CHAOS IN CRYPTOGRAPHY

AMEY P GAIKWAD

RAHUL DANDWATE

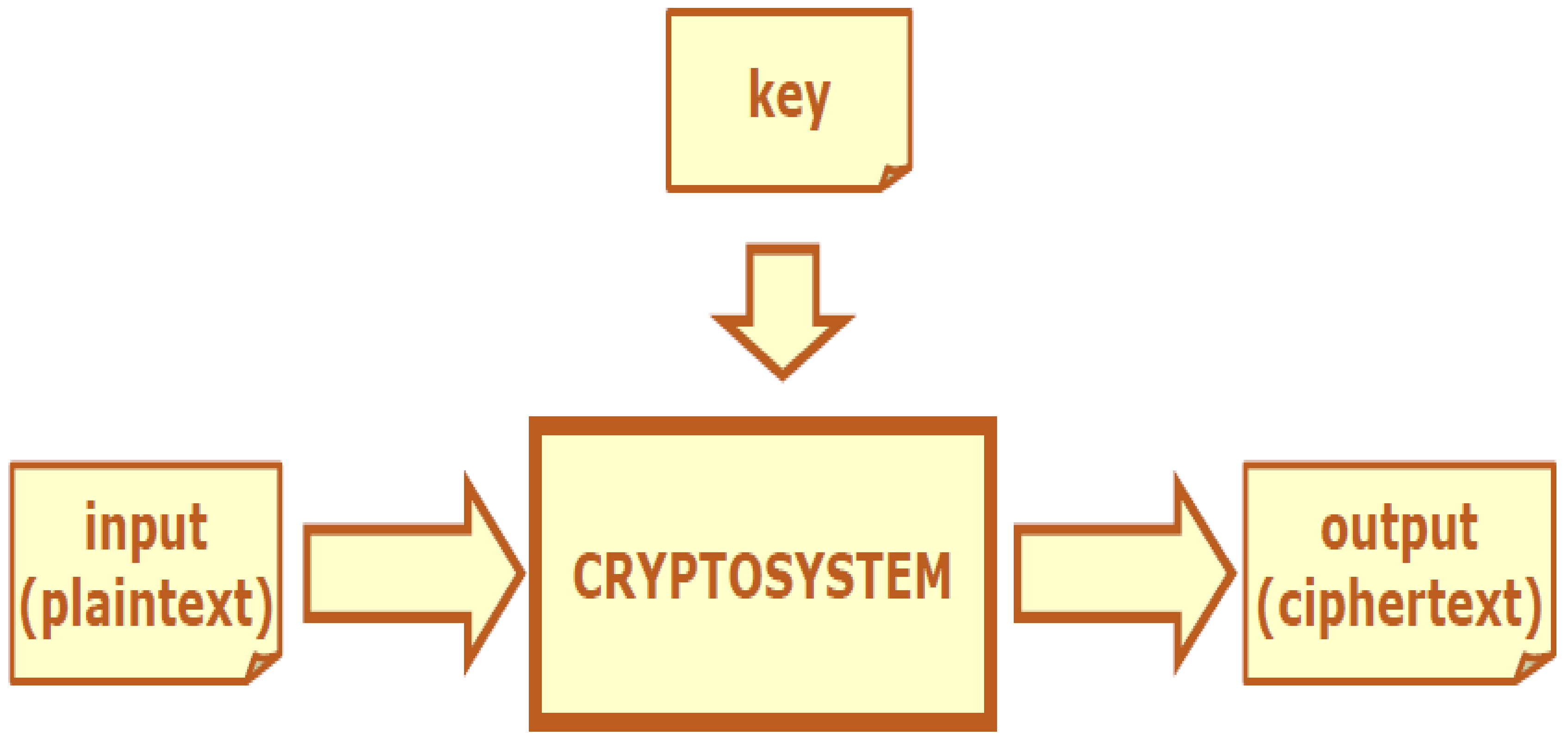
SUMUKH VAIDYA

SIDDHESH PEDNEKAR

PALLAV JOSHI

Cryptography

- Cryptography is the study of methods of converting messages into a disguised form so that only the intended recipient can remove the disguise and read the message
- Cryptography – The art of making
- Cryptanalysis – The art of breaking


key
input
(plaintext)
CRYPTOSYSTEM
output
(ciphertext)

- Mathematically ,

A Cryptosystem is a five tuple (*P,C,K,E,D*) satisfying the following conditions;

1. *P is a set of possible plaintexts*
2. *C is set of possible ciphertexts*
3. *K, the key space, is a finite set of possible keys*
4. *For each k ∈ K, there is an encryption rule* $e_k \in E$ *and a corresponding decryption rule* $d_k \in D$ *where*

$$e_k : P \to C \quad and \quad d_k : C \to P$$

are functions such that $d_k(e_k(x)) = x$ *for every plaintext x*$\in P$

Kerkchoff Shannon Principle

- A cryptosystem must be secure even if everything about the system , except the key is public knowledge.

Shannon’s Principle:

The enemy knows the system i.e the ALGORITHM

Chaos in Cryptography

- Chaotic cryptography is the application of the mathematical chaos theory to the practice of the cryptography, the study or techniques used to privately and securely transmit information with the presence of a third-party or adversary. The use of chaos or randomness in cryptography has long been sought after by entities wanting a new way to encrypt messages.
- The mapping between these two fields will be highlighted later on.

Principles of Modern Ciphers

- Maximum entropy of the cipher
- Maximum diffusion of the key
- Long cycle length of cipher (to avoid repetition)

Diffusion - Transposition

+ $\rightarrow$ Chaos

Confusion - Substitution

Applications Of Chaos In Cryptography

- **Pseudo Random Number Generators (PRNGs) (An overview)**
- **Standalone algorithms developed from chaotic maps**
- Chaos synchronization as means to secure communications
- Cellular Automata cryptography

RNGs and PRNGs

RNGs

- Truly random
- Generated through natural processes (fluctuations) feeding the black box to generate the random numbers.(Hardware)
- Completely random and nondeterministic.

PRNGs

- As the name suggests pseudo random
- Generated through a mathematical algorithm with the help of an initial seed.
- Deterministic (know the seed , know the random number)

TRNGs and PRNGs

- These random number generators are used in variety of cryptosystems (eg. The Vernam Cipher)
- For absolute secrecy the random numbers generated need to be TRNGs. However due to computational limitations PRNGs are used.
- These PRNGs are going to result **in practical rather than perfect secrecy** (due to computational limitations).
- The **seed space** of TRNGs is much larger than the seed space of PRNGs

Chaos and PRNGs

- The pseudo random number generators are deterministic.(given the initial seed since the algorithm is a mathematical computation)
- If we change the initial seed value an entirely different random number is generated.
- One method of producing PRNGs is **the middle squares method** developed by Neumann.
- The random number generated is not due to the error propagation but rather due to the inherent mathematical formulation of the algorithm.
- For the middle squares method taking an n digit seed atmost 10^n numbers are generated before reusing the seed.
- Prevent reusing the seed (OTP).

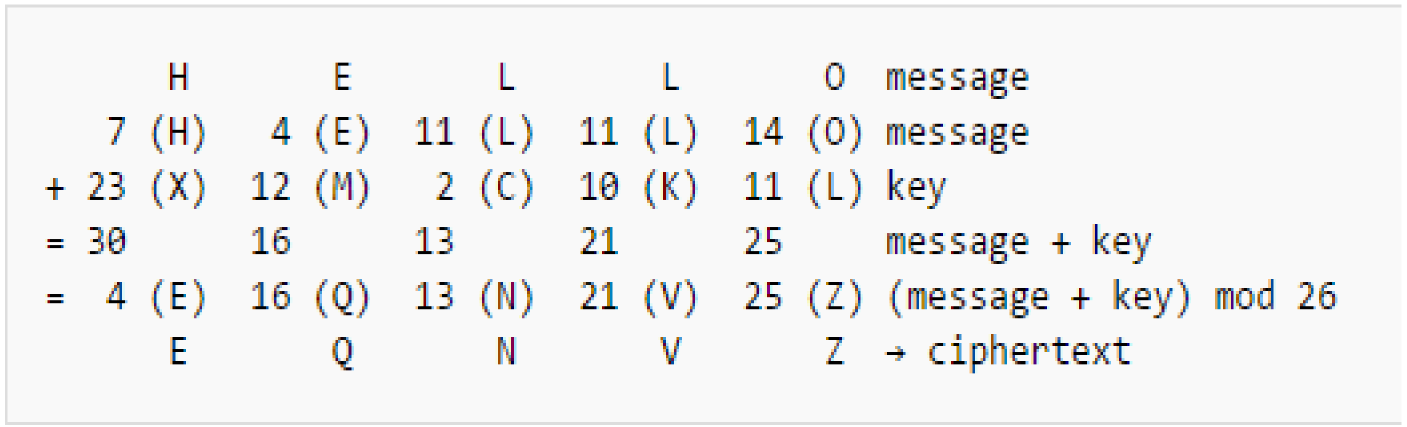
One Time Pads (OTP)

- Before moving to the actual presence of chaos in an actual cryptosystem an example of an ideal cipher design is mentioned for comparisons with the future designs shown
- Perfect secrecy
- Claude Shannon – 1) impossible to break 2) Vernam cipher
- Assumptions :

1. One time use
2. Truly random numbers are needed
3. As long as the plain text (the key)
4. The key is kept complete secret

OTP Cipher Design

- Generally modulo arithmetic is used
- In the simplest possible case, the alphabets are numbered from 0 to 25.
- A TRN is generated and added to the plain text
- The sum is calculated modulo 26.
- The ciphertext is obtained.



```
      H       E       L       L       O  message
   7 (H)   4 (E)  11 (L)  11 (L)  14 (O) message
+ 23 (X)  12 (M)   2 (C)  10 (K)  11 (L) key
= 30      16      13      21      25     message + key
=  4 (E)  16 (Q)  13 (N)  21 (V)  25 (Z) (message + key) mod 26
      E       Q       N       V       Z  → ciphertext
```

Source : Wikipedia

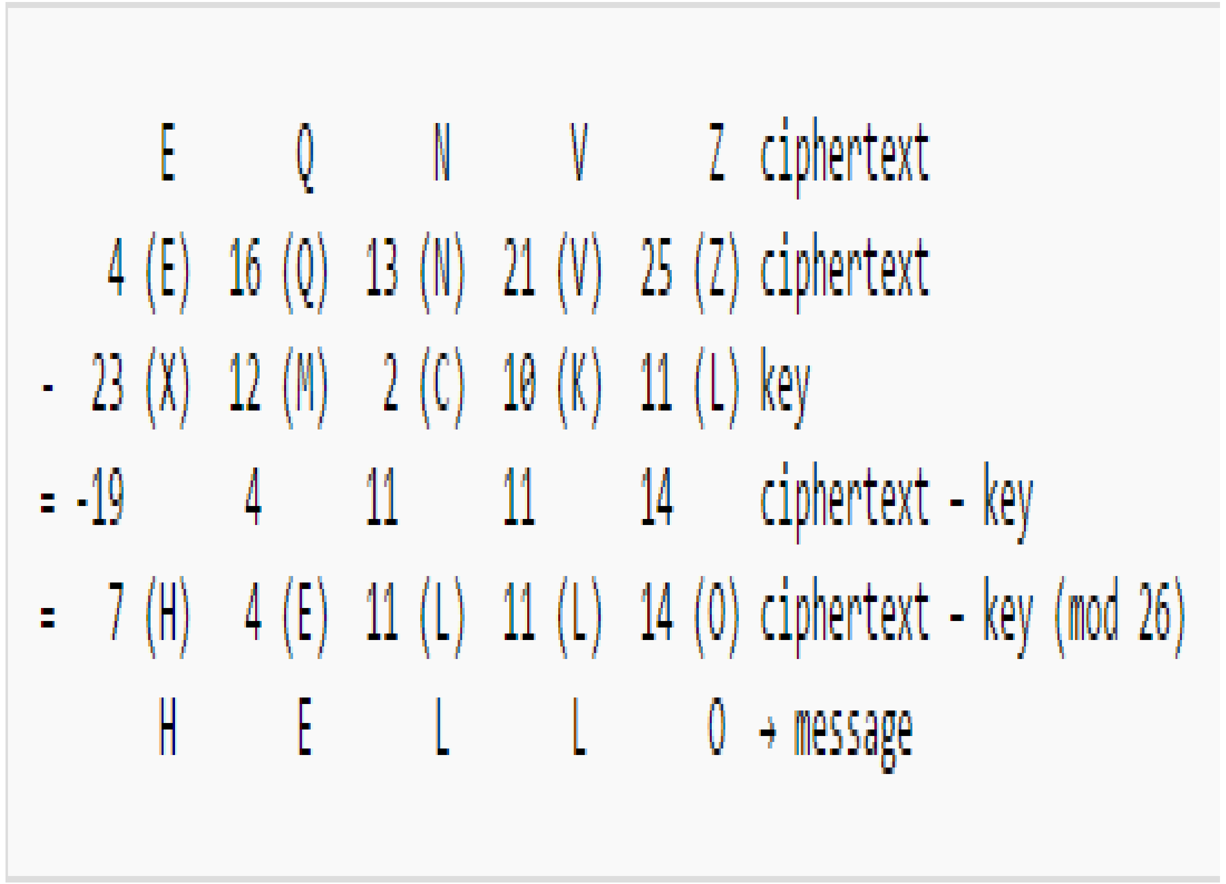
Cipher Evaluation

- The entropy of the cipher text is very large and hence this cryptosystem is a very good and an efficient one.
- Given a plain text the a priori and the posteriori probabilities of the plain text are equal.
- In other words the entropy of a plaintext is equal to the entropy of the plaintext given the ciphertext. Its impossible to predict the plaintext from the ciphertext.
- **$H(M)=H(M|C)$**
- The keyspace consists of just one key vector

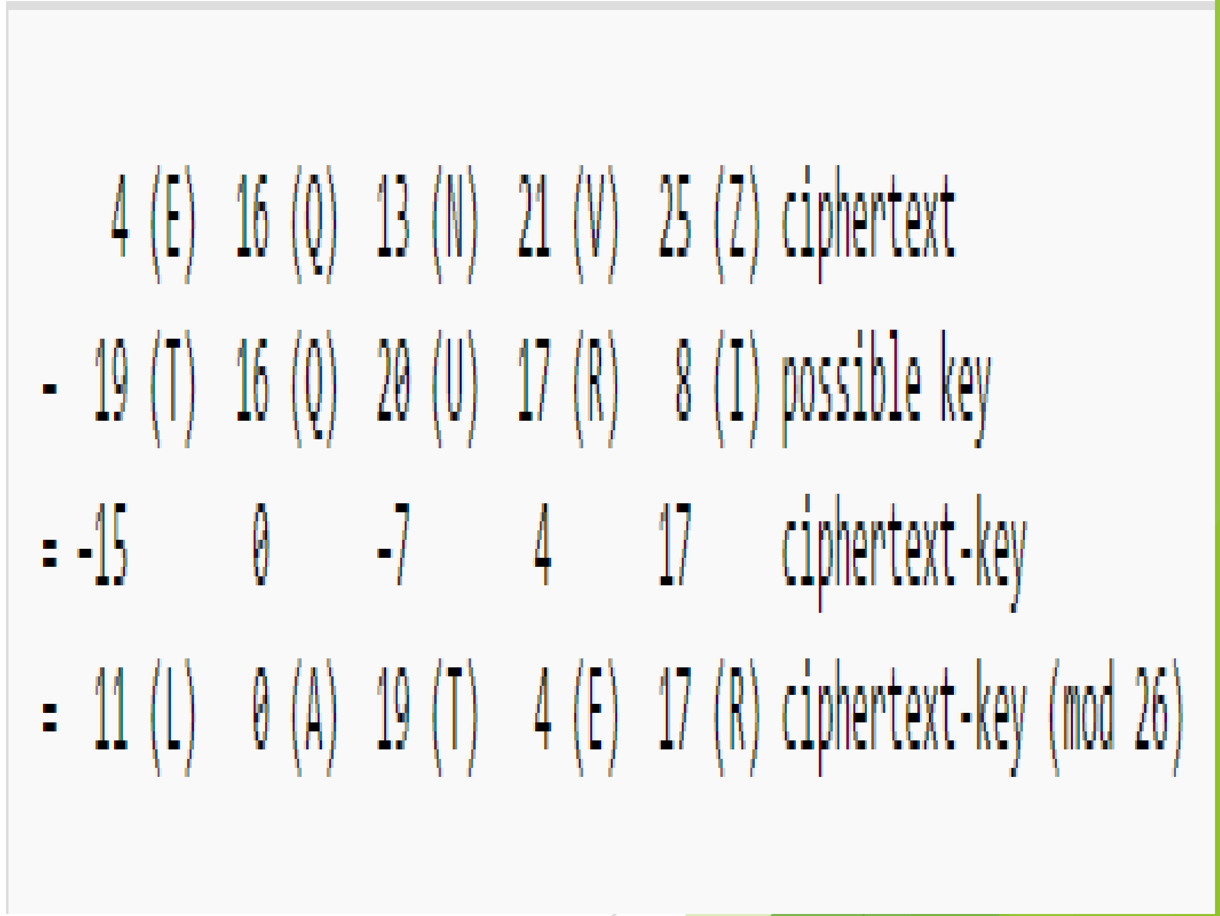
Security Analysis

- If TRNG is used then the OTP is unbreakable. As proven by Shannon **its information theoretically secure** and has the property which he termed as absolute secrecy.
- Given a cipher text **any plain text can be concocted from it** by using random numbers and using the modulo in a reverse manner.
- Even if a PRNG with a very long seed is used then even with the present supercomputers it would take years to break it down.(This is possible since after some finite value the seed is reused).

Source: Wikipedia`



```
        E      Q      N      V      Z  ciphertext
     4 (E) 16 (Q) 13 (N) 21 (V) 25 (Z) ciphertext
-   23 (X) 12 (M)  2 (C) 10 (K) 11 (L) key
= -19       4     11     11     14     ciphertext - key
=    7 (H)  4 (E) 11 (L) 11 (L) 14 (O) ciphertext - key (mod 26)
        H      E      L      L      O  → message
```

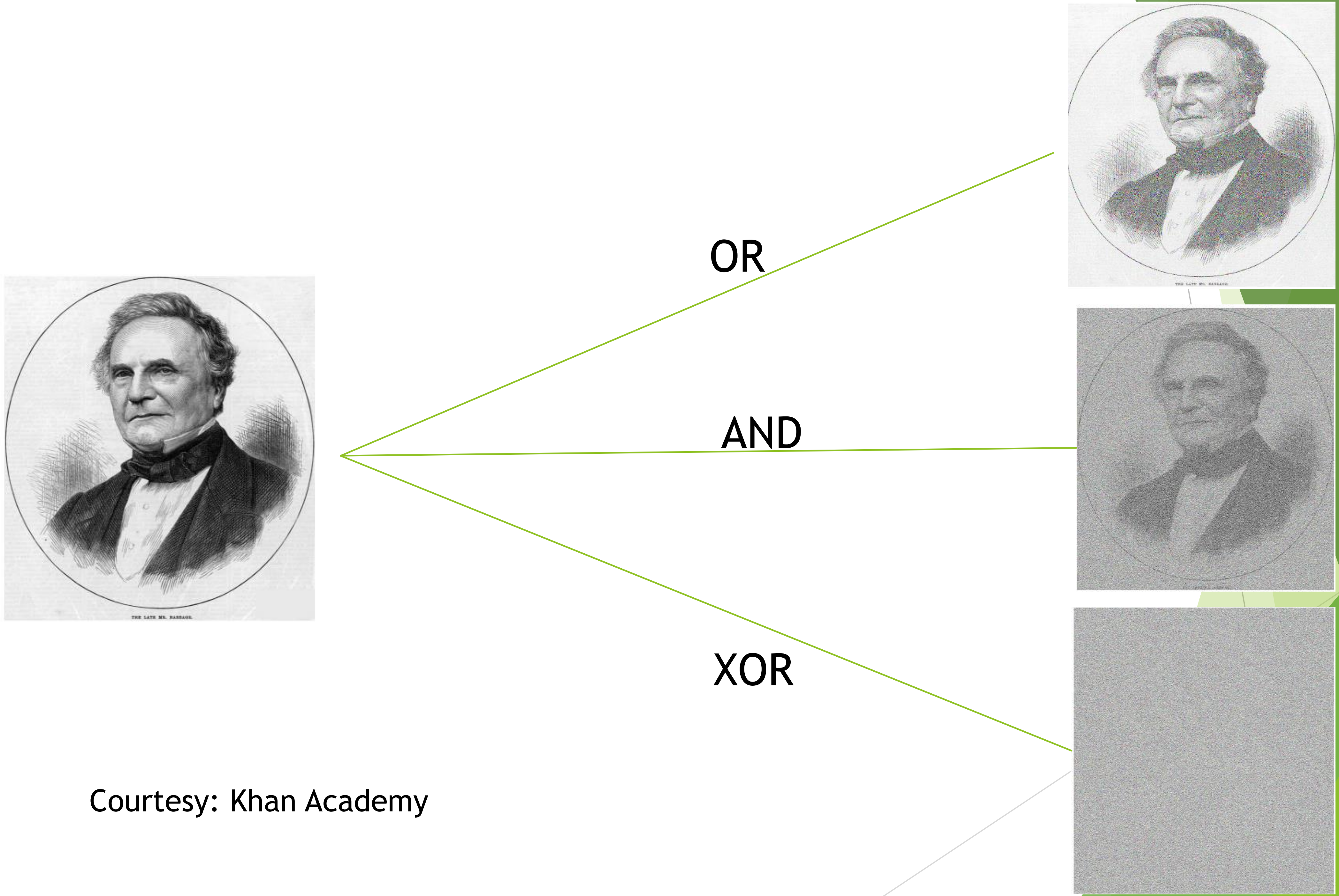


```
     4 (E) 16 (Q) 13 (N) 21 (V) 25 (Z) ciphertext
-   19 (T) 16 (Q) 20 (U) 17 (R)  8 (I) possible key
= -15       0     -7      4     17     ciphertext-key
=   11 (L)  0 (A) 19 (T)  4 (E) 17 (R) ciphertext-key (mod 26)
```

The Vernam Cipher Design

- Convert the plain text and the random number of the same length as the plain text into binary.
- c=p $\oplus n$ (for encryption)
- p=n $\oplus c$ (for decryption)
- **Why XOR?**

Take an image. Suppose each pixel is coded 0 or 1. The XOR function converts the code into the ciphertext with no statistical bias unlike the OR and the AND operators.



Courtesy: Khan Academy

Cipher Evaluation

- Depending on the length of the plain text cryptanalysis of the Vernam cipher varies.
- The key space of the Vernam cipher is one (just one random number is used for converting plain to cipher text.)
- The length of the key will be equal to the length of the plain text.
- It is an example of a **symmetric cryptosystem.**

Security Analysis

- Even when using a PRNG for generating the random numbers if the seed is long enough a sufficiently random number can be generated which leave the cryptosystem as practically secure rather than perfectly secure. (a random number with no statistical bias).
- But **generally practically secure** is more than sufficient for protection of data.
- An important condition for chaos is also present.

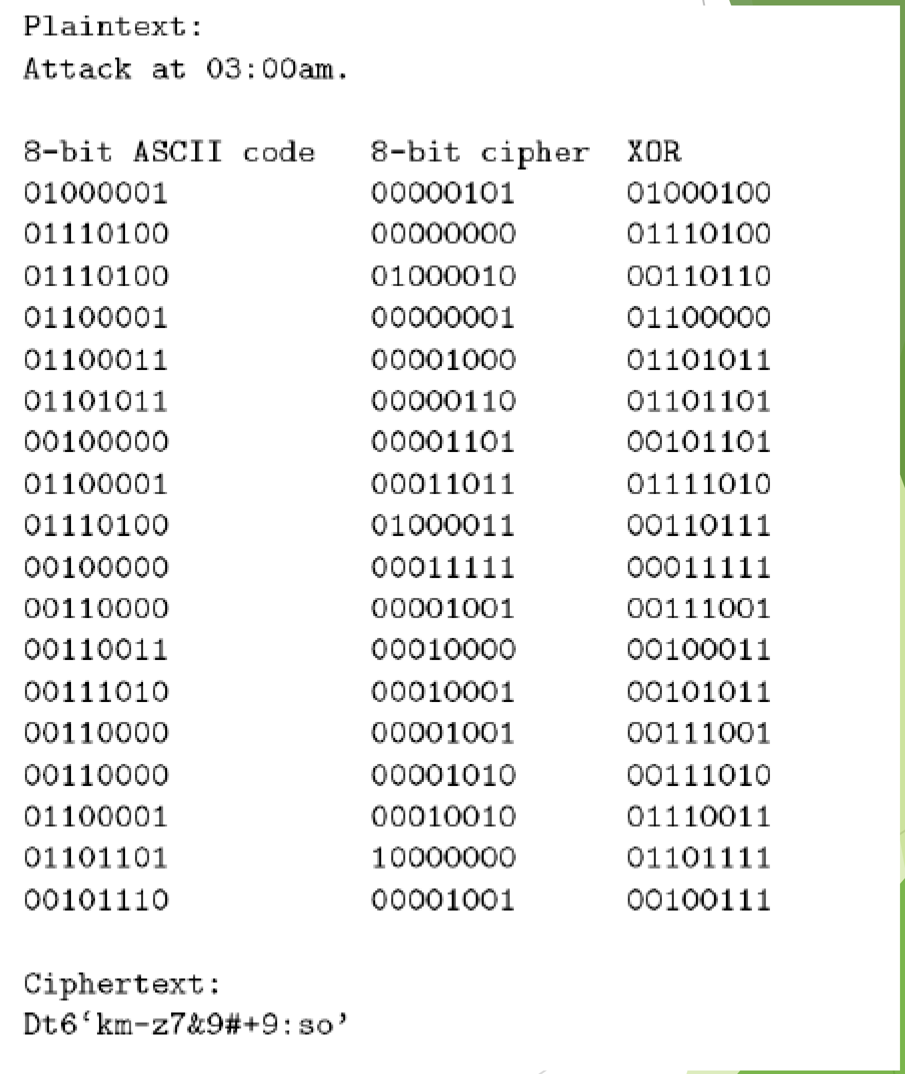
1. Random number with no statistical bias(maximum confusion)
2. Ultrasensitivity to the initial seed K (maximum diffusion.)

