = R \oplus ROL(L \cap KL_{i,1}) \\
L' & = L \oplus ROL(R \cap KL_{i,2})
\end{align*}
\]

The function \( FL() \) returns a 32-bit output \( O \), where \( O = L || R' \)

4.3 Function \( FO() \)

As shown in figure 3, the function \( FO() \) takes a 32-bit input \( I \) and two sets of subkeys: a 48-bit subkey \( KO_i \) and a 48-bit subkey \( KI_i \).

The input data is split into two halves, \( L_0 \) and \( R_0 \), where,

\( I = L_0 || R_0 \)

The subkeys are subdivided into three 16-bit subkeys where,

\[
\begin{align*}
KO_i = KO_{i,1} || KO_{i,2} || KO_{i,3} \\
KI_i = KI_{i,1} || KI_{i,2} || KI_{i,3}
\end{align*}
\]

Function \( FO() \) defines the following operations:

\[
\begin{align*}
& \text{for each integer } j \text{ such that } 1 \leq j \leq 3, \\
& R_j = FL(I_{j-1} \oplus KO_{i,j}, KI_{i,j}) \oplus R_{j-1} \\
& L_j = R_{j-1}
\end{align*}
\]

The function \( FO() \) returns a 32-bit output \( O \), where,

\( O = L_3 || R_3 \)

4.4 Function \( FI() \)
As shown in figure 4, the function $FI()$ takes a 16-bit input $I$ and a 16-bit subkey $KI_{i,j}$.

The input data is split into two unequal components, 9-bit left half $L_0$ and a 7-bit right half $R_0$ where,

$I = L_0 \parallel R_0$

Similarly, the subkey $KI_{i,j}$ is split into 7-bit left half $KI_{i,j,1}$ and 9-bit right half $KI_{i,j,2}$ where,

$KI_{i,j} = KI_{i,j,1} \parallel KI_{i,j,2}$

The function $FI()$ makes use of two S-boxes: $S7$ which maps a 7-bit input to a 7-bit output and $S9$ which maps a 9-bit input to a 9-bit output. The function $FI()$ also uses the following two functions: $ZE()$ - Zero Extend, $TR()$ - Truncate. $ZE(x)$: takes 7-bit value $x$ and converts it to a 9-bit value by adding two zero bits to the most significant end.

Function $FI()$ defines the following operations:

$L_1 = R_0$, $R_1 = S9[L_0] \oplus ZE(R_0)$
$L_2 = R_1 \oplus KI_{i,j,2}$, $R_2 = S7[L_1] \oplus TR(R_1) \oplus KI_{i,j,1}$
$L_3 = R_2$, $R_3 = S9[L_2] \oplus ZE(R_2)$
$L_4 = S7[L_3] \oplus TR(R_3)$, $R_4 = R_3$

The function $FI()$ returns a 16-bit output $O$, where,

$O = L_4 \parallel R_4$

### 4.5 S-boxes

S-boxes have been designed in such a way that they can be implemented in both combinational logic and look-up table. For this paper, look-up table has been used. However, one can use any one of the S-boxes designs. $S7$ and $S9$ are the two types of S-boxes used. Figure 5 represents the look-up table of $S7$ and figure 6 represents the look-up table of $S9$.

**Example:**
If input data value = 1, then using look-up table, $S7[1] = 50$ and $S9[1] = 239$.

### 4.6 Key Schedule

KASUMI consists of 128-bit key $K$. Each round of KASUMI uses 128 bits of key that are derived from $K$. Table 4 represents the each round subkeys used.
128-bit key is subdivided into eight 16-bit values $K1...K8$ where,

$$K = K1 \parallel K2 \parallel K3 \parallel K4 \parallel K5 \parallel K6 \parallel K7 \parallel K8$$

KASUMI key schedule derives the following operation:

for each integer $j$, where, $1 \leq j \leq 8$,

$$K'j = Kj \oplus Cj,$$ where,

$Cj$ is a constant value defined in table 3.

5. STRONG AVALANCHE TEST
5.1 Strong Avalanche Test Definition
A block cipher strong avalanche is defined in figure 7. Let $P$ be a plaintext input value, $BP$ be a plaintext input value with a flipped bit, let $K$ be an encryption key input value, let $BK$ be an encryption key input value with flipped bit. Let $C$ be the ciphertext output when plaintext $P$ is encrypted with key $K$. Let $C'$ be the ciphertext output when plaintext $P$ is encrypted with key $BK$ or when plaintext $BP$ is encrypted with key $K$. Consider certain groups of output bits $G$, where $g = |G|$ is the number of bits in a group. If each bit of $C'$ in a bit group is not equal to the corresponding bit of $C$ in a bit group $G$, then we say that it is a strong avalanche with respect to the input and output values along with the bit groups. The paper [5] mentioned the log Bayes Factor which is used by the strong avalanche test to determine random/non-randomness in the block cipher.