Chaos is beautifully manifested fundamentally in the pseudo random number generation algorithm which carries over to this very simple but strong and efficient cryptosystem.

The example show the action of the Vernam cipher.

Everything depends upon the random number generator.

Change the seed, get a different random number and hence a different cipher text.



Plaintext:
Attack at 03:00am.

8-bit ASCII code	8-bit cipher	XOR
01000001	00000101	01000100
01110100	00000000	01110100
01110100	01000010	00110110
01100001	00000001	01100000
01100011	00001000	01101011
01101011	00000110	01101101
00100000	00001101	00101101
01100001	00011011	01111010
01110100	01000011	00110111
00100000	00011111	00011111
00110000	00001001	00111001
00110011	00010000	00100011
00111010	00010001	00101011
00110000	00001001	00111001
00110000	00001010	00111010
01100001	00010010	01110011
01101101	10000000	01101111
00101110	00001001	00100111

Ciphertext:
Dt6‘km-z7&9#+9:so’

Courtesy : Prof. JM Blackledge

An Improvisation

- The PRNG described earlier shows the problems of periodicity.
- After a large number of cycles the seed used initially is re-used. This leads to periodicity and finite but long cycle lengths of the random numbers
- Something better can be done by changing from periodic to aperiodic (though it will still be a PRNG)
- This is the problem which is addressed by CHAOS

A Chaotic Map (As a PRNG)

- As mentioned earlier of the presence of chaos in the inherent process of random number generation, here we aim to change the periodicity of the random number generated.
- Use of one dimensional maps to generate random numbers proves to be the method for doing so.
- The random number generated depends not only on the initial number generated but also on the parameters involved in the one dimensional map.
- Care must be taken to choose the parameters where the map would show chaos.
- Actual implementation of this method and its security will be discussed later.

Cryptographical Perspective

1. The key space for the cryptosystem has increase from just one (the one random number) to N+1 where N is the number of parameters of the one dimensional map.
2. Comparing to conventional random number generators, the numbers generated, the 1D map generator is infinite, aperiodic and uncorrelated.

The increase in key space is significant making the cryptanalysts job more difficult. Now he has to not only worry about the initial seed but also about the parameter space. **This can also help in multiagorithmicity. Use two or more more algorithms in conjunction and in a periodic manner to further complicate the random number generation.**

The Logistic Map

- To this end a logistic map can be used as a one dimensional map as described.
- The system show chaotic regime for the parameter values of r>3.57 but used value is r=4.
- So choosing a value of r while avoiding the gaps in the middle (for saddle node bifurcations) and maintaining a closed state phase space values between 0 and 1 can lead to a long number of iterates leading to a random number formation.

Problems with this method

- Though this method scores much higher than the earlier implementation, it is not completely devoid of its own drawbacks.

1. Truncation error in computers :

i. Single precision (32 bits) – on an average 5000 iterations

ii. Double precision (64 bits) – on an average 10^9 iterations

2. Correlations between the successive iterates. (Will be mentioned briefly in Simple and Advanced ciphers .)

Conclusion

- Chaos is beautifully manifested fundamentally in the pseudo random number generation algorithm which carries over to this very simple but strong and efficient cryptosystem
- **The entire cryptosystem breaks if the initial seed is known.** The system is completely deterministic.
- Since we using binary or in other cases modulo the **entire ciphertext space is bounded** (another important condition for chaos to be shown.
- Chaos is entwined closely with the cryptographic necessity of **maximum confusion + diffusion**

Summary

- As a conclusion to the application of chaos for random number generation, it is important to note the **way chaos maps itself to cryptographic demands.**
- The details of the mapping will be highlighted in due course.

Chaos based ciphers

Chaotic Maps

Introduction

- Chaos- best known as sensitivity to initial conditions exhibited by dynamical systems described by differential equations or iterated mappings.
- We will be aiming to present the connection between these two fields theoretically and practically through the implementation of two simple ciphers.
- Symmetric Key cryptosystems

Chaos Dynamics

- Mainly iterated mappings will be used
- Elements of the orbit determine chaotic behavior.
- The system must be ergodic and have a continuous invariant density.
- Topological mixing (homogenous invariant measure)
- Positive Lyapunov exponents
- Attractors
- Robust chaos
- Topological transitivity

Mapping chaos to Cryptography

Properties	Chaos theory	Cryptography
Sensitive dependence on initial conditions	Characteristic chaos concept	Key dependent confusion and diffusion
Discrete time	Time based iteration in case of chaotic maps	Time based encryption rounds.
The security manifesto	The system parameters and the initial condition	The cryptographic keys (the number of keys and how each key adds to the security)
Type	True chaotic system	Pseudo chaotic system
Type of system	Deterministic	Deteministic

Chaos and Pseudo Chaos

- True chaos occurs in the chaotic maps and is successfully seen in the Differential equations as shown by Lorenz.
- Pseudo chaos is the best we can get from chaos for application in cryptography.
- This Pseudo chaotic nature which is the limit to which we can tap the true potential of chaos is due to our limitations and not due to the nature of the chaotic maps which are going to be used for encryption.
- Floating Point approximation is the major factor contributing to this/

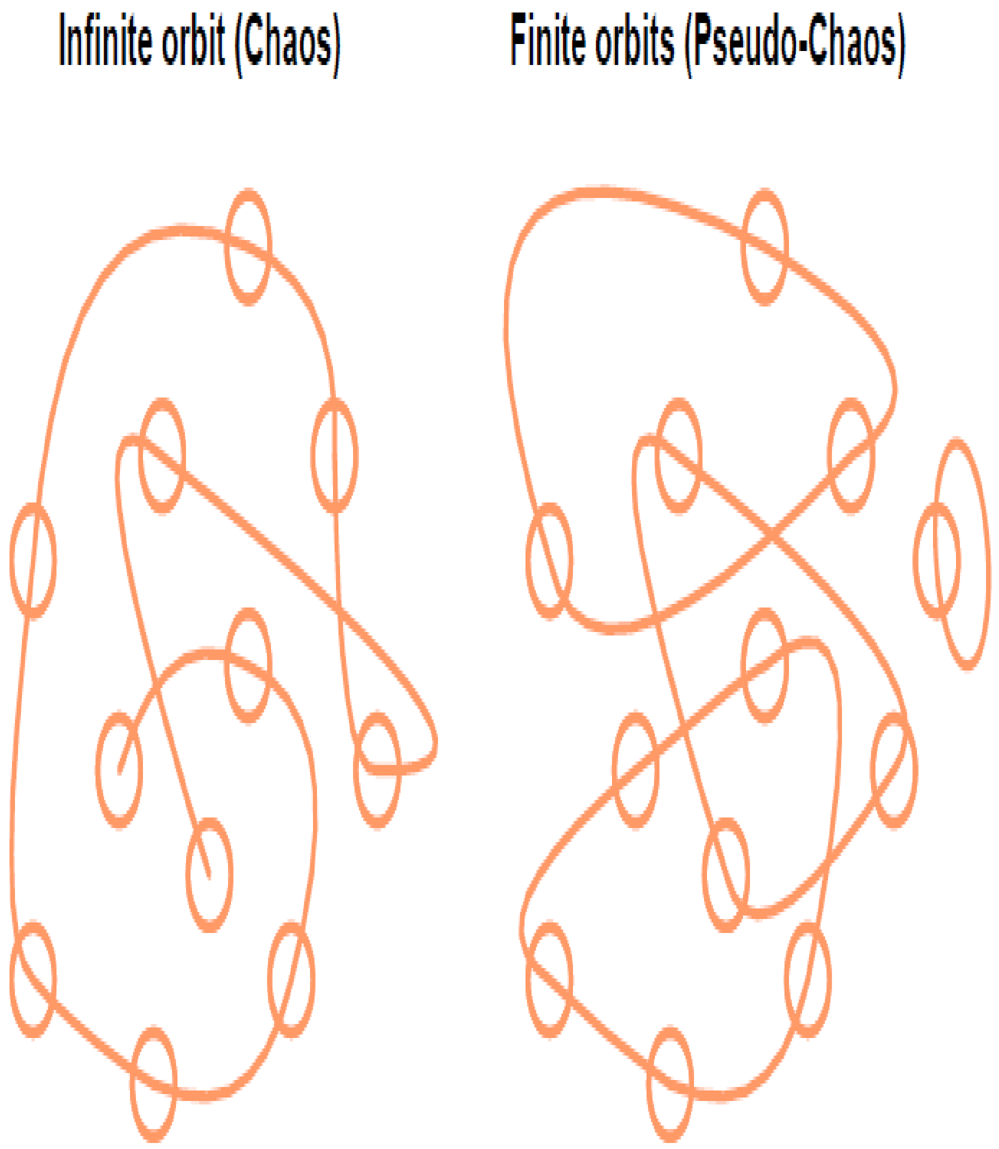
Chaos and Pseudo Chaos

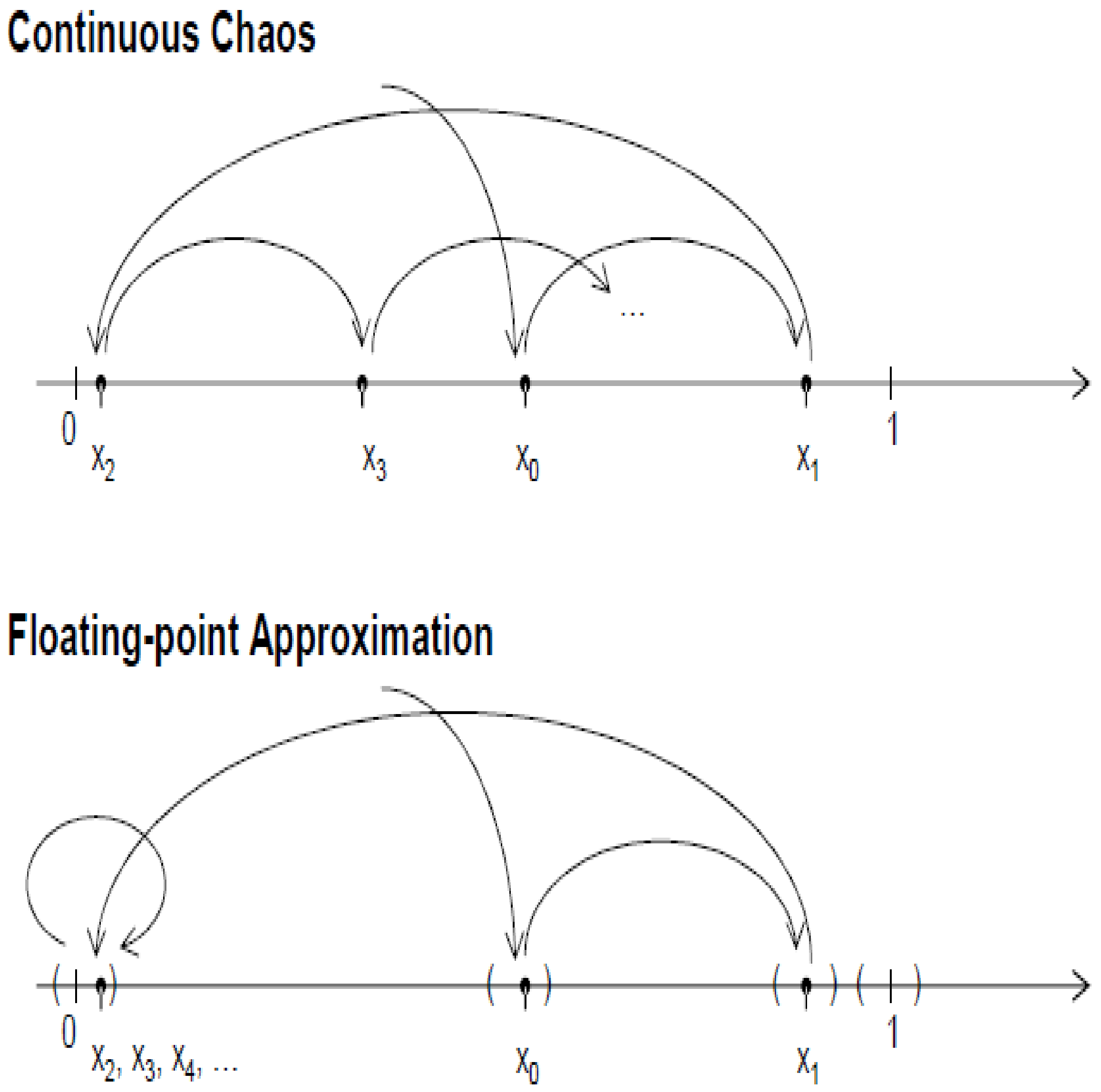
Chaos

- Infinite number of states
- Over the real numbers
- Completely chaotic
- Infinite Orbits

Pseudo chaos

- Finite number of states
- Limit on precision
- Low cycle lengths
- Finite Orbits


Infinite orbit (Chaos)
Finite orbits (Pseudo-Chaos)


Continuous Chaos
0
1
x_2
x_3
x_0
x_1
...
Floating-point Approximation
0
1
$x_2, x_3, x_4, \ldots$
x_0
x_1

Chaotic Cipher Design

- Use space discretized versions of chaotic maps that approximate real valued systems
- Discretization is necessary (why?)
- Must limit the numbers we can represent within an interval over a given set of numbers.
- Performance
- Approximation for chaotic maps is required
- Transformations should generate values far apart in Euclidian space and also in bit values.

Cipher Evaluation

- Measure of entropy of the underlying chaotic maps.

1. Modify the order of the key. (Entropy α log_2k)
2. Initial conditions- Chaos is attained and its properties expressed
3. Randomizing not only the initial conditions but also the number of iterations
4. Size of data blocks-

a) Difficulty in decrypting

b) Time consuming

5. Set of elements are over the real domain

Security Analysis

- At a theoretical level cryptographic primitives are deemed if they possess the following characteristics

1. Randomness increasing
2. Computationally unpredictable
3. The cryptographical primitive must increase the entropy of the system over which it operates
4. Successful cryptanalytic attack depends on the statistical attributes of the cipher.
5. Randomness increasing is a necessary condition for security but its not sufficient.
6. The most important signature of security lies in “randomness” and not in in its increase.

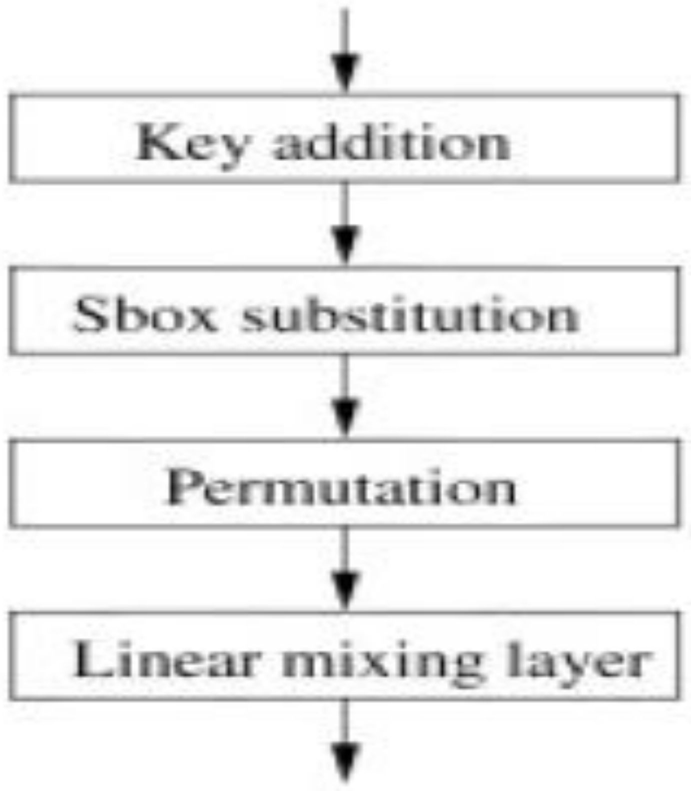


Fig. 4. A typical symmetric key block cipher structure based on the substitution-permutation network design. It contains the four fundamental operations of key addition, S-box substitution, permutation, and linear mixing.

Conclusion

- The chaotic maps are one way to implement chaos in the field of security analysis. As can be seen the method is not at all efficient. Much more simple, efficient and robust algorithms exist for encryption.
- A major setback is that the security properties are not thorough nor are they provable given the complexity involved in the algorithms.
- But these algorithms prove the immense scope of chaos in this field.
- Specific case studies have been performed by us which will be taken up by the other members. Strict adherence to the three main points of design, evaluation and security analysis have been made.

Case Study

1. Simple Cipher
2. Advanced Cipher
3. Baptista Method
4. Rabbit Cipher

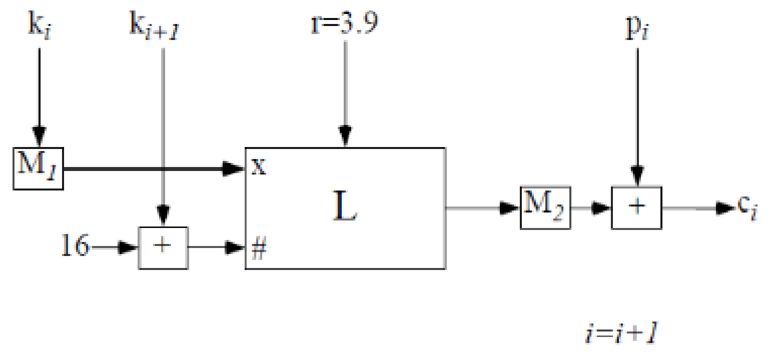
Simple Cipher

- Proposed by Roskin and Casper.
- It is a Block cipher based on unpredictability of logistic map.
- Symmetric Key Encryption Algorithm.

Simple Cipher

- Keys
  1. Array of 50 randomly generated numbers
  2. Initial condition
  3. Parameter of the logistic map

Simple Cipher – Block Diagram

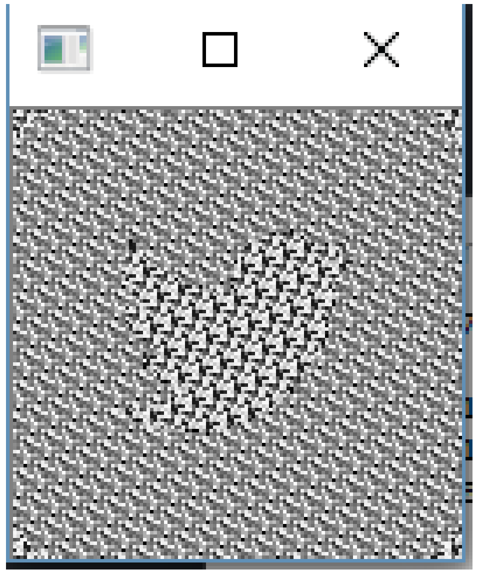


- Encryption
  - Keys mapped from [0, 256] to range [0, 1]
  - The key is iterated in Block L
  - Output mapped from [0, 1] to [0, 256]
  - Final output added to plain text
- Decryption
  - Given the keys and cipher text, receiver can completely determine the message.

Simple Cipher – Test Case

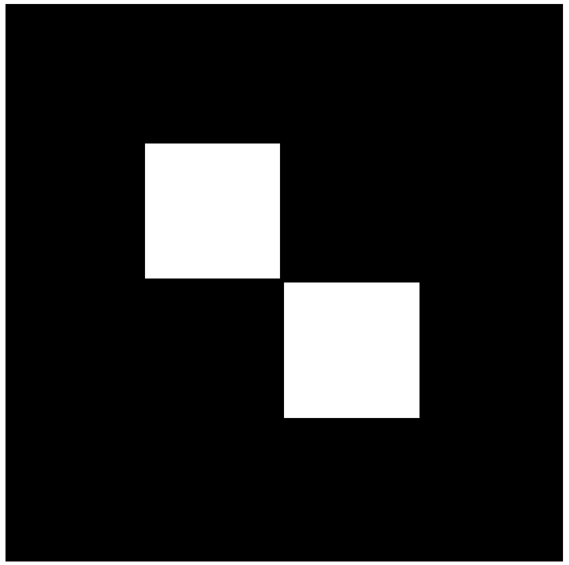
Input Image

Encrypted Image

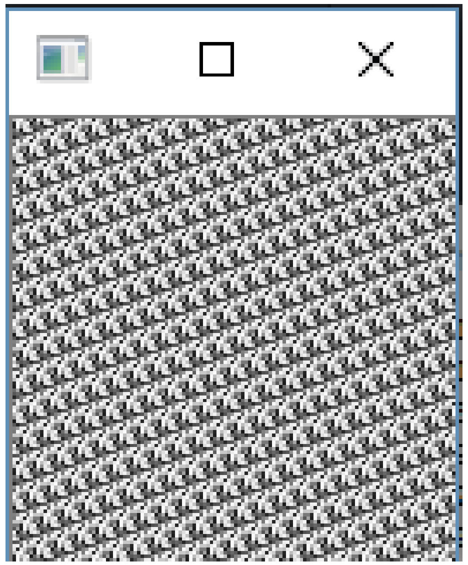


Simple Cipher – Test case

Input Image



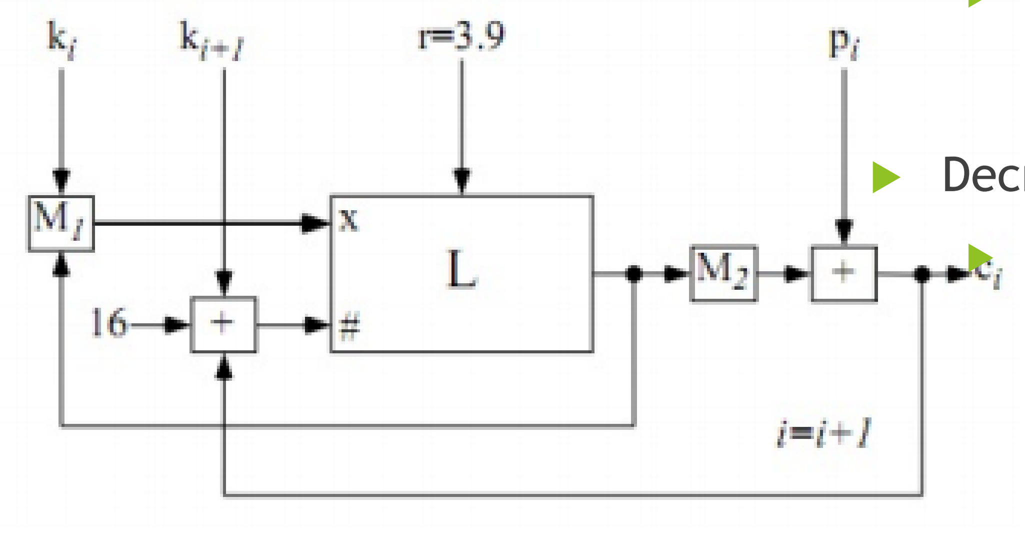
Encrypted Image



Advanced Cipher

- Proposed by Roskin and Casper.
- Extended version of simple cipher with a feedback to reduce the periodicity in the cipher text.

Advanced Cipher – Block Diagram

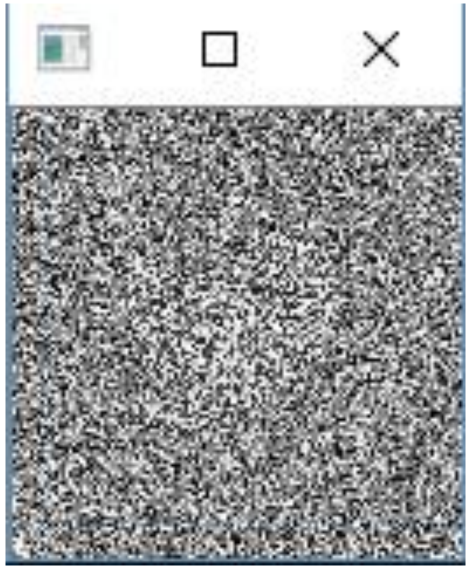


- Encryption
  - Similar to simple cipher but with a additional feedback in the initial value and the number of iteration
- Decryption
  - Similar to simple cipher

Advanced Cipher – Test Case

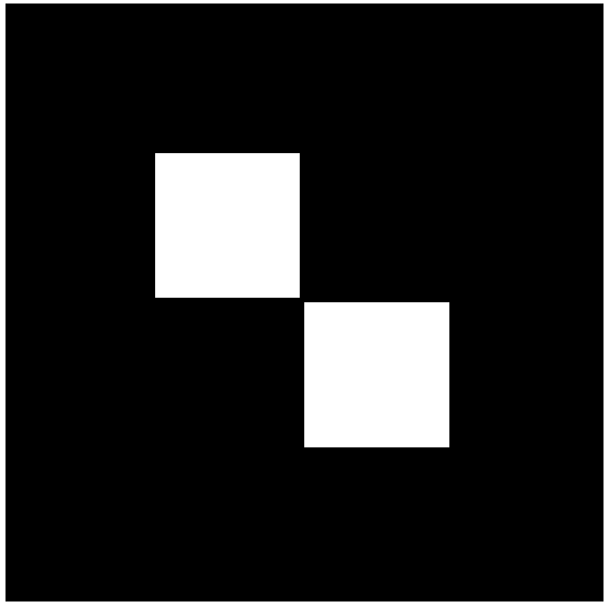
Input Image

Encrypted Image

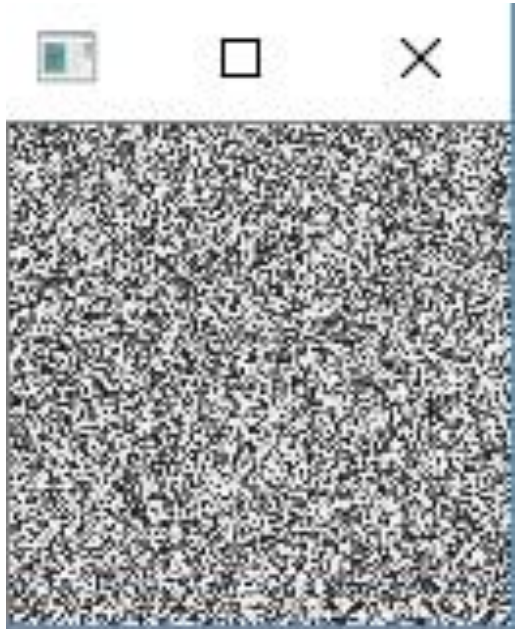


Advanced Cipher – Test Case

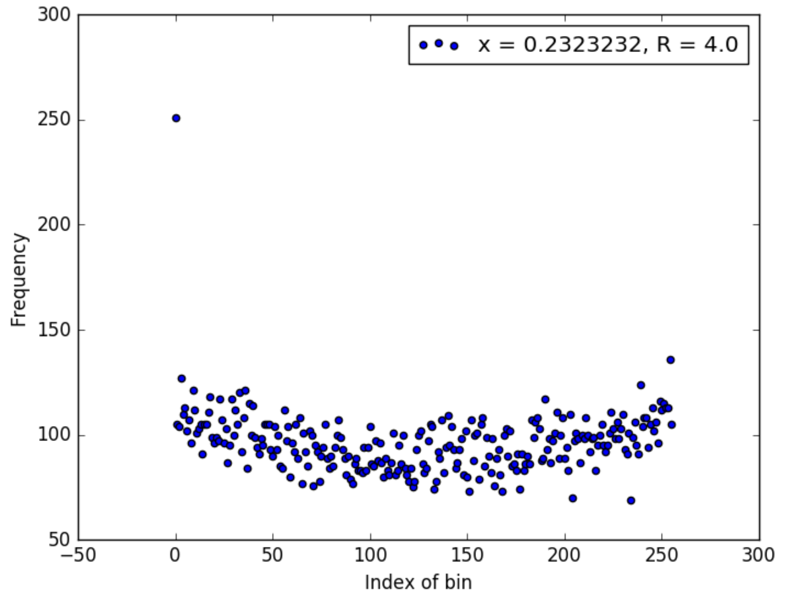
Input Image

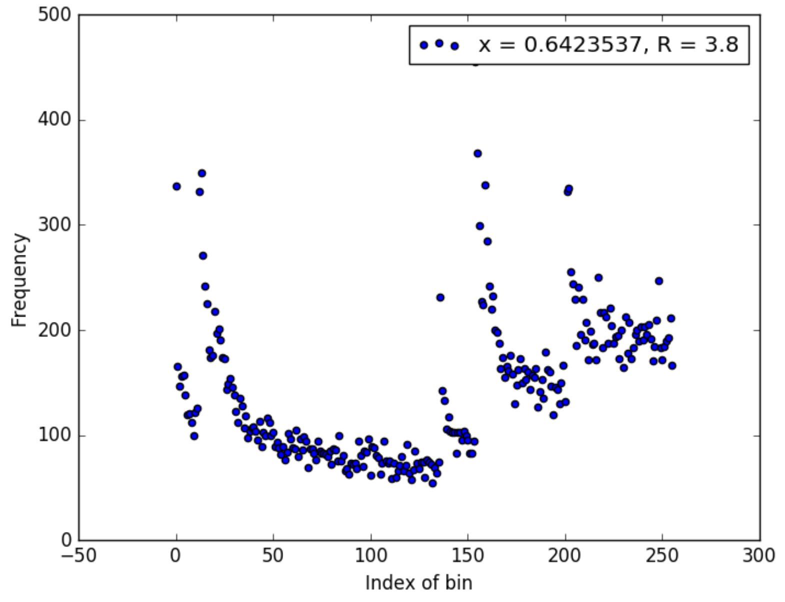


Encrypted Image

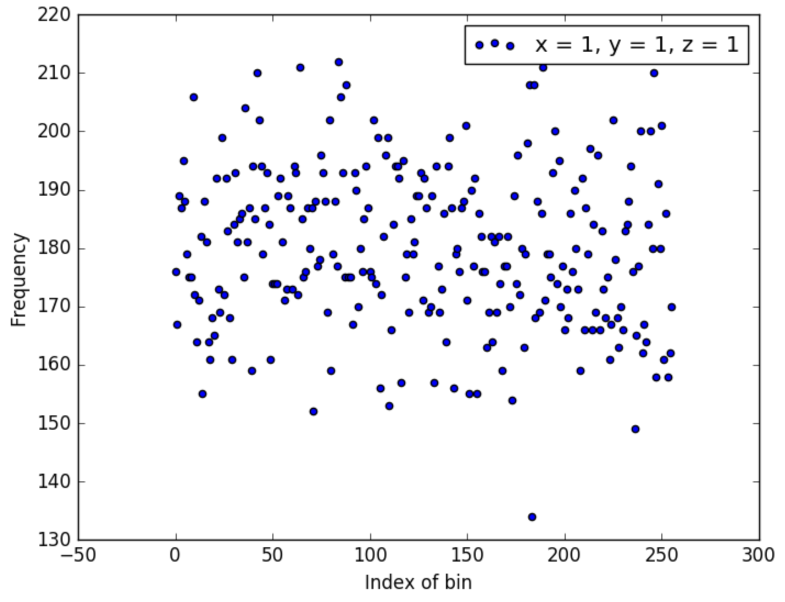


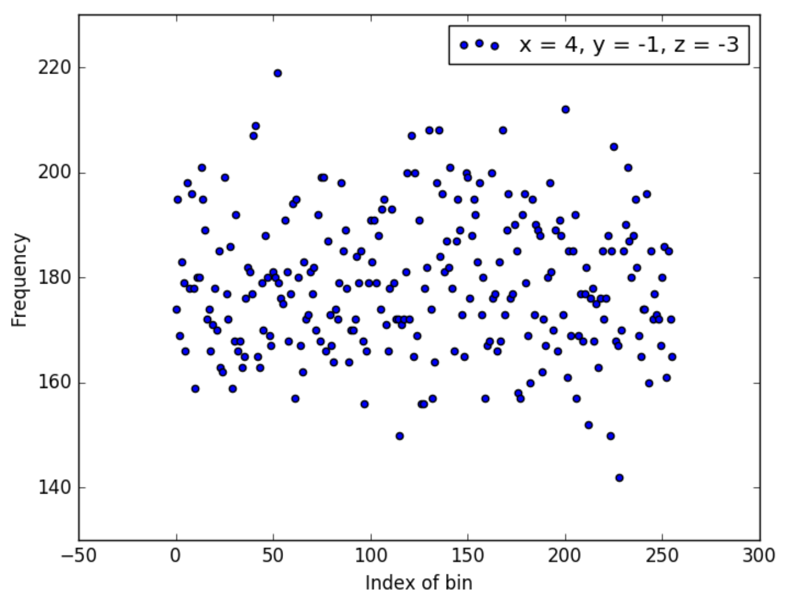
Baptista Method – Logistic Map





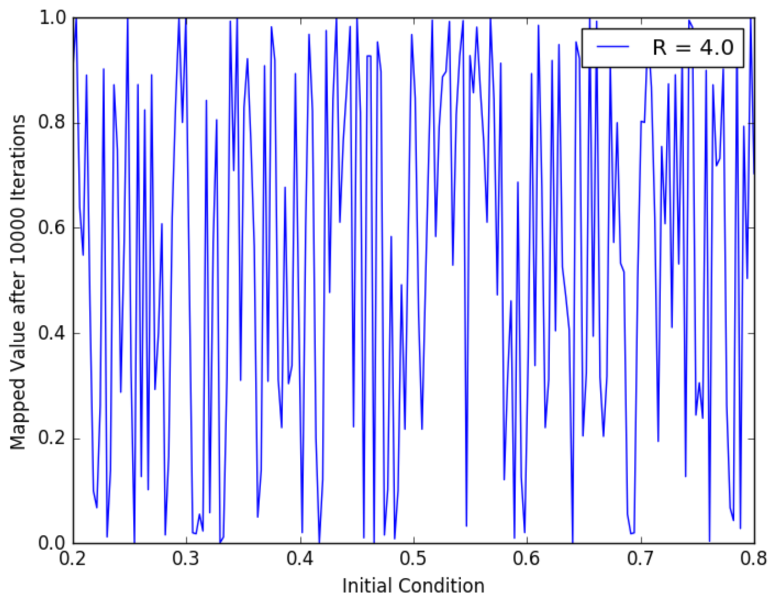
Baptista Method – Lorenz System



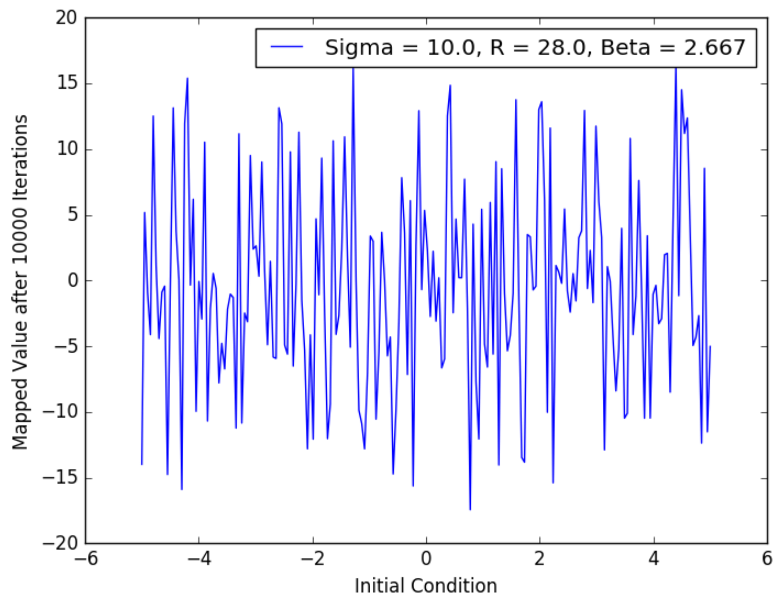


Baptista Method

Logistic Map



Lorenz System



Baptista Method

- It is a block cipher.
- Symmetric Key cipher.

Baptista Method

- Keys
  - Using Logistic Map
    1. Initial value or the seed of the map
    2. Parameter of logistic map
  - Using Lorenz System
    1. Initial value or the seed of the map
    2. Parameter of Lorenz System
    3. Variable which is used for encryption i.e. X or Y or Z
    4. Bound of the phase space used

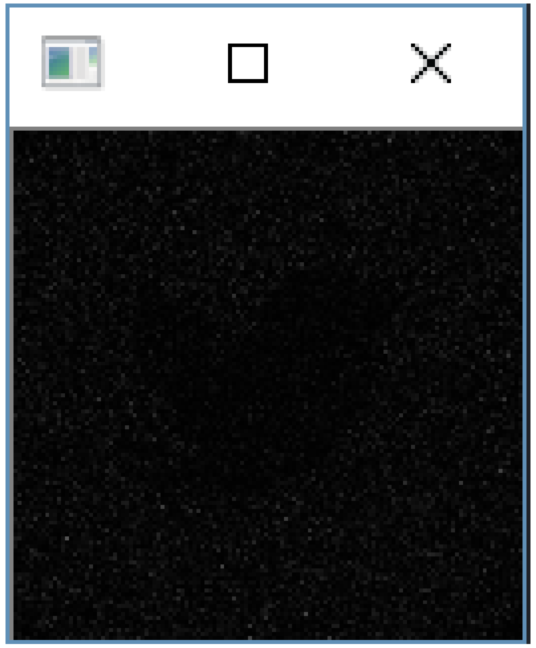
Baptista Method

- Encryption
  - Phase Space is divided into 256 bins.
  - Bin index is the value for the particular pixel / plaintext.
  - Cipher text = Number of iterations to reach a bins
- Decryption
  - The receiver has both the cipher text and the key.
  - The receiver can determine the plain text using the number of iteration.

Baptista Method – Test Case (Logistic Map)

Input Image

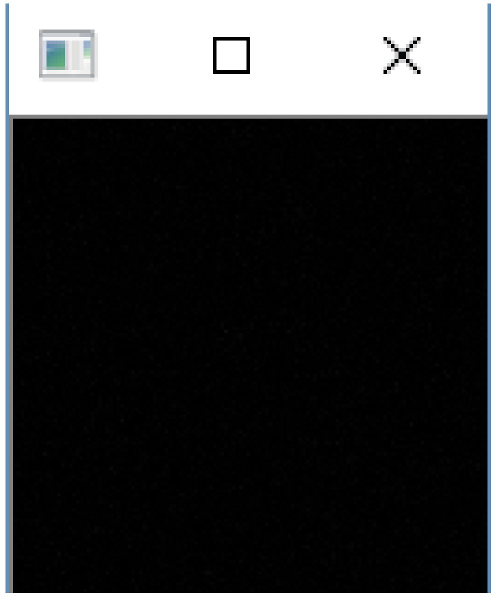
Encrypted Image



Baptista Method – Test Case (Lorenz System)

Input Image

Encrypted Image



Rabbit Cipher

- Inspired by “Random” behaviour of the chaotic maps.
- It takes advantage of the random-like properties of real valued chaotic maps & at the same time secures optimal cryptographic properties when discretizing them.
- Designed to work with 128 - bit data sizes, as both the key and the output are 128 – bit in length.
- Internal data structure consists of 8 state variables and 8 counters.
- Rabbit Cipher takes a 128-bit key as input and generates for each iteration an output block 128 pseudo random bits from a combination of internal state bits

Algorithm

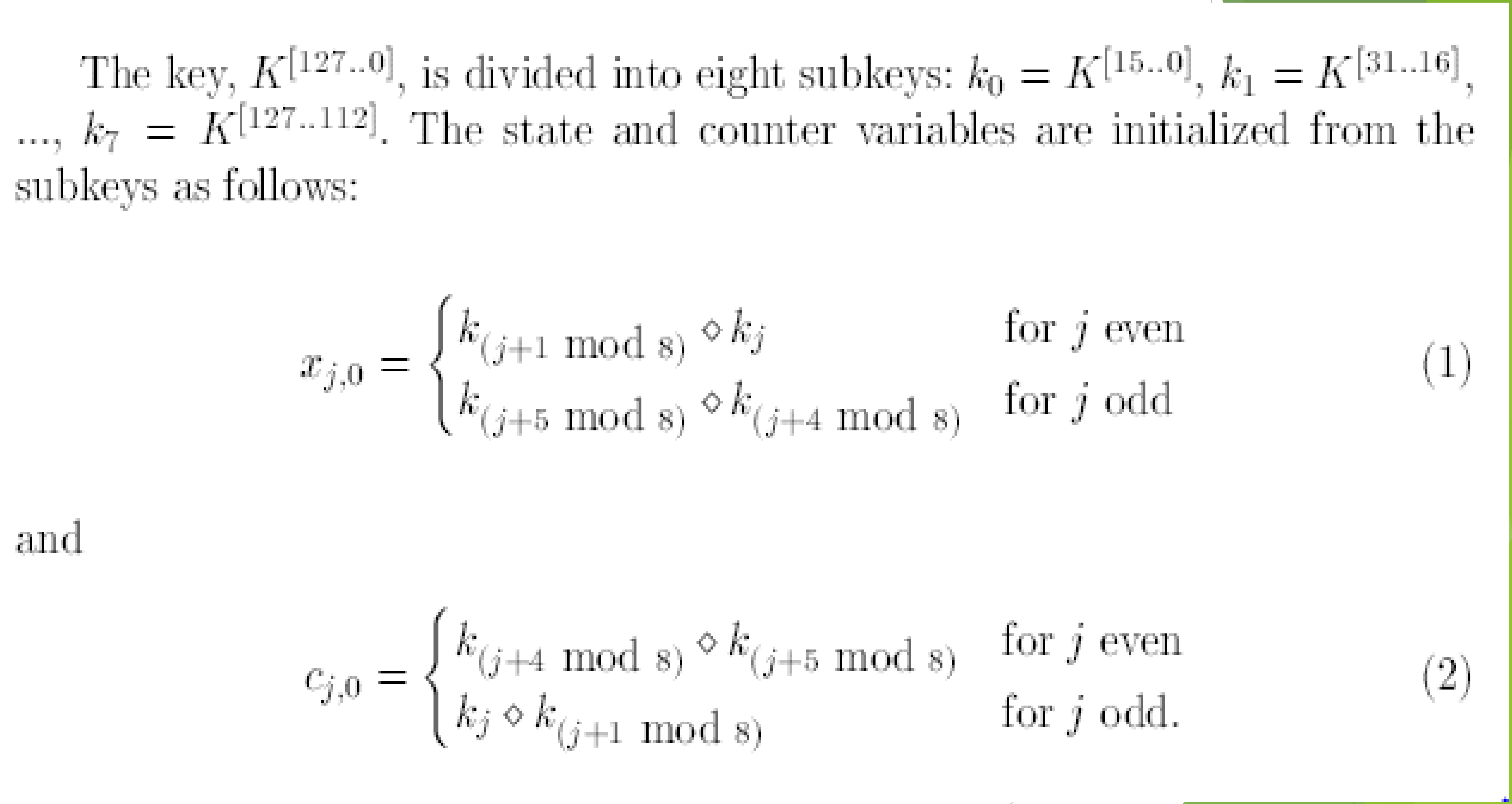
- The algorithm for the cipher can be broken down into the
- following four main components :
- 1) Key setup
- 2) Next state (round) function
- 3) Counter system
- 4) Extraction scheme

Cipher Evaluation

- The design of this system is initiated by constructing a chaotic system of non-linear maps, then the system is restricted to fixed-point valued, i.e. each variable is represented by an integer type number, where a virtual decimal point is introduced manually.
- What is unique is that instead of operating over the domain of real numbers (meaning that implementation requires floating point operations), the domain is scaled up by 2^{32} to translate decimal numbers into integers. This key modification to the phase space maintains the same precision as in the real interval while at the same time improving implementation performance.

Key Setup

- The size of Internal state is 513 bits divided between 8 32- bit counters & 32-bit state variables and one counter carry bit.
- Algorithm is initialized by expanding 128-bit key into both 8 state variables and the 8 counters.
- There is one-to-one correspondence between the between the key and the initial state variables $x_{j,0}$ and the initial counter $c_{j,0}$

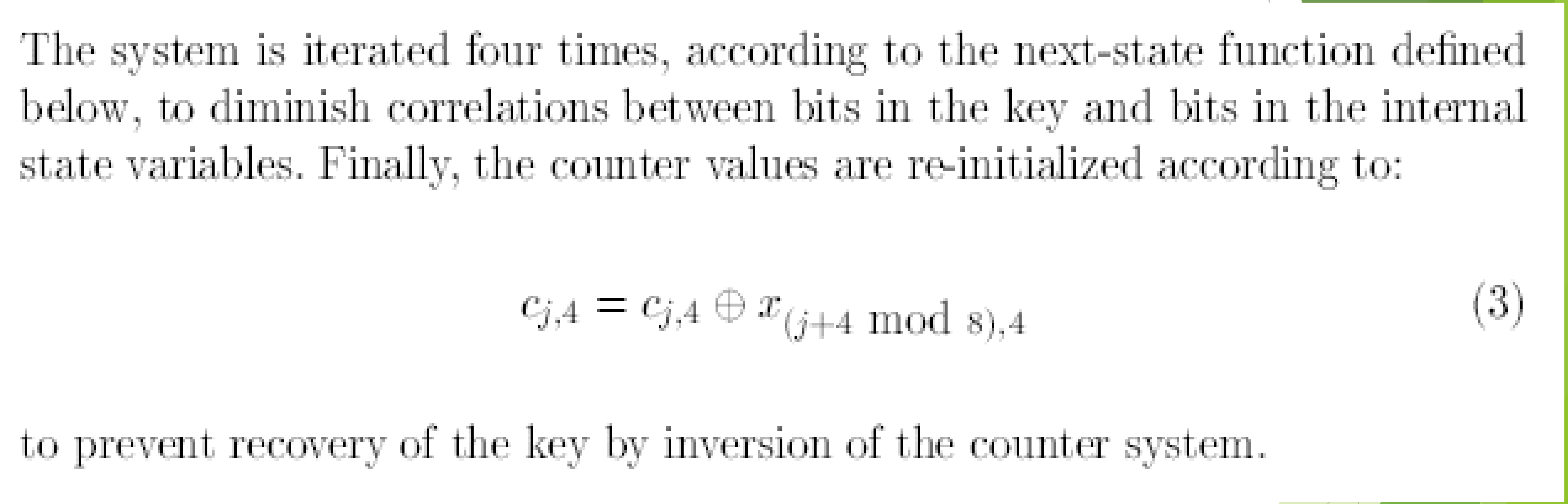


The key, $K^{[127..0]}$, is divided into eight subkeys: $k_0 = K^{[15..0]}$, $k_1 = K^{[31..16]}$, ..., $k_7 = K^{[127..112]}$. The state and counter variables are initialized from the subkeys as follows:

$$x_{j,0} = \begin{cases} k_{(j+1 \bmod 8)} \diamond k_j & \text{for } j \text{ even} \\ k_{(j+5 \bmod 8)} \diamond k_{(j+4 \bmod 8)} & \text{for } j \text{ odd} \end{cases} \qquad (1)$$

and

$$c_{j,0} = \begin{cases} k_{(j+4 \bmod 8)} \diamond k_{(j+5 \bmod 8)} & \text{for } j \text{ even} \\ k_j \diamond k_{(j+1 \bmod 8)} & \text{for } j \text{ odd.} \end{cases} \qquad (2)$$



The system is iterated four times, according to the next-state function defined below, to diminish correlations between bits in the key and bits in the internal state variables. Finally, the counter values are re-initialized according to:

$$c_{j,4} = c_{j,4} \oplus x_{(j+4 \bmod 8),4} \tag{3}$$

to prevent recovery of the key by inversion of the counter system.