The strong avalanche test is performed on KASUMI block cipher which takes in a 64-bit input and produces a 64-bit output under the control of a 128-bit key. The original input of plaintext and key is flipped one bit at a time to produce its corresponding outputs. Each produced output is then checked for its independence with respect to the original output. For each trial in the sample, the strong avalanche test encrypts plaintext using key and examines multiple bit groups in the output. The bit groups used currently are:

- one-bit groups: bit 0, 1, 2, ... 63
- two-bit groups: bit 0-1, 2-3, 4-5, ... 62-63
- four-bit groups: bit 0-3, 4-7, ... 60-63
- eight-bit groups: bit 0-7, 8-15, 16-23, ... 56-63
- sixteen-bit groups: bit 0-15, 16-31, 32-47, 48-63
- thirty-two-bit groups: bit 0-31, 32-63
- sixty-four-bit groups: bit 0-63

6. RESULTS
The strong avalanche test was performed on the KASUMI block cipher.

Figure 8 shows the largest number of rounds for which each test present in the cryptostat library detected non-random behavior on KASUMI block cipher.

Figure 9 shows the randomness margin of each test present in the cryptostat library for KASUMI block cipher. Randomness margin is given by

\[ \text{bits groups are chosen randomly unlike shown.} \]
subkey name | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8
--- | --- | --- | --- | --- | --- | --- | --- | ---
\(K_{L_i,1}\) | \(K_{1\ll 1}\) | \(K_{2\ll 1}\) | \(K_{3\ll 1}\) | \(K_{4\ll 1}\) | \(K_{5\ll 1}\) | \(K_{6\ll 1}\) | \(K_{7\ll 1}\) | \(K_{8\ll 1}\)
\(K_{L_i,2}\) | \(K_3\) | \(K_4\) | \(K_5\) | \(K_6\) | \(K_7\) | \(K_8\) | \(K_1\) | \(K_2\)
\(K_{O_i,1}\) | \(K_{2\ll 5}\) | \(K_{3\ll 5}\) | \(K_{4\ll 5}\) | \(K_{5\ll 5}\) | \(K_{6\ll 5}\) | \(K_{7\ll 5}\) | \(K_{8\ll 5}\) | \(K_{1\ll 5}\)
\(K_{O_i,2}\) | \(K_{6\ll 8}\) | \(K_{7\ll 8}\) | \(K_{8\ll 8}\) | \(K_{1\ll 8}\) | \(K_{2\ll 8}\) | \(K_{3\ll 8}\) | \(K_{4\ll 8}\) | \(K_{5\ll 8}\)
\(K_{O_i,3}\) | \(K_{7\ll 13}\) | \(K_{8\ll 13}\) | \(K_{1\ll 13}\) | \(K_{2\ll 13}\) | \(K_{3\ll 13}\) | \(K_{4\ll 13}\) | \(K_{5\ll 13}\) | \(K_{6\ll 13}\)
\(K{I_i,1}\) | \(K_5\) | \(K_6\) | \(K_7\) | \(K_8\) | \(K_1\) | \(K_2\) | \(K_3\) | \(K_4\)
\(K{I_i,2}\) | \(K_4\) | \(K_5\) | \(K_6\) | \(K_7\) | \(K_8\) | \(K_1\) | \(K_2\) | \(K_3\)
\(K{I_i,3}\) | \(K_8\) | \(K_1\) | \(K_2\) | \(K_3\) | \(K_4\) | \(K_5\) | \(K_6\) | \(K_7\)

Table 4: Round subkeys[7]

Figure 7: Strong Avalanche Test

Randomness margin = 1 - \(r\) \(R\), where, \(r\) is the largest number of round for which each test showed non-randomness and \(R\) is the total number of rounds of a block cipher on which the test is performed.

Figure 10 represents how each test behaves at each round from 1 to 8. The log Bayes factor plays an important role in deciding if a particular test passes or fails at each round. A threshold range of -4.3 to +4.3 of log Bayes factor decides if the mapping is random or not. log Bayes Factor equal to and above +4.3 are considered as pass, log Bayes factor less than -4.3 are considered as fail, log Bayes factor between 0 to +4.3 are conditional pass which means that it requires more analysis to arrive on a decision for it. log Bayes Factor between -4.3 to 0 are considered as conditional fail which means that it requires more analysis to arrive for a decision for it.