Next State Function

$$\begin{aligned}
x_{0,i+1} &= g_{0,i} + (g_{7,i} \lll 16) + (g_{6,i} \lll 16) \\
x_{1,i+1} &= g_{1,i} + (g_{0,i} \lll 8) + g_{7,i} \\
x_{2,i+1} &= g_{2,i} + (g_{1,i} \lll 16) + (g_{0,i} \lll 16) \\
x_{3,i+1} &= g_{3,i} + (g_{2,i} \lll 8) + g_{1,i} \\
x_{4,i+1} &= g_{4,i} + (g_{3,i} \lll 16) + (g_{2,i} \lll 16) \\
x_{5,i+1} &= g_{5,i} + (g_{4,i} \lll 8) + g_{3,i} \\
x_{6,i+1} &= g_{6,i} + (g_{5,i} \lll 16) + (g_{4,i} \lll 16) \\
x_{7,i+1} &= g_{7,i} + (g_{6,i} \lll 8) + g_{5,i}
\end{aligned}$$

$$g_{j,i} = \left((x_{j,i} + c_{j,i})^2 \oplus ((x_{j,i} + c_{j,i})^2 \gg 32)\right) \bmod 2^{32}$$

Counter System

$$\begin{aligned}
c_{0,i+1} &= c_{0,i} + a_0 + \phi_{7,i} \mod 2^{32} \\
c_{1,i+1} &= c_{1,i} + a_1 + \phi_{0,i+1} \mod 2^{32} \\
c_{2,i+1} &= c_{2,i} + a_2 + \phi_{1,i+1} \mod 2^{32} \\
c_{3,i+1} &= c_{3,i} + a_3 + \phi_{2,i+1} \mod 2^{32} \\
c_{4,i+1} &= c_{4,i} + a_4 + \phi_{3,i+1} \mod 2^{32} \\
c_{5,i+1} &= c_{5,i} + a_5 + \phi_{4,i+1} \mod 2^{32} \\
c_{6,i+1} &= c_{6,i} + a_6 + \phi_{5,i+1} \mod 2^{32} \\
c_{7,i+1} &= c_{7,i} + a_7 + \phi_{6,i+1} \mod 2^{32}
\end{aligned}$$

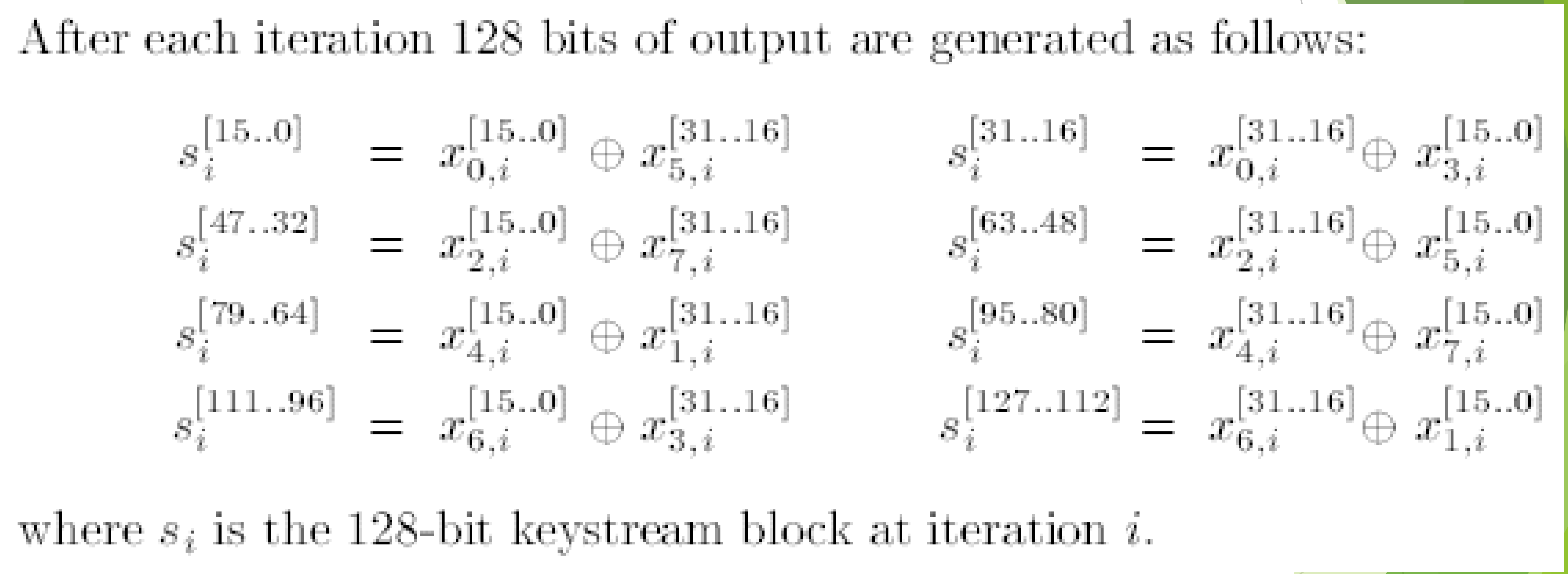
where the counter carry bit, $\phi_{j,i+1}$, is given by

$$\phi_{j,i+1} = \begin{cases} 1 & \text{if } c_{0,i} + a_0 + \phi_{7,i} \geq 2^{32} \wedge j = 0 \\ 1 & \text{if } c_{j,i} + a_j + \phi_{j-1,i+1} \geq 2^{32} \wedge j > 0 \\ 0 & \text{otherwise.} \end{cases}$$

Furthermore, the a_j constants are defined as:

$$\begin{aligned} a_0 &= \text{0x4D34D34D} & a_1 &= \text{0xD34D34D3} \\ a_2 &= \text{0x34D34D34} & a_3 &= \text{0x4D34D34D} \\ a_4 &= \text{0xD34D34D3} & a_5 &= \text{0x34D34D34} \\ a_6 &= \text{0x4D34D34D} & a_7 &= \text{0xD34D34D3}. \end{aligned}$$

Extraction Scheme



After each iteration 128 bits of output are generated as follows:

$$\begin{aligned}
s_i^{[15..0]} &= x_{0,i}^{[15..0]} \oplus x_{5,i}^{[31..16]} & s_i^{[31..16]} &= x_{0,i}^{[31..16]} \oplus x_{3,i}^{[15..0]} \\
s_i^{[47..32]} &= x_{2,i}^{[15..0]} \oplus x_{7,i}^{[31..16]} & s_i^{[63..48]} &= x_{2,i}^{[31..16]} \oplus x_{5,i}^{[15..0]} \\
s_i^{[79..64]} &= x_{4,i}^{[15..0]} \oplus x_{1,i}^{[31..16]} & s_i^{[95..80]} &= x_{4,i}^{[31..16]} \oplus x_{7,i}^{[15..0]} \\
s_i^{[111..96]} &= x_{6,i}^{[15..0]} \oplus x_{3,i}^{[31..16]} & s_i^{[127..112]} &= x_{6,i}^{[31..16]} \oplus x_{1,i}^{[15..0]}
\end{aligned}$$

where s_i is the 128-bit keystream block at iteration i.

Encryption/Decryption Scheme

- Extracted bits are XOR'ed with plaintext/ciphertext to encrypt/decrypt

$$c_i = p_i \oplus s_i,$$
$$p_i = c_i \oplus s_i,$$

- Where c_i and p_i denote the ith vely.

Security Analysis

- One-to-one correspondence between the key, the state and the counter prevents key redundancy. It also distributes the key bits in an optimal way to prepare for the the system iteration.
- After one iteration of the next state function, each key bit has affected all eight state variables.
- The key expansion scheme ensures that after two iterations of the next state function, all state bits are affected by all key bits with a measured probability of 0.5. A safety margin is provided by iterating the system 4 times.
- The counter modification makes it hard to recover the key by inverting the counter system, as this would require additional knowledge of the state variables. It also destroys the one-to-one correspondence between key and counter.
- All the output bytes of the next – state function depend on the maximal 12 input bytes. Consequently, removing any of those input bytes will result in nearly maximal entropy of error of the output bytes.

Resulting Attacks

- Attack on the Key Step Function

1) Counter bits and state bits depend strongly and highly non-linearly on the key bits. This makes attacks made on guessing parts of the key difficult.

2) Although counter modification destroys one-to-one correspondence between the key and counters, but the probability of two randomly selected keys to have the same counters is

$1/2^{128}$.

- Distinguishing and Correlation Attack

1) Attacker tries to distinguish a sequence generated by the cipher from a sequence of truly random numbers.

2) This type of attack is impossible because, in order to observe such a correlation, output from approximately 2^{114} iterations will have to be generated.

- Divide and Conquer Attack

1) Attack is feasible only if a fraction of state variable is known, in order to predict a significant fraction of output bit.

2) Attacker will need to guess each of the 192 bits of input correctly.

Random Numbers and their Analysis

- Randomness is the lack of pattern or predictability in events.

- A random sequence of events, symbols or steps has no order and does not follow an intelligible pattern or combination.

- A numeric sequence is said to be statistically random when it contains no recognizable patterns or regularities.
- Sequences such as the results of an ideal dice roll, or the digits of π exhibit statistical randomness.

- Statistical randomness does not necessarily imply "true" randomness, i.e., objective unpredictability.
- Pseudorandomness is sufficient for many uses, such as statistics, hence the name statistical randomness.

Generating Pseudorandom Numbers

- In our analysis, we used :
- 1. Lorenz System
- 2. Logistic Map
- 3. A Simple cipher using Baptista method
- 4. Advanced cipher using Baptista method

- to encrypt and decrypt an image.

Lorenz System

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{pmatrix} = \begin{pmatrix} -\sigma(x - y) \\ (r - z)x - y \\ xy - bz \end{pmatrix}$$

Logistic Map

- $X_n = r * X_{n-1} * (1 - X_{n-1})$

Simple cipher

- Based on a different implementation of logistic map

Advanced cipher

- Another, more complex implementation of logistic map.

Testing Pseudorandom numbers

- Diehard suite of tests:
- Designed to test for quality of random number generators

- Consists of the following tests:

- Birthday spacings: Choose random points on a large interval. The spacings between the points should be asymptotically exponentially distributed.

- Overlapping permutations: Analyze sequences of five consecutive random numbers. The 120 possible orderings should occur with statistically equal probability.

- Ranks of matrices: Select some number of bits from some number of random numbers to form a matrix over {0,1}, then determine the rank of the matrix. Count the ranks.

- Monkey tests: Treat sequences of some number of bits as "words". Count the overlapping words in a stream. The number of "words" that do not appear should follow a known distribution.

- Count the 1s: Count the 1 bits in each of either successive or chosen bytes. Convert the counts to "letters", and count the occurrences of five-letter "words" .

- Parking lot test: Randomly place unit circles in a 100 × 100 square. A circle is successfully parked if it does not overlap an existing successfully parked one. After 12,000 tries, the number of successfully parked circles should follow a certain normal distribution.

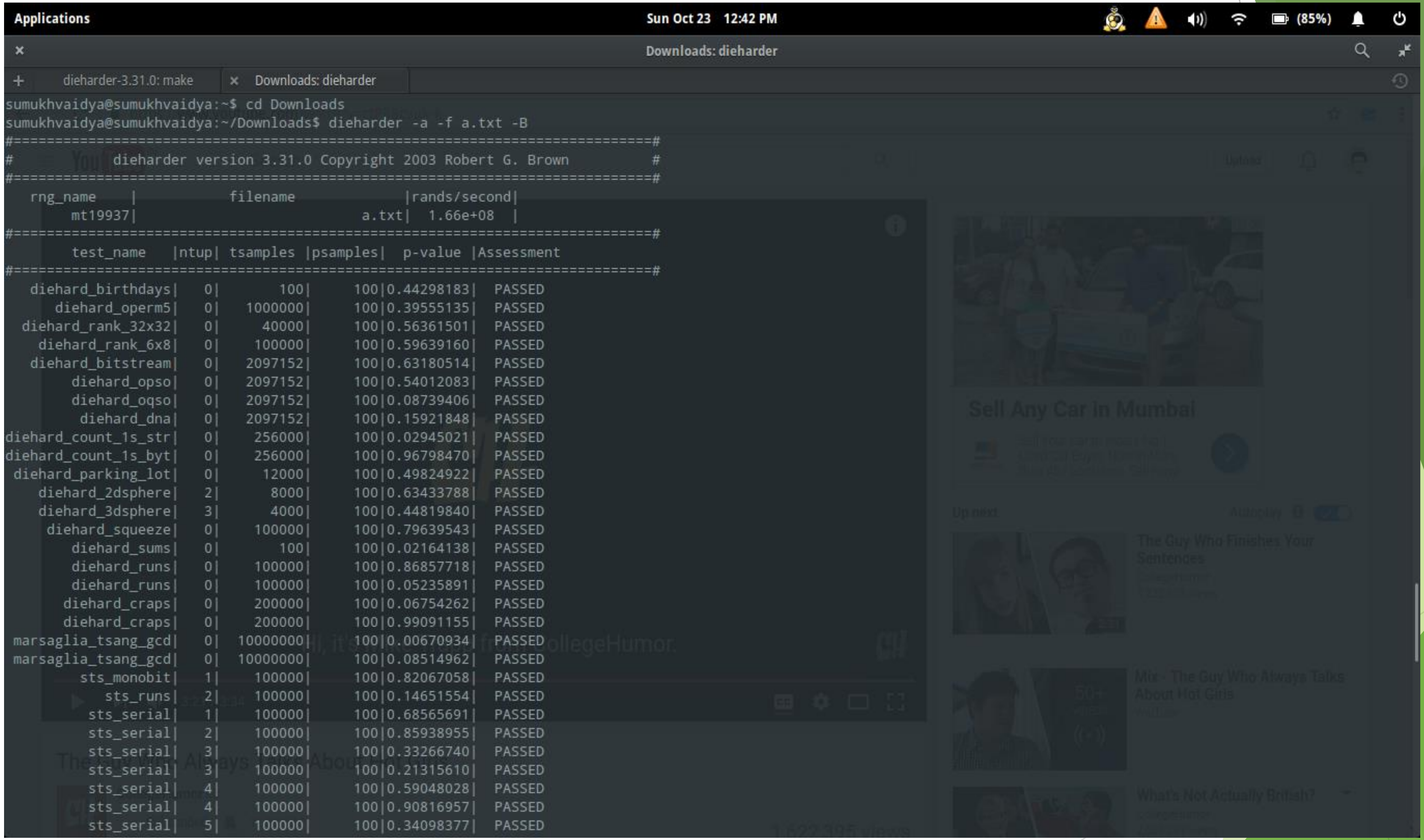
- Minimum distance test: Randomly place 8,000 points in a 10,000 × 10,000 square, then find the minimum distance between the pairs. The square of this distance should be exponentially distributed with a certain mean.
- Random spheres test: Randomly choose 4,000 points in a cube of edge 1,000. Center a sphere on each point, whose radius is the minimum distance to another point. The smallest sphere's volume should be exponentially distributed with a certain mean.

- The squeeze test: Multiply 2^31 by random floats on (0,1) until you reach 1. Repeat this 100,000 times. The number of floats needed to reach 1 should follow a certain distribution.

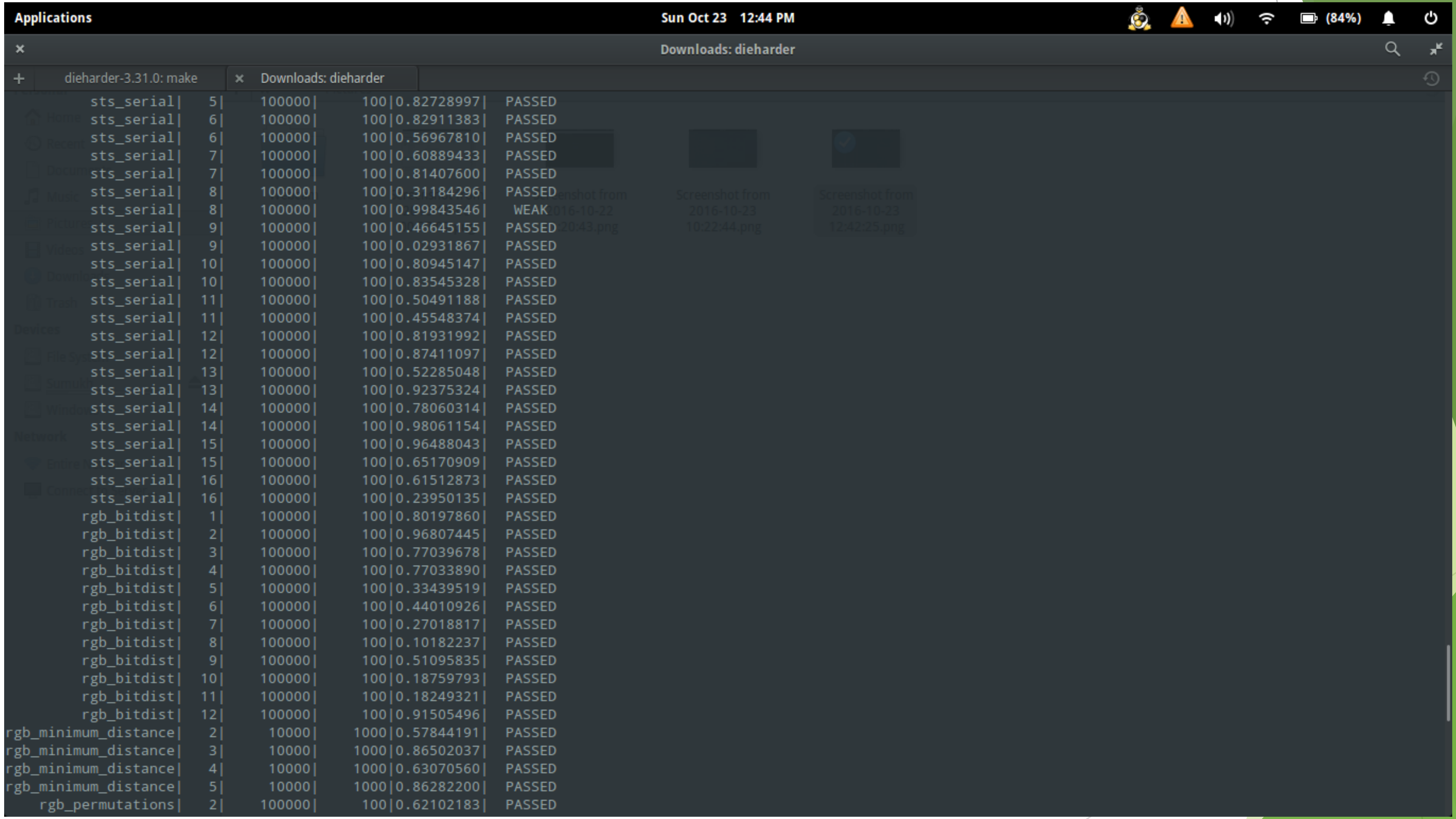
- Overlapping sums test: Generate a long sequence of random floats on (0,1). Add sequences of 100 consecutive floats. The sums should be normally distributed with characteristic mean and variance.

Results of Tests

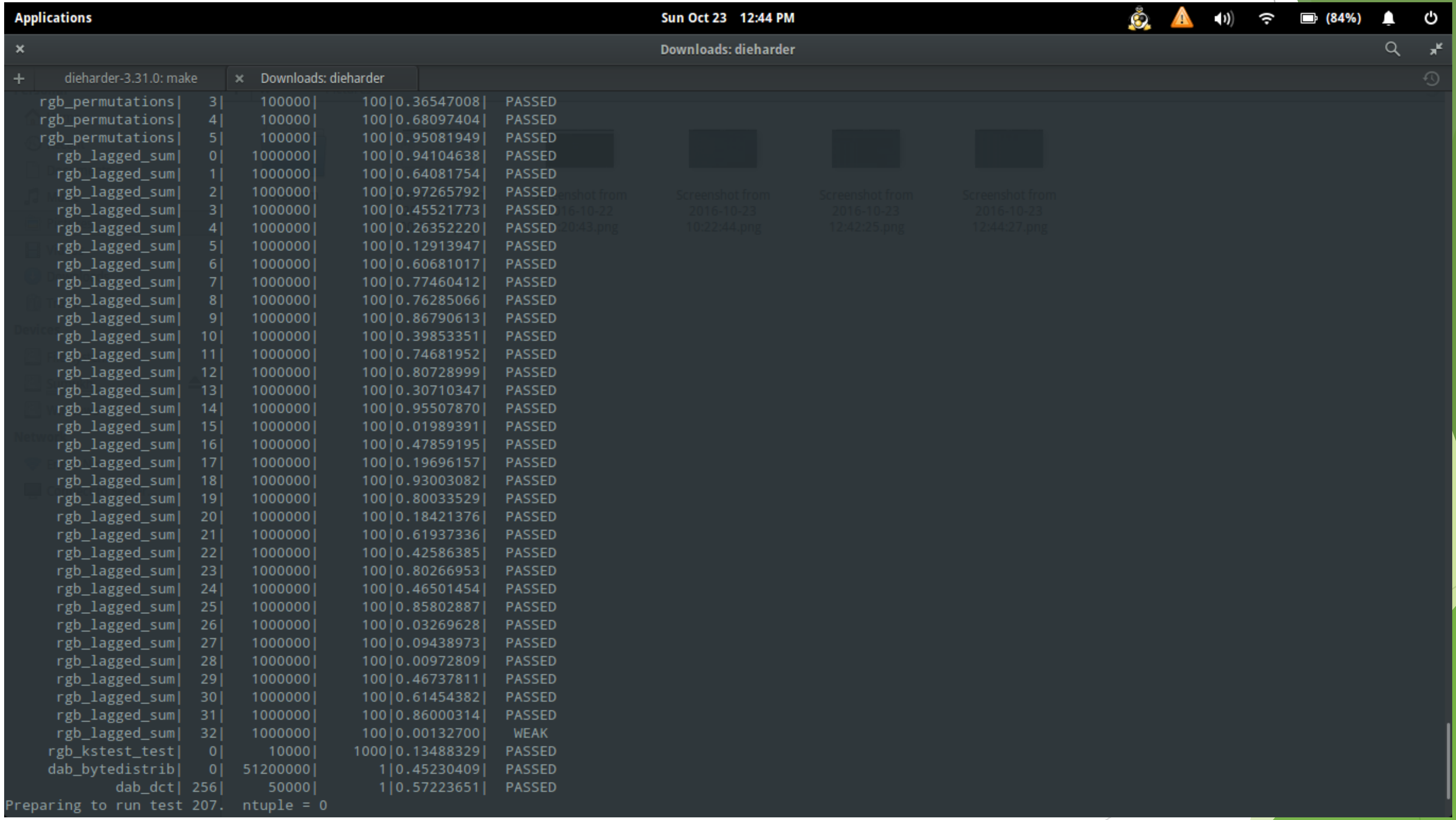
- Results for Logistic Map:



```
sumukhvaidya@sumukhvaidya:~$ cd Downloads
sumukhvaidya@sumukhvaidya:~/Downloads$ dieharder -a -f a.txt -B
#=============================================================================#
#            dieharder version 3.31.0 Copyright 2003 Robert G. Brown          #
#=============================================================================#
   rng_name    |           filename             |rands/second|
        mt19937|                           a.txt|  1.66e+08  |
#=============================================================================#
        test_name   |ntup| tsamples |psamples|  p-value |Assessment
#=============================================================================#
   diehard_birthdays|   0|       100|     100|0.44298183|  PASSED
      diehard_operm5|   0|   1000000|     100|0.39555135|  PASSED
  diehard_rank_32x32|   0|     40000|     100|0.56361501|  PASSED
    diehard_rank_6x8|   0|    100000|     100|0.59639160|  PASSED
   diehard_bitstream|   0|   2097152|     100|0.63180514|  PASSED
        diehard_opso|   0|   2097152|     100|0.54012083|  PASSED
        diehard_oqso|   0|   2097152|     100|0.08739406|  PASSED
         diehard_dna|   0|   2097152|     100|0.15921848|  PASSED
diehard_count_1s_str|   0|    256000|     100|0.02945021|  PASSED
diehard_count_1s_byt|   0|    256000|     100|0.96798470|  PASSED
 diehard_parking_lot|   0|     12000|     100|0.49824922|  PASSED
    diehard_2dsphere|   2|      8000|     100|0.63433788|  PASSED
    diehard_3dsphere|   3|      4000|     100|0.44819840|  PASSED
     diehard_squeeze|   0|    100000|     100|0.79639543|  PASSED
        diehard_sums|   0|       100|     100|0.02164138|  PASSED
        diehard_runs|   0|    100000|     100|0.86857718|  PASSED
        diehard_runs|   0|    100000|     100|0.05235891|  PASSED
       diehard_craps|   0|    200000|     100|0.06754262|  PASSED
       diehard_craps|   0|    200000|     100|0.99091155|  PASSED
 marsaglia_tsang_gcd|   0|  10000000|     100|0.00670934|  PASSED
 marsaglia_tsang_gcd|   0|  10000000|     100|0.08514962|  PASSED
         sts_monobit|   1|    100000|     100|0.82067058|  PASSED
            sts_runs|   2|    100000|     100|0.14651554|  PASSED
          sts_serial|   1|    100000|     100|0.68565691|  PASSED
          sts_serial|   2|    100000|     100|0.85938955|  PASSED
          sts_serial|   3|    100000|     100|0.33266740|  PASSED
          sts_serial|   3|    100000|     100|0.21315610|  PASSED
          sts_serial|   4|    100000|     100|0.59048028|  PASSED
          sts_serial|   4|    100000|     100|0.90816957|  PASSED
          sts_serial|   5|    100000|     100|0.34098377|  PASSED
```

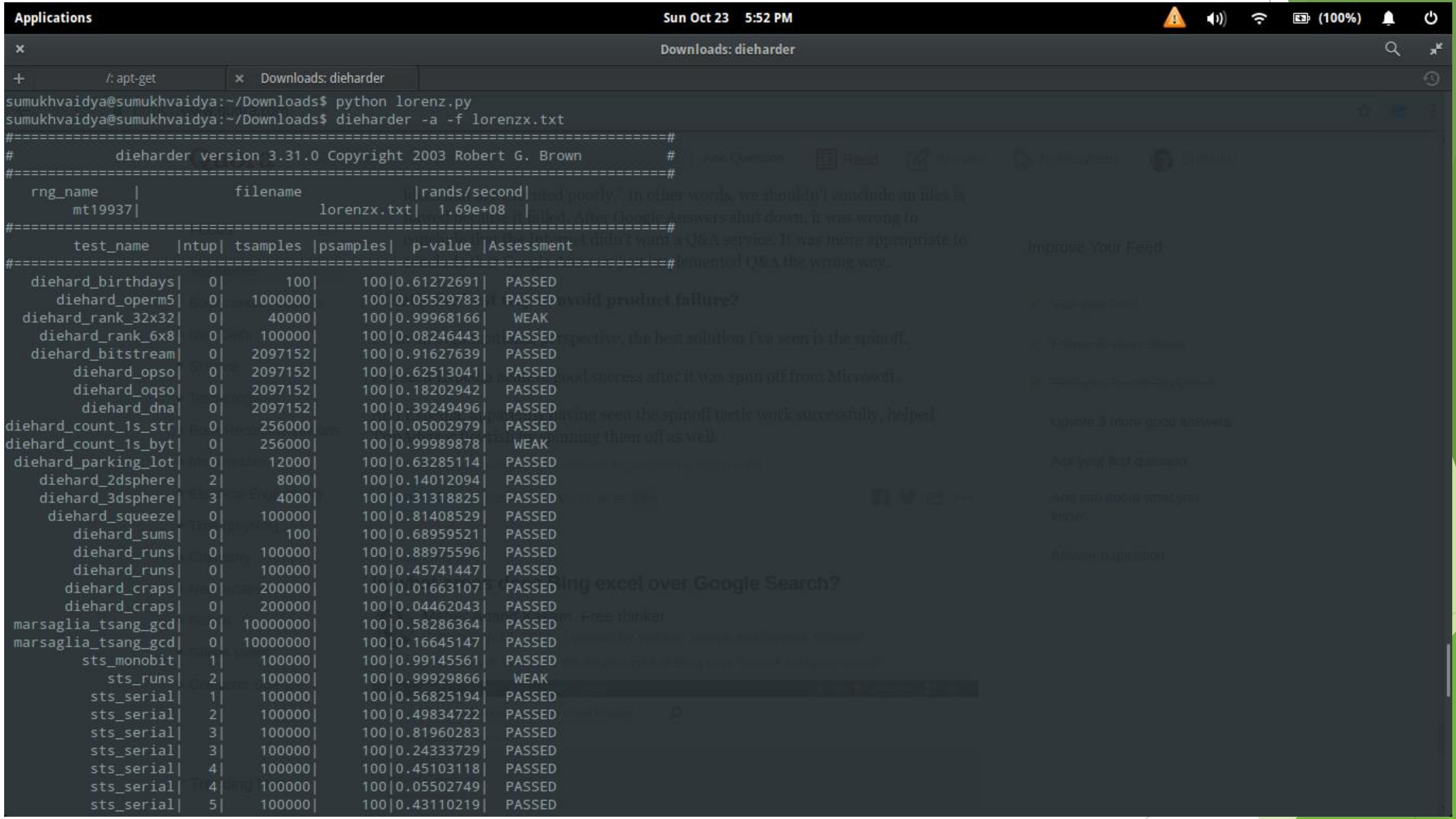


```
           sts_serial|   5|    100000|     100|0.82728997|  PASSED
           sts_serial|   6|    100000|     100|0.82911383|  PASSED
           sts_serial|   6|    100000|     100|0.56967810|  PASSED
           sts_serial|   7|    100000|     100|0.60889433|  PASSED
           sts_serial|   7|    100000|     100|0.81407600|  PASSED
           sts_serial|   8|    100000|     100|0.31184296|  PASSED
           sts_serial|   8|    100000|     100|0.99843546|   WEAK
           sts_serial|   9|    100000|     100|0.46645155|  PASSED
           sts_serial|   9|    100000|     100|0.02931867|  PASSED
           sts_serial|  10|    100000|     100|0.80945147|  PASSED
           sts_serial|  10|    100000|     100|0.83545328|  PASSED
           sts_serial|  11|    100000|     100|0.50491188|  PASSED
           sts_serial|  11|    100000|     100|0.45548374|  PASSED
           sts_serial|  12|    100000|     100|0.81931992|  PASSED
           sts_serial|  12|    100000|     100|0.87411097|  PASSED
           sts_serial|  13|    100000|     100|0.52285048|  PASSED
           sts_serial|  13|    100000|     100|0.92375324|  PASSED
           sts_serial|  14|    100000|     100|0.78060314|  PASSED
           sts_serial|  14|    100000|     100|0.98061154|  PASSED
           sts_serial|  15|    100000|     100|0.96488043|  PASSED
           sts_serial|  15|    100000|     100|0.65170909|  PASSED
           sts_serial|  16|    100000|     100|0.61512873|  PASSED
           sts_serial|  16|    100000|     100|0.23950135|  PASSED
          rgb_bitdist|   1|    100000|     100|0.80197860|  PASSED
          rgb_bitdist|   2|    100000|     100|0.96807445|  PASSED
          rgb_bitdist|   3|    100000|     100|0.77039678|  PASSED
          rgb_bitdist|   4|    100000|     100|0.77033890|  PASSED
          rgb_bitdist|   5|    100000|     100|0.33439519|  PASSED
          rgb_bitdist|   6|    100000|     100|0.44010926|  PASSED
          rgb_bitdist|   7|    100000|     100|0.27018817|  PASSED
          rgb_bitdist|   8|    100000|     100|0.10182237|  PASSED
          rgb_bitdist|   9|    100000|     100|0.51095835|  PASSED
          rgb_bitdist|  10|    100000|     100|0.18759793|  PASSED
          rgb_bitdist|  11|    100000|     100|0.18249321|  PASSED
          rgb_bitdist|  12|    100000|     100|0.91505496|  PASSED
rgb_minimum_distance|   2|     10000|    1000|0.57844191|  PASSED
rgb_minimum_distance|   3|     10000|    1000|0.86502037|  PASSED
rgb_minimum_distance|   4|     10000|    1000|0.63070560|  PASSED
rgb_minimum_distance|   5|     10000|    1000|0.86282200|  PASSED
    rgb_permutations|   2|    100000|     100|0.62102183|  PASSED
```

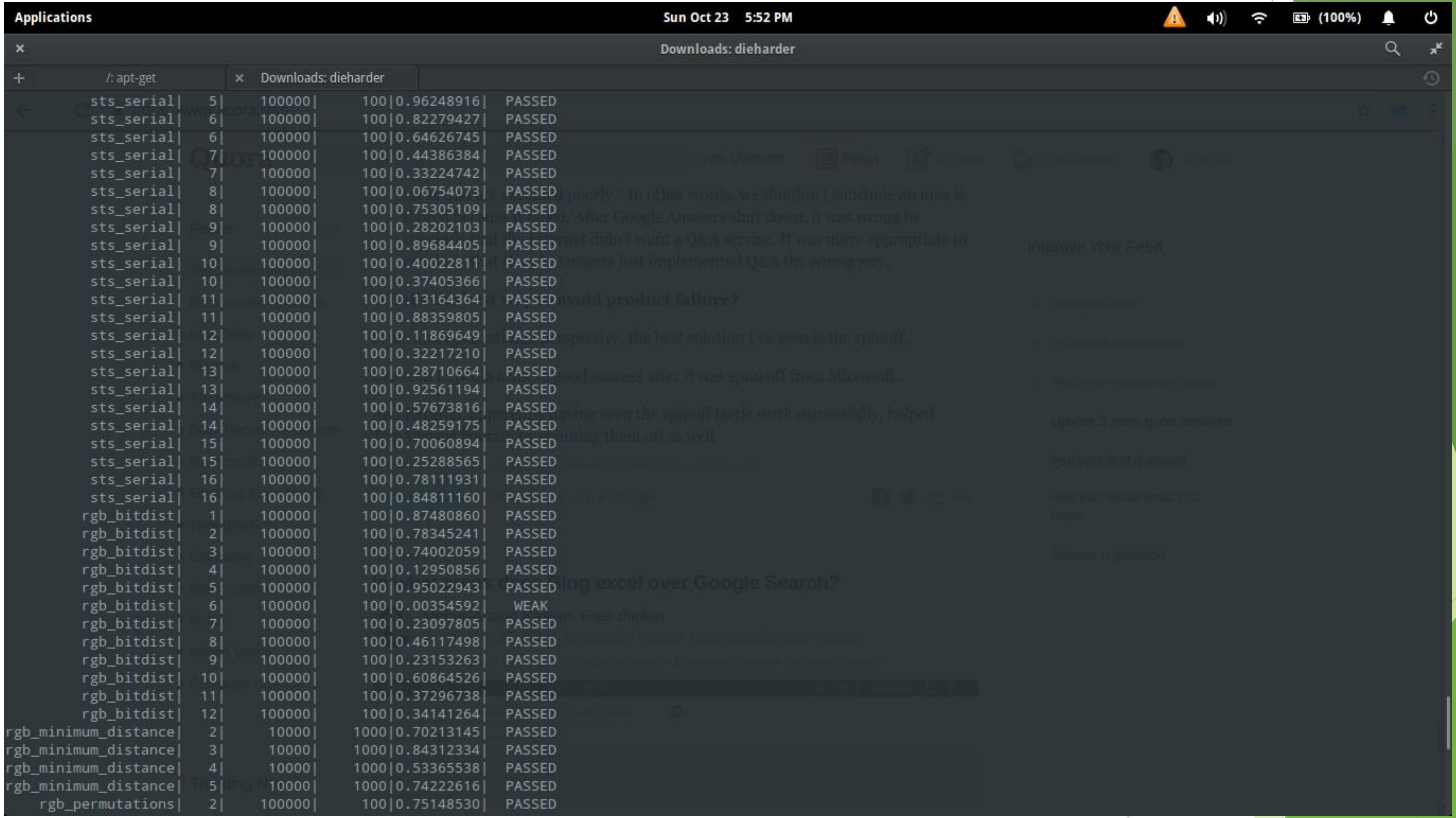


```
   rgb_permutations|   3|    100000|     100|0.36547008|  PASSED
   rgb_permutations|   4|    100000|     100|0.68097404|  PASSED
   rgb_permutations|   5|    100000|     100|0.95081949|  PASSED
     rgb_lagged_sum|   0|   1000000|     100|0.94104638|  PASSED
     rgb_lagged_sum|   1|   1000000|     100|0.64081754|  PASSED
     rgb_lagged_sum|   2|   1000000|     100|0.97265792|  PASSED
     rgb_lagged_sum|   3|   1000000|     100|0.45521773|  PASSED
     rgb_lagged_sum|   4|   1000000|     100|0.26352220|  PASSED
     rgb_lagged_sum|   5|   1000000|     100|0.12913947|  PASSED
     rgb_lagged_sum|   6|   1000000|     100|0.60681017|  PASSED
     rgb_lagged_sum|   7|   1000000|     100|0.77460412|  PASSED
     rgb_lagged_sum|   8|   1000000|     100|0.76285066|  PASSED
     rgb_lagged_sum|   9|   1000000|     100|0.86790613|  PASSED
     rgb_lagged_sum|  10|   1000000|     100|0.39853351|  PASSED
     rgb_lagged_sum|  11|   1000000|     100|0.74681952|  PASSED
     rgb_lagged_sum|  12|   1000000|     100|0.80728999|  PASSED
     rgb_lagged_sum|  13|   1000000|     100|0.30710347|  PASSED
     rgb_lagged_sum|  14|   1000000|     100|0.95507870|  PASSED
     rgb_lagged_sum|  15|   1000000|     100|0.01989391|  PASSED
     rgb_lagged_sum|  16|   1000000|     100|0.47859195|  PASSED
     rgb_lagged_sum|  17|   1000000|     100|0.19696157|  PASSED
     rgb_lagged_sum|  18|   1000000|     100|0.93003082|  PASSED
     rgb_lagged_sum|  19|   1000000|     100|0.80033529|  PASSED
     rgb_lagged_sum|  20|   1000000|     100|0.18421376|  PASSED
     rgb_lagged_sum|  21|   1000000|     100|0.61937336|  PASSED
     rgb_lagged_sum|  22|   1000000|     100|0.42586385|  PASSED
     rgb_lagged_sum|  23|   1000000|     100|0.80266953|  PASSED
     rgb_lagged_sum|  24|   1000000|     100|0.46501454|  PASSED
     rgb_lagged_sum|  25|   1000000|     100|0.85802887|  PASSED
     rgb_lagged_sum|  26|   1000000|     100|0.03269628|  PASSED
     rgb_lagged_sum|  27|   1000000|     100|0.09438973|  PASSED
     rgb_lagged_sum|  28|   1000000|     100|0.00972809|  PASSED
     rgb_lagged_sum|  29|   1000000|     100|0.46737811|  PASSED
     rgb_lagged_sum|  30|   1000000|     100|0.61454382|  PASSED
     rgb_lagged_sum|  31|   1000000|     100|0.86000314|  PASSED
     rgb_lagged_sum|  32|   1000000|     100|0.00132700|   WEAK
    rgb_kstest_test|   0|     10000|    1000|0.13488329|  PASSED
    dab_bytedistrib|   0|  51200000|       1|0.45230409|  PASSED
            dab_dct| 256|     50000|       1|0.57223651|  PASSED
Preparing to run test 207.  ntuple = 0
```

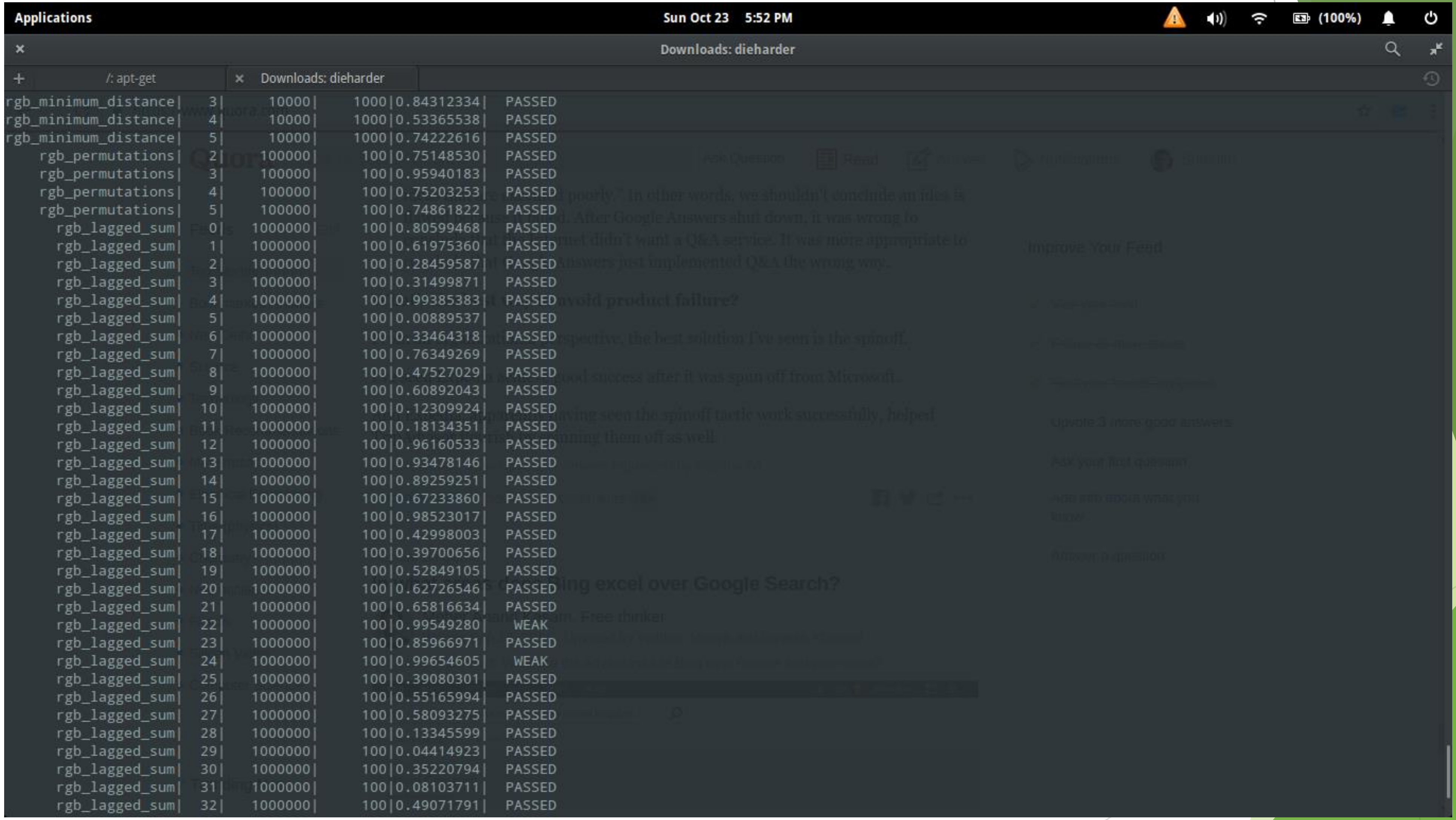
- Results for lorenz map



```
sumukhvaidya@sumukhvaidya:~/Downloads$ python lorenz.py
sumukhvaidya@sumukhvaidya:~/Downloads$ dieharder -a -f lorenzx.txt
#=============================================================================#
#            dieharder version 3.31.0 Copyright 2003 Robert G. Brown          #
#=============================================================================#
   rng_name    |           filename             |rands/second|
        mt19937|                      lorenzx.txt|  1.69e+08  |
#=============================================================================#
        test_name   |ntup| tsamples |psamples|  p-value |Assessment
#=============================================================================#
   diehard_birthdays|   0|       100|     100|0.61272691|  PASSED
      diehard_operm5|   0|   1000000|     100|0.05529783|  PASSED
  diehard_rank_32x32|   0|     40000|     100|0.99968166|   WEAK
    diehard_rank_6x8|   0|    100000|     100|0.08246443|  PASSED
   diehard_bitstream|   0|   2097152|     100|0.91627639|  PASSED
        diehard_opso|   0|   2097152|     100|0.62513041|  PASSED
        diehard_oqso|   0|   2097152|     100|0.18202942|  PASSED
         diehard_dna|   0|   2097152|     100|0.39249496|  PASSED
diehard_count_1s_str|   0|    256000|     100|0.05002979|  PASSED
diehard_count_1s_byt|   0|    256000|     100|0.99989878|   WEAK
 diehard_parking_lot|   0|     12000|     100|0.63285114|  PASSED
    diehard_2dsphere|   2|      8000|     100|0.14012094|  PASSED
    diehard_3dsphere|   3|      4000|     100|0.31318825|  PASSED
     diehard_squeeze|   0|    100000|     100|0.81408529|  PASSED
        diehard_sums|   0|       100|     100|0.68959521|  PASSED
        diehard_runs|   0|    100000|     100|0.88975596|  PASSED
        diehard_runs|   0|    100000|     100|0.45741447|  PASSED
       diehard_craps|   0|    200000|     100|0.01663107|  PASSED
       diehard_craps|   0|    200000|     100|0.04462043|  PASSED
 marsaglia_tsang_gcd|   0|  10000000|     100|0.58286364|  PASSED
 marsaglia_tsang_gcd|   0|  10000000|     100|0.16645147|  PASSED
         sts_monobit|   1|    100000|     100|0.99145561|  PASSED
            sts_runs|   2|    100000|     100|0.99929866|   WEAK
          sts_serial|   1|    100000|     100|0.56825194|  PASSED
          sts_serial|   2|    100000|     100|0.49834722|  PASSED
          sts_serial|   3|    100000|     100|0.81960283|  PASSED
          sts_serial|   3|    100000|     100|0.24333729|  PASSED
          sts_serial|   4|    100000|     100|0.45103118|  PASSED
          sts_serial|   4|    100000|     100|0.05502749|  PASSED
          sts_serial|   5|    100000|     100|0.43110219|  PASSED
```

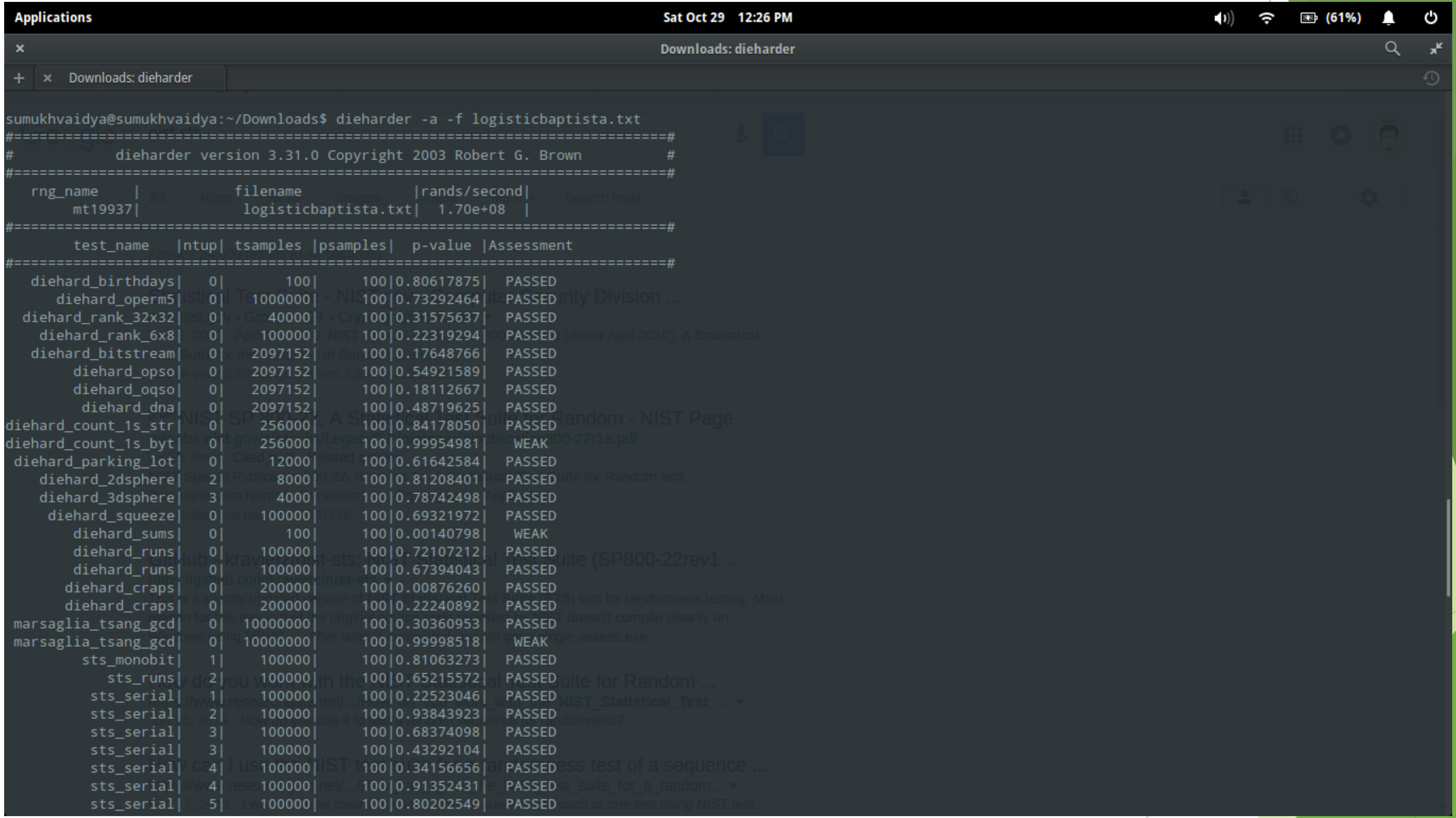


```
        sts_serial|   5|    100000|     100|0.96248916|  PASSED
        sts_serial|   6|    100000|     100|0.82279427|  PASSED
        sts_serial|   6|    100000|     100|0.64626745|  PASSED
        sts_serial|   7|    100000|     100|0.44386384|  PASSED
        sts_serial|   7|    100000|     100|0.33224742|  PASSED
        sts_serial|   8|    100000|     100|0.06754073|  PASSED
        sts_serial|   8|    100000|     100|0.75305109|  PASSED
        sts_serial|   9|    100000|     100|0.28202103|  PASSED
        sts_serial|   9|    100000|     100|0.89684405|  PASSED
        sts_serial|  10|    100000|     100|0.40022811|  PASSED
        sts_serial|  10|    100000|     100|0.37405366|  PASSED
        sts_serial|  11|    100000|     100|0.13164364|  PASSED
        sts_serial|  11|    100000|     100|0.88359805|  PASSED
        sts_serial|  12|    100000|     100|0.11869649|  PASSED
        sts_serial|  12|    100000|     100|0.32217210|  PASSED
        sts_serial|  13|    100000|     100|0.28710664|  PASSED
        sts_serial|  13|    100000|     100|0.92561194|  PASSED
        sts_serial|  14|    100000|     100|0.57673816|  PASSED
        sts_serial|  14|    100000|     100|0.48259175|  PASSED
        sts_serial|  15|    100000|     100|0.70060894|  PASSED
        sts_serial|  15|    100000|     100|0.25288565|  PASSED
        sts_serial|  16|    100000|     100|0.78111931|  PASSED
        sts_serial|  16|    100000|     100|0.84811160|  PASSED
       rgb_bitdist|   1|    100000|     100|0.87480860|  PASSED
       rgb_bitdist|   2|    100000|     100|0.78345241|  PASSED
       rgb_bitdist|   3|    100000|     100|0.74002059|  PASSED
       rgb_bitdist|   4|    100000|     100|0.12950856|  PASSED
       rgb_bitdist|   5|    100000|     100|0.95022943|  PASSED
       rgb_bitdist|   6|    100000|     100|0.00354592|   WEAK
       rgb_bitdist|   7|    100000|     100|0.23097805|  PASSED
       rgb_bitdist|   8|    100000|     100|0.46117498|  PASSED
       rgb_bitdist|   9|    100000|     100|0.23153263|  PASSED
       rgb_bitdist|  10|    100000|     100|0.60864526|  PASSED
       rgb_bitdist|  11|    100000|     100|0.37296738|  PASSED
       rgb_bitdist|  12|    100000|     100|0.34141264|  PASSED
rgb_minimum_distance|   2|     10000|    1000|0.70213145|  PASSED
rgb_minimum_distance|   3|     10000|    1000|0.84312334|  PASSED
rgb_minimum_distance|   4|     10000|    1000|0.53365538|  PASSED
rgb_minimum_distance|   5|     10000|    1000|0.74222616|  PASSED
    rgb_permutations|   2|    100000|     100|0.75148530|  PASSED
```