Table 5 shows the test results of strong avalanche test for each round of KASUMI block cipher.

7. FUTURE WORK

- Run the strong avalanche test on graphics processing unit (GPU) accelerators and supercomputers, which would help in running the tests with larger number of trials and samples
- Adding more Bayesian statistical tests besides the strong avalanche test to create a new cryptographic Bayesian statistical test suite
- To analyze different types of block ciphers, MACs, Hash functions using strong avalanche test
- To figure out the relation between the randomness margin of a particular block cipher with respect to its security margin

8. REFERENCES
<table>
<thead>
<tr>
<th>Round No.</th>
<th>Bit Group</th>
<th>log Bayes Factor</th>
<th>Test Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>28, 13, 24, 4, 0, 17, 1, 30</td>
<td>-8.00431 × 10^8</td>
<td>FAIL</td>
</tr>
<tr>
<td>2</td>
<td>1, 18, 62, 45, 9, 37, 29, 35, 39, 55, 34, 60, 17, 27, 25, 23</td>
<td>-1.06505 × 10^8</td>
<td>FAIL</td>
</tr>
<tr>
<td>3</td>
<td>1, 18, 62, 45, 9, 37, 29, 35, 39, 55, 34, 60, 17, 27, 25, 23</td>
<td>-2.62781 × 10^5</td>
<td>FAIL</td>
</tr>
<tr>
<td>4</td>
<td>34, 42, 0, 40, 14, 41, 37, 18, 1, 55, 9, 52, 45, 43, 4, 13, 29, 44, 23, 16, 59, 22, 48, 33, 58, 15, 3, 49, 60, 30, 25, 2</td>
<td>9.06072 × 10^4</td>
<td>pass</td>
</tr>
<tr>
<td>5</td>
<td>29, 17, 8, 6, 41, 52, 42, 57, 46, 32, 26, 10, 25, 62, 36, 33</td>
<td>8.99498 × 10^4</td>
<td>pass</td>
</tr>
<tr>
<td>6</td>
<td>18, 15, 20, 42, 55, 25, 51, 54, 31, 1, 59, 38, 11, 14, 29, 45, 50, 41, 23, 35, 13, 57, 21, 39, 63, 47, 12, 7, 5, 3, 46, 37</td>
<td>9.04235 × 10^4</td>
<td>pass</td>
</tr>
<tr>
<td>7</td>
<td>29, 28, 50, 19, 33, 51, 59, 37, 35, 52, 36, 22, 45, 40, 6, 48, 10, 11, 25, 4, 8, 46, 9, 44, 58, 32, 2, 39, 18, 27, 49, 56</td>
<td>8.90243 × 10^4</td>
<td>pass</td>
</tr>
<tr>
<td>8</td>
<td>33, 46, 14, 11, 54, 22, 29, 50, 21, 58, 30, 57, 60, 0, 39, 13, 63, 12, 47, 27, 43, 24, 3, 61, 28, 53, 45, 38, 34, 51, 1, 55</td>
<td>8.87490 × 10^4</td>
<td>pass</td>
</tr>
</tbody>
</table>

Table 5: Strong Avalanche Test Results

![Randomness Margin](image)

Figure 9: Randomness Margin

![Log Bayes Factor](image)

Figure 10: Log Bayes Factor


APPENDIX

Cryptostat is a library which consists of Bayesian statistical tests and some of the block ciphers and MACs and Hash functions on which the tests are performed. The library can be found at http://www.cs.rit.edu/~ark/parallelcrypto/cryptostat/

The following are the cryptographic functions implemented in the cryptostat library:

- 3WAY block cipher
- Blowfish block cipher
- Kasumi block cipher
• Present-80 block cipher

The following are the statistical tests that are implemented in cryptostat library:

- Uniformity test
- Ciphertext Independence test
- Complement test
- Input-Output Independence test
- Strong Avalanche test
- Nonlinearity test

CryptoStat performs a series of statistical tests on a cryptographic function. The usage of the library is defined as follows:

```java
pj2 CryptoStat expr T N B W randomfile outputfile
```

where,

- `T` = number of trials
- `N` = number of samples per trial
- `B` = Number of bit groups chosen for each bit group size
- `W` = Number of worker tasks
- `randomfile` = file of random bytes
- `outputfile` = output filename

CryptoStat performs T trials. Each trial computes the cryptographic function on 5N samples of input data. They are:

- N samples : All-0s plaintext, random key
- N samples : All-1s plaintext, random key
- N samples : Random plaintext, all-0s key
- N samples : Random plaintext, all-1s key
- N samples : Random plaintext, random key