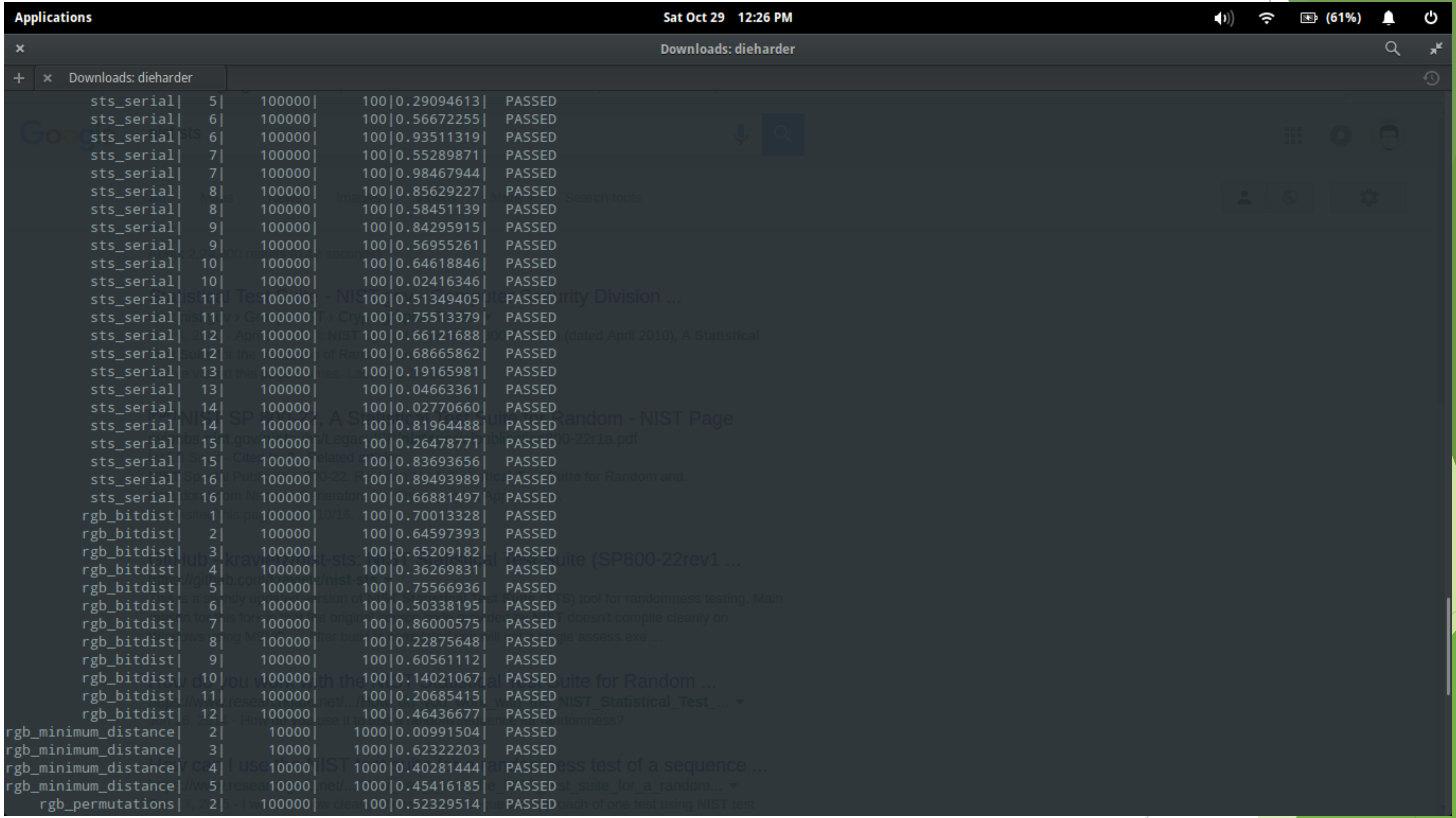


```
rgb_minimum_distance|   3|     10000|      1000|0.84312334|  PASSED
rgb_minimum_distance|   4|     10000|      1000|0.53365538|  PASSED
rgb_minimum_distance|   5|     10000|      1000|0.74222616|  PASSED
    rgb_permutations|   2|    100000|       100|0.75148530|  PASSED
    rgb_permutations|   3|    100000|       100|0.95940183|  PASSED
    rgb_permutations|   4|    100000|       100|0.75203253|  PASSED
    rgb_permutations|   5|    100000|       100|0.74861822|  PASSED
      rgb_lagged_sum|   0|   1000000|       100|0.80599468|  PASSED
      rgb_lagged_sum|   1|   1000000|       100|0.61975360|  PASSED
      rgb_lagged_sum|   2|   1000000|       100|0.28459587|  PASSED
      rgb_lagged_sum|   3|   1000000|       100|0.31499871|  PASSED
      rgb_lagged_sum|   4|   1000000|       100|0.99385383|  PASSED
      rgb_lagged_sum|   5|   1000000|       100|0.00889537|  PASSED
      rgb_lagged_sum|   6|   1000000|       100|0.33464318|  PASSED
      rgb_lagged_sum|   7|   1000000|       100|0.76349269|  PASSED
      rgb_lagged_sum|   8|   1000000|       100|0.47527029|  PASSED
      rgb_lagged_sum|   9|   1000000|       100|0.60892043|  PASSED
      rgb_lagged_sum|  10|   1000000|       100|0.12309924|  PASSED
      rgb_lagged_sum|  11|   1000000|       100|0.18134351|  PASSED
      rgb_lagged_sum|  12|   1000000|       100|0.96160533|  PASSED
      rgb_lagged_sum|  13|   1000000|       100|0.93478146|  PASSED
      rgb_lagged_sum|  14|   1000000|       100|0.89259251|  PASSED
      rgb_lagged_sum|  15|   1000000|       100|0.67233860|  PASSED
      rgb_lagged_sum|  16|   1000000|       100|0.98523017|  PASSED
      rgb_lagged_sum|  17|   1000000|       100|0.42998003|  PASSED
      rgb_lagged_sum|  18|   1000000|       100|0.39700656|  PASSED
      rgb_lagged_sum|  19|   1000000|       100|0.52849105|  PASSED
      rgb_lagged_sum|  20|   1000000|       100|0.62726546|  PASSED
      rgb_lagged_sum|  21|   1000000|       100|0.65816634|  PASSED
      rgb_lagged_sum|  22|   1000000|       100|0.99549280|   WEAK
      rgb_lagged_sum|  23|   1000000|       100|0.85966971|  PASSED
      rgb_lagged_sum|  24|   1000000|       100|0.99654605|   WEAK
      rgb_lagged_sum|  25|   1000000|       100|0.39080301|  PASSED
      rgb_lagged_sum|  26|   1000000|       100|0.55165994|  PASSED
      rgb_lagged_sum|  27|   1000000|       100|0.58093275|  PASSED
      rgb_lagged_sum|  28|   1000000|       100|0.13345599|  PASSED
      rgb_lagged_sum|  29|   1000000|       100|0.04414923|  PASSED
      rgb_lagged_sum|  30|   1000000|       100|0.35220794|  PASSED
      rgb_lagged_sum|  31|   1000000|       100|0.08103711|  PASSED
      rgb_lagged_sum|  32|   1000000|       100|0.49071791|  PASSED
```

Results for Logistic map: Baptista method



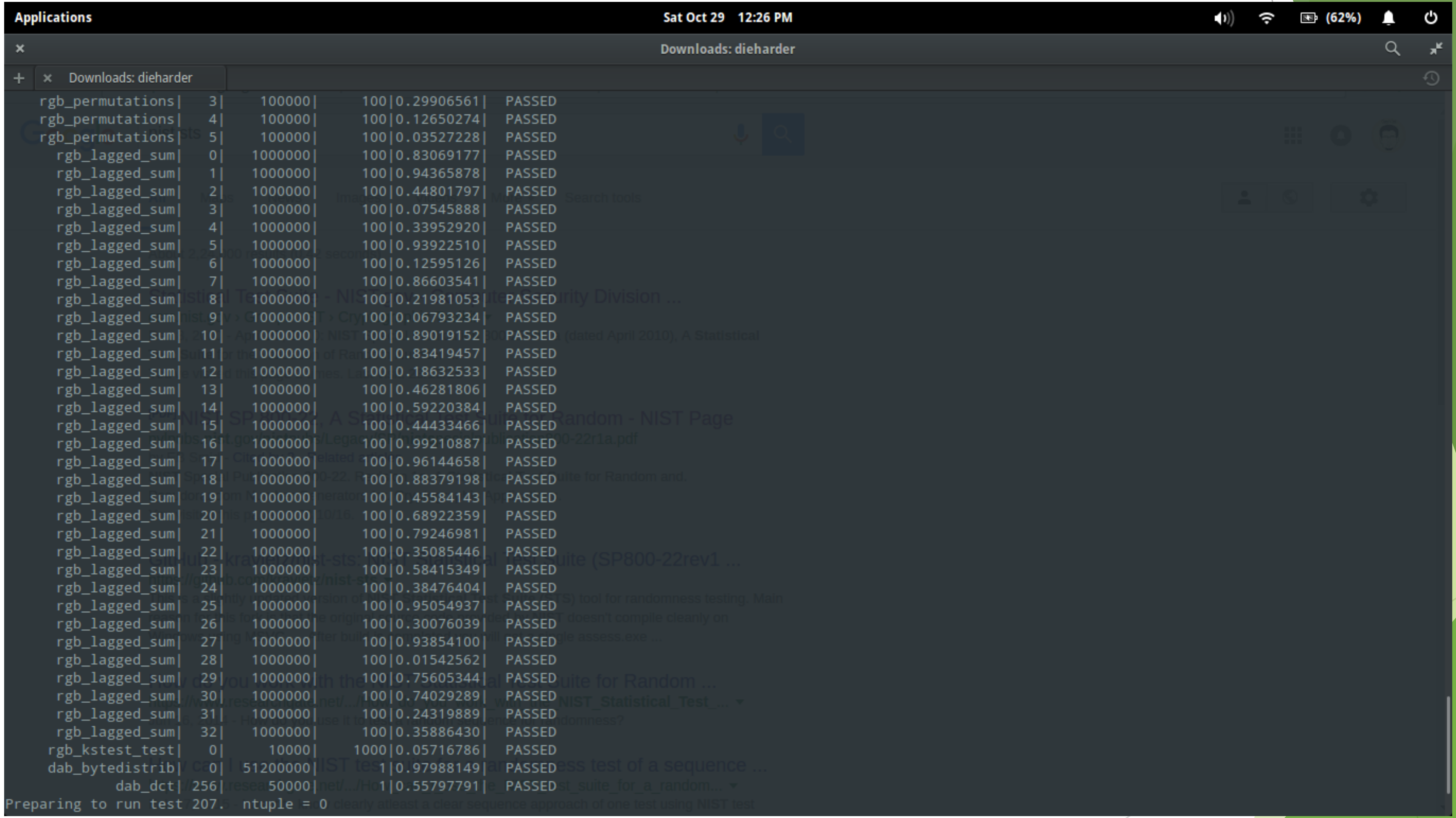
```
sumukhvaidya@sumukhvaidya:~/Downloads$ dieharder -a -f logisticbaptista.txt
#=============================================================================#
#            dieharder version 3.31.0 Copyright 2003 Robert G. Brown          #
#=============================================================================#
   rng_name    |           filename             |rands/second|
        mt19937|            logisticbaptista.txt|  1.70e+08  |
#=============================================================================#
        test_name   |ntup| tsamples |psamples|  p-value |Assessment
#=============================================================================#
   diehard_birthdays|   0|       100|     100|0.80617875|  PASSED
      diehard_operm5|   0|   1000000|     100|0.73292464|  PASSED
  diehard_rank_32x32|   0|     40000|     100|0.31575637|  PASSED
    diehard_rank_6x8|   0|    100000|     100|0.22319294|  PASSED
   diehard_bitstream|   0|   2097152|     100|0.17648766|  PASSED
        diehard_opso|   0|   2097152|     100|0.54921589|  PASSED
        diehard_oqso|   0|   2097152|     100|0.18112667|  PASSED
         diehard_dna|   0|   2097152|     100|0.48719625|  PASSED
diehard_count_1s_str|   0|    256000|     100|0.84178050|  PASSED
diehard_count_1s_byt|   0|    256000|     100|0.99954981|   WEAK
 diehard_parking_lot|   0|     12000|     100|0.61642584|  PASSED
    diehard_2dsphere|   2|      8000|     100|0.81208401|  PASSED
    diehard_3dsphere|   3|      4000|     100|0.78742498|  PASSED
     diehard_squeeze|   0|    100000|     100|0.69321972|  PASSED
        diehard_sums|   0|       100|     100|0.00140798|   WEAK
        diehard_runs|   0|    100000|     100|0.72107212|  PASSED
        diehard_runs|   0|    100000|     100|0.67394043|  PASSED
       diehard_craps|   0|    200000|     100|0.00876260|  PASSED
       diehard_craps|   0|    200000|     100|0.22240892|  PASSED
 marsaglia_tsang_gcd|   0|  10000000|     100|0.30360953|  PASSED
 marsaglia_tsang_gcd|   0|  10000000|     100|0.99998518|   WEAK
         sts_monobit|   1|    100000|     100|0.81063273|  PASSED
            sts_runs|   2|    100000|     100|0.65215572|  PASSED
          sts_serial|   1|    100000|     100|0.22523046|  PASSED
          sts_serial|   2|    100000|     100|0.93843923|  PASSED
          sts_serial|   3|    100000|     100|0.68374098|  PASSED
          sts_serial|   3|    100000|     100|0.43292104|  PASSED
          sts_serial|   4|    100000|     100|0.34156656|  PASSED
          sts_serial|   4|    100000|     100|0.91352431|  PASSED
          sts_serial|   5|    100000|     100|0.80202549|  PASSED
```



Applications   Sat Oct 29 12:26 PM   (61%)

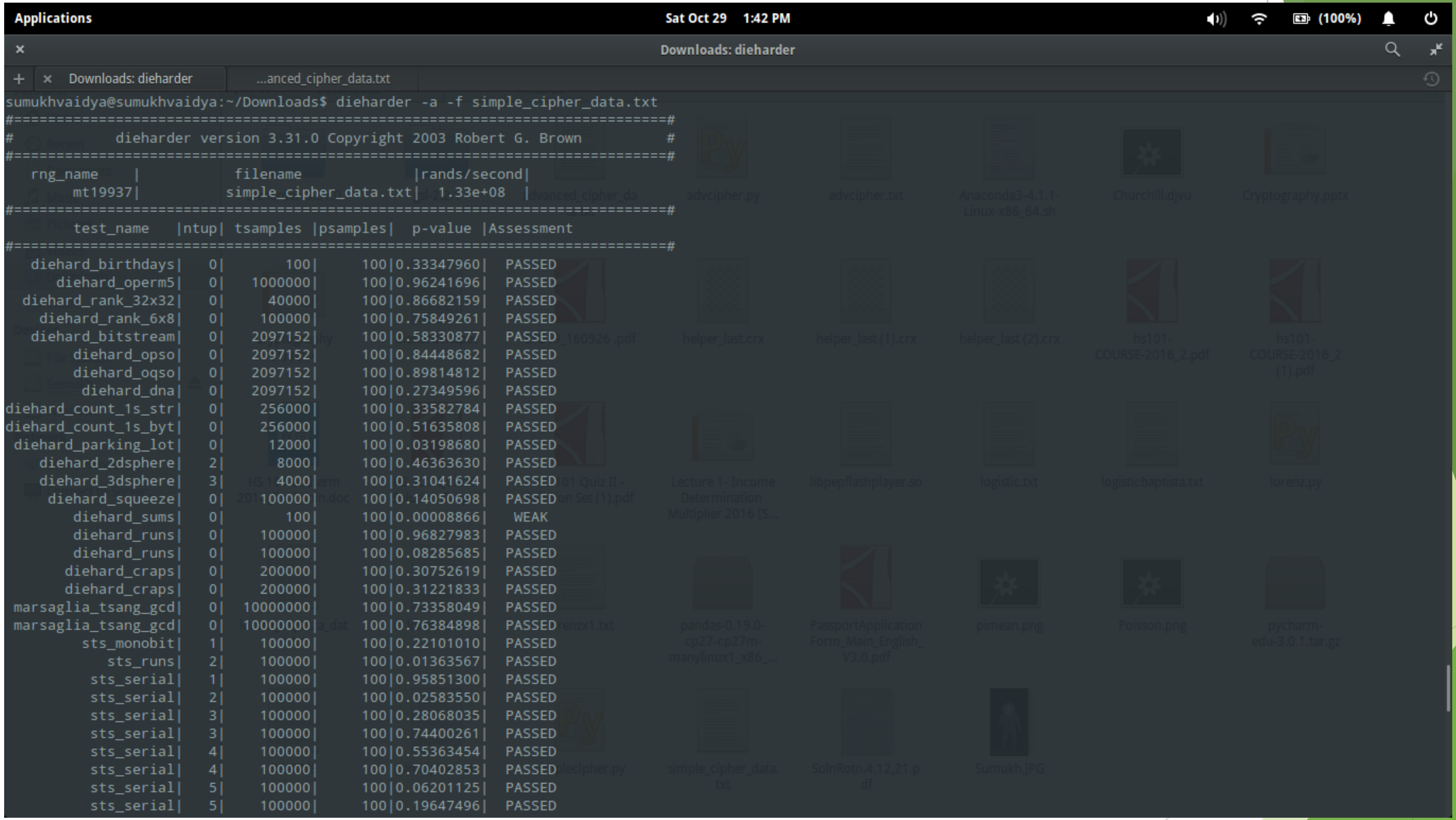
Downloads: dieharder

```
        sts_serial|   5|    100000|     100|0.29094613|  PASSED
        sts_serial|   6|    100000|     100|0.56672255|  PASSED
        sts_serial|   6|    100000|     100|0.93511319|  PASSED
        sts_serial|   7|    100000|     100|0.55289871|  PASSED
        sts_serial|   7|    100000|     100|0.98467944|  PASSED
        sts_serial|   8|    100000|     100|0.85629227|  PASSED
        sts_serial|   8|    100000|     100|0.58451139|  PASSED
        sts_serial|   9|    100000|     100|0.84295915|  PASSED
        sts_serial|   9|    100000|     100|0.56955261|  PASSED
        sts_serial|  10|    100000|     100|0.64618846|  PASSED
        sts_serial|  10|    100000|     100|0.02416346|  PASSED
        sts_serial|  11|    100000|     100|0.51349405|  PASSED
        sts_serial|  11|    100000|     100|0.75513379|  PASSED
        sts_serial|  12|    100000|     100|0.66121688|  PASSED
        sts_serial|  12|    100000|     100|0.68665862|  PASSED
        sts_serial|  13|    100000|     100|0.19165981|  PASSED
        sts_serial|  13|    100000|     100|0.04663361|  PASSED
        sts_serial|  14|    100000|     100|0.02770660|  PASSED
        sts_serial|  14|    100000|     100|0.81964488|  PASSED
        sts_serial|  15|    100000|     100|0.26478771|  PASSED
        sts_serial|  15|    100000|     100|0.83693656|  PASSED
        sts_serial|  16|    100000|     100|0.89493989|  PASSED
        sts_serial|  16|    100000|     100|0.66881497|  PASSED
       rgb_bitdist|   1|    100000|     100|0.70013328|  PASSED
       rgb_bitdist|   2|    100000|     100|0.64597393|  PASSED
       rgb_bitdist|   3|    100000|     100|0.65209182|  PASSED
       rgb_bitdist|   4|    100000|     100|0.36269831|  PASSED
       rgb_bitdist|   5|    100000|     100|0.75566936|  PASSED
       rgb_bitdist|   6|    100000|     100|0.50338195|  PASSED
       rgb_bitdist|   7|    100000|     100|0.86000575|  PASSED
       rgb_bitdist|   8|    100000|     100|0.22875648|  PASSED
       rgb_bitdist|   9|    100000|     100|0.60561112|  PASSED
       rgb_bitdist|  10|    100000|     100|0.14021067|  PASSED
       rgb_bitdist|  11|    100000|     100|0.20685415|  PASSED
       rgb_bitdist|  12|    100000|     100|0.46436677|  PASSED
rgb_minimum_distance|   2|     10000|    1000|0.00991504|  PASSED
rgb_minimum_distance|   3|     10000|    1000|0.62322203|  PASSED
rgb_minimum_distance|   4|     10000|    1000|0.40281444|  PASSED
rgb_minimum_distance|   5|     10000|    1000|0.45416185|  PASSED
    rgb_permutations|   2|    100000|     100|0.52329514|  PASSED
```

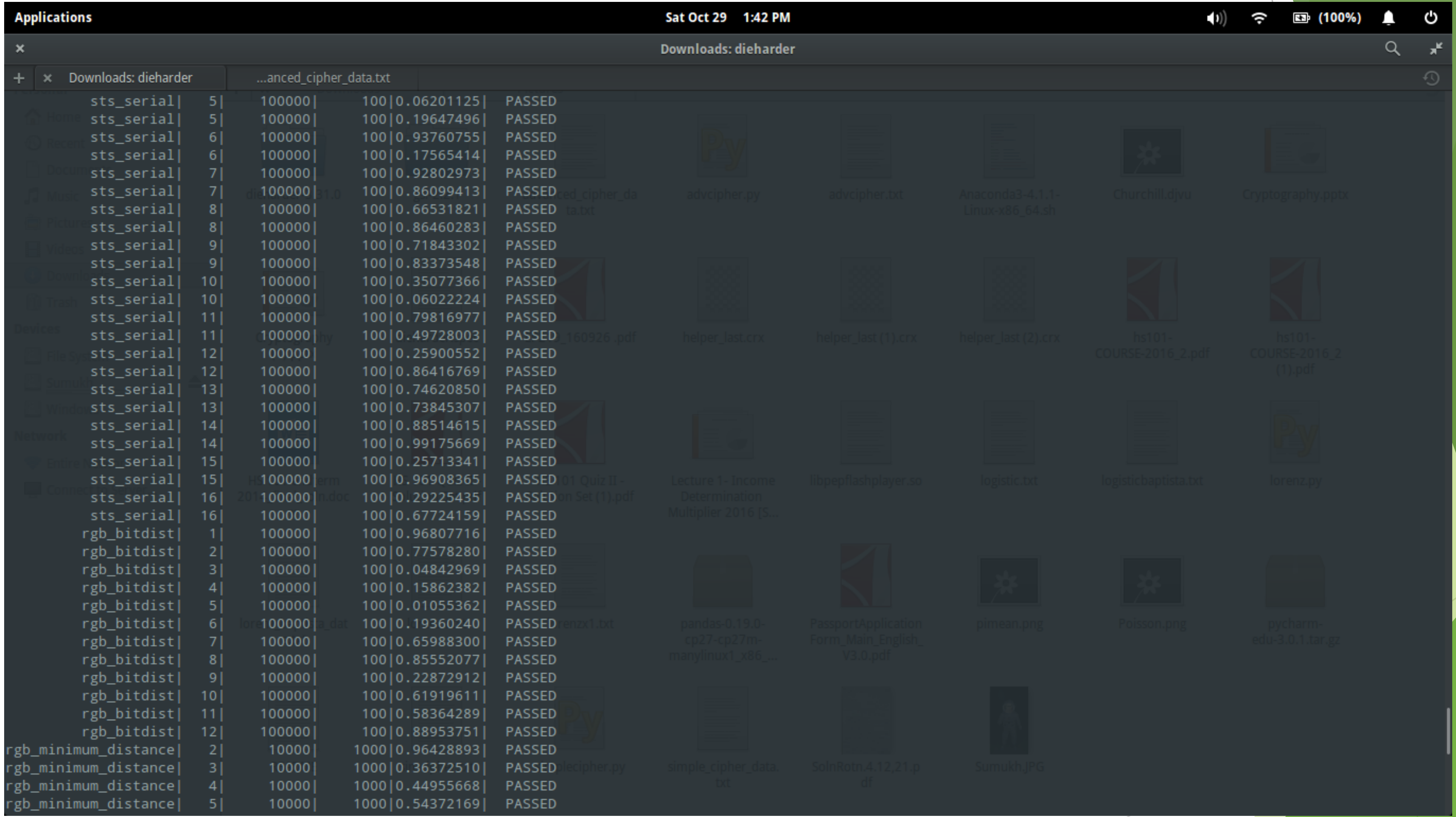


```
   rgb_permutations|   3|    100000|     100|0.29906561|  PASSED
   rgb_permutations|   4|    100000|     100|0.12650274|  PASSED
   rgb_permutations|   5|    100000|     100|0.03527228|  PASSED
     rgb_lagged_sum|   0|   1000000|     100|0.83069177|  PASSED
     rgb_lagged_sum|   1|   1000000|     100|0.94365878|  PASSED
     rgb_lagged_sum|   2|   1000000|     100|0.44801797|  PASSED
     rgb_lagged_sum|   3|   1000000|     100|0.07545888|  PASSED
     rgb_lagged_sum|   4|   1000000|     100|0.33952920|  PASSED
     rgb_lagged_sum|   5|   1000000|     100|0.93922510|  PASSED
     rgb_lagged_sum|   6|   1000000|     100|0.12595126|  PASSED
     rgb_lagged_sum|   7|   1000000|     100|0.86603541|  PASSED
     rgb_lagged_sum|   8|   1000000|     100|0.21981053|  PASSED
     rgb_lagged_sum|   9|   1000000|     100|0.06793234|  PASSED
     rgb_lagged_sum|  10|   1000000|     100|0.89019152|  PASSED
     rgb_lagged_sum|  11|   1000000|     100|0.83419457|  PASSED
     rgb_lagged_sum|  12|   1000000|     100|0.18632533|  PASSED
     rgb_lagged_sum|  13|   1000000|     100|0.46281806|  PASSED
     rgb_lagged_sum|  14|   1000000|     100|0.59220384|  PASSED
     rgb_lagged_sum|  15|   1000000|     100|0.44433466|  PASSED
     rgb_lagged_sum|  16|   1000000|     100|0.99210887|  PASSED
     rgb_lagged_sum|  17|   1000000|     100|0.96144658|  PASSED
     rgb_lagged_sum|  18|   1000000|     100|0.88379198|  PASSED
     rgb_lagged_sum|  19|   1000000|     100|0.45584143|  PASSED
     rgb_lagged_sum|  20|   1000000|     100|0.68922359|  PASSED
     rgb_lagged_sum|  21|   1000000|     100|0.79246981|  PASSED
     rgb_lagged_sum|  22|   1000000|     100|0.35085446|  PASSED
     rgb_lagged_sum|  23|   1000000|     100|0.58415349|  PASSED
     rgb_lagged_sum|  24|   1000000|     100|0.38476404|  PASSED
     rgb_lagged_sum|  25|   1000000|     100|0.95054937|  PASSED
     rgb_lagged_sum|  26|   1000000|     100|0.30076039|  PASSED
     rgb_lagged_sum|  27|   1000000|     100|0.93854100|  PASSED
     rgb_lagged_sum|  28|   1000000|     100|0.01542562|  PASSED
     rgb_lagged_sum|  29|   1000000|     100|0.75605344|  PASSED
     rgb_lagged_sum|  30|   1000000|     100|0.74029289|  PASSED
     rgb_lagged_sum|  31|   1000000|     100|0.24319889|  PASSED
     rgb_lagged_sum|  32|   1000000|     100|0.35886430|  PASSED
    rgb_kstest_test|   0|     10000|    1000|0.05716786|  PASSED
    dab_bytedistrib|   0|  51200000|       1|0.97988149|  PASSED
            dab_dct| 256|     50000|       1|0.55797791|  PASSED
Preparing to run test 207.  ntuple = 0
```

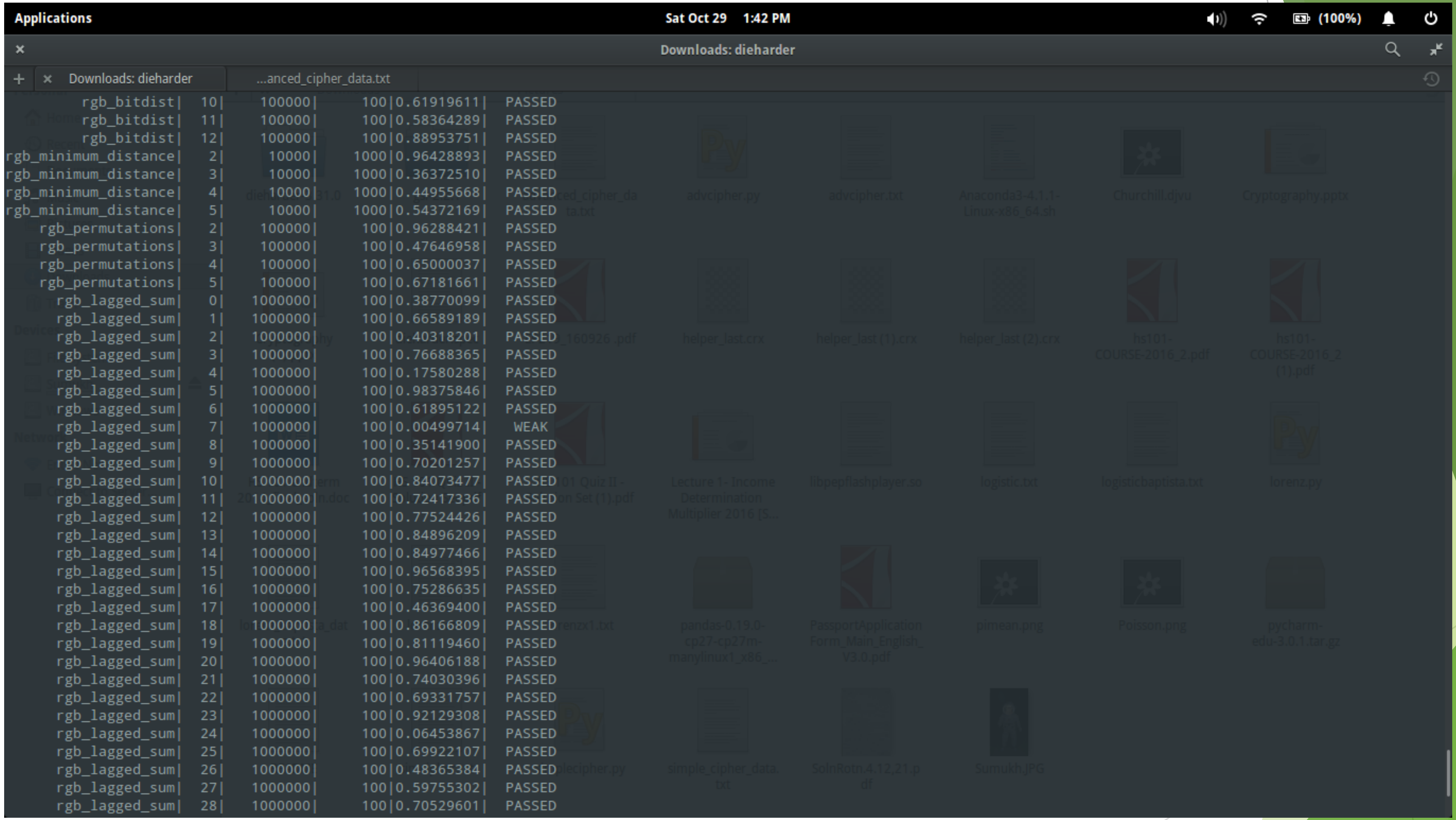
- Results for simple cipher



```
sumukhvaidya@sumukhvaidya:~/Downloads$ dieharder -a -f simple_cipher_data.txt
#=============================================================================#
#            dieharder version 3.31.0 Copyright 2003 Robert G. Brown          #
#=============================================================================#
   rng_name    |           filename             |rands/second|
        mt19937|          simple_cipher_data.txt|  1.33e+08  |
#=============================================================================#
        test_name   |ntup| tsamples |psamples|  p-value |Assessment
#=============================================================================#
   diehard_birthdays|   0|       100|     100|0.33347960|  PASSED
      diehard_operm5|   0|   1000000|     100|0.96241696|  PASSED
  diehard_rank_32x32|   0|     40000|     100|0.86682159|  PASSED
    diehard_rank_6x8|   0|    100000|     100|0.75849261|  PASSED
   diehard_bitstream|   0|   2097152|     100|0.58330877|  PASSED
        diehard_opso|   0|   2097152|     100|0.84448682|  PASSED
        diehard_oqso|   0|   2097152|     100|0.89814812|  PASSED
         diehard_dna|   0|   2097152|     100|0.27349596|  PASSED
diehard_count_1s_str|   0|    256000|     100|0.33582784|  PASSED
diehard_count_1s_byt|   0|    256000|     100|0.51635808|  PASSED
 diehard_parking_lot|   0|     12000|     100|0.03198680|  PASSED
    diehard_2dsphere|   2|      8000|     100|0.46363630|  PASSED
    diehard_3dsphere|   3|      4000|     100|0.31041624|  PASSED
     diehard_squeeze|   0|    100000|     100|0.14050698|  PASSED
        diehard_sums|   0|       100|     100|0.00008866|   WEAK
        diehard_runs|   0|    100000|     100|0.96827983|  PASSED
        diehard_runs|   0|    100000|     100|0.08285685|  PASSED
       diehard_craps|   0|    200000|     100|0.30752619|  PASSED
       diehard_craps|   0|    200000|     100|0.31221833|  PASSED
 marsaglia_tsang_gcd|   0|  10000000|     100|0.73358049|  PASSED
 marsaglia_tsang_gcd|   0|  10000000|     100|0.76384898|  PASSED
         sts_monobit|   1|    100000|     100|0.22101010|  PASSED
            sts_runs|   2|    100000|     100|0.01363567|  PASSED
          sts_serial|   1|    100000|     100|0.95851300|  PASSED
          sts_serial|   2|    100000|     100|0.02583550|  PASSED
          sts_serial|   3|    100000|     100|0.28068035|  PASSED
          sts_serial|   3|    100000|     100|0.74400261|  PASSED
          sts_serial|   4|    100000|     100|0.55363454|  PASSED
          sts_serial|   4|    100000|     100|0.70402853|  PASSED
          sts_serial|   5|    100000|     100|0.06201125|  PASSED
          sts_serial|   5|    100000|     100|0.19647496|  PASSED
```

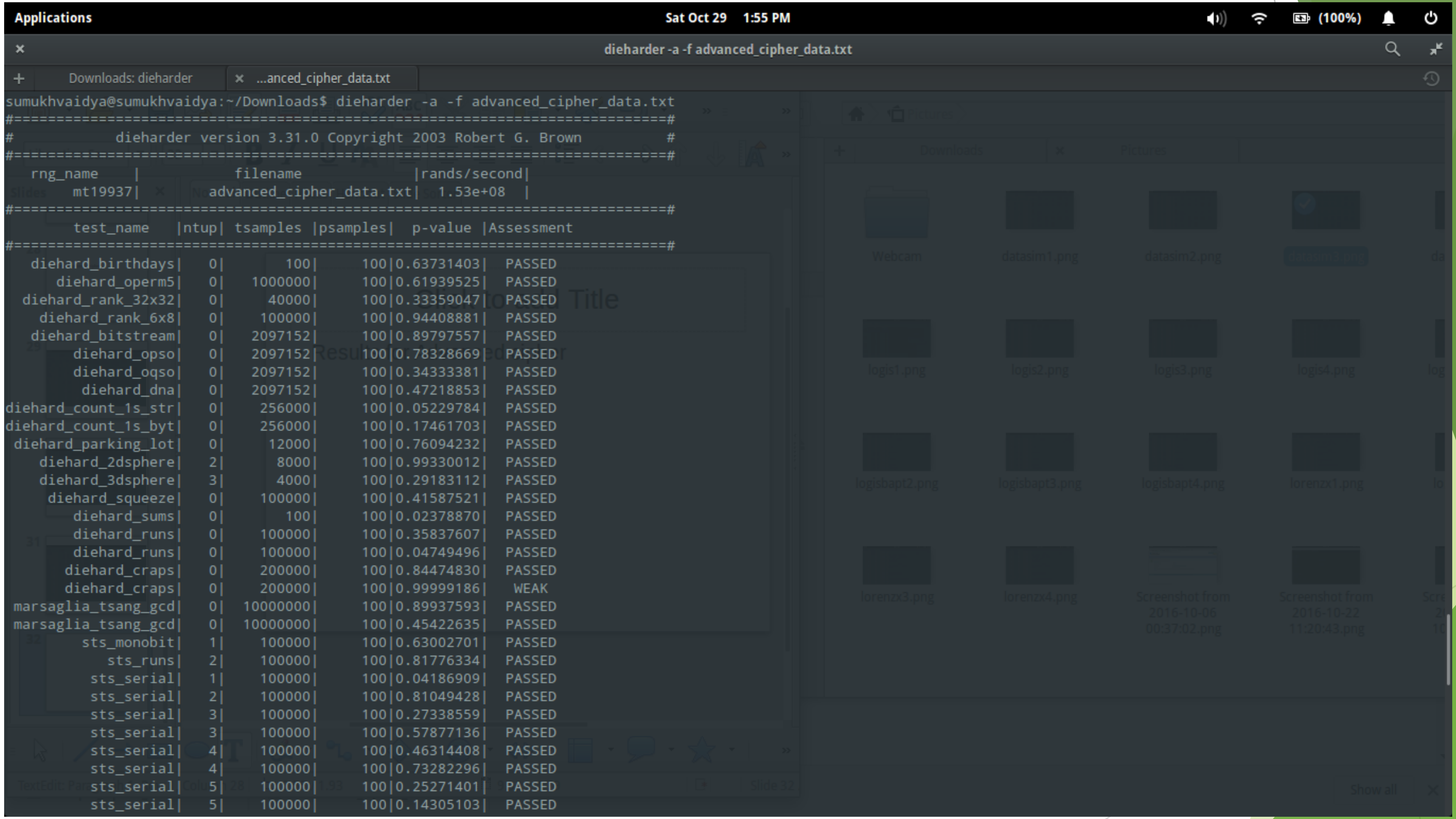


```
        sts_serial|   5|    100000|     100|0.06201125|  PASSED
        sts_serial|   5|    100000|     100|0.19647496|  PASSED
        sts_serial|   6|    100000|     100|0.93760755|  PASSED
        sts_serial|   6|    100000|     100|0.17565414|  PASSED
        sts_serial|   7|    100000|     100|0.92802973|  PASSED
        sts_serial|   7|    100000|     100|0.86099413|  PASSED
        sts_serial|   8|    100000|     100|0.66531821|  PASSED
        sts_serial|   8|    100000|     100|0.86460283|  PASSED
        sts_serial|   9|    100000|     100|0.71843302|  PASSED
        sts_serial|   9|    100000|     100|0.83373548|  PASSED
        sts_serial|  10|    100000|     100|0.35077366|  PASSED
        sts_serial|  10|    100000|     100|0.06022224|  PASSED
        sts_serial|  11|    100000|     100|0.79816977|  PASSED
        sts_serial|  11|    100000|     100|0.49728003|  PASSED
        sts_serial|  12|    100000|     100|0.25900552|  PASSED
        sts_serial|  12|    100000|     100|0.86416769|  PASSED
        sts_serial|  13|    100000|     100|0.74620850|  PASSED
        sts_serial|  13|    100000|     100|0.73845307|  PASSED
        sts_serial|  14|    100000|     100|0.88514615|  PASSED
        sts_serial|  14|    100000|     100|0.99175669|  PASSED
        sts_serial|  15|    100000|     100|0.25713341|  PASSED
        sts_serial|  15|    100000|     100|0.96908365|  PASSED
        sts_serial|  16|    100000|     100|0.29225435|  PASSED
        sts_serial|  16|    100000|     100|0.67724159|  PASSED
       rgb_bitdist|   1|    100000|     100|0.96807716|  PASSED
       rgb_bitdist|   2|    100000|     100|0.77578280|  PASSED
       rgb_bitdist|   3|    100000|     100|0.04842969|  PASSED
       rgb_bitdist|   4|    100000|     100|0.15862382|  PASSED
       rgb_bitdist|   5|    100000|     100|0.01055362|  PASSED
       rgb_bitdist|   6|    100000|     100|0.19360240|  PASSED
       rgb_bitdist|   7|    100000|     100|0.65988300|  PASSED
       rgb_bitdist|   8|    100000|     100|0.85552077|  PASSED
       rgb_bitdist|   9|    100000|     100|0.22872912|  PASSED
       rgb_bitdist|  10|    100000|     100|0.61919611|  PASSED
       rgb_bitdist|  11|    100000|     100|0.58364289|  PASSED
       rgb_bitdist|  12|    100000|     100|0.88953751|  PASSED
rgb_minimum_distance|   2|     10000|    1000|0.96428893|  PASSED
rgb_minimum_distance|   3|     10000|    1000|0.36372510|  PASSED
rgb_minimum_distance|   4|     10000|    1000|0.44955668|  PASSED
rgb_minimum_distance|   5|     10000|    1000|0.54372169|  PASSED
```

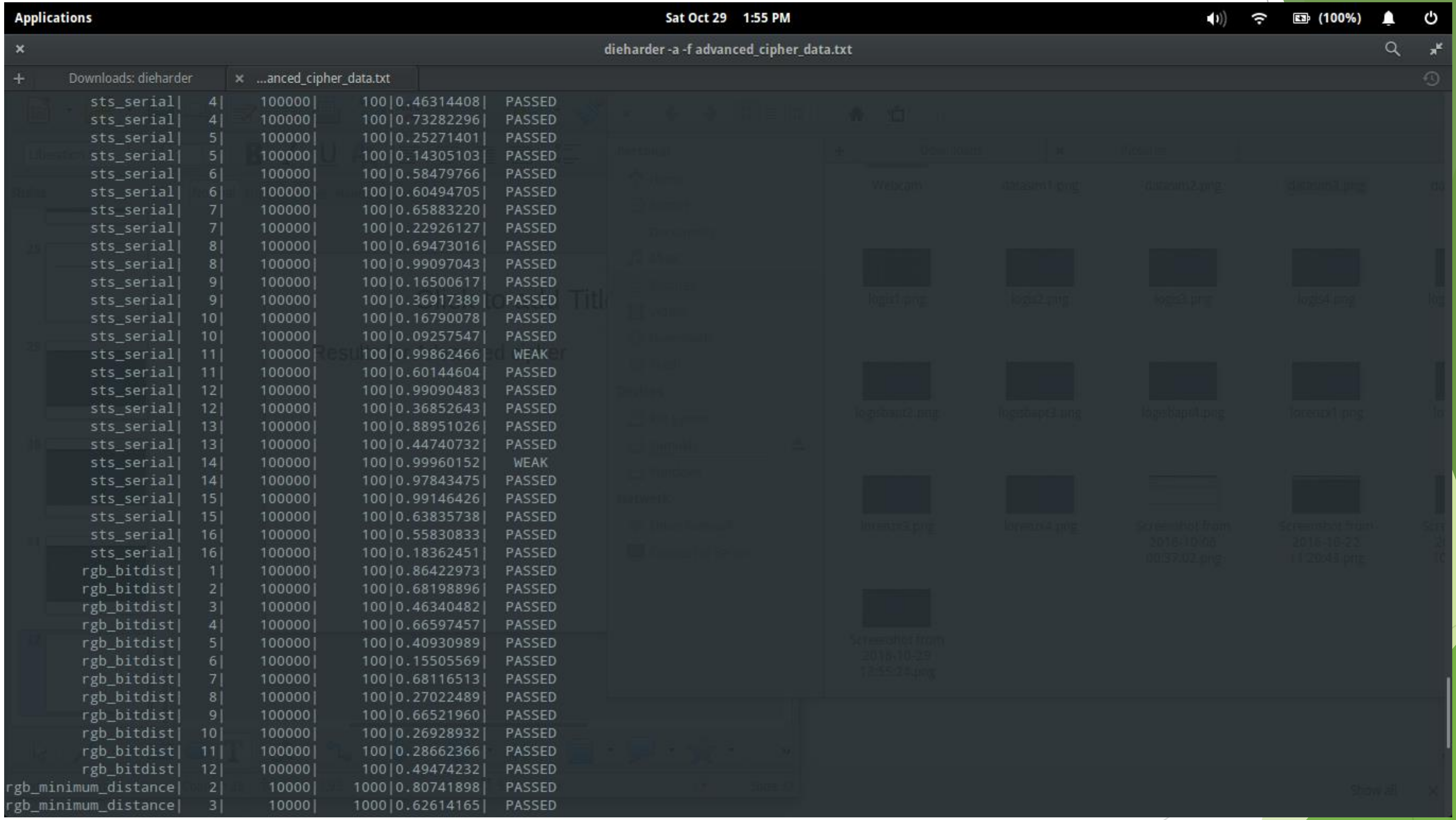


```
          rgb_bitdist|  10|    100000|     100|0.61919611|  PASSED
          rgb_bitdist|  11|    100000|     100|0.58364289|  PASSED
          rgb_bitdist|  12|    100000|     100|0.88953751|  PASSED
 rgb_minimum_distance|   2|     10000|    1000|0.96428893|  PASSED
 rgb_minimum_distance|   3|     10000|    1000|0.36372510|  PASSED
 rgb_minimum_distance|   4|     10000|    1000|0.44955668|  PASSED
 rgb_minimum_distance|   5|     10000|    1000|0.54372169|  PASSED
     rgb_permutations|   2|    100000|     100|0.96288421|  PASSED
     rgb_permutations|   3|    100000|     100|0.47646958|  PASSED
     rgb_permutations|   4|    100000|     100|0.65000037|  PASSED
     rgb_permutations|   5|    100000|     100|0.67181661|  PASSED
       rgb_lagged_sum|   0|   1000000|     100|0.38770099|  PASSED
       rgb_lagged_sum|   1|   1000000|     100|0.66589189|  PASSED
       rgb_lagged_sum|   2|   1000000|     100|0.40318201|  PASSED
       rgb_lagged_sum|   3|   1000000|     100|0.76688365|  PASSED
       rgb_lagged_sum|   4|   1000000|     100|0.17580288|  PASSED
       rgb_lagged_sum|   5|   1000000|     100|0.98375846|  PASSED
       rgb_lagged_sum|   6|   1000000|     100|0.61895122|  PASSED
       rgb_lagged_sum|   7|   1000000|     100|0.00499714|   WEAK
       rgb_lagged_sum|   8|   1000000|     100|0.35141900|  PASSED
       rgb_lagged_sum|   9|   1000000|     100|0.70201257|  PASSED
       rgb_lagged_sum|  10|   1000000|     100|0.84073477|  PASSED
       rgb_lagged_sum|  11|   1000000|     100|0.72417336|  PASSED
       rgb_lagged_sum|  12|   1000000|     100|0.77524426|  PASSED
       rgb_lagged_sum|  13|   1000000|     100|0.84896209|  PASSED
       rgb_lagged_sum|  14|   1000000|     100|0.84977466|  PASSED
       rgb_lagged_sum|  15|   1000000|     100|0.96568395|  PASSED
       rgb_lagged_sum|  16|   1000000|     100|0.75286635|  PASSED
       rgb_lagged_sum|  17|   1000000|     100|0.46369400|  PASSED
       rgb_lagged_sum|  18|   1000000|     100|0.86166809|  PASSED
       rgb_lagged_sum|  19|   1000000|     100|0.81119460|  PASSED
       rgb_lagged_sum|  20|   1000000|     100|0.96406188|  PASSED
       rgb_lagged_sum|  21|   1000000|     100|0.74030396|  PASSED
       rgb_lagged_sum|  22|   1000000|     100|0.69331757|  PASSED
       rgb_lagged_sum|  23|   1000000|     100|0.92129308|  PASSED
       rgb_lagged_sum|  24|   1000000|     100|0.06453867|  PASSED
       rgb_lagged_sum|  25|   1000000|     100|0.69922107|  PASSED
       rgb_lagged_sum|  26|   1000000|     100|0.48365384|  PASSED
       rgb_lagged_sum|  27|   1000000|     100|0.59755302|  PASSED
       rgb_lagged_sum|  28|   1000000|     100|0.70529601|  PASSED
```

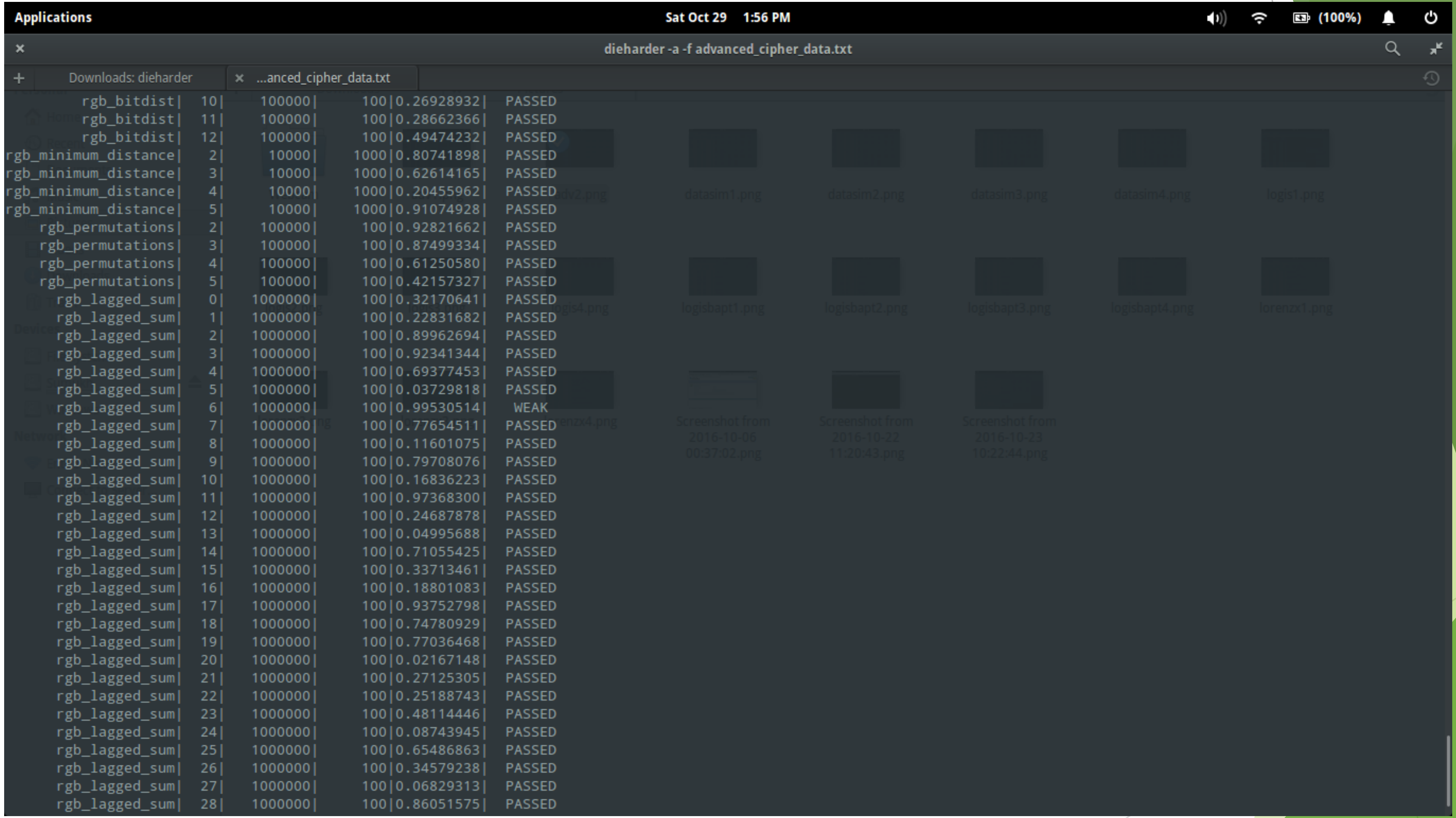
Results for Advanced cipher



```
sumukhvaidya@sumukhvaidya:~/Downloads$ dieharder -a -f advanced_cipher_data.txt
#=============================================================================#
#            dieharder version 3.31.0 Copyright 2003 Robert G. Brown          #
#=============================================================================#
   rng_name    |           filename             |rands/second|
        mt19937|  advanced_cipher_data.txt|  1.53e+08  |
#=============================================================================#
        test_name   |ntup| tsamples |psamples|  p-value |Assessment
#=============================================================================#
   diehard_birthdays|   0|       100|     100|0.63731403|  PASSED
      diehard_operm5|   0|   1000000|     100|0.61939525|  PASSED
  diehard_rank_32x32|   0|     40000|     100|0.33359047|  PASSED
    diehard_rank_6x8|   0|    100000|     100|0.94408881|  PASSED
   diehard_bitstream|   0|   2097152|     100|0.89797557|  PASSED
        diehard_opso|   0|   2097152|     100|0.78328669|  PASSED
        diehard_oqso|   0|   2097152|     100|0.34333381|  PASSED
         diehard_dna|   0|   2097152|     100|0.47218853|  PASSED
diehard_count_1s_str|   0|    256000|     100|0.05229784|  PASSED
diehard_count_1s_byt|   0|    256000|     100|0.17461703|  PASSED
 diehard_parking_lot|   0|     12000|     100|0.76094232|  PASSED
    diehard_2dsphere|   2|      8000|     100|0.99330012|  PASSED
    diehard_3dsphere|   3|      4000|     100|0.29183112|  PASSED
     diehard_squeeze|   0|    100000|     100|0.41587521|  PASSED
        diehard_sums|   0|       100|     100|0.02378870|  PASSED
        diehard_runs|   0|    100000|     100|0.35837607|  PASSED
        diehard_runs|   0|    100000|     100|0.04749496|  PASSED
       diehard_craps|   0|    200000|     100|0.84474830|  PASSED
       diehard_craps|   0|    200000|     100|0.99999186|   WEAK
 marsaglia_tsang_gcd|   0|  10000000|     100|0.89937593|  PASSED
 marsaglia_tsang_gcd|   0|  10000000|     100|0.45422635|  PASSED
         sts_monobit|   1|    100000|     100|0.63002701|  PASSED
            sts_runs|   2|    100000|     100|0.81776334|  PASSED
          sts_serial|   1|    100000|     100|0.04186909|  PASSED
          sts_serial|   2|    100000|     100|0.81049428|  PASSED
          sts_serial|   3|    100000|     100|0.27338559|  PASSED
          sts_serial|   3|    100000|     100|0.57877136|  PASSED
          sts_serial|   4|    100000|     100|0.46314408|  PASSED
          sts_serial|   4|    100000|     100|0.73282296|  PASSED
          sts_serial|   5|    100000|     100|0.25271401|  PASSED
          sts_serial|   5|    100000|     100|0.14305103|  PASSED
```



```
          sts_serial|   4|    100000|     100|0.46314408|  PASSED
          sts_serial|   4|    100000|     100|0.73282296|  PASSED
          sts_serial|   5|    100000|     100|0.25271401|  PASSED
          sts_serial|   5|    100000|     100|0.14305103|  PASSED
          sts_serial|   6|    100000|     100|0.58479766|  PASSED
          sts_serial|   6|    100000|     100|0.60494705|  PASSED
          sts_serial|   7|    100000|     100|0.65883220|  PASSED
          sts_serial|   7|    100000|     100|0.22926127|  PASSED
          sts_serial|   8|    100000|     100|0.69473016|  PASSED
          sts_serial|   8|    100000|     100|0.99097043|  PASSED
          sts_serial|   9|    100000|     100|0.16500617|  PASSED
          sts_serial|   9|    100000|     100|0.36917389|  PASSED
          sts_serial|  10|    100000|     100|0.16790078|  PASSED
          sts_serial|  10|    100000|     100|0.09257547|  PASSED
          sts_serial|  11|    100000|     100|0.99862466|   WEAK
          sts_serial|  11|    100000|     100|0.60144604|  PASSED
          sts_serial|  12|    100000|     100|0.99090483|  PASSED
          sts_serial|  12|    100000|     100|0.36852643|  PASSED
          sts_serial|  13|    100000|     100|0.88951026|  PASSED
          sts_serial|  13|    100000|     100|0.44740732|  PASSED
          sts_serial|  14|    100000|     100|0.99960152|   WEAK
          sts_serial|  14|    100000|     100|0.97843475|  PASSED
          sts_serial|  15|    100000|     100|0.99146426|  PASSED
          sts_serial|  15|    100000|     100|0.63835738|  PASSED
          sts_serial|  16|    100000|     100|0.55830833|  PASSED
          sts_serial|  16|    100000|     100|0.18362451|  PASSED
         rgb_bitdist|   1|    100000|     100|0.86422973|  PASSED
         rgb_bitdist|   2|    100000|     100|0.68198896|  PASSED
         rgb_bitdist|   3|    100000|     100|0.46340482|  PASSED
         rgb_bitdist|   4|    100000|     100|0.66597457|  PASSED
         rgb_bitdist|   5|    100000|     100|0.40930989|  PASSED
         rgb_bitdist|   6|    100000|     100|0.15505569|  PASSED
         rgb_bitdist|   7|    100000|     100|0.68116513|  PASSED
         rgb_bitdist|   8|    100000|     100|0.27022489|  PASSED
         rgb_bitdist|   9|    100000|     100|0.66521960|  PASSED
         rgb_bitdist|  10|    100000|     100|0.26928932|  PASSED
         rgb_bitdist|  11|    100000|     100|0.28662366|  PASSED
         rgb_bitdist|  12|    100000|     100|0.49474232|  PASSED
rgb_minimum_distance|   2|     10000|    1000|0.80741898|  PASSED
rgb_minimum_distance|   3|     10000|    1000|0.62614165|  PASSED
```

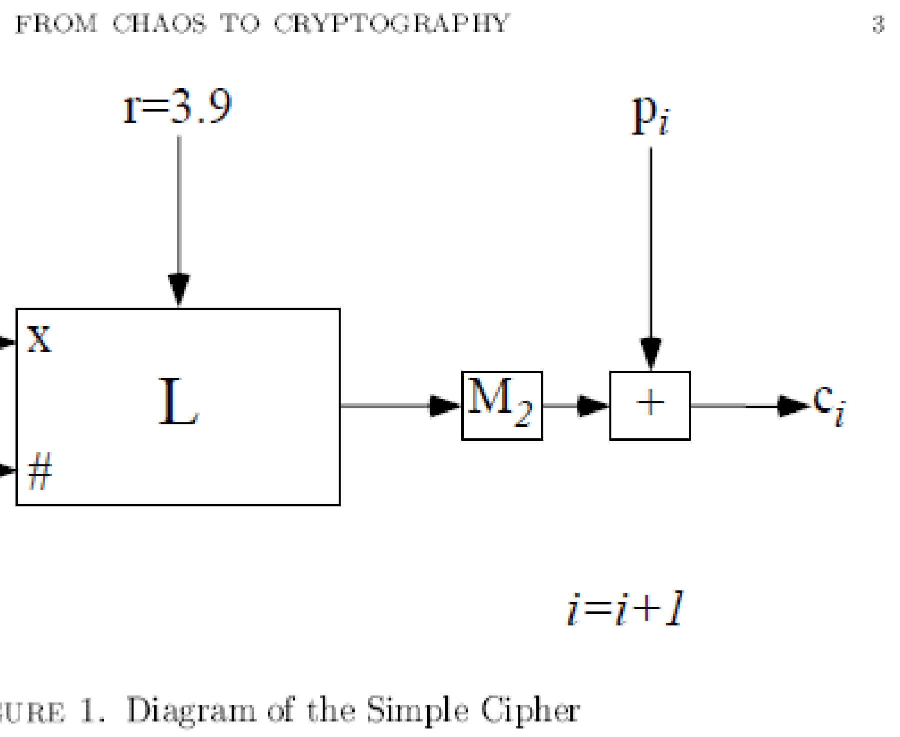


```
         rgb_bitdist|  10|    100000|     100|0.26928932|  PASSED
         rgb_bitdist|  11|    100000|     100|0.28662366|  PASSED
         rgb_bitdist|  12|    100000|     100|0.49474232|  PASSED
rgb_minimum_distance|   2|     10000|    1000|0.80741898|  PASSED
rgb_minimum_distance|   3|     10000|    1000|0.62614165|  PASSED
rgb_minimum_distance|   4|     10000|    1000|0.20455962|  PASSED
rgb_minimum_distance|   5|     10000|    1000|0.91074928|  PASSED
    rgb_permutations|   2|    100000|     100|0.92821662|  PASSED
    rgb_permutations|   3|    100000|     100|0.87499334|  PASSED
    rgb_permutations|   4|    100000|     100|0.61250580|  PASSED
    rgb_permutations|   5|    100000|     100|0.42157327|  PASSED
      rgb_lagged_sum|   0|   1000000|     100|0.32170641|  PASSED
      rgb_lagged_sum|   1|   1000000|     100|0.22831682|  PASSED
      rgb_lagged_sum|   2|   1000000|     100|0.89962694|  PASSED
      rgb_lagged_sum|   3|   1000000|     100|0.92341344|  PASSED
      rgb_lagged_sum|   4|   1000000|     100|0.69377453|  PASSED
      rgb_lagged_sum|   5|   1000000|     100|0.03729818|  PASSED
      rgb_lagged_sum|   6|   1000000|     100|0.99530514|   WEAK
      rgb_lagged_sum|   7|   1000000|     100|0.77654511|  PASSED
      rgb_lagged_sum|   8|   1000000|     100|0.11601075|  PASSED
      rgb_lagged_sum|   9|   1000000|     100|0.79708076|  PASSED
      rgb_lagged_sum|  10|   1000000|     100|0.16836223|  PASSED
      rgb_lagged_sum|  11|   1000000|     100|0.97368300|  PASSED
      rgb_lagged_sum|  12|   1000000|     100|0.24687878|  PASSED
      rgb_lagged_sum|  13|   1000000|     100|0.04995688|  PASSED
      rgb_lagged_sum|  14|   1000000|     100|0.71055425|  PASSED
      rgb_lagged_sum|  15|   1000000|     100|0.33713461|  PASSED
      rgb_lagged_sum|  16|   1000000|     100|0.18801083|  PASSED
      rgb_lagged_sum|  17|   1000000|     100|0.93752798|  PASSED
      rgb_lagged_sum|  18|   1000000|     100|0.74780929|  PASSED
      rgb_lagged_sum|  19|   1000000|     100|0.77036468|  PASSED
      rgb_lagged_sum|  20|   1000000|     100|0.02167148|  PASSED
      rgb_lagged_sum|  21|   1000000|     100|0.27125305|  PASSED
      rgb_lagged_sum|  22|   1000000|     100|0.25188743|  PASSED
      rgb_lagged_sum|  23|   1000000|     100|0.48114446|  PASSED
      rgb_lagged_sum|  24|   1000000|     100|0.08743945|  PASSED
      rgb_lagged_sum|  25|   1000000|     100|0.65486863|  PASSED
      rgb_lagged_sum|  26|   1000000|     100|0.34579238|  PASSED
      rgb_lagged_sum|  27|   1000000|     100|0.06829313|  PASSED
      rgb_lagged_sum|  28|   1000000|     100|0.86051575|  PASSED
```

Simple Cipher

As Implemented By Ruskin and Casper.

Analysis of Simple Cipher



FROM CHAOS TO CRYPTOGRAPHY 3

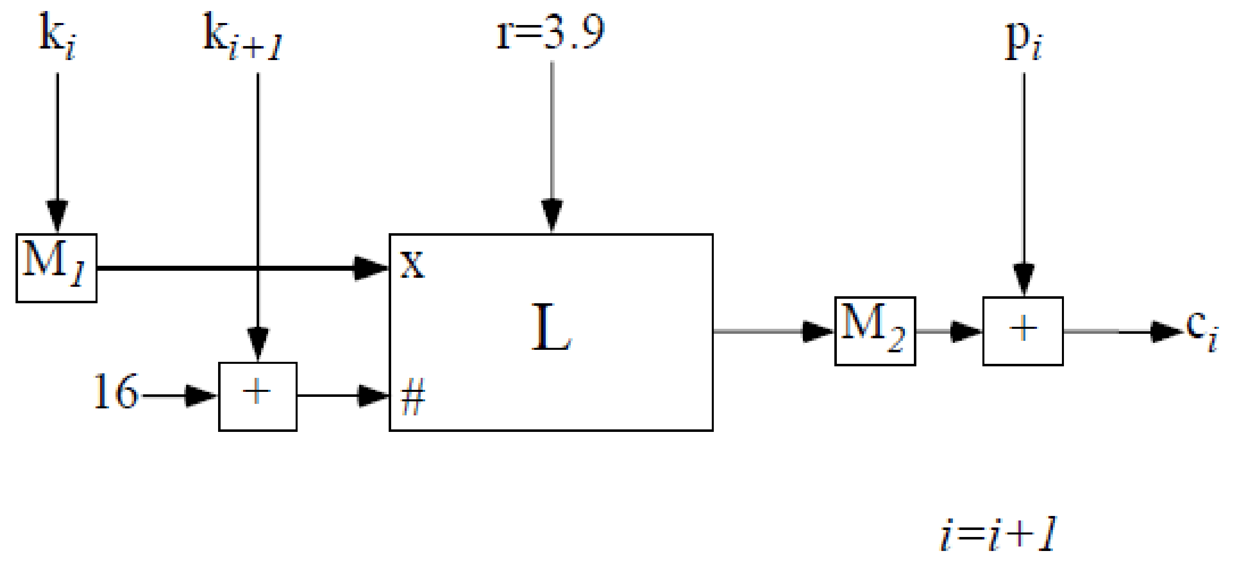


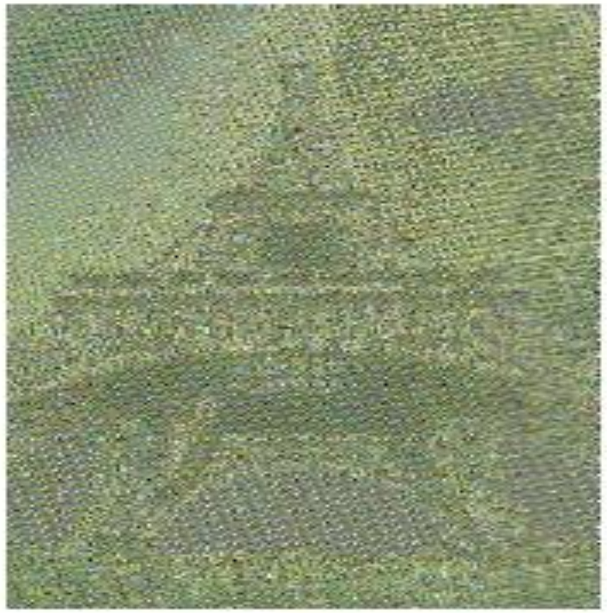
FIGURE 1. Diagram of the Simple Cipher

Analysis of security:

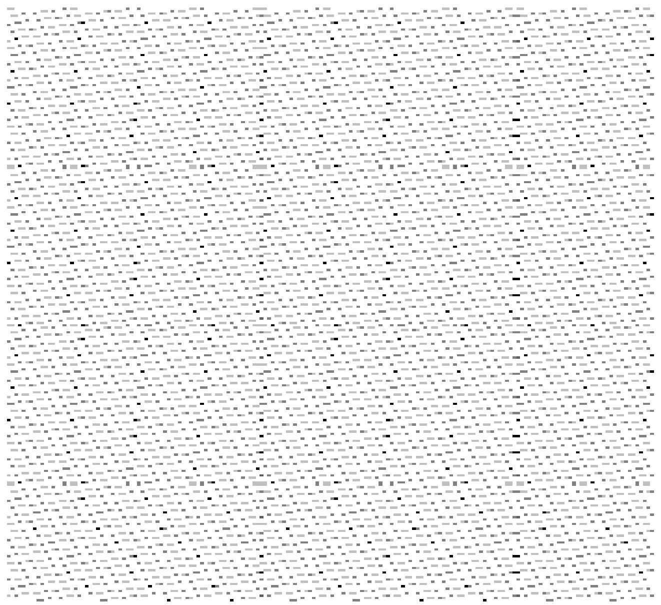
- Cipher has some nice statistical properties.
- Simple cipher was implemented using algorithm by Fridrich Fri(97).

Fridrich’s Implementation

Plain Image for encryption


Encrypted File.

Using Chaos in Simple Cipher.



Encrypted Image using Simple Cipher with Chaos

- Periodic nature observed introduces insecurity in the cipher.

- The pads that are produced by the cipher formed a series that created a pattern of displacement in the cipher text.

Advanced Cipher

Schematic Diagram

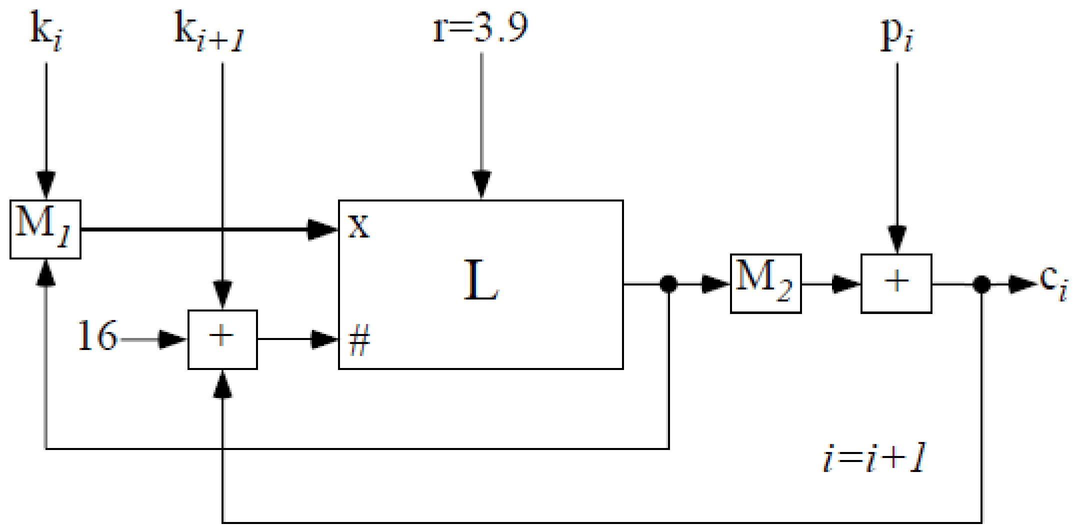
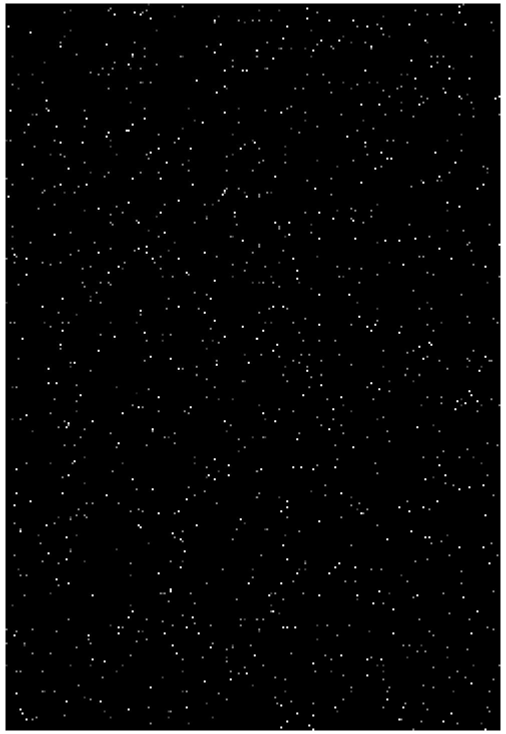


FIGURE 5. Diagram of the Advanced Cipher

Feedback Mechanism.

- Design so as to vary the pad by both the key values and the output of the previous cipher text byte.

Implementation of Advanced Cipher

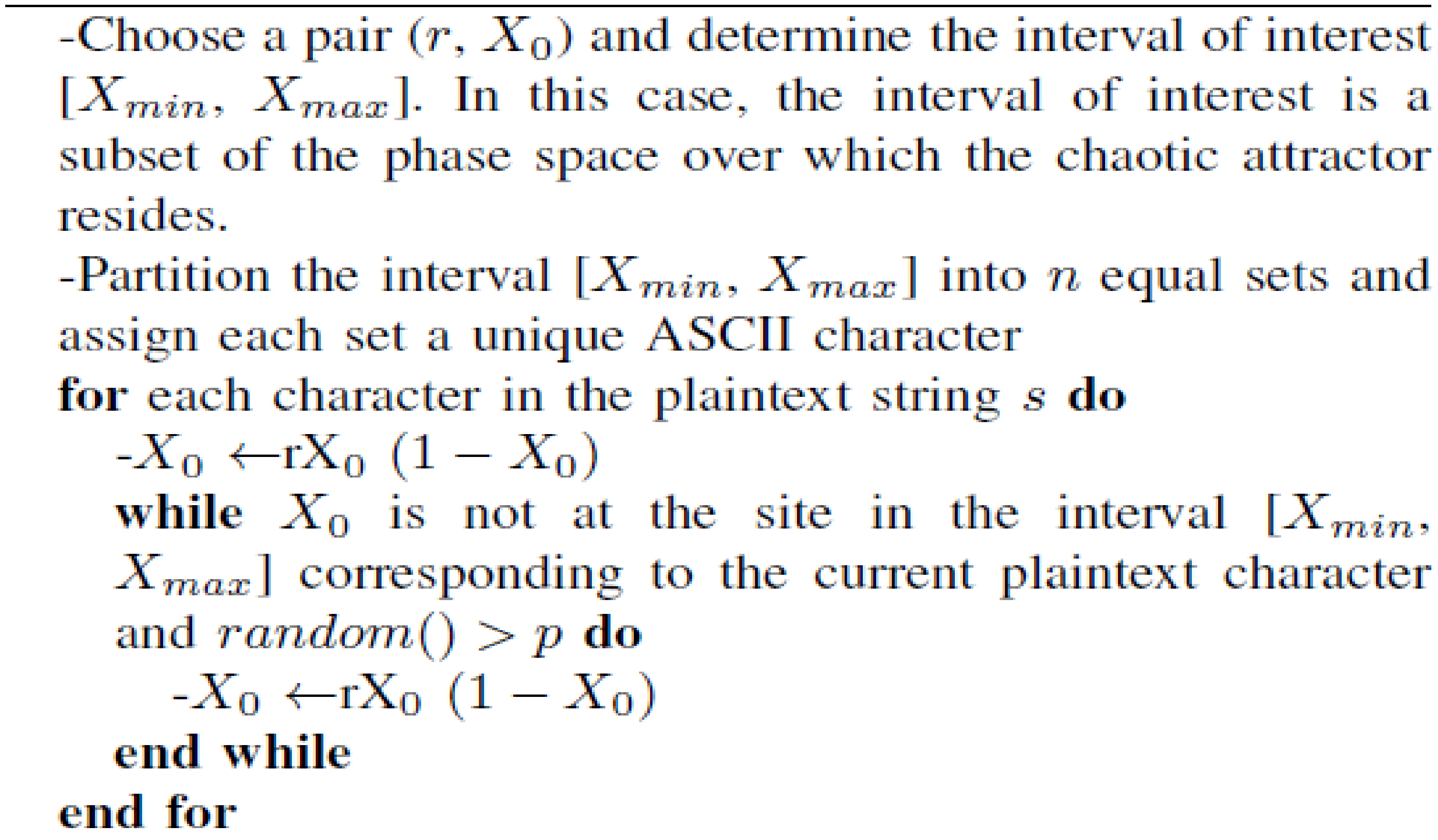


Encrypted Image using
Advanced Cipher

Analysis

- The cipher has very good statistical properties.
- Additionally, it was shown that a change in a single bit in the encryption key changed, on average, 49.6% of the bits in the corresponding cipher text.
- There are residual dark spots that appear around high contrast areas of plain image.
- Advanced Cipher does far better job of encryption owing to its lack of periodicity in the cipher text.

Baptista Algorithm using Logistic Map



```
-Choose a pair ($r$, $X_0$) and determine the interval of interest [$X_{min}$, $X_{max}$]. In this case, the interval of interest is a subset of the phase space over which the chaotic attractor resides.
-Partition the interval [$X_{min}$, $X_{max}$] into $n$ equal sets and assign each set a unique ASCII character
for each character in the plaintext string $s$ do
    -$X_0 \leftarrow rX_0\ (1 - X_0)$
    while $X_0$ is not at the site in the interval [$X_{min}$, $X_{max}$] corresponding to the current plaintext character and $random() > p$ do
        -$X_0 \leftarrow rX_0\ (1 - X_0)$
    end while
end for
```

Analysis

- The implementation is simple enough.
- The performance of the algorithm depends on the randomness of the PRNG.
- Performance depends on the initial values and properties of logistic map.
- The size of ‘p’ should be taken care of.
- Merit: For r = 3.8, besides having the knowledge of S keys the initial conditions play crucial role in decrypting the cipher text.

Conclusion

- Its not the most sought after cryptographic primitive. Security properties are not provable and performance is very slow.
- The relation between the two fields is subtle but very much present.
- The space of reals also poses a problem.
- The field is still emerging and the connection between the two seemingly different fields of chaos and cryptography has just been found. As mentioned earlier due to lack of provable security properties the enthusiasm is low.
- Currently number theory leads the way in cryptography. Chaos theory is a strong contender for leading chaos into the future.

Future Work

- Though the security properties an the performance characteristics are not provable and weak, it is an evolving field and future might hold something truly wonderful which might be manifested in the unique intermixing of these two seemingly different fields but with many topological similarities.
- Future work in this field continues to this day, trying to develop more efficient algorithms. As Computers become more sophisticated and with further advances in information technology, chaos might be the way to future cryptosystems.
- From information theoretic perspective evolution of quantum cryptography and chaos hold the key to future research.

Thank You